CO130 - Databases

Prelude

The content discussed here is part of CO130 - Databases (Computing MEng); taught by Thomas Heinis, in Imperial College London during the academic year 2018/19. The notes are written for my personal use, and have no guarantee of being correct (although I hope it is, for my own sake).

Material Order

These notes are primarily based off the slides on CATe. This is the order in which they are uploaded (and I'd assume the order in which they are taught).

1. *Introduction.pdf*
2. *ER Modelling.pdf*
3. *ER to RM.pdf*
4. *Relational Algebra.pdf*
5. *Tutorial ER.pdf*
6. *Tutorial Translation.pdf*
7. *Relational Algebra.pdf*
8. *Functional Dependencies.pdf*
9. *Tutorial Relational Algebra.pdf*
10. *Normalisation.pdf*
11. *SQL.pdf*
12. *Data Definition.pdf*
13. *Data Manipulation.pdf*
14. *Advanced SQL.pdf*
15. *Functional Dependency Tutorial.pdf*
16. *Transactions.pdf*
17. *SQL Tutorial.pdf*
18. *Storage.pdf*
19. *Indexing.pdf*
20. *NoSQL.pdf*
21. *MapReduce.pdf*

Introduction

We use databases as it's more organised; hence it's easier to model and manage. It's more efficient, as it's fast to search, and update, and integration allows us to minimise data duplication. Concurrent (and therefore multi-user) access allows multiple people to access the database at the same time (will require some techniques).

Transactions are sequences of database actions that execute in a coherent, and reliable way - the classical properties are **ACID**. Consider the two transactions `T1`: $A = A - 100; B = B + 100$, and `T2`: $B = B - 100; A = A + 100$, we can observe ACID properties as follows;

- **atomicity** if one part of a transaction fails, the entire transaction fails

  on completion of `T1`, either $A' = A - 100$, and $B' = B + 100$, or $A' = A$, and $B' = B$, where the former is a successful transaction, and the latter is in the case of a failure.

- **consistency** transactions don't leave the database in an inconsistent state

  the sum of the balances must remain the same, such that $A' + B' = A + B$; we can also have more constraints such as keeping balances positive, or limiting the amount a transfer can do at once
- **isolation** transactions run as if no other transactions are running (may need to wait)

  given the two concurrent transactions `T1`, and `T2`, one has to be completed before the other can start
- **durability** results of successful transactions aren't lost on system failure

  the new values, A', and B' must persist if the transaction completes, even if the system fails (disk failure etc.)

A **Database Management System (DMBS)** creates new databases via a **Data Definition Language (DDL)**, which specifies the structure (**schema**). It also queries, and manipulates through a **Data Manipulation Language (DML)**. Examples of this include *PostgreSQL*, *MySQL*, *SQLite*, and can also fall under *NoSQL*, however SQL remains as the most widespread technology (as of writing this).

It lets us define, query, and manipulate databases with a high-levle declarative language (**Structured Query Language**). It's standardised by the ISO, but each DBMS implements its own variation of the standards, which may be costly if it's complex.

Relational Model

Consider the following model, represented as a table;

heading	`title:string`	`year:int`	`length:int`	`genre:string`
body	Gone with the Wind	1939	231	Drama
	Star Wars	1977	124	Science Fiction
	Wayne's World	1992	95	Comedy

We have the columns be the attribute, with the top row being the heading, and the rest being the body. The attributes are in the format `name:type`. The rows (of the body) are referred to as tuples. A relation is the heading as well as the body (the entire table). The heading is an **unordered set** of attributes, and an attribute is the name as well as the type (typically indivisible types). The body is an unordered set of tuples, and a tuple is the set of attribute values. The schema is for the entire relation is the name of the relation, and the heading, in this case, we'd have; `movies(title:string, year:int, length:int, genre:string)`, and a database is a collection of relations. A schema for a database is the schemas for all relations.

