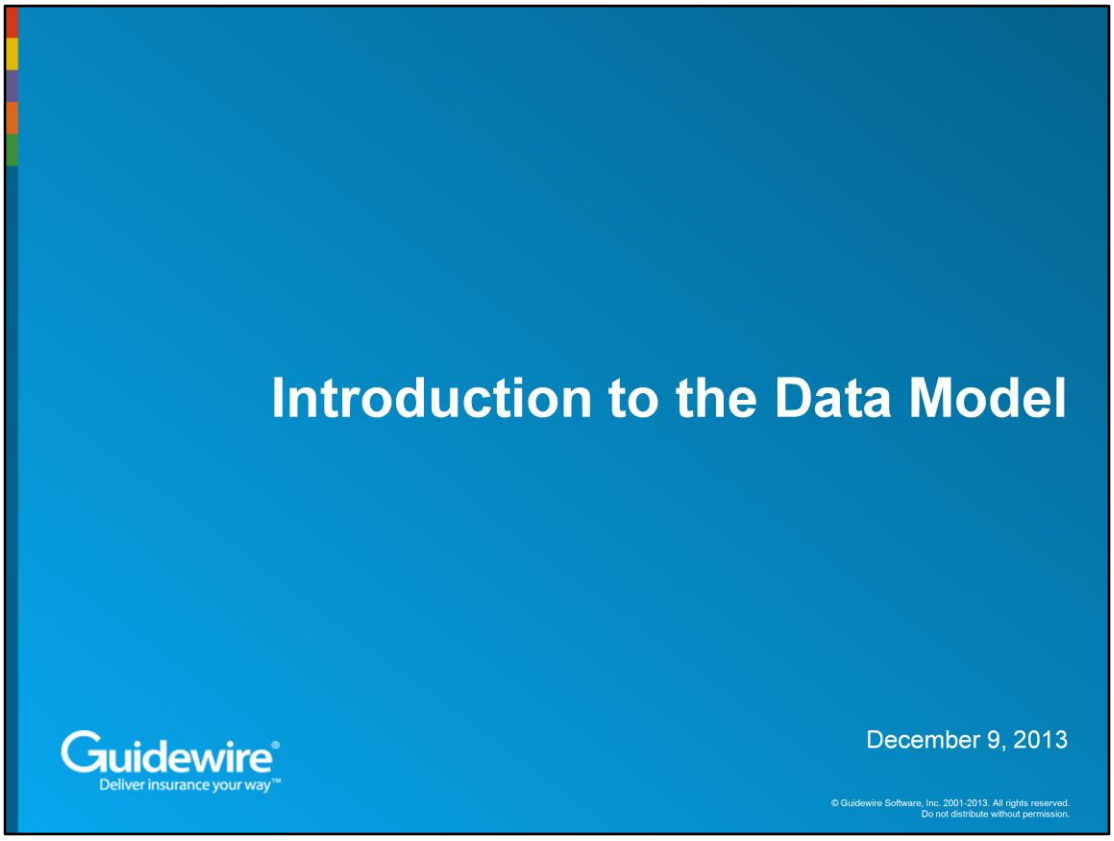


Guidewire training materials contain Guidewire proprietary information that is subject to confidentiality and non-disclosure agreements. You agree to use the information in this manual solely for the purpose of training to implement Guidewire software solutions. You also agree not to disclose the information in this manual to third parties or copy this manual without prior written consent from Guidewire. Guidewire training may be given only by Guidewire employees or certified Guidewire partners under the appropriate agreement with Guidewire.

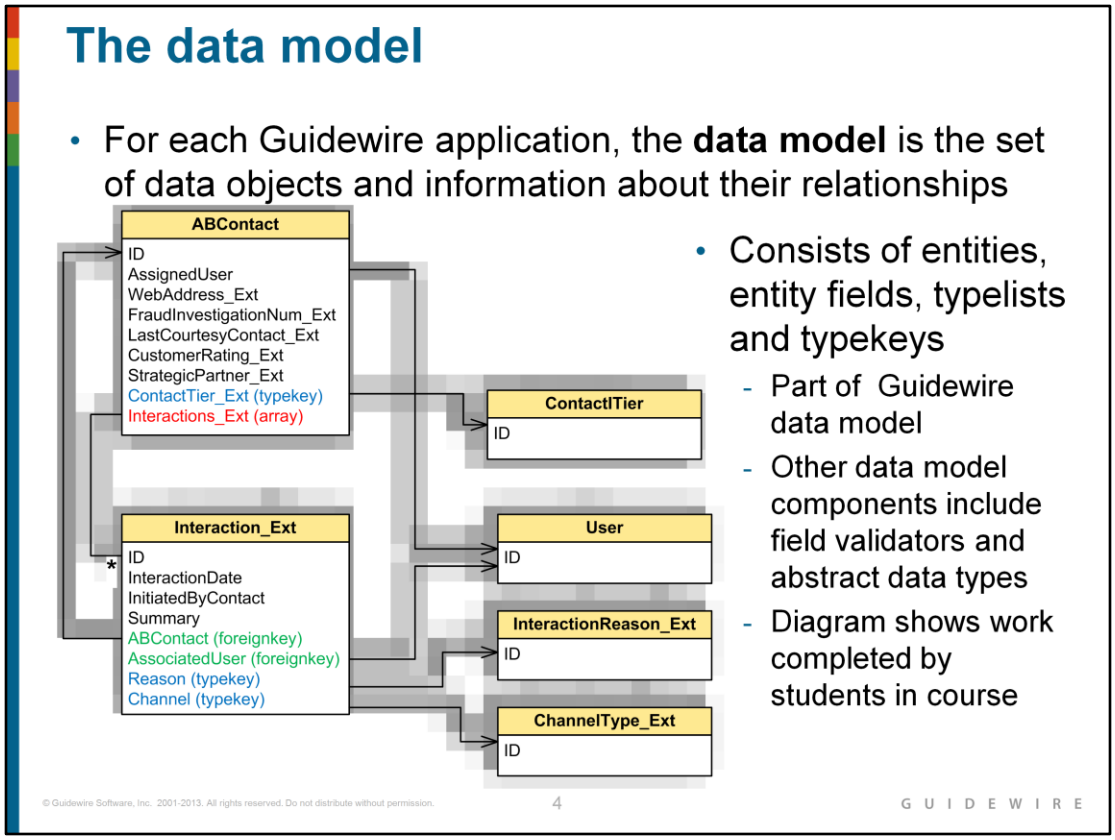
Lesson objectives

- By the end of this lesson, you should be able to:
  - Describe the contents of a Guidewire data model
  - Identify information about a given application's data model
  - Reference entity fields using dot notation

This lesson uses the notes section for additional explanation and information.
To view the notes in PowerPoint, select View → Normal or View → Notes Page.
When printing notes, select Note Pages and Print hidden slides.

Lesson outline

- Contents of the data model
- The Data Dictionary
- Objects and the data model



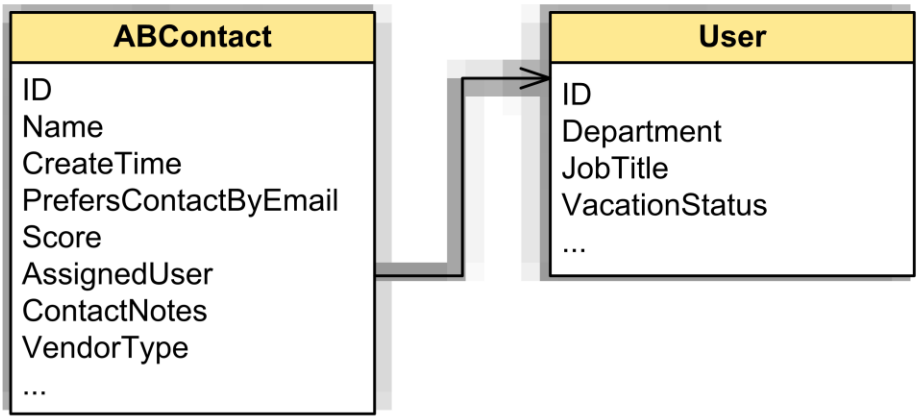
The diagram is a data model for an entity and related entities in TrainingApp that students will create over this course.

