ATG Repositories Item Properties

Presenter's Name
Presenter's Title

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Oracle Training Materials – Usage Agreement

Use of this Site ("Site") or Materials constitutes agreement with the following terms and conditions:

1. Oracle Corporation ("Oracle") is pleased to allow its business partner ("Partner") to download and copy the information, documents, and the online training courses (collectively, "Materials") found on this Site. The use of the Materials is restricted to the non-commercial, internal training of the Partner's employees only. The Materials may not be used for training, promotion, or sales to customers or other partners or third parties.

2. All the Materials are trademarks of Oracle and are proprietary information of Oracle. Partner or other third party at no time has any right to resell, redistribute or create derivative works from the Materials.

3. Oracle disclaims any warranties or representations as to the accuracy or completeness of any Materials. Materials are provided "as is" without warranty of any kind, either express or implied, including without limitation warranties of merchantability, fitness for a particular purpose, and non-infringement.

4. Under no circumstances shall Oracle or the Oracle Authorized Delivery Partner be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this Site of Materials. As a condition of use of the Materials, Partner agrees to indemnify Oracle from and against any and all actions, claims, losses, damages, liabilities and expenses (including reasonable attorneys' fees) arising out of Partner's use of the Materials.

5. Reference materials including but not limited to those identified in the Boot Camp manifest can not be redistributed in any format without Oracle written consent.

Agenda

- Repository items as properties
- Item descriptor inheritance

Learning Objectives

At the end of this lesson you should be able to:

- Use repository items as properties of other repository items
- Use repository items in multi-valued properties using List, Map and Set
- Represent Many-to-Many relationships
- Understand and use cascade data relationships
- Learn about primary tables and auxiliary tables
- Extend repository definitions with item descriptor inheritance
- Use transient properties to store temporary values in repositories

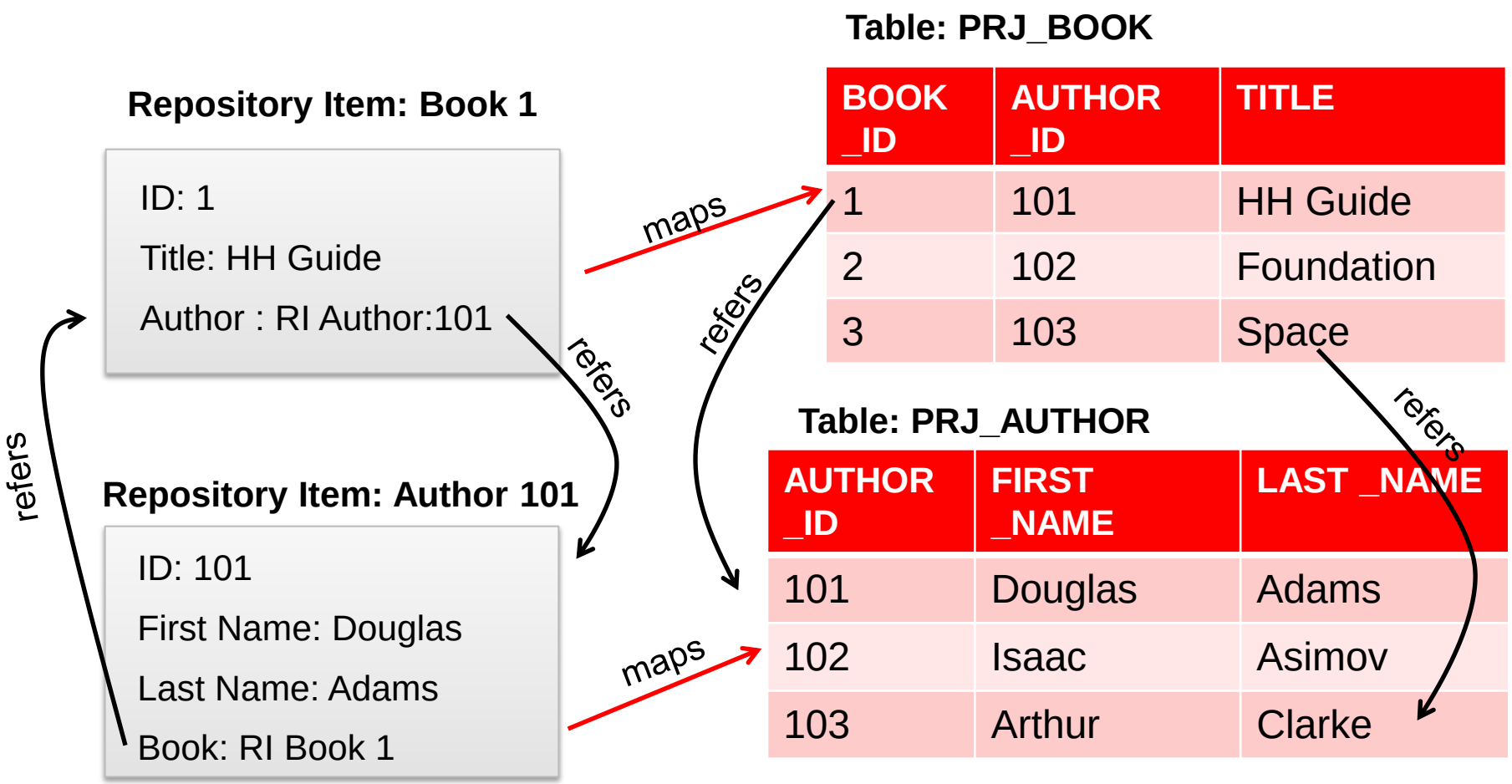
Section 1:
Repository Data Model Definition:

