Cornell University
Center for Advanced Computing

Data Formats and Databases

Steve Lantz
Senior Research Associate
Cornell CAC

Workshop: Data Analysis on Ranger, December 8, 2010

How will you store your data?

- Raw binary is compact but not portable
  - "Unformatted," machine-specific representation
  - Byte-order issues: big endian (IBM) vs. little endian (Intel)
- Formatted text is portable but not compact
  - Need to know all the details of formatting just to read the data
  - 1 byte of ASCII text stores only a single decimal digit (~3 bits)
  - Most of the fat can be knocked out by compression (gzip, bzip, etc.)
  - However, compression is impractical and slow for large files
- Need to consider how data will ultimately be used
  - Are you trying to ensure future portability?
  - Will your favored analysis tools be able to read the data?
  - What storage constraints are there?

Issues beyond the scope of this talk…

- Provenance
  - The record of the origin or source of data
  - The history of the ownership or location of data
  - Purpose: to confirm the time and place of, and perhaps the person responsible for, the creation, production or discovery of the data
- Curation
  - Collecting, cataloging, organizing, and preserving data
- Ontology
  - Assigning types and properties to data objects
  - Determining relationships among data objects
  - Associating concepts and meanings with data (semantics)
- Portable data formats can and do address some of these issues…

Portable data formats: the HDF5 technology suite

- Versatile data model that can represent very complex data objects and a wide variety of metadata
- Completely portable file format with no limit on the number or size of data objects in the collection
- Free and open software library that runs on a range of platforms, from laptops to massively parallel systems, and implements a high-level API with C, C++, Fortran 90, and Java interfaces
- Rich set of integrated performance features that allow for optimizations of access time and storage space
- Tools and applications for managing, manipulating, viewing, and analyzing the data in the collection

Source: www.hdfgroup.org/hdf5

Features lending flexibility to HDF5

- *Datatype definitions* include information such as byte order (endian) and fully describe how the data is stored, insuring portability
- *Virtual file layer* provides extremely flexible storage and transfer capabilities: Standard (Posix), Parallel, and Network I/O file drivers
- *Compression and chunking* are employed to improve access, management, and storage efficiency
- *External raw storage* allows raw data to be shared among HDF5 files and/or applications
- *Datatype transformation* can be performed during I/O operations
- *Complex subsetting* reduces transferred data volume and improves access speed during I/O operations

Source: www.hdfgroup.org/hdf5

Portable data formats: netCDF

- NetCDF (network Common Data Form) is a set of software libraries and machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data
- The free and open netCDF distribution contains the C/C++/F77/F90 libraries, plus the netCDF utilities ncgen and ncdump
- NetCDF for Java is also available (standalone)
- Many other interfaces to netCDF data exist: MATLAB, Objective-C, Perl, Python, R, Ruby, Tcl/Tk
- There is a well-developed suite of software tools for manipulating or displaying netCDF data
- Compatibility with HDF5 was introduced in netCDF-4

Source: http://www.unidata.ucar.edu/software/netcdf

Properties of netCDF data

- *Self-Describing*. A netCDF file includes information about the data it contains
- *Portable*. A netCDF file can be accessed by computers with different ways of storing integers, characters, and floats
- *Direct-access*. A small subset of a large dataset may be accessed without first reading through all the preceding data
- *Appendable*. Data may be appended to a properly structured netCDF file without copying the dataset or redefining its structure
- *Shareable*. One writer and multiple readers may simultaneously access the same netCDF file
- *Archivable*. NetCDF will always be backwards compatible

Source: http://www.unidata.ucar.edu/software/netcdf

Advantages of netCDF

- NetCDF has been around longer, especially in the climate, weather, atmosphere, and ocean research communities (source is UCAR)
- NetCDF has nice associated tools, especially for geo-gridded data
  - *Panoply* (http://www.giss.nasa.gov/tools/panoply/) focuses on the presentation of geo-gridded data. It is written in Java and is platform independent. The feature set overlaps with ncBrowse and ncview.
  - *Ferret* (http://ferret.wrc.noaa.gov/Ferret/) offers a Mathematica-like approach to analysis. New variables and expressions may be defined interactively; calculations may be applied over arbitrarily shaped regions; geophysical formatting is built in.
  - *Parallel-NetCDF* (http://trac.mcs.anl.gov/projects/parallel-netcdf/) is built upon MPI-IO to distribute file reads and writes efficiently among multiple processors.