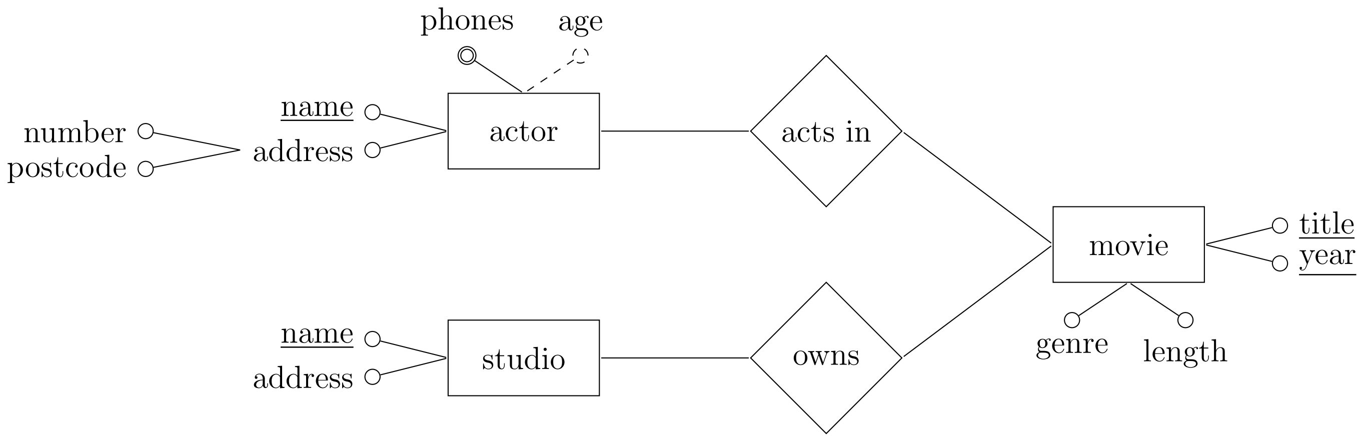
In mathematics, with a set of sets; $S_1, S_2, ..., S_n$, a relation R is a set of tuples $T_1, T_2, ..., T_n$, where $T_k \in S_k$, therefore $R \subset S_1 \times S_2 \times ... \times S_n$. As the idea of relations stems from set theory, it's important to note that the order in which we represent the attributes, and tuples in unimportant. In the Relational model we have **attributed** tuples, rather than **ordered**; R is the set of tuples $(A_1 : S_1 = T_1, ..., A_n : S_n = T_n)$, with $T_k \in S_k$. It's important to note that relations aren't 2-dimensional tables, even though it's more convineient to draw it on paper. We should instead consider them as a set of n-dimensional values, such that we have (`title:string`=StarWars, `year:int`=1997, `length:int`=127, `genre:string`=ScienceFiction), as a 4-dimensional movie value.

Entity Relationship Modelling

When a new database is being developed, it's important to try and model the real-world situation, instead of trying to refine it into an implementation, such as a relational model. In Entity-Relationship modelling, we try to create a diagram which represents the information needed for the database (the Entity Relationship Diagram). As there is no universally accepting notation for ER diagrams, we will use the following notation;

type	description	shape
entity sets	a set of distinguisable entries that share the same set of properties, can be physical (a room etc.), an event (flight, sale, etc.) - they normally correspond to nouns	rectangle
relationship sets	captures how **two or more** entity sets are related (e.g. owns, tutors), we can also have more than one relationship set between entity sets, and they can also have a relationship set on the same entity - they sometimes correspond to verbs	diamonds
attributes	properties of an entity; relationship sets can also have attributes, and primary keys are underlined	small circles

The movie example is represented below. Note that I've also extended it to contain different types of complex attributes. You can see that the address field is subdivided into number, and postcode, we have a multivalued attribute in phones, and a derived attribute in age (can be calculated from date of birth)


phones
age
number
postcode
name
address
actor
acts in
movie
title
year
genre
length
name
address
studio
owns