



IBM Hippocratic Database Active Enforcement

User Guide

Version 1.0 – Including Technical Appendices

1. Introduction

The Hippocratic Database's (HDB) Active Enforcement component operates as a middleware layer, ensuring that enterprise applications accessing a database adhere to fine-grained data disclosure policies. These data disclosure policies are determined by the company's own policy (i.e., security policy, privacy policy, and data management policy), legal and regulatory requirements, customer preferences, and opt-in and opt-out choices. HDB automatically rewrites user queries to return only data that is consistent with these policies, allowing applications to enforce disclosure policies on arbitrary data elements at the time the queries are executed. Active Enforcement includes two main phases – policy creation, and application data retrieval.

In the *policy creation stage*, the company adopts a data disclosure policy to specify (i) who is allowed to access what information; and (ii) for what purposes information may be disclosed to each recipient. Policies are specified using a GUI and stored in database metadata tables.

In the *application data retrieval phase*, all queries to be executed on the data source are transformed by HDB so that the application only retrieves results that are compliant with company disclosure policies and customer preferences (e.g., opt-in and opt-out choices). This process is fully automated and is only dependent upon policies.

The key strengths of Active Enforcement are that: (i) it offers a general methodology for handling and codifying policy information; (ii) its policy enforcement is transparent to enterprise applications (integration assumes a database interface such as ODBC or JDBC); (iii) it is agnostic to underlying database technology; (iv) it allows policy changes without any application rewrites; and (v) in the typical case, it improves query processing speed since the selectivity of the transformed query is higher than that of the original query due to the introduction of policy related predicates into the transformed query.

2. System Requirements and Installation

HDB requires that applications access a backend database using JDBC and SQL92-compliant queries. HDB requires DB2 Version 8. Integration of the Active Enforcement component into a company's infrastructure requires: (i) creating a disclosure policy with user and purpose specifications; (ii) installing the disclosure policy using the HDB Control Center (HDBCC); and (iii) installing the appropriate HDB enforcement JDBC drivers. The HDBCC application requires the variables `JAVA_HOME` and `DB2TEMPDIR` to be defined. `DB2TEMPDIR` is set by default with DB2 8 version. It points to `SQLLIB` directory of DB2 installation directory. Create a directory (e.g., `hdb`) and unzip the `hdbcc.zip` file there. The HDBCC can be started using the `runcc.bat` file located in the `hdb` directory and using a command line. The HDB JDBC driver is located in the `zip` and `jar` files in the `lib` subdirectory. Before connecting to a database using the HDBCC, increase the `applheapsz` parameter for that database using the following `db2` command. Otherwise, the HDBCC may not operate properly.

```
db2 update database configuration for db using applheapsz 2048
```

3. Policy Installation

A company privacy officer begins the policy installation process by creating a data disclosure policy using the HDB Control Center. The HDB Control Center automatically generates the metadata tables for enforcement and populates them with the disclosure policy created. Below is a walkthrough of the policy creation process using the HDB Control Center.

1. **Login:** Upon launching the HDB Control Center, the user is asked to log into the database system (Figure 1) and connect to a selected database. The current version of the Control Center supports DB2 databases, but will be extended to connect to other types of data sources in the future.¹