Portable data formats: Silo for visualization

- Silo is actually a library which implements an API for writing scientific data to binary disk files
- It's the primary file format for VisIt
- Silo upports point meshes, structured meshes, curves, etc.
- Internally, an I/O driver (typically HDF, but also PDB and netCDF) actually reads and writes the files
- Fortran, C, and Python interfaces are provided with the library
  - There is an independent Python interface called Pylo (http://mathema.tician.de/software/pylo) which offers some enhancements for easy use
- Silo builds easily on most POSIX systems

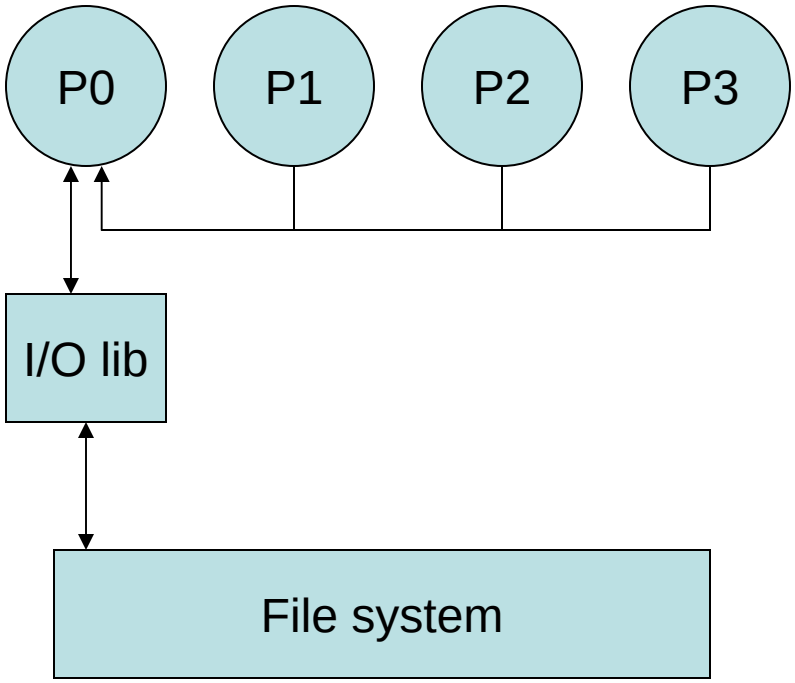
Using Silo

- Silo is a serial I/O library but can be effectively used in what is called "Poor Man's Parallel I/O"
  - Silo files may contain namespaces (directories) within a single file
  - A "multi-block" object can be instantiated which spans multiple files
  - Applications can then divide processors into groups in which each group writes a separate file
  - Each processor with a group writes to its own group (serially within the group; parallel across groups)
  - A single processor is then responsible for writing multi-block to tie the various files together.
  - Silo hides all of this from you with a set of functions to conceal the operations

