A system, method, and computer program product are provided for implementing materialized views on a mobile device in synchronization with an enterprise information system. Changes to an underlying local database, on which the materialized view is based, may originate from the enterprise system or from the client. A solution needs to account for each of these potential sources of changes, and update the materialized view accordingly.
FIG. 1
FIG. 2

START

Register for downloaded change callback

Receive callback for downloaded create/update/delete

Maintain materialized view

END
FIG. 3

302 START

304 Define MBO for materialized view

306 Define triggers for create/update/delete calls

308 Run code generator

310 END
ENTITY TRIGGERS FOR MATERIALIZED VIEW MAINTENANCE

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is related to co-pending U.S. Provisional Patent Application No. 61,425,554, filed on Dec. 21, 2010 and entitled “Bulk Initial Download of Mobile Databases”, which is incorporated herein by reference in its entirety.

BACKGROUND OF INVENTION

1. Field of the Invention

The present invention relates generally to databases and, in particular, to mobile database efficiency.

2. Description of the Background Art

Relative to a server-class computer system, a mobile device has a relatively slow CPU and relatively slow input/output performance (e.g., for reading/writing database files on media cards). A mobile device also has a relatively small amount of memory (e.g., RAM), which limits the effectiveness of any kind of in-memory database cache, such as a row cache or page cache.

As a result, the time taken for a mobile device to query and display a large amount of records (e.g., working with a contact list of 10,000+ records) can be excessive. Desired functionality such as incremental search may be too computationally expensive to perform at all on the mobile device.

Accordingly, what is desired is a mechanism to improve the ability of a resource-limited mobile device to work with local database records.

SUMMARY OF INVENTION

Embodiments of the invention include a method comprising defining a mobile business object for a materialized view, defining triggers, within the mobile business object, for changes to a local database affecting the materialized view, and executing a code generator on the mobile business object definition.

Embodiments of the invention additionally include a computer-readable storage device having computer program logic recorded thereon, execution of which, by a computing device, causes the computing device to perform operations comprising defining a mobile business object for a materialized view, defining triggers, within the mobile business object, for changes to a local database affecting the materialized view, and executing a code generator on the mobile business object definition.

Embodiments of the invention further include a system comprising a memory configured to store modules comprising a first defining module configured to define a mobile business object for a materialized view, a second defining module configured to define triggers, within the mobile business object, for changes to a local database affecting the materialized view, and an executing module configured to execute a code generator on the mobile business object definition, and one or more processors configured to process the modules.

Further features and advantages of the invention, as well as the structure and operation of various embodiments of the invention, are described in detail below with reference to the accompanying drawings. It is noted that the invention is not limited to the specific embodiments described herein. Such embodiments are presented herein for illustrative purposes only. Additional embodiments will be apparent to persons skilled in the relevant art(s) based on the teachings contained herein.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated herein and form a part of the specification, illustrate embodiments of the present invention and, together with the description, further serve to explain the principles of the invention and to enable a person skilled in the relevant art to make and use the invention.

FIG. 1 is an exemplary enterprise network, in accordance with an embodiment of the present invention.

FIG. 2 is a flowchart illustrating steps by which such callback handlers can be utilized to implement a materialized view, in accordance with an embodiment of the present invention.

FIG. 3 is a flowchart illustrating steps by which triggers are defined using an MBO, in accordance with an embodiment of the present invention.

FIG. 4 depicts an example computer system in which embodiments of the present invention may be implemented.

The present invention will now be described with reference to the accompanying drawings. In the drawings, generally, like reference numbers indicate identical or functionally similar elements. Additionally, generally, the leftmost digit(s) of a reference number identifies the drawing in which the reference number first appears.

DETAILED DESCRIPTION

I. Introduction

The following detailed description of the present invention refers to the accompanying drawings that illustrate exemplary embodiments consistent with this invention. Other embodiments are possible, and modifications can be made to the embodiments within the spirit and scope of the invention. Therefore, the detailed description is not meant to limit the invention. Rather, the scope of the invention is defined by the appended claims.

It would be apparent to one of skill in the art that the present invention, as described below, can be implemented in many different embodiments of software, hardware, firmware, and/or the entities illustrated in the figures. Any actual software code with the specialized control of hardware to implement the present invention is not limiting of the present invention. Thus, the operational behavior of the present invention will be described with the understanding that modifications and variations of the embodiments are possible, and within the scope and spirit of the present invention.