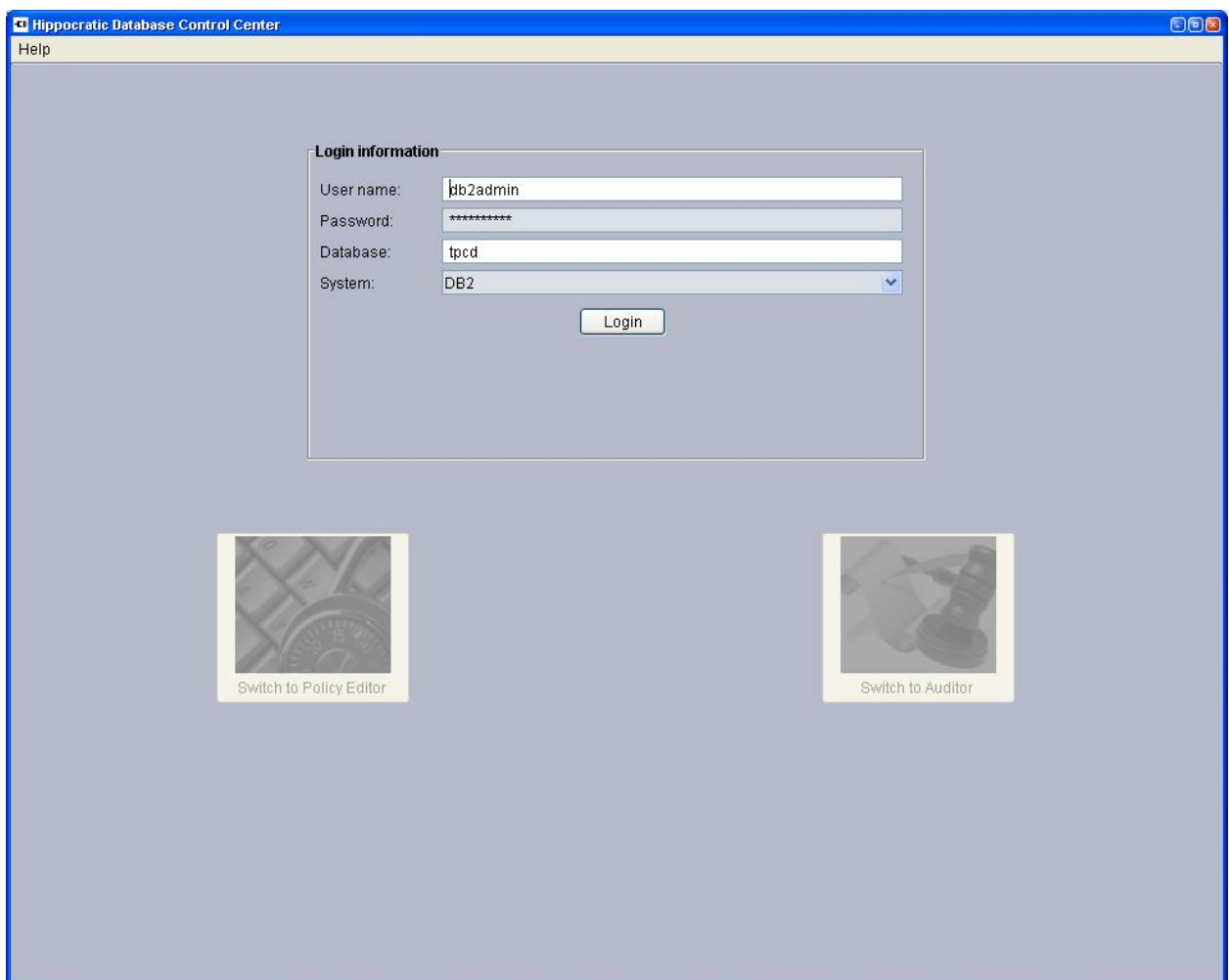


Figure 1. HDB Control Center Login Screen

¹ HDB currently supports both type 3 and 4 jdbc drivers. For type 3 drivers, type in the database name in the “Database Name” for connection, for type 4 (jcc) drivers, specify both the database name and the port used to connect to the database in the “Database Name” field, i.e. <db name>:<port name>. To enable this feature, change the line jdbc.usejcc to true in the file resources/hdb.properties.

2. **Task Selection:** If the login is successful, the user clicks on the “Policy Editor” button to work on disclosure policies for the selected database (Figure 2).

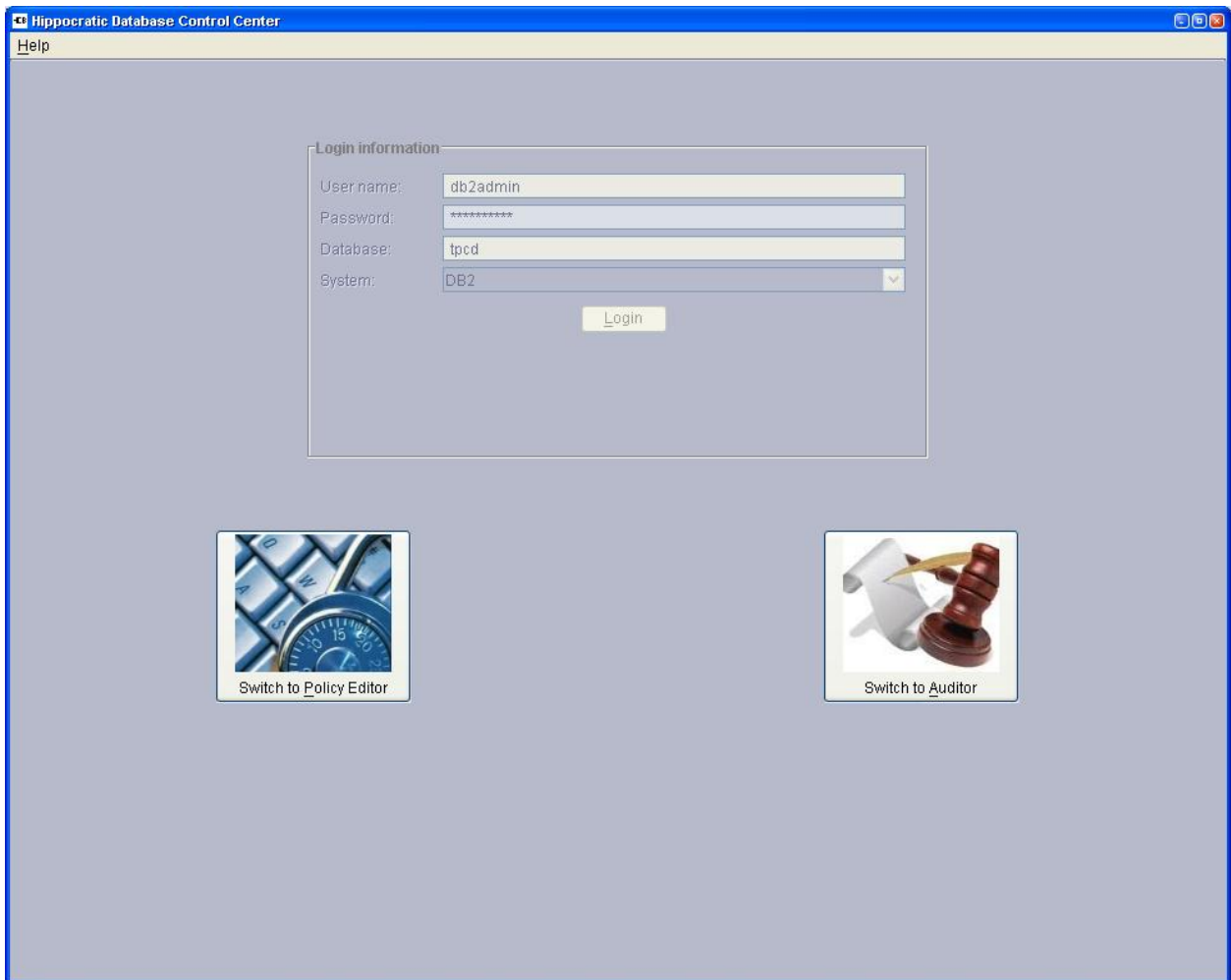


Figure 2. HDB Control Center Task Selection Screen

3. **Policy Editor:** The Policy Editor (Fig 2a) consists of two panels. The left panel shows the list of databases that are currently connected and the policies for those databases. When a policy version is selected, the panel on the right displays the disclosure rules for that policy. The left panel contains a Database tree which shows all the policies defined for the current database. The versions under a policy will also be shown as nodes under the Policy tree. The Applications, Entities, purposes, accessors, recipients for a database can be visually identified by using different icons. Different types of policies and versions have different icons for visual identification.