Writing a mesh file in Silo

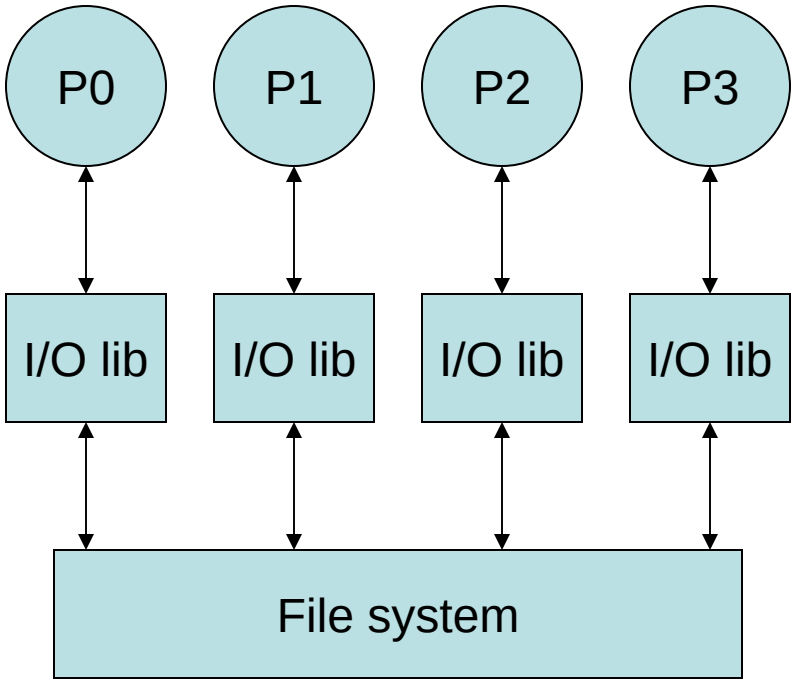
```
#include <silo.h>

DBfile     *file = NULL; /* Silo file pointer */
file = DBCreate("sample.silo", DB_CLOBBER, DB_LOCAL, NULL,
        DB_PDB);

/* Name the coordinate axes 'X' and 'Y' */
coordnames[0] = strdup("X");
coordnames[1] = strdup("Y");


/* How many nodes in each direction? */
dimensions[0] = 4;
dimensions[1] = 4;

/* Assign coordinates to coordinates array */
coordinates[0] = [-1.1,-0.1,...]
coordinates[1] = [-2.4,-1.2,...]

/* Write out the mesh to the file */
DBPutQuadmesh(file, "mesh1", coordnames, coordinates,
    dimensions, 2, DB_FLOAT, DB_COLLINEAR, NULL);

DBClose(file);
```

Path from serial to parallel I/O – part 1



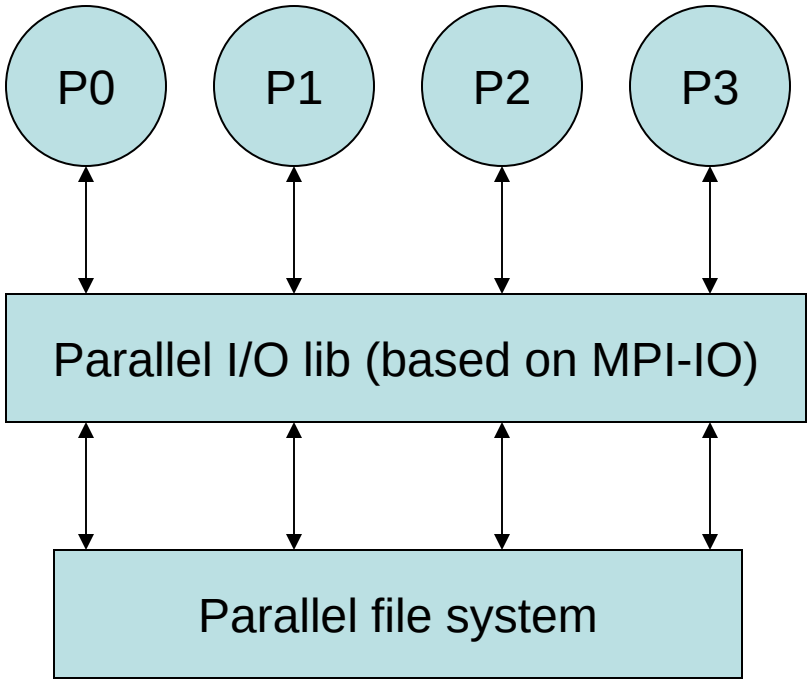
- P0 may become bottleneck
- System memory may be exceeded on P0

Path from serial to parallel I/O – part 2



- Possible to achieve good performance
- May require post-processing
- More work for applications, programmers

Path from serial to parallel I/O – part 3



- Main point: either HDF5 or netCDF can take the place of the parallel I/O library - they've already linked the parallel I/O library for you
- Variant: only P1 and P2 act as parallel writers; they gather data from P0 and P3 respectively (chunking)

Accessing TACC files from Linux (or vice versa)

- SSHFS: the SSH File System
  - The sshfs client lets you mount directories located on a remote server, allowing you to work with files and directories as if they were local
  - The remote machine only needs to be running a simple ssh/sftp server
  - The client uses sftp to get files as you look at them
  - Current implementation is a FUSE (Filesystem in USErspace) plugin

- XUFS : eXtended User-space File System
  - It allows a user's $HOME directory on a personal Linux workstation to "follow" them when remotely connecting to systems via ssh
  - The software was developed by TACC staff; you must contact them directly if you wish to download

What's a database?

