2011 Esri Developer Summit
Palm Springs, CA

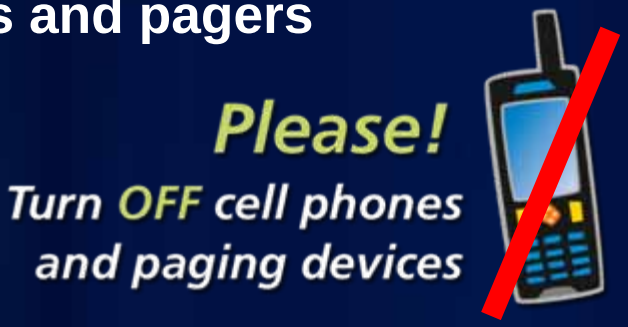
Developer's Guide to the Geodatabase

Craig Gillgrass
Colin Zwicker
David Crawford

esri

Schedule

- 3 hours with break 9:45am to 10:15am
- Cell phones and pagers



- Please complete the session survey – we take your feedback very seriously!

Seminar Roadmap

- We'll go over basic Geodatabase concepts
  - Might be a review for some of you
- Basic programming skills
  - C# demos
  - Code will be available with slides
  - Ability to read OMDs
  - ArcObjects supports: .Net, Java, C++, etc…

Seminar Roadmap …

- Seminar focus on:
  - Overview of the Geodatabase
  - How to work with the Geodatabase API
  - Tip\Tricks for using the API
  - Highlight some of the most common mistakes we've seen developers make with the Geodatabase API

Seminar Roadmap …

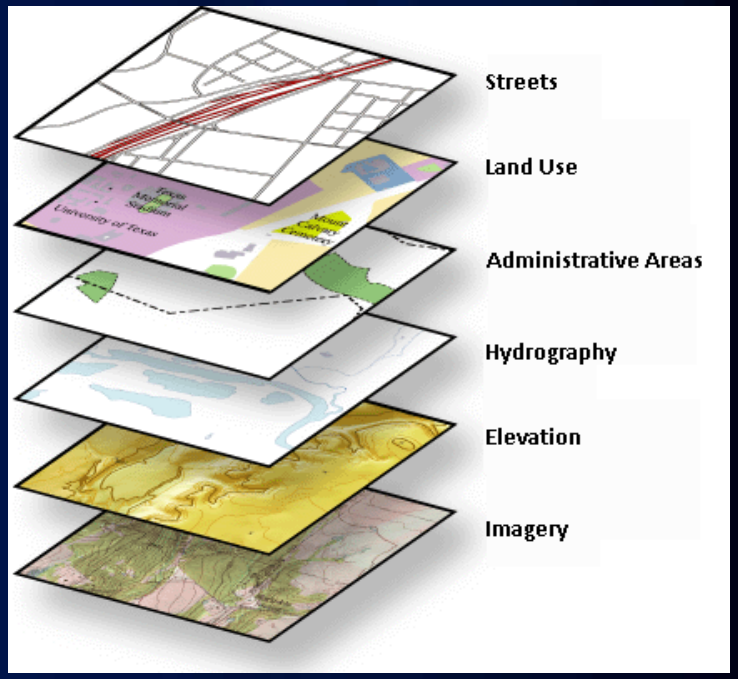
- We'll take questions throughout
  - May want to hold off until we get to the end of each topic
  - Hold questions to the end of demos
- Available at the break
- Showcase area over the next few days

Presentation Outline

- **Introduction to the Geodatabase**
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- File Geodatabase API
- Questions and other information

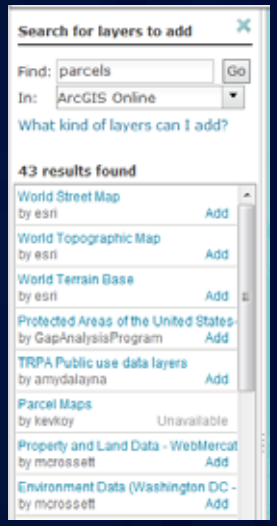
Building a Map – Working with Thematic Layers

- A map is built around thematic layers of information overlayed on a common geographic area.
- E.g. - road network, a collection of parcel boundaries, soil types, an elevation surface, satellite imagery for a certain date, well locations, and so on…
- ArcGIS makes it easy for users to find, create, use and share thematic layers

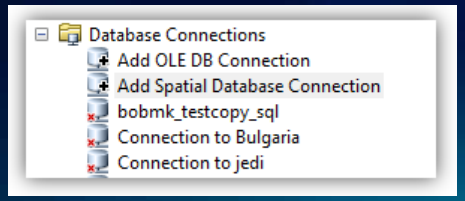


Adding thematic layers

- **ArcGIS Online**
  - **Find useful thematic layers by searching online content**
  - **Work with them alongside your own data**



- **Adding thematic layers from your geodatabase**



What is the geodatabase?

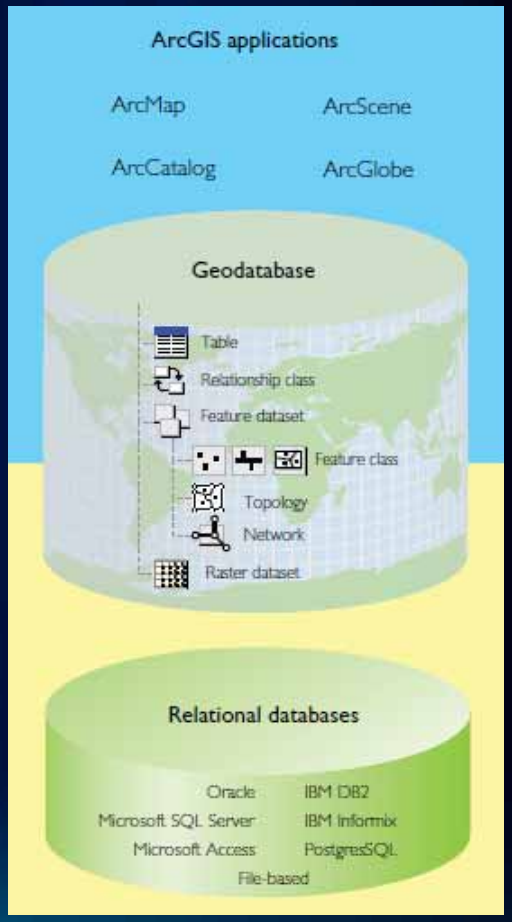
- **A collection of geographic datasets of various types used for managing spatial and non-spatial data.**
  - **Core ArcGIS data model**
  - **A physical store of geographic data**
  - **A transactional model for managing GIS workflows**
  - **Set of COM components for accessing data**

Geodatabase Data Management Approach

- **The geodatabase is built on an extended relational database**
  - **Relational integrity**
  - **Reliability, Flexibility, Scalability**
  - **Supports continuous, large datasets**
  - **Standard relational database schema**
  - **Base short transaction model**
  - **Support structured query language (SQL)**

Geodatabase Data Management Approach …

- Simple features + logic
  - All geographic data stored as tables in a DBMS
  - Extend functionality and data integrity
  - Functionality is consistent across DBMS'
- Application logic (software)
  - Works on standard DBMS tables
  - Implements GIS integrity and behavior
  - Business rules, topology, networks



Geodatabase Data Management Approach …

- Editing and data compilation
  - Maintain spatial and attribute integrity
  - Undo and redo edits
- Versioning work flows
  - Multiple users editing over long periods of time
  - Archiving
  - Distributed data management
- Components for accessing data
  - Robust, customizable framework
  - Build and manage your own specific GIS solution

Geodatabase is based on relational principles

- Leverages key DBMS principles and concepts to store geographic data as tables in a DBMS
  - Data is organized into tables
  - Tables contain rows
  - All rows in a table have the same attributes
  - Each attribute has a type
  - Relational integrity rules exist for tables
- The core of the geodatabase is a standard relational database schema
  - a series of standard database tables, column types, indexes, and other database objects

Geodatabase is based on relational principles …

- A feature class is stored as a simple DBMS table
- Each row represents a feature
- The fields in each row represent various characteristics or properties of the feature
- One of the fields holds the feature geometry which is stored as a spatial type

Parcels

OBJECTID *	SHAPE *	PROPERTY_I *	Res	Zoning_simple	SHAPE_Length	SHAPE_Area
1	Polygon	5001	Non-Residential	<Null>	3597.780813	112552.418591
2	Polygon	5002	Non-Residential	<Null>	814.855837	18488.417709
3	Polygon	1003	Residential	Residential	489.655523	12815.591379
4	Polygon	1004	Residential	Residential	521.761248	14036.135346
5	Polygon	1005	Residential	Residential	453.479649	9816.352665

Geodatabase is based on relational principles …

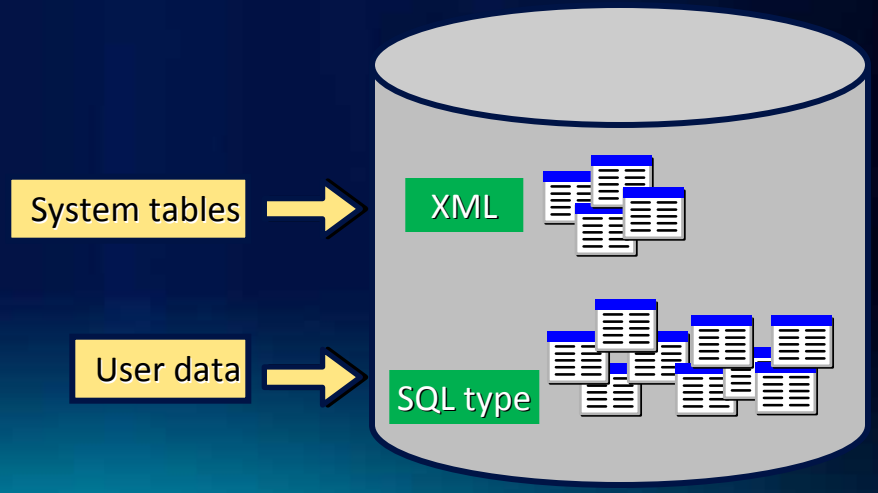
- A feature class is stored as a simple DBMS table
- Each row represents a feature
- The fields in each row represent various characteristics or properties of the feature
- One of the fields holds the feature geometry which is stored as a spatial type

Parcels

OBJECTID *	SHAPE *	PROPERTY_I *	Res	Zoning_simple	SHAPE_Length	SHAPE_Area
1	Polygon	5001	Non-Residential	<Null>	3597.780813	112552.418591
2	Polygon	5002	Non-Residential	<Null>	814.855837	18488.417709
3	Polygon	1003	Residential	Residential	489.655523	12815.591379
4	Polygon	1004	Residential	Residential	521.761248	14036.135346
5	Polygon	1005	Residential	Residential	453.479649	9816.352665

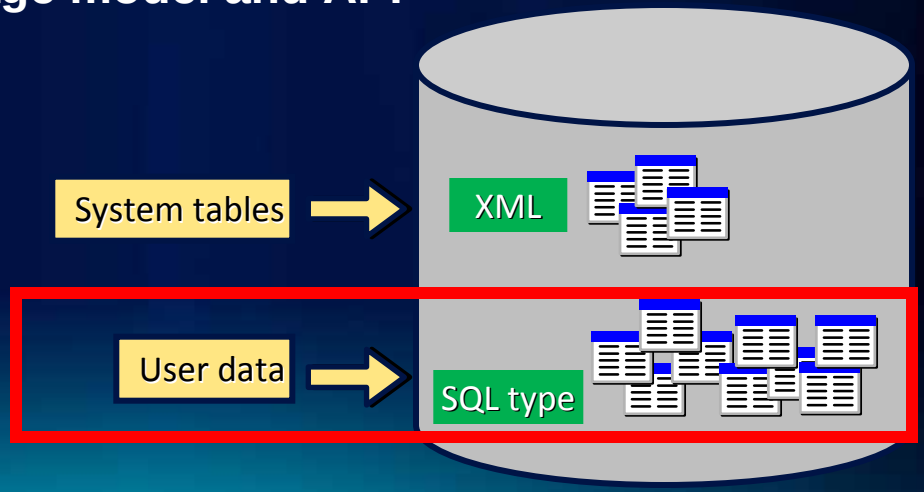
Geodatabase Schema

- There are two sets of tables:
  - Dataset tables (user-defined tables)
  - Geodatabase system tables



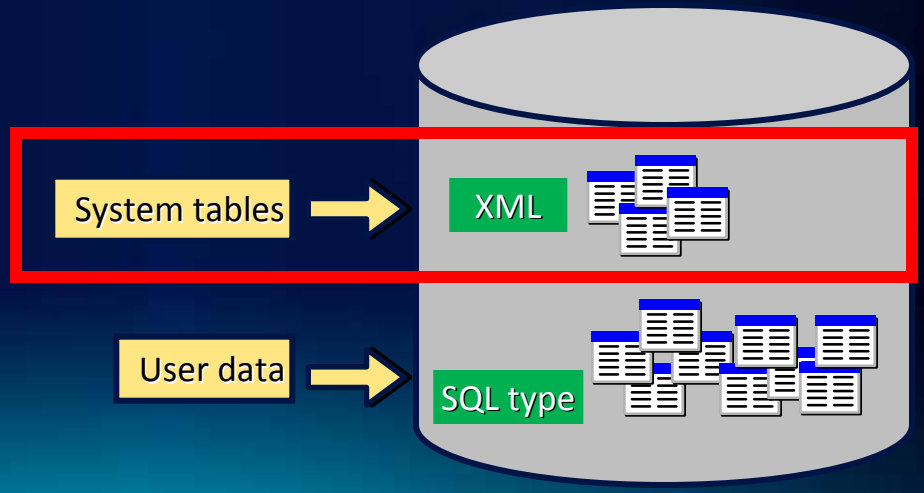
User-defined tables

- Stores the content of each dataset in the geodatabase
- Datasets are stored in 1 or more tables
- Spatial Types enhance the capabilities of the geodatabase
  - SQL access to geometry
  - Industry standard storage model and API



Geodatabase system tables

- System tables store definitions, rules, and behavior for datasets
- Tracks contents within a geodatabase
- 4 main system tables
- Geodatabase schema is stored primarily within an XML field

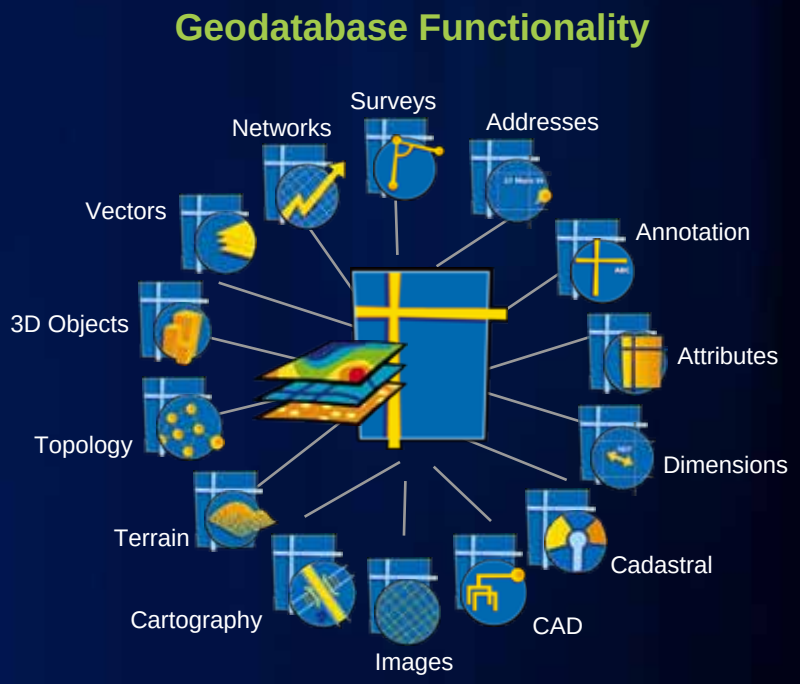


Modeling Real-World Data with the Geodatabase

- A geodatabase contains datasets.
  - Datasets represent collections of information with a real-world interpretation.
  - Types of geographic datasets:
    - Tables
    - Object classes, feature classes, relationship classes
    - Feature datasets
    - Networks, Topologies, Raster and cadastral datasets
- Datasets have associated information to help manage integrity, behavior, and interpretation
  - Domains, Relational integrity, Topology, Metadata

