Lecture 1

• Course Overview & Team
• Assessment
• Introduction to Databases
• Entity Relationship Models
• Case Study: PPID

Course Team

Lecturer: J Douglas Armstrong jda@inf.ed.ac.uk
Lab support: Lena Hansson s0235246@sms.ed.ac.uk
Database admin support: postgres-admin@inf.ed.ac.uk

Who is this course aimed at?

• ‘entry’ level start - no prior DBMS experience is assumed.
  – Will cover basics at a fast pace
  – Research orientated
• Practical Database design, implementation and use.
• Preparation to use DBMS systems in summer projects and beyond.
• Will require some basic programming.

Course Outcomes

• Demonstrate the ability to use and apply DBMS systems.
• Understand the underlying principles.
• Compare and contrast various relational and XML based solutions.
• Appreciate the roles and limitations of DBMS in commercial and research scenarios.

Applied Databases

Introduction

About me...

• Started in Biology (behaviour genetics)
• Got interested in databases (anatomy)
• Commercial and Academic Experience
• ‘wet lab’ and bioinformatics projects
• Office in FH (level E) and Research Lab at HRB

Course Outcomes

• Demonstrate the ability to use and apply DBMS systems.
• Understand the underlying principles.
• Compare and contrast various relational and XML based solutions.
• Appreciate the roles and limitations of DBMS in commercial and research scenarios.
Introduction to Databases

What is a Database?

- **Database** = a very large, integrated collection of data
- Models real-world organisation (e.g. enterprise, university, genome)
  - entities (e.g. students, modules, genes)
  - relationships (e.g. Armstrong is taking AD)
- A Database Management System (DBMS) is a software package designed to store and manage databases

Assessment

- One written assessment for 30% due at the end of the semester.
  - Choose a situation that requires or would benefit from using a DBMS
  - Design and implement the DBMS
  - Develop and test the required queries.
  - Build the appropriate middle-ware and user interface systems.
- Exam 70%
- Plagiarism will be refereed externally
- Late submissions will be penalised

Databases at Edinburgh

- e-Science centre
- National Data Curation Centre
- New DB faculty and further recruitments to come
- New DB courses
  - Advanced Databases
  - Querying and Storing XML
- Scottish Database Group email list (seminars)

Course Design

- Lectures cover essential background. Will generally last ~90 minutes with an optional mid session break.
- Labs to demonstrate essential code in supervised situation
- Later labs will have no set structure and are provided as drop-in support sessions.
- Guest lectures present research perspective on DBs
- Self-study and assignment designed to cover practical implementation

Guest Lectures

Content will be examined

- Natural Language Processing and Databases
  - TREK Genomics, Extraction of information from free text
- The Virtual Observatory
  - Astronomical databases, coordinate systems in Databases
- The MRC Mouse Atlas
  - 4D datasets (3D spatial information over time), Mapping structured text onto volumetric data, Volumetric Queries
- Distributed Database Systems
  - XML databases, web stripping, shared catalogues
Why use a DBMS

A DBMS provides generic functionality that otherwise would have to be implemented over and over again
- Data independence and efficient access
- Data integrity and security
- Uniform data administration
- Concurrent access, recovery from crashes
  = Reduced application development time

Why study Databases?

- Universal problem: Information handling
  - Finding info on the web
  - Storing data from high end scientific analysis
  - Financial organisation
- Datasets are increasing in diversity and volume
  - Meteorological information, particle accelerator traces, whole genomes
  - Need for new database technologies is exploding
- Databases have links to most of computer science
  - Operating systems, languages, theory, AI, multimedia etc

Data Models

- Data model: a collection of concepts for describing data
  - Relations, attributes, tuples (relational model)
  - Classes, subclasses, attributes, objects (object oriented)
  - Entities, relationships, attributes (entity-relationship)
- A schema is a description of a particular collection of data using a given data model
- The relational model of data is the most widely used model today
  - Main concept: relation - table with rows and columns
  - Every relation has a scheme which describes the table.

