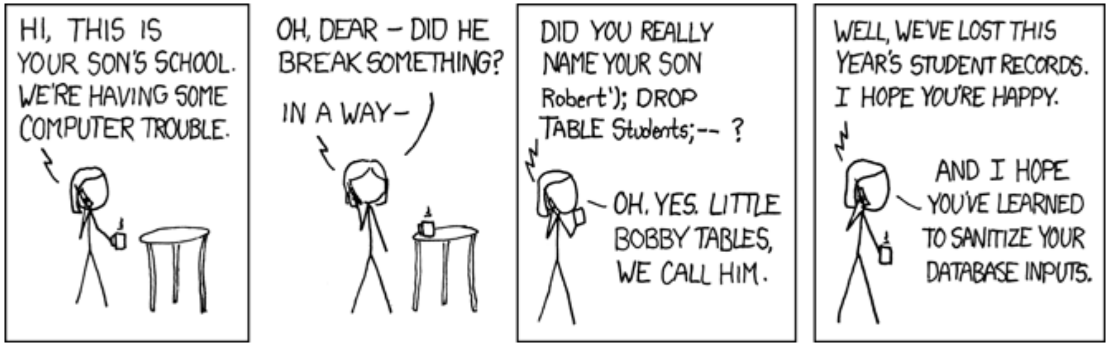
structured data

- Programmatically we deal with arrays, maps, lists, sets, queues, trees, graphs, …
- No unique solution to working with data - depends on the problem being addressed (and personal preference)
- Data can be regarded as structured or connected

what is a database?

- A structured collection of data residing on a computer system that can be easily accessed, managed and updated
- Data is organised according to a database model
- A Database Management System (DBMS) is a software package designed to store and manage databases



why use a dbms?

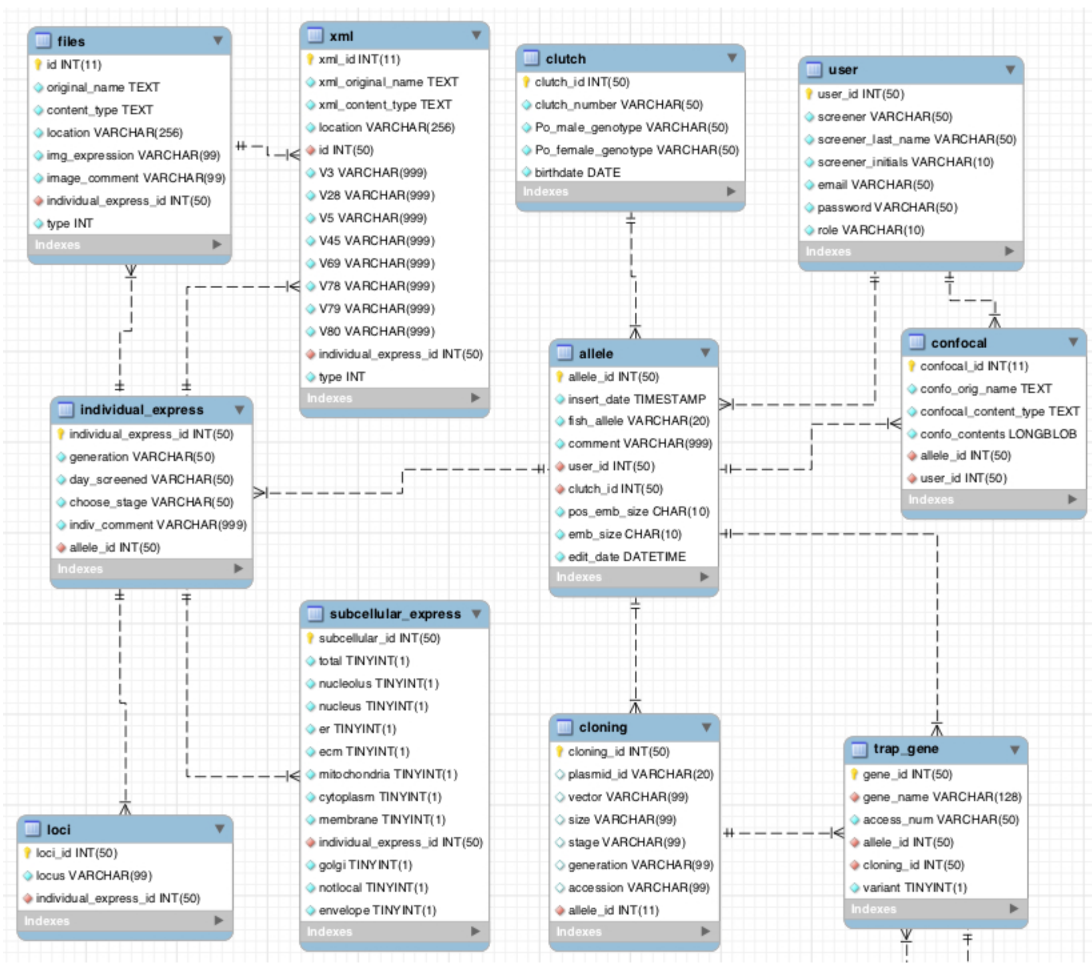
- data independence
- efficient and concurrent access
- data integrity, security and safety
- uniform data administration
- reduced application development time
- data analysis tools

