



# Stashing Your Data

## Data Access, Formats and Bases

Nate Woody



## 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...



## 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?



## 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](http://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](http://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

*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.





## Silo Files – Visualization

- Silo is actually a library which implements an API for writing scientific data to binary disk files.
- Primary file format for VisIt.
- Supports Point meshes, structured meshes, curves, etc.
- Silo uses an I/O driver (typically HDF, but also PDB and netcdf) to actually read and write the files.
- Fortran, C, and Python interfaces are provided with the library and builds easily on most POSIX systems
  - An independent python interface called Pylo (<http://mathema.tician.de/software/pylo>) which offers some enhancements for easy use.



## Silo - Using

- Silo is a serial I/O library but can be effectively used in what is called “Poor Mans Parallel I/O”.
  - Silo files can contain namespaces (directories) with a single file and a “multi-block” object can be instantiated which can span multiple files.
  - Applications then divide processors into groups and each group writes a separate file.
  - Each processor with a group writes to it’s own group (serially with-in the group, parallel across groups), a single processor can then be responsible for writing multi-block to tie the various files together.
  - Silo hides all of this from you with a set of functions to hide the operations.



## Silo – Writing a mesh file

```
#include <siloh.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;

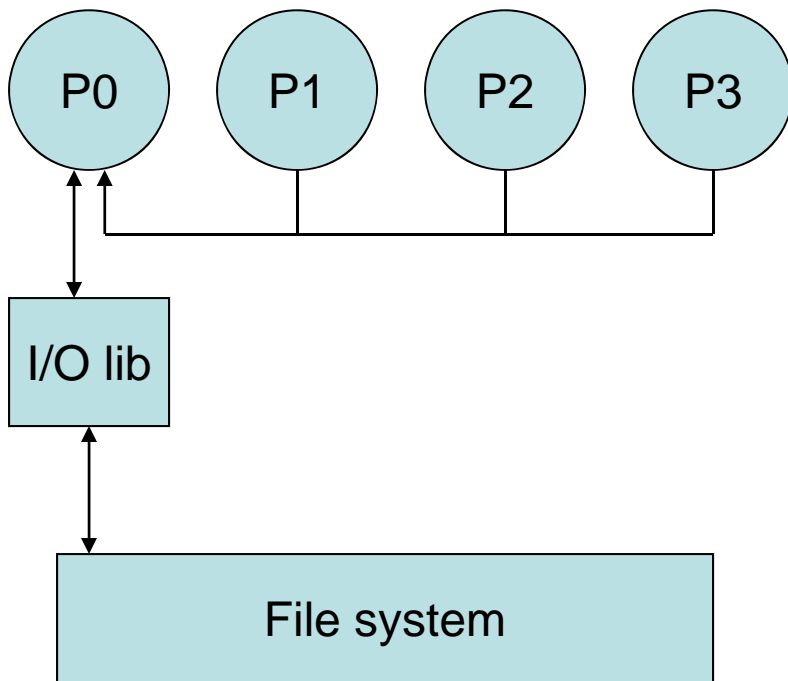
/* Assign coordinates to coordinates array */
coordinates[0] = [-1.1,-0.1,...]
coordinates[1] = [-2.4,-1.2,...]

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

DBClose(file);
```



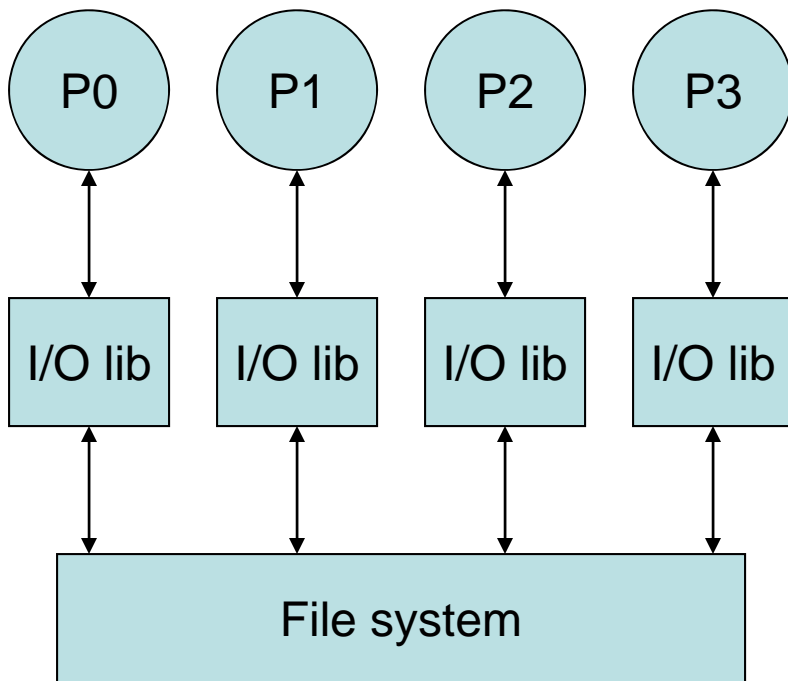
## Path from serial to parallel I/O – part 1