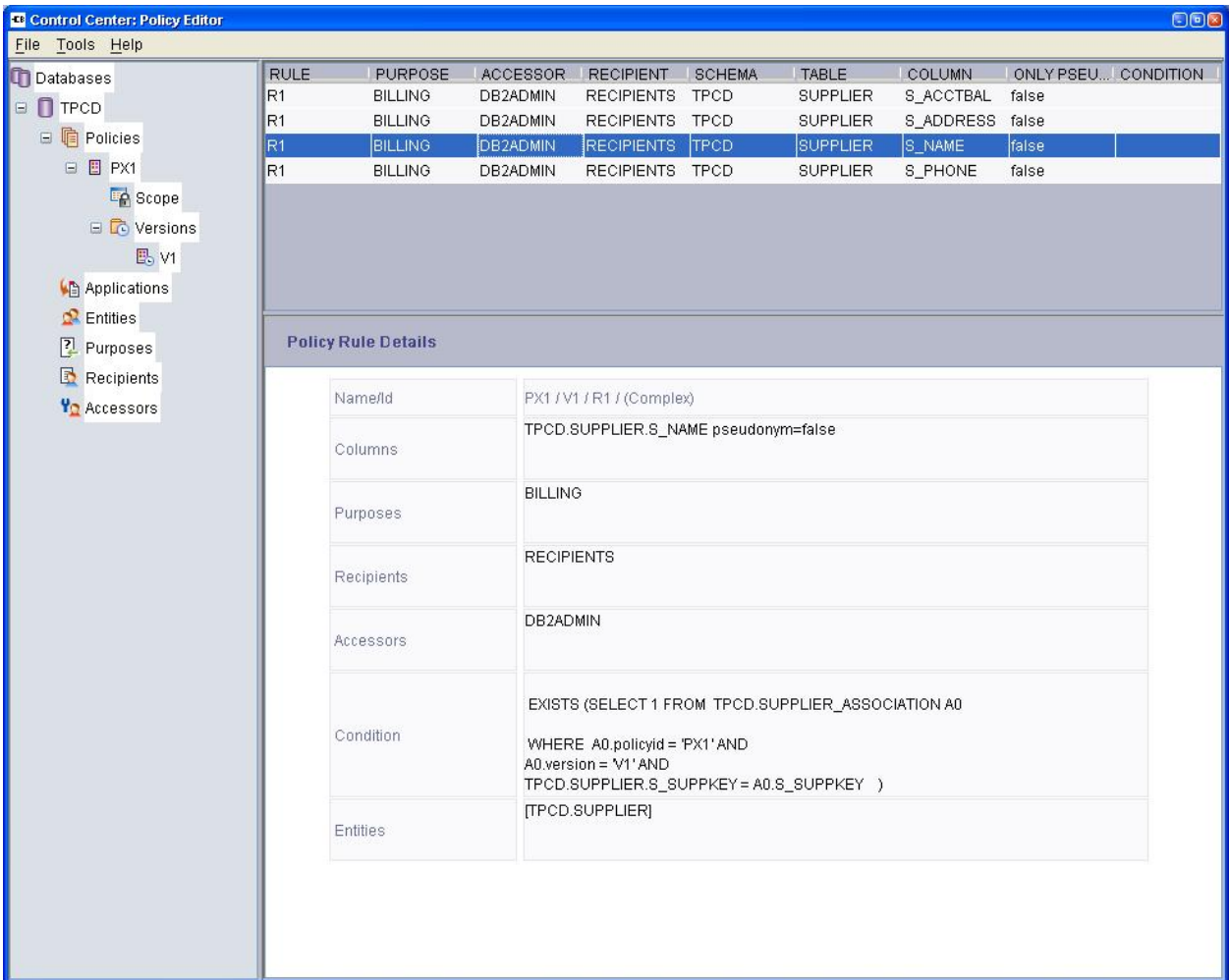


Figure 3a. Policy Editor

4. **Connecting to other databases:** After right clicking the “Database” node, the user can select “Connect to another database” to connect to additional databases on the right pane (Figure 3). Upon successful login, the newly connected database will be added to the panel on the left.