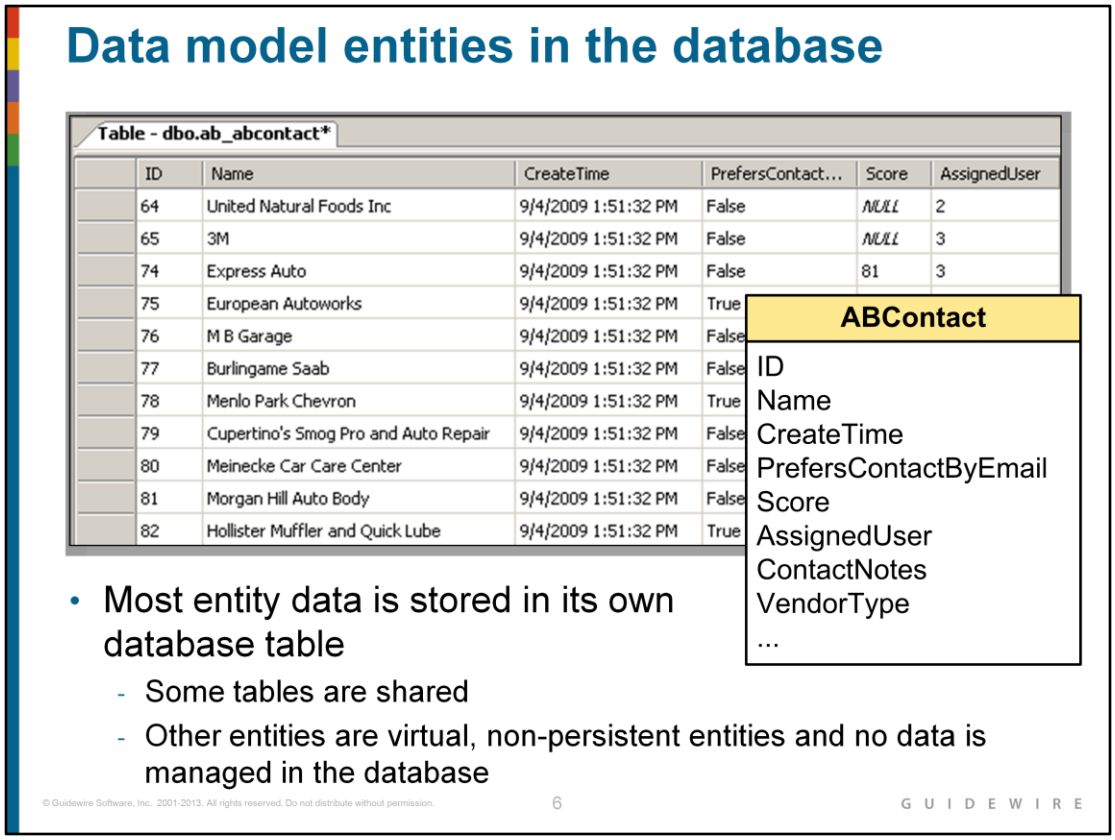
There are two other components to a Guidewire data model: field validators and abstract datatypes. A field validator defines the format requirements for specific fields, such as a postal code that must consist of 5 characters, each of which is a digit 0 through 9. An abstract data type defines a datatype that is built on top of a simple data type and is used to ensure the consistent definition of similar fields. For example, an ExchangeRate datatype could be defined as a decimal with a precision of 7 and a scale of 6. Then, any field that is intended to store exchange rate values could use this abstract data type, thereby ensuring that all exchange rate fields in the data model have been declared in the same way.

Field validators are discussed in this course. Abstract data types are not covered in detail. To learn more about abstract data types, please refer to documentation.

Data model entities

- A **data model entity** is an abstract definition of a group of business objects used by the application
- Examples: ABContact and User

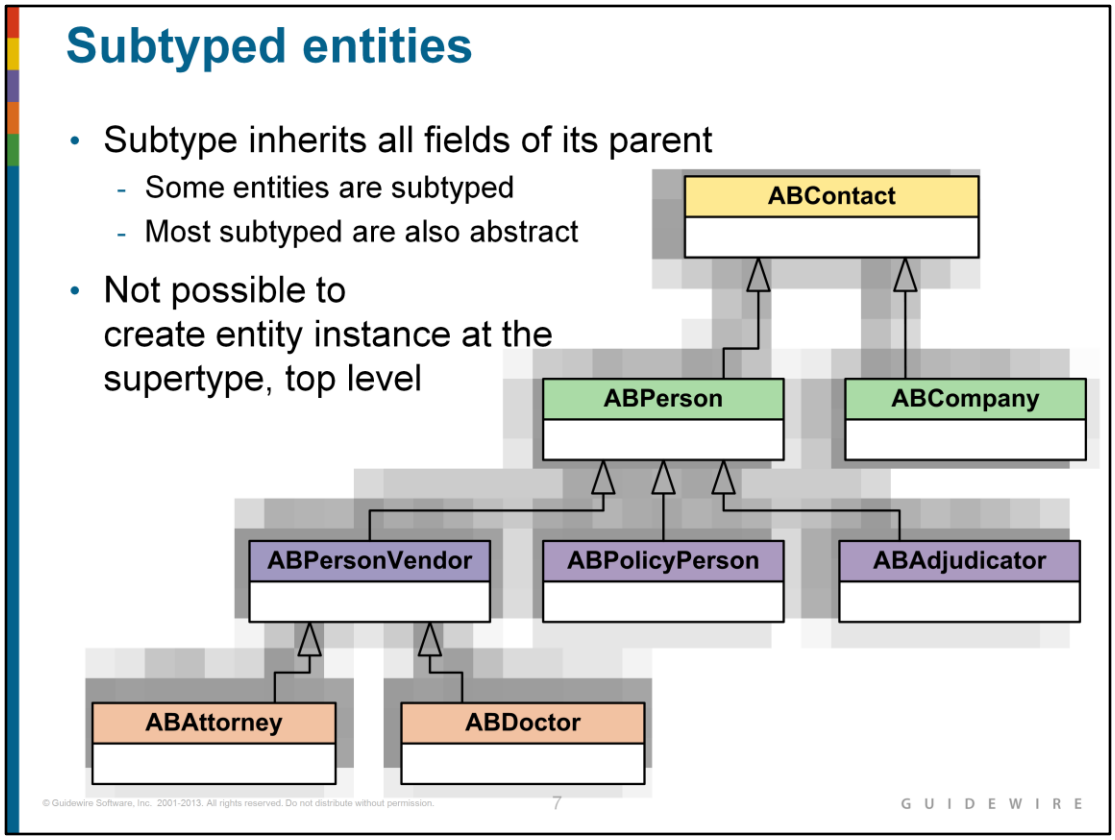




There are two exceptions to the generalization above: virtual and subtype entities.

Virtual entities are entities constructed entirely with code. They exist during run-time, but the data inside them is neither read directly from the database nor written directly to the database. One example of this is BillingCenter's ProducerCodeRoleEntry entity. It is a simple wrapper entity used to support modifying the producer codes by policy role. Its data is assembled from other physical entities.

Subtyped entities are entities that share a parent/child relationship. All fields in the parent entity are inherited by the child entity. A top-level subtyped entity and all of its child subtypes are stored in a single database table, as opposed to each having its own table.



The only major subtyped entity in TrainingApp is ABContact, which is abstract. This means that you cannot create instances of ABContact at the ABContact level, You can create instances at the levels below ABContact (such as instances of ABPerson or ABCompany or ABPlace).

- Major subtyped entities in ClaimCenter include: Contact, Incident, Transaction.
- Major subtyped entities in PolicyCenter include: Contact, Job, Modifier, and Workflow.
- Major subtyped entities in BillingCenter include: Contact, Activity, Plan, Invoice, and Workflow.

Subtype entity data

- Supertype parent table stores all instances of itself AND of subtype data
  - Contains parent fields and all subtype fields
  - Irrelevant fields are null for specific subtypes
  - Subtype column identifies subtype
- Example:
  - Firstname and lastname are always null for subtype = 3

SELECT ID, TAXID, NAME, FIRSTNAME, LASTNAME, VENDORTYPE, SUBTYPE FROM AB_ABCONTACT order by SUBTYPE;

ID	TAXID	NAME	FIRSTNAME	LASTNAME	VENDORTYPE	SUBTYPE
71	3432-32-431134-23-2343	*null*	James	Smythe	*null*	1
72	*null*	*null*	Paul	Peterson	*null*	1
67	2096542-3113-2456902	*null*	Lily	Watson	*null*	2
68	9983200-5335-0023899	*null*	James	Andersen	*null*	2
75	1242577-7777-7752421	Express Auto	*null*	*null*	*null*	3
76	3219251-8888-1529123	European Autoworks	*null*	*null*	*null*	3
77	5040392-9999-2930405	M B Garage	*null*	*null*	*null*	3
78	5647382-7777-2837465	Burlingame Saab	*null*	*null*	*null*	3
79	5646372-4444-2736465	Menlo Park Chevron	*null*	*null*	*null*	3

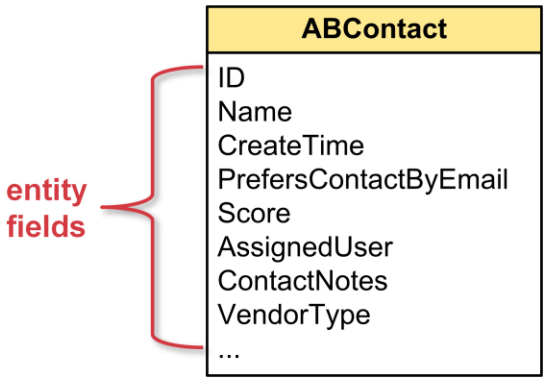
The ab_abcontact table is the parent table or supertype table that stores all instances of ABContact and its subtypes.

