Data Modeling Demystified NEARC Spring 2013

Brian Hebert, Solutions Architect

www.scribekey.com

Abstract and Goal

Classic relational database and object-oriented ***modeling diagrams, tools, and techniques*** are often too ***complicated*** for GIS users and project managers to fully understand. This presentation will demonstrate a ***simplified approach to modeling*** geospatial feature classes and tables, using simple ***attribute lists*** generated through data profiling. Attendees will then be able to ***apply this useful technique on their own data projects*** for either improving existing data models, or creating new ones.

Goal: Provide you with a detailed but straightforward method you can use with your own projects and data.

Additionally help with better understanding of classic modeling tools and techniques.

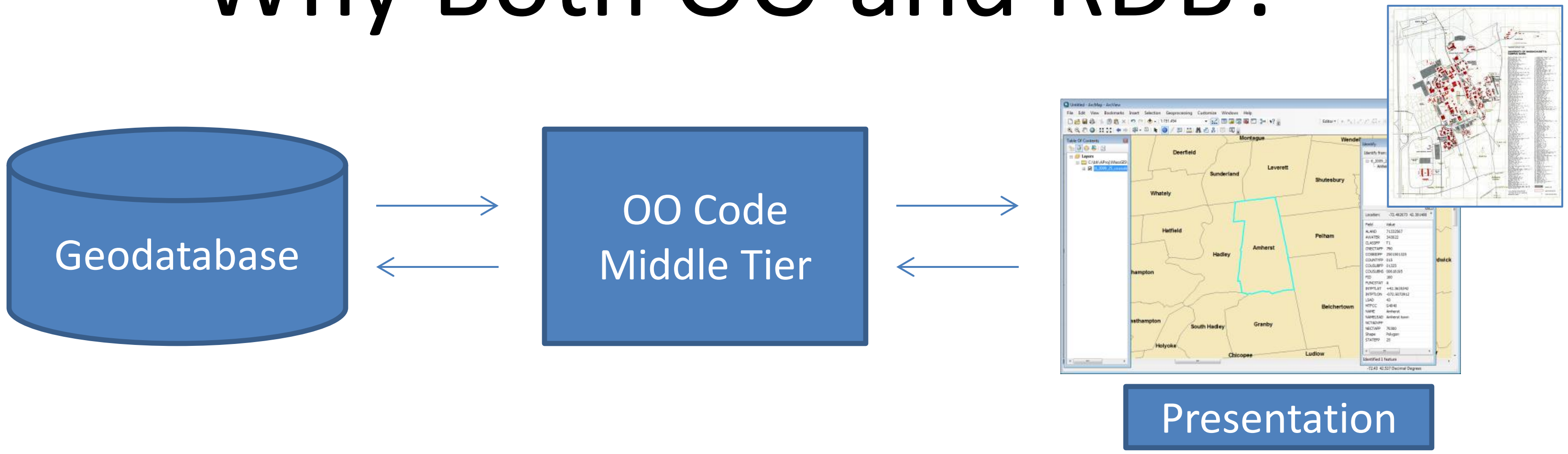
Outline

- Review object oriented and relational modeling
- Discuss RDB normalization
- Identify a communication/skills gap
- Describe refactoring
- Walk through an alternative 3 step table based modeling technique
- Review domains
- Recap
- Q & A

Data Models and Modeling

- Application: Object Oriented (OO) Class Diagrams – Generate C#, Java, C++
- Relational Database: Entity-Relationship (ER) Diagrams – Generate SQL
- Analysis, N-Dimensional Cube: Star, Snowflake Diagrams (generally unfamiliar to GIS practitioners) – Create Cube

Why Both OO and RDB?

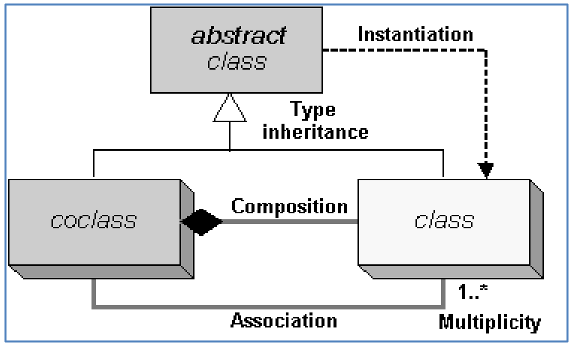


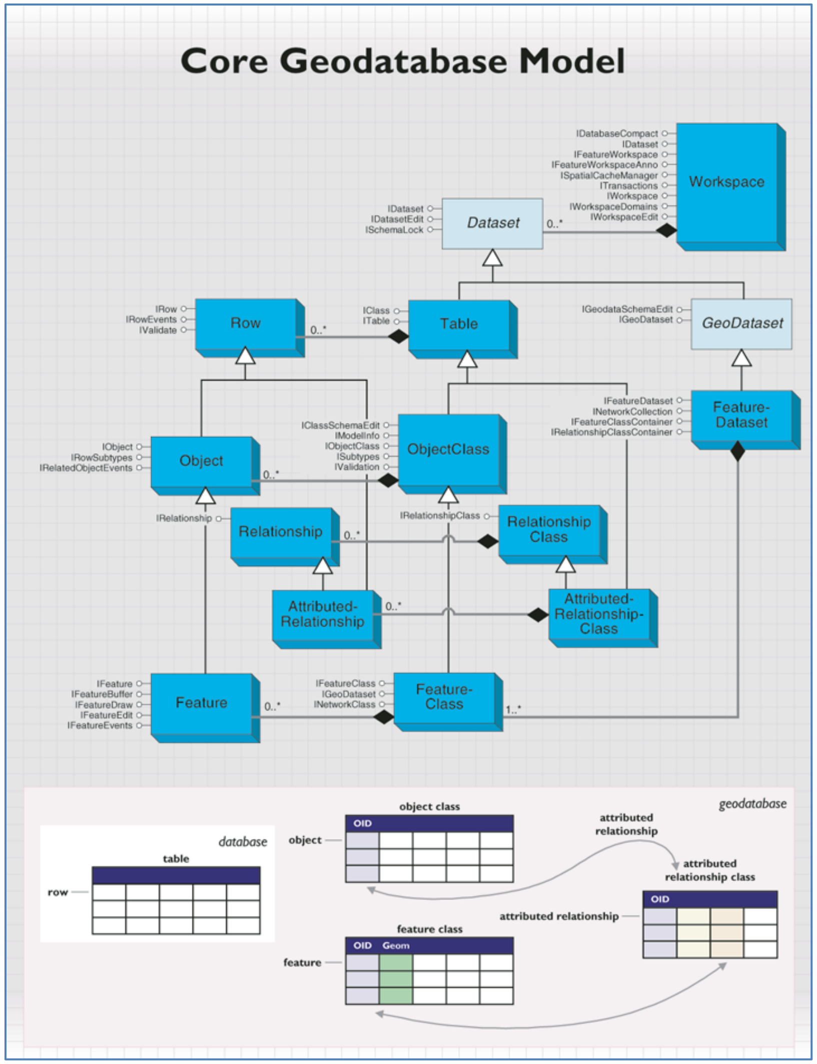
ArcGIS is an Object-Relational System (ORDB). Data and application metadata are stored in the backend geodatabase, loaded into middle tier, written with OO C++ code, and presented through rich GUI presentation applications, ArcMap, ArcCatalog, ArcToolBox, etc. As such, we use both OO and RDB modeling techniques.

OO Databases never really made it. This class 3 tier approach is how OO data models are implemented.

The current trend is to use tools like UML to design OO systems.

