THE UNIVERSITY of EDINBURGH

Advanced Databases

Spring 2020

Lecture #01:

Course Intro & Relational Databases

Milos Nikolic

Course Overview

This course is on the **design** and **implementation** of disk-oriented database management systems

This is **not** a course on how to use a database to build applications or how to administer a database

Recommended prerequisite: Database Systems (INFR10070)

Course Outline

Relational Databases

Storage and File Structure

Indexing

Query Evaluation

Query Optimisation

Transaction Management

Distributed Databases

Learning Outcomes

Gain insights into how DBMSs function internally

Implement major components of a database system

Learn data management techniques that can help **YOU, the future scientist,** to transform data into knowledge

Useful concepts for CS and other sciences in general

Course Logistics

Course policies + schedule

- Refer to course web page: https://course.inf.ed.ac.uk/adbs

Two in-class tutorials, no practical labs

All discussion + announcements will be on Piazza

- https://piazza.com/ed.ac.uk/spring2020/infr11011

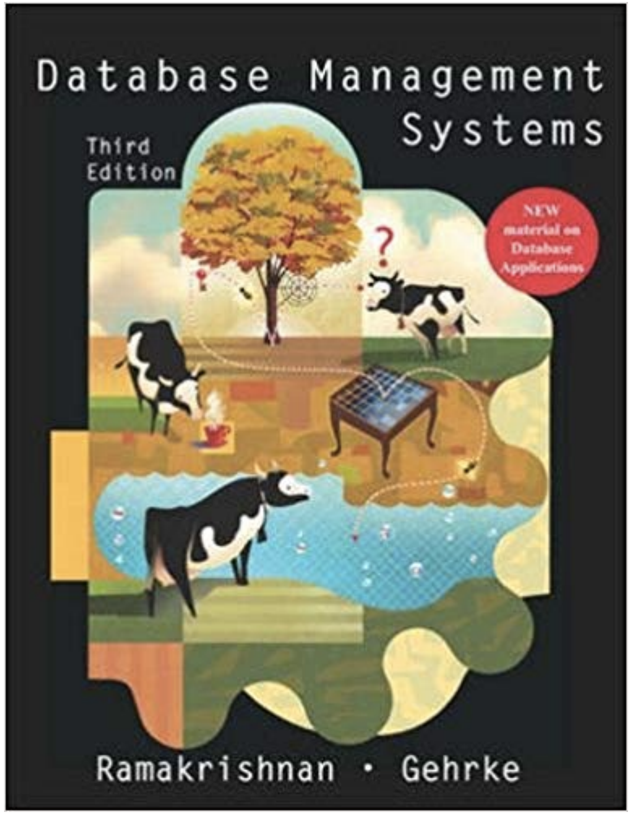
Sign up now with your student email address

Textbook

Database Management Systems
3rd Edition
R. Ramakrishnan and J. Gehrke

Most lectures will closely follow this book

Lectures will also refer to research papers



Assessment

Coursework (30%)

- No requirement to pass the coursework

Written Exam (70%)

School of Informatics uses a Common Marking Scheme

- 1st class or MSc distinction: 70% and above

Coursework

Implement features in an educational database system in C++

1 formative assignment (0%)

- Purpose: get feedback & accommodate to DBMS and C++

1 summative assignment (30%)

- Project: implement features + run experiments + write report

School of Informatics has a policy on coursework deadlines

Plagiarism Policy

All assignments must be your own work

They are **not** group assignments

You may **not** copy source code from other people or the web

You may **not** use public repositories to host your code

See UoE Academic misconduct for more information

Acknowledgements

The lecture notes draw on notes by several people to which I am grateful, in particular:

D. Olteanu and T. Furche from Oxford

P. Guagliardo and S. Viglas from Edinburgh

A. Pavlo from CMU

T. Grust from Tübingen

J. Gehrke from Microsoft

Databases

DATABASE

Organised collection of inter-related data that models some aspect of the real-world

Databases are the core component of most computer applications

- Banking
- Web and mobile apps
- Sales
- Online retailers
- Human resources

Database Example

Create a database that models a university organisation to keep track of students, instructors, and courses

Application program examples:

- Add new students, instructors, and courses
- Register students for courses and generate class rosters
- Assign grades to students, compute GPA, and generate transcripts

FLAT FILE STRAWMAN

Store our database as comma-separated value (CSV) files

Instructor(name, dept, salary)

```
“Jones”, “CS”, 95000
“Smith”, “Physics”, 75000
“Gold”, “CS”, 62000
```

instructor.csv

Course(name, instructor, year)

```
“Databases”, “Jones”, 2018
“Quantum M.”, “Smith”, 2017
“Compilers”, “Jones”, 2017
```

course.csv

Apps have to parse the files each time they want to read/update records

Flat File Strawman

Example: Get the names of all computer science instructors

Instructor(name, dept, salary)

```
"Jones", "CS", 95000
"Smith", "Physics", 75000
"Gold", "CS", 62000
```

instructor.csv

```
for line in file:
  record = parse(line)
  if "CS" == record[1]:
    print record[0]
```

Tight coupling between application logic and physical storage

Flat File: Drawbacks

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

Due to multiple files and formats writing new programs is difficult

Integrity problems

Integrity constraints (e.g., balance > 0) become “buried” in program code

Hard to add new constraints or change existing ones

Flat File: Drawbacks (Cont.)

Atomicity of updates

- Failures may leave database in an inconsistent state with partial updates carried out
- Ex: Moving money between two accounts 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

Security problems

- Hard to provide user access to some, but not all, data

Database systems offer solutions to all the above problems

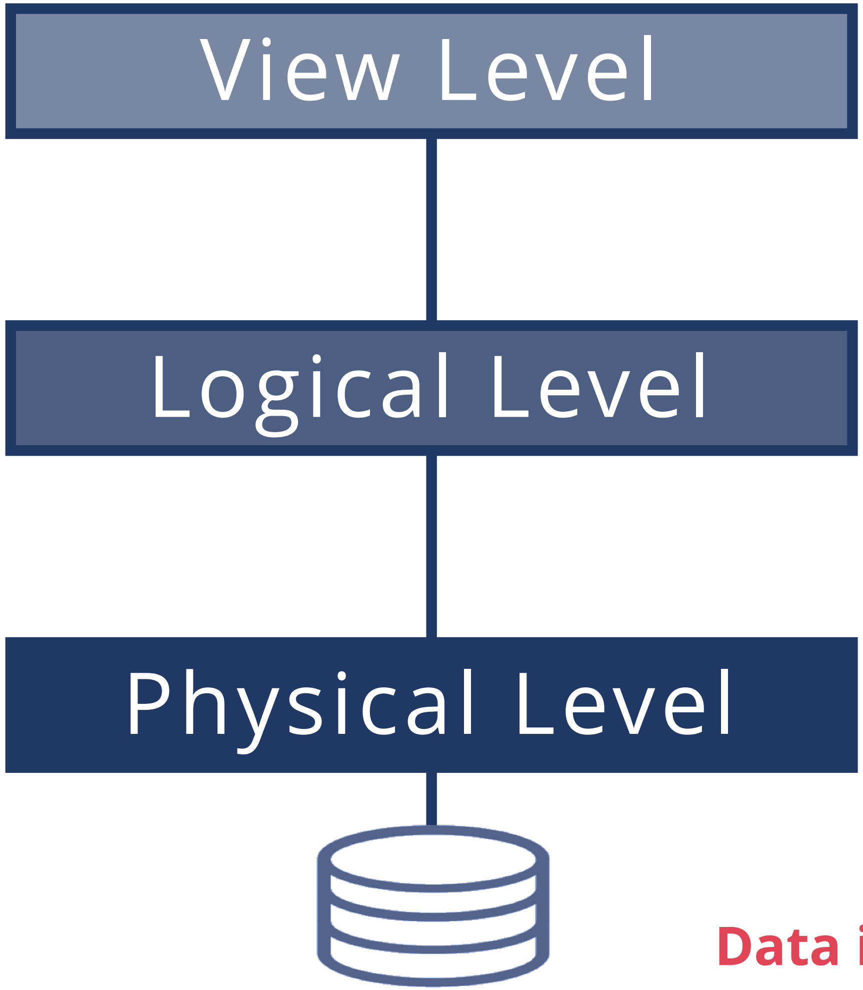
Database Management System (DBMS)

