**Deliverable 1 – Database Design:**

Please look at pages Database Design -1 and Database Design -2 in the attached Visio document.

For the different asset classes, I am using super type/sub type pattern.

The Super type is the SecurityMaster with all the common fields for any security.

--SecurityMaster table

CREATE TABLE SecurityMaster (

MasterSecID int NOT NULL PRIMARY KEY,

Ticker varchar(10),

Name varchar (600),

Currency varchar(5),

AssetType varchar(10),

CreatedDate datetime,

LastUpdateDate datetime,

CONSTRAINT secmaster\_superkey UNIQUE (MasterSecID, AssetType)

);

Currently, there are two sub-type tables – Future and Options.

--Future

CREATE TABLE Future (

MasterSecID int NOT NULL,

AssetType varchar(10) Default 'Future' CHECK (AssetType = 'Future'),

ContractSize int,

ExpirationDate datetime,

PRIMARY KEY (MasterSecID, AssetType),

FOREIGN KEY (MasterSecID, AssetType) REFERENCES SecurityMaster(MasterSecID, AssetType)

);

--Option

CREATE TABLE Options (

MasterSecID int NOT NULL,

AssetType varchar(10) Default 'Options' CHECK (AssetType = 'Options'),

ContractSize int,

ExpirationDate datetime,

StrikePrice numeric(28,12),

OptionType varchar(4),

Underlier int,

PRIMARY KEY (MasterSecID, AssetType),

FOREIGN KEY (MasterSecID, AssetType) REFERENCES SecurityMaster(MasterSecID, AssetType),

FOREIGN KEY (Underlier) REFERENCES SecurityMaster(MasterSecID)

);

The Check constraint makes sure that a sub-type table has only the asset type this table is meant for. This, together with the foreign key constraint makes sure that each row in this sub-type table refers only one row in the super table.

From the super type’s perspective, One MasterSecID (primary key on super type) can only appear in one of the two sub-type tables. So, a join between super-type and subtype table would return only one row.

The Daily Pricing and position information are captured in two separate tables below.

--DailyPrice

CREATE TABLE DailyPrice(

MasterSecID int NOT NULL,

PriceDate datetime NOT NULL,

ClosePrice numeric(24,12),

PRIMARY KEY (MasterSecID, PriceDate),

FOREIGN KEY (MasterSecID) REFERENCES SecurityMaster(MasterSecID)

)

--DailyPosition

CREATE TABLE DailyPosition(

MasterSecID int NOT NULL,

PositionDate datetime NOT NULL,

Quantity int,

PRIMARY KEY (MasterSecID, PositionDate),

FOREIGN KEY ( MasterSecID ) REFERENCES SecurityMaster (MasterSecID)

)

With the super type/sub-type model, we need to create views for each instrument to retrieve the common columns from the super type and the asset class specific columns from the sub type.

CREATE VIEW Future AS

SELECT [MasterSecID]

,[Ticker]

,[Name]

,[Currency]

,[AssetType]

,[ExpirationDate]

,[ContractSize]

,[CreatedDate]

,[LastUpdateDate]

FROM [dbo].[SecurityMaster] secmas

INNER JOIN [dbo].[Future] fut ON secmas.MasterSecID = fut.MasterSecID

CREATE VIEW Future AS

SELECT [MasterSecID]

,[Ticker]

,[Name]

,[Currency]

,underlier.[Ticker] as [Underlier]

,[AssetType]

,[ExpirationDate]

,[ContractSize]

,[StrikePrice]

,[OptionType]

,[CreatedDate]

,[LastUpdateDate]

FROM [dbo].[SecurityMaster] secmas

INNER JOIN [dbo].[Options] opt ON secmas.MasterSecID = opt.MasterSecID

LEFT OUTER JOIN [dbo].[SecurityMaster] undelier ON underlier.MasterSecID = opt.Underlier

CREATE VIEW Equity AS

SELECT [MasterSecID]

,[Ticker]

,[Name]

,[Currency]

,[AssetType]

,[CreatedDate]

,[LastUpdateDate]

FROM [dbo].[SecurityMaster]

**Question 1:**

**Discuss two different ways of saving the securities. What are the advantages and disadvantages of the two approaches?**

Above mentioned method is one way to save securities. There are two main aspects to observe in the above method.