Reference to modules in this specification and the claims means any combination of hardware or software components for performing the indicated function. A module need not be a rigidly defined entity, such that several modules may overlap hardware and software components in functionality. For example, a software module may refer to a single line of code within a procedure, the procedure itself being a separate software module. One skilled in the relevant arts will understand that the functionality of modules may be defined in accordance with a number of stylistic or performance-optimizing techniques, for example.
FIG. 1 is an exemplary enterprise network 100, in accordance with an embodiment of the present invention. The enterprise network 100 includes a mobile device 102, in accordance with a further embodiment of the present invention. Mobile device 102 may include, by way of example and not limitation, mobile devices including Java-enabled phones with CLDC 1.1 Java mobile runtime environment such as the BLACKBERRY by RESEARCH IN MOTION of Waterloo, Ontario, Canada; the APPLE PHONE and IPAD by APPLE COMPUTER, Inc. of Cupertino, Calif.; Java-enabled phones with the GOOGLE ANDROID operating system by GOOGLE INC. of Mountain View, Calif.; and the WINDOWS PHONE by MICROSOFT CORPORATION of Redmond, Wash. One skilled in the relevant arts will recognize that techniques described herein as applicable to mobile device 102 may also generally be applied to non-mobile devices as well, such as, for example, a personal computer.

In accordance with an embodiment of the present invention, mobile device 102 has a client application 104 installed thereon. Client application 104 is able to interface with device inputs and outputs ("I/O") 106, such as, for example, a monitor, keypad, or touchscreen display, in accordance with an embodiment of the present invention. Client application 104 is also able to interface with a local database 108, which stores a set of data intended for use by client application 104. In accordance with an embodiment of the present invention, local database 108 is encrypted by a device-specific encryption key.

Mobile device 102 is in communication with a synchronization server 110, in accordance with an embodiment of the present invention. Additional mobile devices 112a-c are similarly in communication with synchronization server 110, in accordance with a further embodiment of the present invention. The various mobile devices can be connected to synchronization server 110 via any one or more communications channels, as would be understood by one skilled in the relevant art. For example, connectivity between mobile device 102 and synchronization server 110 may involve, in an exemplary embodiment, communications hops over both a cellular communication network and the Internet. The various communication hops may themselves be either public or private networks, and may include components located on the Internet as well as various private intranets.

Synchronization server 110 sits between the one or more mobile devices 102 and 112a-c and an Enterprise Information System ("EIS") 114, in accordance with an embodiment of the present invention. Synchronization server 110 assists in capturing changes to relevant data made by the EIS 114 and providing the changes to mobile devices 102 and 112a-c. Synchronization server 110 also assists in capturing changes made by the mobile devices 102 and 112a-c and providing the changes to EIS 114. In this manner, data available to a mobile device 102 in local database 108 can be synchronized with data from the corresponding data store of EIS 114, the enterprise data system 116 (or "central database"). In accordance with an embodiment of the present invention, synchronization server 110 maintains a cache reflecting the data from enterprise data system 116.

EIS 114 is connected to synchronization server 110 in order to allow synchronization server 110 to provide the aforementioned data synchronization services, in accordance with an embodiment of the present invention. Communications between EIS 114 and synchronization server 110 can likewise be through any communication channels, as would be understood by one skilled in the relevant art. One skilled in the relevant arts will further understand that EIS 114 and synchronization server 110 may share a common physical server or distributed server as separate software components therein, or may even be compiled as a single combined application. Therefore, it is understood that synchronization server 110 and EIS 114 may be disposed in a number of different locations within enterprise network 100, and are shown as separate computing devices in FIG. 1 by way of example, and not limitation.

As previously noted, EIS 114 further includes, or is otherwise communicatively coupled to, an enterprise data system 116, in accordance with an embodiment of the present invention. In accordance with a further embodiment of the present invention, data within local database 108 comprises a subset of data from enterprise data system 116. This data may be in the form of a mobile business object ("MBO"), in accordance with an embodiment of the present invention. An MBO is a business object that can be synchronized between enterprise information system 114 and mobile device 102. The MBO can be persisted, by storage in database 108, in order to allow for access by the mobile device 102 during periods without connectivity to EIS 114. A non-limiting example of MBOs is provided in U.S. patent application Ser. No. 12/505,573, filed on Jul. 15, 2009, Atty. Dkt. No. 1933, 0720001, entitled "Metadata Driver Mobile Business Objects," which is incorporated by reference herein in its entirety.