Levels of Abstraction

- many views,
- single conceptual (logical) schema
  - Views describe how users see the data
  - Conceptual Schema defines logical structure
- and single physical schema
  - Physical schema describes the files and indices used

Schemas are defined using data definition language (DDL)
data is modified/queried using data manipulation language (DML)

Example Database

- Conceptual schema:
  - Student (number: string, name: string, login: string, birthday: date, year: integer)
  - Module (id: string, name: string, credits: integer)
  - Enrolled (number: string, id: string, grade: string)
- Physical schema:
  - Relations stored as unordered files.
  - Index on first column of students
- External schema (View):
  - Module_info (id: string, enrollment: integer)

Data Independence

- Applications insulated from how data is structured and stored
- Logical data independence: Protection from changes in logical structure of the data
- Physical data independence: Protection from physical changes in the structure of the data
Concurrency Control

• Concurrent execution of user data requests is essential for good DBMS performance
  – Disk access is slow therefore most efficient access is for several users concurrently
• Interleaving actions of different user programs/requests can lead to inconsistency
  – e.g. simultaneous money being transferred out of an account
• DBMS ensures such problems do not occur!

Transactions

• A transaction is an atomic sequence of database actions
• Each transaction, executed completely must leave the DB in a consistent state (if the DB was consistent at the start of a transaction).
  – Users can specify simple integrity constraints which will be enforced by the DBMS
  – DBMS is limited in its ability to apply constraints
  – Ultimately DB consistency is the users/designers problem

Life cycle of a transaction

• A transaction is the execution of a program that accesses the DB starting with a BEGIN operation followed by a number of READ and WRITES and ends with COMMIT or ABORT.
  – i.e. transactions run through the following states:

Scheduling Transactions

• The DBMS ensures that execution of \( \{T_1, \ldots, T_n\} \) is equivalent to serial execution
• Before reading/writing an object a transaction requests a lock on the object and waits for the lock to be assigned by the DBMS. Locks are released at the end of the transaction
  – If \( T_i \) requests a lock on X that affects \( T_j \) then \( T_i \) will obtain the lock first and \( T_j \) is forced to wait until \( T_i \) completes.
  – If \( T_i \) already has a lock on Y and \( T_j \) later requests a lock on Y (deadlock) then \( T_i \) or \( T_j \) gets aborted and restarted.

Ensuring Atomicity

• High end DBMS ensure atomicity (all-or-nothing property) of transactions even if system crashes in the middle.
• Keeps a log (history) of all action carried out while being executed
  – Before a change is made to the DB, the corresponding log entry is forced to a safe location
  – After a crash, the effects of a partially executed transaction can be undone from the log

The DB Log

• The following actions are recorded in the log:
  – \( T_i \) writes an object: the old value and the new value (i.e. the log must extend back before the attempted change)
  – \( T_i \) commits/aborts: a log record indicates success/failure
• The log records are chained together by transaction id so its easy to undo a specific transaction in event of a crash or deadlock
• The log is often duplexed and archived
• All log activities are handled by the DBMS
Database People

- Users of query language interfaces
- DB application programmers (e.g. webmasters)
- Database designers
- Tool developers
- DBMS designers and implementers
- Database administrator (DBA)
  - designs logical/physical schemas
  - handles security and authorisation
  - data availability, crash recovery
  - database tuning

Armstrong, 2004
Applied Databases

DBMS structure

- A typical DBMS has a layered architecture
- Concurrency control and recovery not shown
- One of several possible variations

Armstrong, 2004
Applied Databases

Intro Summary

- DBMS used to maintain & query large datasets
- Benefits include recovery from crashes, concurrent access, quick application development, data integrity and security
- Levels of abstraction give data independence (3 schema architecture)
- A DBMS typically has a layered architecture
- Database research and development is one of the broadest areas in computer science and is constantly developing

Armstrong, 2004
Applied Databases

The Entity Relationship Model

Armstrong, 2004
Applied Databases

Database Design is a Process

Data requirements
Conceptual Data Model
Logical Schema
Storage and File system
Requirements analysis
Data analysis, conceptual design
Logical database design
Physical Database design

Armstrong, 2004
Applied Databases
Conceptual Design

The questions to ask when designing:
- What are the entities (objects, data points) in the organisation?
- What relationships exist between them?
- What are the rules of the organisation?
- What integrity constraints arise from these rules?