The ABContact table has columns for all fields defined at the parent level and therefore relevant to all child subtypes (such as TAXID). It also has columns for all fields defined at any of the subtype levels. This includes columns defined at the ABPerson level (such as FIRSTNAME and LASTNAME) and columns defined at the ABCompany level (such as NAME). Null values appear for columns not relevant to a given row's subtype.

A subtype column row value of 3 is for the subtype ABAutoRepairShop.

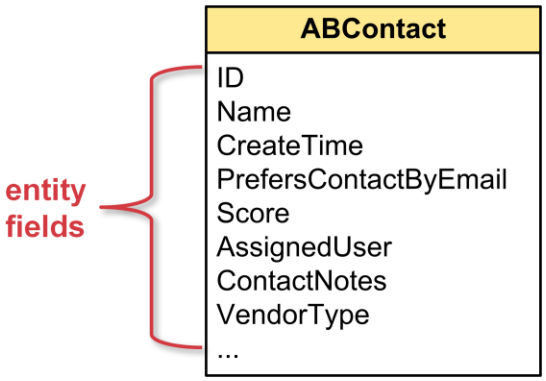
Entity fields

- An **entity field** is a value (or set of values) used to define the state or nature of a specific instance of the entity
  - Example: ABContact's Name field
- Four general types of entity fields
  - Data
  - Foreign key
  - Array key
  - Typekey

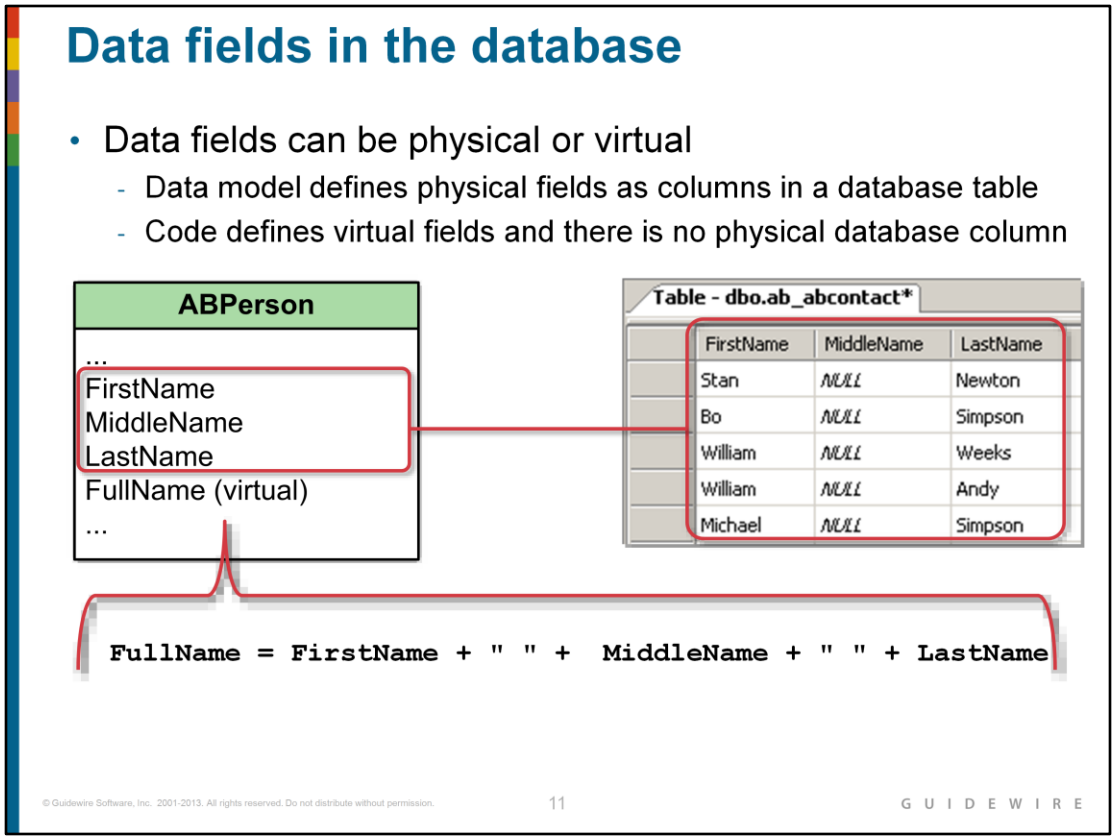


Data fields

- A **data field** stores a single value that does not reference any other object or table
- Examples of single values:
  - Name is a String
  - CreateTime is a datetime
  - PrefersContactByEmail is a bit
  - Score is an integer



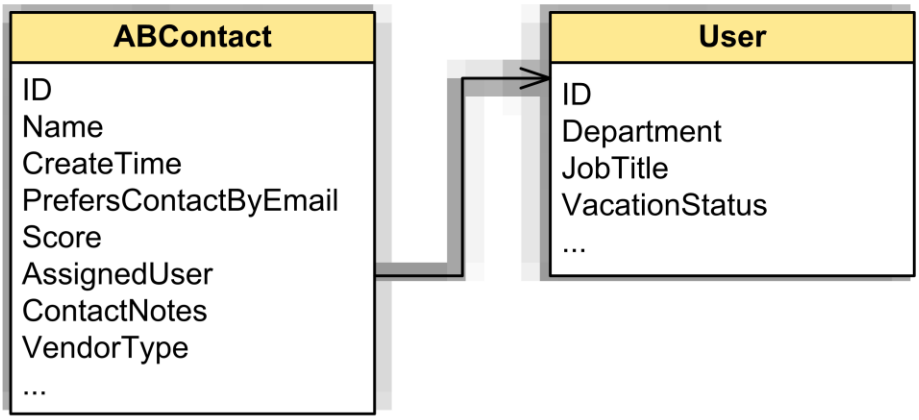
Data fields are defined in XML for an entity file with a <column> element. Therefore, they are sometimes referred to a "column fields". There are other types of fields beyond primitive value fields that are stored in database table columns, however. So one should not assume that the only fields that map to database columns are the "column fields". This course uses the term "data fields" to avoid this possible point of confusion.