Client applications 104 running on mobile device 102 access (read and write) to the local database 108. The local database 108 is accessed by client application 104 via generated classes, where each class (corresponding to the aforementioned MBOs) represents one table, in accordance with an embodiment of the present invention.

Communications between client application 104 and synchronization server 110 may be handled in a number of different ways, as will be recognized by one skilled in the relevant art. In accordance with an embodiment of the present invention, a communications framework is embedded directly within client application 104 as provided by a code generator or by a developer. In accordance with an additional embodiment of the present invention, a helper application may be deployed on mobile device 102 to manage the communications.

II. Exemplary Business Object Models

In accordance with an embodiment of the present invention, MBOs can be modeled using a publicly available XML based domain-specific language called Applications From XML ("AFX"). One skilled in the relevant arts will recognize that MBOs can be modeled in many different ways, including graphically through the use of visual modeling tools, or textually through the use of domain-specific languages such as AFX. Accordingly, examples presented and discussed herein through the use of AFX are provided by way of example, and not limitation.

A package definition provides a database definition, defining the name of a local database 108 within mobile device 102, a database-class definition, and zero or more entity definitions, in accordance with an embodiment of the present invention. The database-class definition identifies a class that represents the local database 108. Entity definitions,
if present, each correspond to an MBO. The entity definition in turn defines the name of a class that represents the corresponding MBO.

[0031] Entities are themselves stored within local database 108, in accordance with an embodiment of the present invention. In accordance with a further embodiment of the present invention, entities are keyed using at least a surrogate key and an alternate key. The surrogate key is a key value used by the mobile device 102 to account for the mobile device's inability to access the EIS' 114 key creation process. If the client becomes aware of the EIS 114 key for the entity, this key can be used as an alternate key. In accordance with a further embodiment of the present invention, surrogate keys are provided by synchronization server 110 to mobile device 102 as a batch of keys to be assigned as needed.

[0032] In accordance with a further embodiment of the present invention, entity definitions include operation definitions for corresponding changes within EIS 114. By way of example, and not limitation, operation definitions are provided for creating, updating, or deleting data within EIS 114. Entity definitions may also provide named queries, which are predefined queries that can be issued by mobile device 102 against local database 108, in accordance with an embodiment of the present invention. In accordance with further embodiments of the present invention, entity definitions may also include definitions indicating that changes made by mobile device 102 can be published to synchronization server 110, and likewise that changes made by EIS 114 are subscribed to for download to mobile device 102.

[0033] A non-limiting example object model incorporating the aforementioned elements may therefore read:

```xml
<package name="com.example.bank">
  <database name="my-bank"/>
  <database-class name="MyDatabase"/>
  <entity name="Account" keys="surrogateKey" alternate-key="accountId">,
    <attribute name="surrogateKey" type="int" generated="true" />
    <attribute name="accountId" type="string" />
    <attribute name="customerId" type="string" />
    <attribute name="balance" type="decimal" />
    <operation name="create">,
      <!-- information describing the EIS operation -->
    </operation>
    <operation name="update">
      <!-- information describing the EIS operation -->
    </operation>
    <operation name="delete">
      <!-- information describing the EIS operation -->
    </operation>
    <query name="findByCustomer" type="Account">,
      <parameter name="id" type="string" />
      <sql>
        select a.* from Account a where a.customerId = :id
      </sql>
    </query>
  </entity>
</package>
```

[0034] In the above exemplary object model, a package "com.example.bank" associated with local database 108 "my-bank" represented by class "MyDatabase" contains a single entity named "Account". This entity is keyed by a surrogate key, with an alternate key defined by the account ID, which in this instance is the actual unique key used by the backend systems of EIS 114.

[0035] The Account entity also has several attributes, including the aforementioned surrogate key and account ID, but also a customer ID corresponding to the account and the balance of the account, in accordance with an embodiment of the present invention. The Account entity also defines a query "findByCustomer" that allows for a particular Account with the corresponding customer ID to be located and used in an instance of the MBO.

[0036] Additionally, the Account entity defines several operations describing EIS 114 operations for creating, updating, and deleting data corresponding to an entity from within enterprise data system 116, in accordance with an embodiment of the present invention.