Repository Items as Properties

Repository Items as Properties

- The value of a property of a repository item can be another repository item.
- Both multi-valued properties and single-valued properties can have repository items as property values.
- Repository item properties are used to implement one-to-one and one-to-many relationships.
- The <property> tag for item properties must use attribute item-type instead of data-type.
- This is a powerful feature allowing for much greater flexibility in defining the data schema.

Example: Repository Item as Properties



- In this example, Book repository Item has a reference to an Author Repository Item and vice versa.

Example code: Repository Items as properties

```
<!-- The "book" item type -->
<item-descriptor name="book" default="true">
  <table name="PRJ_BOOK" type="primary" id-column-names="id">
    <property name="title"/>
    <property name="author" column-name="author_id"
          item-type="author"/>
  </table>
</item-descriptor>

<!-- The "author" item type -->
<item-descriptor name="author">
  <table name="author" id-column-names="author_id"
                              type="primary">
    <property name="lastName" column-name="FIRST_NAME" />
    <property name="firstName" column-name="LAST_NAME" />
  </table>
  <table name="book" id-column-names="author_id"
                              type="auxiliary">
    <property name="book" item-type="book"
                  column-name="book_id"/>
  </table>
</item-descriptor>
```

Using Repository Items in Multiple Valued Properties

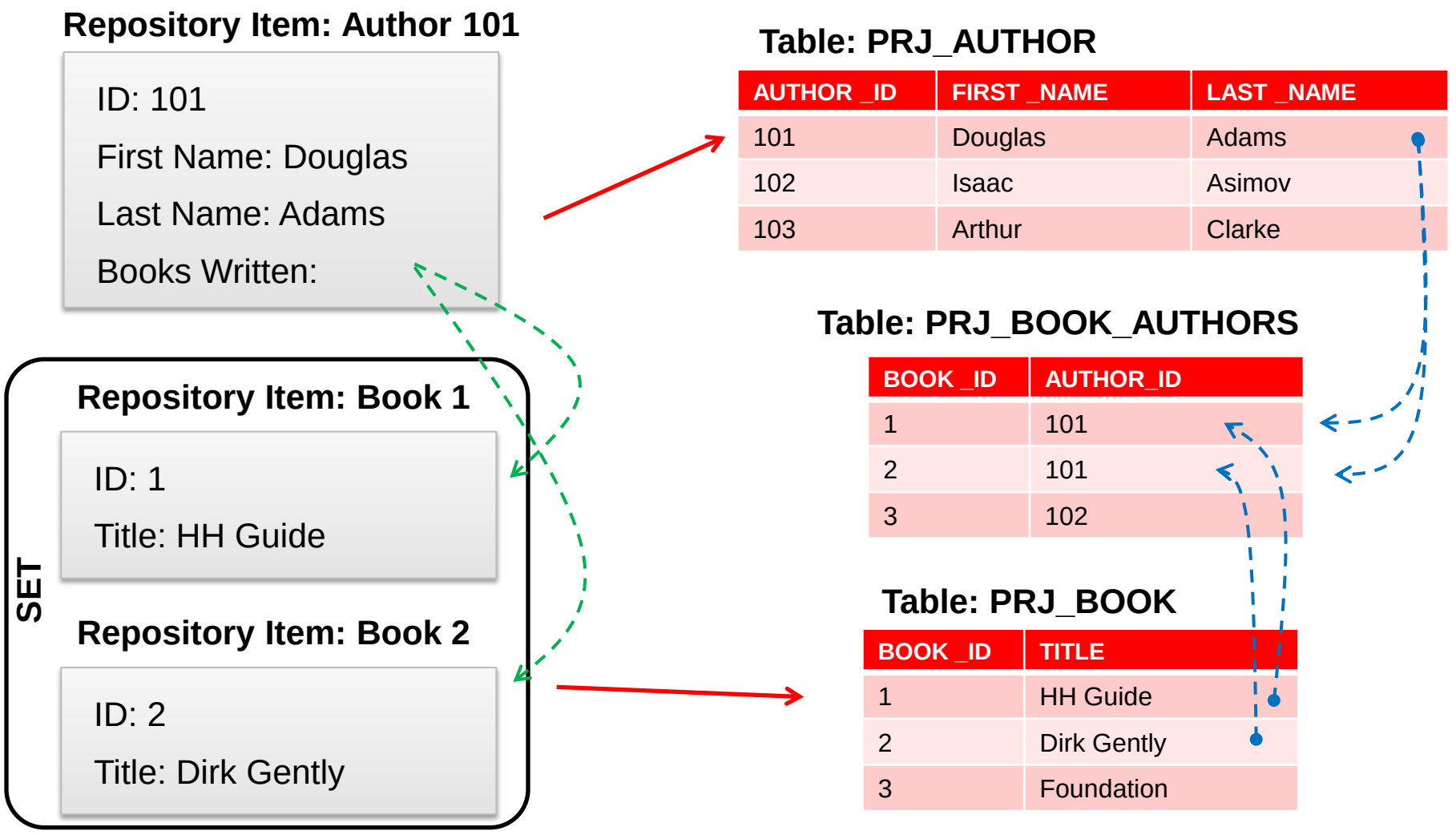
- Similar to using non repository items as multi valued properties, other repository items may be used as well.
- The <property> tag for multi-item properties needs to set the following attributes:
  - data-type:
    - Set,
    - List,
    - Map,
    - Array.
  - component-item-type: repository item type name. component-data-type attribute is used instead when defining multi-valued properties.

