A Method for Imposing Fine-grain Next Generation Access Control over Database Queries

David Ferraiolo, Serban Gavrila, Gopi Katwala, and Joshua Roberts
National Institute of Standards and Technology

1.0 Background

Relational Database Management Systems (RDBMSs) do not typically impose access control directly on its data. To restrict access to sensitive data that might reside in a RDBMS, controls are typically implemented at the application level. These controls take on many forms to include role-based access to "screens" with parameters that can be characterized, and subsequently used to formulate and issue SQL queries. SQL queries comprise four basic types of operations – Select, Insert, Update, and Delete – that respectively read, create, write, and delete data in tables. An important feature of RDBMSs is that they are able to specify criteria and extract and/or alter data that might reside in one or more tables, very efficiently. For example "give me all the employees over 50 years old that live in Virginia". In this paper we describe a method that leverages ANSI/INCITS Next Generation Access Control (NGAC) standard for imposing access control over database queries independent of the application while retaining database performance. In particular, the method automatically generates composite objects from a database schema and expresses and enforces access control policies in terms of those composite objects. The method uses NGAC as an authorization engine to manage access control policies and compute authorization responses. The method also includes an Access Manager for trapping and enforcing policy over SQL queries issued by applications and a Translator for converting SQL statements to NGAC inputs and converting NGAC authorization responses to either an access Deny or one or more permitted SQL statements.

2.0 NGAC Overview

The Policy Machine (PM) is an access control framework that served as the basis for the development of an ANSI/INCITS standard call Next Generation Access Control (NGAC). NGAC consists of:

- a standard set of data elements and relations that can be configured to express arbitrary access control policies in support of a wide variety of data services and applications;
- a generic set of operations that include read, write, operations that can be performed on resource data, and administrative operations for configuring (creating and deleting) the data elements and relations that represent policies; and
- a standard set of functions for computing access control decisions and enforcing policy over user access requests to perform read/write and its administrative operations.

NGAC is a flexible access control framework in that it can be molded in support of combinations of diverse access control policies. NGAC can often provide much of the same data service

functionality that is provided by existing application products and system utilities, such as file management, workflow, and internal messaging and with similar performance. An advantage of NGAC is that access control policies are comprehensively enforced over its data services, while the non-NGAC data service counterparts are not. Although it is possible to develop a NGAC relational DBMS data service with similar features of today's commercially available RDBMS products, the NGAC data service would pale in performance. Furthermore, the NGAC enabled RDBMS data service could not directly accommodate the broadly recognized SQL standard for accessing databases.

3.0 Method

The method provides a means of leveraging NGAC for expression and enforcement of access control policies over SQL queries for accessing data in tables, rows, and columns in existing RDBMS products. By leveraging NGAC the method provides a means of access control policy support that goes beyond the state of the art, with minimal impact on performance. It can impose forms of mandatory, discretionary, and history-based access control policies. The method also could be deployed external to a RDBMS, providing a general solution for a variety of RDBMS products or it could be implemented as a database-kernel loadable module.

Included among NGAC's data elements and relations used to express and enforce policies are Object Attributes. Object Attributes are containers that group and characterize data objects in diverse ways. Data objects and object attributes are placed into containers through an assignment relation. Vis. Figure 1, the method begins with an existing RDBMS schema that includes columns and tables that are automatically converted into NGAC corresponding object attributes and assignments.