[0037] In accordance with an embodiment of the present invention, a code generator is run on the package definition in order to generate code for inclusion in client application 104. This allows a developer of client application 104 to rapidly generate the client application 104 to coordinate the downloading and uploading of changes between mobile device 102 and synchronization server 110. In accordance with a further embodiment of the present invention the code generator can also generate code for execution on the synchronization server 110.

[0038] Based on the above exemplary package definition, it is possible to code client application 104 to utilize generated code derived by the code generator from the package definition to perform certain functions. For example, code generated from the above package generation can be called to resolve the following code sample from client application 104:

```javascript
var accounts = Account.findByCustomer("123");
for (var a in accounts) {
    print("Account: " + a.customerId + " balance: " + a.balance);
}
```

[0039] This code utilizes calls to classes automatically generated by the code generator, in accordance with an embodiment of the present invention. For example, the segment `Account.findByCustomer("123")` utilizes the named query "findByCustomer" in order to return matching accounts into the "accounts" variable. Then, for each result "a" in "accounts", the corresponding account ID and balance can be displayed.

[0040] Rather than using the named query findByCustomer, an end-user may specify selection criteria in a dynamic query, in accordance with an embodiment of the present invention. A code generator may specify several options for a developer to choose from in defining the necessary query. For example, the following example code can be customized by a developer to search for accounts matching specific attributes:

```javascript
var attribute = ...
var testValue = ...
var query = new Query();
query.where(AttributeTest.equal(attribute, testValue));
var accounts = Account.findAllWithQuery(query);
```
In accordance with an embodiment of the present invention, a drop-down menu or other input interface functionality may be provided to the developer in order to customize the attribute and text value used in the above example.

III. Updating the Local Database

The data on central database 116 is synchronized with the data in local database 108 through the use of a database synchronization system, such as synchronization server 110, in accordance with a non-limiting example embodiment of the present invention. For example, when client application 104 first creates a local database 108, it sends a subscription request to the server system 114. The server 114 responds by sending, for each database table, a set of rows that need to be inserted into the client’s local database. This set of rows may or may not be the same set of rows that would be sent to a different mobile device 112a-c for a different user. The client application inserts these rows one at a time into local database 108, and maintains table indexes, until all the rows have been inserted.

The potentially numerous insertions that must be made by client application 104 into local database 108 can overwhelm the resources of mobile device 102. A number of solutions have been considered to alleviate this problem. One solution is prepared statement caching. Rather than the client application 104 asking the mobile database software managing local database 108 to parse a SQL ‘insert’ statement every time a row is to be inserted, the parsed statement can be reused. This may potentially reduce the initial download time by 5-10%, for example.

Another approach is to use long transactions. Since most database systems incur a not insignificant cost for every committed transaction, multiple insert statements can be executed in a single transaction to reduce transaction commit costs as a proportion of the overall download time. Either the complete download can operate as a single transaction, or as a set of transactions each of which is for a batch of inserted rows. As an example, the use of long transactions might improve download performance by a factor of between 3 and 10, although during that shortened download period, the device can still become quite unresponsive.

An additional approach is to send preformatted rows. Typically, a mobile device database 108 stores each table row as a contiguous sequence of bytes, and multiple rows (as many as will fit) are grouped together into fixed length pages (e.g., 1024 bytes). If a SQL ‘insert’ statement is executed with a set of prepared statement parameters, the corresponding sequence of bytes must be constructed from the set of parameter values. If the server knows the expected row representation of the mobile database, the server can send each row as a preformatted sequence of bytes, reducing the CPU utilization of the mobile device. As an example, the use of preformatted rows might improve download performance by a factor of 5, although during the shortened download period, the device can still become quite unresponsive, because packing of rows into pages and index maintenance still consumes a lot of CPU time and results in many input/output operations to the storage media on which local database 108 is held.

Yet another approach is to send pregenerated databases to mobile device 102. The server 114 can be configured to occasionally generate a client database as a file, which is downloaded by each mobile device 102 and 112a-c when it first subscribes. Simple file download to a mobile device 102 can be quite fast, e.g., a few seconds per megabyte, depending on the speed of the network (or cable) connection between the device and the server. If a database contains a lot of reference data, such as a product catalog, this technique can be particularly effective. However, when the set of rows is not the same for each device, then a pregenerated database will not be appropriate. Also if the client database must be encrypted with a device-specific encryption key, then the server will not be able to send to the device an appropriately encrypted database, unless the client first sends its encryption key to the server, which may be considered an unacceptable security risk.