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



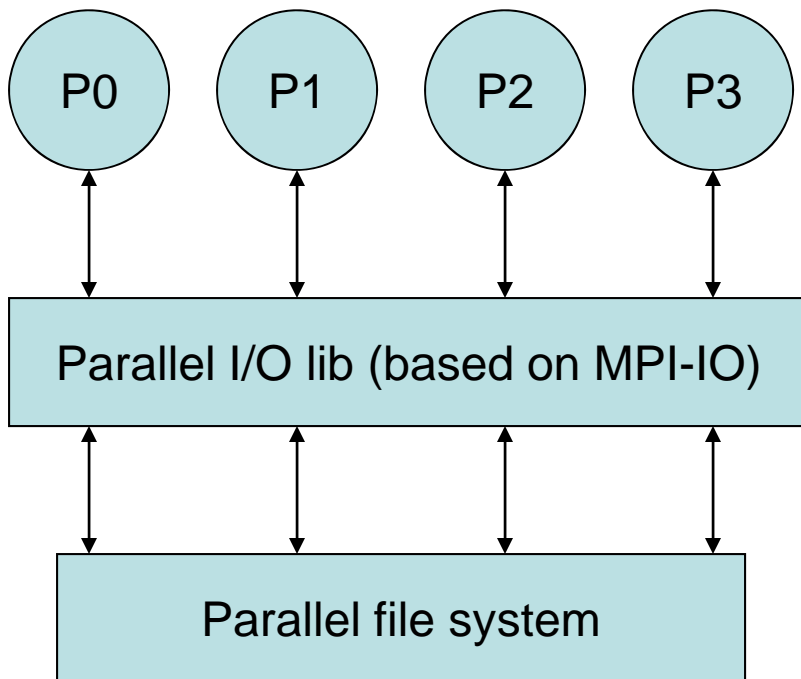
## Path from serial to parallel I/O – part 2



- 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
- Variant: only P1 and P2 act as parallel writers; they gather data from P0 and P3 respectively (chunking)



## XUFS

- sshfs on steroids, and backwards

```
[ajd27@v4linuxlogin1 ~]$ xufs/bin/ussd tg123123@ranger.tacc.utexas.edu
```

```
Password:
```

```
login3% pwd
```

```
/share/home/00933/tg459569/xufs-rhome
```

```
login3% ls -la
```

```
total 15340
```

```
drwx----- 15 tg459569 G-80907 4096 Mar 27 15:14 .
```

```
drwxr--r-- 23 tg459569 G-80907 4096 Mar 27 15:14 ..
```

```
drwxr-xr-x  2 tg459569 G-80907 4096 Mar 27 15:14 Desktop
```

```
drwxr-xr-x  2 tg459569 G-80907 4096 Mar 27 15:14 VTune
```

```
drwxrwxrwx  2 tg459569 G-80907 4096 Mar 27 15:14 WINDOWS
```

```
drwxrwxrwx  2 tg459569 G-80907 4096 Mar 27 15:14 bin
```

```
drwxrwxrwx 20 tg459569 G-80907 4096 Mar 27 15:14 dev
```



## XUFS Appropriateness

- Similar to GPFS-WAN, sshfs, and many others, but...
- You already have a fair amount of disk space on your home machine.
- You don't want two copies of your code floating around.
- No need for a lightning-fast synchronization when writing.
- Sharing among accounts at TG institution is rare.
- With striped gridftp underneath, there is no loss of efficiency.