scale of databases

- "DBs own the sweet spot of 1GB to 100TB" (Gray & Hey, 2006)
  - SQLite
  - MySQL, PostgreSQL
  - SQLServer, Oracle
  - *Hive/HadoopDB, SciDB, Redis, MonetDB, NuoDB

- A collection of concepts describing how structured data is represented and accessed
- Within a data model, the **schema** is a set of descriptions of a particular collection of data
- The schema is stored in a **data dictionary** and can be represented in SQL, XML, RDF, etc.
- In semantics a data model is equivalent to an ontology - "a formal, explicit specification of a shared conceptualisation"

data model example



flat (file) model

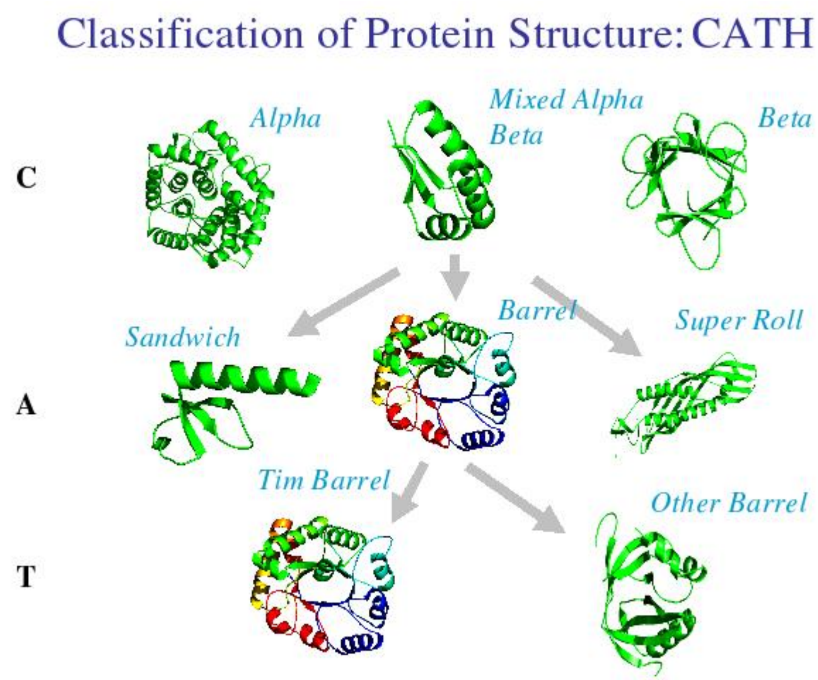
- Data files that contain records with no structural relationships
- Additional information is required to interpret these files such as the file format properties
- Hollerith 1889 patent "Art of Compiling Statistics" describes how every US resident can be represented by a string of 80 characters and numbers
- Examples: delimiter-separated data (CSV, TSV), HTML table

hierarchical model

- Data is organized in a tree structure
- Levels consist of records of the same type - same set of field values - with a sort field to ensure a particular order
- 1:N (parent-child) relationship between two record types: child may only have one parent but a parent can have many children
- Popular in the late 1960s/1970s with IBM's Information Management System (IMS)
- Structure of XML documents