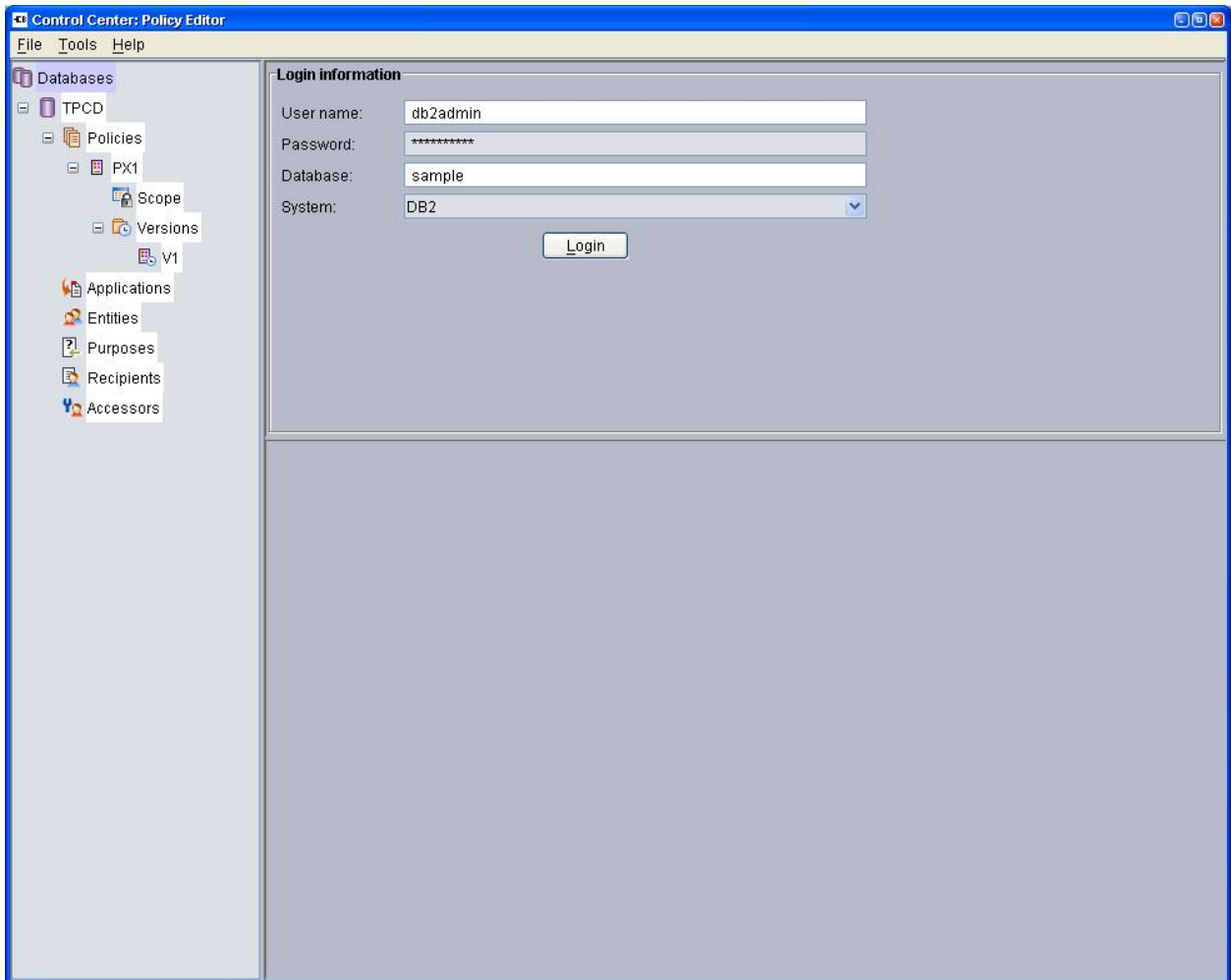


Figure 4. Additional Database Connection Screen

5. **Install Policy Metadata:** If HDB policy metadata tables do not exist on the database, the user must right click on the node with the database name and select “Install Policy Metadata Tables.” In response, the Control Center will generate appropriate SQL scripts to create the metadata tables.
6. **Define Entities:** After the HDB policy metadata tables are created (or already exist in the database), the first step in creating a new disclosure policy is to define new **entities** by right clicking on the “Database” node (Figure 4) and selecting “Define Entities.” An *entity* is a table with tuples that represent individuals who subscribe to specific versions of policies, and who make policy opt-in/out choices. For example, in a table patientinfo representing individual patient information, the entity would be the patientinfo table. Technically, for tables selected as entities, two additional tables are created; one stores individual policy subscription choices, the other stores individual opt-in/out choices. The primary key of these two tables is created to match the primary key of the selected entity table. In the patient information example, suppose each patient is uniquely identified by the patient number (e.g., column “pno”) stored in the “patientinfo” table. The policy subscription table that is created is “patientinfo_association(pno, policyid, version)” and its primary key is also “pno”. For example, the tuples <‘1’,‘P1’, ‘V1’>, <‘1’,‘P2’,‘V1’>

represent that the patient with pno 1 subscribes to policy P1 version V1 and to policy P2 version V1. When using complex policy versioning, the HDB CC will generate policy rule conditions that check whether the entity subscribes to the policy/version of the rule being defined. These conditions will then be introduced into transformed queries when the active enforcement component modifies a query to enforce this policy rule. The choice table that is created is “patientinfo_choices(pno, choiceid, value)” and its primary key is similarly “pno”. Opt-in/out choices of individuals recorded in this table can be referenced in policy rule conditions to control disclosure based upon individual choices. For example, the tuples <'1', 'telemarketing', '0'>, <'1', 'research', '1'> represent that the patient with pno 1 agrees to the research choice but declines the telemarketing choice. While association and choices tables are referenced in policy rules, it is the responsibility of applications to populate and update these tables in conjunction with the entity table. For example, if an application adds a new patient to patientinfo, the application should also add the policies to which the patient subscribes to in the association table and the individual choices made by the patient in the choices tables. If there are other tables storing information related to the entity table, these tables should not be defined as separate entities. For example, the treatment table stores hospital treatments received by individual patients, and the table has a foreign key, pno, to the patientinfo table. No association or choices tables are needed for treatment since policy subscription and individual choices for columns of treatment are made by patients and recorded in patientinfo_association, and patientinfo_choices. Policy conditions generated with complex versioning will use foreign-key pno in the treatment table to access the subscriptions and choices of patients regarding columns of treatment.