Example : Multiple Item Properties(Set)

- One author has a set of several books.
- The data-type attribute is set to set.
- We use an intermediate table called PRJ_BOOK_AUTHORS to store the relationships.

```
<item-descriptor name="author">
  <table name="PRJ_AUTHOR" type="primary" id-column-name="ID">
    <property name="id" column-name="ID" data-type="string"/>
        <property name="firstName" column-name="FIRST_NAME"
                                   data-type="string"/>
        <property name="LastName" column-name="LAST_NAME"
                                   data-type="string"/>
  </table>
  <table name="PRJ_BOOK_AUTHORS" type="multi"
                        id-column-name="AUTHOR_ID">
    <property name="booksWritten" data-type="set"
              component-item-type="book"  column-name="book_id"/>
  </table>
</item-descriptor>
```

Example: Multi Item Properties (Set)

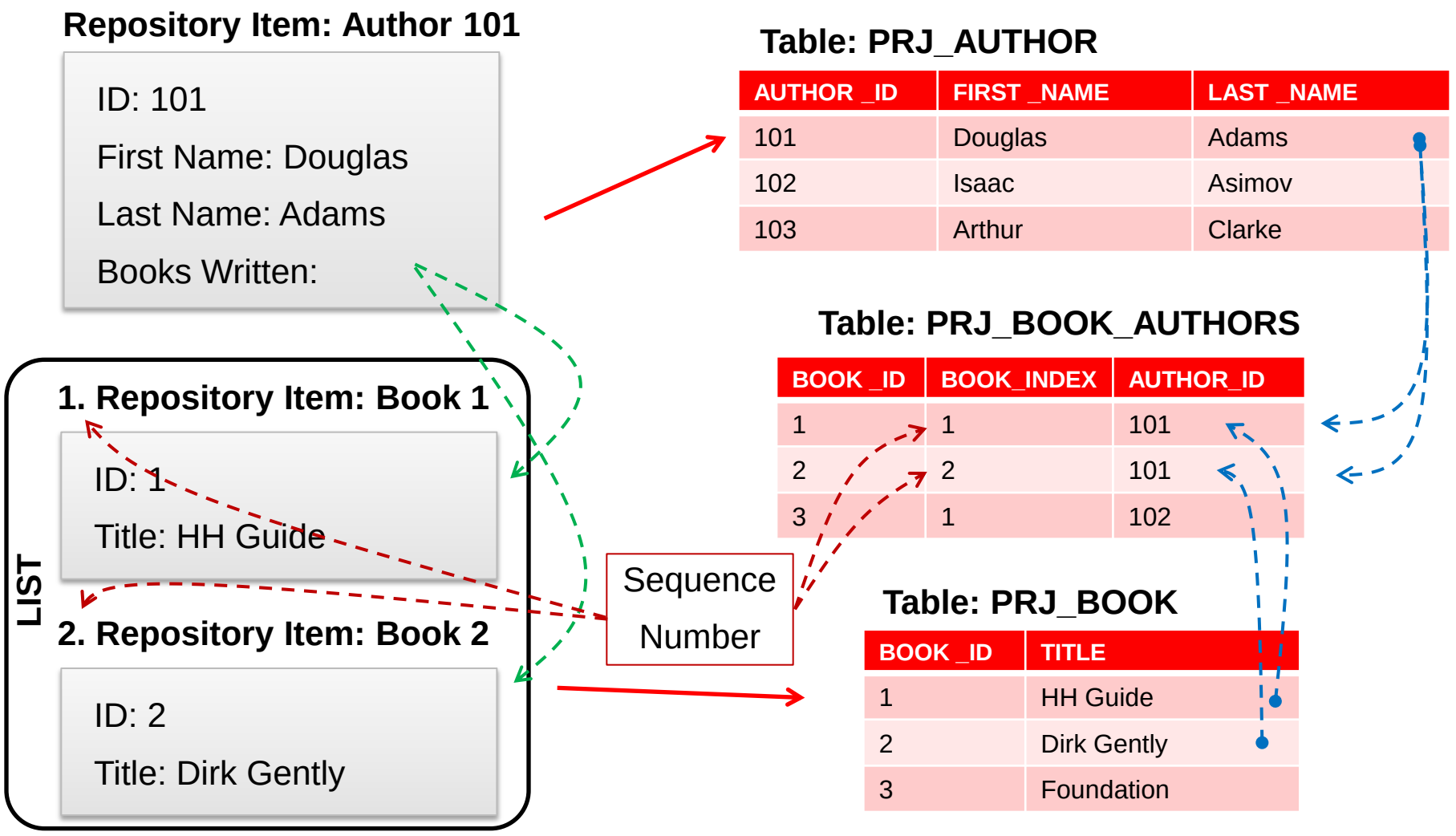


Example: Multiple Item Properties (List)

- One author has a list of ordered books. We need to store the sequence number.
- The data-type attribute should be set to list.
- A new column called BOOK_INDEX stores the sequence number.

```
<item-descriptor name="author">
  <table name="PRJ_AUTHOR" type="primary" id-column-name="ID">
    <property name="id" column-name="ID" data-type="string"/>
        <property name="firstName" column-name="FIRST_NAME"
             data-type="string"/>
        <property name="LastName" column-name="LAST_NAME"
             data-type="string"/>
  </table>
  <table name="PRJ_BOOK_AUTHORS" type="multi"
         id-column-name="AUTHOR_ID" multi-column-name="BOOK_INDEX" >
    <property name="booksWritten" data-type="list"
        component-item-type="book" column-name="book_id"/>
  </table>
</item-descriptor>
```

Example: Multi Item Properties (List)