The answers are represented in an ER diagram.

Entities and Entity Sets

- Entity: An object distinguishable from other objects (e.g. a student, an employee, a gene sequence)
  - An entity is described by a set of attributes

- Entity Set: A collection of similar entities (e.g all students, employees, gene sequences etc)
  - All entities in an entity set have the same set of attributes
  - Each attribute has a domain
  - Each entity set has a key
    - (i.e. one or more attributes that uniquely identify an entity)

In programming terms
- Entity set == type definition
- Entity == variable

Relationships

- Relationship: An association among two or more entities (e.g. “Works_in” the accounts receivable department)
  - Relationships can be characterized by attributes

- Relationship Set: A collection of similar relationships
  - An n-ary relationship set R relates n entity sets \( E_1, \ldots, E_n \)
  - Each relationship in \( R \) involves entities \( e_1 \in E_1, \ldots, e_n \in E_n \)
Roles in Relationships

• The same entity set can
  - participate in different relationship sets
  - participate more than once in one relationship role (taking on different roles)

Key Constraints

• Works_In
  - an employee can work in many departments
  - a department can have many employees

• Manages
  - a department can have at most one manager

• This is expressed as a key constraint on Manages

Key Constraints and Ternary Relationships

“An employee works in at most one department and at a single location”

The meaning of general key constraints:
Suppose: Entity set E has a key constraint in relationship set R
Then: Each entity in an instance of E appears in at most one relationship of an instance of R

Participation Constraints

“Every department has a manager”

The participation constraint (= bold line) says:
Every department must occur in some manages relationship
or: The participation of Department in Manages is total.

Participation Constraints cont’d

“Every employee works in some dept, and every dept has employees”

The participation constraint can but need not be combined with key constraints

Weak Entities

“The dependents of an employee receive benefits as long as the employee stays with the company”

Every dependent is related to only one employee (key constraint)
Employee is a weak entity and Employees is the owner
Dependent has total participation in ‘Benefit’
Benefit is the identifying relationship
A dependent is identified by its name and employees ssn.
Conceptual Design using ER Model

- Design Choices
  - Should a concept be modeled as an entity or an attribute
  - Should a concept be modeled as an entity of a relationship
  - Identifying relationships: binary or ternary?

- Constraints in the ER Model:
  - A lot of data semantics can (and should) be captured
  - But some constraints cannot be captured in ER diagrams

Entity vs. Attribute

- Question: Should address be an attribute of Employee or an entity (connected to Employee by a relationship)?
- Answer: Depends on how you want to use address:
  - An employee may have several addresses
    - address must be an entity (since attributes cannot be set valued)
  - The structure (city, street, etc.) is important (e.g. which employees are stuck on the Glasgow train this morning?)
    - address must be an entity (since attribute values are atomic).

Entities vs. Attributes cont’d

- Works_In2:
  - we cannot record that an employee works for two or more periods in a department

- Works_In3:
  - we can record arbitrarily many work periods for every employee and department.

Similar to employee with several addresses: we want to record several values of the descriptive attributes for each instance of the Works_In relationship

Conceptual Design Summary

- Conceptual design follows requirements analysis
  - yields a high level description of data to be stored
- The Entity-Relationship Model is popular for conceptual design
  - Constructs are expressive and intuitive, i.e., close to the way people think about the application
- Basic constructs: entities, relationships and attributes
- There are many variations
- See Ramakrishan Chapter 2

Let’s start with a case study

- Proteins are the fundamental building blocks of life.
- Proteins work by forming specific shapes that can interact with each other and other molecules.
- Proteins act as catalysts (enzymes), structural scaffolds, channels, chemical signaling molecules etc.
Alcohol dehydrogenase

Converts toxic alcohol into even more toxic acetaldehyde (acetaldehyde is rapidly converted to safe chemicals)
Also converts meths into formaldehyde (embalming fluid)

Protein Interactions

- Individual Proteins form functional complexes
- These complexes are semi-redundant
- The individual proteins are sparsely connected
- The networks can be represented and analysed as an undirected graph

Ricin

- The Ricin toxin is made of TWO proteins: A and B
- Ricin A chain is a toxin
  - blocks the ability of cells to make new proteins
  - one molecule is sufficient to kill a cell
  - cannot enter cells unless injected
- Ricin B chain is not toxic
  - Ricin B chain binds onto external bits of cells
  - these get dragged into the cell bringing anything attached with them
- Ricin A and B form a complex that is ‘dangerous’.