- Everyone is familiar with the concept of a database, but variants abound; it's easy to be confused. Fundamentally, it's…

A structured collection of data

- Enterprise-class relational databases:
  - Oracle, MySQL, PostgreSQL, Microsoft SQL Server
- Small, light relational databases:
  - SQLite, SmallSQL
- Special purpose databases:
  - Apache Derby, Gadfly, Cayuga
- Object databases (layered on other databases)
- Data warehouses
- Federated databases

Why use databases?

- They have built-in data integrity checks
  - Management of row duplication
  - Enforcement of data ranges and types
- They encourage forethought about the data to be stored
  - What are the fields of the data?
  - Can some values be NULL?
- They provide lots of features
  - SQL connectors in most languages
  - Advanced query capabilities
  - Thread safety for transactional operations
- They may help your application scale up
  - A naïve flat file search won't scale

Let's represent some data

- Let's say we want to write a Facebook application that allows people to show their arXiv.org papers on their Facebook profile page
  - True example! A CAC staff member was recently asked to do this
- The application needs to store information about papers in such a way that it can extract papers based on queries about authors
- Conceptually we have a couple of objects we want to connect:
  - Authors (Facebook ID, arXiv info, etc.)
  - Papers (title, abstract, journal reference, etc.)
- Let's look at some representations of this data

Table-based flat file

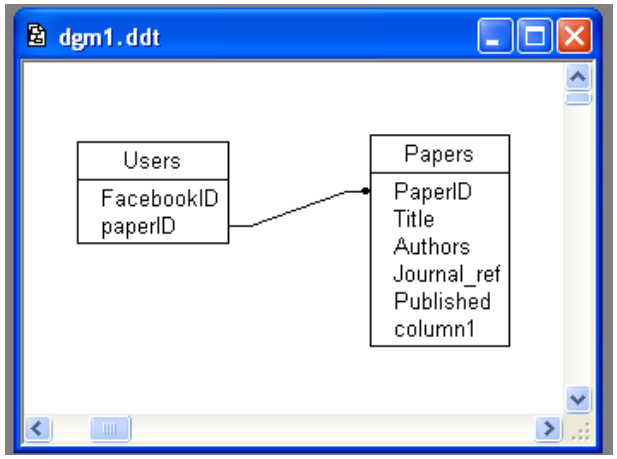
- Add a row to the table for every unique paper an author has written
- Search the table for all rows that have the appropriate ID

Facebook ID	Paper ID	Paper Title	Paper Authors
2341234	http://arxiv.org/abs/1234	Particle Pleasantry	CH Foo, BC Lars
2341234	http://arxiv.org/abs/3234	Reading is neat	CH Foo, RG Fields
1234123	http://arxiv.org/abs/4321	Science in Teaching	DS Henry, RG Fields
1234123	http://arxiv.org/abs/3234	Reading is neat	CH Foo, RG FIelds

- Problems:
  - A row is duplicated for every author of a paper (e.g., *Reading is neat*)
  - To match a given Facebook ID, a linear scan of the entire file is required
- Benefits:
  - Easy to add new entries (depending on sorting)
  - Simple to code the read/write functions