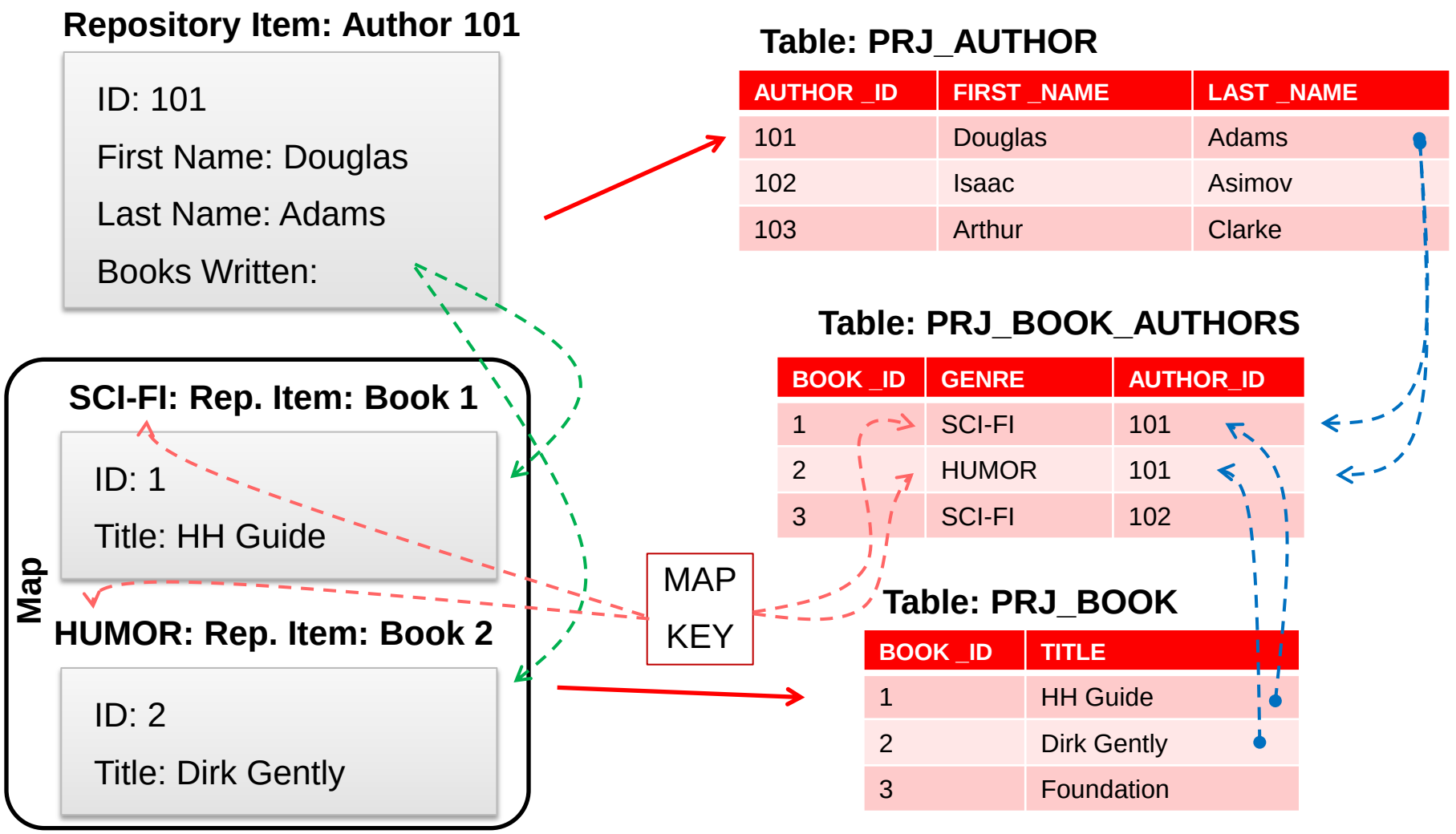


Example: Multiple Item Properties (Map)

- One author has a map of books. Genre is the key to the map.
- The data-type attribute should be set to map.
- A new column called GENRE stores the key.

```
<item-descriptor name="author">
  <table name="PRJ_AUTHOR" type="primary" id-column-name="ID">
    <property name="id" column-name="ID" data-type="string"/>
        <property name="firstName" column-name="FIRST_NAME"
             data-type="string"/>
        <property name="LastName" column-name="LAST_NAME"
             data-type="string"/>
  </table>
  <table name="PRJ_BOOK_AUTHORS" type="multi"
         id-column-name="AUTHOR_ID" multi-column-name="GENRE" >
    <property name="booksWritten" data-type="map"
        component-item-type="book‟ column-name="book_id" />
  </table>
</item-descriptor>
```

Example: Multi Item Properties (Map)



Many-to-Many Relationships

- Many-to-many relationships are one-to-many relationships from both sides.
- No extra table is necessary. The same intermediate table can be leveraged from both sides.
- The data-type attribute on both sides can be set to the appropriate one.
- The example provided shows a set relationship between authors to books and books to authors.

Example: Many-to-Many Relationships

```
<item-descriptor name="author">
  <table name="PRJ_AUTHOR" type="primary" id-column-name="ID">
    <property name="id" column-name="ID" data-type="string"/>