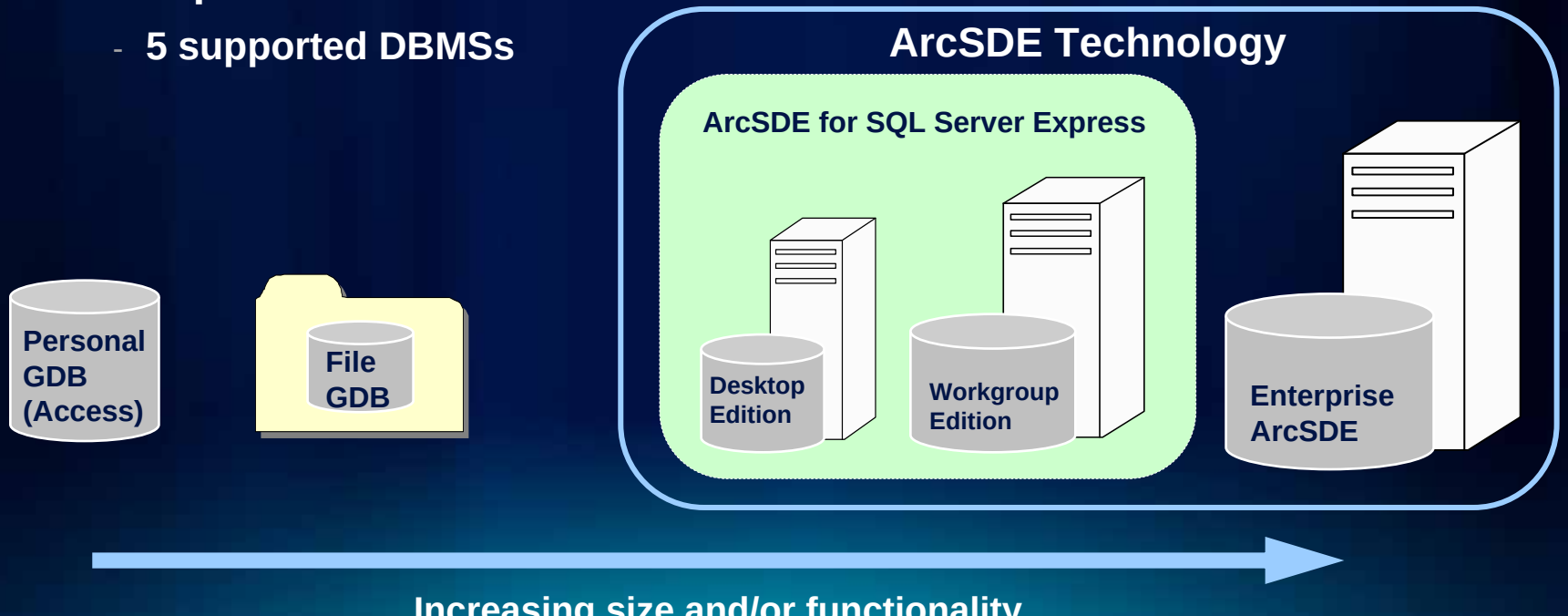
Modeling Real-World Data with the Geodatabase …

- **The geodatabase enhances data and thematic layers by adding rules and behavior**
  - Spatial and relational integrity rules
  - Data validation
  - Business logic
- **Create thematic layers with behavior**
  - Road and utility networks
  - Parcel fabrics
  - Terrain and 3D surfaces
  - Location services
- **Extended framework for advanced workflows and editing**
  - Multiuser editing, Data Replication, Editor tracking, Archiving

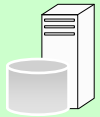


3 Types of Geodatabases

- Geodatabases can be stored different ways
  - Personal geodatabase (Access mdb file)
  - File geodatabase
    - Directory of binary files
  - ArcSDE for SQL Server Express
  - Enterprise ArcSDE
    - 5 supported DBMSs

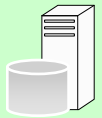


3 Types of Geodatabases…

	Personal GDB	File GDB	SDE GDB (3 editions)
			
Storage format	Microsoft Access	Folder of binary files	DBMS
Storage capacity	2 GB	1 TB per table*	Depends on edition
Supported O/S platform	Windows	Any platform	Depends on edition
Number of users	Single editor Multiple readers	Single editor Multiple readers	Multiple editors & readers
Distributed GDB functionality	Check out/check in replication	Check out/check in replication	Replication (all types) & versioning

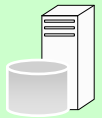
* By default; option to have 256 TB per table

3 Types of Geodatabases…

	Personal GDB	File GDB	SDE GDB (3 editions)
			
Storage format	Microsoft Access	Folder of binary files	DBMS
Storage capacity	2 GB	1 TB per table*	Depends on edition
Supported O/S platform	Windows	Any platform	Depends on edition
Number of users	Single editor Multiple readers	Single editor Multiple readers	Multiple editors & readers
Distributed GDB functionality	Check out/check in replication	Check out/check in replication	Replication (all types) & versioning

* By default; option to have 256 TB per table

3 Types of Geodatabases…

	Personal GDB	File GDB	SDE GDB (3 editions)
			
Storage format	Microsoft Access	Folder of binary files	DBMS
Storage capacity	2 GB	1 TB per table*	Depends on edition
Supported O/S platform	Windows	Any platform	Depends on edition
Number of users	Single editor Multiple readers	Single editor Multiple readers	Multiple editors & readers
Distributed GDB functionality	Check out/check in replication	Check out/check in replication	Replication (all types) & versioning

* By default; option to have 256 TB per table

Demo

- Tour around a geodatabase – Part 1
- Tour around a geodatabase – Part 2
- Create a file geodatabase (.gdb)
- Create a feature dataset and feature class
- Tour around a geodatabase – Part 3
- Creating domains, subtypes and rules

Presentation Outline

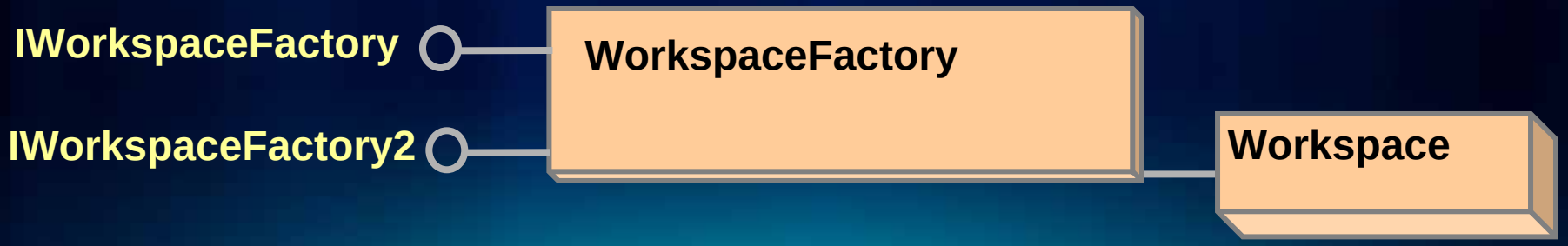
- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- File Geodatabase API
- Questions and other information

Licensing

- At ArcGIS 10 engine applications are required to Bind to a product before performing the required initialization of a license level.
  - ESRI.ArcGIS.Version.RunTimeManager.Bind(ESRI.ArcGIS.ProductCode.Desktop)
- Required to initialize licensing to correct level when working with geodatabase in Engine environment
  - If not, get an error on call to open the workspace
- Configure license at application start time
  - Geodatabase Update extension
  - esriLicenseProductCodeEngineGeoDB

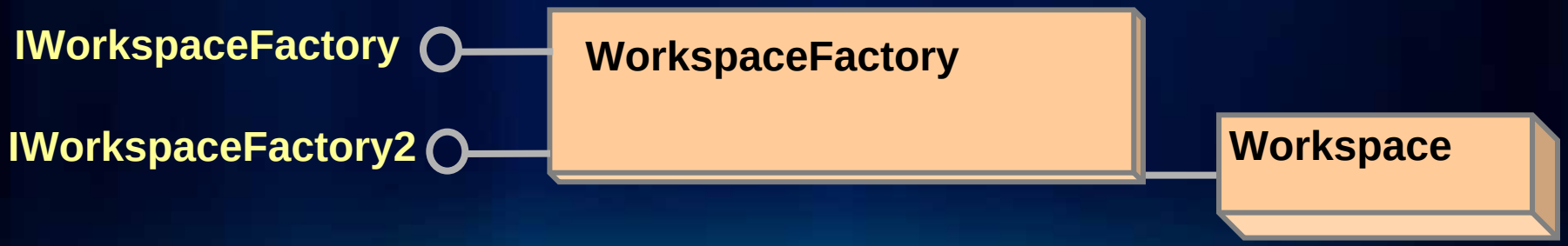
Workspace Factory and Workspaces

- Workspace Factory is a dispenser of workspaces
  - Personal, File, ArcSDE workspaces, SQL workspaces **(new @ 10)**
- Create a factory from a workspace factory coclass
- Workspace factories are singleton objects
  - One instance created per Component Object Model (COM) apartment
  - Further calls return a reference to the existing object



Workspace Factory and Workspaces

- A workspace is a container of datasets.
  - Geodatabase, coverage workspace, folder of shapefiles
- Workspaces are not singleton objects
  - But, requesting a workspace with the same properties as an existing instance returns a reference to it
  - The Geodatabase guarantees unique instancing



Making a connection

- Create a workspace from a factory
  - Path to data and window handle (app ID)

```
IWorkspaceFactory sdeWkspFact = new SdeWorkspaceFactoryClass();
IPropertySet propset = new PropertySetClass();

propset.SetProperty("SERVER",“crimsontide");
propset.SetProperty("INSTANCE","5151");
propset.SetProperty("USER","brent");
propset.SetProperty("PASSWORD","brent");
propset.SetProperty("DATABASE","null");
propset.SetProperty("VERSION","SDE.DEFAULT");
propset.SetProperty("AUTHENTICATION_MODE","DBMS");

IWorkspace workspace = sdeWkspFact.Open(propset, 0);
```

- Connecting using a connection file

```
string nameOfFile = "D:\\data\\redarrow.sde”;
IWorkspace workspace = workspaceFactory.OpenFromFile(nameOfFile, 0);
```

Geoatabase specific information

- Some aspects of Geodatabases differ from one another
- Don’t assume field names remain constant
  - Can differ between geodatabases
  - IFeatureClass::ShapeFieldName property
- ISQLSyntax interface can be used to determine SQL predicates, clauses, and other database specific constraints are supported by a workspace
  - ISQLSyntax.GetSpecialCharacter(esriSQL_DelimitedIdentifierPrefix)
  - ISQLSyntax.GetFunctionName(esriSQL_UPPER)
  - Important for queries
    - Can determine supported clauses for a data source
- IWorkspaceProperties

Creating Datasets

- Dataset model
  - Open methods on IFeatureWorkspace, IFeatureClassContainer
- Dataset Extensibility model
  - Open methods on IDatasetContainer2
- Use a name – "Streets“
  - Use ISQLSyntax::QualifyTableName for SDE
  - Owner not required to qualify name
  - IWorkspace2::NameExists
- Use a ClassID
  - ClassIDs are unique and sequential within geodatabase

Creating Datasets …

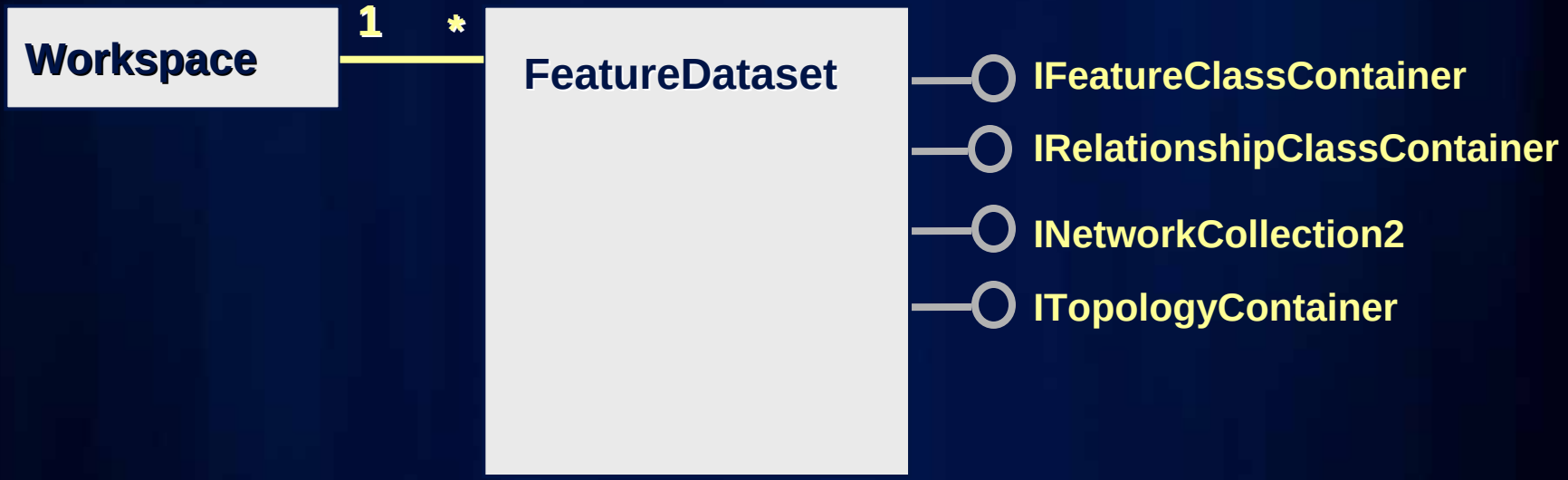
- Create methods to match the open methods
  - FeatureClass
  - FeatureDataset
  - Table
- Dataset model
  - Properties of dataset to be created arguments to Create method
- Dataset Extensibility model
  - Use a Data Element that has been pre-populated
  - For Network Datasets, Terrains, Representations, Cadastral Fabrics

Creating Datasets …

- Creation of datasets is a type of DDL command
  - Data definition language - database commands that modify the schema of a database
- Should never be called inside of an edit session
- Commit any transactions that are currently open, making it impossible to rollback any unwanted edits should an error occur