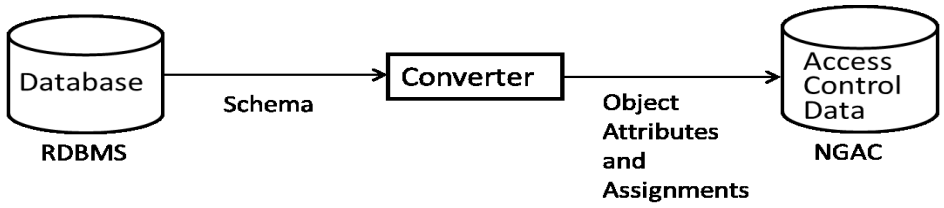

Figure 1. Converting Database Schema to NGAC Access Control Data

Because, rows are also object containers, existing rows could be automatically converted as well. Also included among NGAC data elements and relations are User Attributes, a generic set of operations, and three types of relations for specifying an access policy. Once the RDBMS schema has been converted, NGAC relations are configured in formulating policy in terms of the created object attributes and assignments, using NGAC's API. The resulting data elements and relations are stored as NGAC Access Control Data. In addition to the conversion and the additional data elements and relations, the method includes an Access Manager for trapping SQL queries from applications, and a Translator for converting SQL queries along with a user identity to NGAC inputs and NGAC authorization responses to those inputs to either an access Deny or permitted SQL queries.

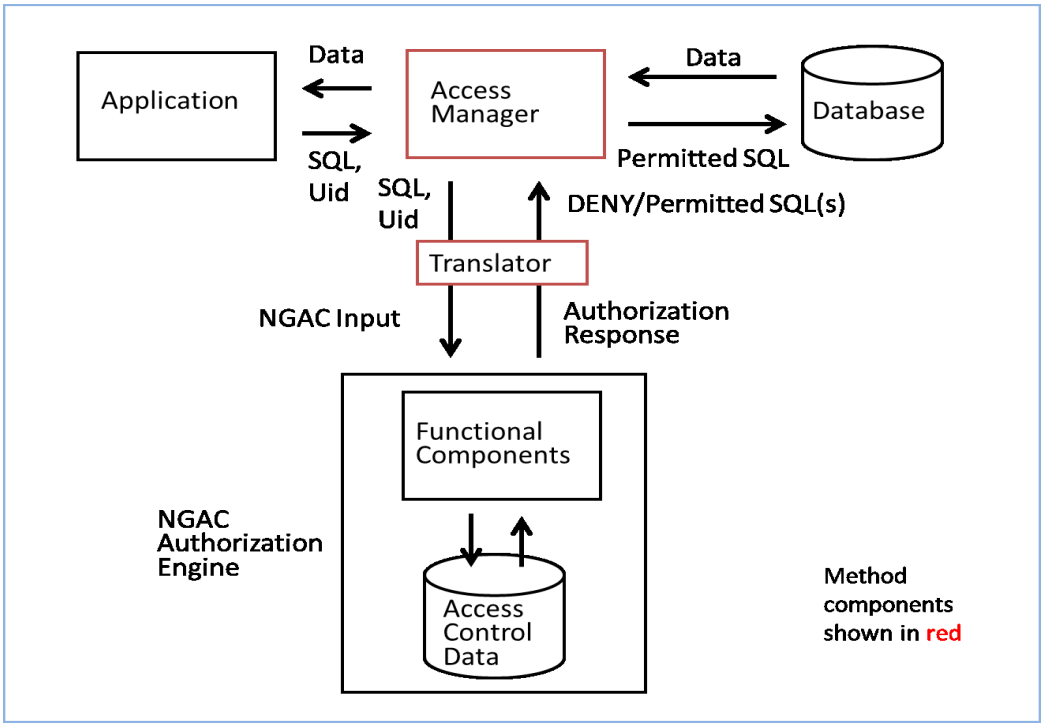

Figure 2. Placement of method with respect to existing components

Figure 2 shows the placement of the Access Manager and the Translator in the authorization flow that involve Applications, a target Database, and an NGAC authorization Engine. The authorization flow is as follows:

- The SQL statement from a user of the Application is intercepted by the Access Manager and sent to the Translator.
- The Translator coverts the SQL statement from the user into NGAC inputs that are fed to an NGAC implementation (engine).
- Using its Access Control Data, the NGAC implementation computes and renders an Authorization Response that is sent back to the Translator.
- Translator converts the Authorized Response into either an access DENY or one or more SQL Statements that are permitted for the user and are sent back to the Access Manager.
- The Access Manager submits the Permitted SQL Statements to the Database.
- In the case of a Select operation, Data extracted from the database is sent back to the Access Manager and forwarded to the Application and user.