UML OO Diagraming

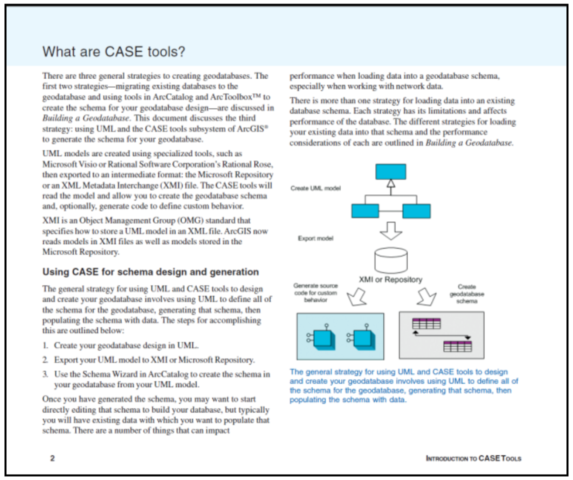




UML (Unified Modeling Language) diagrams capture type hierarchies. They show **type family trees**, focusing on how more specific sub-classes inherit properties and behavior from more general super-classes.

Becoming familiar with UML and related tools requires significant time and experience.

Designing in ArcGIS: CASE Tools or ArcCatalog



What are CASE tools?

There are three general strategies to creating geodatabases. The first two strategies—migrating existing databases to the geodatabase and using tools in ArcCatalog and ArcToolbox™ to create the schema for your geodatabase design—are discussed in *Building a Geodatabase*. This document discusses the third strategy: using UML and the CASE tools subsystem of ArcGIS® to generate the schema for your geodatabase.

UML models are created using specialized tools, such as Microsoft Visio or Rational Software Corporation's Rational Rose, then exported to an intermediate format: the Microsoft Repository or an XML Metadata Interchange (XMI) file. The CASE tools will read the model and allow you to create the geodatabase schema and, optionally, generate code to define custom behavior.

XMI is an Object Management Group (OMG) standard that specifies how to store a UML model in an XML file. ArcGIS now reads models in XMI files as well as models stored in the Microsoft Repository.

Using CASE for schema design and generation

The general strategy for using UML and CASE tools to design and create your geodatabase involves using UML to define all of the schema for the geodatabase, generating that schema, then populating the schema with data. The steps for accomplishing this are outlined below:

1. Create your geodatabase design in UML.
2. Export your UML model to XMI or Microsoft Repository.
3. Use the Schema Wizard in ArcCatalog to create the schema in your geodatabase from your UML model.

Once you have generated the schema, you may want to start directly editing that schema to build your database, but typically you will have existing data with which you want to populate that schema. There are a number of things that can impact performance when loading data into a geodatabase schema, especially when working with network data.

There is more than one strategy for loading data into an existing database schema. Each strategy has its limitations and affects performance of the database. The different strategies for loading your existing data into that schema and the performance considerations of each are outlined in *Building a Geodatabase*.

Create UML model
Export model
XMI or Repository
Generate source code for custom behavior
Create geodatabase schema

The general strategy for using UML and CASE tools to design and create your geodatabase involves using UML to define all of the schema for the geodatabase, generating that schema, then populating the schema with data.

2 INTRODUCTION TO CASE TOOLS

ESRI CASE Tools

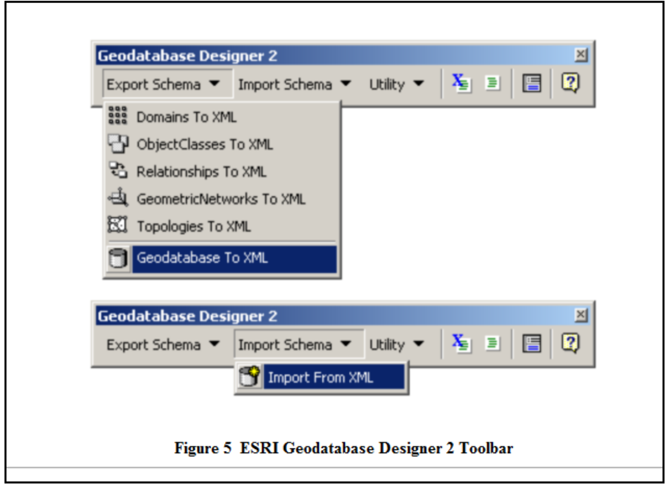
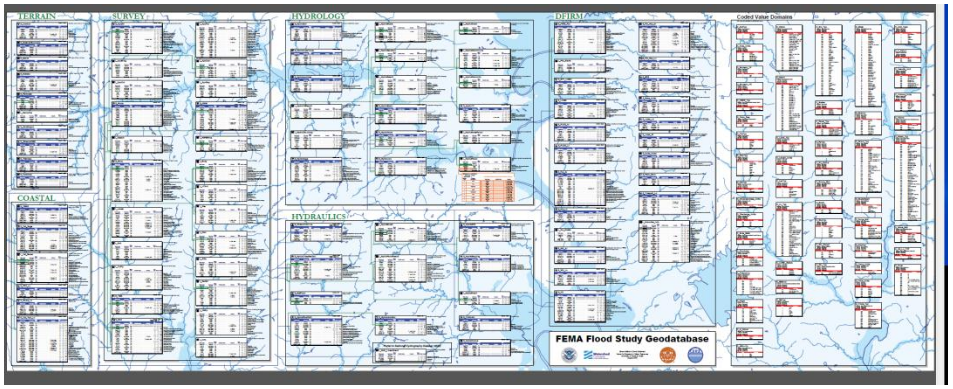