U.S. Provisional Patent Application No. 61,425,554, filed on Dec. 21, 2010 and entitled “Bulk Initial Download of Mobile Databases”, which is incorporated herein by reference in its entirety, describes a further embodiment for impersonating the client to generate a client database file. This file can then be transmitted to the client in a manner that avoids the need for multiple or complicated and resource-intensive queries (e.g., insert statements) executed on the client, and that has been generated to conform to the client’s specific requirements.

IV. Materialized Views—Background

On a mobile device, such as mobile device 102, resources for querying and otherwise working with local database 108 are limited by comparison to the resources of EIS 114. However, mobile devices are an integral part of many people’s work lives, and delivering enterprise-level functionality to them is necessary. This means, for example, finding ways for mobile devices to display a scrollable list of contacts for a large organization having over 10,000 contacts. For even better end-user access of the contact list, supporting incremental search on a contact list of that size compounds the problem even further. Additionally, options can be provided for sorting the list “first, last” or “last, first”, removing several optimization options.

In the aforementioned non-limiting example, it turns out that it is not necessary to load all 10,000 contacts initially into the graphical user interface (“GUI”) component (e.g., a list view) responsible for managing the contact list display. Instead, in accordance with an embodiment of the present invention, materialized views are used to manage the data set.

A materialized view is, in accordance with a non-limiting embodiment of the present invention, a database object that contains the results of a query. Additional queries can be run on the materialized view, providing quick access to relevant information from a portion of the database.

One of the difficulties of proper implementation of materialized views is maintenance of their data. When data from an underlying source table, from which the materialized view is formed, is updated, the data in the materialized view does not immediately reflect the update. In some cases this is not critical, as the old data can continue to be used for certain
operations. In other cases, the materialized view may be updated whenever it is next needed in order to defer costs of the update.

Prior efforts in improving the performance and usability of materialized views are directed to single system approaches. However, a different approach is needed to implement and maintain materialized views in the Sybase Unwired Platform ("SUP"), for example. Systems that rely on synchronization techniques such as Message-Based Synchronization ("MBS") or Replication-Based Synchronization ("RBS") do not implement materialized views without accounting for the various sources of updates to the data in a materialized view. U.S. patent application Ser. No. 12/813, 104, filed on Jun. 16, 2010 and entitled “Message Based Synchronization for Mobile Business Objects”, which is incorporated herein by reference in its entirety, provides further details regarding synchronization techniques.

V. Implementing Materialized Views with Callback Handlers

One approach to implementing materialized views on a mobile device, such as mobile device 102, is through the use of callback handlers. FIG. 2 is a flowchart 200 illustrating steps by which such callback handlers can be utilized to implement a materialized view, in accordance with an embodiment of the present invention. In accordance with a non-limiting embodiment of the present invention, the database synchronization system (such as, e.g., SUP) provides an API defining a callback handler interface. A client application 104 can then register an instance of the callback handler with the database, or with an individual MBO.

The method starts at step 202 and proceeds to step 204 where a client application 104 registers for a download change callback. In this case, client application 104 registers for the aforementioned callback handler and provides an implementation of an onImport method, although one of ordinary skill in the relevant art will appreciate that other callback mechanisms can be utilized and are contemplated within the scope of this disclosure. The onImport method, as implemented by client application 104, is called to handle a downloaded change at step 206. As a result, client application 104 is notified that a change to local database 108 (e.g., create, update, delete) has been received.

At step 208, client application 104 is able to take any necessary action to reflect the changes to local database 108 on the materialized view. One of ordinary skill in the relevant art will appreciate that the materialized view maintenance can be implemented in a number of ways within the scope of this invention, including re-running the query used to generate the materialized view subsequent to update of local database 108, or performing incremental updates of the materialized view based on the effect of the downloaded changes. The method then ends at step 210.

While this approach is sufficient to implement and maintain a materialized view given downloaded changes, it does not maintain materialized views given changes to the local database 108 for other reasons. For example, client application 104 may itself make changes directly to local database 108, or another client application may do the same. Accounting for this becomes difficult, as each location in the code of client application 104 that makes changes to local database 108 must be extended with code to maintain materialized views.

Rather than rewriting significant portions of the code of client application 104, the callback handler interface can be extended to capture create/update/delete calls issued by client application 104, in addition to those issued by downloaded changes. Such callbacks would operate in similar manner to FIG. 2, but the callback from step 206 would be received for create/update/delete calls issued by client application 104 as well.

