# CMPUT391

# Intro to CMPUT 391

Instructor: Denilson Barbosa

University of Alberta

October 10, 2023

Slides by D. Barbosa, with suggested corrections and improvements by (in alphabetical order) C. Bins, D. Caminhas, K. Guhzva, Q. Lautischer, E. Macdonald, M. A. Nascimento, K. Newbury, M. Strobl, D. Sunderman, K. Wang, and K. Wong.

# Is CMPUT391 what you **imagine**?

## App development?

CMPUT391 is NOT about application development — Web and/or mobile database applications are covered in other courses.

## Data Science?

CMPUT391 is NOT a crash-course on Data Mining, Analytics, Big Data or Data Science.

# CMPUT 291 vs CMPUT 391?

**CMPUT 291 focused on using DBMSs**

1 - data modelling
2 - translating questions into queries
3 - application development with a DBMS vs using files

**CMPUT 391 focuses on how DBMSs work**

1 - start with a single-user DBMS using a single CPU computer
2 - look at how data is stored, queries are processed, and updates are scheduled
3 - then look at a multi-user distributed DBMS

# What was CMPUT291 about?

## Organizing data into a relational database

Application requirements $\rightarrow$ ER diagram + constraints

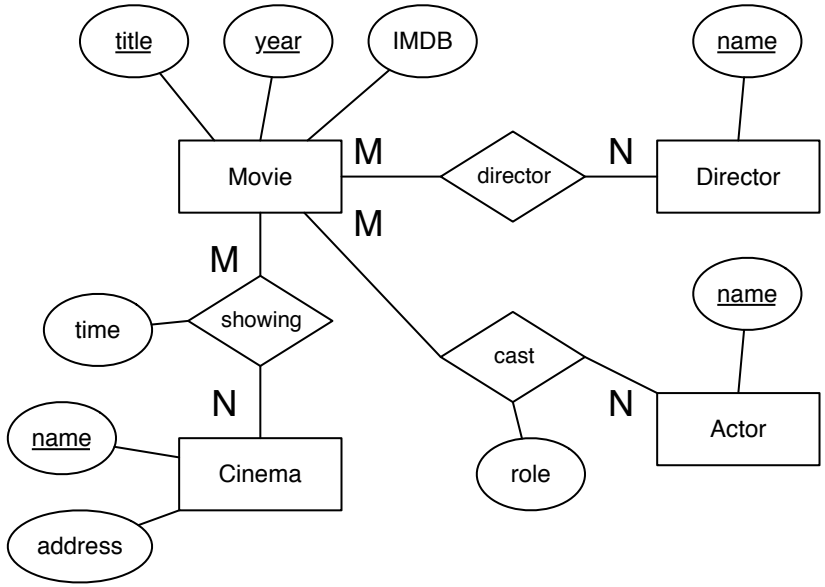
ER diagram + constraints $\rightarrow$ Initial Schema $\rightarrow$ Normalized schema

## Querying the database to satisfy an information need

Application/user information **need** $\rightarrow$ SQL query

# Example

**Requirements**: create an application to keep track of movies, their directors, their actors (with the role they play), and the cinemas where the movies are showing (with the start time).




Must handle remakes (i.e., newer movie with the same title of an old one), and record IMDB scores (0 to 10) for the movies[1].

[1] A better design would be to also record the date when the IMDB score was collected, as they changes over time.

```
CREATE TABLE Movie (title CHAR(20), year INT, imdb FLOAT,
   PRIMARY KEY (title, year), CHECK(imdb >= 0 AND imdb <= 10));
CREATE TABLE Director (name CHAR(20), PRIMARY KEY (name));
CREATE TABLE Actor (name CHAR(20), PRIMARY KEY (name));
CREATE TABLE Cinema (name CHAR(20), address CHAR(100),
   PRIMARY KEY(name));

CREATE TABLE MovieDirector (title CHAR(20), year INT, name CHAR(20)
  PRIMARY KEY (title, year, name),
  FOREIGN KEY (title, year) REFERENCES Movie(title, year),
  FOREIGN KEY (name) REFERENCES Director(name));

CREATE TABLE Cast (title CHAR(20), year INT, name CHAR(20), role CHAR(20),
  PRIMARY KEY (title, year, name, role),
  FOREIGN KEY (title, year) REFERENCES Movie(title, year)
  FOREIGN KEY (name) REFERENCES Actor(name));

CREATE TABLE Guide (theater CHAR(20), film CHAR(20), year INT, start INT,
  PRIMARY KEY (theater, film, year, start),
  FOREIGN KEY (theater) REFERENCES Cinema(name),
  FOREIGN KEY (film, year) REFERENCES Movie(title, year));
```