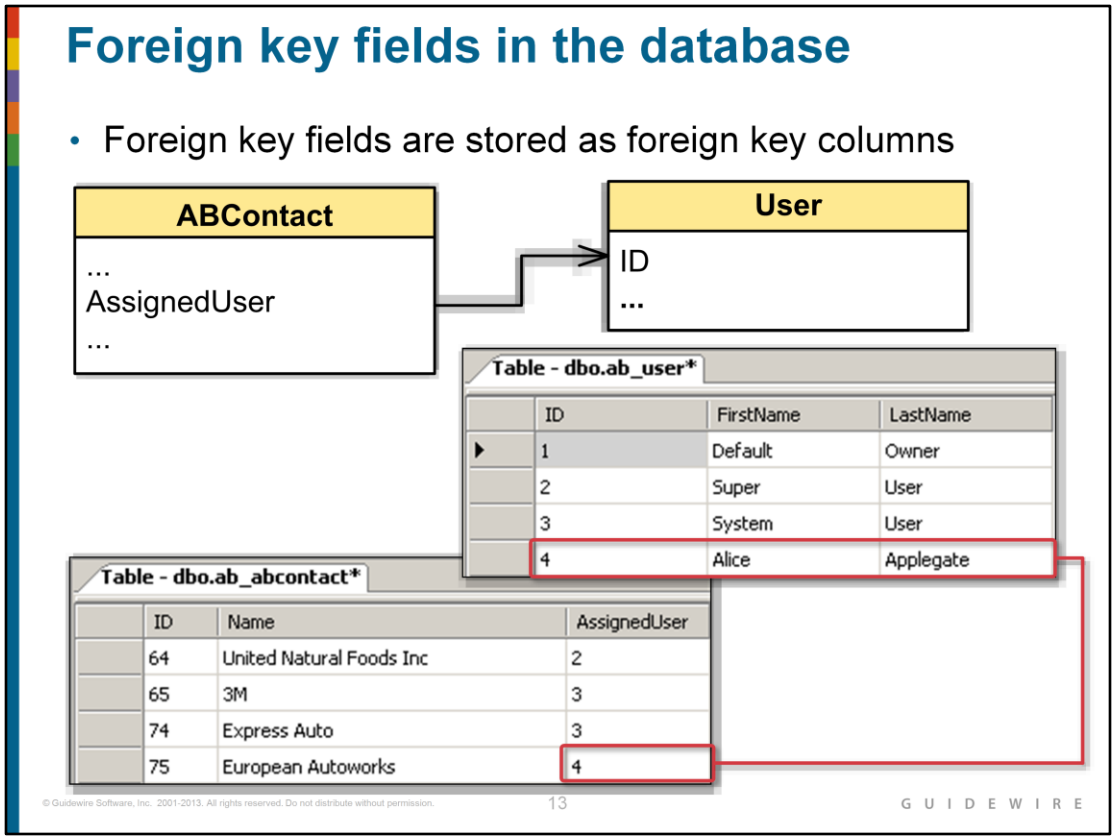


In the example above, the FullName field is a virtual field or property. FullName concatenates the FirstName, MiddleName, and LastName.
Entity enhancements and are discussed in another lesson.

Foreign key fields

- A **foreign key field** stores a reference to a related object in the data model
- AssignedUser in ABContact is the foreign key field for User

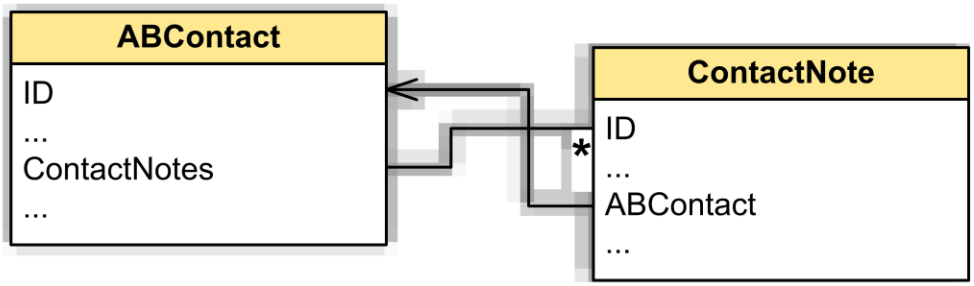




In the slide example, the assigned user (4) for European Autoworks is Alice Applegate. The assigned user (3) for 3M and Express Auto is System User. The assigned user (2) for United Natural Foods Inc is Super User.

Array key fields

- An array key field stores references to a set of related objects in the data model
- Not stored in database
- Populated at runtime by queries

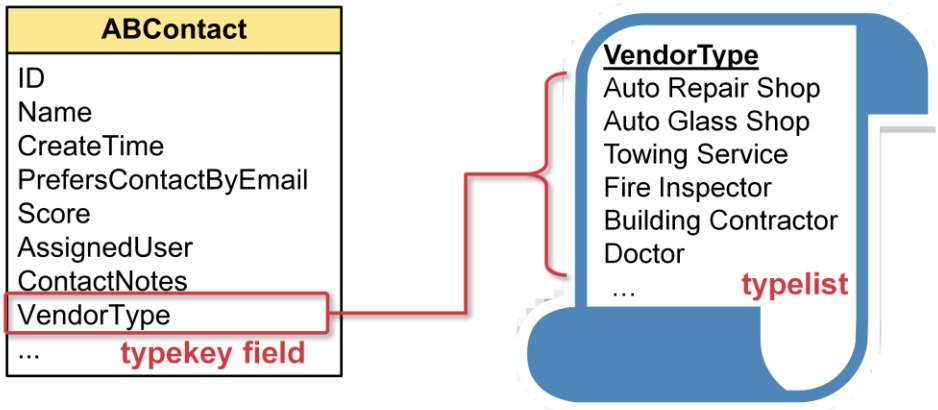


Array keys are managed during run-time by code that queries the database for objects of a given type with foreign keys that point to the array's parent.

For example, assume you have an ABContact object named ericAndy that stores the ABContact data for Eric Andy. Whenever you reference EricAndy.ContactNotes, Guidewire queries the database for all ContactNote objects whose ABContact field points to Eric Andy and returns those objects in an array.

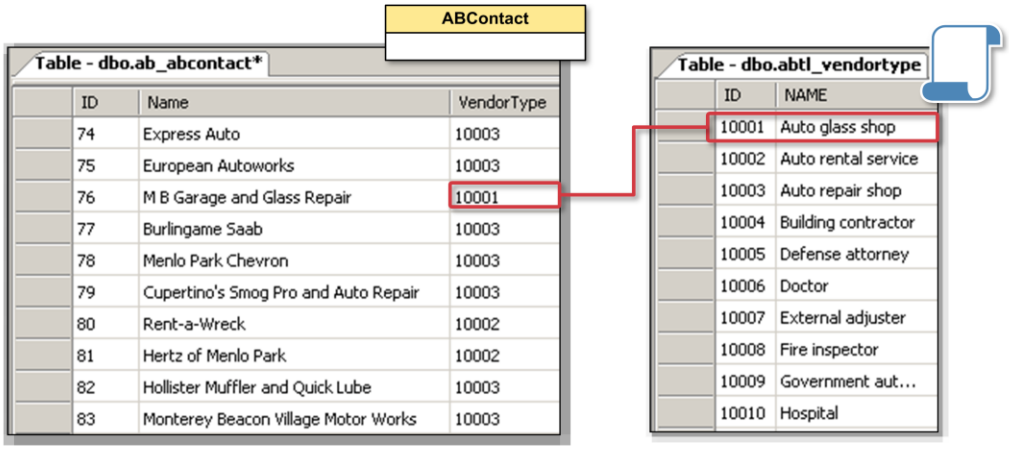
Typekey fields and typelists

- A **typelist** is a predefined list of values that constrains a field
- A **typekey** field is a field associated with a specific typelist
- The typelist defines the possible values of the typekey field

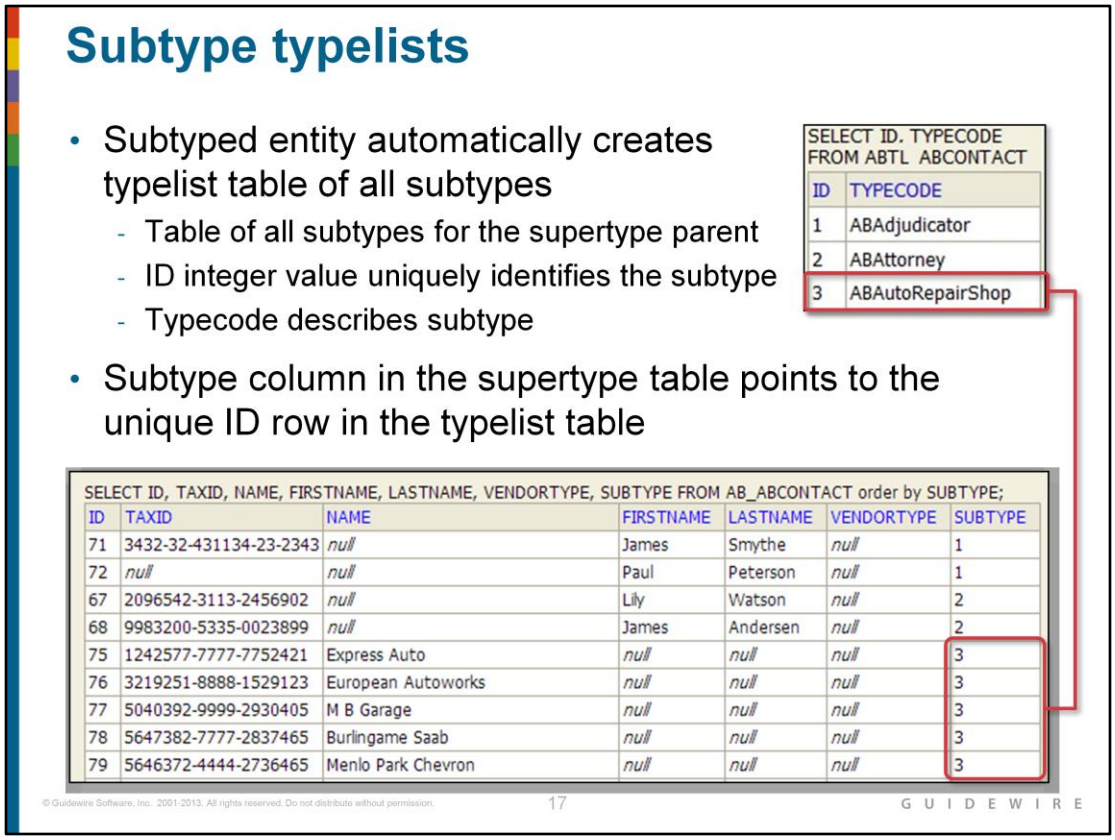


Typelists in the database

- Each typelist is stored in its own table
- Each typekey is a foreign key a single row in a typelist table



Typelist tables are named <applicationcode>tl_<typelistname>.