          <property name="firstName" column-name="FIRST_NAME"
                              data-type="string"/>
          <property name="LastName" column-name="LAST_NAME"
                              data-type="string"/>
  </table>
  <table name="PRJ_BOOK_AUTHORS" type="multi‟ id-column-name="AUTHOR_ID">
    <property name="booksWritten" data-type="set"
              component-item-type="book‟  column-name="book_id"/>
  </table>
</item-descriptor>

<item-descriptor name="book">
  <table name="PRJ_BOOK" type="primary" id-column-name="ID">
    <property name="id" column-name="ID" data-type="string"/>
          <property name="title" column-name="TITLE‟ data-type="string"/>
  </table>
  <table name="PRJ_BOOK_AUTHORS" type="multi” id-column-name="BOOK_ID">
    <property name="authors" data-type="set"
              component-item-type="author‟ column-name="author_id"/>
  </table>
</item-descriptor>
```

Cascading Data Relationships

- When an item property refers to another item, both items can be deleted, inserted, or updated at the same time.
- It is recommended to use the cascade attribute in a <property> tag to handle hierarchical properties.
- Cascade attribute can be set to one or more of these values (separated with character “,”):
  - insert,
  - update,
  - delete.
- As an example in the author and books example:

If an author is deleted, related books can be deleted as well.