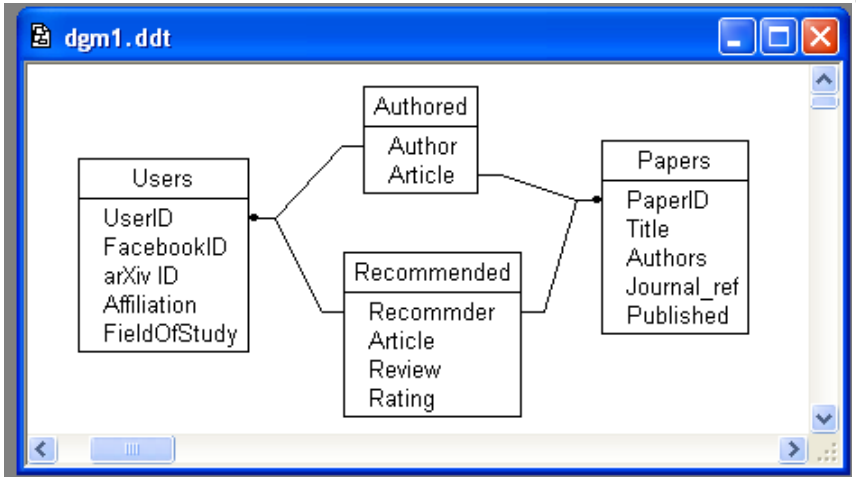
Relational database

- Define *two* tables plus a link between them
- One table is Users/Authors; the other is Papers
- The link is between the two ***paperID***s
  - ***paperID*** in Users is called a ***Foreign Key*** because it points to a row in another table
  - ***PaperID*** in Papers is called a ***Primary Key*** because it uniquely identifies a row
- Benefits:
  - Fast location of papers from author
  - Easy to add fields, papers and authors
- Problems (eased by software):
  - Database management



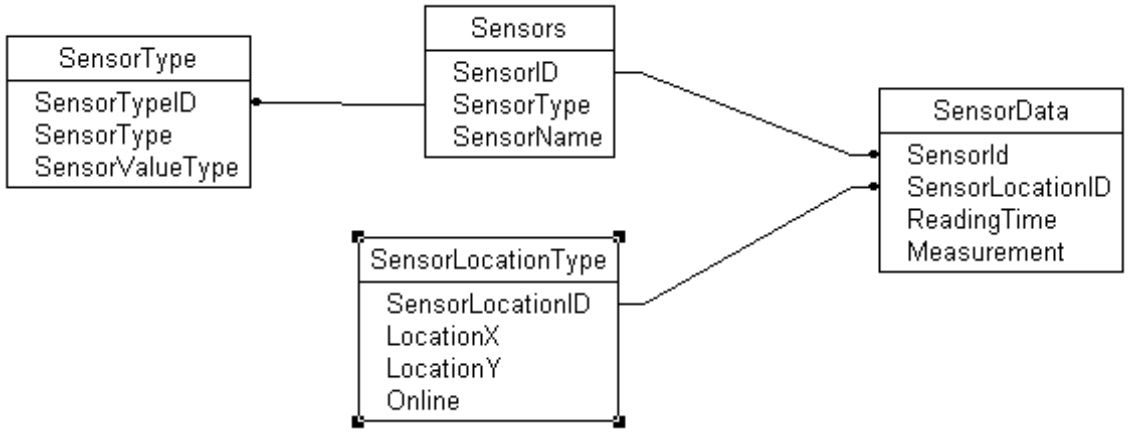
More relationships

- We can create representations that are much more fined-grained
  - Make the Users table (fewer rows, more fields) separate from Authored
  - Provide a Recommended table to enable reviews
  - Provide a Reading table that users can update
- ***Join Table*** – table that contains links to other tables (Foreign Keys) and perhaps other information about the relationship.



Relational data

- Relational databases are based on the *relational model*
- Practically this means that data can be expressed by a set of binary relationships
  - This is commonly seen in scientific data involving metadata that would need to be replicated for every row of data
  - The replication gets worse when the metadata is hierarchical.



How do you decide?

- Flat files are useful for:
  - Small amounts of data
  - Static dumping of data
- Databases are useful for:
  - Constantly updating/evolving data
  - Data where searching/querying is important/complex
  - Expressing relationships that are not captured in a row-based table
  - Threading/transactions
- Other factors to consider:
  - Size of data (cost/expertise)
  - Expectations about sharing data

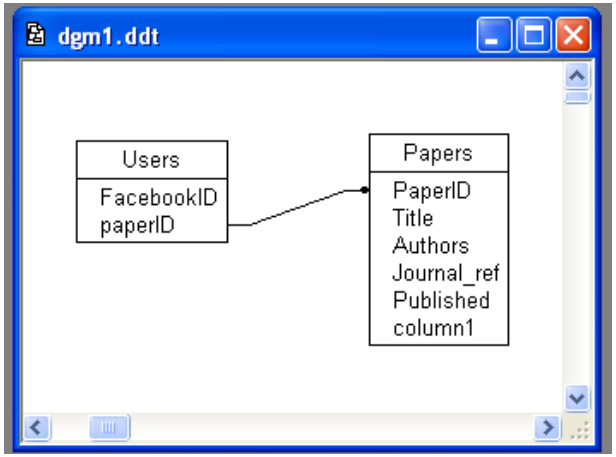
Talking to a database

- Talking to a database requires a software connector that allows you to speak SQL to the database.
- **SQL – Structured Query Language**
  - SQL is a computer language designed for the creation, management, modification and retrieval of data from a database
  - Essentially all databases speak SQL, though many also provide some form extensions (which are less standard).
  - Using a database generally requires some basic knowledge of SQL
- **PL/SQL** and **SQL/PSM**
  - These are database extensions that provide stored procedures in the database, allowing some functionality to be moved into it
  - This is the MOST abused part of database usage