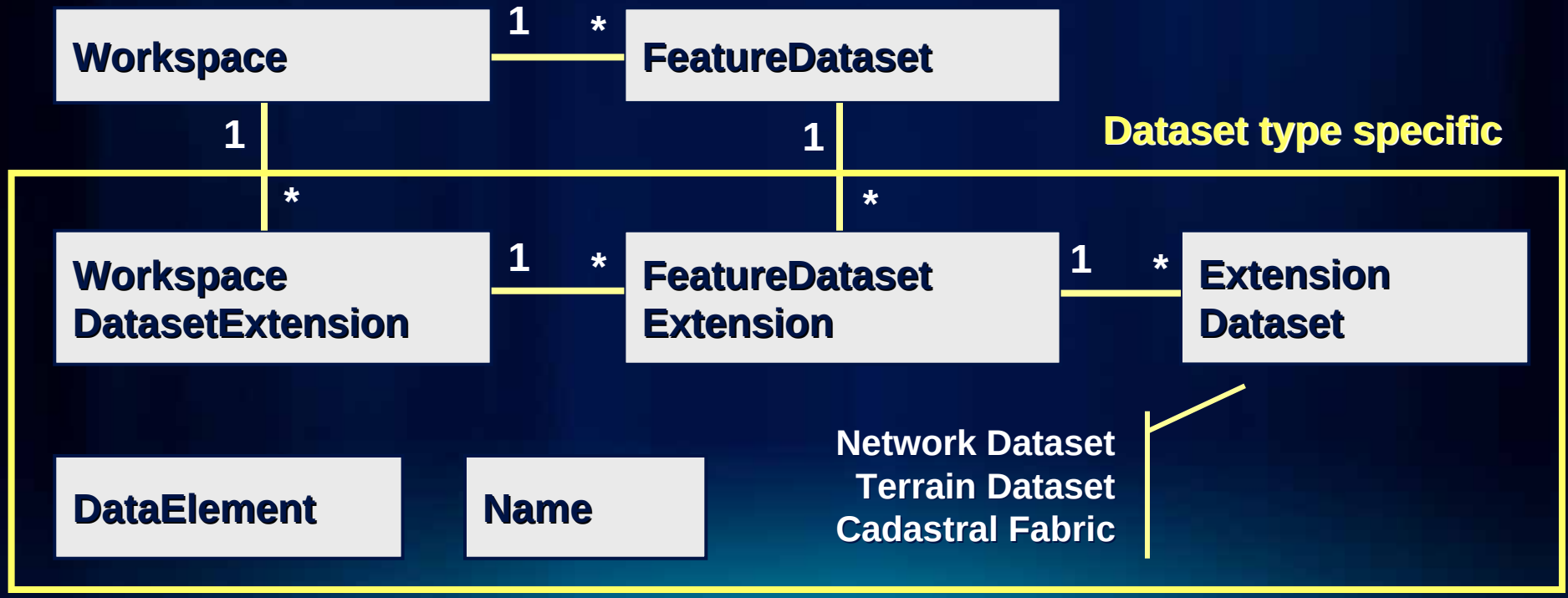
Dataset Model

- Extend existing objects with new interfaces for each specific dataset



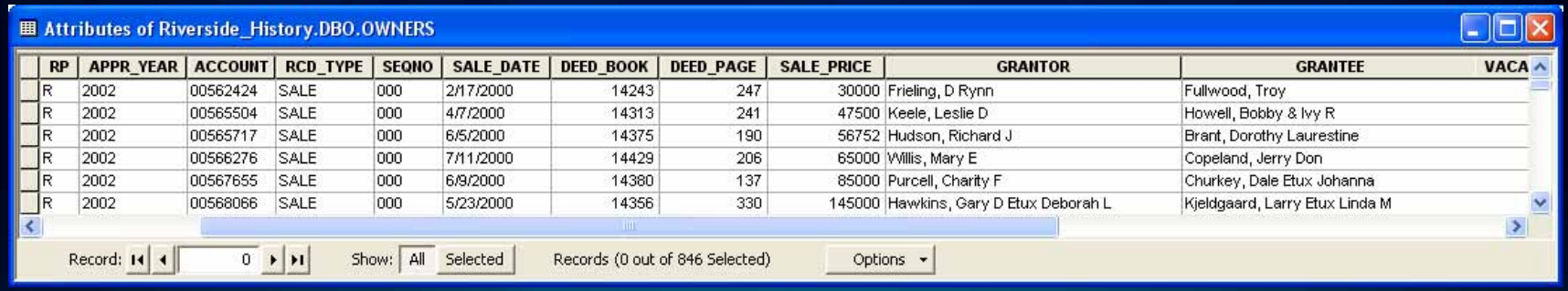
Dataset Extensions Model

- Utilizes an enhanced workspace extension operating with a new dataset, feature dataset extension (optional), data element, and name object



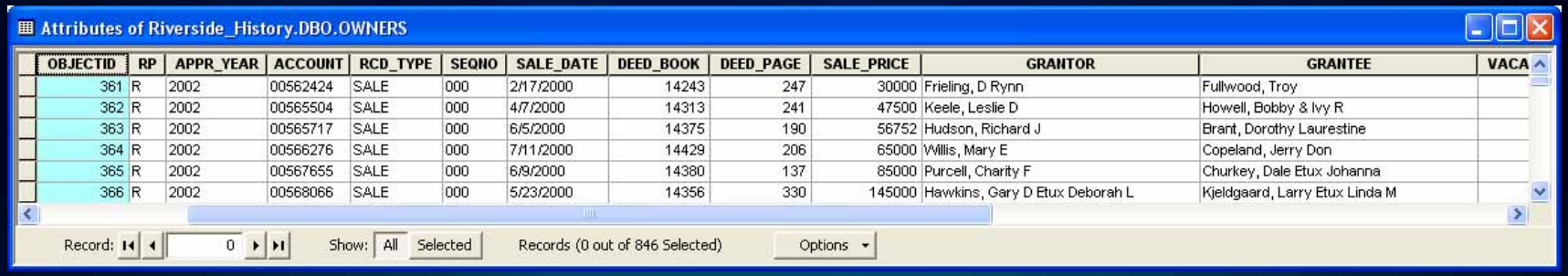
Rows and Tables

- Type of Dataset in the Geodatabase
  - Containing zero to many rows
  - One to many columns
  - All rows in the table have the same schema
- ITable interface provides Table management abilities
  - View and modify schema
  - Add and remove rows
  - Perform queries



Objects and Object Classes

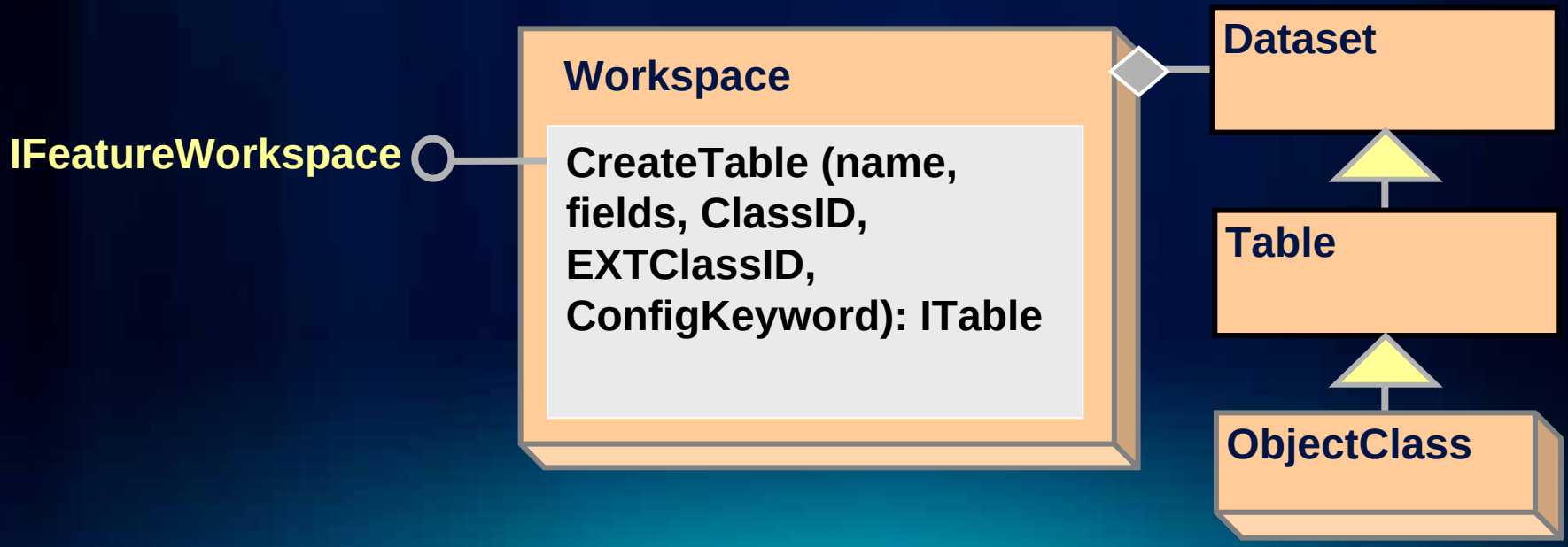
- Objects are entities with properties and behavior
  - Extend the Row concept
- An Object Class is a Table with Geodatabase behavior
  - Registered with the Geodatabase
  - An object is an instance of an object class
  - All objects in an object class have the same properties and behavior
  - An object can be related to other objects via relationships.



OBJECTID	RP	APPR_YEAR	ACCOUNT	RCD_TYPE	SEQNO	SALE_DATE	DEED_BOOK	DEED_PAGE	SALE_PRICE	GRANTOR	GRANTEE	VACA
361	R	2002	00562424	SALE	000	2/17/2000	14243	247	30000	Frieling, D Rynn	Fullwood, Troy	
362	R	2002	00565504	SALE	000	4/7/2000	14313	241	47500	Keele, Leslie D	Howell, Bobby & Ivy R	
363	R	2002	00565717	SALE	000	6/5/2000	14375	190	56752	Hudson, Richard J	Brant, Dorothy Laurestine	
364	R	2002	00566276	SALE	000	7/11/2000	14429	206	65000	Willis, Mary E	Copeland, Jerry Don	
365	R	2002	00567655	SALE	000	6/9/2000	14380	137	85000	Purcell, Charity F	Churkey, Dale Etux Johanna	
366	R	2002	00568066	SALE	000	5/23/2000	14356	330	145000	Hawkins, Gary D Etux Deborah L	Kjeldgaard, Larry Etux Linda M	

Create an Object Class

- Creates an ObjectClass, returns ITable interface
  - ObjectID field. Values are Never Reused.
  - From a database perspective, can be viewed as a primary key
- Can also use it to create a Table
- Custom behavior ID's *(can use null for EXTClassID)*
- Configuration keywords *(can use "")*
- Can create Fields first : IObjectClassDescription:RequiredFields



Create an Object Class …

- Object Classes have geodatabase specific functionality that Tables do not, through several interfaces
  - IObjectClass
    - AliasName
    - ObjectClassID
    - RelationshipClasses
  - ISubtypes
    - AddSubtype
- Standard table operations, such as queries and row creation, are still performed on an object class using the ITable interface.

Features and Feature Classes

- Builds on the Relational Model
  - Extends the Object Class concept with a spatial attribute
- A feature is a spatial object.
- A feature is an instance of a feature class.
- Extended the relational model with
  - Geometry attribute types

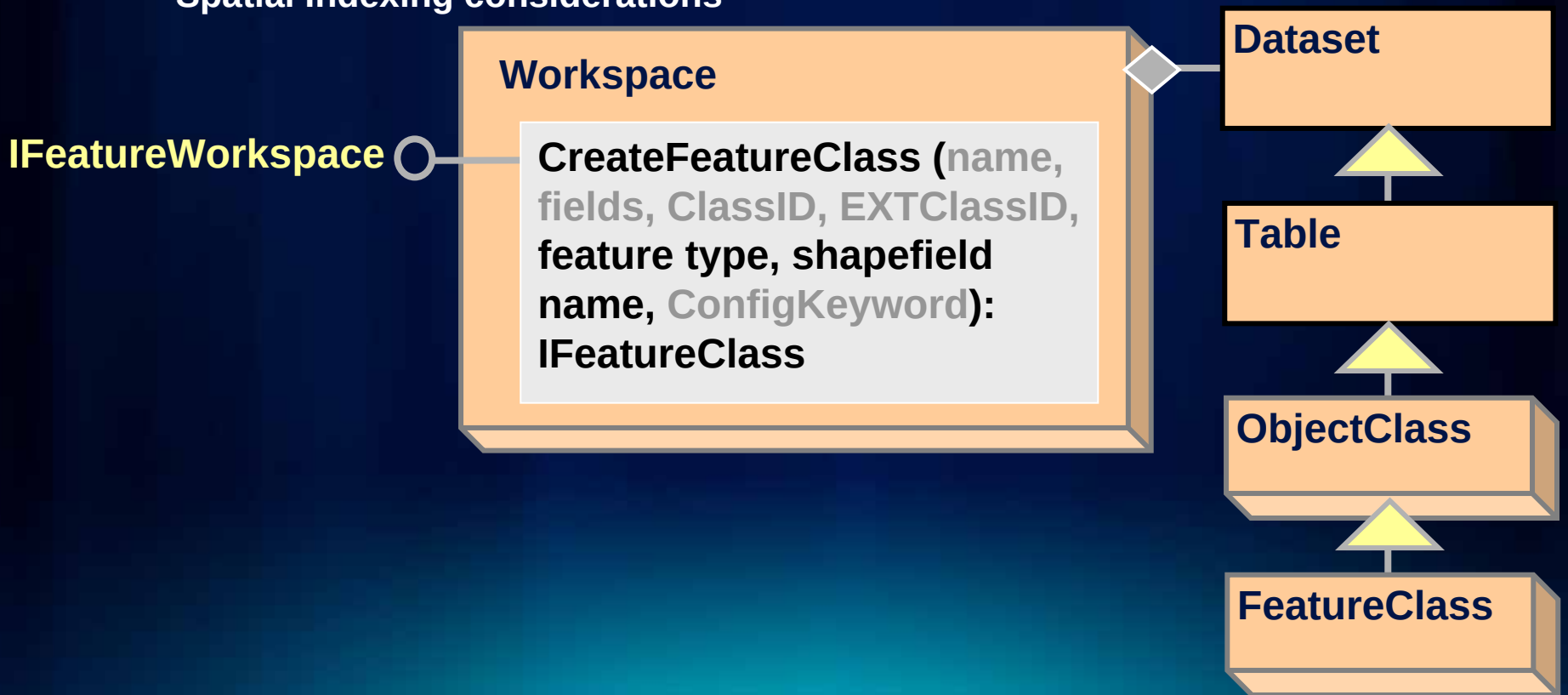
A feature class is a table of rows, where each row has a geographic column

Attributes of Parcels

OBJECTID*	SHAPE*	PARCEL_ID*	ZONE_CODE*	SHAPE_Length	SHAPE_Area
4513	Polygon	67970	W	544.053559	9259.209935
4514	Polygon	67971	W	158.545394	774.602847
4515	Polygon	67973	R60M	400.003008	7499.965473
4516	Polygon	67974	B1	236.126101	2905.890606
4517	Polygon	67982	B1	550.458538	17499.011493

Create a Feature Class

- **Same as CreateTable except:**
  - **Needs Shape type and Shape field name**
    - **IGeometryDefEdit used when defining new feature class**
    - **Use IObjectClassDescription:RequiredFields**
  - **Spatial Indexing considerations**



Create a Feature Class …

- Feature classes that store simple features can be organized either inside or outside a feature dataset
  - Outside a feature dataset are called standalone feature classes
  - Feature classes which store topological features, for example those participating in geometric networks, must be contained within a feature dataset to ensure a common spatial reference
- Each dataset in a geodatabase must have a unique name for the type of dataset
  - Feature classes must have a unique name

Table – Object Class – Feature Class

	Table	Object Class	Feature Class
Contains	Rows	Objects	Features
Row Creation Method	ITable.CreateRow	ITable.CreateRow	IFeatureClass. CreateFeature
Required Fields	None	ObjectID	ObjectID, shape
Registered with the Geodatabase	FALSE	TRUE	TRUE
Supports subtypes, domains	FALSE	TRUE	TRUE

Table – Object Class – Feature Class

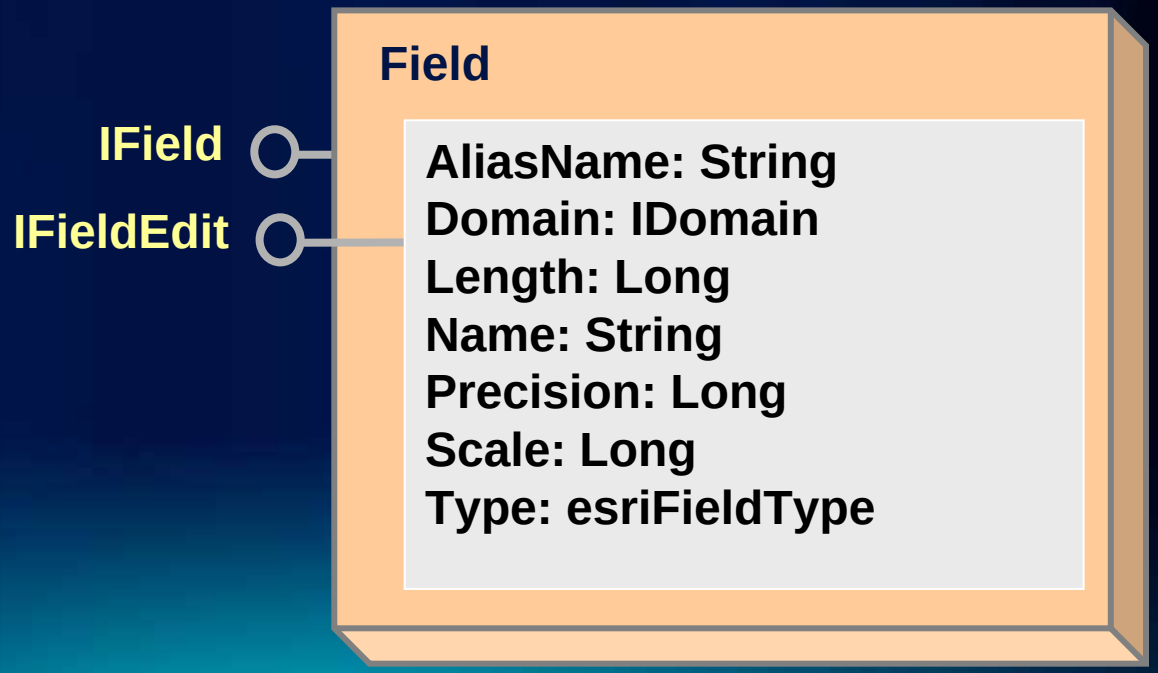
	Table	Object Class	Feature Class
Contains	Rows	Objects	Features
Row Creation Method	ITable.CreateRow	ITable.CreateRow	IFeatureClass.CreateFeature
Required Fields	None	ObjectID	ObjectID, shape
Registered with the Geodatabase	FALSE	TRUE	TRUE
Supports subtypes, domains	FALSE	TRUE	TRUE