Cascade Insert

- When you create a repository item that contains a property with the item-type attribute and the properties cascade attribute is set to insert:
  - An item of the type declared by the item-type attribute is also created.
  - The property is set to point to the newly created item.
- In the example, creating an author will result in the creation of a new address as well.

```
<item-descriptor name="author" >
  <table name="PRJ_AUTHOR" type="primary"
                            id-column-names="id">
    <property name="name"/>
    <property name="address" column-name="address_id"
          item-type="address" cascade="insert"/>
  </table>
</item-descriptor>
```

Cascade Update

- If a repository item property references other items and the property's cascade is set to update:
  - When you call addItem(), any new items referenced by this property are added automatically to the repository.
  - When you call updateItem(), any modified referenced items are automatically updated. Any referenced items that are new items are added.
- In the example below, when author is added or updated, address is also added or updated to the repository.

```
<item-descriptor name="author" >
  <table name="PRJ_AUTHOR" type="primary"
                           id-column-names="id">
    <property name="name"/>
    <property name="address" column-name="address_id"
          item-type="address" cascade="update"/>
  </table>
</item-descriptor>
```

Cascade Delete

- If a repository item property references other items and the property's cascade is set to delete:
  - When an item is deleted, the referenced item is deleted as well.
- Exercise caution when using with multi-valued items especially on the many side of the relationship.
- Use the cascadeDeleteOrder attribute to control the order of deletion.
- In the example below, when author is deleted, address is deleted as well.

```
<item-descriptor name="author" >
  <table name="PRJ_AUTHOR" type="primary"
                           id-column-names="id">
    <property name="name"/>
    <property name="address" column-name="address_id"
          item-type="address" cascade="delete"/>
  </table>
</item-descriptor>
```

Cascade Operations

- You can set cascade to a comma separated list of values as shown in the example.
- Whenever an author item is created, added, updated, or deleted, the same operation is reflected on the address repository item.

```
<item-descriptor name="author" >
  <table name="PRJ_AUTHOR" type="primary"
                            id-column-names="id">
    <property name="name"/>
    <property name="address" column-name="address_id"
        item-type="address" cascade="insert,update,delete"/>
  </table>
</item-descriptor>
```

Section 1
Check Your Understanding

What attribute on the property description indicates that the property refers to another repository item instead of a primitive type?

Answer: The item-type attribute indicates this. The data-type attribute indicates it is a primitive type property.

Section 1
Check Your Understanding

What data structures can be used to represent multi-valued properties referring to repository items?

Answer: Set, map, list, and array can be used

Section 1 
Check Your Understanding

What is the component-item-type set to when referring to multi-valued properties that refer to repository items?

Answer: Component-item-type refers to the reference by the multi-valued property.

Section 1 
Check Your Understanding

How are many-to-many relationships represented in ATG?

Answer: They are represented as one-to-many from both sides of the relationship.

Section 1 ✔
Check Your Understanding

What is the cascade attribute used for on a property?

Answer: It is used to specify insert, update, or delete cascade relationship between parent and child.

Summary

- Properties of repository items can refer to other repository items.
- This can be done for both single-valued and multi-valued properties.
- Multi-valued can be one-to-many. Many-to-many is implemented as one-to-many on both sides.
- One-to-many can be implemented as set, list, or map.
- Cascade data relationships can define how the referenced (child) repository items behave when operations are performed on the parent repository item.
- The cascade operations supported are insert, update, and delete.

Section 2:
Repository Data Model Definition:

Item Descriptor Inheritance and Other Advanced Concepts

Primary Table

- Each item-descriptor:
  - Must specify no more than one primary table with attribute type="primary.“
  - Must set id-column-names attribute as table primary key.
  - Can set id-space-names attribute to control ID generation.
- It can use additional tables to store values using one-to-one (auxiliary) or one-to-many (multivalued, repository items) or many-to-many(two sided).

```
<item-descriptor name="user">
   <table name="prj_user" type="primary" id-column-names="id">
      <property name="login" data-type="string"/>
   </table>
</item-descriptor>
```

Auxiliary Table

- An auxiliary table is used to add more properties to an existing item type in a new table.
- An auxiliary table definition:
  - Must set attribute type =" auxiliary ."
  - Must set id-column-names attribute to reference the original item ID.
  - Indicates how to join auxiliary tables to the primary table.
- Auxiliary tables allow for logical separation of data into various tables.
- In the example, the user item has login and address as properties.