SQL language – select

- To introduce the language, we will go over some basic queries that you would perform using our simplest table design
- First, retrieve the title of a specific paper:

```
SELECT Title FROM Papers
        WHERE PaperID = 200
```



- Retrieve an entire row:

```
SELECT * FROM Papers
        WHERE PaperID = 200
```

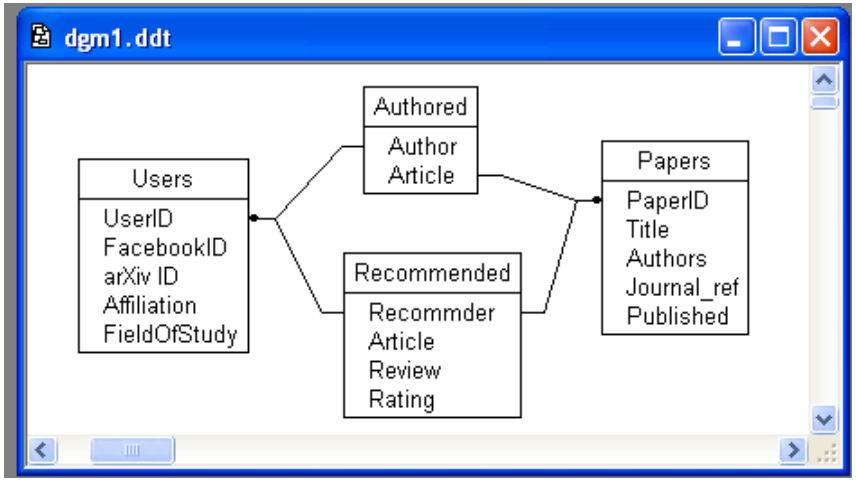
- Retrieve a paper with a “Henry” author:

```
SELECT * FROM Papers
        WHERE Authors LIKE  ‘%Henry%’
```

SQL language – join

- Retrieve the list of articles authored by a particular user:

```
SELECT UserID FROM Users
        WHERE FacebookID = “id123”
```



- Now get the authored article ids:

```
SELECT article FROM Authored
        WHERE Author = “UserID”
```

- Now get the papers:

```
SELECT paperID,Title FROM Papers WHERE PaperID IN (‘1234’,’4321’)
```

- Alternatively, use the ***JOIN*** keyword:

```
Select paperID, title from Papers
        INNER JOIN Authored.Article=Papers.PaperID
        Where (Authored.Author = “UserID”)
```

SQL language – insert, update, commit

- SQL also allows inserting new rows into the database as well as updating exisiting rows.
- Insert a new Paper for an existing Author

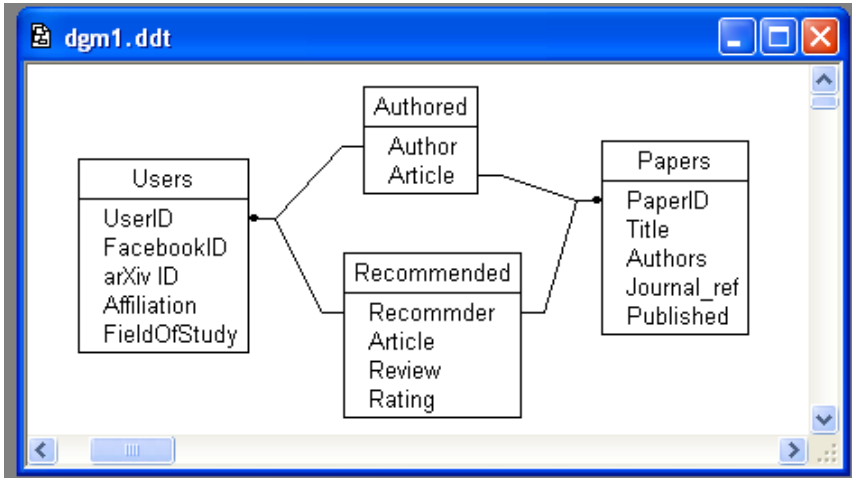
```
INSERT INTO Papers (PaperID, Title, Authors)
        Values(5678,'Really neat stuff','RG Fields')
```

```
INSERT INTO Authored
        Values(12345678,5678)
```