Figure 5 ESRI Geodatabase Designer 2 Toolbar

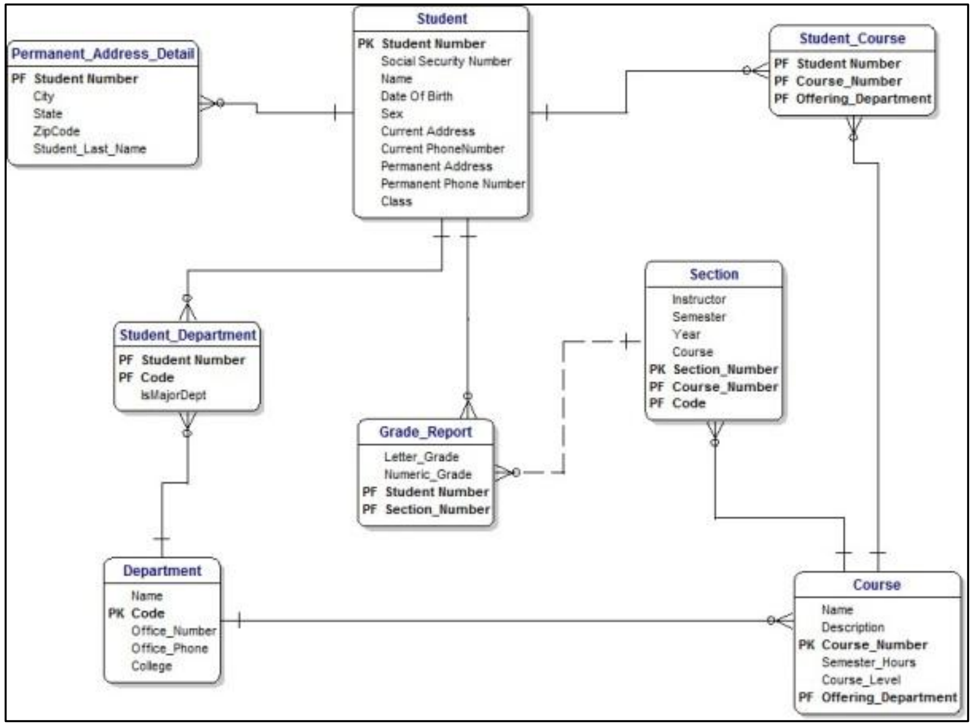
ArcCatalog Tools



Geodatabase Diagram

With CASE or ArcCatalog Tools, the goal is the same, to generate a Geodatabase

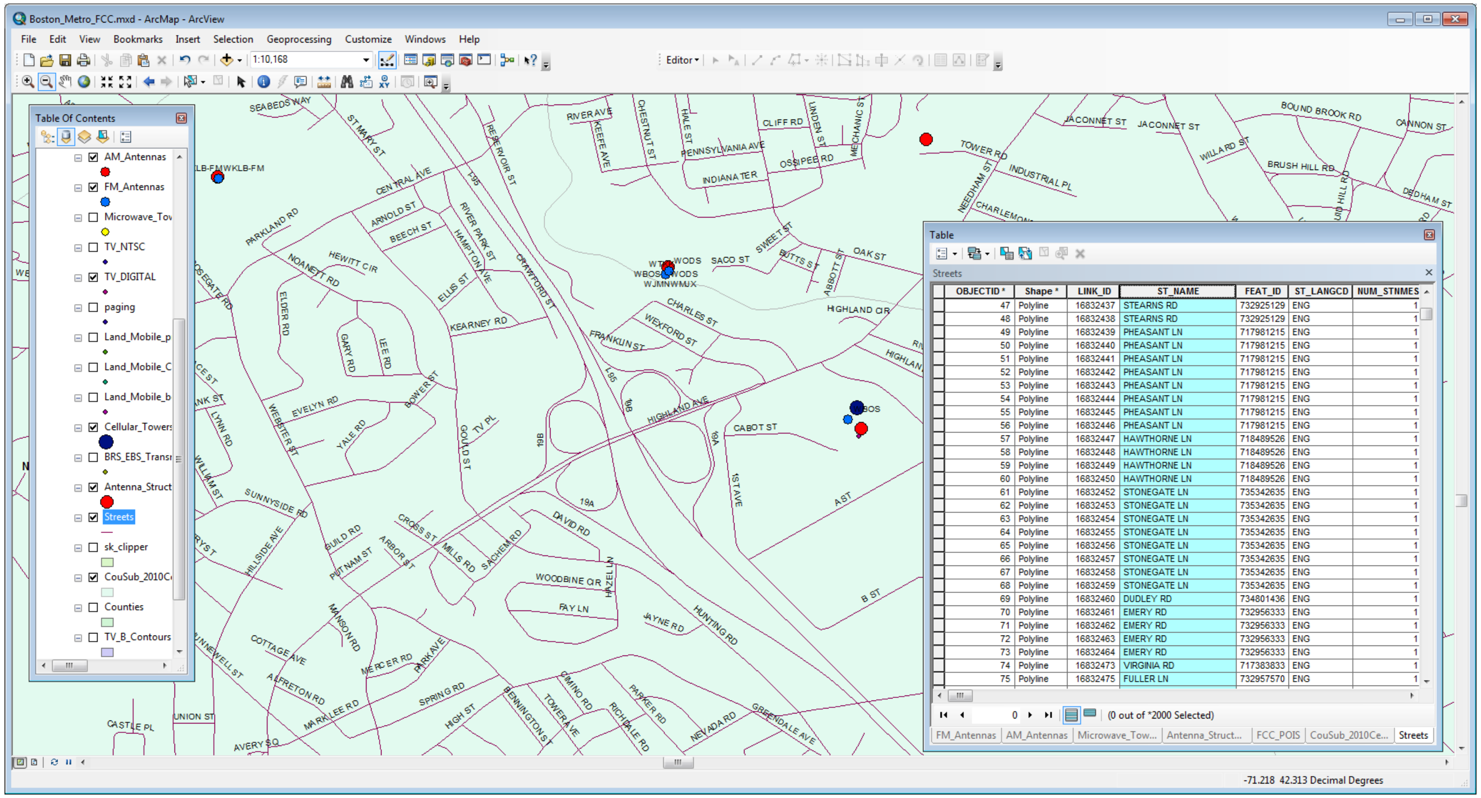
RDB Entity-Relationship Diagrams and Tools



Entity Relationship diagrams and related tools are typically used to create highly normalized, editing/production databases. Normalization protects referential integrity and prevents anomalies that can compromise data quality, break relationships, etc.

The models we typically use in GIS are often highly de-normalized. De-normalized databases are much easier to query and use for maps and reports.

De-Normalized Data Example: Navteq



OBJECTID *	Shape *	LINK_ID	ST_NAME	FEAT_ID	ST_LANGCD	NUM_STNMES
47	Polyline	16832437	STEARNS RD	732925129	ENG	1
48	Polyline	16832438	STEARNS RD	732925129	ENG	1
49	Polyline	16832439	PHEASANT LN	717981215	ENG	1
50	Polyline	16832440	PHEASANT LN	717981215	ENG	1
51	Polyline	16832441	PHEASANT LN	717981215	ENG	1
52	Polyline	16832442	PHEASANT LN	717981215	ENG	1
53	Polyline	16832443	PHEASANT LN	717981215	ENG	1
54	Polyline	16832444	PHEASANT LN	717981215	ENG	1
55	Polyline	16832445	PHEASANT LN	717981215	ENG	1
56	Polyline	16832446	PHEASANT LN	717981215	ENG	1
57	Polyline	16832447	HAWTHORNE LN	718489526	ENG	1
58	Polyline	16832448	HAWTHORNE LN	718489526	ENG	1
59	Polyline	16832449	HAWTHORNE LN	718489526	ENG	1
60	Polyline	16832450	HAWTHORNE LN	718489526	ENG	1
61	Polyline	16832452	STONEGATE LN	735342635	ENG	1
62	Polyline	16832453	STONEGATE LN	735342635	ENG	1
63	Polyline	16832454	STONEGATE LN	735342635	ENG	1
64	Polyline	16832455	STONEGATE LN	735342635	ENG	1
65	Polyline	16832456	STONEGATE LN	735342635	ENG	1
66	Polyline	16832457	STONEGATE LN	735342635	ENG	1
67	Polyline	16832458	STONEGATE LN	735342635	ENG	1
68	Polyline	16832459	STONEGATE LN	735342635	ENG	1
69	Polyline	16832460	DUDLEY RD	734801436	ENG	1
70	Polyline	16832461	EMERY RD	732956333	ENG	1
71	Polyline	16832462	EMERY RD	732956333	ENG	1
72	Polyline	16832463	EMERY RD	732956333	ENG	1
73	Polyline	16832464	EMERY RD	732956333	ENG	1
74	Polyline	16832473	VIRGINIA RD	717383833	ENG	1
75	Polyline	16832475	FULLER LN	732957570	ENG	1

(0 out of *2000 Selected)

-71.218 42.313 Decimal Degrees

If Navteq Streets data were normalized there would be multiple separate tables for street names, town names, states, zip codes, street types, etc. If this were the case, creating Geocoding or Routing applications would be MUCH more complex

SQL Query: De-Normalized and Normalized

```
SELECT ID, NAME FROM STREETS WHERE NAME='Elm St'
```

```
SELECT STREETS.ID, STREET_NAMES.NAME
FROM STREETS INNER JOIN STREET_NAMES ON
STREETS.STREET_NAME_ID = STREET_NAMES.ID AND
STREET_NAMES.NAME='Elm St'
```

Queries for normalized data are much more complex. Additionally, if data is being used in a read-only mode, there is typically no reason for highly normalized models.

WARNING! In spite of this, many data RDB data modelers start a project with the assumption that the data should be highly normalized.

Layers
Attributes
Symbols
…

?

UML, XSD
GML
ISO 19XXX,
…

GIS Users

The Tower of Babel

Data Modelers
ISO Standards Bodies
OGC
(Druids)

What do
they mean?