Depending on the type of query (Select, Update, Insert, or Delete) the Translator issues different inputs to the NGAC Authorization Engine. These details are discussed later in the paper.

4.0 Expressing Access Control Policies in Terms of NGAC Object Attributes Corresponding to RDBMS Schema Objects

4.1 Basic Elements, Containers and Relations

NGAC access control data includes users, data objects, generic operations, and user and object attributes among its elements. NGAC treats both user attributes and object attributes as containers. Containers are instrumental in both formulating and administering access policies and attributes. NGAC expresses access policies through configurations of relations that include among others assignments (define membership in containers), associations (to derive privileges), and prohibitions (exceptions to privileges).

User attribute containers characterize their members. These containers can represent user names, roles, affiliations, or other common characteristics pertinent to policy, such as security clearances.

Object attribute containers characterize data by identifying collections of objects, such as those associated with certain projects, applications, or security classifications. Object containers can also represent tables, columns, and rows.

NGAC uses a tuple (x, y) to specify the assignment of element x to element y. The assignment relation always implies containment (i.e., x is contained in y).

Users and objects may be contained in one or more containers, and containers may be contained by or contain other containers. For object containers, this enables the representation of complex data structures such as relational database tables with distinguished fields. Rows of a table may be expressed as containers of data objects corresponding to the row's fields, and columns may be expressed as containers of data objects corresponding to column fields. Figure 3(b) illustrates a table using ovals to represent containers and dots to represent individual data objects. The vertically oriented ovals represent columns (Name, Phone, SSN, and Salary), and the horizontally oriented ovals represent rows (AliceRecord, BobRecord, and TomRecord), and their intersections represent fields in one or more tables. Figure 3(b) further illustrates a container of rows (Gr2Records) and two containers of columns (Public and Sensitive). All rows and all columns are represented by the object container EmployeeTable. Note that for this example, the containers shown in red are the object attributes that were automatically created by the Converter (see figure 1). All other NGAC elements and relations are assumed to be created through an NGAC administrative API by an authorized user. This authorized user may be a policy administrator, or as we discuss later, the user submitting Insert or Delete SQL queries.

Figure 3(a) illustrates user containers for the grouping and characterization of users. The container named Staff includes three users (u1, u2, and u4), and a container HR that includes two users (u3, and u5). In addition, figure 3(a) shows three containers Bob, Alice and Tom that respectively contain u1, u2, and u4. Finally, figure 3(a) shows Gr2Mng containing user u2.

NGAC recognizes a generic set of operations that include basic input and output operations (i.e., read and write) that can be performed on the contents of data objects, and a standard set of administrative operations that can be performed on NGAC data elements and relations that represent policies and attributes.

To be able to carry out an operation, one or more access rights are required. As with operations, two types of access rights apply: non-administrative access rights, and administrative access rights.