Auxiliary Table Example

Repository Item: User

ID: 1
Login: jbruin
Address1: My Street
City: My City
State: CA
Zip: 90210

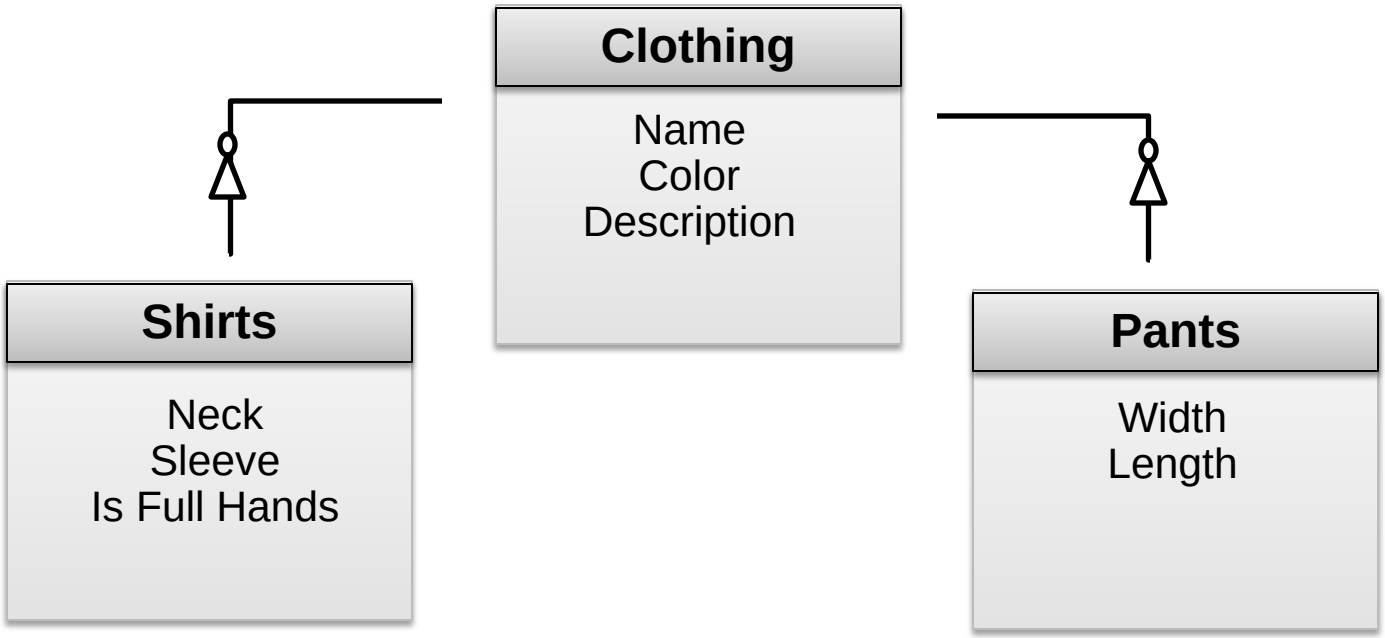
```
<item-descriptor name="user">
   <table name="prj_user" type="primary" id-column-names="id">
      <property name="login" data-type="string"/>
   </table>
   <table name="prj_address" type="auxiliary"
                id-column-names="id">
     <property name="address1" data-type="string"/>
     <property name="city" data-type="string"/>
     <property name="state" data-type="string"/>
     <property name="zip" data-type="string"/>
   </table>
</item-descriptor>
```

Item Descriptor Inheritance

- SQL Repository supports a simplified form of inheritance.
- It uses optional one-to-one relationship between the primary and an auxiliary table.
- It is similar to object-oriented sub-classing.
- Item types may be created as sub-types of other item types:
  - Super-type properties are inherited.
  - Sub-type may add properties stored in an auxiliary or multi table.

Example of Item Descriptor Inheritance

- The base item descriptor is clothing. It defined properties common to shirts and pants.
- The item descriptors shirts and pants are sub types of clothing.
- They inherit the common properties and add their own.



Example of Inheritance – Base Item

- The base or parent item descriptor uses the sub-type-property attribute set to an enumerated data-type.
- An enumerated property type is defined for each sub type repository item.

```
<!-- The "clothing" item type, a base type -->
<item-descriptor name="clothing" sub-type-property="type">
  <table name="clothing" type="primary" id-column-names="id">
    <property name="type" data-type="enumerated">
      <option value="shirt"/>
      <option value="pants"/>
    </property>
    <property name="name"/>
    <property name="color"/>
    <property name="description"/>
  </table>
</item-descriptor>
```

Example of Inheritance – Sub Item

- The sub types define a super-type attribute pointing to the base item.
- The sub-type-value is set to one of the enumerated types.

```
<item-descriptor name="shirt" super-type="clothing"
        sub-type-value="shirt">
  <table name="shirt" type="auxiliary" id-column-names="id">
    <property name="neck"/>
       <property name="sleeve"/>
       <property name="isFullHands" data-type="boolean"/>
  </table>
</item-descriptor>
<item-descriptor name="pants" super-type="clothing"
          sub-type-value="pants">
  <table name="pants" type="auxiliary" id-column-names="id">
    <property name="length"/>
       <property name="width"/>
  </table>
</item-descriptor>
```

Benefits of Item Descriptor Inheritance

- Item Descriptor inheritance is built in capability of SQL Repositories.
- It lets you easily query across complex hierarchy of sub types.
- It optimizes the database by not having duplicate columns.
- It lets you extend repository items to handle custom fields that are not applicable to all sub types.
- Your application needs only to be modified to handle the new sub type where it is different.