- Update the journal_ref

```
UPDATE Papers
        SET Journal_ref="someref"
        WHERE PaperID='5678'
```

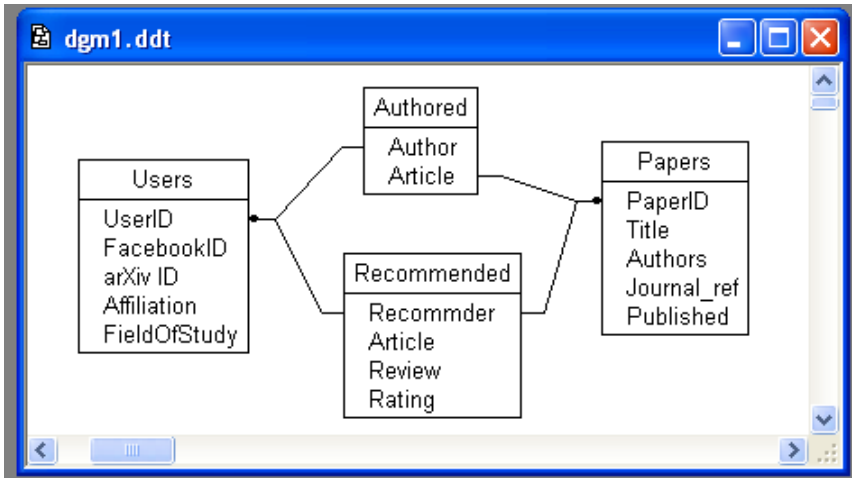
```
COMMIT
```



SQL language – delete, order, etc.

- Dropping rows from the table is just like SELECTing, where rows are selected using a WHERE clause.

```
DELETE FROM Authored
        WHERE Article='1234'
```



- Functions can be used so that records are retrieved in a way meaningful to what you're doing

```
SELECT PaperID,Authors FROM Papers
        ORDER BY Published
```

```
SELECT PaperID,Authors FROM Papers
        WHERE Journal_ref LIKE '%Chimica%'
        AND Published > somedate
```

Using SQL in application code

- All languages out there have SQL connectors which are software modules that provide a connection to the database and a cursor

```
import MySQLdb
conn = MySQLdb.connect(host="h", user="u", passwd="p321", db="test")
cursor = conn.cursor()
cursor.execute("SELECT * FROM Papers WHERE paperID = '1234'")
row = cursor.fetchone()
cursor.execute ("SELECT * FROM Authored WHERE Author = '1234567'")
row = cursor.fetchall()
cursor.close()
conn.close()
```

- Note: Many connectors have a special executeQuery function which returns an iterable to retrieve rows (res->next())

SQL language assessment

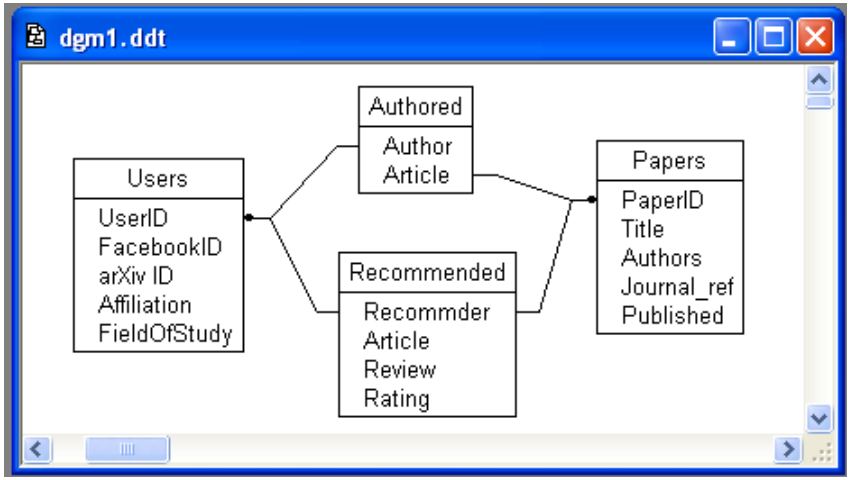
- Benefits:
  - SQL is a relatively simple language, its learning curve is very gentle
  - Connectors exist from every language to every type of database; all reasonable databases support SQL; therefore SQL is a ubiqutious choice
  - Lines of code can be drastically reduced by taking advantage of powerful SQL commands for searching and retrieving objects from the database
- Problems:
  - SQL queries can turn out to be amazingly inefficient even though it is not obvious why they are inefficient; you may need to play around with a query to optimize it
  - It's yet another language to learn