* The external ticker symbol is not used as the primary key on the table. Instead, an integer identifier is used as the key.
* The Super-type/Sub-type model – modelling the tables more naturally and hierarchically without repeating a lot of data.

Advantages:

* With an integer identifier, a ticker change can be easily handled. Every other table will be referencing only the integer ID. No need to run batch queries to update ticker symbol across tables.
* Super type/sub-type pattern models the natural hierarchy in the asset classes, without repeating a lot of redundant data.
* Adding new asset classes will be a lot easier and with less impact on existing system. It involves creating a new table with similar constraints on it. The Super type table need not be touched.

Disadvantages:

* An integer identifier just adds another level of indirection. This drawback has to be viewed against the cost involved in handling ticker changes, how frequent a ticker change due to external factors or operation/vendor data error.
* Super type/sub-type model leads to more joins to get useful data. Views must be designed to view an asset class in entirety.

Another approach is to use a flat wide sparse table with columns that apply for all asset class.

Advantages:

* Just one table to maintain. Querying will be easier.

Disadvantages:

* A wide sparse tables causes lot of redundancy and null columns. A natural null value in a column has to be differentiated from a column that’s null because it doesn’t apply to this asset class.
* Adding a new asset class means modifying the master table that impacts all asset classes. It can also introduce complications with default values.

Question 2:

**Suppose that the Ticker of a security changes over time. For example, the ticker of Google from 8/19/2004 to 1/20/2015 is GOOG, and after that it changes to GOOG2. How would you store this information in the database? Discuss and contrast two different ways of saving this information?**

Such a change has to be tracked with a time series data set. One approach is to add the time series feature to the SecurityMaster table. In such a case, the primary key on the table will be the MasterSecID, EffectiveStartDate and EffectiveEndDate. There are advantages and disadvantages in this approach. Adding time series to this table will provide a single source of truth with historic data.

But since this will be one of the tables that can be queried very often in number of places within the firm, this table can just be left transactional with no history tracking. A Change in ticker means just an update to the ticker column (only with the approach above using integer ID as primary key). This update on the SecurityMaster can be done after end of the stock’s trading day.

An appropriate place to track the historic changes will be a **data warehouse**. SecurityMaster table will always capture the live value of the data while the dimension table below tracks history.

Below is the DimSecurityMaster dimension table that has two pair of timestamp columns.

* **EffectiveStartDate/EffectiveEndDate** – the business effective date of the record. In our case, 8/19/2004 to 1/20/2015 will be the effective date for the record with ticker as GOOG.

1/21/2015 to 12/31/9999 will be the effective date for the record with ticker as GOOG2. The effective end date is just a hypothetical date indicating end of life (or until another ticker change comes in).

* **KnowledgeStartDate/KnowledgeEndDate –** the system time when the Security Master application is made aware of this change.

--DimSecurityMaster

CREATE TABLE DimSecurityMaster(

ID int NOT NULL, -- surrogate key

EffectiveStartDate Datetime Not NULL,

EffectiveEndDate datetime NOT NULL,

KnowledgeStartDate datetime NOT NULL,

KnowledgeEndDate datetime NOT NULL,

MasterSecID int, --natural key

Ticker varchar(10),

Name varchar(600),

Currency varchar(5),

AssetType varchar(10),

ExpirationDate datetime,

ContractSize int,

StrikePrice numeric(28,12),

OptionType varchar(4),

Underlier varchar(10),

LastModifiedDate datetime,

LastModifiedBy varchar(60),

CreatedDate datetime,

CreatedBy varchar(60),

Primary key (ID, EffectiveStartDate, EffectiveEndDate, KnowledgeStartDate, KnowledgeEndDate)

);

ID column is an integer ID called Surrogate Key. Each natural key will have an associated surrogate key. Though we are already using an integer ID (MasterSecID) as a key in the original table, a data warehouse is always recommended to maintain its own integer key to separate itself from any issues in a transactional system. If a situation arises where the MasterSecID itself has to be updated, the data warehouse will be able to handle it. This policy also aligns with the general principle of data warehouse with any source system in the firm.