E=MC2,
Abracadabra
…

An Alternate Technique

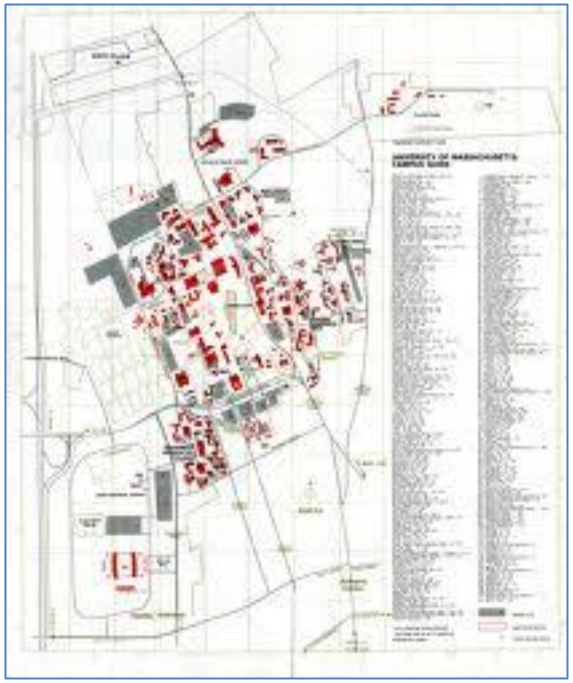
1) Use database tables to capture/store information about feature classes, tables, attributes, and domain values. This is metadata.
2) Arrange lists of attributes and entities in cross reference matrices.
3) Use SQL queries to discover your classes, and help you create your entities.

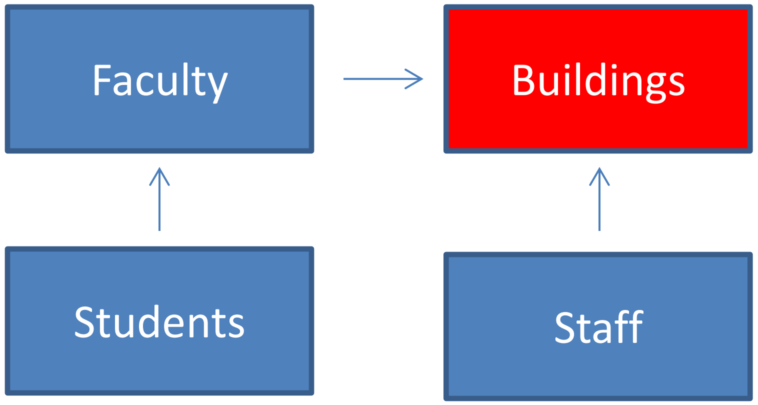
GIS users are typically very familiar with data in tables. There is no immediate need to learn an unfamiliar or complex diagraming technique to do what you need to do.

This helps involves the people that know the most about the data in the modeling process.

Scenario

- A university has asked us to integrate existing tabular data describing Students, Faculty, and Staff, with GIS building layer.
- They have asked for a class diagram.
- They also want to create tables in ArcGIS
- They want to determine who is where, when, etc.
- The focus is not GIS per se, but will help us concentrate on important concepts.
- Reduced attribute sets will be used.





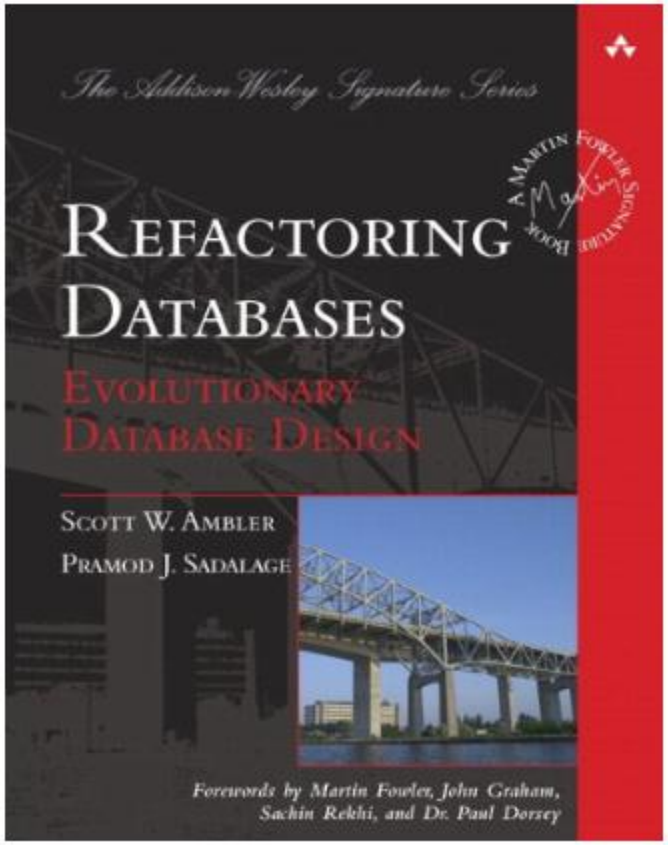
Attribute Lists

STUDENTS		FACULTY		STAFF
Student_Name		PROF_Name		Name
ST_DOB		DOB		Birthday
Major		Department		Title
GPA		Salary		Supervisor
Student_Year		Tenure		Salary
Dormatory		Highest Degree		Office
Advisor		Office		

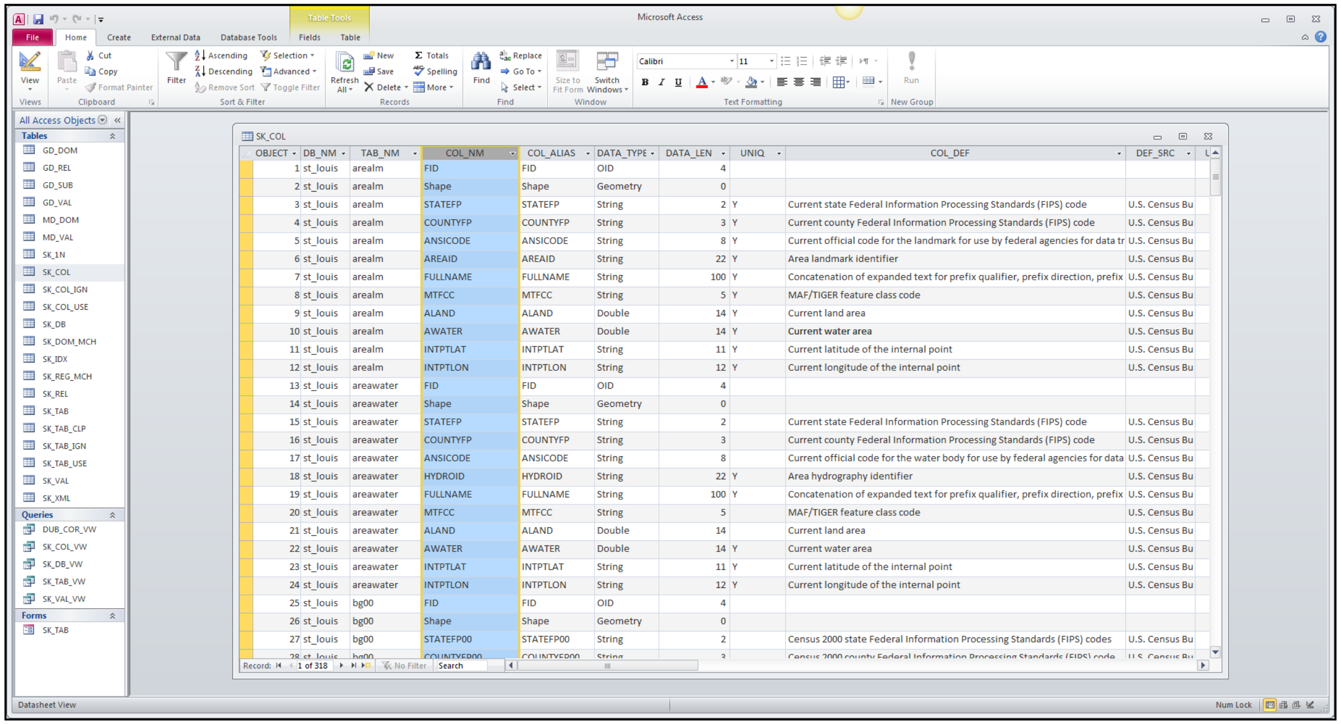
From an initial view of the 3 entity attributes we see that they share some common general attributes as well as having some specialized attributes of their own.