In CMPUT291 you learned about schema refinement by **normalization**, through decomposing relations that did not conform with a normal form.

- Normalization removes the chance of many update anomalies (which can cause many data inconsistencies).
- However, decomposing a “conceptual” relation into several smaller ones means we often need to **join** them back together to answer user queries.

In some applications, the extra cost of the joins might be unacceptable, in which case the schema is de-normalized for performance reasons.

In CMPUT391 we will use a de-normalized, but simpler schema, where we represent the relationship Movie–Director as an attribute of Movie, we eliminate the need for joins:

```
CREATE TABLE Movie (title CHAR(20), year INT, imdb FLOAT,
   director CHAR(20), PRIMARY KEY (title, year),
   CHECK(imdb >= 0 AND imdb <= 10));
```

PROS: querying for movie directors is faster… CONS: can't record movies with multiple directors; and directors disappear from the database if we delete their movies.

# Simplified schema used in the examples

```
CREATE TABLE Movie (title CHAR(20), year INT, imdb FLOAT,
   director CHAR(20) NOT NULL, PRIMARY KEY (title, year),
   CHECK(imdb >= 0 AND imdb <= 10));

CREATE TABLE Cinema (name CHAR(20), address CHAR(100),
   PRIMARY KEY(name));

CREATE TABLE Cast (title CHAR(20), year INT,
   actor CHAR(20) NOT NULL, role CHAR(20) NOT NULL,
   FOREIGN KEY (title, year) REFERENCES Movie(title, year));

CREATE TABLE Guide (theater CHAR(20), film CHAR(20), year INT, start INT,
  PRIMARY KEY (theater, film, year, start),
  FOREIGN KEY (theater) REFERENCES Cinema(name),
  FOREIGN KEY (film, year) REFERENCES Movie(title, year));
```

# Querying

In CMPUT 291 you focused on translating **user/application needs** like “who are the actors in movies directed by Debra Granik rated 7 or higher on IMDB?”

Into **SQL**:

```
SELECT c.actor
FROM Cast c, Movie m
WHERE c.title=m.title AND c.year = m.year AND
      m.director = 'Debra Granik' AND m.IMDB >= 7;
```

In 391 we care only about **how** to execute queries fast.

# What is CMPUT391 about?

## How is data stored?

- Hardware and I/O
- heap and sequential files
- index and hash files

## Query processing?

- what does an algorithm to perform a join look like?
- what is its cost?