So, in this particular example with Google,

On 1/19/2015, this is how the entry would look like.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | MasterSecID | Ticker | EffStartDate | EffEndDate | KnoldgeStartDate | KnoldgeEndDate |
| 1 | 2347 | GOOG | 1/1/1900 | 12/31/9999 | 1/1/1900 | 12/31/9999 |

Assuming that we are notified that GOOG is undergoing this ticker change effective 1/21/2015 on 1/20/2015.

The current knowledge is not valid anymore. So, we expire the knowledge date on the existing effective date range and add a new row (with old ticker) that is only effective until 1/20/2015.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | MasterSecID | Ticker | EffStartDate | EffEndDate | KnoldgeStartDate | KnoldgeEndDate |
| 1 | 2347 | GOOG | 1/1/1900 | 12/31/9999 | 1/1/1900 | 1/20/2015 |
| 1 | 2347 | GOOG | 1/1/1900 | 1/20/2015 | 1/1/1900 | 12/31/9999 |

Now, to record the new knowledge, we add another row with the new ticker and effective date from 1/21/2015 to end of time.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | MasterSecID | Ticker | EffStartDate | EffEndDate | KnoldgeStartDate | KnoldgeEndDate |
| 1 | 2347 | GOOG2 | 1/21/2015 | 12/31/9999 | 1/20/2015 | 12/31/9999 |

**Deliverable 3 – Class Diagram**

Attached SecuritiesMaster.zip file contains the source code to calculate market value for Equity, Options and Future asset classes.

Please look at the Class Design page in the attached Visio document to view the class hierarchy.

An Asset Class is represented by the base type **SecurityBase**. SecurityBase will contain the common properties and methods for any asset class.

SecurityBase is further derived and extended by **Equity**, **Future** and **Options** types. Each of these add more properties and methods to the SecurityBase.

**TickerPrice** and **Position** types are used to represent the pricing and position information for any given day.

An Interface called **IMarketValueCalculator** is defined with a method contract called CalculateMarketValue.

Each asset classes have separate formula to calculate Market Value. So, each of the following types under SecurityMaster.Calculator namespace define the strategy to calculate Market Value – **EquityMarketValueCalculator**, **FutureMarketValueCalculator**, **OptionsMarketValueCalculator**. This strategy can be extended to multiple formula and calculations for any asset type.

**CalculatorFactory** is a static class that can be used as a factory that provides different calculator based on the asset type. In this case, the CalculatorFactory contains a method called GetMarketValueCalculatorByAssetClass. This factory can be extended to work to provide any new calculator implemented or return a calculator that did not even exist when the above method was created.

CalculatorFactory is the provider. But It doesn’t actually create the instances that it provides by itself. This is injected through the IOC Container – **Structuremap**. The structuremap container is responsible for the scope and lifetime of the object. In our case, the calculators are singleton instances. Each of these calculator instances don’t have any state other than the actual method. So, we don’t have to worry about thread safety with this singleton class. We also don’t have to create multiple instances for each row in list of securities.

The named calculator instances are configured in the StructureMap.xml configuration file and loaded in the static constructor of the CalculatorFactory static class.

**SecurtiesController** is the central class that uses the above setup. I have used the assumption provided in the question and stubbed it with sample data. This type contains the method in question CalculateMarketValue(date). This uses the assumed methods to retrieve the positions, prices and all the securities. It goes over each security, finds the asset type, uses the factory to get the strategy to calculate Market Value. The factory uses DI through the Structuremap IOC container to get the algorithms.

Using the Factory model and DI greatly helps in reducing the cyclomatic complexity. Most of the implementation is written to the interface and the decision are delayed until the run time to select the correct behavior.

**Deliverable 2 – Architecture Design:**

**Suppose that the application will initially only have a web front end, but in the future we may need to develop native mobile applications. How would you design the architecture of the application? Create a high level diagram showing the physical as well as logical components in the application.**

Please look at the Architecture Design page in the attached Visio document for a high level diagram.

The application is built on top of SQL Server database. Entity Framework ORM is used for abstraction. IRepository interface is used to abstract from any particular database or ORM implementation.

A SecuritiesController API is a Restful Web API that will act as the backend for the web application. This Rest API is the key to extending the capability to cater future need for a mobile application. The service provides API that return JSON serialized data. This, along with a smart implantation of paging and caching technique can enhance the performance of mobile applications.

The Web client itself can be wrapped as a hybrid mobile app using frameworks like PhoneGap or as a native desktop app using HTML containers like OpenFin.