Table – Object Class – Feature Class

	Table	Object Class	Feature Class
Contains	Rows	Objects	Features
Row Creation Method	ITable.CreateRow	ITable.CreateRow	IFeatureClass. CreateFeature
Required Fields	None	ObjectID	ObjectID, shape
Registered with the Geodatabase	FALSE	TRUE	TRUE
Supports subtypes, domains	FALSE	TRUE	TRUE

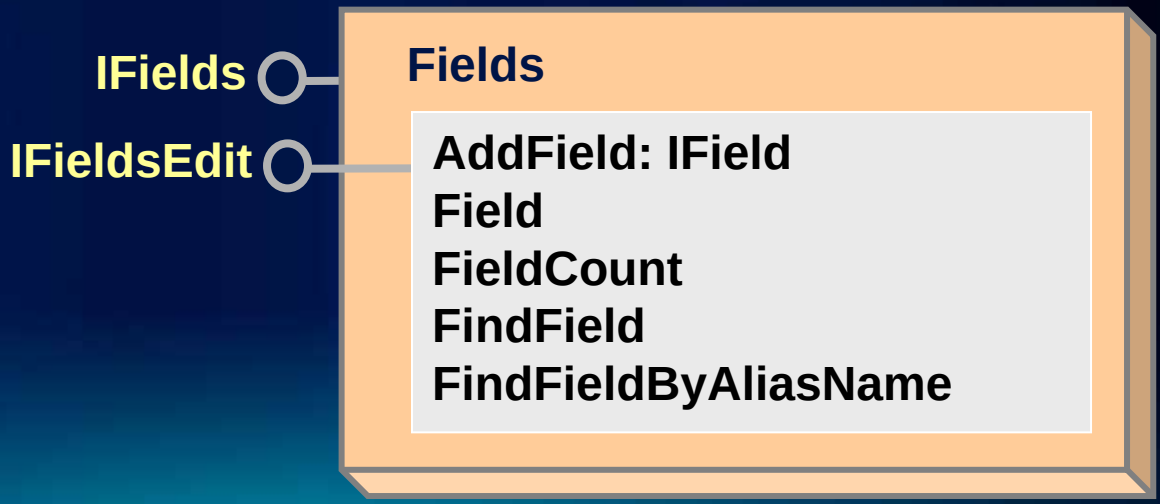
Create a Field

- Synonymous with a column
- Set properties with IFieldEdit
- Types include: Integer, Single, Double, String, Date, OID, Geometry, Blob, GUID, GlobalID, and Raster
- Geodatabase tables must have an ObjectID field
  - Using Class Descriptions will take care of this



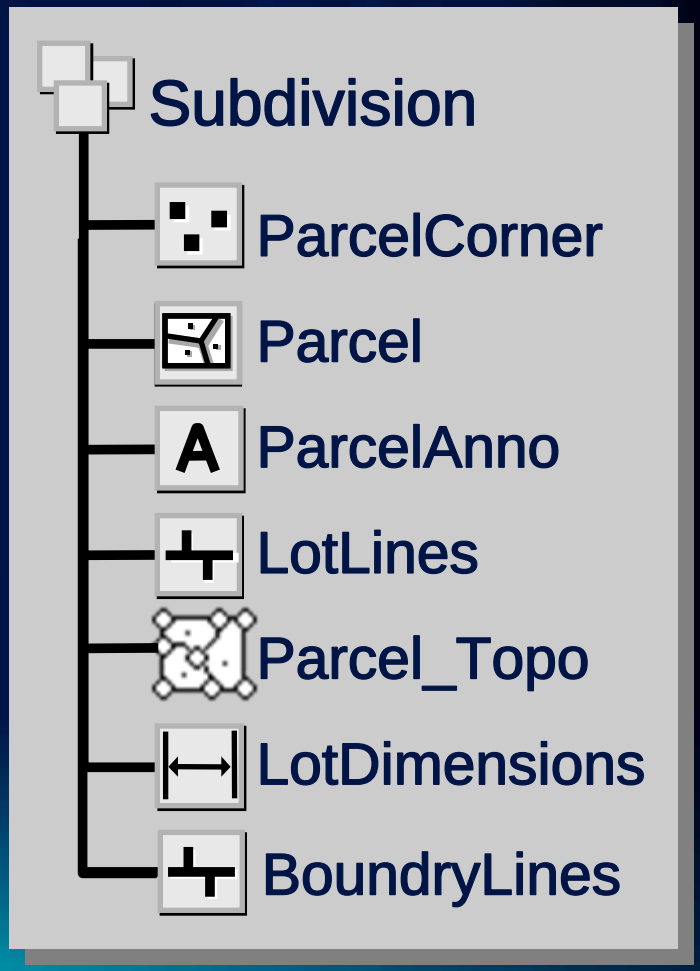
Create a Field …

- Use a Field collection (Fields) when creating datasets
  - For existing tables use IClass::AddField method to add fields and IClass::DeleteField method to delete fields
- Set properties for the Field with the IFieldEdit interface
- Leverage Class Description whenever possible
  - ObjectClassDescription, FeatureClassDescription, etc



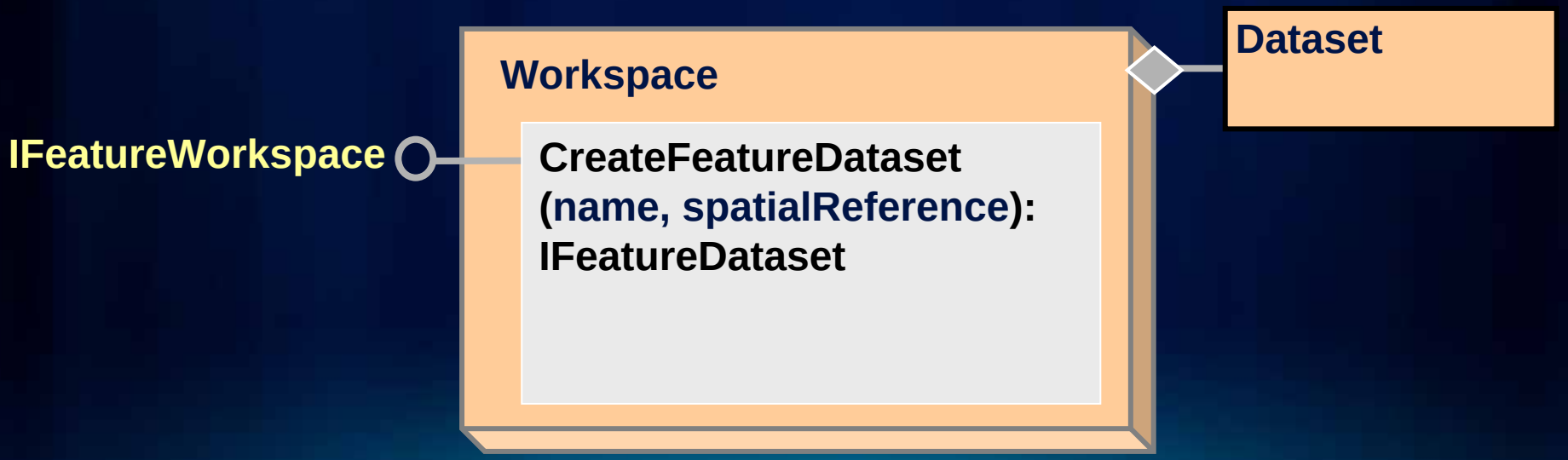
Feature Datasets

- A container object for datasets with the same spatial reference
  - Only exist within a Geodatabase
- Analogous to a coverage
  - Less restrictive
- Container for:
  - Geometric networks
  - Topologies
  - Network Datasets
  - Terrains
  - Cadastral Fabrics
  - Optionally relationship classes



Create a Feature Dataset

- Allows the creation of a Feature Dataset:
  - Methods supported by the returned feature dataset allow the creation of feature classes within the feature dataset
  - Can also be used to access datasets through the Dataset model
- Spatial Reference is required



Feature Datasets …

- Need to remember that feature classes may or may not belong to a feature dataset
  - The following code assumes a feature dataset exists and may fail:

```
IFeatureDataset featureDataset = featureClass.FeatureDataset;
IWorkspace workspace = featureDataset.Workspace;
```

  - This piece of code will work for standalone feature classes and those in feature datasets:

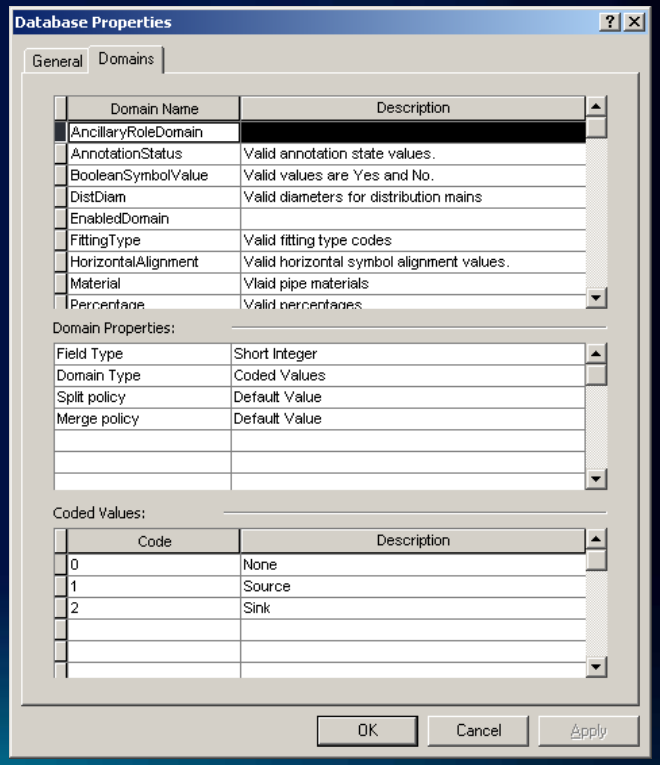
```
IDataset dataset = (IDataset)featureClass;
IWorkspace workspace = dataset.Workspace;
```

Demo

- Tour around a geodatabase – Part 1
- Tour around a geodatabase – Part 2
- Create a file geodatabase (.gdb)
- Create a feature dataset and feature class
- Tour around a geodatabase – Part 3
- Creating domains, subtypes and rules

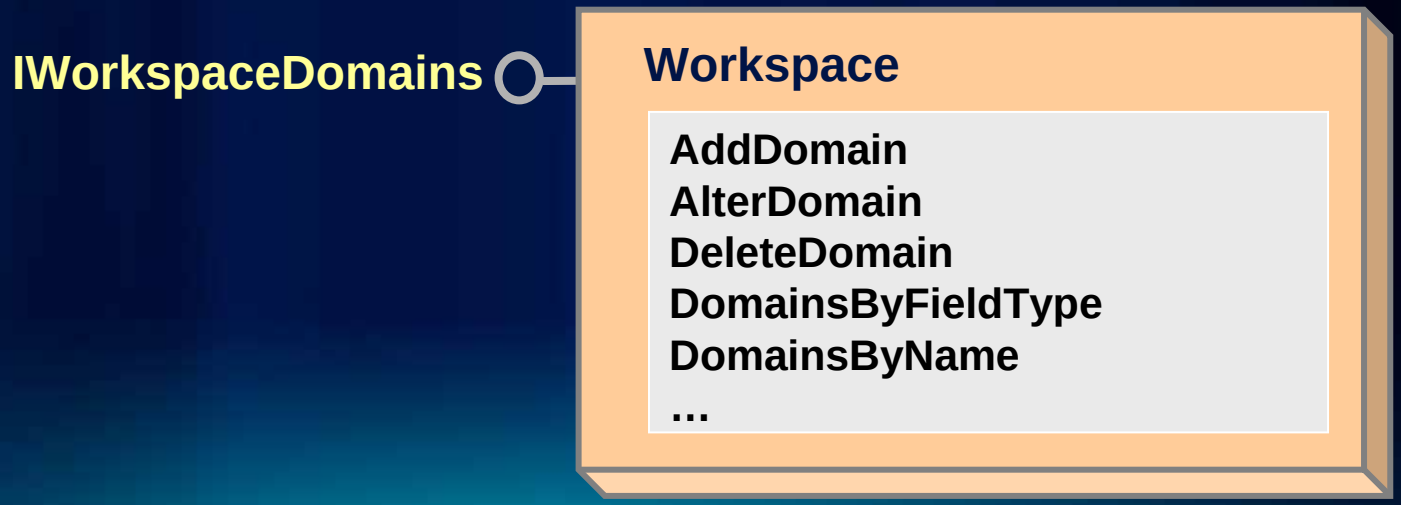
Domains

- Describe the legal values of a field type.
  - Used to ensure attribute integrity
- Defined at the workspace level
- Domains constrain field values
- Associate a domain to a field(s)
  - During create use IFieldEdit:Domain
  - Or IClassSchemaEdit:AlterDomain
- Two types:
  - Coded Value and Domain



Domains …

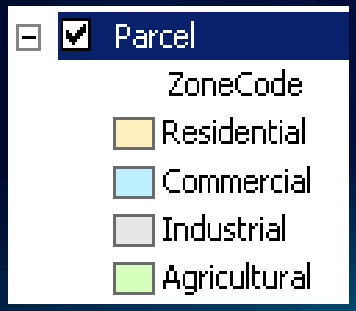
- Use the IWorkspaceDomains interface to manage the collection of domains found within a workspace.
  - Since Domains are shared amongst feature classes; the management of them is at the workspace level
- DeleteDomain will fail if the domain is associated with a field
- Domain names are unique across a workspace
  - Need to check for existence of Domain name prior to creation or trap for the error



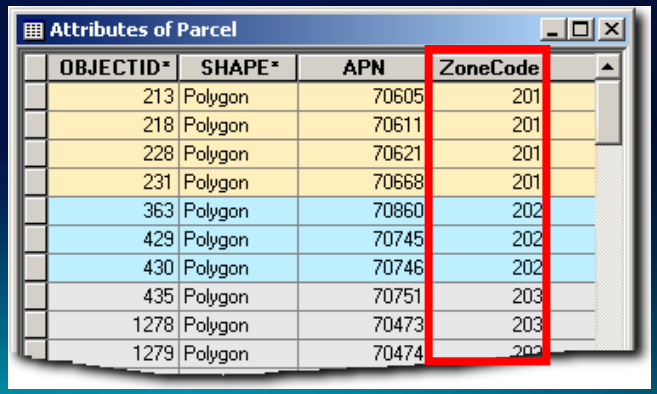
Subtypes

- Subtypes are specific to feature class
  - Partition the objects in a class into like groups
  - Can be used with Tables and Feature Classes
- Defined at the class level
  - Can only be Short or Long Integers
- Defined by the value of a subtype field
  - Have the same attribute\behavior schema
  - Can have different default values and domains for each field
  - Can define topology rules between subtypes