Sidebar: Refactoring

- A great deal of complex OO and RDB data modeling is carried out in data-free environments, assuming a blank slate to start with.
- In reality, this is rarely the case, and we typically have to fit existing data into our models.
- As such, it is typically helpful to use a data-driven approach to modeling, where the focus is on changing existing entities, rather than creating new ones.
- This is related to a technique called REFACTORING.


The Addison Wesley Signature Series
A Martin Fowler Signature Book
Refactoring Databases
Evolutionary Database Design
Scott W. Ambler
Pramod J. Sadalage
Forewords by Martin Fowler, John Graham, Sachin Rekhi, and Dr. Paul Dorsey

Creating Entity and Attribute Metadata Tables



SK_COL

OBJECT	DB_NM	TAB_NM	COL_NM	COL_ALIAS	DATA_TYPE	DATA_LEN	UNIQ	COL_DEF	DEF_SRC
1	st_louis	arealm	FID	FID	OID	4			
2	st_louis	arealm	Shape	Shape	Geometry	0			
3	st_louis	arealm	STATEFP	STATEFP	String	2	Y	Current state Federal Information Processing Standards (FIPS) code	U.S. Census Bu
4	st_louis	arealm	COUNTYFP	COUNTYFP	String	3	Y	Current county Federal Information Processing Standards (FIPS) code	U.S. Census Bu
5	st_louis	arealm	ANSICODE	ANSICODE	String	8	Y	Current official code for the landmark for use by federal agencies for data tr	U.S. Census Bu
6	st_louis	arealm	AREAID	AREAID	String	22	Y	Area landmark identifier	U.S. Census Bu
7	st_louis	arealm	FULLNAME	FULLNAME	String	100	Y	Concatenation of expanded text for prefix qualifier, prefix direction, prefix	U.S. Census Bu
8	st_louis	arealm	MTFCC	MTFCC	String	5	Y	MAF/TIGER feature class code	U.S. Census Bu
9	st_louis	arealm	ALAND	ALAND	Double	14	Y	Current land area	U.S. Census Bu
10	st_louis	arealm	AWATER	AWATER	Double	14	Y	Current water area	U.S. Census Bu
11	st_louis	arealm	INTPTLAT	INTPTLAT	String	11	Y	Current latitude of the internal point	U.S. Census Bu
12	st_louis	arealm	INTPTLON	INTPTLON	String	12	Y	Current longitude of the internal point	U.S. Census Bu
13	st_louis	areawater	FID	FID	OID	4			
14	st_louis	areawater	Shape	Shape	Geometry	0			
15	st_louis	areawater	STATEFP	STATEFP	String	2		Current state Federal Information Processing Standards (FIPS) code	U.S. Census Bu
16	st_louis	areawater	COUNTYFP	COUNTYFP	String	3		Current county Federal Information Processing Standards (FIPS) code	U.S. Census Bu
17	st_louis	areawater	ANSICODE	ANSICODE	String	8		Current official code for the water body for use by federal agencies for data	U.S. Census Bu
18	st_louis	areawater	HYDROID	HYDROID	String	22	Y	Area hydrography identifier	U.S. Census Bu
19	st_louis	areawater	FULLNAME	FULLNAME	String	100	Y	Concatenation of expanded text for prefix qualifier, prefix direction, prefix	U.S. Census Bu
20	st_louis	areawater	MTFCC	MTFCC	String	5		MAF/TIGER feature class code	U.S. Census Bu
21	st_louis	areawater	ALAND	ALAND	Double	14		Current land area	U.S. Census Bu
22	st_louis	areawater	AWATER	AWATER	Double	14	Y	Current water area	U.S. Census Bu
23	st_louis	areawater	INTPTLAT	INTPTLAT	String	11	Y	Current latitude of the internal point	U.S. Census Bu
24	st_louis	areawater	INTPTLON	INTPTLON	String	12	Y	Current longitude of the internal point	U.S. Census Bu
25	st_louis	bg00	FID	FID	OID	4			
26	st_louis	bg00	Shape	Shape	Geometry	0			
27	st_louis	bg00	STATEFP00	STATEFP00	String	2		Census 2000 state Federal Information Processing Standards (FIPS) codes	U.S. Census Bu
28	st_louis	bg00	COUNTYFP00	COUNTYFP00	String	3		Census 2000 county Federal Information Processing Standards (FIPS) code	U.S. Census Bu

Record: 1 of 318

Step 1 is to capture Attribute Lists for each of the Entities in your model. If the model is based on existing data, this can be done with a data profiling tool. Otherwise, it can be done manually.

Here Attribute information including Entity, Data Type, Length, Uniqueness, and FGDC XML Metadata definitions have been captured with a data profiler in MS Access.

Standardize Raw Data Attribute Names

- Mapping original raw attribute names to standardized sets is a key part of refactoring.
- This helps determine the unique list of attributes found across all entities, and is required for this technique to work.

Entity	Attribute_Raw	Attribute_Standard
Student	Student_Name	NAME
Student	ST_DOB	DOB
Student	Major	DEPARTMENT
Student	GPA	GPA
Student	Student_Year	STUDENT_YEAR
Student	Dormatory	BUILDING
Student	Advisor	FACULTY
Faculty	PROF_Name	NAME
Faculty	DOB	DOB
Faculty	Department	DEPARTMENT
Faculty	Salary	SALARY
Faculty	Tenure	TENURE
Faculty	Highest Degree	HIGHEST_DEGREE
Faculty	Office	BUILDING
Staff	Name	NAME
Staff	Birthday	DOB
Staff	Title	TITLE
Staff	Supervisor	SUPERVISOR
Staff	Salary	SALARY
Staff	Office	BUILDING

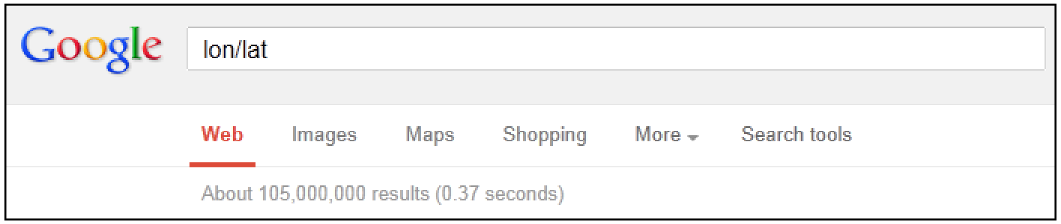
Step 1: Standardized Entity Attribute List

What's in a name?

How we name entities, attribute, and domain values is very important for easy-to-use, self describing data models. One important rule of thumb: **An attribute meaning the same thing should have the same name, data type, length, and domain, in all the entities it appears in.**

In general, it is best to pick very commonly used and understood words vs. trying to be consistent.

When choosing names and terms it can be helpful to see which ones are the most commonly used by getting word counts with Google. For example, Google returns 105 million hits when searching for 'lon/lat' and 23.8 million when searching for 'lat/lon' indicating 'lon/lat' is a far more common term.