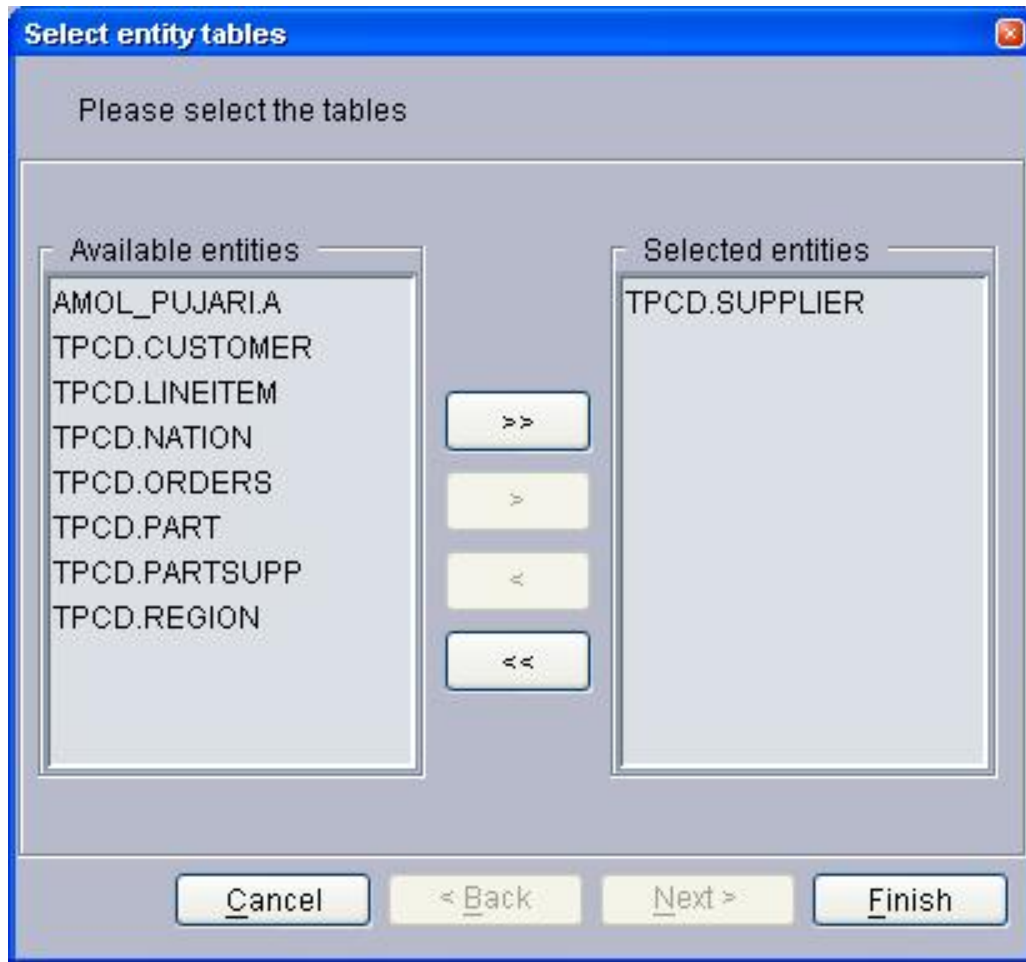


Figure 5. Entity Definition Screen

7. **Creating a Policy:** Next, the user creates a new disclosure policy by selecting “Add Policy” under the “Policies” node.

Users can not select Create Policy without first installing policy metadata. If a user wants to add a policy to a database for which the policy metadata is already installed, Create Policy will start the wizard which will allow user to add new policy (Figure 5).