hierarchical example

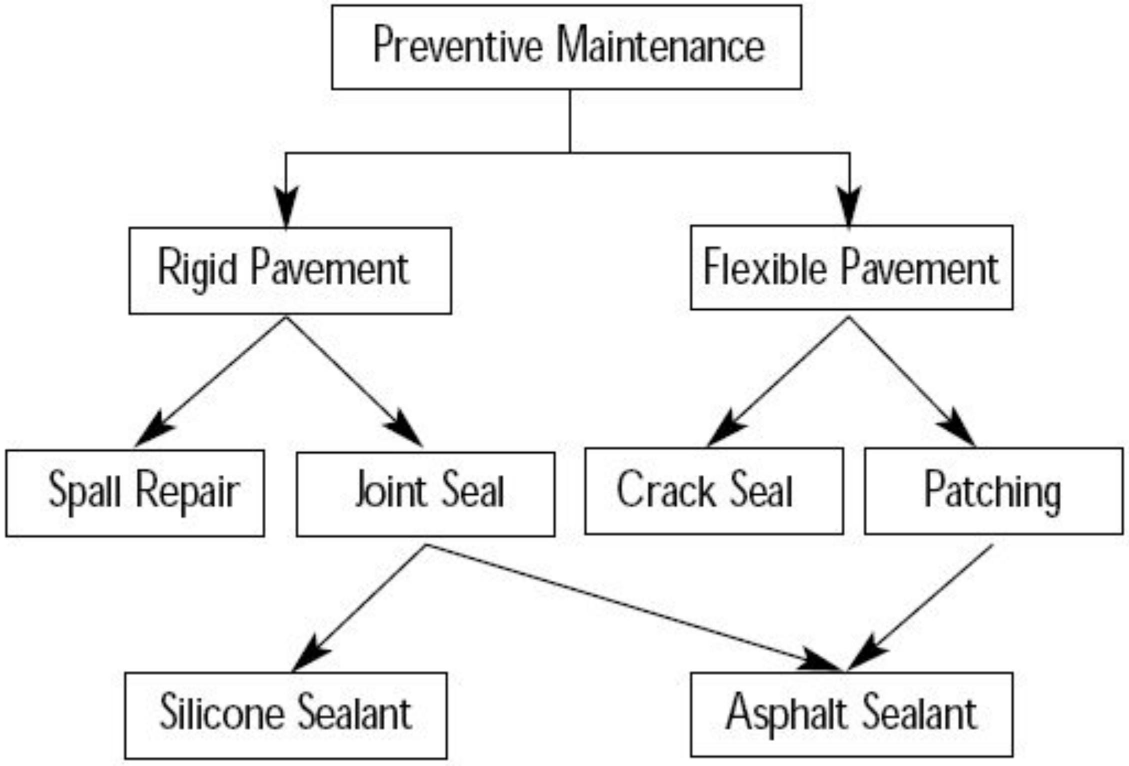
CATH database of protein structures in the Protein Data Bank:
Levels: Class, Architecture, Topology, Homologous Superfamily, Sequence Family



network model

- Data is organized as sets and records
- A set has an owner, a name and one-or-more members
- A record may be an owner in any numbers of sets and a member in any number of sets
- Allows modelling of many-to-many relationships
- Formally defined by the Conference on Data Systems Languages (CODASYL) specification in 1971

Network Model



object-oriented model

- Adds database functionality to object-oriented languages by allowing persistent storage of programming objects
- Avoids overhead of converting information from database representation to application representation (impedance mismatch)
- Applications require less code and use more natural data modelling
- Good for complex data and relationships between data

object-oriented example

Object-Oriented Model

Object 1: Maintenance Report

Date	
Activity Code	
Route No.	
Daily Production	
Equipment Hours	
Labor Hours	

Object 1 Instance

01-12-01
24
I-95
2.5
6.0
6.0

Object 2: Maintenance Activity

Activity Code	
Activity Name	
Production Unit	
Average Daily Production Rate	

relational model

- Data is organized as **relations**, **attributes** and **domains**
- A **relation** is a table with columns (attributes) and rows (tuples)
- The **domain** is the set of values that the attributes are allowed to take
- Within the relation, each row is unique, the column order is immaterial and each row contains a single value for each of its attributes
- Proposed by E. F. Codd in 1969/70

Relational Model

Activity Code	Activity Name
23	Patching
24	Overlay
25	Crack Sealing

Key = 24

Activity Code	Date	Route No.
24	01/12/01	I-95
24	02/08/01	I-66

Date	Activity Code	Route No.
01/12/01	24	I-95
01/15/01	23	I-495
02/08/01	24	I-66

associative model

- Data is modelled as entities - having a discrete independent existence - and associations
- It is organised as items - an identifier, name and type - and links - an identifier, source, verb and target

Example:

Flight BA1234 arrived at LAX on 10-Apr-12 at 1:25pm

Items: Flight BA1234, LAX, 10-Apr-12, 1:25pm, arrived at, on, at

Links: (((Flight BA1234 arrived at LAX) on 10-Apr-12) at 1:25pm)

other models

- semi-structured model
  - -- graph-based for information that cannot be constrained by schema, e.g., Web
- object-relational
  - -- adds object capabilities to relational systems, e.g., GIS data

relational model

- Data is organized as **relations**, **attributes** and **domains**
- A **relation** is a table with columns (attributes) and rows (tuples)
- The **domain** is the set of values that the attributes are allowed to take
- Within the relation, each row is unique, the column order is immaterial and each row contains a single value for each of its attributes
- Proposed by E. F. Codd in 1969/70