Whenever an entity is subtyped, a typelist is automatically created for it. This typelist contains one typecode for every subtype of the parent entity. Typelist tables are prefixed with "xxtl_", where "xx" is the two-letter application code.

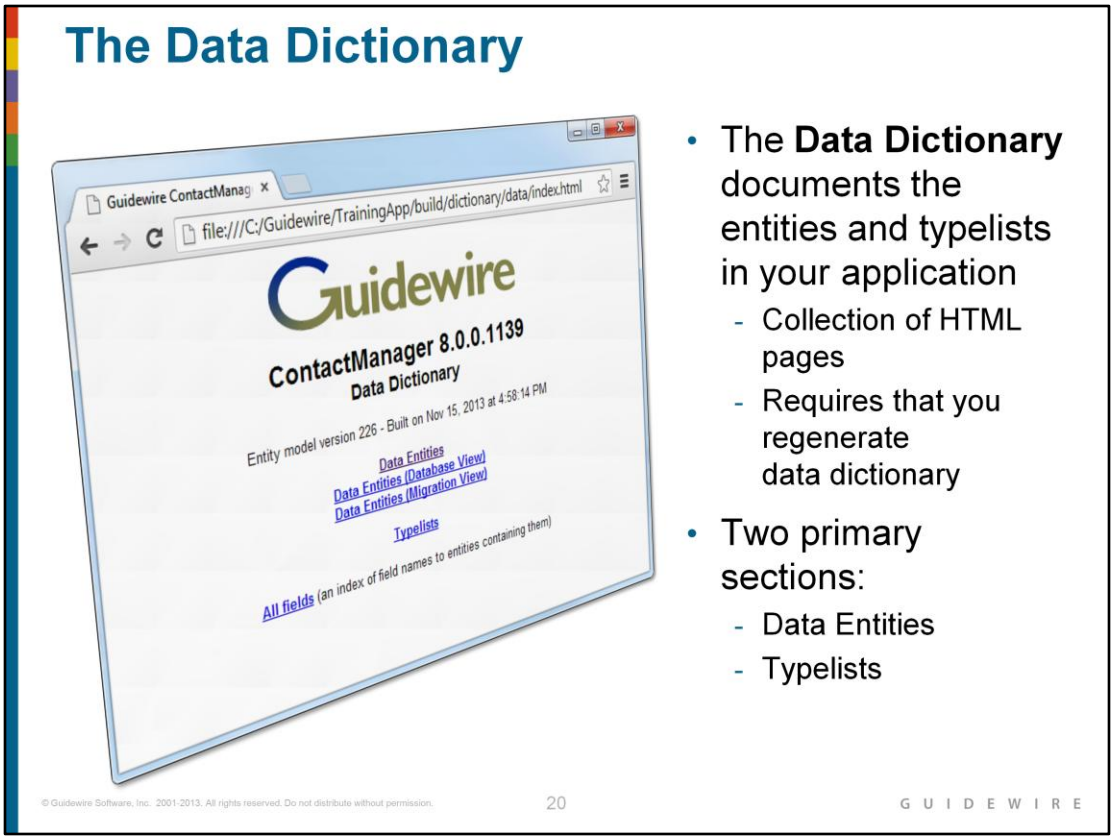
The ab_abcontact table is the parent or supertype table that stores all instances of ABContact and its subtypes.

Data model configuration tasks

- Creating new entities
- Extending existing entities
- Creating typelists
- Extending existing typelists
- Following lessons discuss data model configuration tasks

Lesson outline

- Contents of the data model
- The Data Dictionary
- Objects and the data model



The data dictionary is located at: <install directory>\build\dictionary\data\index.html. The entity pages of the Data Dictionary make use of active content.

Following are important notes regarding active content in different browsers:

If you are using Google Chrome, you will not be able to view active content (for example, clicking on the question marks will not show anything).

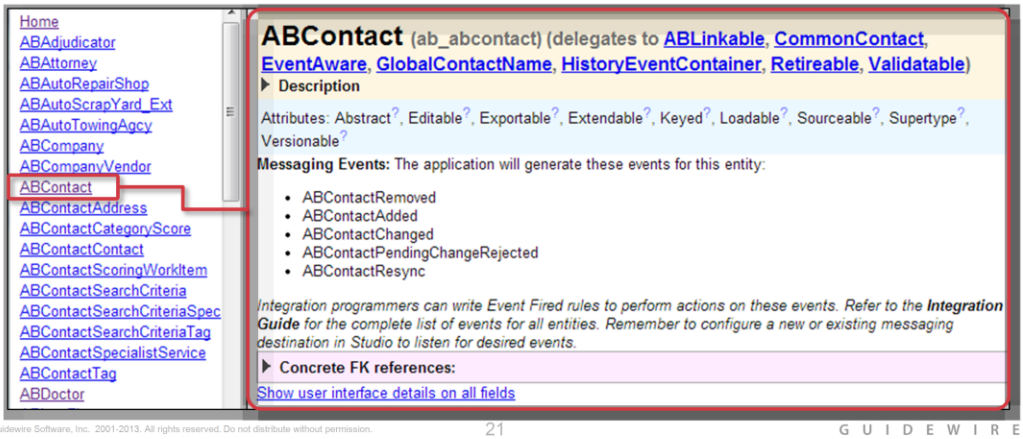
If you are using Internet Explorer (IE) - IE blocks active content and displays a warning bar that states "Internet Explorer has restricted this webpage from running scripts or ActiveX controls...". You can right-click the alert bar to allow the blocked content to run, but you have to do this every time you navigate to the Data Dictionary. Alternately, you can allow all active content in HTML pages on your machine to run automatically by doing the following:

- Select Tools > Internet Options from the menu bar.
- Click the Advanced tab.
- Scroll down to the Security section. Enable the "Allow active content to run in files on 'My Computer'" option.
- Restart Internet Explorer.

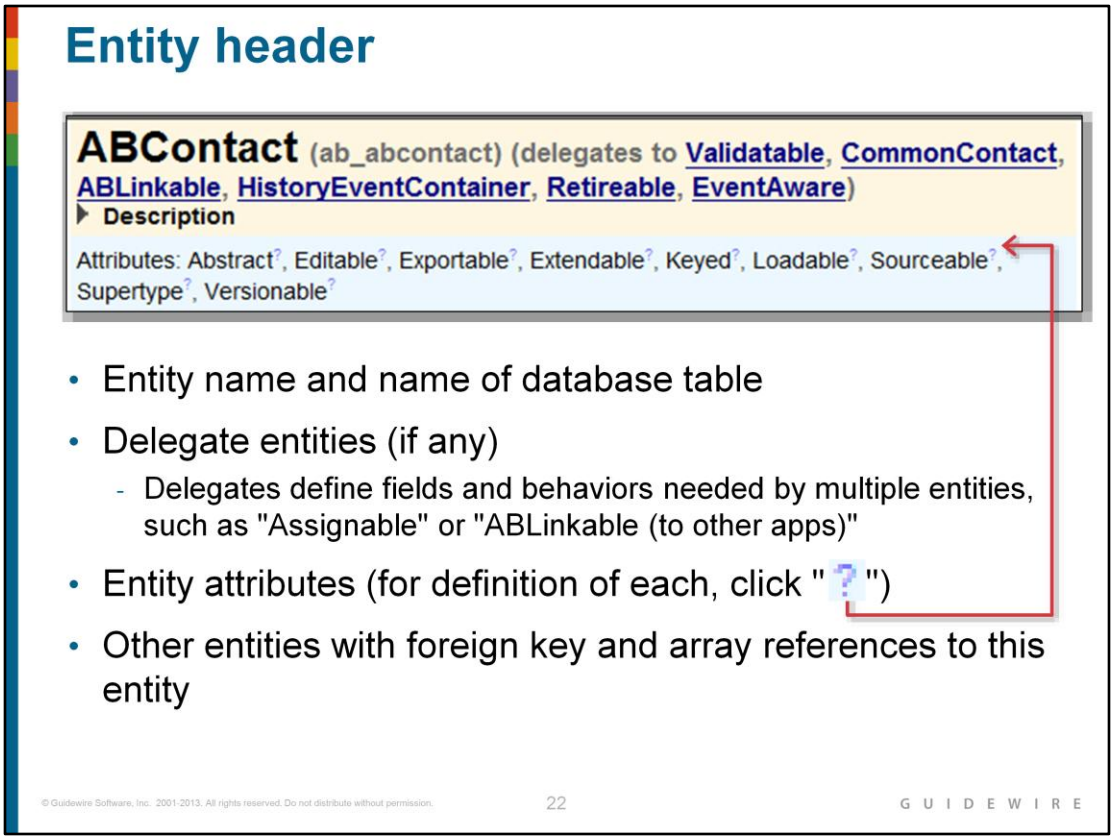
The Data Dictionary's conversion view is designed for data model work when converting data from a legacy application. This view provides a subset of the information in the standard view of the application entities that is more useful for those working on the conversion of legacy data. For more information on the conversion view, refer to the Configuration Guide for your product.