Descriptions

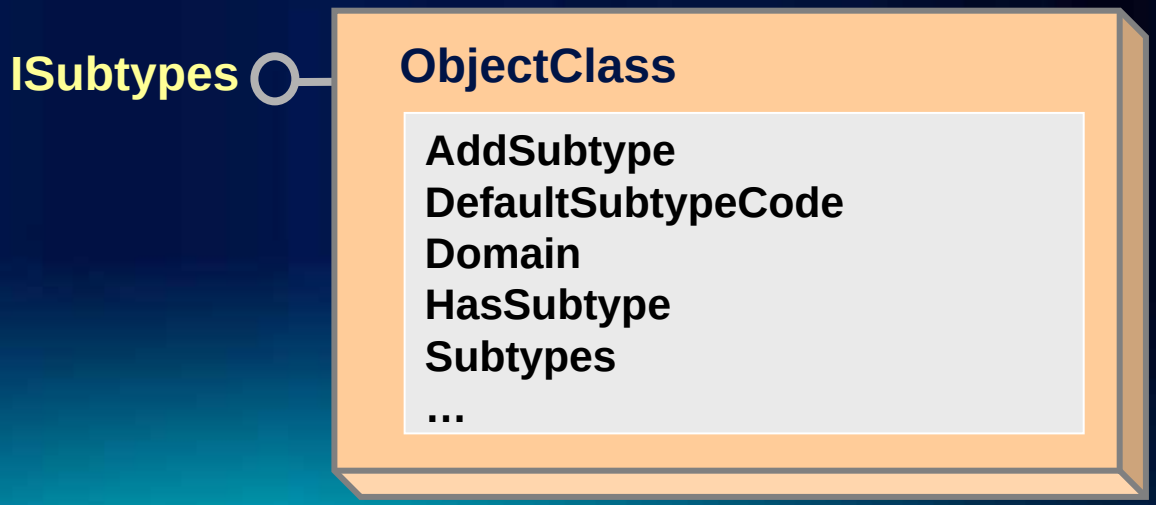


Codes



Subtypes …

- Once set, need to check attribute, connectivity and relationship rules at subtype level
- Each object class has a default subtype code
  - Important for feature creation and editing
- Subtypes are the first to be checked during Validation
- The ISubtypes interface is used for managing subtypes and the associated default values and attribute domains



Subtypes …

- **Basic process for setting Subtypes**
  - Define subtype field
    - ISubtypes:SubtypeFieldName
  - Check valid values of field
    - For populated classes
  - Assign Subtype code and name
    - ISubtypes:AddSubtype (code, name)
  - ISubtypes:DefaultSubtypeCode
    - Define default subtype code
  - Assign Default Values and Domains at the Subtype level
    - ISubtypes:DefaultValue
    - ISubtypes:Domain

Validation Rules

- Store attribute, connectivity, and relationship rules on objects as part of the geodatabase.
- Predefined, parameter driven
  - Subtypes
  - Attribute range rule or Attribute set rule
  - Connectivity rule
  - Relationship rule
- Rules are evaluated at a user specified time
  - Not performed when the feature is created, stored, edited, etc
  - Geodatabase has an optimistic view of the data
  - Allows you to load your data; then validate and correct it

Validation Rules

- Use the IValidate::Validate or IValidation::Validate methods to evaluate rules
  - There is an order to how rules are validated:
    - Validate the subtype
    - Validate the attribute rules
    - Validate the network connectivity rules (if network feature)
    - Perform custom validation (using optional class extension)
    - Validate the relationship rules
  - Validation stops once a check returns false
- Perform custom validation by writing code

Schema Locks

- Prevent clashes with other users when changing the geodatabase structure
- Exclusive and Shared
  - ISchemaLock primarily used for establishing exclusive lock
  - Shared locks are applied when accessing the object
  - Promote a Shared lock to an Exclusive lock
  - Only one Exclusive lock allowed
- Exclusive locks are not applied or removed automatically

Schema Locks …

- When to use?
- You must promote a shared lock to exclusive when performing schema modification such as:
  - Modifications to attribute domains; coded or range
  - Adding or deleting a field to a feature or object class
  - Associating a class extension with a feature class
  - Creating a Topology, Geometric Network, Network Dataset, Terrain, Schematic Dataset, Representation or Cadastral Fabric on a set of feature classes
  - Any use of the IClassSchemaEdit interfaces
  - IFeatureClassLoad.LoadOnlyMode
  - Rebuilding spatial and attribute indexes

Schema Locks …

- Demote Exclusive lock to Shared lock when the modification is complete
  - Includes when errors are raised during the schema modification
- Keep your use of Exclusive schema locks tight
  - Prevents clashes with other applications and users
- If your application keeps a reference to an object with an Exclusive schema lock
  - You will need to handle the Exclusive Schema lock when the object is used

Demo

- Tour around a geodatabase – Part 1
- Tour around a geodatabase – Part 2
- Create a file geodatabase (.gdb)
- Create a feature dataset and feature class
- Tour around a geodatabase – Part 3
- Creating domains, subtypes and rules

Creating Rows and Features

- Basic process to create row or feature
  - CreateRow or CreateFeature
    - Can also use InsertCursor, more later
  - If subtypes present, set IRowSubtypes::SubtypeCode
  - If default values, call IRowSubtypes::InitDefaultValues
  - Set attribute values
  - Create geometry and set Shape
  - Call Store
    - Writes the values to the record in the table

Simple vs. Complex Features

- Within the geodatabase, behavior is dependent upon whether a feature is simple or complex
- Simple features
  - Point, line, polygon, multipoint, multipatch features
  - Simple Relationships
- Complex features
  - Network features (simple edge, simple junction, complex edge)
  - Annotation features
  - Dimension features
- Any dataset specific behavior (i.e.: for features created in geometric networks) is handled at creation time
  - Not required to call Connect or create Dirty Areas

Simple vs. Complex Features …

- Cursors
  - Insert cursors can perform direct inserts outside of an edit session on simple data
    - Same rule applies to update cursors
    - Offers performance advantages; i.e.: events not fired
  - Using these APIs on complex objects (or on objects participating in composite relationships or relationships with notification) negates any performance advantages

Create Features Demo

Walkthrough the basic process to create a feature

Presentation Outline

- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- File Geodatabase API
- Questions and other information

Geodatabase Editing - Edit Session

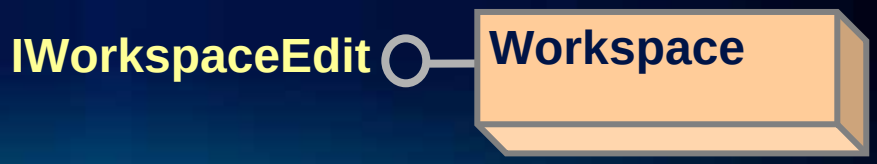
- Geodatabase explicitly stores change information when edited
- Only see the changes you've made within the edit session
  - Changes made by other applications are not seen
  - Until Save or Discard
- Edits should be made within an edit operation
  - StartEditOperation – StopEditOperation
  - Perform the edit as quickly as possible
  - Keep edit operation "tight and compact"
  - Collect the required information before starting the edit operation

Geodatabase Editing - Edit Session …

- Each edit operation represents a transaction
  - Stop commits the change
  - Abort rolls back, like undo
- Applications are responsible for calling:
  - AbortEditOperation when errors are detected
  - StopEditOperation to complete edit operations
    - Pushes the edit operation onto the undo stack
- UndoEditOperation, RedoEditOperation
  - Geodatabase moves the operation between the undo and redo stacks

Editing the Geodatabase

- When to use edit sessions?
  - Must use with topologies, geometric networks, etc
  - Use IObjectClassInfo2::CanBypassEditSession
- When to use IEditor or IWorkspaceEdit?
  - Use IEditor to edit within an application, like ArcMap
  - Ensures undo/redo consistency between edits made programmatically and through the UI
  - Must use IWorkspaceEdit in Engine environment
- Similar methods on each



Other Useful Method When Editing

- IDatasetEdit.IsBeingEdited
  - Determine if a particular dataset is participating in the edit session
- IWorkspaceEdit2.IsInEditOperation
  - Determine if the workspace is currently in an edit operation
  - Use when deciding whether to start an edit operation
- IWorkspaceEdit2.EditDataChanges
  - Determine which features have been changed with the scope of an edit session or edit operation.

Editing Demo

- Update Feature
- Edit Operations

Presentation Outline

- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- **Beyond Basics**
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- File Geodatabase API
- Questions and other information

Beyond Basics

- Relationship classes
- Rasters
  - Raster Datasets, Raster Catalogs and Mosaic Datasets
- Geodatabase XML

Relationship Classes

- **What is a Relationship Class?**
  - **An association between two object classes that is persisted within the Geodatabase**
  - **A class may participate in multiple relationship classes.**

Parcel

OID	Zone	Area
28	Commercial	10000
794	Residential	5000
858	Residential	6050

Buildings

OID	Parcel_ID	Building Type
1	28	Office Building
2	794	Townhouse
3	794	Townhouse
4	858	Condo
5	858	Condo

Relationship Classes

- **What is a Relationship Class?**
  - **An association between two object classes that is persisted within the Geodatabase**
  - **A class may participate in multiple relationship classes.**

ParcelToBuilding

Parcel

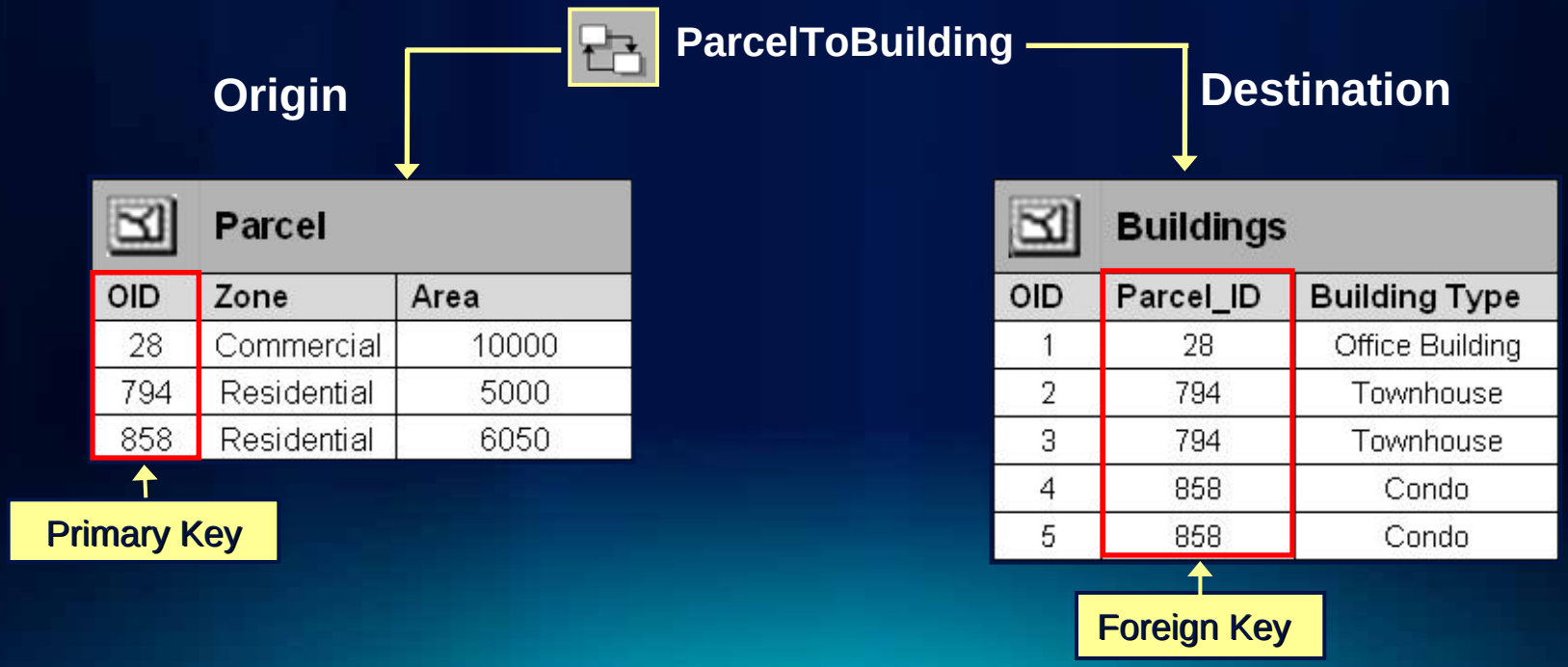
OID	Zone	Area
28	Commercial	10000
794	Residential	5000
858	Residential	6050

Buildings

OID	Parcel_ID	Building Type
1	28	Office Building
2	794	Townhouse
3	794	Townhouse
4	858	Condo
5	858	Condo

Relationship Classes

- **What is a Relationship Class?**
  - An association between two object classes that is persisted within the Geodatabase
  - A class may participate in multiple relationship classes.



Types of Relationship Classes

- **Simple**
  - Related objects can exist independently of each other.
  - When an origin feature is deleted the Foreign key in the destination is set to Null.
- **Composite**
  - Related objects are dependent on the lifetime of the origin objects.
  - When an origin feature is deleted all of the related features in the destination are also deleted

Why use a Relationship Class?

- Stored in the Geodatabase
- Navigating
  - Identify related objects
- Editing
  - Enforces referential integrity
  - Facilitate editing with automatic updates
  - Related objects can message each other which can trigger specific behavior (cascade deletes, move to follow, custom)
  - Relationship rules define and control how objects relate
  - Relationship rules can be enforced through Validation

How to create a Relationship Class?

- In the Root level of the Geodatabase
  - IFeatureWorkspace.CreateRelationshipClass
- In a Feature Dataset
  - IRelationshipClassContainer.CreateRelationshipClass
- You set parameters to define the relationship
  - Origin and destination tables (Object Classes)
  - Primary and Foreign keys (Field names)
  - Cardinality (1:1, 1:M or M:N)
  - Type of Relationship (Simple or Composite)
  - Attributes, Messaging, labels, etc
- IRelClassSchemaEdit
  - Can only change labels, and relationship type

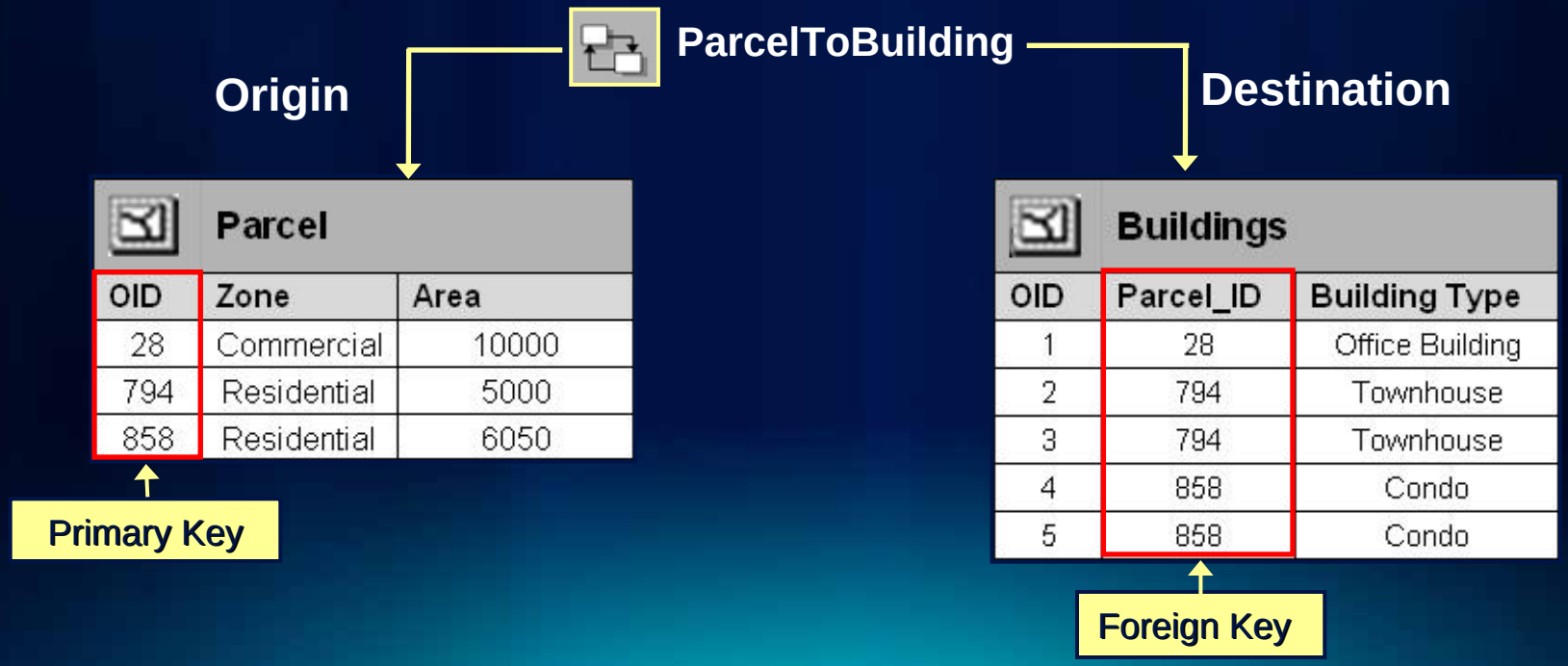
Relationship Class Messaging

- Used to notify related objects of changes
  - So further behavior can occur
- Does come at a cost
  - Edits and inserts to datasets that trigger notification is slower than the same operation on datasets that do not trigger any notification
- For inserts, ensure that all notified classes are opened prior to inserts
  - Failing to open the notified class may cause performance to degrade by an order of magnitude per class

