

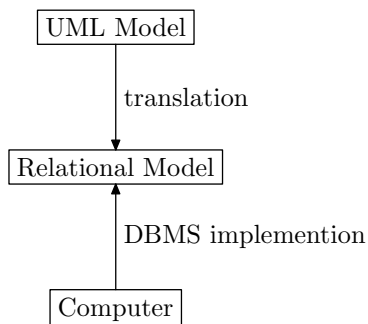
Lecture 4: Overview of Database Design and DBMS Implementation

the rest of the overview

- Modelling with UML
- Translating a UML Class Diagram to a Database Schema
- Normal Forms and Bottom-up Schema Synthesis
- File access methods
- Query Optimisation
- Transactions and Recovery

- There will be NO LECTURE next Wednesday 6 August (and probably other Wednesdays in lab-weeks, stay tuned)
- Greg will be taking the Tuesday 9am lab, not the Thursday 9am one as previously speculated
- Some people have not yet enrolled in a lab group, please do so before next week
- Why only 5 enrolments in the Friday 9am group???

Today: Everything but the Relations!



- Last Thursdays lectures introduced the middle of the three conceptual levels
- Today we overview the top and bottom
- This material will be covered in the 2nd half of the course

ER and Conceptual Modelling

- The entity-Relationship model, and its diagrams, were introduced by Chen in 1976 to simplify database design
- This began a huge research field “conceptual modelling”, which aims to capture requirements using “real world” concepts (entity, relationship) rather than computer or mathematical concepts

UML and Model Driven Development

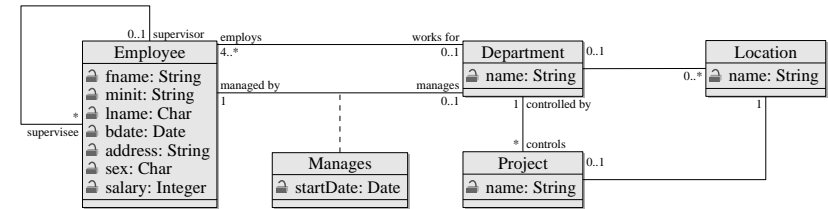
- conceptual modelling and object oriented ideas used by “methodologists” around 1990
- Mid 1990’s, main methodologists join forces creating the Unified Modelling Language (UML)
- Complete systems created using just UML and tools (way of the future?)
- Many diagram types, but we will only use simple class diagrams, mostly equivalent to ER
- (we ignore important semantic differences between ER and UML)
- I am biased, but I think the UML diagrams look much simpler and are easier to understand!

UML Elements - Class

The boxes represent classes. They represent a *kind of* thing that can exist in our scenario. Each class has many *objects*, its *instances*.

- the top compartment contains the class name
- the middle compartment contains the attributes (warning - same word, different context)
- the bottom compartment contains operations, but we will not use them

Lab Scenario in UML



The database we will use in the first lab is adapted from [E&N Figure 5.5, 5.6]. It is shown as a UML class diagram in [E&N Figure 3.16]. Here is an adapted (and corrected!) version to match our lab material.

UML Elements - Associations

The lines joining classes are associations. They represent a kind of relationship that can exist between objects of the end classes.

- The ends of the associations have “verb phrases”, that make sense when used with the classes at the end
- For example, an Employee can **work for** a Department
- Association ends can have *multiplicity constraints*, written (lower bound) .. (upper bound)
- An Association Class is an Association with its own attributes (or operations or behaviour)

Translation to Database Schema

We can design a database by

- 1 Creating a UML model of the subject matter
- 2 Translating the model into a database schema (manually or automatically)

Translation works like this:

- 1 each class becomes a relation
- 2 if an association has a finite upper bound at one end, it becomes attributes of the class at the far end (a foreign key)
- 3 otherwise, it becomes a relation

Though there are many options to choose from.

DBMS's and the Storage Hierarchy

- CPU's and memory go millions of times faster than hard disks
- it is worthwhile to think hard about minimising hard disk access
- and having the CPU work hard on this too
- there are a range of ways of organising and accessing data in files, each has costs and benefits, each is suitable for different situations
- there are alternative ways of processing any given query, DBMS can estimate their costs

Functional Dependencies and Normal Forms

- we saw that a relation with a primary key is a *function*, from the key attributes to the non-key attributes
- that is, the non-key attributes are *functionally dependent* on the key attributes
- functional dependency is the key idea behind the *normal forms* (first normal form 1NF, second normal form 2NF, ...)
- these are tests of database schema quality
- functional dependencies are also the main idea behind an alternative form of database design: schema synthesis, or “bottom-up” design
- the higher normal forms can cause performance problems though: in practice, “denormalised” relations are often used

File Access: Ordered

Relations are typically stored as files, with tuples as records, attributes as fields.

- records are stored in order of value for some field (eg student number)
- enable binary search
- retrieval of ranges
- insertion requires reorganisation

File Access: Hashed

- a function determines record location from value of some field
(eg last digit of student number)
- very quick access
- uses more space

File Access: Indicies

- like in a book
- its a separate file that maps field value to record location
- can have several indicies
- uses extra space
- must be maintained along with file
- different issues depending on whether index field(s) are key, ordering

Query Optimisation

$$a \times (b + c) = (a \times b) + (a \times c)$$

- each SQL query can be written as an expression of *relational algebra*
- like ordinary algebra, each term can be rewritten into different forms
- each form can have very different file access costs
- DBMS generate several candidate expressions, and estimates their processing cost, chooses the best

Transactions and Recovery

How often does a bank make a transaction on one of its ATM's?

- each transaction probably involves updating several relations in the database
- the bank does not want some of these updated executed and others not
- if something goes wrong in the middle of processing, need ability to back out
- also must be careful about using data that is mid-update
- eg: calculating total annual salary while another process gives everybody a pay-rise
- we will learn how the operations of several transactions can be safely scheduled, and “rolled back”

Tomorrow

More on SQL and the relational model, preparing for next weeks lab sessions.