The Ricin Complex

- The Ricin complex is SIMPLE
- It is non-redundant: removing either A or B chain effectively disables the toxic effect of the molecule.
- In transgenic biology, Ricin A and B have been used separately in many research situations since individually they are ‘safe’
- Other protein complexes are have redundancy and are extremely complex.

Yeast proteome

- The proteome is the entire set of proteins and their interactions that underlie living organisms.
- Studying protein interactions in yeast is easy compared to other species.
- Every gene (thus protein) is known
- Interaction maps have been created of the entire proteome.
- Can be represented as an undirected graph
The yeast proteome

Seth Grant’s Group
(Neuroscience, Edinburgh)

- Identified a series of key proteins involved in learning and memory in mammals.
- Using proteomic techniques they have isolated protein clusters containing these key proteins at the mammalian synapse.
- Identified each of the proteins in these complexes.

Synapse Protein Interactions

- Given the identity of the individual nodes in the network, we need to find the interactions (edges)
- Wealth of biochemistry literature on known protein interactions
- New high throughput techniques for finding potential interactions - publicly available

The Data

- Lists of proteins from several complexes
- Public databases have information on sequences, homologues etc
- Protein interactions can be mined from the literature.
  - Curated by Dr Holgar Husi
  - Collect binding pairs and literature links
- Create a ‘huge’ Excel spreadsheet with lots of complex information.
- Could be browsed and read but not queried.
Our first entity is going to be for Protein.

Proteins have more than one name (average 4.4).

The same protein in two species may be known by a different name.

- PSD-95 (mouse)
- DLG4 (human)
- NB: may have several acronyms in one species.

Deciding if one protein in one organism is the same as one in another organism is a complex art.

Therefore some part of Name(s) has to be an entity.

The users (biologists) want to have a 'main name' and a set of known synonyms that can be used in searches.

We need something to identify a protein so we use an internal unique identifier.

The Protein entity will have an attribute called PPID which must be unique (Primary Key).

Our main database identifier for a protein entity (primary key)

type: Attribute describing the class of protein, e.g. enzyme

OMIM: External reference number to the OMIM database

(Online Mendelian Inheritance in Man : http://www.ncbi.nlm.nih.gov/omim/)

So what about that network information we wanted to store in the first place?
Problem: cannot search on all Main names and synonyms at the same time easily.

Customise the primary name to the user group (e.g. clinician, rodent researcher) or on a user by user basis.

PPID development summary
- The PPID database (www.ppid.org) is under constant development
- The conceptual schema will probably be completely overhauled every 18 months
- Typical for databases in research active fields
  - Significant cost overhead in DBMS design expertise
- Entire database is running on PostgreSQL
- Porting to MySQL took a couple of hours

DBMS review
- Designing a database is a process that can be broken into lots of simple steps.
- Each design step should reflect the nature of the data and take into consideration what the database will be used for.
- The ER model is a common way to representing a database system when being designed
- The ER model is not tied to any specific DB product
Lab Class 1

- Appleton Tower Level 5
- Two ‘classes’: 11am and 12
  - 1100 Class: A-M
  - 1200 Class: N-Z
- If you want to swap, find someone to swap with.
  - Don’t let me know, just make sure you make it to one of the classes and priority for seats goes to those scheduled.

Lab Class 1

- You must have a DICE login
  - If you don’t then contact me urgently by late this afternoon.
    - I need to know why you don’t have a DICE account
    - Which dept etc you are affiliated with
    - What is your EUCS login id

SQL/DB Tutorials

- http://www.sqlcourse.com/

Finally

Just remember, database design is really an exercise in stating the bloody obvious - in a way that makes sense to the DBMS