## SQL

- embedded SQL in C
- recursion
- triggers

## Transactions?

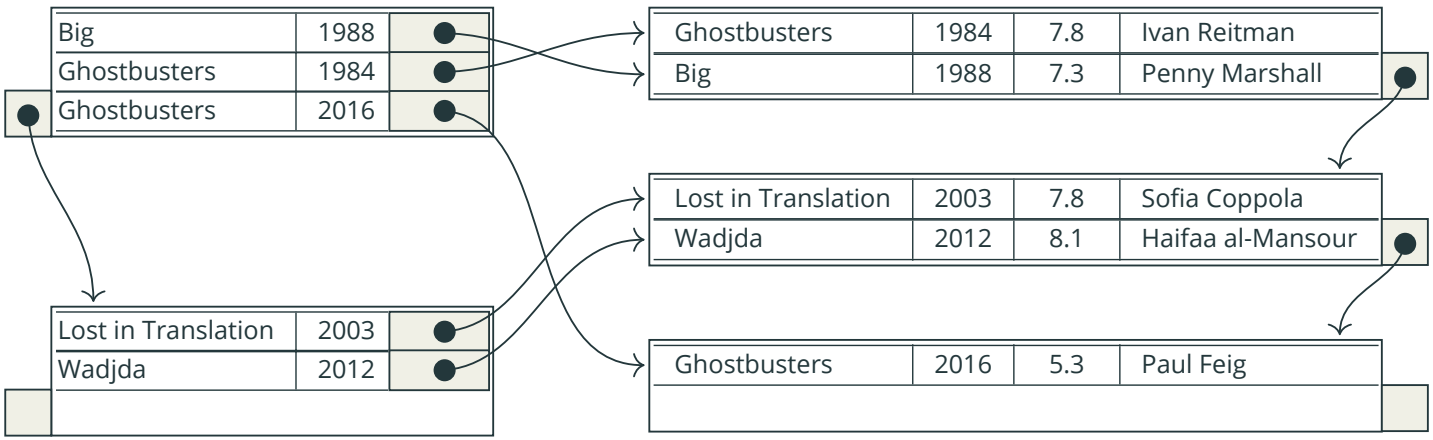
- database logging
- ACID transactions
- Concurrency control

All of that in single computer or distributed/parallel computer settings.

# How do we look at a database in 391?

The schema (normalized or not) goes way.

We look at the files that store data (i.e., tables) and files that help us get to the data fast (i.e., indexes).



# What about queries and SQL updates?

We want to understand what is the most efficient way to execute them, keeping in mind:

(a) data is kept in secondary memory and must be brought into memory before anything can be done to them;

(b) the most efficient way of processing a statement depends both on *static factors* (e.g., data organization) and *dynamic factors* (e.g., the system load);

(c) the DBMS **must support** multiple users/applications reading/writing the same database **concurrently**

(d) the DBMS **must protect** the data from system failures, and ensure the database remains **consistent** after each update

# What else is covered in CMPUT 391?

1 - Advanced relational constraints (aka Business Rules or Triggers)

2 - Advanced SQL (recursion, crisp semantics and evaluation algorithms)

3 - Non-relational data models and queries

  i - RDF and SPARQL (**on your own –** homework)

  ii - XML and XQuery

  iii - JSON and JSONiq (**on your own –** homework)

  iv - Spatial data and Nearest Neighbor queries (**on your own – coding asisgnments**)

# Prerequisites

Prerequisite material which you are **expected to know** and will be tested on:

**CMPUT 291**

- the relational model, SQL (DDL and DML) and the algebra
- writing queries and translating between the algebra and SQL

**CMPUT 201**

- C programming; compiling and debugging C programs
- working with the command line and large files; git

**CMPUT 204**

- algorithms and data structures (e.g., linked lists, sorting, hashing)
- estimating time and space costs (e.g., $O(1)$, $O(n)$, $O(\log n)$).

# Common student complaints – and the truth behind them

1 - not enough time to do the assignments – you have at least 3 weeks for each assignment

2 - **too much reading** – no comment

3 - **we are supposed to learn things on our own** – no comment

4 - no part marks for "getting the idea right" – no reasonable answer is turned down in 391

5 - assignments are not worth enough – they are there for you to practice and learn; plus **you will be tested on then in the exams**

6 - exams cover material not taught in class – **yes they do**! homework and assignments are covered in the exams.

# Plagiarism – what you should know

**Every single case of plagiarism in CMPUT391 will be reported to the Faculty of Science.**

In previous academic terms sanctions were applied by the Faculty in every case reported.

Most cases resulted in at least one letter grade reduction.

Repeating offenders received reductions of 3 letter grades.

Students have failed CMPUT391 because of cheating.

# How to succeed in CMPUT391?

(1) Start early on the homework. They do take a lot of time.

(2) **Consult a** database **systems** book regularly. Some slides condense many pages of material that cannot be explained in detail in class.

(3) Read the supplemental material referenced from the slides.

(4) Answer the practice questions on your own. Most exam questions will be very similar to them.

(5) Consult your TA regularly; ask them to help you with the practice questions.