Microsoft Dynamics® AX 2012

Using date effective data patterns

White Paper

This white paper describes how to use valid time state tables to store and track object history over its lifetime in Microsoft Dynamics AX 2012. Topics include the difference between the two date-time entity types for valid time state tables; update modes; using the ValidTimeState keyword in queries; runtime behaviors; and scenarios that are not supported by the framework. Some date effective data design patterns are suggested for forms that use valid time state tables. The integration of the security framework and valid time state tables is also discussed.

Date: July 2011

Author: Xinhua Huang

Table of Contents

Introduction

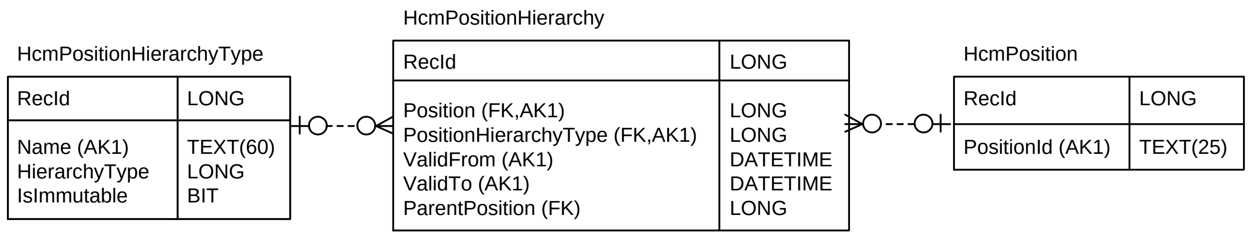
Many business scenarios require tracking object history over its lifetime. For example, a bank provides a certificate of deposit with a rate that is effective for a certain time period, student records in a university, employee data in company, and other complex business information.

In previous versions of Microsoft Dynamics AX, various forms of implementing date effectiveness were implemented by in application modules, such as Human resources, Procurement and sourcing, and across the finance modules. Microsoft Dynamics AX 2012 offers the date effective functionality at the kernel to provide the ease, consistency, and scalable behavior of application code for date effective scenarios.

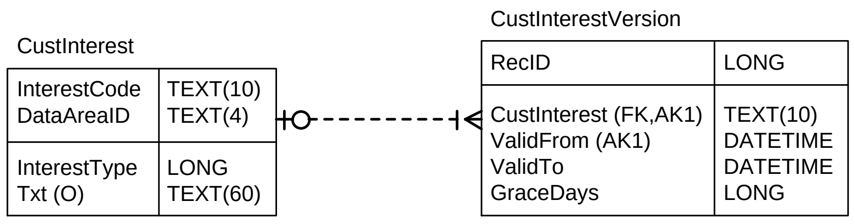
Out-of-the-box, date effective functionality enables the design time, run time, programmability aspects, and intuitive user interface concepts for optimal end-to-end scenario development.

The following are the date effective data modeling patterns that are supported

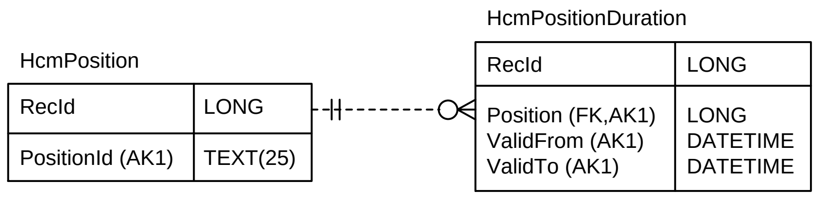
- **Date Effective Association Collection**



- **Date Effective Attributes**



- **Date Effective Attribute Collection**



Terminology

The following terms are used throughout this white paper.

Term	Definition
valid time state table	A table that tracks the state of an entity over time using ValidFrom and ValidTo columns.
ValidTimeStateKey	The alternate key from the table that is used to enforce the valid time state semantics.
gap	A condition in which a particular record does not have any occurrence for a time period, but it has some occurrences before and after the gap interval.
overlap	The same record has more than one occurrence over an overlapping time period.
current record	A record that is effective at the present time.
past record	A record that was effective in an earlier time period.
future record	A record that will be effective in a future time period.
CreateNewTimePeriod	An update mode only to date effective tables. When a current record is updated in CreateNewTimePeriod mode, the initial record is closed, and a new current record is created.
Correction	An update mode that is similar to non-date effective tables.
EffectiveBased	When updating a valid time state table in EffectiveBased mode, if the record being updated is a current record, the record is updated in CreateNewTimePeriod mode. If the record being updated is a future record, the record is updated in Correction mode. If the record is a past record, the record cannot be updated.

Designing a valid time state table in the AOT

To use the date effective feature, you must create a valid time state table in the Application Object Tree (AOT). When you create the table, consider the following two points.

1. The effective granularity for the **ValidFrom** and **ValidTo** fields. You can select one of two date effective data patterns: **UtcDateTime** or **Date**.
2. Whether the table allows gap.

For example, the HcmPositionWorkerAssignment table applies UtcDateTime granularity and allows gap, and the CustInterestVersion table, applies Date granularity and does not allow gap. These examples will be used throughout this white paper.

The following UML diagrams define and describe the object structure and behavior for the HcmPositionWorkerAssignment and CustInterestVersion tables.

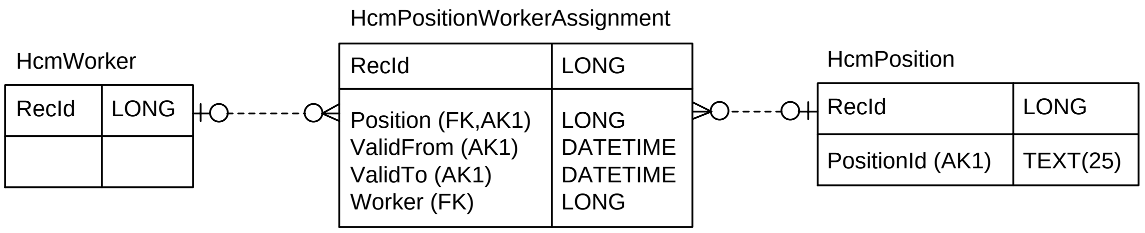


Figure 1: HcmPositionWorkerAssignment UML

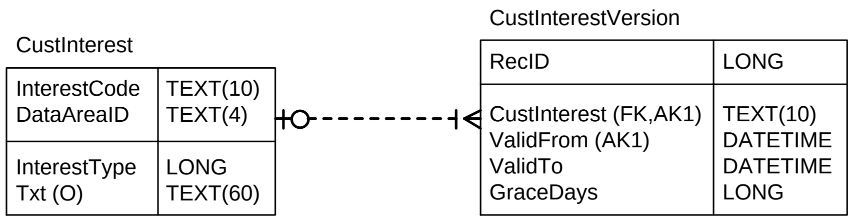


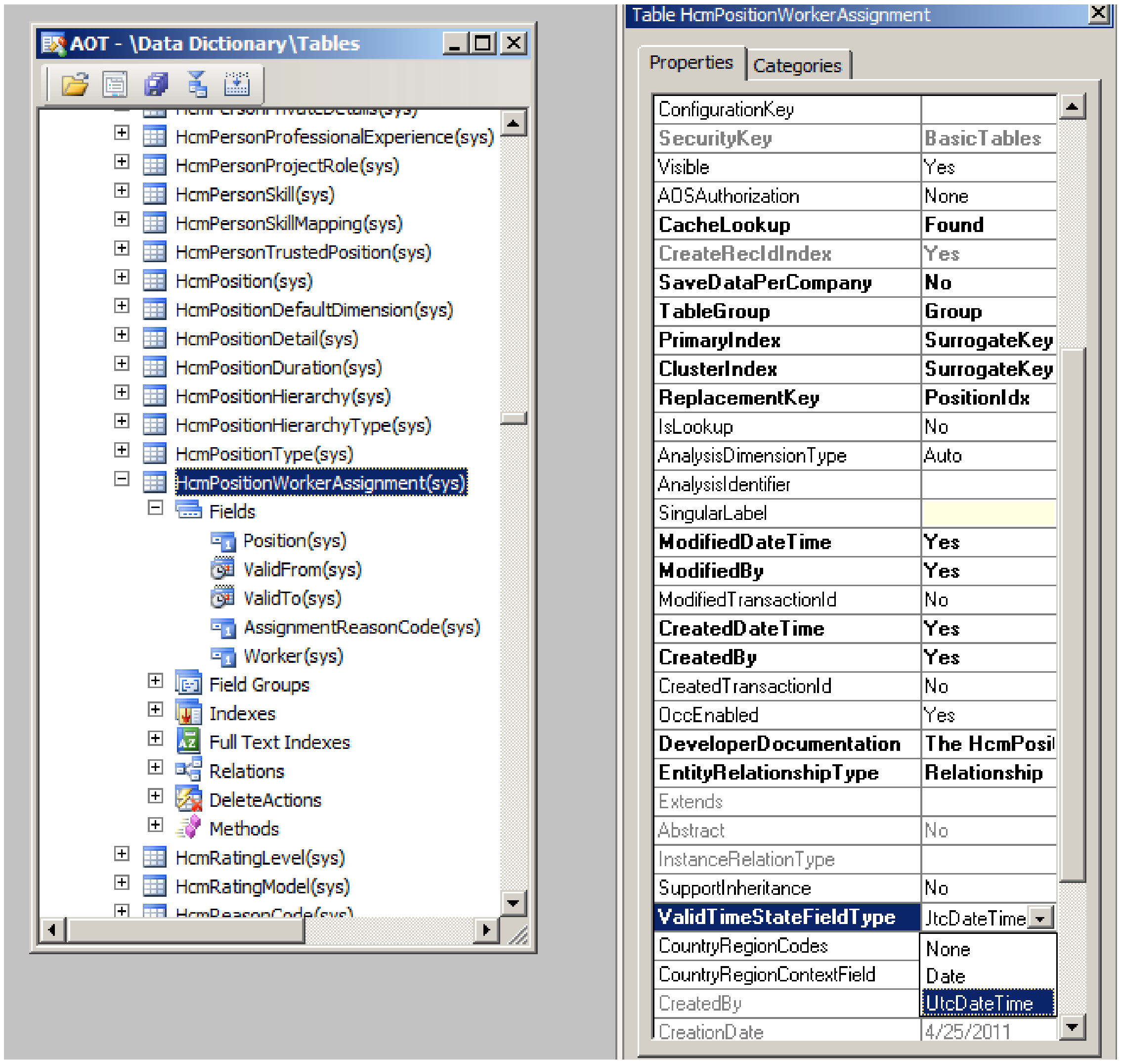
Figure 2: CustInterestVersion UML

The HcmPositionWorkerAssigment table defines the schema for storing positions assigned to workers. At a specific time period, each position can only be assigned to only one worker. During a time period, a position can have no workers assigned to it. For an example, see the data in the following table.