About 105,000,000 results (0.37 seconds)

ENTITY	ATTRIBUTE
Student	Name
Student	DOB
Student	Major
Student	GPA
Student	Student_Year
Student	Building
Student	Advisor
Faculty	Name
Faculty	DOB
Faculty	Department
Faculty	Salary
Faculty	Tenure
Faculty	Highest_Degree
Faculty	Building
Staff	Name
Staff	DOB
Staff	Title
Staff	Supervisor
Staff	Salary
Staff	Building

Step 2: Create an Attribute Entity Matrix

ID	ATTRIBUTE	STUDENTS	FACULTY	STAFF	GROUP
1	Advisor	1	0	0	100
2	**Department**	1	1	0	110
3	DOB	1	1	1	111
4	GPA	1	0	0	100
5	Highest Degree	0	1	0	010
6	Name	1	1	1	111
7	**Building**	1	1	1	111
8	Salary	0	1	1	011
9	Student_Year	1	0	0	100
10	Supervisor	0	0	1	001
11	Tenure	0	1	0	010
12	Title	0	0	1	001

Pivot the initial list of Entities and Attribute, create a unique list of attributes and pivot the Entities to create a matrix. **Note some attributes which actually use the same domain might have different names**, so choose a single name for these, e.g., Major and Department become Department, Dormitory and Building use Building This can be done with SQL and/or other tools.

Use TEXT data types for entity flags and PATTERN

UPDATE ATT_ENT_MATRIX SET GROUP = STUDENTS & FACULTY & STAFF

Step 3: Get Distinct Attribute Groups

ATTRIBUTE	GROUP
Supervisor	001
Title	001
Highest Degree	010
Tenure	010
Building	011
Salary	011
Advisor	100
GPA	100
Student_Year	100
Department	110
DOB	111
Name	111

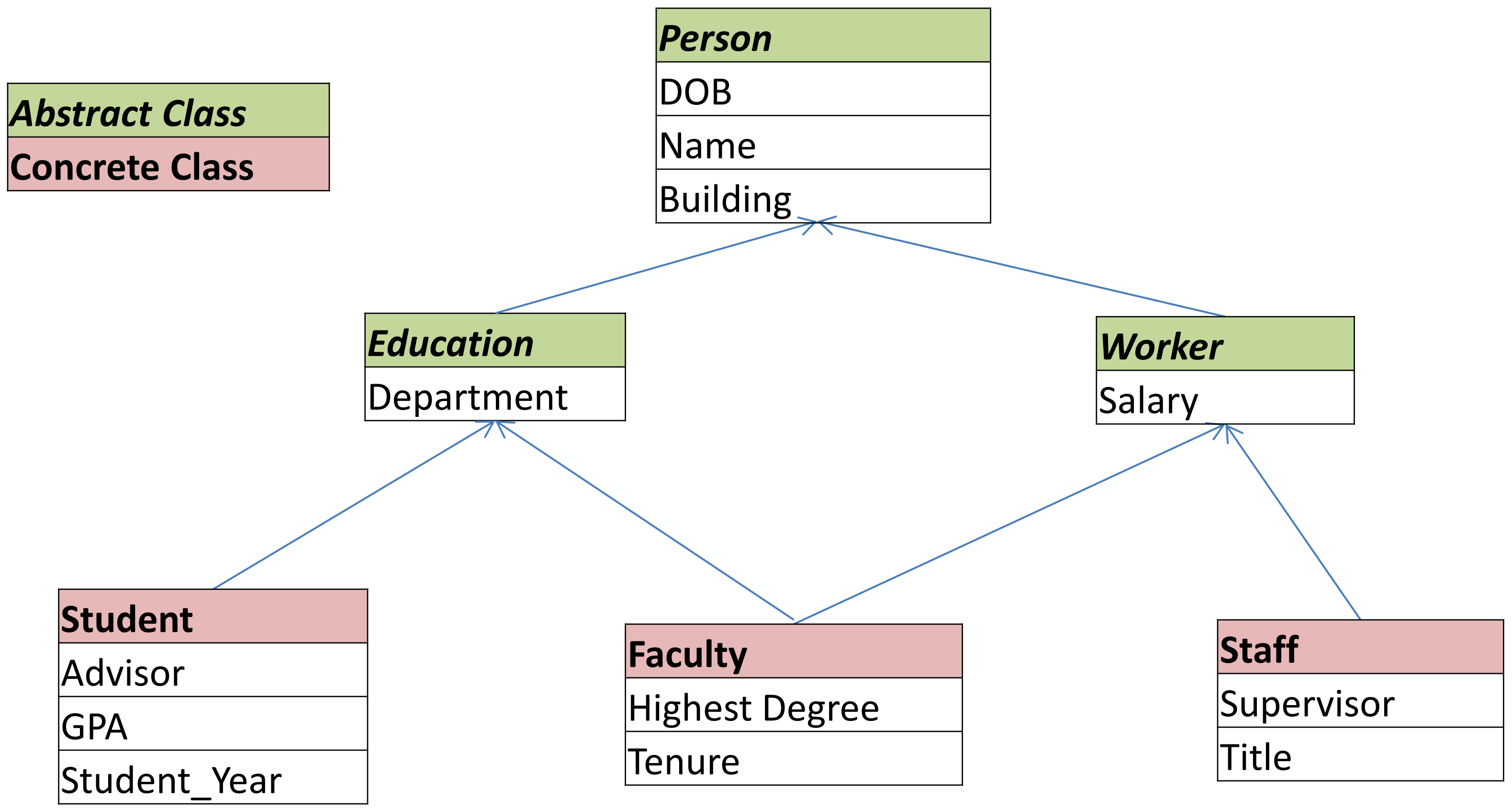
$\rightarrow$

ID	GROUP	GROUP_MEMBERS	CLASS_NAME
1	001	Staff_Info	**Staff_Info**
2	010	Faculty_Info	**Faculty_Info**
3	011	Faculty_Staff_Info	**Worker**
4	100	Student_Info	**Student_Info**
5	110	Student_Faculty_Info	**Education**
6	111	Student_Faculty_Staff_Info	**Person**

Sort Attributes by Group. A supplemental, and sometimes challenging, step is to name the Groups.

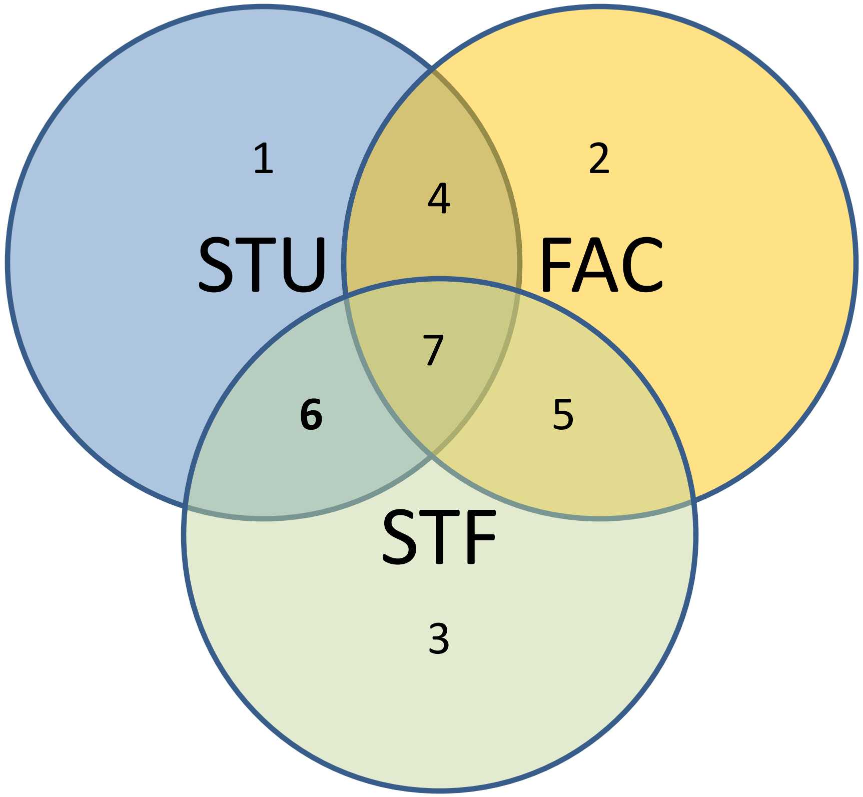
Here we have 6 distinct sets of Attributes. Each Entity uses a set of groups. These equate to classes in OO design. Each feature class we create will contain all of the Attribute Groups it uses.