- An atomic sequence of actions (read/write) in the database
- Each transaction has to be executed **completely** and must leave the database in a consistent state
- If the transaction fails or aborts midway, the database is "rolled back" to its initial consistent state

Example:
Authorise Paypal to pay $100 for my eBay purchase:
- Debit my account $100
- Credit the seller's account $100

By definition, a database transaction must be:

- **A**tomic: all or nothing
- **C**onsistent: no integrity constraints violated
- **I**solated: does not interfere with any other transaction
- **D**urable: committed transaction effects persist

- DBMS ensures that interleaved transactions coming from different clients do not cause inconsistencies in the data
- It converts the concurrent transaction set into a new set that can be executed sequentially
- Before reading/writing an object, each transaction waits for a **lock** on the object
- Each transaction releases all its locks when finished

locks

- DMBS can set and hold multiple locks simultaneously on different levels of the physical data structure

- Granularity: at a row level, page (a basic data block), extent (multiple array of pages) or even an entire table

  - Exclusive vs. shared
  - Optimistic vs. pessimistic

- Ensures atomicity of transactions

- Recovering after a crash, effects of partially executed transactions are undone using the log

- Log record:
  - -- Header (transaction ID, timestamp, ...)
  - -- Item ID
  - -- Type
  - -- Old and new value

partitions

- Horizontal: different rows in different tables
- Vertical: different columns in different tables (normalisation)
- Range: rows where values in a particular column are inside a certain range
- List: rows where values in a particular column match a list of values
- Hash: rows where a hash function returns a particular value

- First normal form: no repeating elements or groups of elements table has a unique key (and no nullable columns)
- Second normal form: no columns dependent on only part of the key

Star Name | Constellation | Area

- Third normal form: no columns dependent on other non-key columns

Star Name | Magnitude | Flux

structured query language

- Appeared in 1974 from IBM
- First standard published in 1986; most recent in 2008
- SQL92 is taken to be default standard
- Different flavours:

Microsoft/Sybase	Transact-SQL
MySQL	MySQL
Oracle	PL/SQL
PostgreSQL	PL/pgSQL

SELECT *selectionList* FROM *tableList* WHERE *condition* ORDER BY *criteria*

```
SELECT name, constellation FROM star WHERE dec > 0
    ORDER by vmag

SELECT * FROM star WHERE ra BETWEEN 0 AND 90

SELECT DISTINCT constellation FROM star

SELECT name FROM star LIMIT 5
    ORDER BY vmag
```

- Inner join: combining related rows

```
SELECT * FROM star s INNER JOIN stellarTypes t ON s.stellarType = t.id

SELECT * FROM star s, stellarTypes t WHERE s.stellarType = t.id
```

- Outer join: each row does not need a matching row

```
SELECT * from star s LEFT OUTER JOIN stellarTypes t ON s.stellarType = t.id

SELECT * from star s RIGHT OUTER JOIN stellarTypes t ON s.stellarType = t.id

SELECT * from star s FULL OUTER JOIN stellarTypes t ON s.stellarType = t.id
```

COUNT, AVG, MIN, MAX, SUM

```
SELECT COUNT(*) FROM star

SELECT AVG(vmag) FROM star

SELECT stellarType, MIN(vmag), MAX(vmag) FROM star
    GROUP BY stellarType

SELECT stellarType, AVG(vmag), COUNT(id) FROM star
    GROUP BY stellarType
    HAVING vmag > 14
```

CREATE DATABASE *databaseName*
CREATE TABLE *tableName* (name1 type1, name2 type2, . . .)

```
CREATE TABLE star (name varchar(20), ra float, dec float, vmag float)
```

Data types:
- boolean, bit, tinyint, smallint, int, bigint;
- real/float, double, decimal;
- char, varchar, text, binary, blob, longblob;
- date, time, datetime, timestamp

```
CREATE TABLE star (name varchar(20) not null, ra float default 0, ...)
```

keys

```
CREATE TABLE star (name varchar(20), ra float, dec float, vmag float,
    CONSTRAINT PRIMARY KEY (name))
```

- A primary key is a unique identifier for a row and is automatically not null

```
CREATE TABLE star (name varchar(20), ..., stellarType varchar(8),
    CONSTRAINT stellarType_fk FOREIGN KEY (stellarType)
    REFERENCES stellarTypes(id))
```

- A foreign key is a referential constraint between two tables identifying a column in one table that refers to a column in another table.