Position	Worker	ValidFrom	ValidTo
Upgrade	Anders Jensen	5/30/2000 00:00:00 am	12/31/2002 23:59:59 pm
Upgrade	Emma Baker	1/1/2005 00:00:00am	12/31/2005 23:59:59 pm

To enable the HcmPositionWorkerAssignment table to be a valid time state table with a date effective data type of UtcDateTime and a table that allows gap, the following properties in the AOT must be set:

1. Launch the Microsoft Dynamics AX 2012 development workspace.
2. In the AOT, navigate to **Data Dictionary** > **Tables**.
3. Find the **HcmPositionWorkerAssigment** table.
4. On the **Properties** tab, **ValidTimeStateFieldType** should be set to **UtcDateTime**.
5. Validate that the **ValidFrom** and **ValidTo** fields have the **UtcDateTime** type.



AOT - \Data Dictionary\Tables

```
HcmPersonProfessionalExperience(sys)
HcmPersonProjectRole(sys)
HcmPersonSkill(sys)
HcmPersonSkillMapping(sys)
HcmPersonTrustedPosition(sys)
HcmPosition(sys)
HcmPositionDefaultDimension(sys)
HcmPositionDetail(sys)
HcmPositionDuration(sys)
HcmPositionHierarchy(sys)
HcmPositionHierarchyType(sys)
HcmPositionType(sys)
HcmPositionWorkerAssignment(sys)
    Fields
        Position(sys)
        ValidFrom(sys)
        ValidTo(sys)
        AssignmentReasonCode(sys)
        Worker(sys)
    Field Groups
    Indexes
    Full Text Indexes
    Relations
    DeleteActions
    Methods
HcmRatingLevel(sys)
HcmRatingModel(sys)
```

Table HcmPositionWorkerAssignment

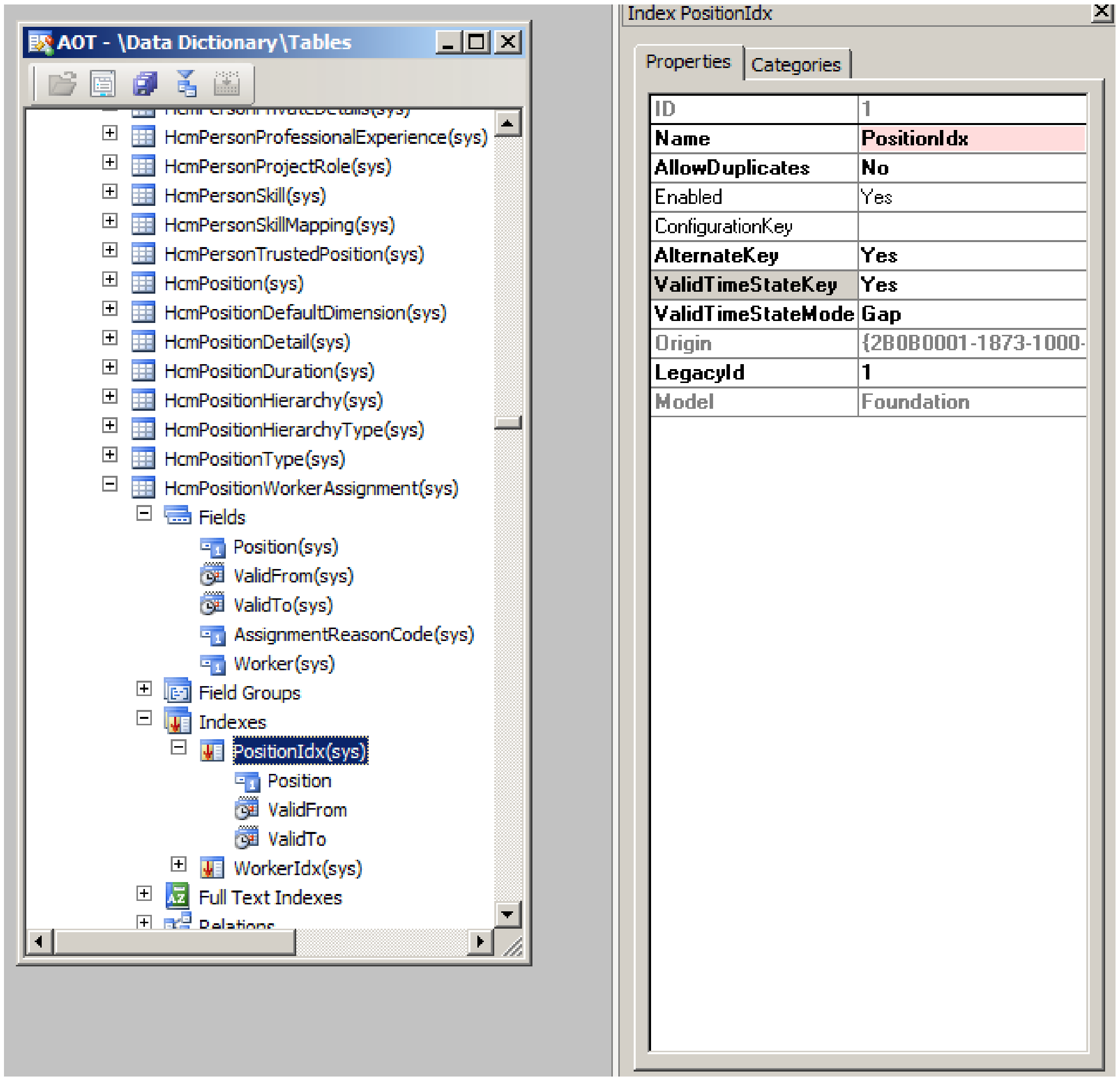
Properties | Categories

ConfigurationKey	
SecurityKey	BasicTables
Visible	Yes
AOSAuthorization	None
CacheLookup	**Found**
CreateRecIdIndex	Yes
SaveDataPerCompany	**No**
TableGroup	**Group**
PrimaryIndex	**SurrogateKey**
ClusterIndex	**SurrogateKey**
ReplacementKey	**PositionIdx**
IsLookup	No
AnalysisDimensionType	Auto
AnalysisIdentifier	
SingularLabel	
ModifiedDateTime	**Yes**
ModifiedBy	**Yes**
ModifiedTransactionId	No
CreatedDateTime	**Yes**
CreatedBy	**Yes**
CreatedTransactionId	No
OccEnabled	Yes
DeveloperDocumentation	**The HcmPosi**
EntityRelationshipType	**Relationship**
Extends	
Abstract	No
InstanceRelationType	
SupportInheritance	No
ValidTimeStateFieldType	UtcDateTime
CountryRegionCodes	None
CountryRegionContextField	Date
CreatedBy	UtcDateTime
CreationDate	4/25/2011

6. In the AOT, expand the **Indexes** node, and then select **Position.Idx**.
7. On the **Properties** tab:
    1. AllowDuplicates should be set to No. AlternateKey should be set to Yes.
    2. ValidTimeStateKey should be set to Yes,
    3. ValidTimeStateMode should be set to Gap.

The **Position** and **ValidFrom** fields should appear under the **PositionIdx** node. The following section discusses the need for **ValidTo** in the **ValidTimeStateKey** index and offers a performance recommendation.

For new tables, new development, or major refactoring of a date effective-related feature, it is a best practice to add **ValidTo** to the **ValidTimeStateKey** index and on the **Properties** tab to select **Yes** from the **Included Column** drop-down list.



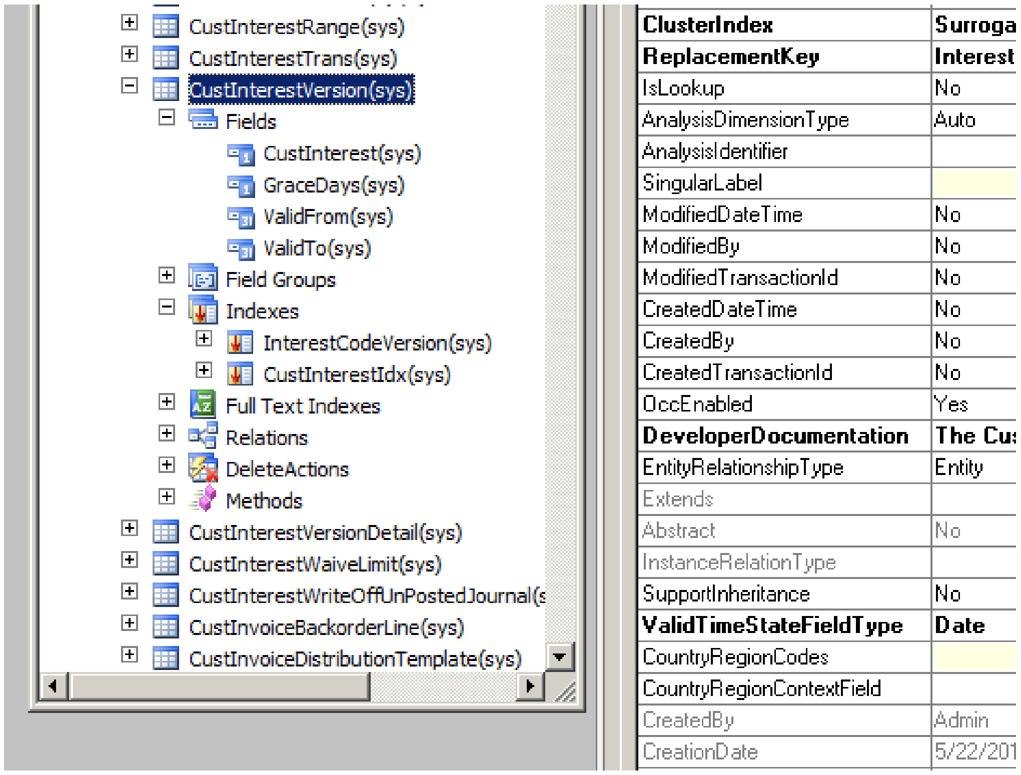
ID	1
Name	**PositionIdx**
AllowDuplicates	**No**
Enabled	Yes
ConfigurationKey	
AlternateKey	**Yes**
ValidTimeStateKey	**Yes**
ValidTimeStateMode	**Gap**
Origin	{2B0B0001-1873-1000-
LegacyId	**1**
Model	Foundation

Figure 3: PositionIdx Properties

The **CustInterestVersion** table defines the schema for storing the interest rates and their effective date time periods. This table does not allow gap.

To enable the **CustInterestVersion** table to be a valid time state table, the following properties in **AOT** must be set:

1. Launch the Microsoft Dynamics AX 2012 development workspace.
2. In the AOT, navigate to **Data Dictionary** > **Tables**.
3. Select the **CustInterestVersion** table.
4. On the Properties tab, ValidTimeStateFieldType should be set to Date.
5. Validate that the ValidFrom and ValidTo fields added by system have the type of Date.