## What's a database

- Everyone is familiar with the concept of a database, but there are a lot of variants out there, don't be confused.

***A structured collection of data***

- Enterprise class relational databases:
  - Oracle, MySQL, PostgreSQL, SQLServer
- Small light relational databases:
  - SQLite, SmallSQL
- Special Purpose Databases
  - Apache Derby, GadFly, Cayuga
- Object Databases (layered on other databases)
- Data warehouse
- Federated Databases



## Why would you use databases

- Data integrity checks
  - Manage row duplication
  - Enforce data ranges and types
- Forces you to think about the data you're going to store up front
  - What are the fields of the data?
  - Can some values be NULL?
- Lots of provided features
  - SQL connectors for most any language is available.
  - Advanced query capabilities are provided for you.
  - Thread safety/Transactional operations
- Highly extensible as your application scales up
  - A naïve flat file search won't scale



## Let's represent some data


- I recently needed to write a facebook application for arXiv.org, which allows people to show their arXiv papers on their profile page.
- We need to store in a web application the information about papers that people have authored that can be queried to extract the papers for a particular user.
- So we have a couple of objects:
  - 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

- The simplest thing is to create a table that contains a row 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	<a href="http://arxiv.org/abs/1234">http://arxiv.org/abs/1234</a>	Particle Pleasantry	CH Foo, BC Lars
2341234	<a href="http://arxiv.org/abs/3234">http://arxiv.org/abs/3234</a>	Reading is neat	CH Foo, RG Fields
1234123	<a href="http://arxiv.org/abs/4321">http://arxiv.org/abs/4321</a>	Science in Teaching	DS Henry, RG Fields
1234123	<a href="http://arxiv.org/abs/3234">http://arxiv.org/abs/3234</a>	Reading is neat	CH Foo, RG Fields

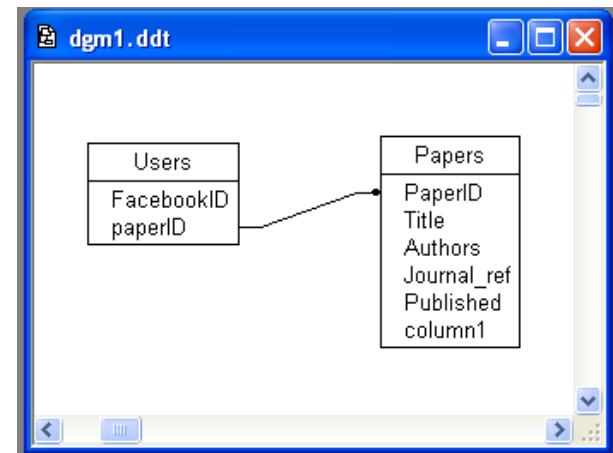


- **Problems:**
  - Duplicate rows for every author of a paper (*Reading is neat*).
  - Linear scan of file for matching facebook ID.
- **Benefits:**
  - Easy to add new entries (depending on sorting)
  - Write the read/write functions while sleeping.



## Relational Database

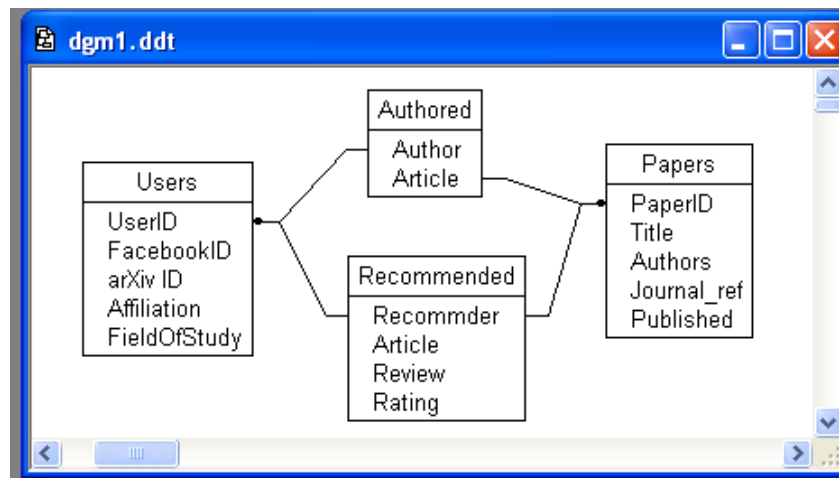
- This particular instance is well-suited for a relational database where we define two tables and a link between them.
- Since we have more papers than Users/Authors, we create a table that contains all paper information and a separate table that contains an entry for every authored paper.
- **paperID** in Users is called a **Foreign Key** which just means it points to a row in another table.
- **PaperID** in Papers is called a **Primary Key** which means it uniquely identifies a row.
- Benefits:
  - Fast location of papers from author
  - Easy to add fields, papers and authors
- Problems:
  - Database management





