Contents

This topic explains how DevForce replaces temporary **foreign key id** values with permanent ids during **id fix-up**. **Foreign key id fix-up** is phase withing the larger fix-up process. This topic is only relevant when ids are generated by the database or by a custom id generator.

Foreign Key Ids

When two entities are related, the dependent entity (child entity) holds the key of the related principal entity (parent entity) in a foreign key property.

For example, suppose an Order entity is related to a parent SalesRep entity. SaleReps have many orders. An Order has one parent SalesRep. The Order entity has a SalesRepID foreign key property that holds the ID of the parent SalesRep who took the order.

Let's create a SalesRep called "newSalesRep". Assume that the key of the SalesRep entity is a store-generated integer id. When we create a new SalesRep, DevForce assigns it a temporary id automatically. That temporary id will be a negative integer. Let that temporary id be -101.

Now let's create an Order called "newOrder" and set its navigation property to "newSalesRep".

```
newOrder.SalesRep = newSalesRep;
```

```
newOrder.SalesRep = newSalesRep
```

Setting the SalesRep navigation property for "newOrder" also sets its SalesRepID property. The SalesRepID value will be -101, the same as the temporary id of "newSalesRep".

Fix up

The application begins to save these new entities. Because temporary ids have been generated, the fix-up process begins. The permanent id for the "newSalesRep" is determined; let's suppose the value is 300.

The fix-up process knows about relationships between the SalesRep and other entities. In particular, it knows about SalesRep's relationship with Order and it knows about Order's SalesRepID.

Accordingly, it finds the "newOrder" and replaces its current SalesRepID value of -101 with 300, the permanent id of "newSalesRep". "newOrder" has been fixed-up and is ready to be saved to the database.

This all happens automatically.

Fix up failure

All works fine as long as every entity relationship is defined in the DevForce metadata. Such will be the case when the DevForce entity classes are generated from an Entity Framework Entity Data Model (EDM) which accurately reflects every pertinent foreign key relationship in the database.

There is usually nothing to worry about. But it is possible to omit a relationship, especially if that relationship is not enforced by foreign key constraints in the database.

The telltale sign of a problem is negative values in foreign key id columns of the database.

Let's extend our previous example. Suppose that Customers are also assigned to SalesReps. The SalesRep is the parent and the Customer is the child in this relationship. Therefore, the Customer entity carries a SalesRepID that points to its parent SalesRep.

Customer parallels Order in this regard ... except in one detail.

DevForce doesn't know that Customer and SalesRep are related. For some reason there is no association between them. Customer has a SalesRepID property but no known relation to SalesRep and no navigation property to SalesRep either. It just has the SalesRep's id.

A missing relationship is a critical omission, as we will see.

This time we'll start with an existing Customer, held in a variable called "aCustomer". We're intent on reassigning this customer to the "newSalesRep" which we just created.

Although there is no SalesRep navigation property, we are determined to establish the association implicitly by setting "aCustomer"'s SalesRepID directly.

```
aCustomer.SalesRepID = newSalesRep.Id;
```

```
aCustomer.SalesRepID = newSalesRep.Id
```

The absence of the myCustomer.SalesRep navigation property should have been a warning that a critical relation might be missing.

Let's see replay the fix-up scenario with this new twist:

- The value of newSalesRep.ID is updated to its permanent value, 300.
- newOrder.SalesRepID is fixed up to 300 because there is a relation between Order and SalesRep.
- aCustomer.SalesRepID retains the temporary id = –101 because there is no relation between Customer and SalesRep; fix-up doesn't know it should replace aCustomer's SalesRepID.

Only bad things can happen as DevForce continues trying to save these entities.

If the database schema actually has a foreign key constraint on the Customer's SalesRepID column, the save will fail with an exception because the database can't find a SalesRep with the negative id = -101.

If there is no foreign key constraint, "aCustomer" will be written to the database with a negative SalesRepID = -101. That is a data integrity violation even if there is no code or database constraint to catch it.

Important: Map *all* entity relations.