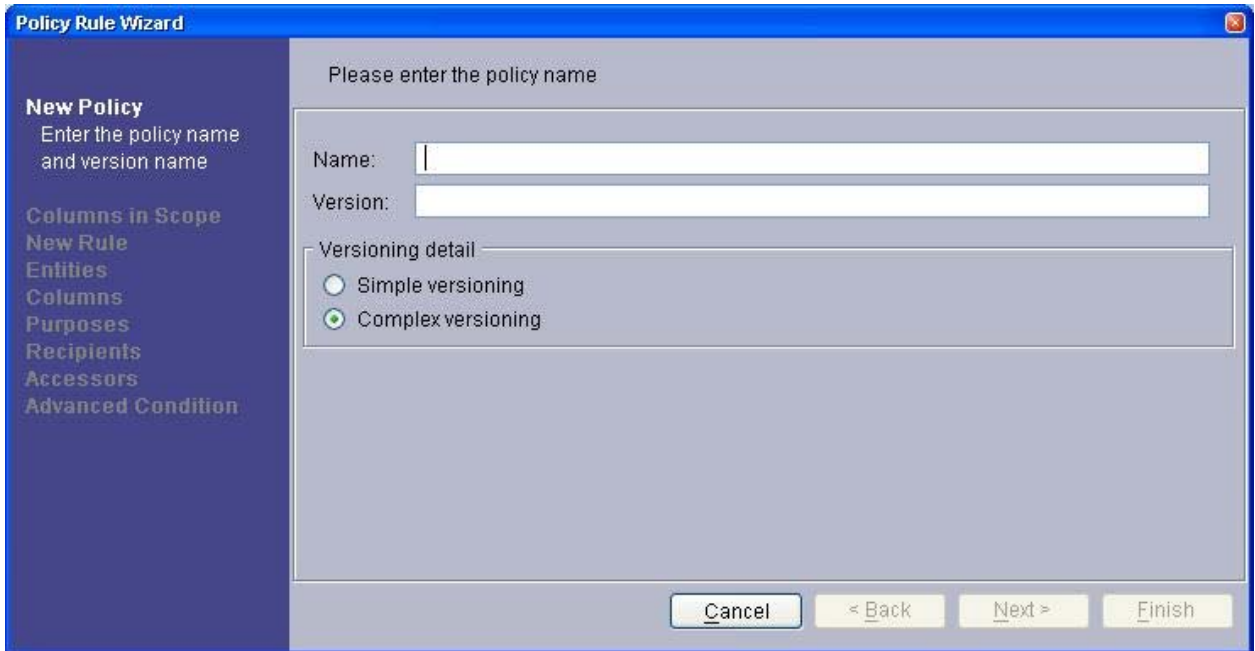


Figure 6. New Policy Creation Screen

8. **Defining Scope for a Policy** After naming the policy, the user defines the policy scope. The scope specifies which columns are governed by the selected policy (Figure 6). For any column within a policy's scope, no data will be disclosed if the policy does not define specific disclosure rules for that particular column. If multiple versions of the same policy govern the disclosure of a column; with complex versioning, the policy allows disclosure if any of the version allow disclosure. However, using complex versioning, individuals subscribe to only one version of any policy. Therefore, disclosure will be restricted to individuals subscribing to versions of the policy that indeed allow disclosure. If simple versioning is in effect for a policy, only one version is active at any point in time; other versions have no effect on policy enforcement. If a column is within the scope of multiple policies, data in that column will be disclosed only if all the involved policies permit disclosure.

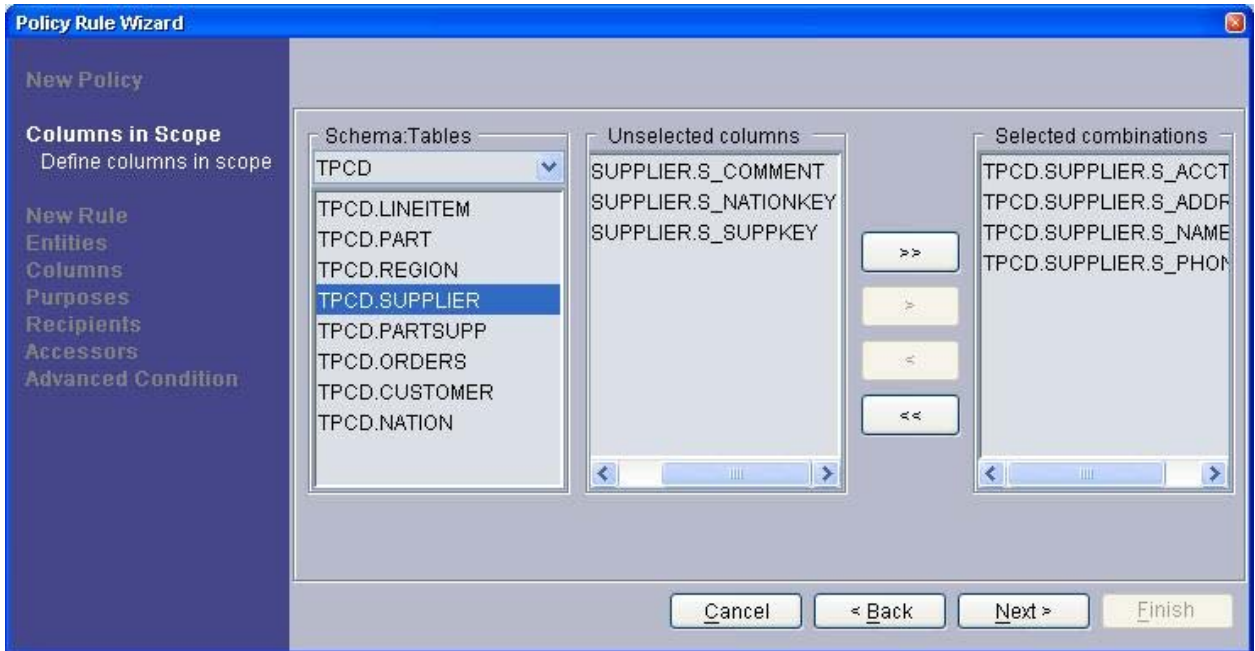


Figure 7. Scope Selection Screen

9. **Adding Disclosure rules to a Policy** The next step in the process is to create new disclosure rules for the selected policy. To launch the Disclosure Wizard (Figure 7), the user right clicks on the policy name in the Database tree and selects “Create Rule.” The first screen in the wizard goes through the process of creating a name for the rule.