Relationship Class Demo

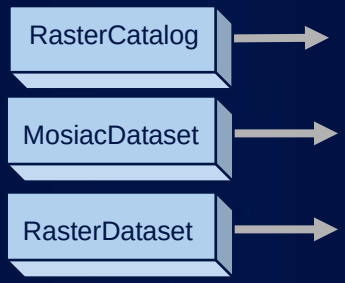
Relationship Rules:

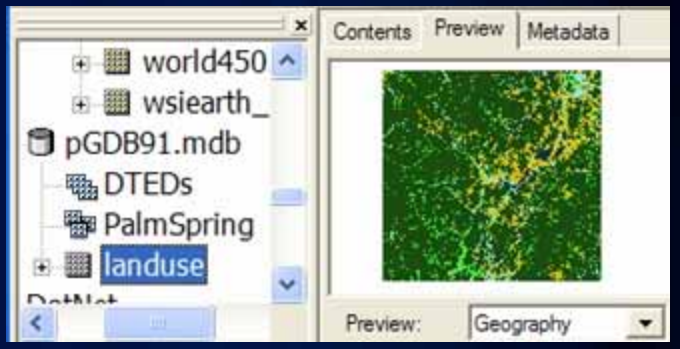
1. Wooden utility poles can support up to 3 transformers
2. Steel utility poles can support up to 5 transformers



Raster Data Model

- **Raster datasets**
  - **Geodata transforms**
  - **Function raster datasets and raster functions**





- **Raster catalogs and raster values**
- **Feature class with raster field**
  - **Raster as attribute**
  - **Example, parcels with pictures of houses**
- **Mosaic datasets**

Raster Dataset

- Single image in any supported format (including geodatabase formats)
  - Many raster formats (20+ new formats)
- Common interfaces to open
  - IRasterWorkspace, IRasterWorkspaceEx
- Common interfaces to access properties
  - IRasterDataset, IRasterBandCollection
  - …

Raster Catalog

- Special feature class with both shape and raster fields
- Store footprints and pixels in geodatabase
- Support referenced raster catalog in file GDB
- Display footprints vs. pixels
- Limit:
  - Display all images are slow as it does not have overviews
  - Does not handle sensor data and metadata well
  - Can not publish by ArcGIS Server as image service
- New solution: Mosaic datasets

Mosaic Dataset

- A new geodatabase raster data model for managing images
  - References data like an unmanaged raster catalog
  - Use like a raster dataset
- Support multiple sensor platforms (raster types)
  - Landsat/QuickBird/IKONOS/etc
- On-the-fly processing (raster function)
- Overviews at multiple levels for fast display (LOD)

Raster Demo

- Mosaic Datasets

Presentation Outline

- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- **Cursors and Queries**
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- File Geodatabase API
- Questions and other information

Cursors

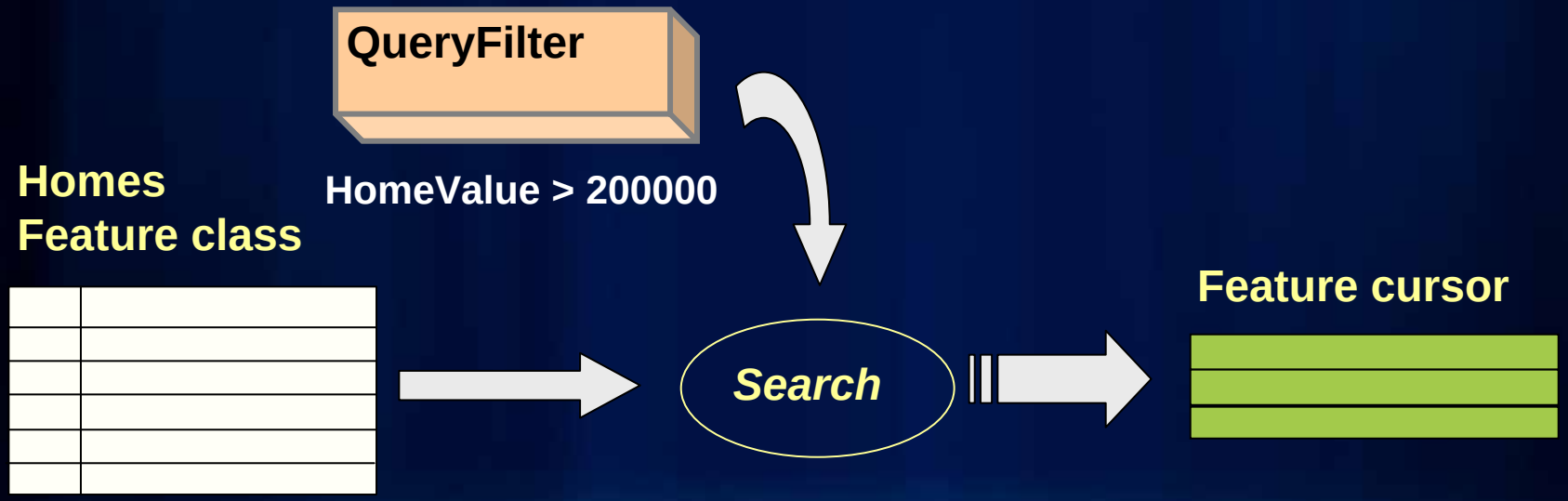
- A geodatabase object used for the iteration of records returned from a query
- 3 Class Cursors
  - Search (general query cursor)
  - Update (positioned update cursor)
  - Insert (bulk inserts)
- 1 QueryDef Cursor
  - Defined query (e.g. IQueryDef.Evaluate)
- What's the difference?
  - Rows created by Class cursors are bound to the class which created the cursor, rows created by a QueryDef cursor are not bound to a class

Class Cursors

- A table and a query return a cursor
- Used to:
  - Iterate over a set of rows in a table
  - Insert new rows into a table
- A cursor gives you access to one row at a time

Creating a Cursor

- QueryFilter contains a SQL-like statement
- The cursor contains a subset
  - No filter\nothing, all rows returned



IQueryFilter

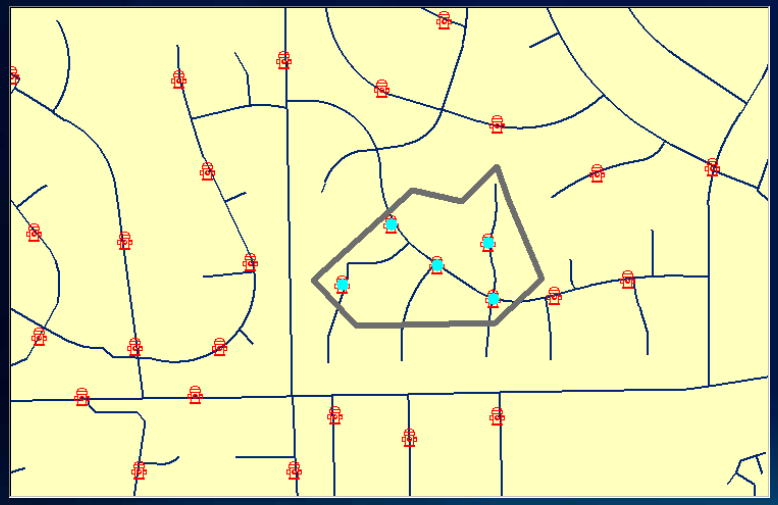
```
IQueryFilter queryFilter = new QueryFilterClass();
queryFilter.SubFields = "OBJECTID,FULLNAME,ParcelID";
queryFilter.WhereClause = “FULLNAME like ‘D%’";

IQueryFilterDefinition queryFilterDef =
(IQueryFilterDefinition)queryFilter;
queryFilterDef.PostFixClause = "ORDER BY FULLNAME";

IFeatureCursor featureCursor = featureClass.Search(queryFilter, true);
```

Creating a Cursor …

- SpatialFilter need a geometry and relationship
- Below the geometry is a polygon
- Below the spatial relationship is *contains*



Contains

Crosses

Intersects

Overlaps

Touches

Within

ISpatialFilter

- Used to query spatial aspects of a feature class
  - Inherits from IQueryFilter

```
ISpatialFilter spatialFilter = new spatialFilterClass();

spatialFilter.SubFields = “OBJECTID,FULLNAME,ParcelID,SHAPE”;
spatialFilter.Geometry = envelope;
spatialFilter.SpatialRel = within;
spatialFilter.WhereClause = “FULLNAME like ‘D%’”;

IFeatureCursor featureCursor = featureClass.Search(spatialFilter, true);
```

Types of Class Cursors

- Search cursors
  - Returns rows specified by a Query or Spatial Filter
- Update cursors
  - Update and delete rows specified by the filter
  - Specify the ObjectID field
- Insert cursors
  - Used for inserting rows into a table
- Accessed by
  - Corresponding methods on table or feature class

Types of Class Cursors …

- Forward only, do not support
  - Backing up and retrieving rows already retrieved
  - Making multiple passes
  - Resetting
- Solution:
  - Re-execute the query
- Release Class Cursors with
  - Marshal.ReleaseComObject
  - Cleaner.release()

Types of Class Cursors …

- Insert cursors are used to bulk insert rows
  - Faster for loading simple data than IFeature.Store
    - Bypasses events
    - IObjectClassInfo2 and IWorkspaceEditControl to override
  - Not Faster for non-simple data
    - Behavior, composite relationships, and notification
    - Need CreateRow and Store methods, so no performance gain
  - Use of Buffering is key
    - Pre-define attribute values
    - Buffers inserts on client, sends to database on Flush
- Flush – Call or not
  - Interval flushing: Check for room or handle errors
  - Careful: Insert cursors flush on destruction
    - No chance to detect errors

Types of Class Cursors …

- Scope cursors to edit operations
- Cursor is bound to a specific state of the geodatabase
- When state of the geodatabase changes cursor is no longer valid and should not be used
  - Performing edits on a cursor that is incorrectly scoped can cause unexpected behavior.

Recycling Method

- Recycling
  - A recycling cursor is a cursor that does not create a new client side row object for each row retrieved from the database
  - Allocate a single row object
    - Re-hydrate on each fetch
  - Performance advantages
  - Primarily used for reading data
- Non Recycling
  - A different row object on each fetch
  - Always has full set of fields, even if IQueryFilter::Subfields used

```
pCursor = theMeds.Update(pFilter,false)
```

Cursors - Efficient Use of FindField

- FindField is the API used to get a index value
- The Fields collection of a cursor created by a class is identical to the Fields collection of the class regardless of the SubFields specified in a QueryFilter
  - Index of the field is consistent
  - Not true for cursors created by IQueryDef.Evaluate
- Call FindField on the coarsest grain object with the matching Fields collection (i.e. class or cursor)
  - Avoid calls to FindField in a loop
  - use of Fields collection of a row and calling FindField is rare

Cursors - Efficient Use of FindField

```
public void EfficientExample(IFeatureWorkspace featureWorkspace)
{
  ITable testTable = featureWorkspace.OpenTable("Parcels");
  int parcelIdIndex = testTable.FindField("PARCEL_ID");
  int parcelKeyIndex = testTable.FindField("PARCEL_KEY");

  ICursor cursor1 = testTable.Search(null, true);
  IRow row1 = null;
  while ((row1 = cursor1.NextRow()) != null)
  {
    Console.WriteLine("PARCEL_ID = {0}", row1.get_Value(parcelIdIndex));
    Console.WriteLine("PARCEL_KEY = {0}", row1.get_Value(parcelKeyIndex));
  }

  ICursor cursor2 = testTable.Search(null, true);
  IRow row2 = null;
  while ((row2 = cursor2.NextRow()) != null)
  {
    Console.WriteLine("PARCEL_ID = {0}", row2.get_Value(parcelIdIndex));
    Console.WriteLine("PARCEL_KEY = {0}", row2.get_Value(parcelKeyIndex));
  }
}
```

Cursors - Inefficient Use of FindField

```
public void InefficientExample(IFeatureWorkspace featureWorkspace)
{
  ITable testTable = featureWorkspace.OpenTable("Parcels");

  ICursor cursor1 = testTable.Search(null, true);
  IRow row1 = null;
  while ((row1 = cursor1.NextRow()) != null)
  {
    Console.WriteLine("PARCEL_ID = {0}", row1.get_Value(testTable.FindField("PARCEL_ID")));
    Console.WriteLine("PARCEL_KEY = {0}", row1.get_Value(testTable.FindField("PARCEL_KEY")));
  }

  ICursor cursor2 = testTable.Search(null, true);
  IRow row2 = null;
  while ((row2 = cursor2.NextRow()) != null)
  {
    Console.WriteLine("PARCEL_ID = {0}", row2.get_Value(testTable.FindField("PARCEL_ID")));
    Console.WriteLine("PARCEL_KEY = {0}", row2.get_Value(testTable.FindField("PARCEL_KEY")));
  }
}
```

QueryDef Cursors

- Query based on one or more tables
  - Analogous to a SQL Query
  - Get a cursor back
  - Not bound to one class
- Tables must be in database
  - Do not have to be registered with the geodatabase
- Can result in a feature layer
  - Need to use IQueryName2 if no ObjectIDs in input tables
- Are also used to establish joins

QueryDef Cursors (IQueryDef)

- Simple QueryDef between 2 tables

```
//Create the query definition
IQueryDef queryDef = featureWorkspace.CreateQueryDef();

//Provide a list of tables to join
queryDef.Tables = "PoleFeature, TransformerFeature";

//Retrieve the fields from all tables
queryDef.SubFields = "bob.PoleFeature.TAG, bob.PoleFeature.SHAPE,
bob.TransformerFeature.Tag_Val";

//Set up the join based on the owner_name attribute
queryDef.WhereClause = "PoleFeature.TAG = TransformerFeature.Tag_Val";

ICursor cursor = queryDef.Evaluate();
IRow row = cursor.NextRow();
```

GetFeature vs GetFeatures

- Similar methods for getting features with ObjectIDs
  - GetFeature returns a single feature based on an ObjectID
  - GetFeatures returns a cursor with features specified in an integer array parameter
  - Methods are available on IGeodatabaseBridge for use in .Net and Java
- Use GetFeatures any time more than one feature is being retrieved using a known Object ID
  - Avoid looping over GetFeature
- GetFeatures outperforms GetFeature example on as few as two features
  - Difference will grow as more features are requested
  - With 1000 features GetFeature will often take up to 20 times as long

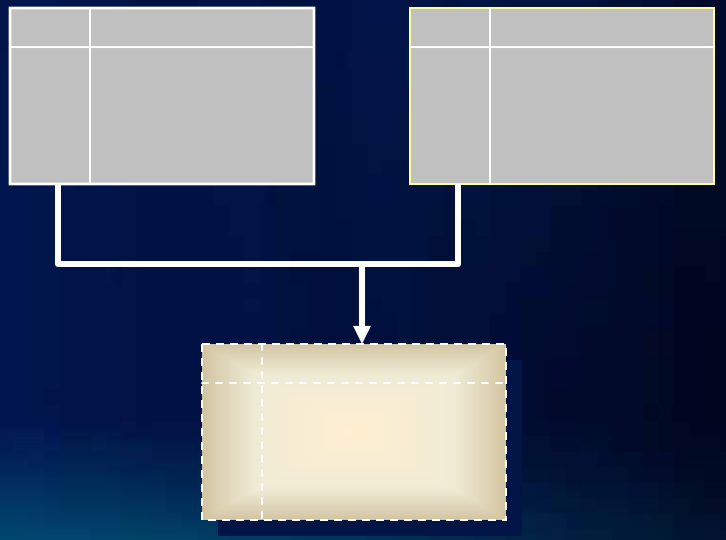
Cursor demo

- **Cursors examples**
  - **Search**
  - **Update**
  - **Insert**

Views and Joins

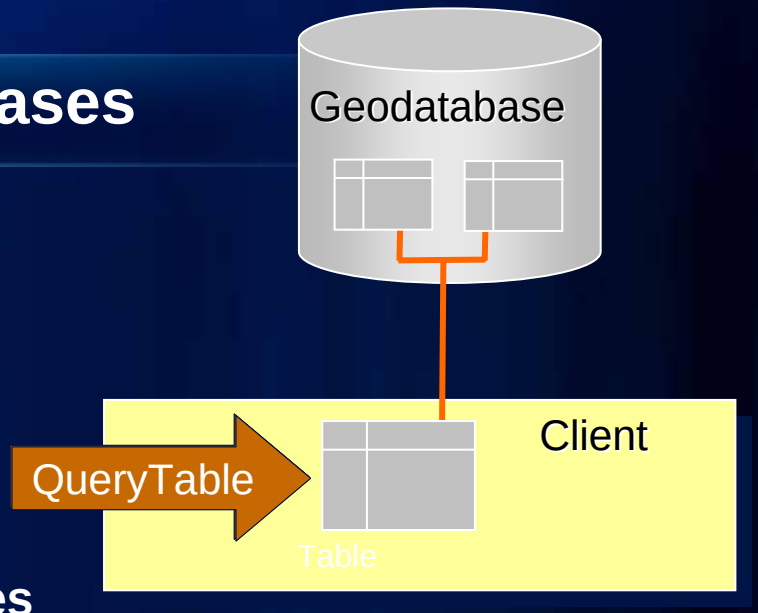
- Joining data involves appending the fields from one or more tables to another table
- Method used depends on the data sources as well as the cardinality of the data.