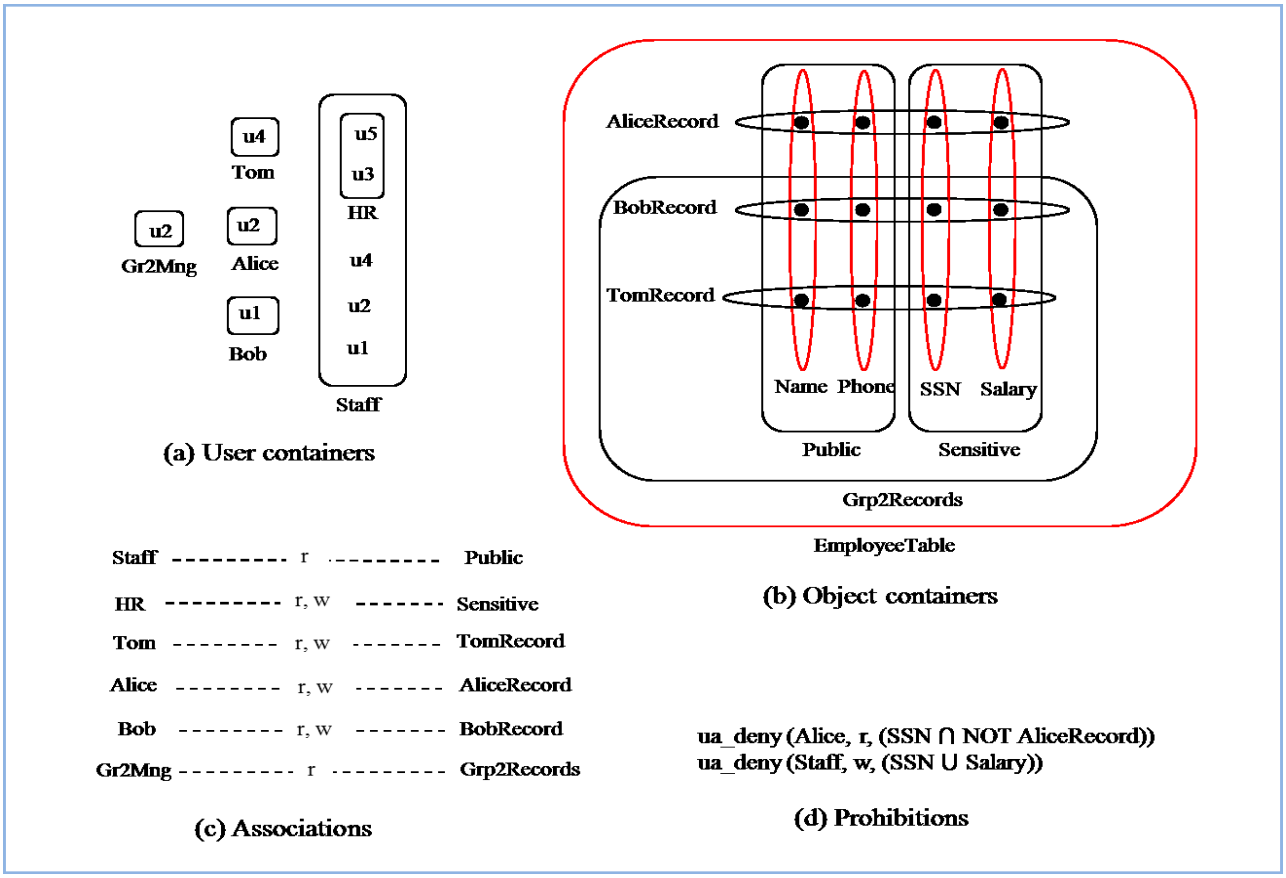

Figure 3. Example Access Policy Configuration

4.2 Associations

Access rights to perform operations are acquired through associations. An association is a triple, denoted by *ua---ars---pe*, where *ua* is a user attribute, *ars* is a set of access rights, and *pe* is a policy element, where *pe* may comprise either a user attribute or an object attribute. The policy element *pe* in an association is used as a referent for itself and the policy elements contained by the policy element. The meaning of the association *ua---ars---pe* is that the users contained in *ua* can execute the operations enabled by the access rights in *ars* on the policy elements referenced by *pe*. The set of referenced policy elements are dependent on (and meaningful to) the access rights in ars.

Figure 3(c) lists six association relations in terms of the user and object attributes (containers) illustrated in figures 3(a) and 3(b). Remember that the set of referenced policy elements are dependent on the access rights in ars. Note that the policy element of each association is an object attribute and the access rights are read/write. In the association HR---{r, w}---Sensitive, the policy elements referenced by Sensitive are data objects (the dots) contained in Sensitive, meaning that user u3 and u5 can read and write those objects. If we had an association HR---{create assign-to}---Sensitive, where "create assign-to" is an administrative access right, then the policy elements referenced by Sensitive would be Sensitive, SSN, and Salary, meaning that users u3 and u5 may create assignments to Sensitive, SSN, or Salary.