Class Diagram from Attribute Groups



We see the same information we created using Attribute Groups here as a classic OO type hierarchy. Note that Faculty inherits from both Education and Worker. Instead of using inheritance trees it is sometimes easier to use the concept of interfaces. Note with C# and Java, multiple inheritance is not allowed, as it is with C++. Note also that the classes are divided into 2 types, abstract which will never actually be created, and concrete, which inherit from the abstract classes, but only show specific attributes. Note this diagram does not show relationships between entities.

Attribute Sets and Interfaces



Another way to look at the Attribute Groups is as overlapping sets. Note out of a possible 7 groups, we ended up with a total of 6. This is because there were no attributes used only by Students and Staff, shown in the area labeled 6 above. These overlapping areas are equivalent to the class interfaces.

The Resulting Tables

Student_Year
DOB
Name
Building
Department
Adivsor
GPA
Student_Year

Faculty
DOB
Name
Building
Department
Salary
Highest_Degree
Tenure

Staff
DOB
Name
Building
Salary
Supervisor
Title

Simply listing the attributes for each entity provides one of the easiest to understand and useful views.
The tables holding the entity and attribute information in the metadata database can be used by relatively simple application to generate the tables.
So, why is it so important to know what attribute groups different entities share?

SQL Union Views

If you know what attribute groups different entities have in common, you can construct views in relational databases, using the UNION clause, to see different things as the same type of things.

SELECT NAME, 'STUDENT' AS TYPE, BUILDING FROM STUDENTS

UNION

SELECT NAME, 'FACULTY' AS TYPE, BUILDING FROM FACULTY

UNION

SELECT NAME, 'STAFF' AS TYPE, BUILDING FROM STAFF

You could then add the UNION view to ArcMap and relate it to the buildings polygon layer, to help see all the people in a single building.

You could do this with any of the attribute sets (classes) the entities have in common, e.g., Worker.

NO.	NAME	STUDENT	BUILDING
1			
2			
3			
NO.	**NAME**	**FACULTY**	**BUILDING**
1			
2			
3			
NO.	**NAME**	**STAFF**	**BUILDING**
1			
2			
3			

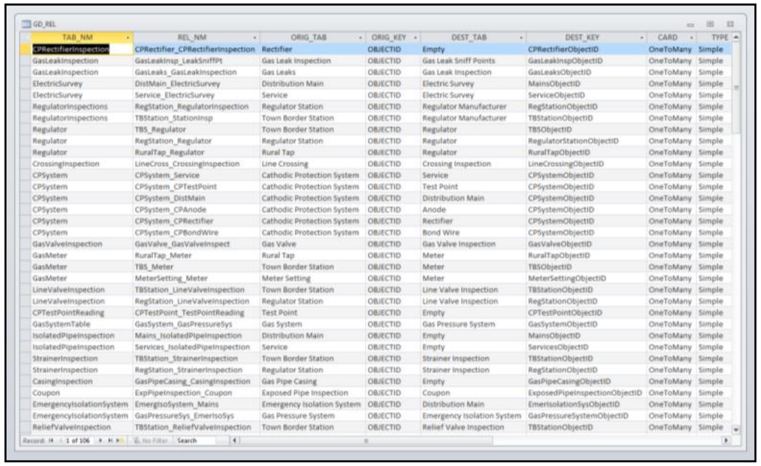
Union of 3 **Different** Tables

NO.	NAME	TYPE	BUILDING
1			
2			
3			
4			
5			
6			
7			
8			
9			

1 Resulting View with **Same** Data

Capturing Relationships

- Describing relationships between entities is an important part of data modeling
- Again, it can be useful to capture this information in metadata tables and matrices.



GD_REL

TAB_NM	REL_NM	ORIG_TAB	ORIG_KEY	DEST_TAB	DEST_KEY	CARD	TYPE
CPRectifierInspection	CPRectifier_CPRectifierInspection	Rectifier	OBJECTID	Empty	CPRectifierObjectID	OneToMany	Simple
GasLeakInspection	GasLeakInsp_LeakSniffPt	Gas Leak Inspection	OBJECTID	Gas Leak Sniff Points	GasLeakInspObjectID	OneToMany	Simple
GasLeakInspection	GasLeaks_GasLeakInspection	Gas Leaks	OBJECTID	Gas Leak Inspection	GasLeaksObjectID	OneToMany	Simple
ElectricSurvey	DistMain_ElectricSurvey	Distribution Main	OBJECTID	Electric Survey	MainsObjectID	OneToMany	Simple
ElectricSurvey	Service_ElectricSurvey	Service	OBJECTID	Electric Survey	ServiceObjectID	OneToMany	Simple
RegulatorInspections	RegStation_RegulatorInspection	Regulator Station	OBJECTID	Regulator Manufacturer	RegStationObjectID	OneToMany	Simple
RegulatorInspections	TBStation_StationInsp	Town Border Station	OBJECTID	Regulator Manufacturer	TBStationObjectID	OneToMany	Simple
Regulator	TBS_Regulator	Town Border Station	OBJECTID	Regulator	TBSObjectID	OneToMany	Simple
Regulator	RegStation_Regulator	Regulator Station	OBJECTID	Regulator	RegulatorStationObjectID	OneToMany	Simple
Regulator	RuralTap_Regulator	Rural Tap	OBJECTID	Regulator	RuralTapObjectID	OneToMany	Simple
CrossingInspection	LineCross_CrossingInspection	Line Crossing	OBJECTID	Crossing Inspection	LineCrossingObjectID	OneToMany	Simple
CPSystem	CPSystem_Service	Cathodic Protection System	OBJECTID	Service	CPSystemObjectID	OneToMany	Simple
CPSystem	CPSystem_CPTestPoint	Cathodic Protection System	OBJECTID	Test Point	CPSystemObjectID	OneToMany	Simple
CPSystem	CPSystem_DistMain	Cathodic Protection System	OBJECTID	Distribution Main	CPSystemObjectID	OneToMany	Simple
CPSystem	CPSystem_CPAnode	Cathodic Protection System	OBJECTID	Anode	CPSystemObjectID	OneToMany	Simple
CPSystem	CPSystem_CPRectifier	Cathodic Protection System	OBJECTID	Rectifier	CPSystemObjectID	OneToMany	Simple
CPSystem	CPSystem_CPBondWire	Cathodic Protection System	OBJECTID	Bond Wire	CPSystemObjectID	OneToMany	Simple
GasValveInspection	GasValve_GasValveInspect	Gas Valve	OBJECTID	Gas Valve Inspection	GasValveObjectID	OneToMany	Simple
GasMeter	RuralTap_Meter	Rural Tap	OBJECTID	Meter	RuralTapObjectID	OneToMany	Simple
GasMeter	TBS_Meter	Town Border Station	OBJECTID	Meter	TBSObjectID	OneToMany	Simple
GasMeter	MeterSetting_Meter	Meter Setting	OBJECTID	Meter	MeterSettingObjectID	OneToMany	Simple
LineValveInspection	TBStation_LineValveInspection	Town Border Station	OBJECTID	Line Valve Inspection	TBStationObjectID	OneToMany	Simple
LineValveInspection	RegStation_LineValveInspection	Regulator Station	OBJECTID	Line Valve Inspection	RegStationObjectID	OneToMany	Simple
CPTestPointReading	CPTestPoint_TestPointReading	Test Point	OBJECTID	Empty	CPTestPointObjectID	OneToMany	Simple
GasSystemTable	GasSystem_GasPressureSys	Gas System	OBJECTID	Gas Pressure System	GasSystemObjectID	OneToMany	Simple
IsolatedPipeInspection	Mains_IsolatedPipeInspection	Distribution Main	OBJECTID	Empty	MainsObjectID	OneToMany	Simple
IsolatedPipeInspection	Services_IsolatedPipeInspection	Service	OBJECTID	Empty	ServicesObjectID	OneToMany	Simple
StrainerInspection	TBStation_StrainerInspection	Town Border Station	OBJECTID	Strainer Inspection	TBStationObjectID	OneToMany	Simple
StrainerInspection	RegStation_StrainerInspection	Regulator Station	OBJECTID	Strainer Inspection	RegStationObjectID	OneToMany	Simple
CasingInspection	GasPipeCasing_CasingInspection	Gas Pipe Casing	OBJECTID	Empty	GasPipeCasingObjectID	OneToMany	Simple
Coupon	ExpPipeInspection_Coupon	Exposed Pipe Inspection	OBJECTID	Coupon	ExposedPipeInspectionObjectID	OneToMany	Simple
EmergencyIsolationSystem	EmergIsoSystem_Mains	Emergency Isolation System	OBJECTID	Distribution Main	EmerIsolationSysObjectID	OneToMany	Simple
EmergencyIsolationSystem	GasPressureSys_EmerIsoSys	Gas Pressure System	OBJECTID	Emergency Isolation System	GasPressureSystemObjectID	OneToMany	Simple
ReliefValveInspection	TBStation_ReliefValveInspection	Town Border Station	OBJECTID	Relief Valve Inspection	TBStationObjectID	OneToMany	Simple