- Joining Data
  - TableQueryName
  - Query Layers
  - RelQueryTables



On-The-Fly Joins Across Databases

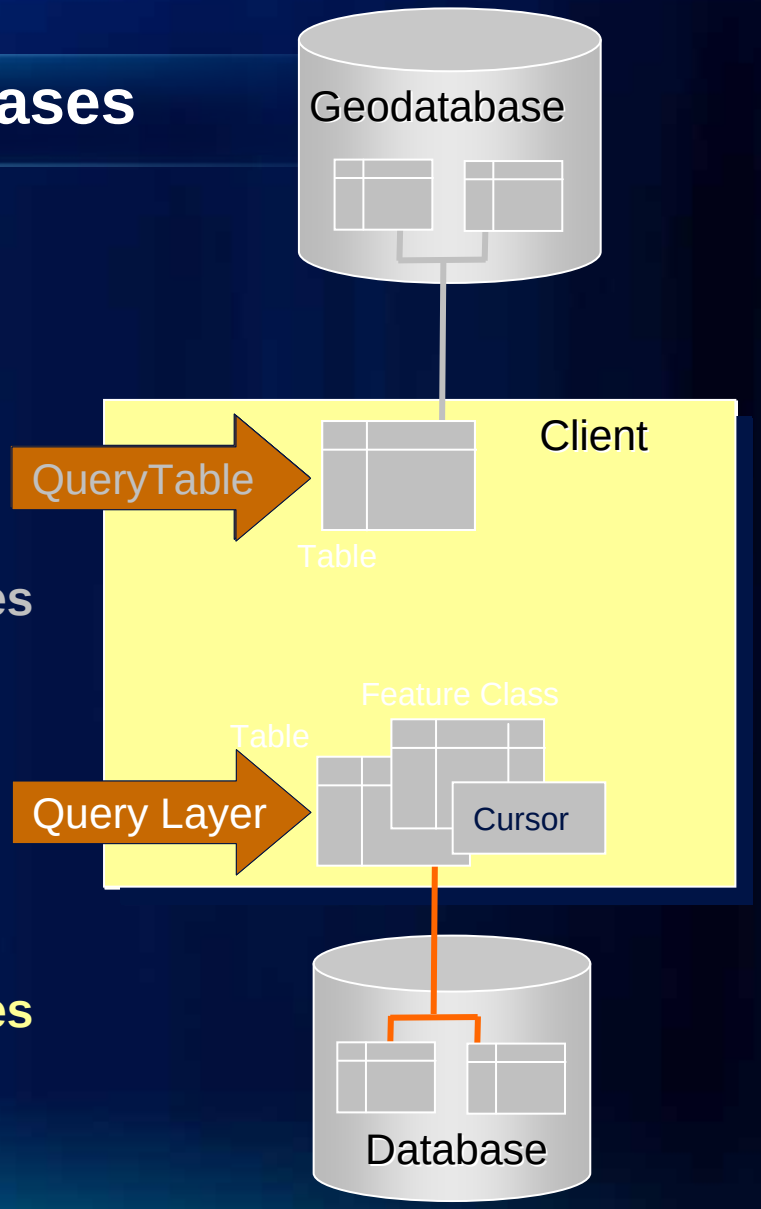
- **QueryTables (`TableQueryName`)**
  - **Tables in same datasource**
  - **Uses QueryDef object**
  - **Can be used with non-spatial tables**
  - **Supports versioning**



On-The-Fly Joins Across Databases

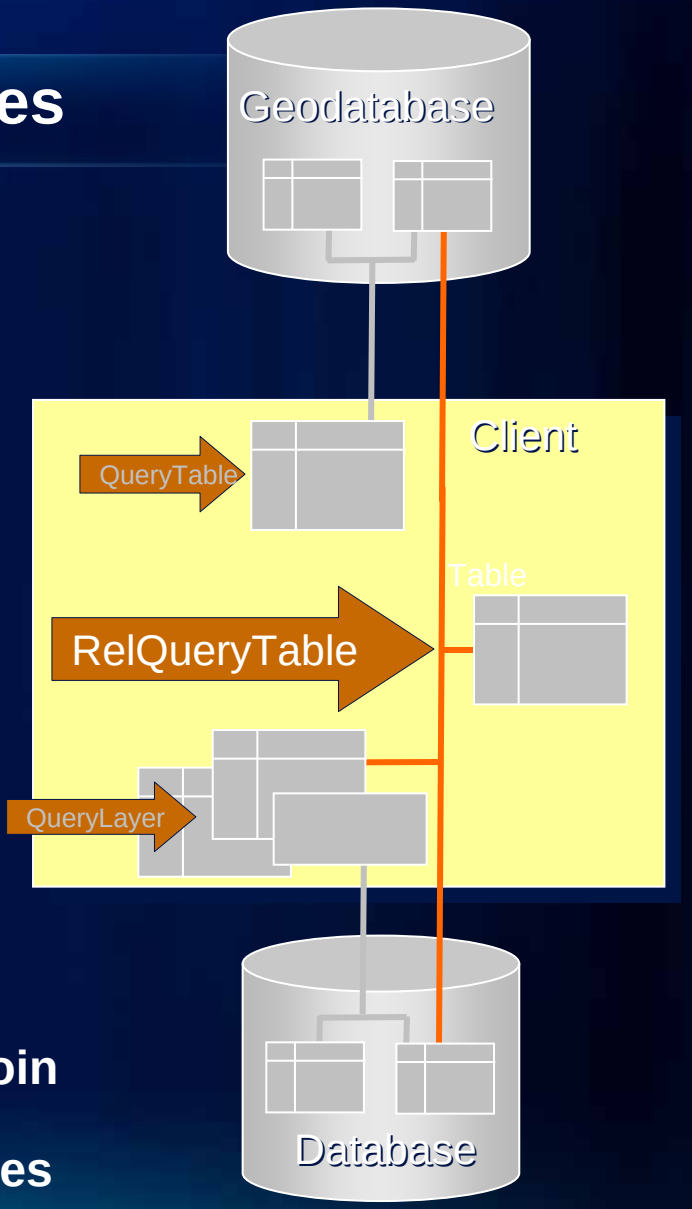
- QueryTables (`TableQueryName`)
  - Tables in same datasource
  - Uses QueryDef object
  - Can be used with non-spatial tables
  - Supports versioning

- Query Layers (`IQueryCursor`)
  - Passes through Native SQL
  - Can be used with non-spatial tables
  - Does not support versioning



On-The-Fly Joins Across Databases

- QueryTables (`TableQueryName`)
- Query Layers (`IQueryCursor`)
- **RelQueryTables (`IRelQueryTable`)**
  - **Tables in different datasources**
  - **Matches only first candidate on 1:M join**
  - **Uses in-memory or relationship classes**

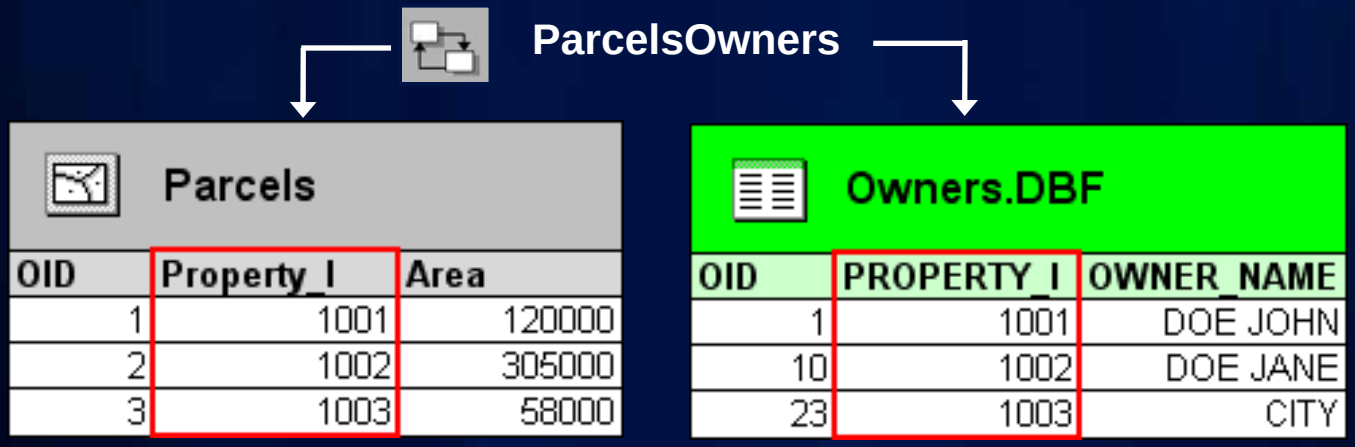


Which methodology to use?

- RelQueryTables
  - Joining across datasources
    - Geodatabase Feature Class to DBF file
    - FGDB Feature class to ArcSDE Geodatabase Table
    - …
- TableQueryName
  - Joins in File Geodatabase or Personal Geodatabase
  - Joins between versioned datasets in an ArcSDE Geodatabase
- Query Layers
  - All other cases
    - Joining non-versioned objects in a Enterprise Geodatabase
    - Joining tables in a database

Join Demo

- Need to join a feature class and a DBF file
  - To answer a question with information from both tables
  - Who are the owners for certain parcels?

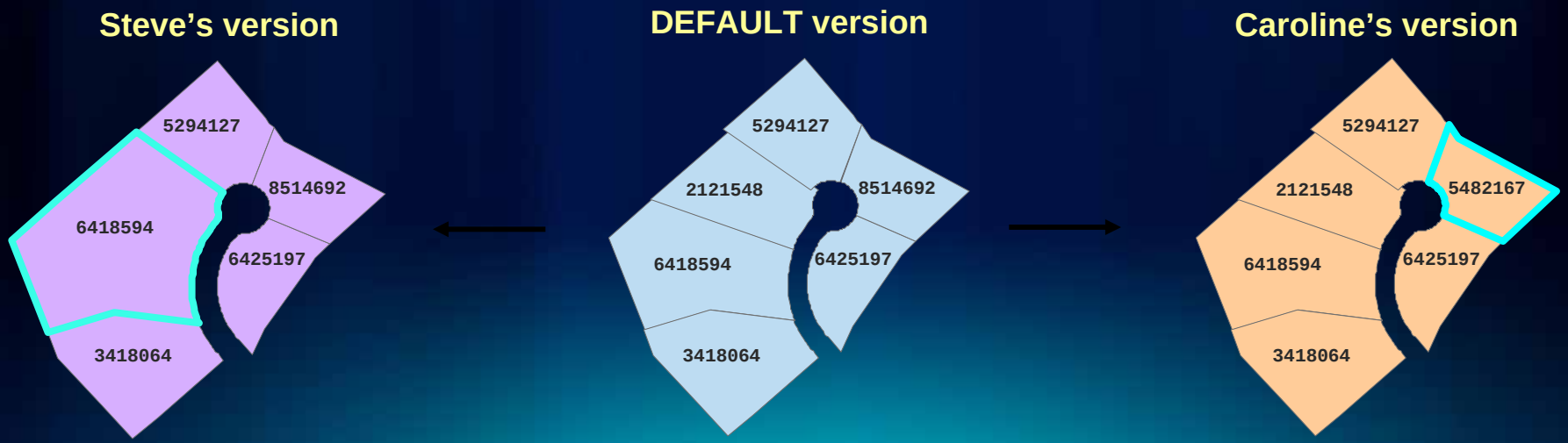


Presentation Outline

- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- **Versioning**
- Conversions and Loading
- Extending the Geodatabase
- File Geodatabase API
- Questions and other information

What is Versioning?

- Our solution to allow multi-user editing of geographic data
  - A version is just a state in the database.
- A method for presenting and tracking changes
  - Changes accessed through a version
  - Changes preserved in delta tables
- Includes mechanisms for reconciling versions
  - Provides tools to resolve conflicts



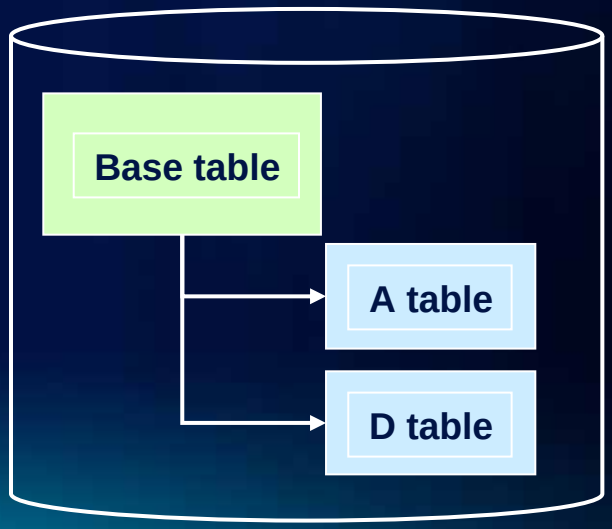
Versioned editing

- **Advantages include:**
  - **Isolate editor’s work over an extended period**
  - **Reconcile and resolve conflicts between edits**
  - **Implement workflow for business procedures**
  - **Perform geodatabase archiving and replication**
  - **Features not locked during editing**

Versioned editing

- Advantages include:
  - Isolate editor's work over an extended period
  - Reconcile and resolve conflicts between edits
  - Implement workflow for business procedures
  - Perform geodatabase archiving and replication
  - Features not locked during editing
- **Architecture:**
  - **Change to each feature class is preserved in A and D tables**
  - **Change accessed through a version**