The access policy specified by the list of associations in figure 3(c) is as follows:

- Staff users can read Name and Phone fields of all records in EmployeeTable.
- In addition to being able to read Name and Phone fields, HR users can read and write SSN and Salary fields of all records in EmployeeTable.
- Bob, Tom, and Alice can read and write all fields (SSN, Salary, Name, and Phone) in their own record (respectively, BobRecord, TomRecord, and AliceRecord).
- Gr2Mng can read all fields (SSN, Salary, Name, and Phone) of all records in Gr2Reccords (i.e., BobRecord and TomRecord)

4.3 Prohibitions

In addition to assignments and associations, NGAC includes three types of prohibition relations. In general, prohibition relations specify privilege exceptions. One of these relations is user attribute-deny. We denote the user attribute-based deny relation by ua_deny(*ua*, *ars*, *pes*), where *ua* is a user attribute, *ars* is an access right set, and *pes* is a policy element set used as a referent for policy elements contained by the policy element(s). The meaning of the relation is that the users assigned to *ua* cannot execute the operations enabled by the access rights in *ars* on the policy elements in *pes*.

The prohibitions listed in figure 3(d) further constrain the access policy as follows:

- Staff users can read Name and Phone fields of all records in EmployeeTable.
- In addition to being able to read Name and Phone fields, HR users can read and write SSN and Salary fields of all records in EmployeeTable.
- Bob, Tom, and Alice can read all fields (SSN, Salary, Name, and Phone) and write to Name and Phone fields in their own record (respectively, BobRecord, TomRecord, and AliceRecord).
- Gr2Mng can read all fields of all records in Gr2Records with the exception of the SSN field.

An example set of Employee Records with data content is shown in the top table of figure 4 under the object containers depicted in figure 3(b). The bottom three tables show the access capabilities for users u1, u2, and u3, under the access control policy expressed in figure 3, where read access is highlighted in black, and read/write access is highlighted in red.

Name	Phone	SSN	Salary
Bob	301-976-4454	122-54-4537	$38,341
Alice	301-976-3042	945-39-4034	$72,440
Tom	301-976-2067	304-75-3995	$62,550

Name	Phone	SSN	Salary
Bob	301-976-4454	122-54-4537	$38,341
Alice	301-976-3042		
Tom	301-976-2067		

Name	Phone	SSN	Salary
Bob	301-976-4454		$38,341
Alice	301-976-3042	945-39-4034	$72,440
Tom	301-976-2067		$62,550

Name	Phone	SSN	Salary
Bob	301-976-4454	122-54-4537	$38,341
Alice	301-976-3042	945-39-4034	$72,440
Tom	301-976-2067	304-75-3995	$62,550

Figure 4. Example set of records with data content and the access capabilities for users u1, u2, and u3 under the access control policy of figure 3

5.0 Translator

As discussed in section 3, the method includes a Translator. The Translator on one side converts a SQL statement generated by an application and the identity of the application's user to an NGAC input. On the other side, the Translator takes an NGAC authorization response to the input, and converts it to either one or more permitted SQL statements or an access DENY in the case of a Select statement, or to a GRANT or DENY status in the case of an Update, Insert, or Delete statement. The Translator treats Select and Update operations differently than Insert and Delete Operations. This is because Select and Update operations are directly mapped to NGAC read and write operations on data, and Insert and Delete operations are mapped to create and delete administrative operations on NGAC object containers that correspond to rows.

5.1 Select and Update

Select SQL statements include a specification of one or more tables and one or more columns

from those tables along with criteria for identifying rows from the table(s). Update SQL statements include a specification of one table with one or more columns with criteria for identifying rows. The method to translate a user's requested Select statement to one or more permitted SQL statements or an Update statement to a GRANT or DENY result is based on NGAC's ability to review the access capabilities of users. In particular, the method identifies a set of objects that are accessible to a user for either read for Select or write for Update, as well as attributes that contain those objects. In the algorithms that follow by row, column, table we mean object attributes that correspond to those entities. Possible algorithms for Select and Update are as follows:

For Select:
(1) Identify a set of rows in the table of the Select SQL statement that contain objects accessible by the user under the read operation.
(2) For each row identified in (1), identify a maximal set of columns that: is a subset of the columns in the Select statement and each identified column contains an object that is also contained in the row. These columns are said to be associated with the row.
(3) For each row, column association, remove the columns that are also included in any deny relation for the user with respect to read.
(4) For each subset of identified rows so that each row in the subset has a common associated set of columns, generate a Select SQL statement for that set of columns with the original table and original condition augmented with a condition that limits the Select to the subset of identified rows.
(5) If the set of rows or columns are empty the Translator issues a DENY response.