## More relationships

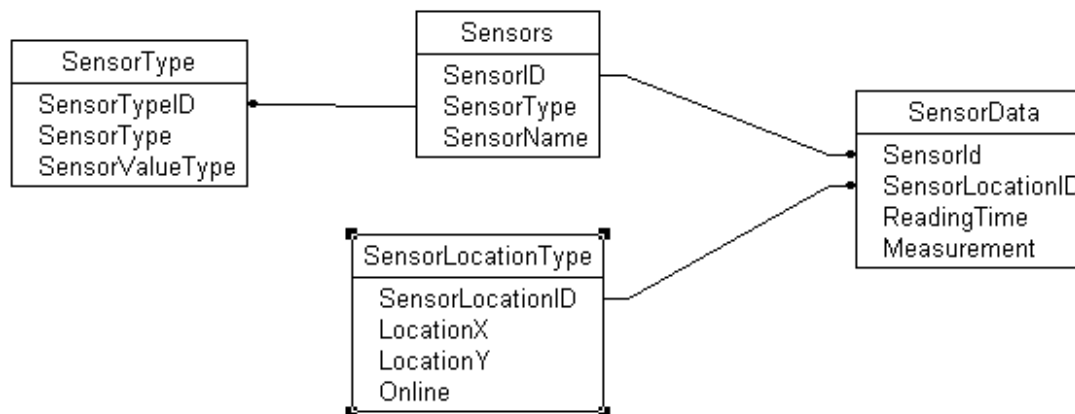
- We actually have much more fined-grained representations we would like to make. Authored is just one example.
- Recommended – Papers I read and wrote a review for.
- Reading – Papers I'm currently reading/interested in.
- **Join Table** – table that contains links to other tables and perhaps other information about the relationship. This contains Foreign Keys and possible other fields.





## 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 around metadata that would need to be replicated for every row of data. This replication get's worse when the metadata is hierarchical.





## How do you decide?

- I tend to use flat files when
  - Small amounts of data
  - Static dumping of data
- I tend to use a database when
  - Constantly updating/evolving data
  - Where searching/querying data is important/complex
  - When row based tables stink (relationships, etc)
  - Threading/transactions
- Other factors to consider
  - Size of data (cost/expertise)
  - What are the expectations on 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**
  - Computer language designed for 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 requires some basic knowledge of the SQL language.
- **PL/SQL** and **SQL/PSM** – Database extensions that provide stored procedures in the database. This allows control-of-flow constructs to be used and allows you to move some functionality into the database.
  - This is the MOST abused function in database usage.



## SQL language - Select

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

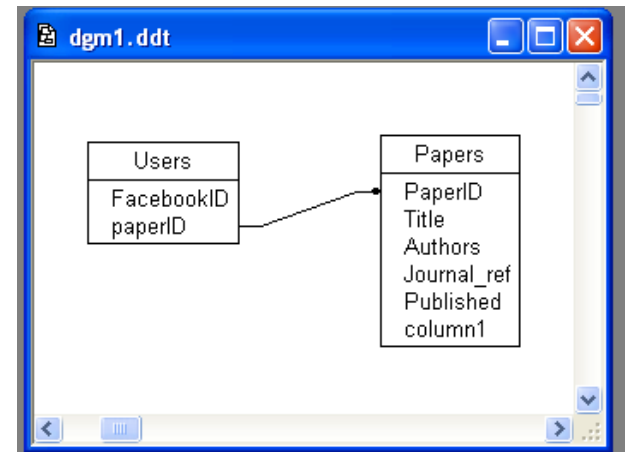
```
SELECT Title FROM Papers  
WHERE PaperID = 200
```