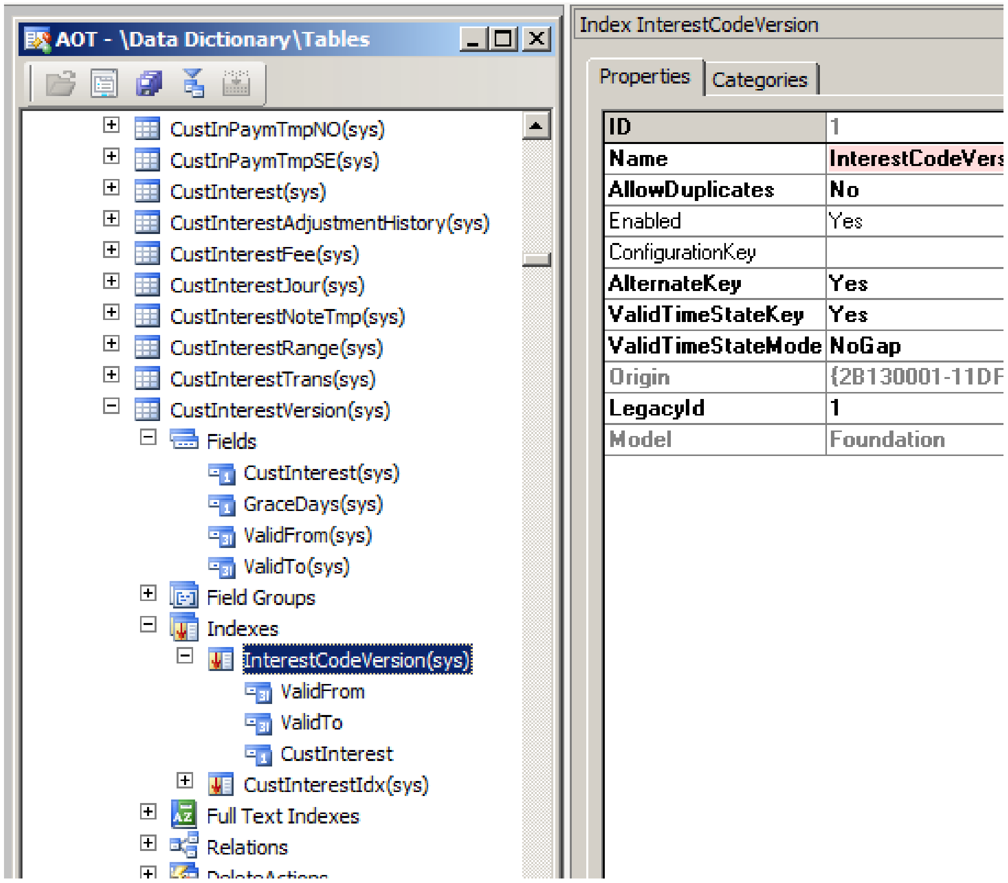


```
+ CustInterestRange(sys)
+ CustInterestTrans(sys)
- CustInterestVersion(sys)
    - Fields
          CustInterest(sys)
          GraceDays(sys)
          ValidFrom(sys)
          ValidTo(sys)
    + Field Groups
    - Indexes
        + InterestCodeVersion(sys)
        + CustInterestIdx(sys)
    + Full Text Indexes
    + Relations
    + DeleteActions
    + Methods
+ CustInterestVersionDetail(sys)
+ CustInterestWaiveLimit(sys)
+ CustInterestWriteOffUnPostedJournal(s
+ CustInvoiceBackorderLine(sys)
+ CustInvoiceDistributionTemplate(sys)
```

ClusterIndex	**Surroga**
ReplacementKey	**Interest**
IsLookup	No
AnalysisDimensionType	Auto
AnalysisIdentifier	
SingularLabel	
ModifiedDateTime	No
ModifiedBy	No
ModifiedTransactionId	No
CreatedDateTime	No
CreatedBy	No
CreatedTransactionId	No
OccEnabled	Yes
DeveloperDocumentation	**The Cu:**
EntityRelationshipType	Entity
Extends	
Abstract	No
InstanceRelationType	
SupportInheritance	No
ValidTimeStateFieldType	**Date**
CountryRegionCodes	
CountryRegionContextField	
CreatedBy	Admin
CreationDate	5/22/201

6. In the AOT, expand the **Indexes** node, and then select **InterestCodeVersion**.

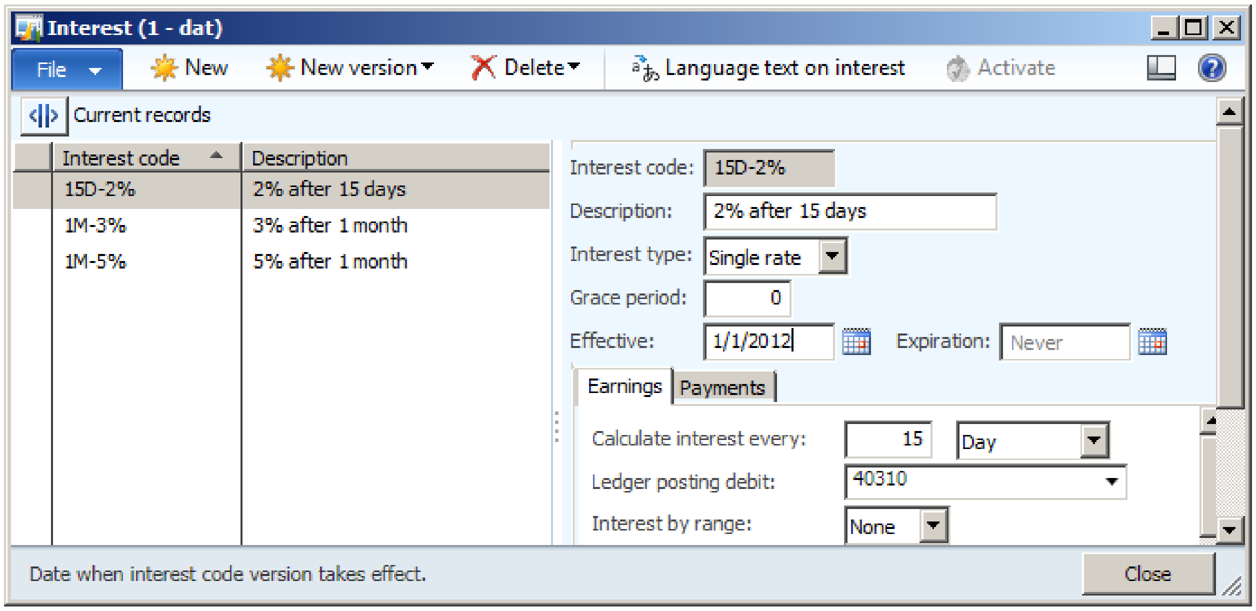
7. On the **Properties** tab:
    1. AllowDuplicates should be set to No.
    2. AlternateKey should be set to Yes.
    3. ValidTimeStateKey should be set to Yes
    4. ValidTimeStateMode should be set to NoGap.
    5. The CustInterest and ValidFrom fields should appear under the InterestCodeVersion node.



ID	1
Name	InterestCodeVers
AllowDuplicates	No
Enabled	Yes
ConfigurationKey	
AlternateKey	Yes
ValidTimeStateKey	Yes
ValidTimeStateMode	NoGap
Origin	{2B130001-11DF
LegacyId	1
Model	Foundation

Date vs. UtcDateTime

For the valid time state tables that have the property **ValidTimeStateFieldType** set to **Date**, the **ValidFrom** and **ValidTo** fields added by the system are type Date. The date value entered by a user on a form is stored in the database without any time zone conversion. For example, suppose a user enters the data shown on the **Interest** form in Figure 4. This data is stored in the **CustInterestVersion** table.



Interest code	Description
15D-2%	2% after 15 days
1M-3%	3% after 1 month
1M-5%	5% after 1 month

Interest code: 15D-2%
Description: 2% after 15 days
Interest type: Single rate
Grace period: 0
Effective: 1/1/2012
Expiration: Never

Earnings | Payments

Calculate interest every: 15 Day
Ledger posting debit: 40310
Interest by range: None

Date when interest code version takes effect.

Figure 4: Interest form

The data shown in the form means that if a customer selects the interest "15D-2%", the interest rate 2% will be effective on 1/1/2012, no matter where in the world the customer is.

For the valid time state tables that have the property **ValidTimeStateFieldType** set to **UtcDateTime**, the **ValidFrom** and **ValidTo** fields added by the system are type of **UtcDateTime** in Microsoft Dynamics AX 2012, which has the granularity of a second. The date time value entered by a user will be converted to the UTC time using the user preferred time zone and stored in the database. This means that a record will be effective at the same UTC time everywhere in the world, and it will expire at the same UTC time.

When updating records with the **CreateNewTimePeriod** mode, which is described in the following sections, the **ValidFrom** or **ValidTo** data for the records are updated with the current time or date by the system. The current time and date values are read from the machine clock on the AOS to which the user client is connected.

Inclusivity of ValidFrom and ValidTo

Date effective data records use the value in fields **ValidFrom** and **ValidTo** to determine the time period that they are effective, and follow the [closed, closed] representation, meaning that both the **ValidFrom** date and **ValidTo** date are inclusive in the date range. For example, suppose the HcmPositionWorkerAssignment table has the following data:

Position	Worker	ValidFrom	ValidTo
Upgrade	Anders Jensen	5/30/2000 00:00:00 am	12/31/2002 00:00:00am
Upgrade	Emma Baker	1/1/2003 00:00:00 am	12/31/2005 00:00:00 am

Anders Jensen filled the position Upgrade from 5/30/2000 to 12/31/2002, inclusive. After that, Emma Baker filled the position on 1/1/2003, and then left the position on 12/31/2005.

Overlap and Gap

Two rows cannot *overlap* if they have the same alternate key. However, date overlap can occur if the alternate keys are different. For example, in the HcmPositionWorkerAssignment table, two workers cannot be assigned to one position at the same time, hence the following data is **not allowed:**

Position	Worker	ValidFrom	ValidTo
Upgrade	Anders Jensen	5/30/2000 00:00:00 am	12/31/2002 23:59:59 pm
Upgrade	Emma Baker	1/1/2001 00:00:00 am	12/31/2005 23:59:59 pm

Gap exists when date continuum is broken between the ValidTo date and the ValidFrom date on two consecutive records. In an HcmPositionWorkerAssignment table that allows gap, the following data can exist, but the data cannot exist if the table does not allow gap:

Position	Worker	ValidFrom	ValidTo
Upgrade	Anders Jensen	5/30/2000 00:00:00 am	12/31/2002 23:59:59 pm
Upgrade	Emma Baker	1/1/2005 00:00:00am	12/31/2005 23:59:59 pm

Integration with table inheritance

Valid time state tables support table inheritance. However, the properties that are related to the valid time state tables can only be set on the root table.

Configuration key

Configuration keys cannot be set on the ValidFrom and ValidTo fields because Microsoft Dynamics AX 2012 needs them to track the date or date time change in the records. We recommend that you do not set configuration keys on any of the ValidTimeStateKey index columns.