For Update:
(1) Identify the set of rows in the SQL database that meet the criteria included in the Update SQL statement.
(2) Identify a set of rows in the table of the Update SQL statement containing objects accessible by the user under the write operation.
(3) If the rows identified by (1) are a subset of those identified in (2), proceed to (4), otherwise DENY access.
(4) For each row identified in (1), verify the existence of objects common to the row and the set of columns included in the SQL Update statement. If the condition fails, DENY access, otherwise proceed to (5).
(5) For the columns included in the SQL Update statement, verify that the columns are not included in any deny relation for the user. If the condition holds, GRANT the SQL Update Statement, otherwise DENY access.

5.2 Delete and Insert

The execution of a SQL Delete statement removes one or more rows from a table in accordance with criteria included in the statement. The method Grants or Denies a user's request to delete one or more rows in a database table and in the case of a Grant, subsequently deletes the corresponding NGAC object attributes and relations. The execution of a SQL Insert statement creates a new row with specified column values, in a specified table. The method either Grants or Denies a user's request to insert a row in the database, and in the case of an Grant, subsequently creates an NGAC object attribute corresponding to the row and creates objects (representing the

values) and assigns those objects to the row attribute and appropriate column attributes. A user's capability to perform a SQL Delete or Insert operation is dependent on the existence of administrative privileges.

Creating and deleting objects, object attributes and assignments are achieved through execution of administrative operations. A users capabilities to execute administrative operations are established through administrative privileges.

5.2.1 Administrative Operations

Administrative operations are implemented using parameterized routines, prefixed by a precondition, with a body that describes how a data set or relation (denoted by Y) changes to Y′. The precondition tests the validity of the actual parameters. If the condition evaluates to false, then the routine fails:

Rtnname (x_1, x_2, …, x_k) {
…preconditions…
{
$Y' = f(Y, x_1, x_2, \ldots, x_k)$
}

Consider, as an example, the administrative operation CreateOinOA shown below, which specifies the creation of an object x and assigning the object to an object attribute y. The preconditions here stipulate that x parameter is not a member of objects (O) and y parameter is a member of object attributes (OA). The body describes the addition of the x to the set of objects (O) which changes the state of the set to O' and the addition of the tuple (x, y) to the set of assignments (ASSIGN) relation, which changes the state of the relation to ASSIGN'.

CreateOinOA(x, y)
$x \notin O \wedge y \in OA$
{
O' = O ∪ {x}
ASSIGN' = ASSIGN ∪ {(x, y)}
}

Each administrative routine entails a modification to the NGAC configuration.

5.2.2 Administrative Privileges

In order to execute an administrative operation the requesting user must possess appropriate access rights. Just as access rights to perform read/write operations on data objects are defined in terms of associations, so too are capabilities to perform administrative operations on policy elements and relations.

For example, consider the following two associations in support of the configuration depicted by Figure 3(b):

TableAdmin --- {*create-oa*, *create-o, create ooa*} ---EmployeeTable

TableAdmin --- {*delete-o*, *delete-oa*, *delete-ooa, delete-oaoa*}--- EmployeeTable

The meaning of the first association is that a user assigned to TableAdmin can:

(1) create an object attribute (e.g., corresponding to a row) assigned to an object attribute (e.g., EmployeeTable) in EmployeeTable;
(2) create an object assigned to an object attribute (e.g., an existing row) in EmployeeTable; and,
(3) create an object to object-attribute assignment from an object (e.g., an object in a row) to an object attribute (e.g., corresponding to a column) in EmployeeTable.