Entities in the Data Dictionary

- Entities section lists each entity and information about it
- To view information about given entity, click its name
- Hyperlink document is fully navigable



To view the Data Entities section of the data dictionary, click the "Data Entities" link on the Data Dictionary home page.



A delegate is an abstract entity that defines a set of fields and methods needed by other entities to enable a specific type of behavior. Any entity that needs the behavior "delegates to" the delegate entity. For example, ClaimCenter has four entities that can be assigned to users: Claim, Exposure, Activity, and Matter. It also has an "Assignable" delegate entity that contains the fields needed for assignment (such as AssignedUser and AssignedGroup) and methods. The Claim, Exposure, Activity, and Matter entities all delegate to this entity and therefore all have these fields and methods.

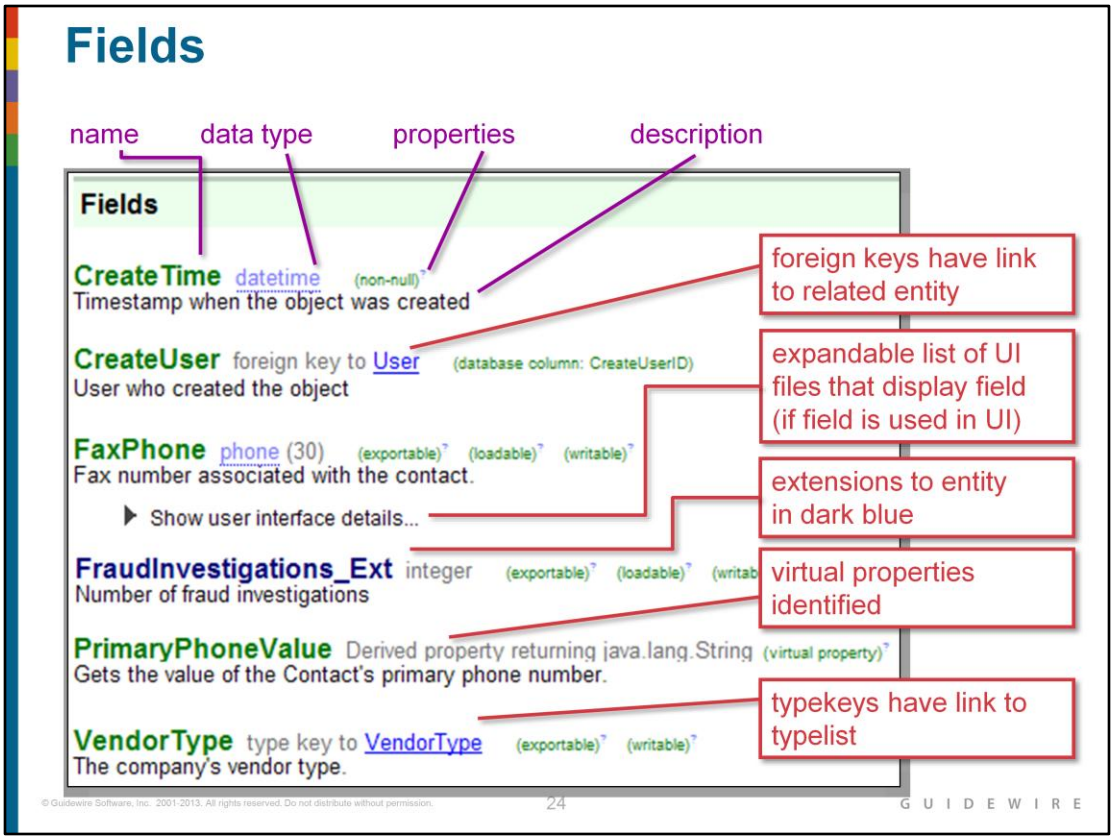
The Data Dictionary lists delegates using the term "delegates to". However, when you configure one entity to reference another entity as a delegate, the keyword that you use is "implements". Therefore, the relationship between an entity and its delegate entities can also be described with "implements", as in "ABContact implements CommonContact and ABLinkable".

Subtypes list

Subtypes

Code	Name
ABCompany	ABCompany
ABCompanyVendor	ABCompanyVendor
ABAutoRepairShop	ABAutoRepairShop
ABAutoScrapYard_Ext	ABAutoScrapYard_Ext
ABAutoTowingAgcy	ABAutoTowingAgcy
ABLawFirm	ABLawFirm
ABMedicalCareOrg	ABMedicalCareOrg
ABPolicyCompany	ABPolicyCompany
ABPerson	ABPerson
ABAdjudicator	ABAdjudicator
ABPersonVendor	ABPersonVendor
ABAttorney	ABAttorney
ABDoctor	ABDoctor
ABPropInspector_Ext	ABPropInspector_Ext
ABPolicyPerson	ABPolicyPerson
ABUserContact	ABUserContact
ABPlace	ABPlace
ABLegalVenue	ABLegalVenue

- If entity is subtyped, it has subtypes list
- Identifies all child subtypes
- Includes links to each subtype



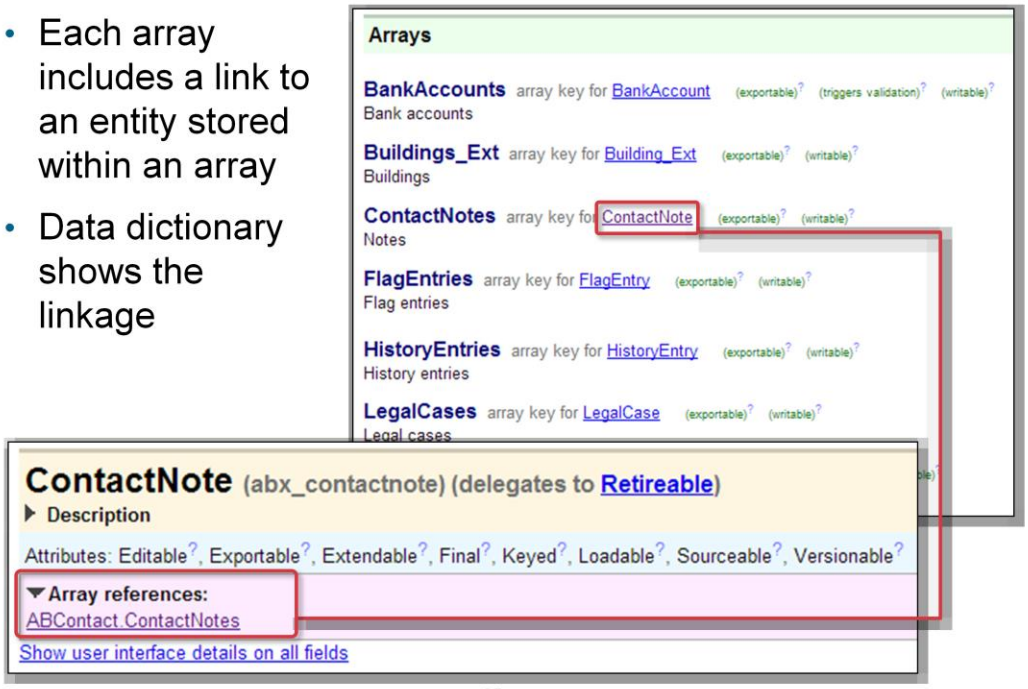
Both foreign keys and typekeys include links to the related entity or typelist.