Performance considerations when designing valid time state tables

To help improve performance of valid time state tables, you should index them correctly. Valid time state tables are modeled with an alternate key that includes the **ValidFrom** column. In some models, the **ValidTo** column may have also be included in the alternate key, but this is not necessary for uniqueness, and it should be removed from the alternate key constraint. If the **ValidFrom** column is a key column of the clustered index, the **ValidTo** column should not also be a key column of the clustered index. If the **ValidFrom** column is a key column of a non-clustered index, the **ValidTo** column should be made an **included** column in the non-clustered index, which provides coverage for range queries that involve both **ValidTo** and **ValidFrom** columns.

Programming with valid time state tables

Because overlap is not allowed in the valid time state tables, when a record is inserted or updated, other records may be automatically adjusted by Microsoft Dynamics AX 2012 to enforce the no overlap rule. However, in some cases, errors are issued if the inserts or updates are not allowed. Specific cases are described in the tables in the following sections.

Inserting data into a valid time state table

Suppose R1, R2, R3, and R4 are records that exist in a valid time state table. Inserting a new record at the beginning or end of the table is allowed. Inserting a new record in the middle of a record or inserting a record that overlaps with multiple records is not allowed. For example:

Allowed: Insert a new record with a ValidFrom and ValidTo date that precedes the first record

```
|---| ←New record
      |_____|_____|_____|____|
         R1     R2    R3     R4
```

Consequence: Update the ValidFrom date on R1 to match the ValidTo date on the new record.

- **If gap is not allowed:** The ValidFrom date on R1 changes to 12/31/1999 to match the ValidTo date on the new record.
- **If gap is allowed:** The R1 ValidFrom date remains unchanged (01/01/2000).

Before inserting the new record:

	ValidFrom	ValidTo
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

New record:

	ValidFrom	ValidTo
New	01/01/1999	12/30/1999

After inserting the new record (no gap):

	ValidFrom	**ValidTo**
New	01/01/1999	12/30/1999
R1	**12/31/1999**	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

Allowed: Insert a new record with a ValidTo date that overlaps the ValidFrom date of the first record

```
|---|
  |_____|_____|_____|____|
      R1      R2    R3      R4
```

Consequence: A new record is inserted. The ValidTo date overlaps with the ValidFrom date on R1. Update the ValidFrom date on R1 to match the ValidTo date on the new record.

Before inserting the new record:

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

New record:

	ValidFrom	**ValidTo**
New	01/01/1999	05/01/2000

After inserting the new record:

	ValidFrom	**ValidTo**
New	01/01/1999	05/01/2000
R1	**05/02/2000**	01/01/2001
R2	01/01/2001	01/01/2002
R3	01/01/2002	01/01/2003
R4	01/01/2003	01/01/2154

Allowed: Insert a new record with a ValidFrom and ValidTo date that follows the ValidTo date of the last record

```
                                          |--|
|_____|_____|_____|____|
   R1      R2    R3       R4
```

Consequence: A new record is inserted following R4.

- **If gap is not allowed:** The ValidTo date on R4 changes to 12/31/2008 to match the ValidTo date on the new record.
- **If gap is allowed:** The ValidTo date on R4 remains unchanged (01/01/2008).

Before inserting the new record:

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2008

New record:

	ValidFrom	**ValidTo**
New	01/01/2009	01/01/2154

After inserting the new record (no gap):

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	**12/31/2008**
New	01/01/2009	01/01/2154

Not Allowed: A new record is inserted that includes ValidFrom and ValidTo dates that are within the date range of an existing record.

```
             |--|
|_____|_____|_____|____|
    R1      R2     R3       R4
```

Consequence: If you attempt to insert a record that has a ValidFrom and ValidTo that are within the range of R2, the record is not inserted, and an error message is returned.

Before inserting the new record:

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

New record:

	ValidFrom	**ValidTo**
New	03/01/2001	06/01/2001

Result: No change.

Not Allowed: A new record is inserted that spans the date range of more than one other record.

```
              |-----------------|
|_____|_____|_____|____|
    R1       R2     R3        R4
```

Consequence: If you attempt to insert a record with a date range that spans R2, R3, and R4, the record is not inserted, and an error message is returned.

Before inserting the new record:

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

New record:

	ValidFrom	**ValidTo**
New	06/01/2001	06/01/2004

Result: No change.

Update data in a valid time state table

There are three modes for updating a record in a valid time state table: **Correction** mode, **CreateNewTimePeriod** mode, and **EffectiveBased** mode.

In **Correction** mode, updating a record in a valid time state table is a similar activity to updating a record in a non-valid time state table. Examples are described in the Correction mode section.

In **CreateNewTimePeriod** mode, only the records that are currently effective can be updated. Also, the existing current record will be expired, and a new record will be created and it will be set to effective. The behavior for expiring and newly inserted a record depends on the **ValidTimeStateFieldType** value:

- **UtcDateTime**: The **ValidTo** date on the expired record will be set to the current time minus 1 second, and the **ValidFrom** date on the new record will be set to the current time.
- **Date**: The **ValidTo** date on the expired record will be set to the current date minus 1 day, and the **ValidFrom** date on the new record will be set to the current date.

For example, suppose the **CustInterestVersion** table has the following data:

Interest code	Grace Period	ValidFrom	ValidTo
15D2%	0	1/1/2012	12/31/2154

On 5/31/2012, if the grace period is updated to 15, in the **CreateNewTimePeriod** mode, the table will have the following data:

Interest code	Grace Period	ValidFrom	ValidTo
15D2%	0	1/1/2012	**5/30/2012**
15D2%	**15**	**5/31/2012**	**12/31/2154**

EffectiveBased mode is the combination of **CreateNewTimePeriod** and **Correction** modes. If a current effective record is updated in **EffectiveBased** mode, it behaves the same as in **CreateNewTimePeriod** mode. If a record that will be effective in the future is updated in the **EffectiveBased** mode, it behaves the same as in **Correction** mode. Updating a record that was effective in the past in **EffectiveBased** mode is not allowed.

Correction mode examples

Suppose R1, R2, R3, and R4 are records that exist in a valid time state table. **Correction** can be used for past, current, and future records.

Allowed: Update the ValidFrom date on the second record in a table

```
      |--
|_____|_____|_____|____|
    R1      R2     R3      R4
```

Consequence: Update the ValidTo date on R1 to match the new ValidFrom date on R2.

Before updating R2:

	ValidFrom	ValidTo
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

Update R2: The new ValidFrom date is 06/01/2000.

After updating R2:

	ValidFrom	**ValidTo**
R1	01/01/2000	**05/31/2000**
R2	**06/01/2000**	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

Allowed: Update the ValidTo date on the first record in a table

```
     --|←
|_____|_____|_____|____|
    R1      R2     R3      R4
```

Consequence if gap is not allowed:

Update the ValidFrom date on R2 to match the new ValidTo date on R2. In this example, the new R2 ValidFrom date is 06/02/2000.

Consequence if gap is allowed:

No update to R2 because the gap exists.

Before updating R1:

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

Update R1:

The new ValidTo date is 06/01/2000.

After updating R1 (no gap):

	ValidFrom	**ValidTo**
R1	01/01/2000	**06/01/2000**
R2	**06/02/2000**	01/01/2002
R3	01/02/2002	01/01/2003

R4	01/02/2003	01/01/2154

Allowed: Update the ValidTo date on the last record in a table

```
                                        ---|
|______|______|______|_____|
   R1      R2     R3       R4
```

Consequence: The ValidTo date on R4 is changed. No other records are affected.

Before updating R4:

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2009

Update R4: The new ValidTo date is 01/01/2010.

After updating R4:

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	**01/01/2010**

Allowed: Update the ValidFrom date on the first record in a table

```
|---
   |______|______|______|_____|
      R1      R2     R3       R4
```

Consequence: Update the ValidFrom date on R1 to a new value.

Before updating R1:

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

Update R1: The new ValidFrom date is 01/01/1999.

After updating R1:

	ValidFrom	**ValidTo**
R1	**01/01/1999**	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

Not Allowed: Update the ValidFrom date on a record in a table to be earlier than the ValidFrom date on an earlier record

```
        |-----------
|_____|_____|_____|____|
   R1      R2     R3       R4
```

Consequence: If you attempt to update the ValidFrom date on R3 to a value that is earlier than the ValidFrom date on R2, the record is not inserted, and an error message is returned.

Before updating R3:

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

Update R3:

The new ValidFrom date is 06/01/2000.

Result: No change.

Not Allowed: Update the ValidTo date on a record in a table to be later than the ValidTo date on a later record

```
                  ------------|
|_____|_____|_____|____|
     R1       R2     R3        R4
```

Consequence: If you attempt to update the ValidTo date on R2 to a value that is later than the ValidTo date on R3, the record is not updated, and an error message is returned.

Before updating R2:

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

UpdateR2:

The new ValidTo date is 06/01/2003.

Result: No change.

Not Allowed: Update the ValidFrom and ValidTo dates on a record in a single operation

```
              |---          ----|
|_____|_____|_____|____|
     R1       R2     R3        R4
```

Consequence: If you update the ValidFrom date on R3 to a new value and update the ValidTo date on R3 to a new value in a single operation, no changes are made, and an error message is returned

Before updating R3:

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002

R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

Update R3:

The new ValidFrom date is 06/01/2001.

The new ValidTo date is 06/01/2003.

Result: No change.

Delete data from a valid time state table

Suppose R1, R2, R3, and R4 are records that are stored in a valid time state table. When deleting a record in a valid time state table that does not allow gaps, the ValidTo field of an earlier record may be updated so that a gap does not appear. If the table allows gap, no other records will be adjusted. The following table shows the data change.

Allowed: Delete a record from a valid time state table

```
|_____|_____|__x___|____|
   R1     R2    R3     R4
```

Delete R3.

Consequence if gap is not allowed: Update the ValidTo date on R2 to match the ValidFrom date on R4. The new ValidTo date on R2 is 01/01/2003.

Consequence if gap is allowed: No update to R2 because the gap exists. The ValidTo date on R2 remains 01/01/2002.

Before deleting R3:

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

After deleting R3 (no gap):

	ValidFrom	**ValidTo**

R1	01/01/2000	01/01/2001
R2	01/02/2001	**01/01/2003**
R4	01/02/2003	01/01/2154

Allowed: Delete the current record from a valid time state table

```
|_____|_____|_____|__x__|
   R1     R2    R3     R4
```

R4 is the current record. Delete R4.

Consequence: R4 is deleted. No automatic adjustments are made to the ValidTo date of the preceding record.