The meaning of the second association is that a user assigned to TableAdmin can:

(1) delete an object to object-attribute assignment (e.g., delete object assignments to attributes corresponding to a row and column) in EmployeeTable;
(2) delete an object in EmployeeTable;
(3) delete an object-attribute to object-attribute assignment (e.g., a row assigned to EmployeeTable) in EmployeeTable; and
(4) delete an object attribute (e.g., corresponding to a row) in EmployeeTable.

5.2.3 Administrative Routines

The administrative operations necessary to insert or delete an object container corresponding row in another object container corresponding to a table do not need to be executed on an individual basis, but instead can be executed as an NGAC administrative routine.

An *administrative routine* consists mainly of a parameterized interface and a sequence of administrative operation invocations. The body of an administrative routine is executed as an atomic transaction—an error or lack of user privileges that causes any of the constituent operations to fail execution causes the entire routine to fail, producing the same effect as though none of the operations were ever executed.

The following routine (in the context of figure 3(b)) creates an object attribute (corresponding to a row) assigned to EmplyeeTable, creates new objects (corresponding to values), and assigns those objects to object attributes (corresponding to columns) and the object attribute corresponding to the row. Assume the columns Name, Phone, SSN, and Salary already exist and are assigned to the object attribute EmployeeTable.

```
Insert_Row_in_EmployeeTable(row, name, phone, ssn, salary)
{   CreateOAinOA(row, EmployeeTable)
    CreateOinOA(name, row)
    Assign(name, Name)
    CreateOinOA(phone, row)
    Assign(phone, Phone)
    CreateOinOA(ssn, row)
    Assign(ssn, SSN)
```

```
    CreateOinOA(salary, row)
    Assign(salary, Salary)
}
```

Although the Insert routine applies to the object attributes corresponding to the example schema of figure 3, a similar and corresponding routine could be automatically created for each table of an RDBMS schema or a generic Insert routine could exist that uses a template specific to each table.

An administrative Delete routine could be used to delete an object attribute, objects and assignments corresponding to a RDMBS row, and column values. Consider, for example the following routine in the context of figure 3(b):

```
Delete_Row_from_EmployeeTable(row)
{   For each object obj in row {
      DeleteO (obj) /*includes deletion of assignments of obj*/
    }
    DeleteOAinOA(row, EmployeeTable) /*includes deletion of assignments row to
      EmployeeTable*/
}
```

Similar to Insert a Delete routine could be automatically created for each table of an RDBMS schema or a generic Delete routine could exist that uses a template specific to each table.

Administrative routines not only allow consistence between RDBMS rows and corresponding NGAC object attributes, objects, and assignments, but also provide a means for testing a user's authority to Insert and Delete RDBMS rows.

For Insert:
The algorithm for translating an Insert statement to an NGAC authorization response assumes the existence of an NGAC administrative Insert routine. The algorithm is as follows:

(1) Invoke the routine corresponding to the table specified in the Insert statement, using the identity of the user that issued the Insert statement with the specified row, and column values, thereby creating an object attribute that corresponds to the row, creating objects that represent and correspond to column values that are assigned to the row and are appropriately assigned to object attributes that correspond to columns.
(2) If the routine successfully executes, GRANT the SQL Insert statement, otherwise DENY access.

For Delete:
Similarly to insert, the algorithm for translating a Delete statement to an NGAC authorization response assumes the existence of an NGAC administrative Delete routine, particularized for the referenced table. The algorithm is as follows:

(1) Identify the set of rows in the SQL database that meet the criteria included in the Delete

SQL statement.

(2) For each row identified in (1), sequentially invoke, using the identity of the user that issued the statement, the Delete routine of the table specified in the Delete statement, using and caching the parameters of the object attribute corresponding to the identified row, and the objects contained in the object attribute.

(3) If any invocation of the routine fails to successfully execute, DENY the SQL Delete statement, and roll back changes due to previous invocations by applying the cache as NGAC administrative Insert routine parameters, otherwise GRANT.