Since callback handlers are registered with a database or MBO classes by the client application at runtime, they are not known to the MBO code generator. In this case, it is not possible to guarantee that the materialized views will be maintained. Additionally, materialized views outside of the client application can become problematic, such as during server-side database preparation for the bulk initial download. In that case, the callback handlers are not defined within the deployment unit, and cannot be called.

VI. Implementing Materialized Views with Entity Triggers

In accordance with a further embodiment of the present invention, materialized views are implemented through the use of entity triggers. Entity triggers are the entity (MBO) analogues to SQL database triggers. An entity trigger is defined within the deployment unit (AFX model), as a static operation in a materialized view entity (like a special client-only MBO). Thus, code generation can ensure that triggers are always called at the appropriate times.

By way of non-limiting example, suppose an Account MBO has been defined, and an AccountView materialized view needs to be defined. In accordance with this example, AccountView is defined as a client-only MBO, but it can have extra trigger operations defined.

```xml
<entity name="AccountView" key="id">  
  <attribute name="id" type="long" />  
  <attribute name="name" type="string" />  
  <operation name="onBulkDownload" static="true" bulk-creates="true">  
    <code>  
      var accounts = Account.findAll();  
      for (var account in accounts)  
      {  
        onCreateAccount(account);  
      }  
    </code>  
  </operation>  
</entity>
```

```xml
<entity name="AccountView" key="id">  
  <attribute name="id" type="long" />  
  <attribute name="name" type="string" />  
  <operation name="onCreateAccount" static="true" after-create="true">  
    <parameter name="account" type="Account" />  
    <code>  
      var av = new AccountView  
      {  
        id : account.surrogateKey,  
        name : account.firstName + " " + account.lastName,  
      };  
      av.create();  
    </code>  
  </operation>  
</entity>
```

```xml
<entity name="AccountView" key="id">  
  <attribute name="id" type="long" />  
  <attribute name="name" type="string" />  
  <operation name="onUpdateAccount" static="true" after-update="true">  
    <parameter name="account" type="Account" />  
    <code>  
      var av = AccountView.load(account.surrogateKey);  
      av.id = account.id;  
      av.name = account.firstName + " " + account.lastName,  
      av.update();  
    </code>  
  </operation>  
</entity>
```

```xml
<entity name="AccountView" key="id">  
  <attribute name="id" type="long" />  
  <attribute name="name" type="string" />  
  <operation name="onDeleteAccount" static="true" after-delete="true">  
    <parameter name="account" type="Account" />  
    <code>  
      var av = AccountView.load(account.surrogateKey);  
    </code>  
  </operation>  
</entity>
```
FIG. 3 is a flowchart 300 illustrating steps by which triggers are defined using an MBO, in accordance with an embodiment of the present invention. The method begins at step 302 and proceeds to step 304 where an MBO is defined for the materialized view. In the above example, this is the MBO AccountView for the AccountView materialized view. In accordance with an embodiment of the present invention, the MBO is defined as a client-only MBO so that it is not persisted to EIS 114.

At step 306, an entity trigger is defined for each of create/update/delete, in accordance with an embodiment of the present invention. Using these triggers, it is possible to code a behavior responsive to each type of call that will update the materialized view accordingly. In accordance with an embodiment of the present invention, the relevant behavior is coded using any one of XScript, CH, Java, or Objective-C, by way of non-limiting example. A code generator (e.g., AFX code generator) is then run on the MBO definition, and the method ends at step 310.

By providing materialized view maintenance in entity triggers, the functionality can be separated from that of callback handlers, such as user interface refreshing. Additionally, if the trigger behavior is coded using XScript, the code generator can produce equivalent code for the triggers that will execute at both the mobile device 102 and EIS 114. This is useful, for example, when performing bulk initial download using the aforementioned technique of impersonating the client at server. Additionally, XScript code is deployable on multiple platforms, but end-user developers are able to utilize any language they wish to work with in the code generator.

For MBS, the AFX code generator is readily capable of ensuring that entity triggers are called at the appropriate time. For RBS, there is some added complexity. In particular, it is not possible to directly capture when downloaded create/update/delete statements are executed because they are executed by the local database 108 engine, not the MBO layer.