Versioned editing:
Changes stored in A and D tables

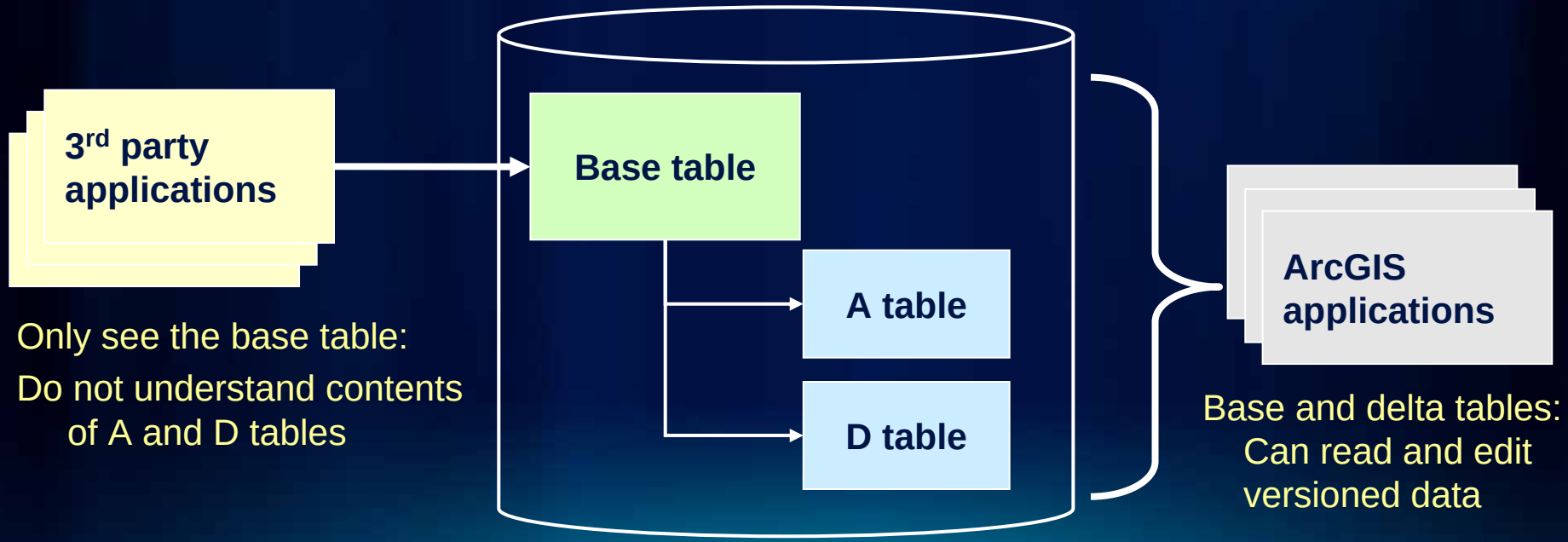


Versioned editing

- Once an editors work is complete
- Changes can be integrated into other versions
  - Through a mechanism called Reconcile and Post
  - Compares changes in your edited version with the version into which you want to merge the edits
- Identifies any features edited within both versions as a Conflict
- IVersionEdit::Reconcile4 (VersionName, acquireLock, abortIfConflicts, ChildWins, ColumnLevel)

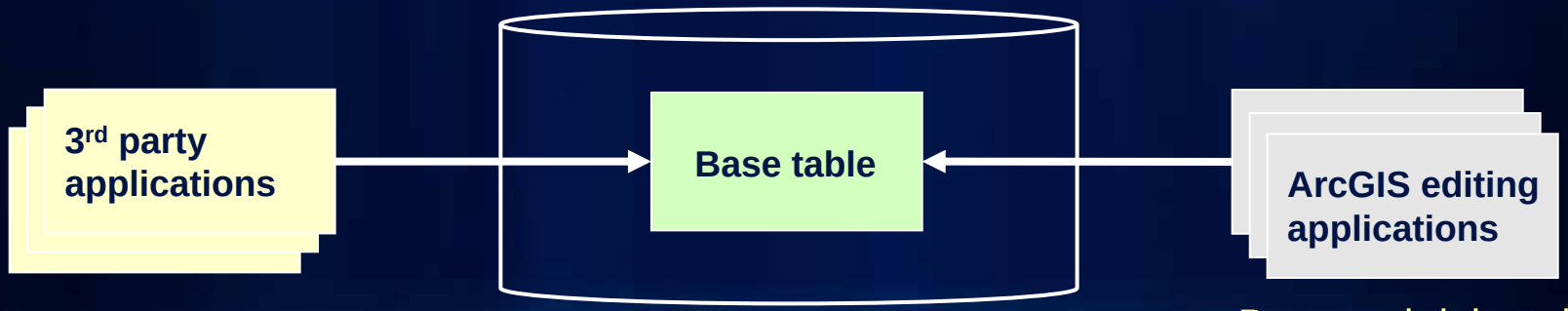
Versioned editing

- **Does not easily support:**
  - **Non-ESRI client access to versioned data**
    - **Edits are preserved in delta tables**
    - **Cannot see edits in the base table**
  - **DBMS behavior such as triggers and constraints**



Non-versioned editing

- Easy to implement
- No A and D tables
  - Edits immediately saved to base tables
- DBMS behavior is easy to implement
  - Uses the underlying database transaction model
- Easy IT integration
  - Non-ESRI applications see edits in base tables



Only see the base table:

Do not understand contents of A and D tables

Base and delta tables:
Can read and edit versioned data

Comparison of Editing Models

- Non-versioned editing provides
  - Direct editing using database transactions
  - Immediate visibility of edited information upon saving
  - Supported only on simple features and tables
    - points, lines, polygons, annotation, relationships
- Versioned editing provides
  - Long transactions, design versions, proposals
  - Geodatabase archiving
  - Geodatabase replication
  - Supported on all Geodatabase feature types and datasets

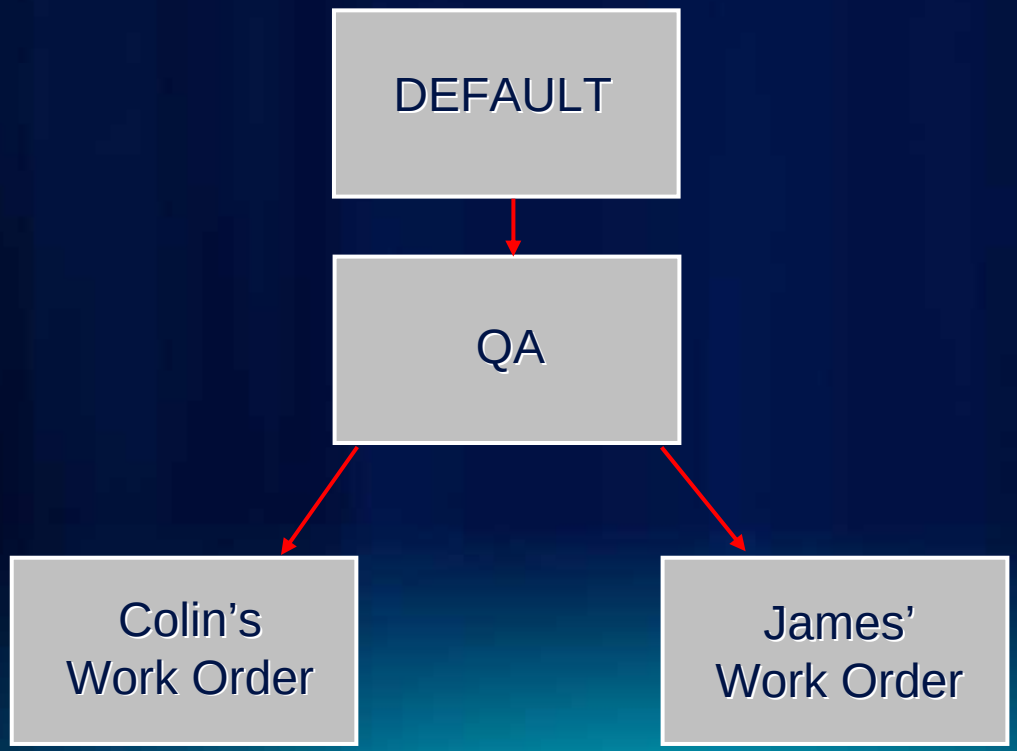
Comparison of Editing Models …

- IWorkspaceEdit vs IMultiUserWorkspaceEdit

- IWorkspaceEdit
  - Local geodatabase data (Personal, File) and ArcSDE versioned data
- IMultiUserWorkspaceEdit
  - Leveraged by non-versioned editing
  - ArcSDE data only
  - Can be used on both versioned and non-versioned data

Versioning Demo

- The QA version has two children, EditorA and EditorB
- EditorB has already edited a feature and posted the changes to the QA version
- EditorA's will attempt to reconcile and post



Presentation Outline

- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- **Conversions and Loading**
- Extending the Geodatabase
- File Geodatabase API
- Questions and other information

Conversion and Loading

- Loading into the geodatabase
  - IFeatureDataConverter
    - Used for converting simple features only
    - A lot of set-up required
    - Provides many options and a lot of control
  - Geoprocessing tools
    - Course grained
    - Less options available
    - Facilitates loading

Conversion and Loading

- **Between Geodatabases**
  - **IGeoDBDataTransfer**
    - **Supports simple and non-simple features**
    - **Works at Dataset level, not the workspace level**
  - **IGdbXmlImport and IGdbXmlExport**
    - **Supports simple and non-simple features**
    - **Workspace\Dataset level**
  - **Geoprocessing tools**
    - **FeatureClass to FeatureClass**
    - **Copy Features**
  - **Can also use methods outlined in previous slide**
    - **Not as optimal as methods outlined above**

Conversion and Loading

- IFeatureClassLoad::LoadOnlyMode
  - ArcSDE and File Geodatabases
- For a feature class or table in load-only mode:
  - Disable updating of spatial and attribute indexes (File Geodatabase only)
- Taking the feature class or table out of load-only mode rebuilds indexes
- Keep your use of LoadOnlyMode tight!
  - While in load-only mode, other applications cannot work with the data
  - Acquire an exclusive schema lock on the feature class prior to putting it into Load Only Mode

Presentation Outline

- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- **Extending the Geodatabase**
- File Geodatabase API
- Questions and other information

Level of Customization

- Application level
  - Pros
    - Business logic is stored within application
    - Can access data without customization
  - Cons
    - Only available when application is running
    - Users do not always interact with application customization

Level of Customization

- **Database level**
  - **Class / Workspace extensions**
  - **Pros**
    - **Business Logic is stored with data**
    - **Always available, regardless of application**
  - **Cons**
    - **One class extension per feature class**
    - **All users require dll to even view data**
    - **Database is unusable if code fails**
    - **Impacts on performance**

- **Use for important business rules that can be simply implemented without serious performance considerations.**

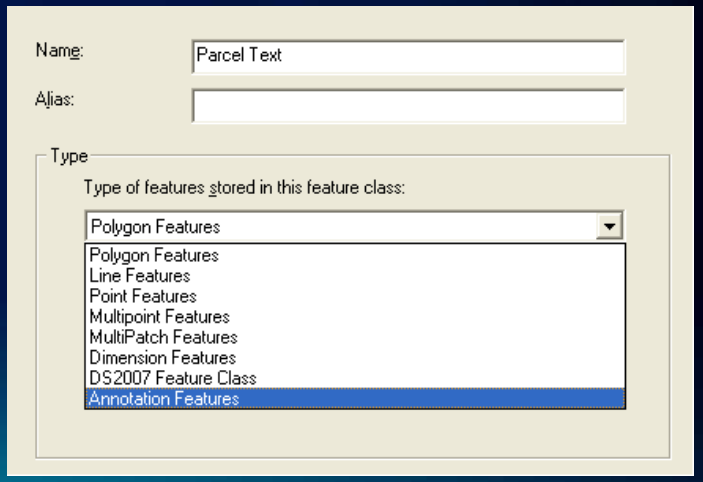
Level of Customization

- Custom features
  - Pros
    - Provide near total control over functionality.
  - Cons
    - Performance can suffer, since code is executed redundantly for every feature.
    - Handling of row and relationship events is less stable than class extensions.
    - Technically challenging to implement.
    - Only supported in C++
- Problem can often be solved by class extension or application customization

Examples of Customization

- Schema generation
- Custom drawing
- Custom property inspection and validation
- Custom split policies
- Related object creation notification
- PlugIn Data Sources
- Workspace logs

Your description here
(In ArcCatalog)



Presentation Outline

- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- **File Geodatabase API**
- Questions and other information

File Geodatabase API

- Provide a non-ArcObjects based means by which advanced developers can work with File Geodatabases
- C++ API with coarse grained access to File Geodatabase
- Will not replace ArcObjects as the recommended approach to interacting with the File Geodatabase

File Geodatabase API…

- Leveraging the work done with simplifying the Geodatabase
  - Will only support file geodatabases created with 10.0 and newer clients
  - No support for pre-10.0 file geodatabases
- Target audience
  - Advanced developers who require access to the File Geodatabase without an ArcObjects license for purposes of interoperability

Coarse-Grained Tasks possible with API

- Create, Open, Delete file geodatabases
- Read the schema of a geodatabase
  - All content within a geodatabase can be opened for read access
- Create schema for objects within the simple feature model:
  - Tables
  - Point, Line, Polygon feature classes
  - Feature datasets
  - Domains
  - Subtypes

Coarse-Grained Tasks possible with API…

- Read the contents of datasets in a geodatabase
  - The majority of dataset content within a geodatabase can be read
    - Some exceptions such as network indexes
- Insert, Delete and Edit the contents of simple datasets:
  - Tables
  - Point, Line, Polygon, Multipoint, Multipatch feature classes

Coarse-Grained Tasks possible with API…

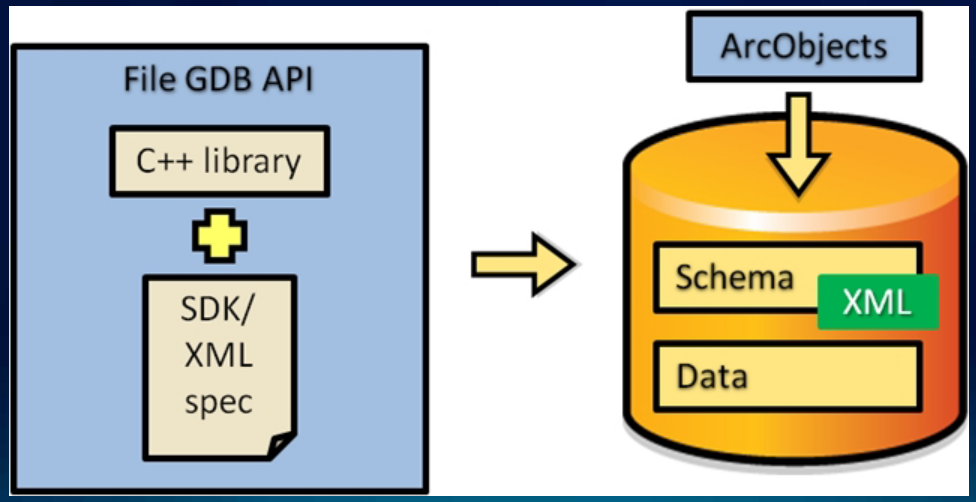
- Perform attribute and (limited) spatial queries on datasets
  - Spatial queries will be limited to the envelope-intersects operator
- Spatial References are limited to pre-defined GCS, PCS and Unknown
  - Custom coordinate systems are not supported

Coarse-Grained Tasks possible with API…

- Supports for a subset of the SQL 92 standard
  - e.g. Select statements, Order By, Joins

File Geodatabase API Overview

- Single downloadable ZIP file containing:
  - C++ library (single dll, lib, .h) built on Windows and Linux platforms
  - API documentation (html) and Samples
- Freely available from the Geodatabase Resource Center



Presentation Outline

- Introduction to the Geodatabase
- Accessing and Creating Data
- Editing
- Beyond Basics
- Cursors and Queries
- Versioning
- Conversions and Loading
- Extending the Geodatabase
- File Geodatabase API
- Questions and other information

Common Geodatabase API Programming Mistakes

- http://help.arcgis.com/en/sdk/10.0/arcobjects_net/conceptualhelp/index.html#/Geodatabase_API_best_practices/000100000047000000/
  - Recycling and Cursors
  - Overuse of FindField
  - Scoping cursors to edit operations
  - Performing DDL inside of edit sessions
  - Calling Store inside of Store-triggered events
  - GetFeature and GetFeatures
  - Careless reuse of variables
  - Inserts and relationship class notification
  - Modifying schema objects

Additional Resources

- ***Meet the Teams***
  - ***Tuesday at 6:00 – 7:30 pm***

- **ESRI Showcase**
  - **Monday, March 7, 11:00 a.m.–7:00 p.m.**
  - **Tuesday, March 8, 12:30 p.m.–6:00 p.m.**
  - **Wednesday, March 9, 10:00 a.m.–6:00 p.m.**

- ***ESRI Resource Centers***
  - **PPTs, code and video**