Before deleting R4:

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003
R4	01/02/2003	01/01/2154

After deleting R4:

	ValidFrom	**ValidTo**
R1	01/01/2000	01/01/2001
R2	01/02/2001	01/01/2002
R3	01/02/2002	01/01/2003

Query data from a valid time state table

Valid time state tables can be queried through an X++ statement and the Query API. In an X++ statement, keyword **ValidTimeState** is added to query records that are effective at a specific time or in a date range. For the Query API, four new methods are added to achieve the same results as an X++ statement. The following section demonstrates the usage in more detail. For the purposes of this example, we assume that the **CustInterestVersion** table has the following data:

Interest code	Grace Period	ValidFrom	ValidTo
1M-5%	0	1/1/2001	12/31/2002
1M-5%	0	1/1/2003	12/31/2012
1M-5%	0	1/1/2013	12/31/2154
1M-3%	0	1/1/1900	12/31/2154
15D-2%	0	1/1/1900	12/31/2154

We assume that the **HcmPositionWorkerAssignment** table has the following data:

Position	Worker	ValidFrom	ValidTo
10	AJE	5/31/2000 05:00:00 am	12/31/2154 06:00:00 am
11	AJE	5/14/1995 05:00:00 am	5/31/2000 05:00:00 am
12	EPE	12/31/1999 06:00:00 am	12/31/2154 06:00:00 am
13	EWA	12/31/2000 06:00:00 am	12/31/2154 06:00:00 am
15	EWA	6/30/2000 05:00:00 am	6/30/2000 05:00:00 am
16	EWA	4/30/1996 05:00:00 am	6/30/2000 05:00:00 am

X++ statement

There are three modes to query data effective records: **Default** mode, **AsOfDate** mode, and **AsofDateRange** mode. In **Default** mode, no special keyword is needed for the select statement on a valid time state table, and the current records will be returned. This is achieved by the system implicitly appending the following to the where clause:

```
ValidFrom <= today && ValidTo >= today
```

For example, the following code sample returns the records that are currently effective:

```
static void QueryCurrent(Args _args)
{
    CustInterestVersion interestVersion;
    CustInterest        interest;

    while select * from interestVersion join interest
        where interestVersion.CustInterest == interest.RecId &&
        Interest.InterestCode == "1M-5%"
    {
        info(strFmt("%1, %2, %3, %4", interest.InterestCode, interestVersion.GraceDays,
interestVersion.ValidFrom, interestVersion.ValidTo));
    }
}
```

The following data is returned if the code is executed on 5/18/2012:

Interest code	Grace Period	ValidFrom	ValidTo
1M-5%	0	1/1/2003	12/31/2012

In **AsOfDate** or **AsOfDateRange** mode, a new keyword **ValidTimeState** is introduced. **ValidTimeState(date1)** can be used to query the records that are effective at a specific time. **ValidTimeState(date1, date2)** can be used to query the records that are effective during a time period. Date1 and Date2 can be the type of **Date** or **UtcDateTime**, depending on the **ValidFrom** and **ValidTo** type of the valid time state table. For example, the following code sample returns the records that are effective on 1/1/2002:

```
static void QueryCurrent(Args _args)
{
    CustInterestVersion interestVersion;
    CustInterest        interest;
    Date               asOfDate = 1\1\2002;

    while select validtimestate(asOfDate) * from interestVersion join interest
        where interestVersion.CustInterest == interest.RecID
    {
        info(strFmt("%1, %2, %3, %4", interest.InterestCode, interestVersion.GraceDays,
interestVersion.ValidFrom, interestVersion.ValidTo));
    }
}
```

The following data is returned:

Interest code	Grace Period	ValidFrom	ValidTo
1M-5%	0	1/1/2001	12/31/2002

The following code sample returns the records that are effective during 1/1/2009 and 1/1/2012:

```
static void QueryCurrent(Args _args)
{
    CustInterestVersion interestVersion;
    CustInterest        interest;
    Date               fromdate = 1\1\2009;
    Date                todate = 1\1\2012;

    while select validtimestate(fromdate, todate) * from interestVersion join interest
        where interestVersion.CustInterest == interest.RecID
    {
        info(strFmt("%1, %2, %3, %4", interest.InterestCode, interestVersion.GraceDays,
interestVersion.ValidFrom, interestVersion.ValidTo));
    }
}
```

The following data is returned:

Interest code	Grace Period	ValidFrom	ValidTo
1M-5%	0	1/1/2003	12/31/2012
1M-3%	0	1/1/1900	12/31/2154
15D-2%	0	1/1/1900	12/31/2154

The date or UtcDateTime value used in the keyword **ValidTimeState** must match the **ValidTimeStateFieldType** of the valid time state table. If the **ValidTimeStateFieldType** of a valid time state table is **Date**, the value in the **ValidTimeState** keyword must be type of **Date**. The same rule applies to the valid time state tables where the **ValidTimeStateFieldType** is set to **UtcDateTime**.

Query API

Just like the query in X++ using the select statement, four new methods have been added to query a valid time state table in **AsofDate** mode or **AsofDateRange** mode:

API	Description	ValidTimeStateFieldType
Query::ValidTimeStateAsOfDate(asOfDate)	Query records that are effective at asOfDate	Date
Query::ValidTimeStateDateRange(fromdate, todate)	Query records that are effective during fromdate and todate	Date
Query::ValidTimeStateAsOfDatetime(asOfdate time)	Query records that are effective at asOfdatetime	UtcDateTime
Query::ValidTimeStateDateTimeRange(fromda tetime, todatetime)	Query records that are effective during fromdatetime and todatetime	UtcDateTime

The following code samples are used to query the data that is effective on 1/1/2002:

```
static void QueryAsOfDateUsingQueryAPI(Args _args)
{
    Query q;
    QueryBuildDataSource qdbs;
    QueryRun qr;
    date asOfDate = 1\1\2002;
    CustInterestVersion version;

    q = new Query();
    qdbs = q.addDataSource(tableName2id("CustInterestVersion"));
    q.ValidTimeStateAsOfDate(asOfDate);
    qr = new QueryRun(q);

    while (qr.next())
    {
        version = qr.get(tableName2id("CustInterestVersion"));
        info(strFmt("%1, %2, %3", version.CustInterest, version.ValidFrom, version.ValidTo));
    }
}
```

The following code samples are used to query the data that is effective during 1995-01-01T00:00:00 and 1999-12-31T00:00:00:

```
static void QueryRangeUsingQueryApi(Args _args)
{
    Query q;
    QueryBuildDataSource qdbs;
    QueryRun qr;
    UtcDateTime fromdatetime = 1995-01-01T00:00:00;
    utcDateTime todatetime = 1999-12-31T00:00:00;
    HcmPositionWorkerAssignment assignment;


    q = new Query();
    qdbs = q.addDataSource(tableName2id("HcmPositionWorkerAssignment"));
    q.validTimeStateDateTimeRange(fromdatetime, todatetime);
    qr = new QueryRun(q);

    while (qr.next())
    {
        assignment = qr.get(tableName2id("HcmPositionWorkerAssignment"));
        info(strFmt("%1, %2, %3, %4", assignment.Position, assignment.Worker,
assignment.ValidFrom, assignment.ValidTo));
    }

}
```

AOT query

A valid time state table can be one of the data sources of an AOT query. Currently, there are no properties in the AOT query to specify the date range or AsOfDate for the records. This can be achieved programmatically or through form and **SysQueryForm**, which is demonstrated in the later section

View

A valid time state table can be one of the data sources in a view. By default, the view returns all data in a valid time state table. If the property **ValidTimeStateEnabled** is set to **Yes**, and the view fields contain the **ValidFrom** and **ValidTo** fields of the valid time state table in the view data source, the view returns current records.

Caching

Valid time state tables support all of the cache options. However, **EntireTableCache** is not recommended because valid time state tables usually change very frequently.

If effective valid time state table is queried in the **Default** mode to return a current record or in **AsOfDate** mode to return a record that is effective at a specific time, the record will be fetched from the database for the first time, and fetched from the cache for subsequent queries.

Set-based operations

Valid time state tables support set-based operations such as **Insert_recordset**, **update_recordset**, and **delete_from** by reverting them to record-by-record operations.

For example, suppose the **CustInterestVersion** table has the following data:

Interest code	Grace Period	ValidFrom	ValidTo
1M-5%	0	1/1/2001	12/31/2002
1M-3%	0	1/1/1900	12/31/20020

Suppose the **SourceTable** table has the following data:

Interest code	Grace Period	ValidFrom	ValidTo
1M-5%	0	1/1/2004	12/31/2154
1M-3%	30	1/1/2010	12/31/2154

Run the following code using update_recordset:

```
static void Insert_RecordSetPass(Args _args)
{
    CustInterestVersion target;
    SourceTable         source;

    insert_recordset target (CustInterest, GraceDays, ValidFrom, ValidTo)
        select CustInterest, GraceDays, ValidFrom, ValidTo from source

}
```

The **CustInterestVersion** table contains the following data:

Interest code	Grace Period	ValidFrom	ValidTo
1M-5%	0	1/1/2001	12/31/2003
1M-5%	30	1/1/2004	12/31/2154
1M-3%	0	1/1/1900	12/31/2009
1M-3%	30	1/1/2010	12/31/2154

However, if a record is inserted in the middle of another record or causes multiple records to overlap, the operation will fail. For example, the preceding code will fail if the SourceTable contains the following data:

Interest code	Grace Period	ValidFrom	ValidTo
1M-3%	0	1/1/2001	12/31/2002

For **Update_RecordSet** and **Delete_from**, the system will not append any condition for ValidFrom and ValidTo in the where clause. For example, the following code will change the ValidFrom for all records:

```
static void Update_RecordSetTest(Args _args)
{
    CustInterestVersion target;

    ttsbegin;
     target.validTimeStateUpdateMode(ValidTimeStateUpdate::Correction);
    update_recordset target
        setting ValidFrom = target.ValidFrom + 1

    ttscommit;
}
```

The following code will delete all of the records:

```
static void Update_RecordSetTest(Args _args)
{
    CustInterestVersion target;

    ttsbegin;
    delete_from target
    ttscommit;
}
```