In accordance with an embodiment of the present invention, SUP provides a “ChangeLog” feature, although one of ordinary skill in the art will appreciate that features in other technologies can be similarly utilized. In this case, the code generator is configured such that if a deployment unit (AFX model) contains entity triggers, then the code generator generates a database class (e.g., “runEntityTriggers”) operation which would retrieve change logs (e.g., “getChangeLogs”) to determine the changed entities. From there, the appropriate entity triggers are directly invoked as appropriate.

In the case of SUP, the “ChangeLog” API cannot distinguish downloaded creates from updates (rather, all that can be seen are “upserts” and deletes). In such a situation, an extra kind of trigger after-save is introduced in accordance with an additional embodiment of the present invention. Furthermore, the technique of relying on change logs may not be atomic, and may be inadvertently repeated in a failure scenario. Therefore, in accordance with a further embodiment of the present invention, a delete trigger is programmed not to fail if the source MBO instance no longer exists. Exemplary, non-limiting code implementing these features is provided by way of illustration:

In accordance with a further embodiment of the present invention, it is also possible to implement derived materialized views. Derived materialized views are views whose triggers are for create, update and delete operations on other materialized views. These are particularly useful with RBS. In the case of RBS, an after-save trigger for Account would not correctly maintain a letter-count summary view (a view that has one row for each letter of the alphabet, with each row having a letter column and a count column, the count column tracking the number of accounts with a name starting with the corresponding letter of the alphabet), as it would not be able to distinguish create calls from delete calls. Instead, in accordance with an embodiment of the present invention, a trigger is defined against AccountView. Since all changes to AccountView are initiated by the MBO layer (not syncable), it is possible to correctly distinguish create calls from update calls.

VII. Example Computer System Implementation

Various aspects of the present invention can be implemented by software, firmware, hardware, or a combination thereof. FIG. 4 illustrates an example computer system 400 in which the present invention, or portions thereof, can be implemented as computer-readable code. For example, the methods illustrated by flowcharts 200 of FIG. 2 and 300 of FIG. 3, can be implemented in system 400. Various embodiments of the invention are described in terms of this example computer system 400. After reading this description, it will become apparent to a person skilled in the relevant art how to implement the invention using other computer systems and/or computer architectures.

Computer system 400 includes one or more processors, such as processor 404. Processor 404 can be a special
purpose or a general purpose processor. Processor 404 is connected to a communication infrastructure 406 (for example, a bus or network).

Computer system 400 also includes a main memory 408, preferably random access memory (RAM), and may also include a secondary memory 410. Secondary memory 410 may include, for example, a hard disk drive 412, a removable storage drive 414, and/or a memory stick. Removable storage drive 414 may comprise a floppy disk drive, a magnetic tape drive, an optical disk drive, a flash memory, or the like. The removable storage drive 414 reads from and/or writes to a removable storage unit 418 in a well known manner. Removable storage unit 418 may comprise a floppy disk, magnetic tape, optical disk, etc. that is read by and written to by removable storage drive 414. As will be appreciated by persons skilled in the relevant art(s), removable storage unit 418 includes a computer usable storage medium having stored therein computer software and/or data.

In alternative implementations, secondary memory 410 may include other similar means for allowing computer programs or other instructions to be loaded into computer system 400. Such means may include, for example, a removable storage unit 422 and an interface 420. Examples of such means may include a program cartridge and cartridge interface (such as that found in video game devices), a removable memory chip (such as an EPROM, or ROM) and associated socket, and other removable storage units 422 and interfaces 420 that allow software and data to be transferred from the removable storage unit 422 to computer system 400.

Computer system 400 may also include a communications interface 424. Communications interface 424 allows software and data to be transferred between computer system 400 and external devices. Communications interface 424 may include a modem, a network interface (such as an Ethernet card), a communications port, a PCMCIA slot and card, or the like. Software and data transferred via communications interface 424 are in the form of signals that may be electronic, electromagnetic, optical, or other signals capable of being received by communications interface 424. These signals are provided to communications interface 424 via a communications path 426. Communications path 426 carries signals and may be implemented using wire or cable, fiber optics, a phone line, a cellular phone line, an RF link or other communications channels.

In this document, the terms “computer program medium” and “computer usable medium” are used to generally refer to media such as removable storage unit 418, removable storage unit 422, and a hard disk installed in hard disk drive 412. Signals carried over communications path 426 can also embody the logic described herein. Computer program medium and computer usable medium can also refer to memories, such as main memory 408 and secondary memory 410, which can be memory semiconductors (e.g., DRAMs, etc.). These computer program products are means for providing software to computer system 400.