Software that stores, manages, and facilitates access to databases

Mediates interactions between users and databases

Provide users with an **abstract view** of the data

Levels of Abstractions



Simplifies interaction with the database, hides info (e.g., salary) for security purposes

Describes data stored in the DB

```
type instructor = record
  name: string;
  dept: string;
  salary: integer;
end
```

Describes how a record is stored

Data independence: Insulate users from changes in lower levels

DATA MODELS

Data model

collection of concepts for describing the data in a database

Schema

description of a particular collection of data, using a given model

Models in practice

relational, key-value, graph, document, array, hierarchical, network

Most DBMSs implement the relational data model

Relational Model

Proposed in 1970 by Edgar T. Codd

Structure

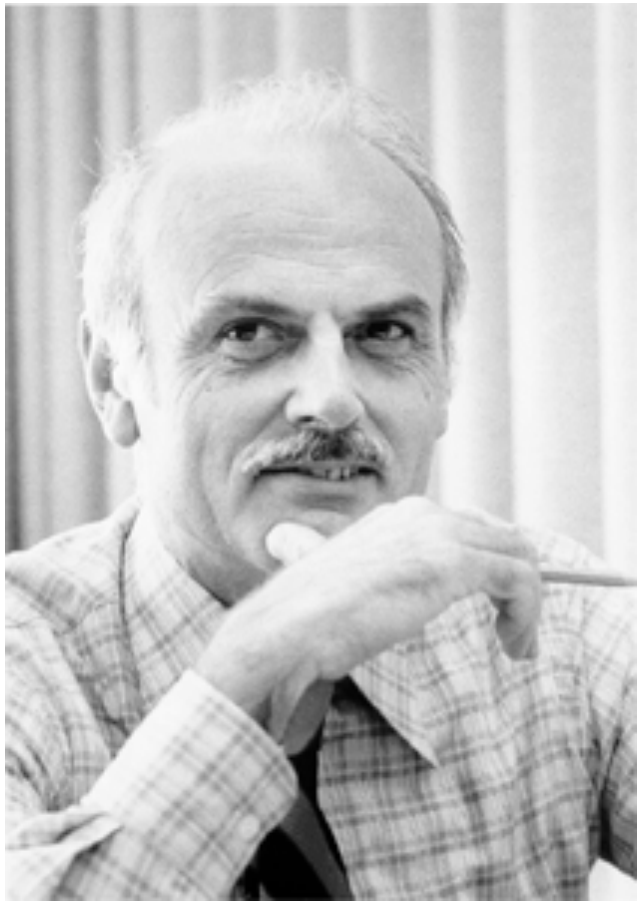
The definition of relations and their contents

Integrity

Ensure the database's contents satisfy constraints

Manipulation

How to access and modify a database's contents



RELATIONAL MODEL

Data organised in relations (tables)

Relation schema

relation name + distinct (typed) attributes + constraints

Relation is a set of records

Order of records is irrelevant

Record is a set of attribute values

Values are (normally) atomic / scalar

`Instructor(name, dept, salary)`

name	dept	salary
Jones	CS	95000
Smith	Physics	75000
Gold	CS	62000

Relational Model: Primary Keys

A **primary key** uniquely identifies a single tuple

DBMSs may automatically create an internal primary key if you don't define one

`Instructor(id, name, dept, salary)`

id	name	dept	salary
123	Jones	CS	95000
456	Smith	Physics	75000
789	Gold	CS	62000

Auto-generation of unique integer primary keys

- SEQUENCE (SQL:2003)
- AUTO_INCREMENT (MySQL)

Relational Model: Foreign Keys

A **foreign key** specifies that an attribute from one relation has to map to a tuple in another relation

Instructor(id, name, dept, salary)

id	name	dept	salary
123	Jones	CS	95000
456	Smith	Physics	75000
789	Gold	CS	62000

Course(id, name, instr, year)

id	name	instr	year
11	Databases	123	2018
22	Quantum M.	456	2017
33	Compilers	123	2017