- 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 = "someid"
```

- Now get the authored article ids

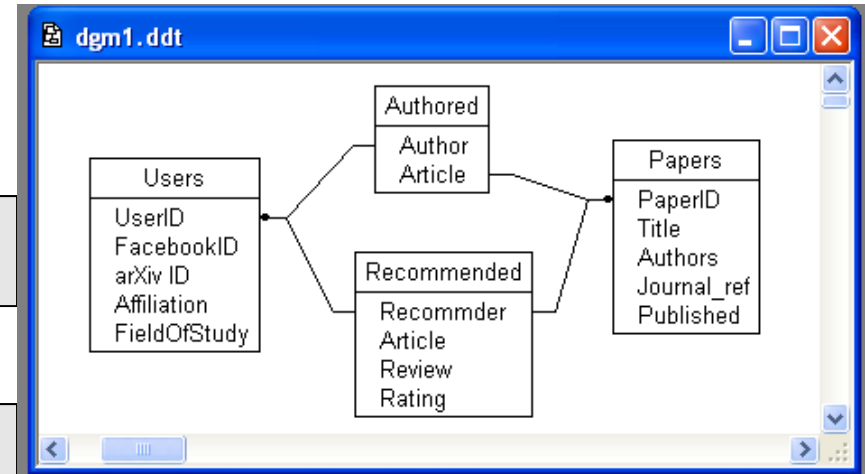
```
SELECT article FROM Authored  
WHERE Author = "UserID"
```

- 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

- SQL also allows the insertion of new rows into the database as well as updated existing rows.
- Insert a new Paper for an existing Author

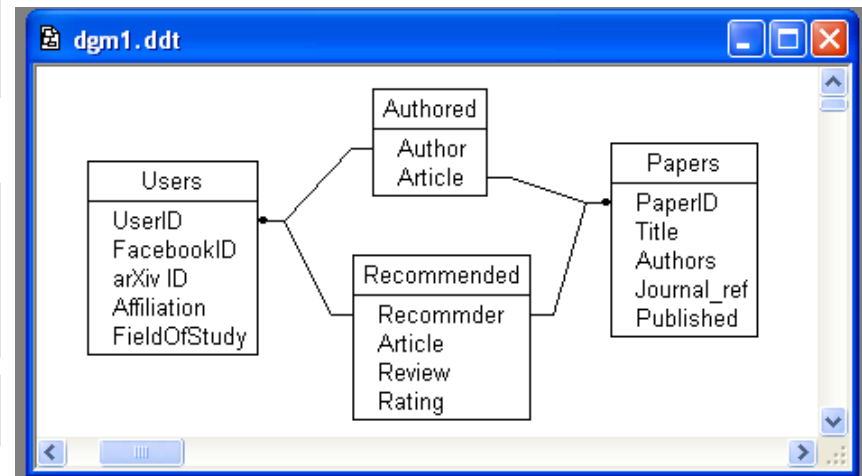
```
INSERT INTO Papers (PaperID, Title, Authors)  
Values(5678,'Really neat stuff','RG Fields')
```

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

- Update the journal\_ref

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

```
COMMIT
```





## SQL Language – Delete/Order

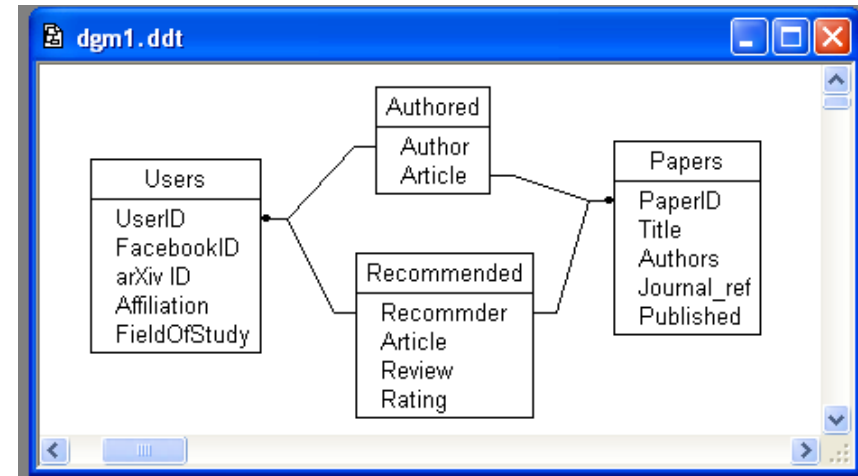
- Dropping rows from the table is just like SELECTing, where rows are selected using a WHERE clause.

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

- Many functions can be used to get records 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

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