Developing forms with valid time state tables

AOT properties

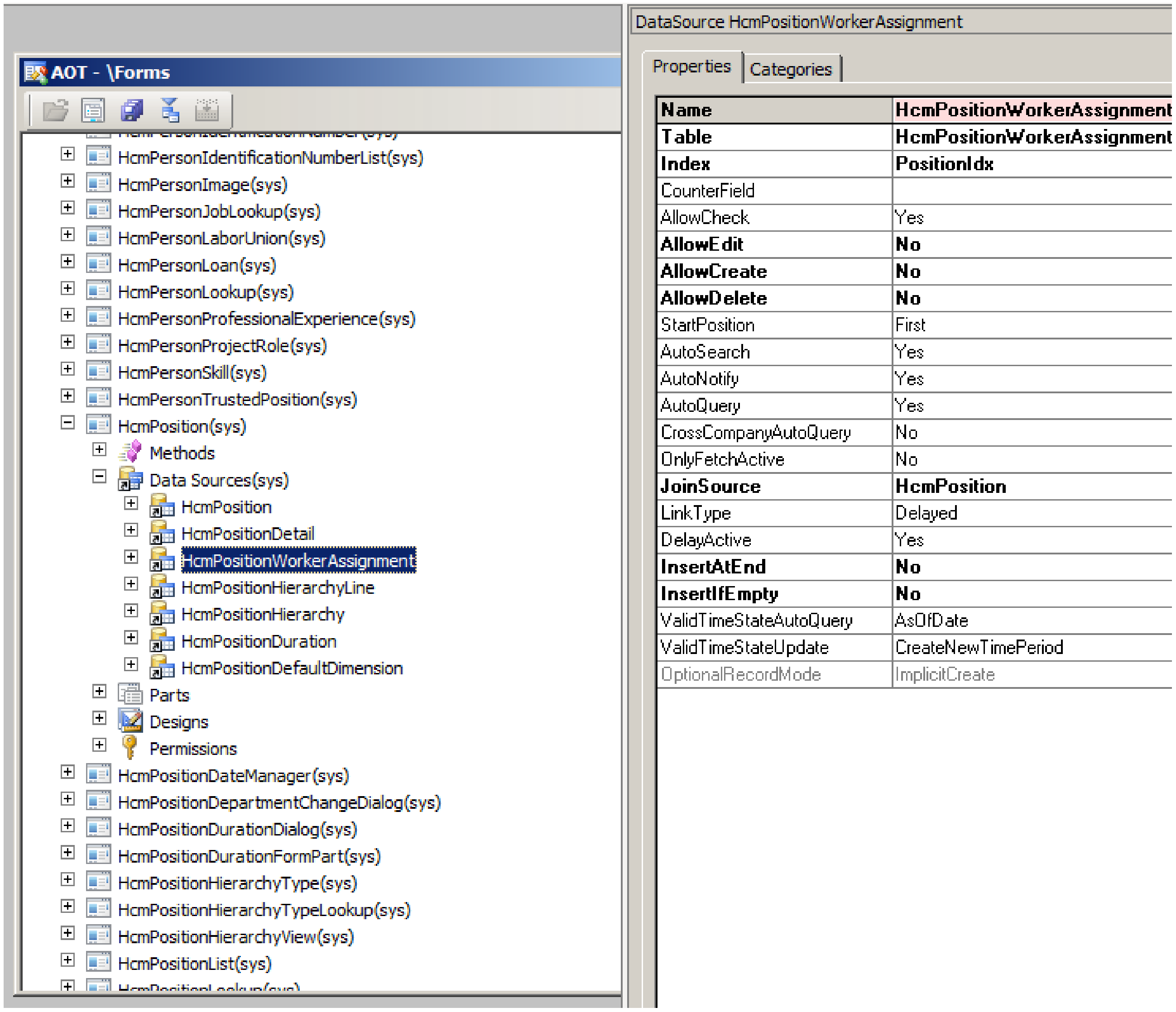
ValidTimeStateAutoQuery and **ValidTimeStateUpdate** are added to forms. **ValidTimeStateAutoQuery** allows user to access records that are effective at a specific time or during a time period.

ValidTimeStateUpdate allows user to update date effective records with different update modes.

ValidTimeStateAutoQuery can be set only on the master data source, which is the root table in a form or a query. It cannot be set on the inner/outer/exist/notexist joined child table. **ValidTimeStateUpdate** can only be set on data sources that reference a valid time state table. The data source could be one or more within a given form or query.

In **HcmPosition** form, the worker assignment section only shows the current worker who is assigned to the position. When a new worker is assigned to the position, the current worker will be expired from the position, which is achieved by using the **CreateNewTimePeriod** update mode.

The following screen shot shows the properties for the HcmPosition in AOT:



AOT - \Forms

- HcmPersonIdentificationNumberList(sys)
- HcmPersonImage(sys)
- HcmPersonJobLookup(sys)
- HcmPersonLaborUnion(sys)
- HcmPersonLoan(sys)
- HcmPersonLookup(sys)
- HcmPersonProfessionalExperience(sys)
- HcmPersonProjectRole(sys)
- HcmPersonSkill(sys)
- HcmPersonTrustedPosition(sys)
- HcmPosition(sys)
  - Methods
  - Data Sources(sys)
    - HcmPosition
    - HcmPositionDetail
    - HcmPositionWorkerAssignment
    - HcmPositionHierarchyLine
    - HcmPositionHierarchy
    - HcmPositionDuration
    - HcmPositionDefaultDimension
  - Parts
  - Designs
  - Permissions
- HcmPositionDateManager(sys)
- HcmPositionDepartmentChangeDialog(sys)
- HcmPositionDurationDialog(sys)
- HcmPositionDurationFormPart(sys)
- HcmPositionHierarchyType(sys)
- HcmPositionHierarchyTypeLookup(sys)
- HcmPositionHierarchyView(sys)
- HcmPositionList(sys)

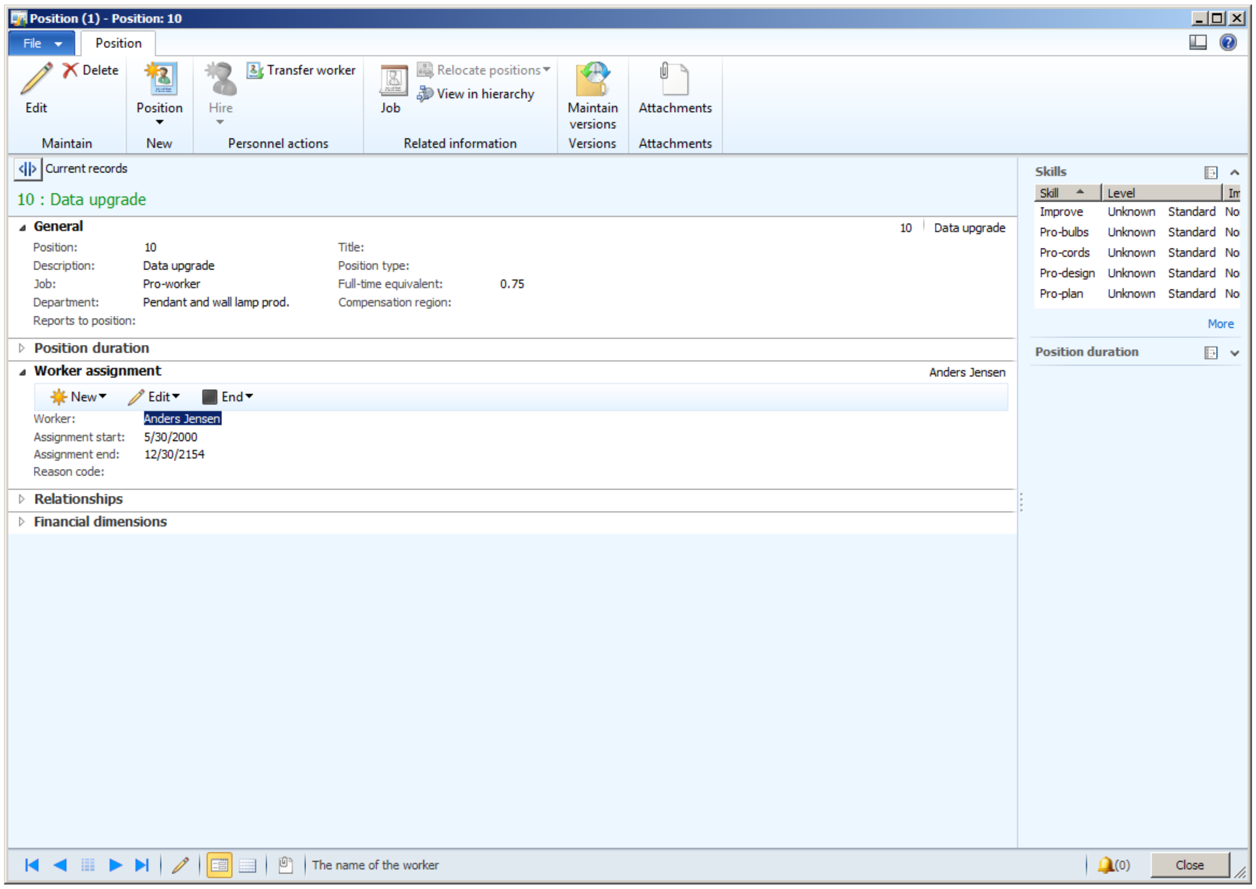
DataSource HcmPositionWorkerAssignment

Properties | Categories

Name	**HcmPositionWorkerAssignment**
Table	**HcmPositionWorkerAssignment**
Index	**PositionIdx**
CounterField	
AllowCheck	Yes
AllowEdit	**No**
AllowCreate	**No**
AllowDelete	**No**
StartPosition	First
AutoSearch	Yes
AutoNotify	Yes
AutoQuery	Yes
CrossCompanyAutoQuery	No
OnlyFetchActive	No
JoinSource	**HcmPosition**
LinkType	Delayed
DelayActive	Yes
InsertAtEnd	**No**
InsertIfEmpty	**No**
ValidTimeStateAutoQuery	AsOfDate
ValidTimeStateUpdate	CreateNewTimePeriod
OptionalRecordMode	ImplicitCreate

Figure 5: HcmPosition properties

When the HcmPosition form is opened, only the current record for a position is shown:



10 : Data upgrade

General 10 Data upgrade

Position:	10	Title:	
Description:	Data upgrade	Position type:	
Job:	Pro-worker	Full-time equivalent:	0.75
Department:	Pendant and wall lamp prod.	Compensation region:	
Reports to position:			

Position duration

Worker assignment Anders Jensen

New Edit End

Worker:	Anders Jensen
Assignment start:	5/30/2000
Assignment end:	12/30/2154
Reason code:	