Item Descriptor Inheritance Queries

- Easily query across a complex hierarchy of sub-types.
- Query against a parent item descriptor returns items of a child item descriptor.
- Given below, you can query for:
  - Clothing whose neck is 15,
  - Clothing whose color is red (returns shirts and pants),
  - Clothing whose Length is 32.

Limitations of Item Inheritance

- An item type can only inherit from one parent.
- An item type hierarchy can only have one sub-type-property value.
- All parent item descriptors must be fully defined by the time they are referenced in the XML.
- Avoid using too many levels of inheritance.
- Queries against items whose properties span multiple sub-types may require joins to all tables in the hierarchy.
- There may be performance implications to the join tables.

Transient Properties

- Transient property is not associated with any column in tables.
- It is never stored or read from the persistent data store.
- It is readable and writable, but not query able.
- You can specify a transient property by defining a <property> tag that is not associated with any database table, but it must be a direct child of an <item-descriptor> tag.
- When a repository item is destroyed, the transient value is lost.
- A common example is the users' LoggedIn state.

Example: Transient Properties

- In the following example, the user item descriptor has a transient property that specifies whether the user is logged in.
- As can be seen, the loggedIn property is a child of the item-descriptor tag and does not appear in any table definition.

```
<item-descriptor name="user" sub-type-property="userType">
    <property name="loggedIn" data-type="boolean">
    <table name="user" type="primary"
                       id-column-names="id">
        <property name="userType" data-type="enumerated"
                  column-name="user_type">
    </table>
</item-descriptor>
```

Property Fetching Groups

- To improve performance, you can group properties into logical groups.
- Properties in the same group are retrieved in the same select statement.
- By default, properties in the same table are in the same group.
- Specify a different group name using the group property attribute tag.

```
<property name="login" group="display"/>
<property name="name" group="display"/>
<property name="address1" group="address"/>
<property name="city" group="address"/>
<property name="state" group="address"/>
<property name="zip" group="address"/>
```

Categorization and Sorting Properties

- To control properties displaying in BCC:
  - Use category attribute to group similar properties.
  - Using propertySortPriority attribute to control display order of categorized properties.

```
<property category="Basics" name="firstName"
          data-type="string" display-name="First name">
   <attribute name="propertySortPriority" value="-3"/>
</property>

<property category="Basics" name="middleName"
        data-type="string‟ display-name="Middle name">
   <attribute name="propertySortPriority" value="-2"/>
</property>

<property category="Basics" name="lastName"
        data-type="string"  display-name="Last name">
   <attribute name="propertySortPriority" value="-1"/>
</property>
```

Section 2 
Check Your Understanding

How many primary and auxiliary tables can be there for a given item descriptor?

Answer: An item descriptor can have at most one primary table and many auxiliary tables.

Section 2 
Check Your Understanding

How do you identify auxiliary tables in the repository definition file?

Answer: The type attribute must be set to auxiliary for auxiliary tables.

Section 2

Check Your Understanding

How many repository items can an item inherit from?

Answer: Repository item can at most inherit from one other repository item.

Section 2 
Check Your Understanding

How do you identify a repository item that extends to another repository item?

Answer: The super-type attribute on the item will point to the parent repository item.

Section 2 
Check Your Understanding

How do you create properties that are not stored in the repository?

Answer: You can create transient properties by declaring them outside the table tag in the item-descriptor.

Questions

- How many primary and auxiliary tables can be there for a given item descriptor?
  - An item descriptor can have at most one primary table and many auxiliary tables.
- How do you identify auxiliary tables in the repository definition file?
  - The type attribute must be set to auxiliary for auxiliary tables.
- How many repository items can an item inherit from?
  - Repository item can at most inherit from one other repository item.
- How do you identify a repository item that extends to another repository item?
  - The super-type attribute on the item will point to the parent repository item.
- How do you create properties that are not stored in the repository?
  - You can create transient properties by declaring them outside the table tag in the item-descriptor.

Summary

- There should be one primary table per item descriptor. There can be many auxiliary or multi valued tables.
- Item descriptor inheritance uses auxiliary table relationships to extend and inherit a parent repository item to form a child item.
- Transient properties are properties that are not persisted in the data store.
- Properties can be grouped to optimize data retrieval.
- Properties can be categorized and sorted to facilitate merchandizing through the BCC.

Q&A

ORACLE IS THE INFORMATION COMPANY


ORACLE®