Object-relational mapping

- OR mapping – When you just don't have time for SQL
- Object-relational mapping (also ORM and O/R mapping) converts data between a database and an object-oriented programming language
- An ORM tool lets you create and use a database within a standard OO programming paradigm
  - Database tables are created from class definitions
  - SQL queries are basically written for you by the tool, which can be highly beneficial in most cases
- The ORM tools also allow you direct SQL access in cases where optimized queries are needed

OR mapping – create script

```
BEGIN;
CREATE TABLE 'Users' (
    'UserID' integer NOT NULL PRIMARY KEY;
    'FacebookID' integer NOT NULL,
    'arxivID' varchar(100) NOT NULL,
    'Affiliation' varchar(100) NOT NULL,
    'FieldOfStudy' varchar(100) NOT NULL
)
;

CREATE TABLE 'Papers' (
    'PaperID' integer NOT NULL PRIMARY KEY;
    'Title' varchar(128) NOT NULL;
    'Authors' varchar(128) NOT NULL;
)
;

CREATE TABLE 'Authored' (
    'id' integer AUTO_INCREMENT NOT NULL PRIMARY KEY,
    'Author' integer NOT NULL,
    'Article' FOREIGN KEY REFERNCES Papers(PaperID)
)
;
ALTER TABLE 'Authored' ADD CONSTRAINT Author_refs_id
    FOREIGN KEY ('Author') REFERENCES 'Users'('UserID');

COMMIT;
```



- A script is often used to generate the database (so it can be regenerated as needed)
- Generally it will look something like the script on the left

OR mapping – data structure

- For an OR Mapper, we specify the structure of the data using classes and member variables.
- Things like null-ability, default values, and foreign keys are specified in a simpler fashion.

Specify the primary key (otherwise one is generated)

This means varchar(128)

This field can be NULL

```
class Users(Model):
    UserID = models.IntegerField(primary_key=True)
    FacebookID = models.CharField(max_length=128, null=False)
    arxivID = models.CharField(max_length=128)
    Affiliation = models.CharField(max_length=128, null=True)
    FieldOfStudy = models.CharField(max_length=128, default='HEP')

class Papers(Model):
    PaperID = models.IntegerField(primary_key=True)
    Title = models.CharField(null=False)
    Authors = models.CharField(null=False)
    ...

class Authored(Model):
    Author = models.ForeignKey(Users)
    Article = models.ForeignKey(Paper)
    otherData = models.CharField(null=True)
```

OR mapping – programming

- The notation for dealing with an OR-mapped version is relatively simple but has several important features:
  - Transactions/sessions are managed by the mapper
  - Type checking is enforced by the language rather than at runtime in SQL.
  - Changing data tables means just changing a class structure.

```
#Create or Update a new Users row
u = Users(UserID="0770", FacebookID="1241341234", arxivID="user_21")
#If a UserID == 0770 exists, then we are doing an update of FacebookID and
#arxivID fields. But we need to commit it.
u.save()
#Find the user matching a UserID
qs = Users.objects.filter(UserID__exact==someuid)
#a "queryset" is returned which we should test that it
#actually returned something, but we won't.
auser = qs[0]
#Use Foreign keys
qs = auser.Authored_set.select_related()
#This queryset contains a list of papers authored by auser
for paper in qs:
    print paper.Title
```

Summary – databases

- Databases can be an effective way to improve your ability to share and manage your data.
- Databases and database technologies are increasingly embedded in a variety of systems and the technology stacks to support easy use of these systems are increasingly omnipresent.
- Database languages and tools can help reduce the amount of code you manage in your projects.