Relationships

Financial dimensions

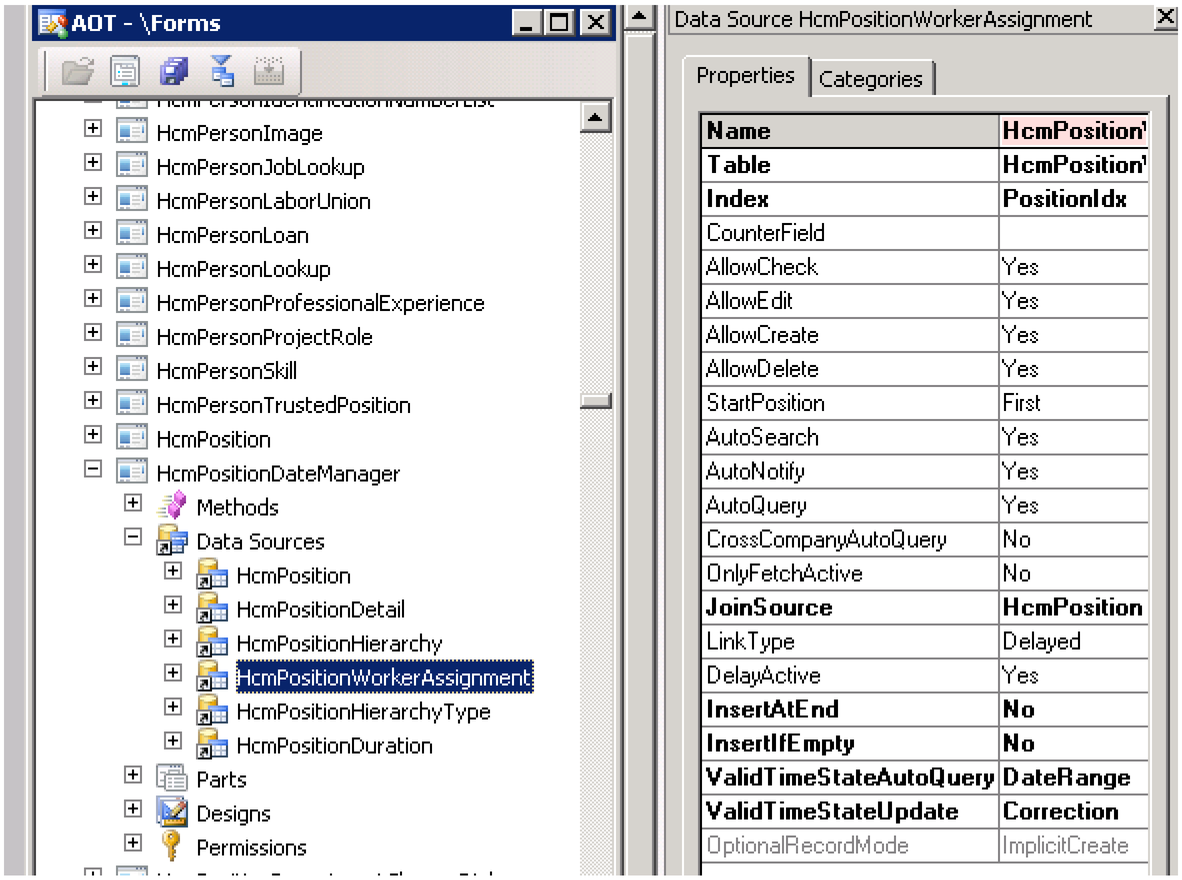
Skills

Skill	Level		Im
Improve	Unknown	Standard	No
Pro-bulbs	Unknown	Standard	No
Pro-cords	Unknown	Standard	No
Pro-design	Unknown	Standard	No
Pro-plan	Unknown	Standard	No

Position duration

Figure 6: Position form

In the **HcmPositionDateManager** form, all records are shown. When a record is updated, it is updated in **Correction** mode. Therefore, in this form for the data source **HcmPositionWorkerAssignment**, the **ValidTimeStateAutoQuery** property is set to **DateRange**, and the **ValidTimeStateUpdate** property is set to **Correction**.

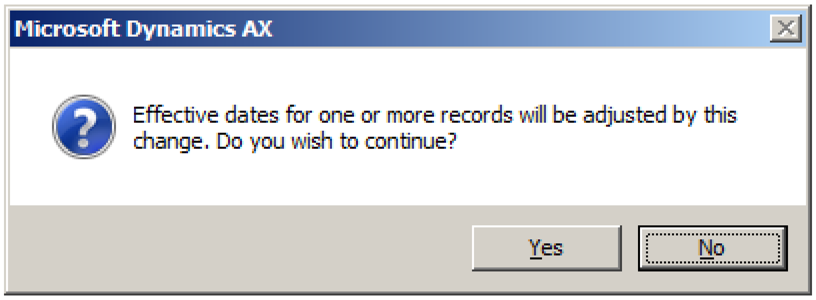


Name	**HcmPosition'**
Table	**HcmPosition'**
Index	**PositionIdx**
CounterField	
AllowCheck	Yes
AllowEdit	Yes
AllowCreate	Yes
AllowDelete	Yes
StartPosition	First
AutoSearch	Yes
AutoNotify	Yes
AutoQuery	Yes
CrossCompanyAutoQuery	No
OnlyFetchActive	No
JoinSource	**HcmPosition**
LinkType	Delayed
DelayActive	Yes
InsertAtEnd	**No**
InsertIfEmpty	**No**
ValidTimeStateAutoQuery	**DateRange**
ValidTimeStateUpdate	**Correction**
OptionalRecordMode	ImplicitCreate

Figure 7: HcmPositionWorkerAssignment properties

Records change notification

When inserting, updating, and deleting a record in a valid time state table through forms, if adjacent records will be adjusted, the following dialog will be displayed.



Effective dates for one or more records will be adjusted by this change. Do you wish to continue?

Figure 8: Effective date warning message

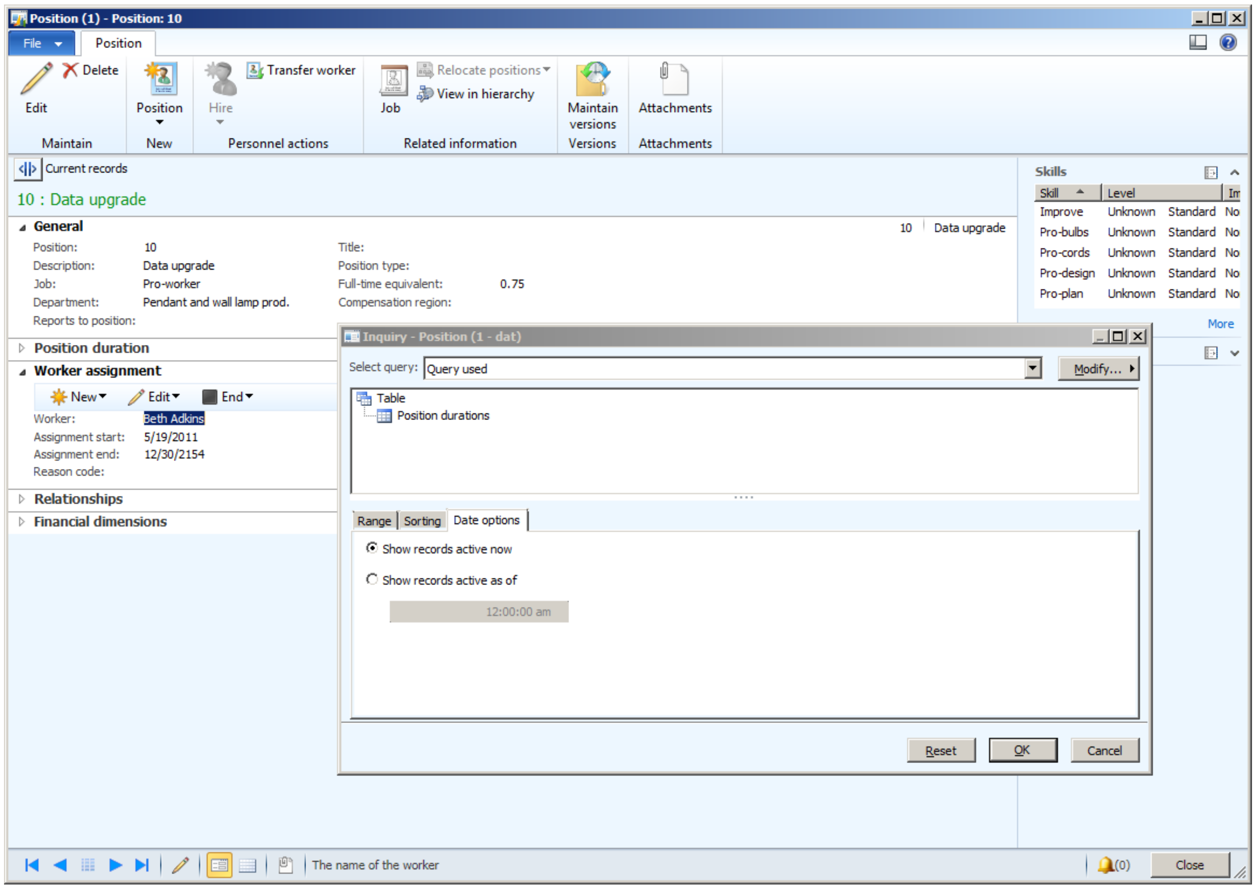
Click **Yes** to make the adjustment for the affected records.

Click **No** to not make the adjustment. The user needs to Restore (press Ctrl+F5) to discard the changes.

Filtering by using the Inquiry form (SysQueryForm)

SysQueryForm can be used to further filter the records in a form, report, or runbase dialog. For forms that have valid time state tables as the data source, a **Date options** tab is added to the SysQueryForm so that the records can be filtered by using date effective semantics.

For example, in the **HcmPosition** form where the **ValidTimeStateAutoQuery** is set to **AsOfDate**, the following **Inquiry** form will be displayed when the user opens the form and presses Ctrl+F3.



10 : Data upgrade

General

Position:	10	Title:	
Description:	Data upgrade	Position type:	
Job:	Pro-worker	Full-time equivalent:	0.75
Department:	Pendant and wall lamp prod.	Compensation region:	
Reports to position:			

10 Data upgrade

Skill	Level		Im
Improve	Unknown	Standard	No
Pro-bulbs	Unknown	Standard	No
Pro-cords	Unknown	Standard	No
Pro-design	Unknown	Standard	No
Pro-plan	Unknown	Standard	No

Worker: Beth Adkins
Assignment start: 5/19/2011
Assignment end: 12/30/2154
Reason code:

Select query: Query used

Table
Position durations

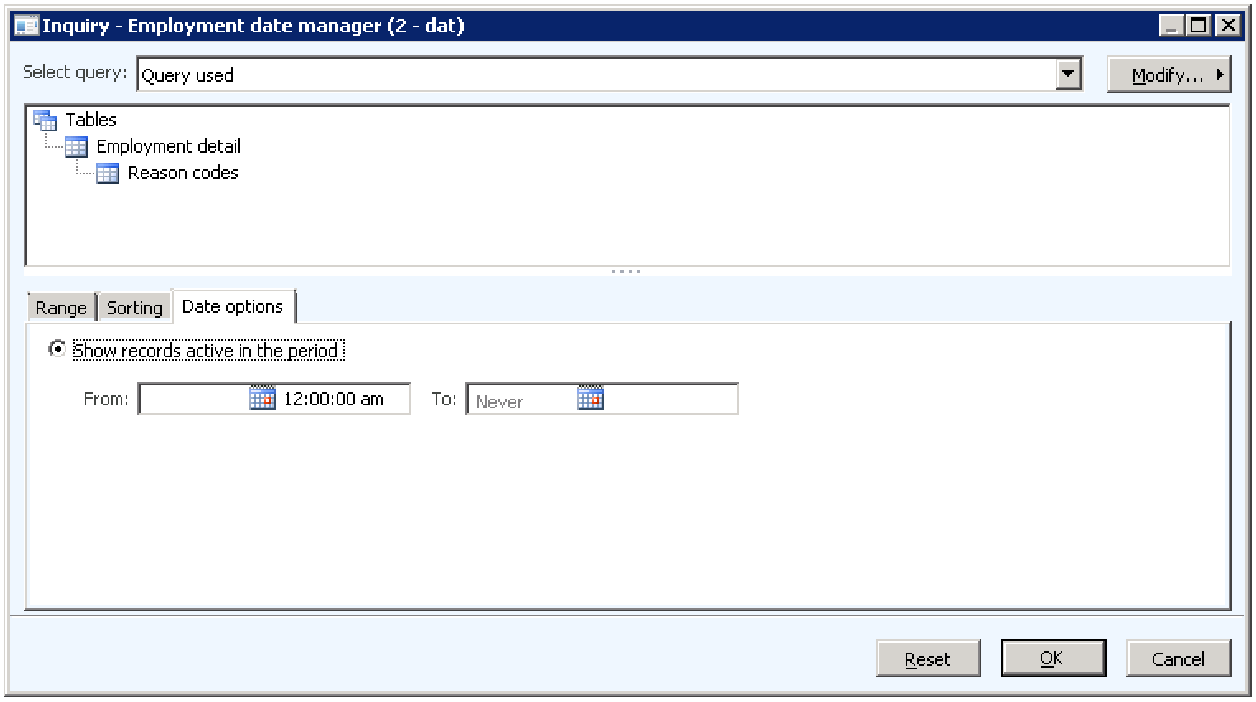
[✓ Show records active now]
Show records active as of
12:00:00 am

The name of the worker

Figure 9: Inquiry form from HCMPosition, in which ValidTimeStateAutoQuery is set to AsOfDate

To display the records that are currently effective, select **Show records active now**. Otherwise, select **Show records active as of**, and specify a date time.