Relational Model: Foreign Keys

Instructor(id, name, dept, salary)

id	name	dept	salary
123	Jones	CS	95000
456	Smith	Physics	75000
789	Gold	CS	62000

InstructorCourse(instr_id, course_id)

instr_id	course_id
123	11
789	11
456	22
123	33

Course(id, name, year)

id	name	year
11	Databases	2018
22	Quantum M.	2017
33	Compilers	2017

Relational Query Languages

How to retrieve information from a database

Procedural

The query specifies the (high-level) strategy the DBMS should use to find the desired result

⬅ **Relational** Algebra

Declarative

The query specifies only what data is wanted and not how to find it

⬅ **Relational** Calculus

Relational Algebra

A relational algebra expression

- takes as input one or more relations
- applies a sequence of operations
- returns a relation as output

Example: $\pi_{\text{name}}(\sigma_{\text{dept} = \text{'CS'}}(\text{Instructor}))$

RA is based on set semantics

- but can also be defined over bags (multisets)

σ	Selection
π	Projection
$\cup$	Union
$\cap$	Intersection
$-$	Difference
$\times$	Product
$\bowtie$	Natural Join
ρ	Renaming

RELATIONAL ALGEBRA: SELECTION

Choose a subset of the tuples from a relation that satisfy a selection predicate

Can combine multiple predicates using conjunctions / disjunctions

Syntax: $\sigma_{\text{predicate}}(R)$

R(aid, bid)

aid	bid
a1	101
a2	102
a2	103
a3	104

$\sigma_{\text{aid='a2'}}(R)$

aid	bid
a2	102
a2	103

$\sigma_{\text{aid='a2'} \wedge \text{bid} > 102}(R)$

aid	bid
a2	103

Relational Algebra: Projection

Generate a relation with tuples that contains only the specified attributes

- Can rearrange attributes' ordering
- Can manipulate the values

Syntax: $\pi_{A1,\ A2,\ \ldots,\ An}(R)$

R(aid, bid)

aid	bid
a1	101
a2	102
a2	103
a3	104

$\pi_{\text{bid-100, aid}}(\sigma_{\text{aid='a2'}}(R))$

bid-100	aid
2	a2
3	a2

Relational Algebra: Union

Generate a relation that contains all tuples that appear in either only one or both input relations.

- Relations must have the same set of attributes
- Same holds for intersection and difference

Syntax: R ∪ S

R(aid, bid)

aid	bid
a1	101
a2	102
a3	103

S(aid, bid)

aid	bid
a3	103
a4	104
a5	105

R ∪ S

aid	bid
a1	101
a2	102
a3	103
a4	104
a5	105

Relational Algebra: Product

Generate a relation that contains all possible combinations of tuples from the input relations

Syntax: R × S

R(aid, bid)

aid	bid
a1	101
a2	102
a3	103

S(bid, cid)

bid	cid
b3	23
b4	24

R × S

aid	R.bid	S.bid	cid
a1	101	b3	23
a1	101	b4	24
a2	102	b3	23
a2	102	b4	24
a3	103	b3	23
a3	103	b4	24

Relational Algebra: Natural Join

Generate a relation that contains all tuples that are a combination of two tuples (one from each input relation) with a common value(s) for one or more attributes

Syntax: R ⋈ S

R(aid, bid)

aid	bid
a1	101
a2	102
a3	103

S(bid, cid)

bid	cid
101	c3
101	c4
105	c5

R ⋈ S

aid	bid	cid
a1	101	c3
a1	101	c3

Observation

Relational algebra still defines the high-level steps of how to compute a query

$\sigma_{bid = 102}(R \bowtie S)$ vs. $(R \bowtie (\sigma_{bid = 102}(S))$

A better approach is to state the high-level answer that you want the DBMS to compute

Retrieve the joined tuples from **R** and **S** where **bid** equals 102

Relational Model: Queries

SQL is the *de facto* standard

```
for line in file:
  record = parse(line)
  if "CS" == record[1]:
    print record[0]
```

```
SELECT name FROM instructor
 WHERE dept = 'CS';
```

DBMS needs to figure out HOW to compute a query

The passage from WHAT to HOW goes through relational algebra

Next Lecture

Database Architecture