If a property is virtual, then its data type is listed as "Derived property returning <data type>" and it has the "virtual property" property. In the example above, PrimaryPhoneValue returns the home phone number if the contact's primary phone is set to "home", returns the work phone number if the contact's primary phone is set to "work", and returns the fax phone number if the contact's primary phone is set to "fax".

Virtual fields are displayed in green, whether they are extensions or part of the base application.

Arrays

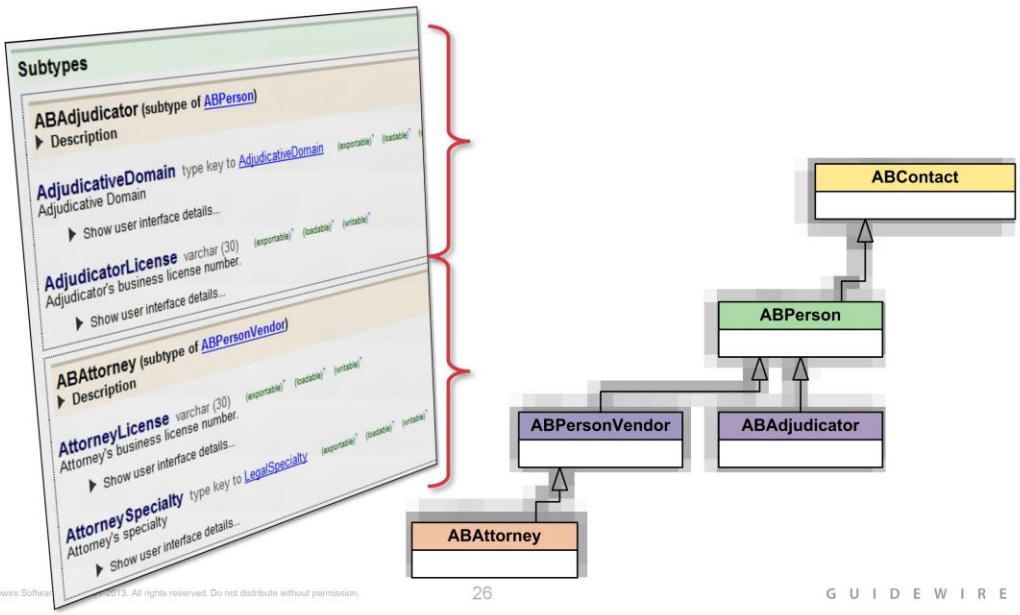
- Each array includes a link to an entity stored within an array
- Data dictionary shows the linkage



GUIDEWIRE

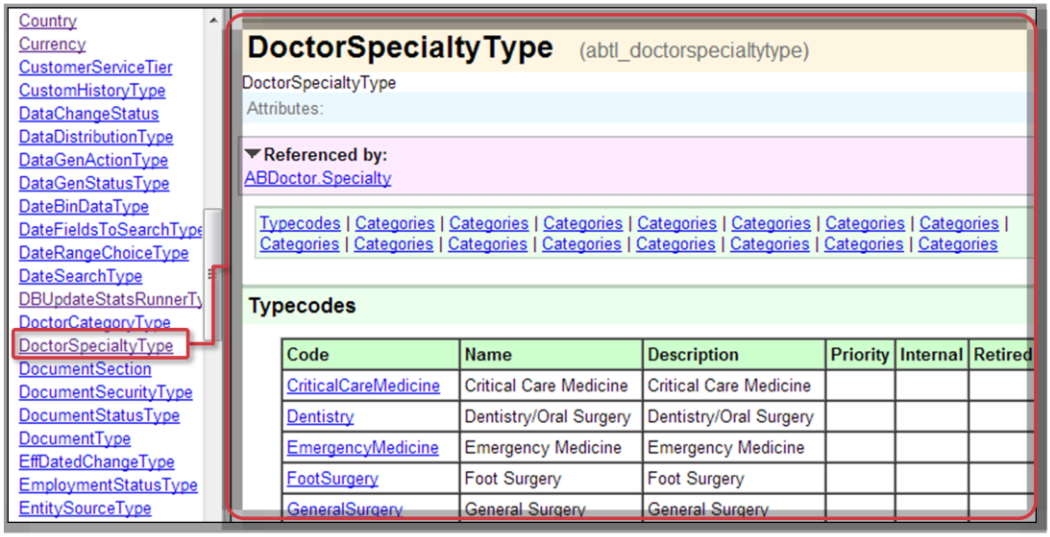
Subtype fields

- If entity is subtyped, fields and arrays declared at each subtype level are listed in their own section



Typelists in the Data Dictionary

- Typelists section lists each typelist and its typecodes
  - To view information about given typelist, click its name



To view the Typelists section of the data dictionary, click the Typelists link on the Data Dictionary home page.

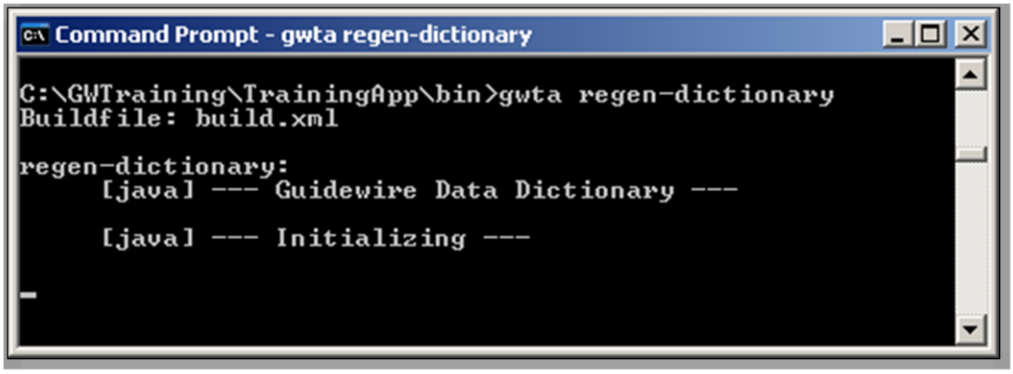
You can view the typecode and name for all elements in the list. You can also access typelists from the main page or from a typekey field.

In the example in the screenshot on the right:

- There is an "Attributes" field on the User entity.
- There is an "AttributeType" field on the AttributeCriteriaElement entity.
- There is an "AttributeValue" field on the AttributeCriteriaElement entity.
- There is an "Author" field on the Document, DocumentSearchCriteria, Note, and NoteSearchCriteria entities.

...and so on.

Generating the data dictionary



- You can (re)generate Data Dictionary:
  - After initial install (Data Dictionary is not pre-generated)
  - Any time you extend data model
- To regenerate dictionary, from bin directory, execute:
  - **gwXX regen-dictionary**
  - where **XX** is application's two-letter code

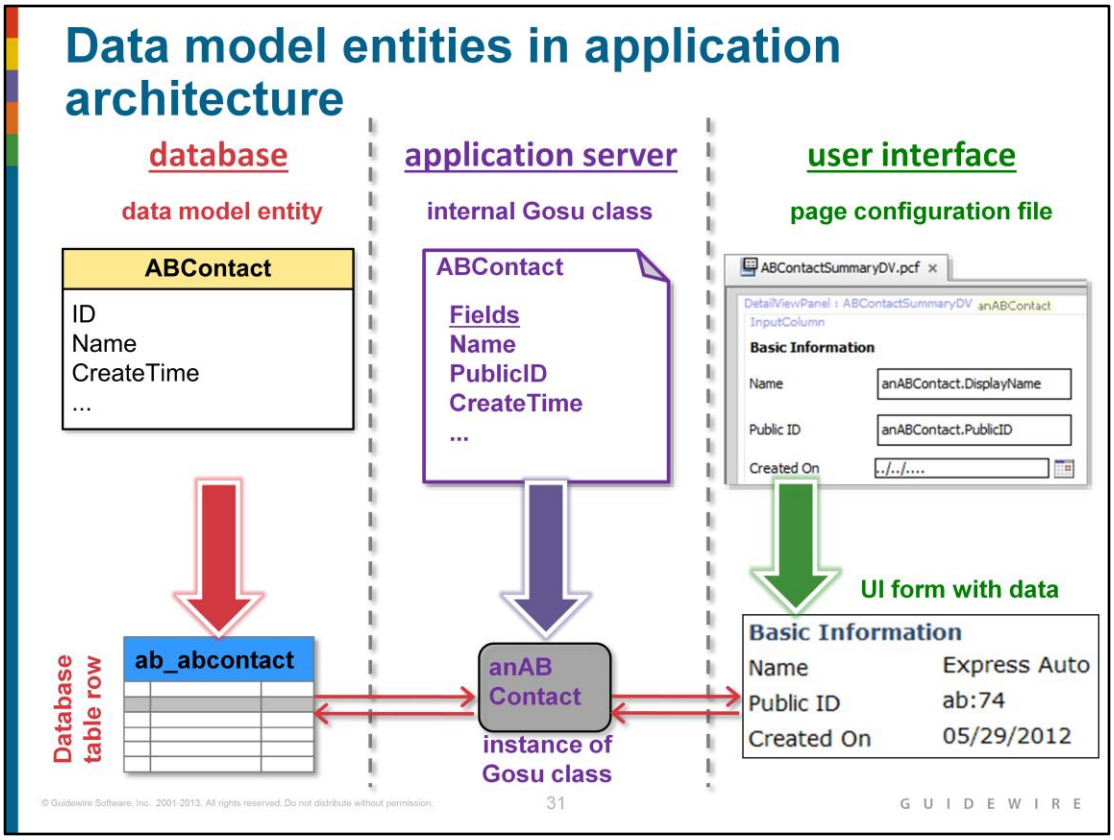
The codes for each application are:

- TA for TrainingApp
- CC for ClaimCenter
- PC for PolicyCenter
- BC for BillingCenter
- AB (address book) for ContactManager

The regen-dictionary command also regenerates the Security Dictionary. This dictionary lists all system permissions and other information that pertains to controlling access to a Guidewire application.

Lesson outline

- Contents of the data model
- The Data Dictionary
- Objects and the data model



For every data model entity, the application automatically creates an internal Gosu class with the same name. For every field in the data model entity, there is a field in the corresponding internal Gosu class. For example, the ABContact data model entity has a "FaxPhone" field, and the internal ABContact Gosu class also has a "FaxPhone" field.

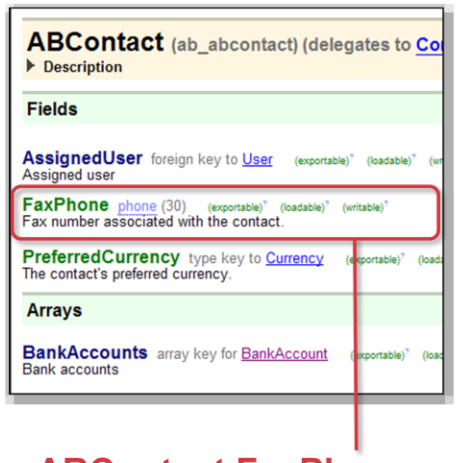
The Gosu classes that map to data model entities are internal and cannot be manipulated directly. However, it is possible to add things to them indirectly.
You can add fields to base application entities as discussed in the "Extending Base Entities" lesson.
You can add fields to a custom entity as discussed in the "Creating New Entities" lesson.
You can add methods to any entity as discussed in the "Enhancements" lesson.

Gosu classes that are automatically created from data model entities are sometimes referred to as "database-backed classes" because these classes have corresponding database tables, and instances of these classes can be saved to the database. Whenever a row is read from a database table, an instance of the corresponding Gosu class is created, and the data is placed into that instance. The data in this instance can then be displayed in the user interface and modified by the user prior to saving the data. Whenever the data in a run-time instance of a database-backed class is changed and saved, that data gets written to the corresponding database table. If the data was initially read from the database (such as when a user looks for an existing ABContact and then modifies it), the existing database row is updated to reflect the changes. If the data was created entirely in the run-time environment (such as when a user creates a new ABContact), then a new row is inserted into the database table.

It is also possible to create Gosu classes that are not database-backed. Instances of those classes can be created and modified during runtime, but they cannot be saved to the database. These types of classes are discussed in detail in the "Gosu Classes" lesson.

Dot notation

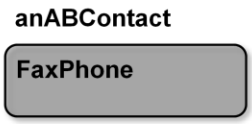
- **Dot notation** is a Gosu language syntax used to identify data objects and properties
  - Starts with object and ends with field, related object, or related array
- Relies on data model
  - Not used to configure data model
  - Available for User Interface configuration and Gosu classes
- Examples include data fields, object collections, and methods



Dot notation is a language syntax used to identify data elements such as text fields, related objects or collections of objects. The syntax is not used in data model configuration itself. It is useful to mention at this point in time because the data model definition informs the possible dot notation expressions for referencing entity data in code.

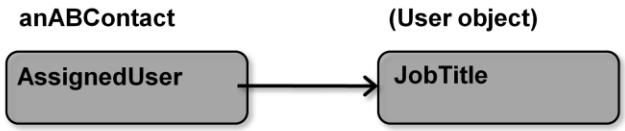
Dot notation syntax

- Referencing data field on given object
  - object.field




```
anABContact.FaxPhone
```

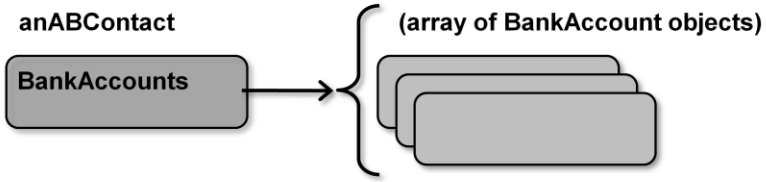
- Referencing field on related object
  - object.foreignKey.field




```
anABContact.AssignedUser.JobTitle
```

Dot notation syntax (2)

- Referencing array on given object
  - object.array




```
anABContact.BankAccounts
```

- Referencing field at subtype level
  - (object as subtype).field


```
(anABContact as ABPerson).DateOfBirth
```


Referencing fields at the subtype level is discussed in detail in the "Subtypes" lesson and in the "Atomic Widgets" lesson.

Lesson objectives review

- You should now be able to:
  - Describe the contents of a Guidewire data model
  - Identify information about a given application's data model
  - Reference entity fields using dot notation

Review questions

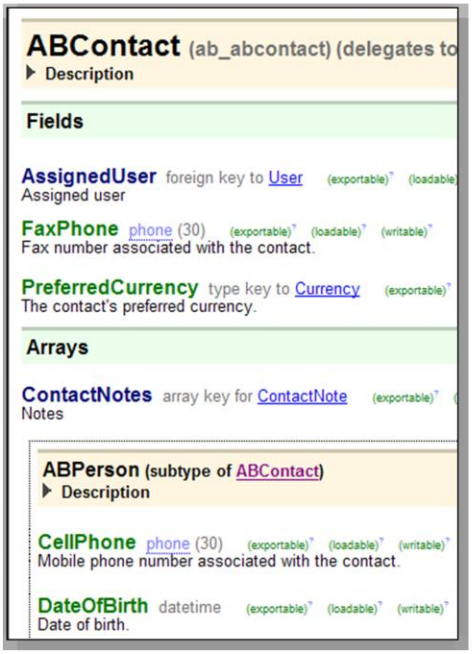
1. For each of the following, identify if it is stored as a database table, as a table column, or not stored in the database at all:
   a) An entity (such as ABContact)
   b) A physical data field (such as ABContact.CreateTime)
   c) A virtual data field (such as ABContact.FullName)
   d) A foreign key field (such as ABContact.AssignedUser)
   e) An array key field (such as ABContact.ContactNotes)
   f) A typelist (such as VendorType)
   g) A typekey field (such as ABContact.VendorType)
2. Name two circumstances in which you would execute the regen-dictionary command.

(continued)

Answers
1a) A table
1b) A column
1c) Not stored in the database (Technically, the source values are probably stored in the database. But the value of a virtual field itself is not stored as a separate value.)
1d) A column (specifically a foreign key column to the relevant entity's table)
1e) Not stored in the database
1f) A table
1g) A column (specifically a foreign key column to the typelist's table)

2. Possible answers: You would execute regen-dictionary when you install the application and whenever you extend the data model. You could also execute the command when you modify system permissions or some other aspect of access and therefore need to regenerate the Security Dictionary.

Review questions



3. For the object "myContact" of type ABContact, what is the dot notation for the object's:
   a) Preferred currency?
   b) Notes?
   c) Level of experience of myContact's assigned user?
   d) Mobile phone (if it is a person)

Answers

3a) myContact.PreferredCurrency
3b) myContact.ContactNotes
3c) myContact.AssignedUser.ExperienceLevel
3d) (myContact as ABPerson).CellPhone

Notices