Databases

A/Prof Alan Fekete
University of Sydney

Overview

• Data, databases, and database management systems
• Creating a database with a RDBMS
• Tradeoffs

Data

• Facts about the world that are worth knowing
  – Eg NGC 450 is at 18.876,-0.861
  – Eg The AAT has a 3.9 m reflector
  – The redshift of NGC 450 was measured with the AAT’s 2dF spectrograph
• Data is very valuable
  – Data can be analysed to produce theories, or to test them
  – It takes a lot of effort to gather data

Persistence

• Values of variables in a running program are stored in the memory (RAM)
  – They are lost when the program ends, or the machine crashes
• Computers have persistent storage (hard disk, flash memory etc) whose values persist
• The operating system organizes the persistent storage into files
  – So store valuable data in files

Sharing

• Once someone has collected data and got it into a computer, they can use it for many purposes, and many others can use it
• The naïve approach: copy the file around
  – Or email, or set up an ftp site, etc
• This is very bad, if the data might change
  – More data (more galaxies, more facts about each galaxy!)
  – Corrections to existing data
• Copies can become inconsistent; so share one store instead

A Database

• A store of data relevant to some domain, stored in a computer, and shared (between application programs or between users)
• Typically, the data is structured
  – Many facts in each of a few types
  – Eg many galaxies, each has name, ra, dec, redshift etc
Clients and servers

- The database server is the machine where the database is stored.
- Each user must run some presentation client code on their machine, to display the data.
  - Usually this is just a web browser.
- There may be other machines in between:
  - E.g., a web server, an application server.
  - Note that a machine may be a server which receives some requests, and also a client of others as it tries to respond!
- We focus on the database server, and the “database client” where the application code is running that understands the data.

Two approaches

- **Purpose-written application code,** accessing data stored in files, using knowledge of the structure of the files.
  - This is common in science.

- **Use a database management system (DBMS) which manages the data and provides higher-level facilities to access it.**
  - You may write applications that call the DBMS, or even use it directly, interactively.
  - This is standard in industry, and is becoming more frequent in science.

DBMS Features

- A query language:
  - Describe the data to retrieve.
  - Can also describe updates etc.
- A data definition language:
  - Describe the logical structure of the data.
- Storage management:
  - Control the layout in files, and with facilities to improve performance.
- Access control.

Different Data Models

- Each database has a model or schema which tells what sort of data is stored:
  - E.g., each galaxy has id, name, ra, dec, etc.
- Using a DBMS means that the data model must fit the style supported by the DBMS.
- The most common DBMSs all support the relational style of data model.

Relational Data Models

- Database consists of tables, each with a name.
- Each table has many rows, each with exactly the same structure:
  - Several attributes, each with a name and type (integer, float, string of length 20, etc).
  - Each row has a value for each attribute.
- Rows are distinguished by the values of a primary key (one or more attributes).

Example

- **An Instance**

  **Schema:**
  - galaxy (gal_id INTEGER, gal_ra FLOAT, gal_dec FLOAT, gtype STRING)
  - observation (obs_id INTEGER, obs_ra FLOAT, obs_dec FLOAT, obs_flux FLOAT, gal_id INTEGER, source STRING)

  **An Instance**
  
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>13.1</th>
<th>7.2</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>14.7</td>
<td>5.3</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>11.2</td>
<td>4.8</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15.3</td>
<td>4.6</td>
<td>S</td>
<td></td>
</tr>
</tbody>
</table>

  **Note use of “foreign key” to connect information in different tables**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>14.6</th>
<th>5.4</th>
<th>13.4</th>
<th>2</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11.2</td>
<td>4.9</td>
<td>9.6</td>
<td>3</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>14.5</td>
<td>4.3</td>
<td>13.3</td>
<td>2</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>11.1</td>
<td>4.8</td>
<td>9.7</td>
<td>3</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>
Views

• You can define a view on a database
• Just like a table, but the data isn’t inserted or stored explicitly
  – It is computed when needed from the other tables
• A view can be used in queries just like a table

Queries

• User can write a query, and run it
  – Answer is a collection of relevant data
• Query may be typed interactively, or written into a program for repeated use
  – Perhaps with parameters that vary
• Answer may be viewed on the screen, imported into a program, printed, or stored back in the database

SQL

• A standardized language in which queries can be written against databases in relational DBMSs
  – But each platform has slight variations
• SQL is declarative
  – Query describes what data is needed
  – RDBMS has the job of working out how to trawl through the database to compute the answer

Creating a database

• Decide on the platform to use
• Decide on the schema that will represent your data
• Create a database and tables
• Load the data
• Allow users to access it

Relational DBMS platforms

• Enterprise solutions (Oracle, IBM DB2, SQL Server) are very expensive
  – And require professional administration too
• Vendors also offer light-weight cheap/free variants
• Free open-source platforms: PostgreSQL, MySQL, Cloudscape

Table Design

• One table for each type of object in the domain
  – Columns for each attribute of importance
  – Make sure there is a primary key
    • Invent a new identifier for this purpose if necessary
• Use foreign keys to represent relationships between objects
  – Or have a separate “association table” where one object can be related to many, in each direction
Normalisation

• It is important that the schema doesn’t allow the same facts to be stored repeatedly in different rows, or different tables
  – Eg observation(obs_id, obs_ra,...,gal_id, gal_ra, ...
  – This would risk inconsistency when updates occur
• There is theory about this, but sensibly defining one table per object will avoid the problems

Indices

• The performance of queries can vary greatly depending on how the data is stored in the DBMS
• Having an index on a column usually speeds up queries that select rows with given values (or ranges of values) in that column
  – SQL has CREATE INDEX statement

Permissions

• Each DBMS can control which users are allowed to run different sorts of statements on each table separately
• Typically, for scientific data one allows SELECT access to all users (or all registered users)
  – INSERT access to a few trusted users
  – DELETE, UPDATE only for the administrators
  • Usually have a separate account, so even the admin people can’t do this accidentally while acting as scientists

Loading data

• SQL has INSERT statement which adds rows to a table
• If data is already in a file in a well-known format (eg csv), then DBMS has non-standard commands to import it directly into a table

Trade-offs

Compared to writing applications against data stored directly in files, using DBMS to manage the data:
• Allows users to perform unpredicted (ad-hoc) queries
  – without being programmers, and without knowing the structure in files
• Gives better support as data schema evolves
  – Existing queries can continue to run
• Has more predictable performance
  – Easier to avoid very slow execution
  – But you may not be able to get very fast execution
• Has better security
  – But considerable overhead in administration