Record: 1 of 106 No Filter Search

Geodatabase Relationships captured by data profiling tool, in MS Access, showing ORIGIN, DESTINATION, and CARDINALITY

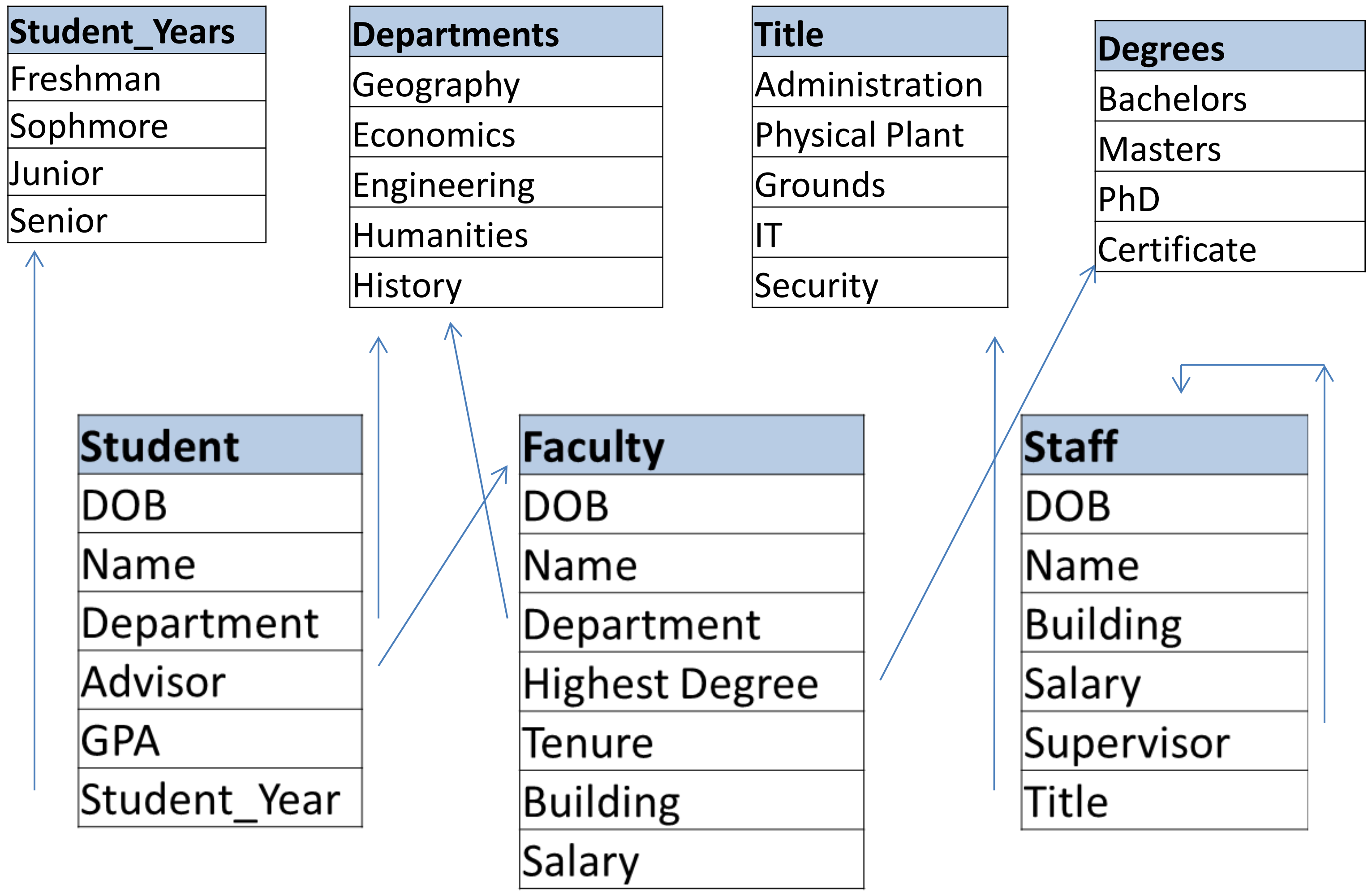
	Student	Faculty	Staff	Building
Student		N->1	NA	N->1
Faculty			NA	NA
Staff				N->1
Building				

Easy to construct entity relations matrix

A Word about Domains

- Explicitly specifying the set of values that can be used for an attribute is an important part of data modeling.
- Even if you are not building a normalized data model, it is still useful to capture domains; this information can be used for both data validation and application GUI construction.
- **This information can also be stored in the metadata repository database as well.**
- Here are 3 methods for this:
  1) Lookup Tables: Valid list of values, e.g., Departments
  2) Regular Expressions: allowed patterns, e.g., MM/DD/YYYY
  3) SQL Expressions: Flexible assertion, e.g., range, match to another attribute, etc.

Domains as Lookup Tables



Student_Years
Freshman
Sophmore
Junior
Senior

Departments
Geography
Economics
Engineering
Humanities
History

Title
Administration
Physical Plant
Grounds
IT
Security

Degrees
Bachelors
Masters
PhD
Certificate

Student
DOB
Name
Department
Advisor
GPA
Student_Year

Faculty
DOB
Name
Department
Highest Degree
Tenure
Building
Salary

Staff
DOB
Name
Building
Salary
Supervisor
Title

Even if you are not doing full normalization, it's important to capture domain values in tables. These can then be used to validate data and help build GUI front end pick lists. Note an empty database, with no features or rows, still has domain tables.

Recap

- Consider a simple, table-centric, systematic and mechanical approach to data modeling.
- You can use a table/matrix approach to modeling your entities (feature classes and tables) as collections of attribute sets, with some attribute sets common to multiple entities. You can capture relationships this way too.
- You can easily capture this information in a metadata database, then use querying and sorting to help you with your task.
- The resulting attribute sets will match a class inheritance and/or an class interface diagram.
- When you know what attribute sets are common to multiple entities, you can use UNION views to query on the common elements.
- Don’t automatically start creating a normalized set of entities if your database is to support querying, reports, and map making.
- Even if you aren’t normalizing, it’s still useful to store domains in your metadata database as lookup tables, regular expressions, and SQL Statements.

Thank You – Q&A

Brian Hebert
www.scribekey.com