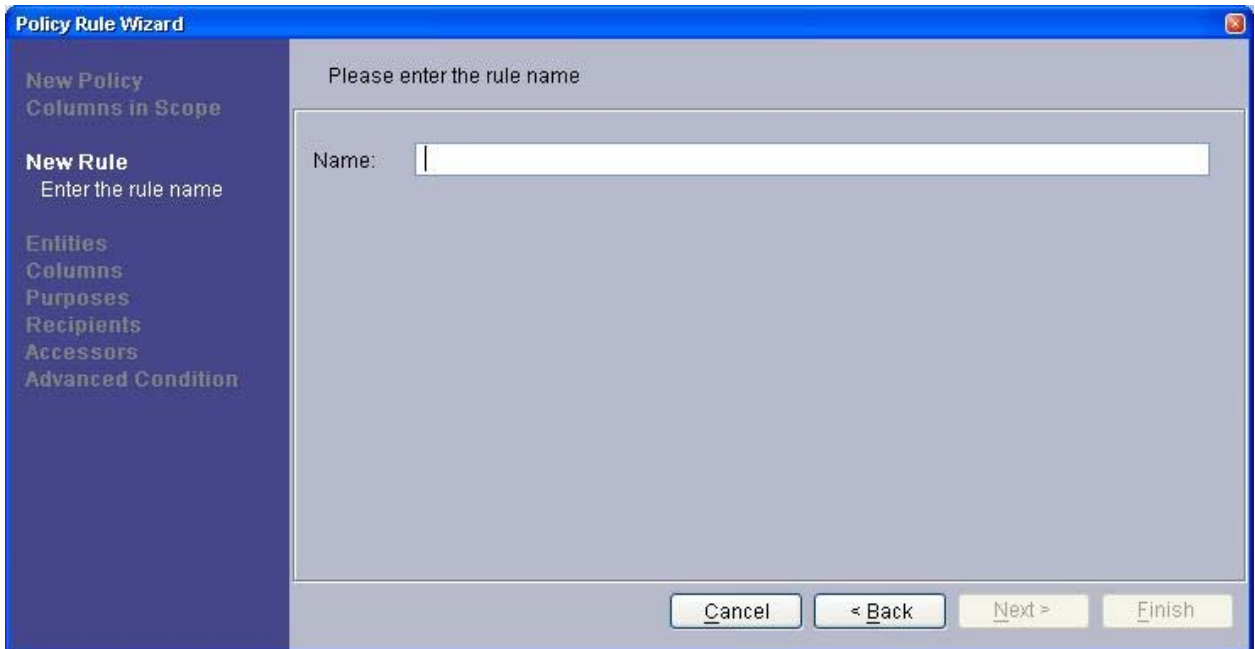


Figure 8. Rule Naming Screen

10. **Entity Associations** The user then selects the entities for which each rule applies (Figure 8). To support multiple disclosure policies and multiple versions of policies, HDB maintains an association table in its metadata. This table stores the names of the policies and versions to which each entity subscribes. During application data retrieval, the association table is consulted to ensure that the entity has subscribed to the policy rule that is being enforced. The rule wizard automatically generate the condition expression for the rule to check subscription by the entity.

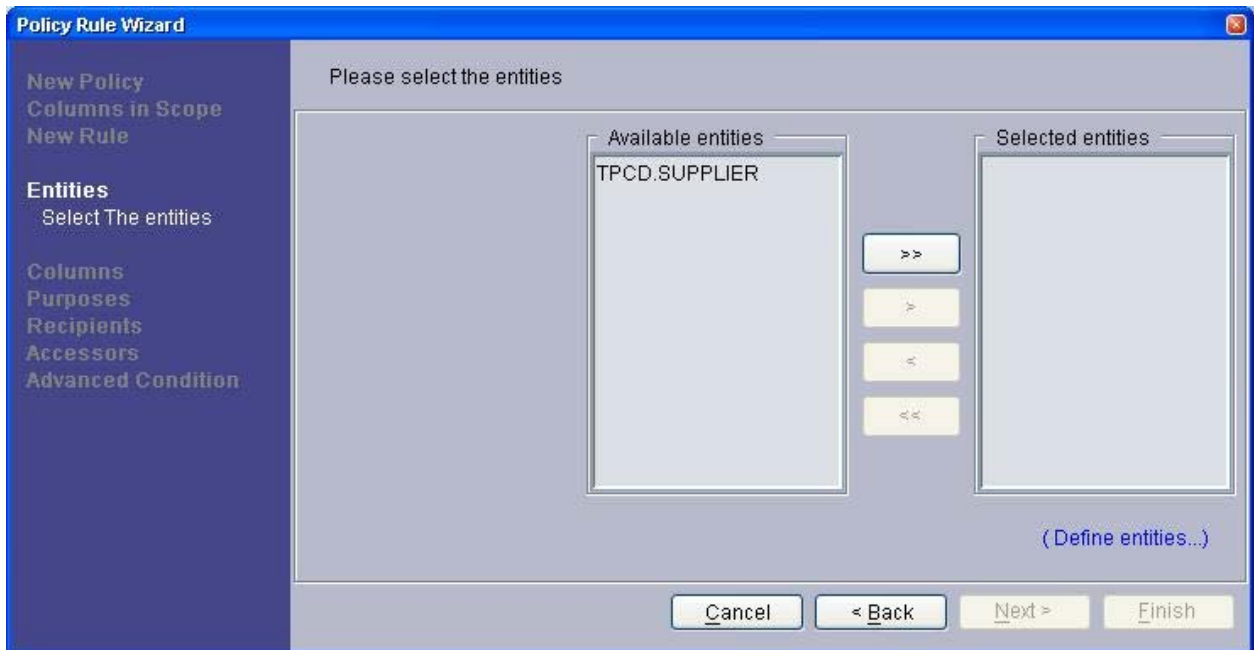


Figure 8. Entity Selection Screen for Rule Creation

11. **Column Selection and Pseudonyms** Next, the set of columns that are included in the rule are selected (Figure 9). Optionally the user can also select the columns whose access is limited to pseudonyms only by checking the box next to the column name in the lower panel. The pseudonyms are created by HDB using db2 encryption schemes based on password provided by the user.