```
import MySQLdb

#Generate a connection string to a specific database specifying
#the connection string parameters
conn = MySQLdb.connect(host = "localhost",
                       user = "testuser",
                       passwd="testpass",
                       db="test")

#Once the connection is established, we can retrieve
#a cursor object, which is a SQL cursor
cursor = conn.cursor()
#We can use the cursor to execute SQL statments
cursor.execute("SELECT * FROM Papers WHERE paperID = '1234'")
#And use fetchone() and fetchall() to retrieve the result of the query
#Row is an array with each field from Papers in an element
row = cursor.fetchone()
#cursors are reusable, so we can enter more SQL
cursor.execute("SELECT article from Authored WHERE Author='1234567'")
rows = 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

- Benefits:
  - SQL is a relatively simple language that has a very gentle learning curve
  - Connectors from every language to every type of database is widely available and all reasonable databases support SQL, making it a ubiquitous choice.
  - Lines of code can be drastically reduced by taking advantage of the databases code for searching and retrieving objects from the database
- Problems:
  - SQL will allow you to make amazingly inefficient queries that may not be obvious why they are inefficient, you may need to play with a query to optimize it.
  - Yet another language to learn



## OR Mapping

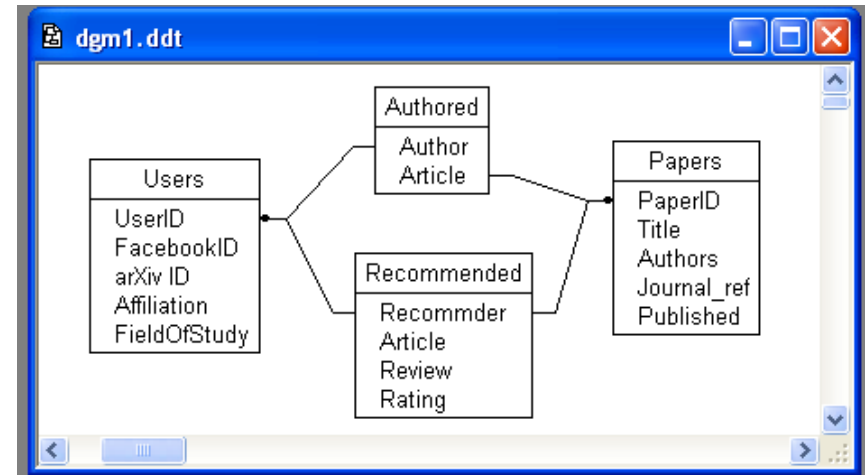
- OR Mapping – When you just don't have time for SQL
- Object-relational mapping (also ORM and O/R Mapping) converts data between a database and an object-oriented programming language.
- An ORM tool creates database tables from class definitions and so let's you create and use a database with standard object programming paradigms.
- The ORM tool basically writes the SQL for you, which can be highly beneficial in most cases, but the 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 REFERENCES Papers(PaperID)  
);  
;  
ALTER TABLE 'Authored' ADD CONSTRAINT Author_refs_id  
  FOREIGN KEY ('Author') REFERENCES 'Users' ('UserID');  
;  
COMMIT;
```



- A script is usually used to generate the database (so it can be regenerated as needed) and looks something like the thing on the right.



## OR Mapping

- 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 will be generated).

This means varchar(128)

```
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)
```

This field can be NULL



## 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
```



## Conclusions

- Databases are always the right solution.
  - Well...
- 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.