SHOW …

```
SHOW TABLES
```

```
SHOW INDEXES IN star
```

```
SHOW WARNINGS
```

DESCRIBE

```
DESCRIBE star
```

INSERT INTO *tableName* VALUES(val1, val2, ...)

```
INSERT INTO star VALUES('Sirius', 101.287, -16.716, -1.47)

INSERT INTO star(name, vmag) VALUES('Canopus', -0.72)

INSERT INTO star
    SELECT ...
```

LOAD DATA INFILE *"path/to/file"* INTO TABLE *tableName* FIELDS TERMINATED BY *"delimiter"*

```
LOAD DATA INFILE "data.csv" INTO TABLE star FIELDS TERMINATED BY ","

SELECT * INTO OUTFILE "/tmp/star" FIELDS TERMINATED BY "," FROM star
  WHERE vmag > 16
```

DELETE FROM *tableName* WHERE *condition*
TRUNCATE TABLE *tableName*
DROP TABLE *tableName*

```
DELETE FROM star WHERE name = 'Canopus'

DELETE FROM star WHERE name LIKE 'C_n%'

DELETE FROM star WHERE vmag > 0 OR dec < 0

DELETE FROM star WHERE vmag BETWEEN 0 and 5
```

update

UPDATE *tableName* SET *columnName* = val1 WHERE *condition*

```
UPDATE star SET vmag = vmag + 0.5

UPDATE star SET vmag = -1.47 WHERE name LIKE 'Sirius'

UPDATE star INNER JOIN temp on star.id = temp.id SET star.vmag = temp.mag
```

ALTER TABLE *tableName* ...

```
ALTER TABLE star ADD COLUMN bmag double AFTER vmag

ALTER TABLE star DROP COLUMN bmag
```

CREATE VIEW *viewName* AS …

```
CREATE VIEW region1View AS
    SELECT * FROM star WHERE ra BETWEEN 150 AND 170
        AND dec BETWEEN -10 AND 10

SELECT id FROM region1View WHERE vmag < 10

CREATE VIEW region2View AS
    SELECT * FROM star s, stellarTypes t WHERE s.stellarType = t.id
        AND ra BETWEEN 150 AND 170 AND dec BETWEEN -10 AND 10

SELECT id FROM regionView2 WHERE vmag < 10 and stellarType LIKE 'A%'
```

CREATE INDEX *indexName* ON *tableName*(*columns*)

```
CREATE INDEX vmagIndex ON star(vmag)
```

- A clustered index is one in which the ordering of data entries is the same as the ordering of data records
- Only one clustered index per table but multiple unclustered indexes
- Typically implemented as B+ trees but alternate types such as bitmap index for high frequency repeated data

CREATE PROCEDURE *procedureName* @param1 type, . . .
AS ...

```
CREATE PROCEDURE findNearestNeighbour @starName varchar(20) AS
BEGIN
    DECLARE @ra, @dec float
    DECLARE @name varchar(20)
    SELECT @ra = ra, @dec = dec FROM star WHERE name LIKE @starName
    SELECT name FROM getNearestNeighbour(@ra, @dec)
END

EXEC findNearestNeighbour 'Sirius'
```

DECLARE *cursorName* CURSOR FOR SELECT ...
OPEN *cursorName*
FETCH *cursorName* INTO ...
CLOSE *cursorName*

- A cursor is a control structure for successive traversal of records in a result set
- Slowest way of accessing data

cursors example

- For each row in the result set, update the relevant stellar model

```
DECLARE @name varchar(20)
DECLARE @mag float
DECLARE starCursor CURSOR FOR
    SELECT name, AVG(vmag) FROM star
        GROUP BY stellarType
OPEN starCursor
   FETCH starCursor INTO @name, @mag
   EXEC updateStellarModel @name, @mag / CALL updateStellarModel(@name, @mag)
CLOSE starCursor
```

CREATE TRIGGER *triggerName* ON *tableName* ...

A trigger is procedural code that is automatically executed in response to certain events on a particular table:

- INSERT
- UPDATE
- DELETE

```
CREATE TRIGGER starTrigger ON star FOR UPDATE AS
    IF @@ROWCOUNT = 0 RETURN
    IF UPDATE (vmag) EXEC refreshModels
GO
```

- Python:

```
import MySQLdb
Con = MySQLdb.connect(host="127.0.0.1", port=1234, user="mjg",
    passwd="mjg", db="test")
Cursor = Con.cursor()
sql = "SELECT * FROM star"
Cursor.execute(sql)
Results = Cursor.fetchall()
...
Con.close()
```