For the forms in which **ValidTimeStateAutoQuery is** set to **Date Range,** pressing Ctrl+F3 will display the following **Inquiry** form.


Inquiry - Employment date manager (2 - dat)

Select query: Query used

Tables
Employment detail
Reason codes

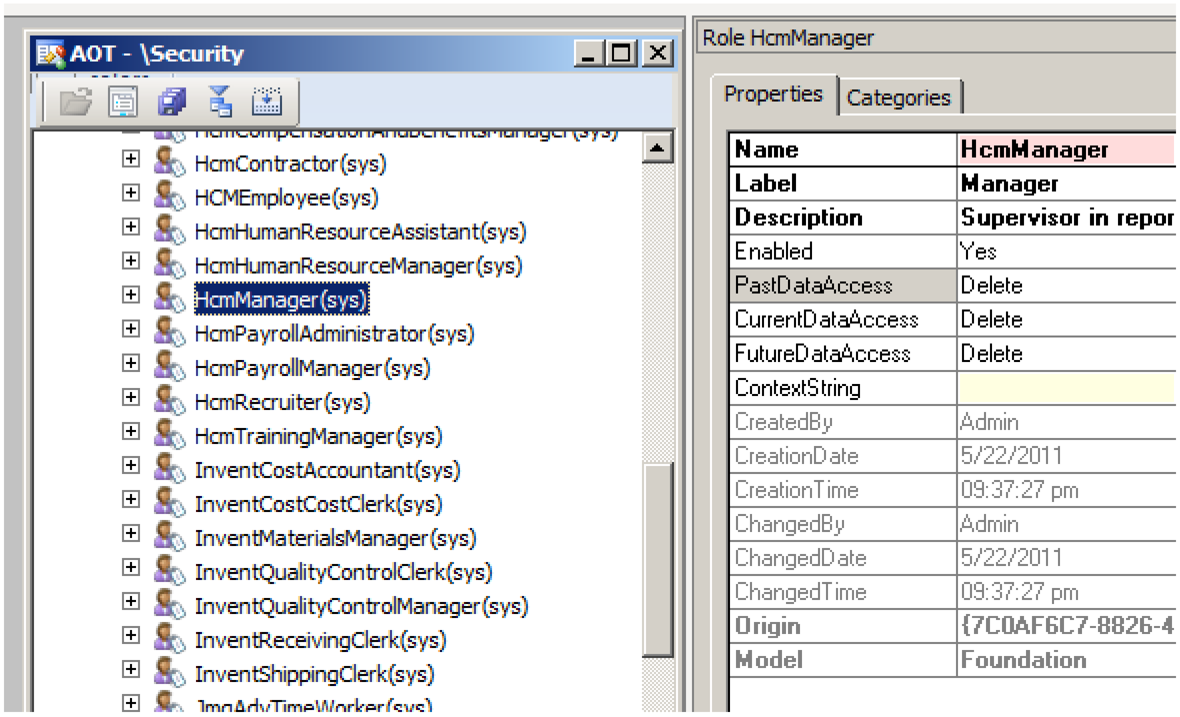
(●) Show records active in the period

From: 12:00:00 am To: Never

Figure 9: Inquiry form from Employment data manager, in which ValidTimeStateAutoQuery is set to Date Range

Security integration

The security framework in Microsoft Dynamics AX 2012 has been changed to a role-based security model. For valid time state tables, a role can be defined to have access to **current**, **past**, or **future** records for all data effective tables.



AOT - \Security
HcmContractor(sys)
HCMEmployee(sys)
HcmHumanResourceAssistant(sys)
HcmHumanResourceManager(sys)
HcmManager(sys)
HcmPayrollAdministrator(sys)
HcmPayrollManager(sys)
HcmRecruiter(sys)
HcmTrainingManager(sys)
InventCostAccountant(sys)
InventCostCostClerk(sys)
InventMaterialsManager(sys)
InventQualityControlClerk(sys)
InventQualityControlManager(sys)
InventReceivingClerk(sys)
InventShippingClerk(sys)
Role HcmManager
Properties
Categories

Name	**HcmManager**
Label	**Manager**
Description	**Supervisor in repor**
Enabled	Yes
PastDataAccess	Delete
CurrentDataAccess	Delete
FutureDataAccess	Delete
ContextString	
CreatedBy	Admin
CreationDate	5/22/2011
CreationTime	09:37:27 pm
ChangedBy	Admin
ChangedDate	5/22/2011
ChangedTime	09:37:27 pm
Origin	**{7C0AF6C7-8826-4**
Model	**Foundation**

Figure 10: Properties for HcmManager role

For example, for role **HcmManager**, **PastDataAccess**, **CurrentDataAccess**, and **FutureDataAccess** are used to define the effective access for the past records, the current records, and the future records. Each property can have one of the following values: **No access**, **Read**, **Update**, **Create**, **Correct**, and **Delete**. Because there are three update modes for valid time state tables, **Update** access is granted for updating records in **CreateNewPeriod** mode. **Correct** access is granted for updating records in **Correction** or **EffectiveBased** mode.

Valid time state tables can also have effective access set in a permission which is a part of a privilege that is assigned to a role. The effective access for a table in a permission also can have one of the following values: **No access**, **Read**, **Update**, **Create**, **Correct**, and **Delete**. Permission is granted at the minimum level of the effective access on a table and the effective access defined in **PastDataAccess**, **CurrentDataAccess**, and **FutureDataAccess** on the role.

For example, suppose a role has **PastDataAccess** set to **Read**, **CurrentDataAccess** set to **Update**, and **FutureDataAccess** set to **Delete**. The role has a privilege that contains a permission that has the valid time state table with effective access set to **Correct**. The role will have **Read** access on past records, **Update** access on current records, and **Correct** access on future records.

If a user has multiple roles, the maximum level of permission granted for all roles that the user has will be used. For example, suppose a user has two roles. The first role is described in the previous paragraph. The second role has **PastDataAccess** set to **No Access**, **CurrentDataAccess** set to **No Access**, and **FutureDataAccess** set to **No Access**. The user will have **Read** access on past records, **Update** access on current records, and **Correct** access on future records.

X++ behavior

The following table defines whether a user action is allowed in X++ when an access right is granted to a role:

Permission on table	Create/Read/Update/Delete operations allowed
No Access	No Access
Read	Read (Query the record)
Update	Update in CreateNewTimePeriod mode, Read
Create	Create (Insert record), Update, Read
Correct	Update in Correction mode, Create, Update, Read
Delete	Delete, Correct, Create, Update, Read

The permissions described in the table combine with the settings in **PastDataAccess**, **CurrentDataAccess**, and **FutureDataAccess** to further secure the data at the record level.

For example, suppose a worker is not allowed to view the positions assigned to her in the future, the role that the worker uses should have **FutureDataAccess** set to **No Access**, **PastDataAccess** set to **Read**, and **CurrentDataAccess** set to **Read**.

A Human resources employee should be allowed to update the currently effective positions to a new worker and correct a future position, but should not be allowed to change a past position. The role that the employee uses can have **PastDataAccess** set to **Read**, **CurrentDataAccess** set to **Update**, and **FutureDataAccess** set to **Correct**. For the **HcmManager** role, she can access any records.

When accessing data through X++, **table::RecordLevelSecurity()** must be used to enforce the security check. For example, if a role has the **PastDataAccess** set to **NoAccess**, **CurrentDataAccess** set to **Update**, and **FutureDataAccess** set to **Read**, the following code sample returns the records that are effective from now to 12/31/2154:

```
public static void Main(Args args)
{
    HcmPositionWorkerAssignment assignment;
    utcDateTime fromtime = 1900-01-01T00:00:00;
    utcDateTime totime = 2154-12-31T00:00:00;

    assignment.recordLevelSecurity(true);
    while select validTimeState(fromtime, totime) * from assignment
    {
        info(strFmt("%1, %2, %3, %4",
                    assignment.Position,
                    assignment.Worker,
                    assignment.ValidFrom,
                    assignment.ValidTo));
    }
}
```

Form behavior

When accessing records through a form, the Create/Read/Update/Delete operations are allowed.

Form Mode	Permission on table	Create/Read/Update/Delete operations allowed	Comments
Correction	No Access	No Access	
Correction	Read	Read	
Correction	Update	Read	
Correction	Create	Read	
Correction	Correct	Correct (Update in Correction mode), Create, Update (Update in CreateNewTimePeriod mode), Read	
Correction	Delete	Delete, Correct, Create, Update, Read	
CreateNewTimePeriod	No Access	No Access	
CreateNewTimePeriod	Read	Read	
CreateNewTimePeriod	Update	Update in CreateNewTimePeriod mode, Read	Only applicable on current records.
CreateNewTimePeriod	Create	Create, Update, Read	
CreateNewTimePeriod	Correct	Create, Update, Read	
CreateNewTimePeriod	Delete	Delete, Create, Update, Read	
EffectiveBased	No Access	No Access	
EffectiveBased	Read	Read	
EffectiveBased	Update	Update in CreateNewTimePeriod mode, Read	Only applicable on current records.
EffectiveBased	Create	Update, Read	
EffectiveBased	Correct	Update in Correction mode, Update, Read	Correction mode only applicable on future records. Update only applicable on current records.
EffectiveBased	Delete	Delete, Correct, Create, Update, Read	

Microsoft Dynamics is a line of integrated, adaptable business management solutions that enables you and your people to make business decisions with greater confidence. Microsoft Dynamics works like and with familiar Microsoft software, automating and streamlining financial, customer relationship and supply chain processes in a way that helps you drive business success.

U.S. and Canada Toll Free 1-888-477-7989

Worldwide +1-701-281-6500

www.microsoft.com/dynamics