Computer programs (also called computer control logic) are stored in main memory 408 and/or secondary memory 410. Computer programs may also be received via communications interface 424. Such computer programs, when executed, enable computer system 400 to implement the present invention as discussed herein. In particular, the computer programs, when executed, enable processor 404 to implement the processes of the present invention, such as the steps in the methods illustrated by flowcharts 200 of FIG. 2 and 300 of FIG. 3, discussed above. Accordingly, such computer programs represent controllers of the computer system 400. Where the invention is implemented using software, the software may be stored in a computer program product and loaded into computer system 400 using removable storage drive 414, interface 420, hard drive 412 or communications interface 424.

The invention is also directed to computer program products comprising software stored on any computer useable medium. Such software, when executed in one or more data processing devices, causes a data processing device(s) to operate as described herein. Embodiments of the invention employ any computer useable or readable medium, known now or in the future. Examples of computer useable mediums include, but are not limited to, primary storage devices (e.g., any type of random access memory), secondary storage devices (e.g., hard drives, floppy disks, CD ROMS, ZIP disks, tapes, magnetic storage devices, optical storage devices, MEMS, nanotechnological storage device, etc.), and communication mediums (e.g., wired and wireless communications networks, local area networks, wide area networks, intranets, etc.).

VIII. Conclusion

While various embodiments of the present invention have been described above, it should be understood that they have been presented by way of example only, and not limitation. It will be understood by those skilled in the relevant art(s) that various changes in form and details may be made therein without departing from the spirit and scope of the invention as defined in the appended claims. It should be understood that the invention is not limited to these examples. The invention is applicable to any elements operating as described herein. Accordingly, the breadth and scope of the present invention should not be limited by any of the above described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.

What is claimed is:
1. A method comprising:
   defining a mobile business object for a materialized view;
   defining triggers, within the mobile business object, for changes to a local database affecting the materialized view;
   and
   executing a code generator on the mobile business object definition.
2. The method of claim 1, wherein the changes to the local database comprise at least one of a create, update, or delete call.
3. The method of claim 1, wherein the changes to the local database comprise a bulk initial download.
4. The method of claim 1, further comprising:
   writing platform-specific code for the triggers to maintain the materialized view.
5. The method of claim 1, wherein the changes to the local database comprise a download, and wherein defining triggers comprises defining a trigger activated upon saving the download to the local database.
6. The method of claim 1, wherein defining triggers comprises defining a trigger for a delete call configured to not fail if a source MBO for the delete call is not present.
7. A computer-readable storage device having computer program logic recorded thereon, execution of which, by a computing device, causes the computing device to perform operations comprising:
   defining a mobile business object for a materialized view;
   defining triggers, within the mobile business object, for changes to a local database affecting the materialized view; and
   executing a code generator on the mobile business object definition.
8. The computer-readable storage device of claim 7, wherein the changes to the local database comprise at least one of a create, update, or delete call.
9. The computer-readable storage device of claim 7, wherein the changes to the local database comprise a bulk initial download.
10. The computer-readable storage device of claim 7, the operations further comprising:
    writing platform-specific code for the triggers to maintain the materialized view.
11. The computer-readable storage device of claim 7, wherein the changes to the local database comprise a download, and wherein defining triggers comprises defining a trigger activated upon saving the download to the local database.
12. The computer-readable storage device of claim 7, wherein defining triggers comprises defining a trigger for a delete call configured to not fail if a source MBO for the delete call is not present.
13. A system comprising:
    a memory configured to store modules comprising:
    a first defining module configured to define a mobile business object for a materialized view,
    a second defining module configured to define triggers, within the mobile business object, for changes to a local database affecting the materialized view, and
    an executing module configured to execute a code generator on the mobile business object definition; and
    one or more processors configured to process the modules.
14. The system of claim 13, wherein the changes to the local database comprise at least one of a create, update, or delete call.
15. The system of claim 13, wherein the changes to the local database comprise a bulk initial download.
16. The system of claim 13, further comprising:
    a writing module configured to write platform-specific code for the triggers to maintain the materialized view.
17. The system of claim 13, wherein the changes to the local database comprise a download, and wherein defining triggers comprises defining a trigger activated upon saving the download to the local database.
18. The system of claim 13, wherein defining triggers comprises defining a trigger for a delete call configured to not fail if a source MBO for the delete call is not present.

* * * * *