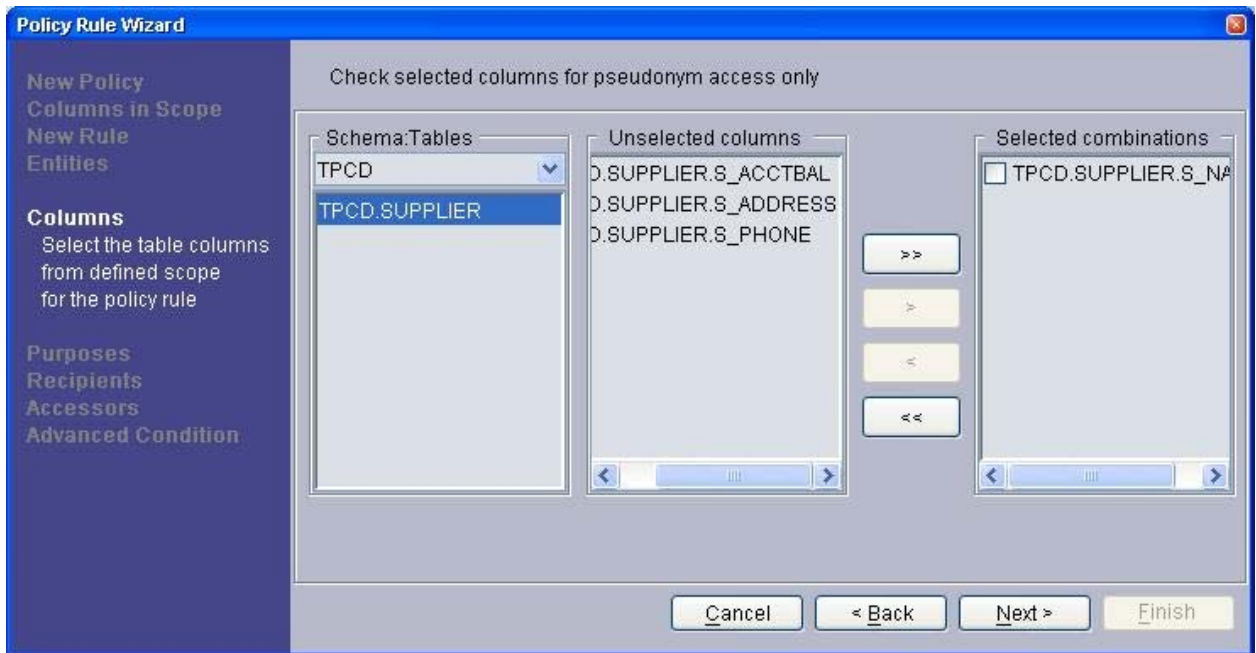


Figure 9. Column Selection Screen

12. Purposes, Accessors and Recipients The next three steps define the purposes, recipients, and accessors for the disclosure rule (Figures 10 – 12).

Purpose refers to the use for which data is requested from the database, and is generally obtained from the application that submits the query. Recipients are the parties who will receive the query results. The selected purposes, recipients are those allowed by the rules being created.

Accessors are the authenticated users submitting queries to the database. Multiple accessors may be selected in the wizard for each disclosure rule.

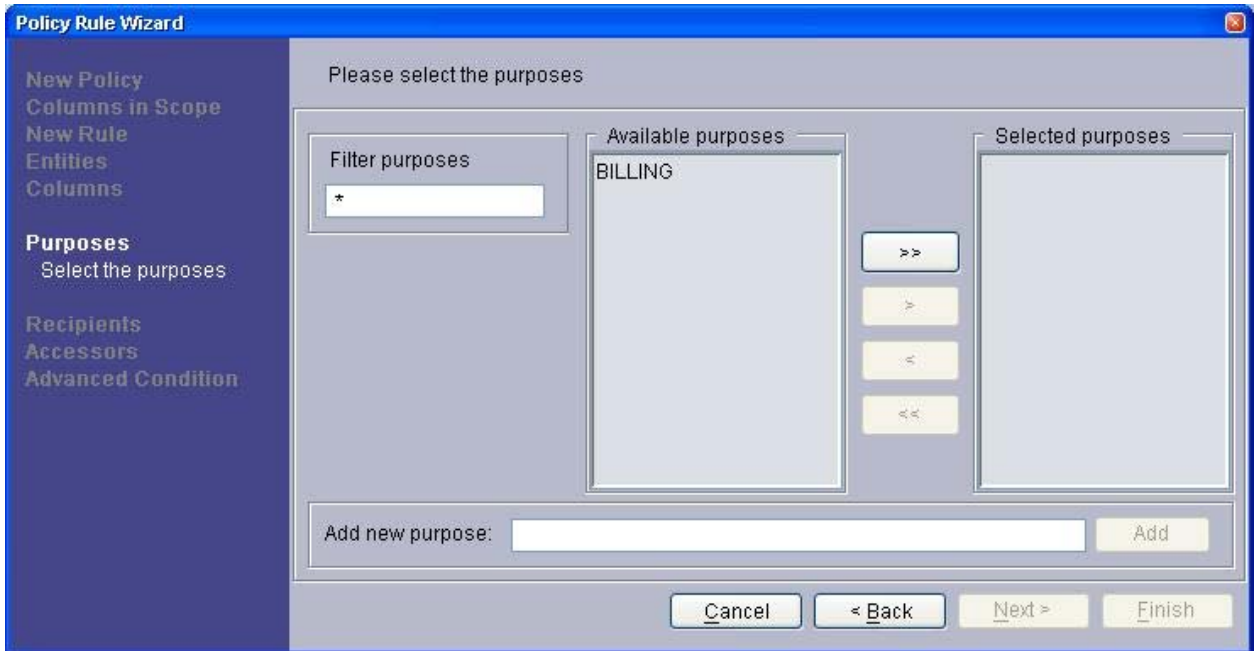


Figure 9. Purpose Selection Screen

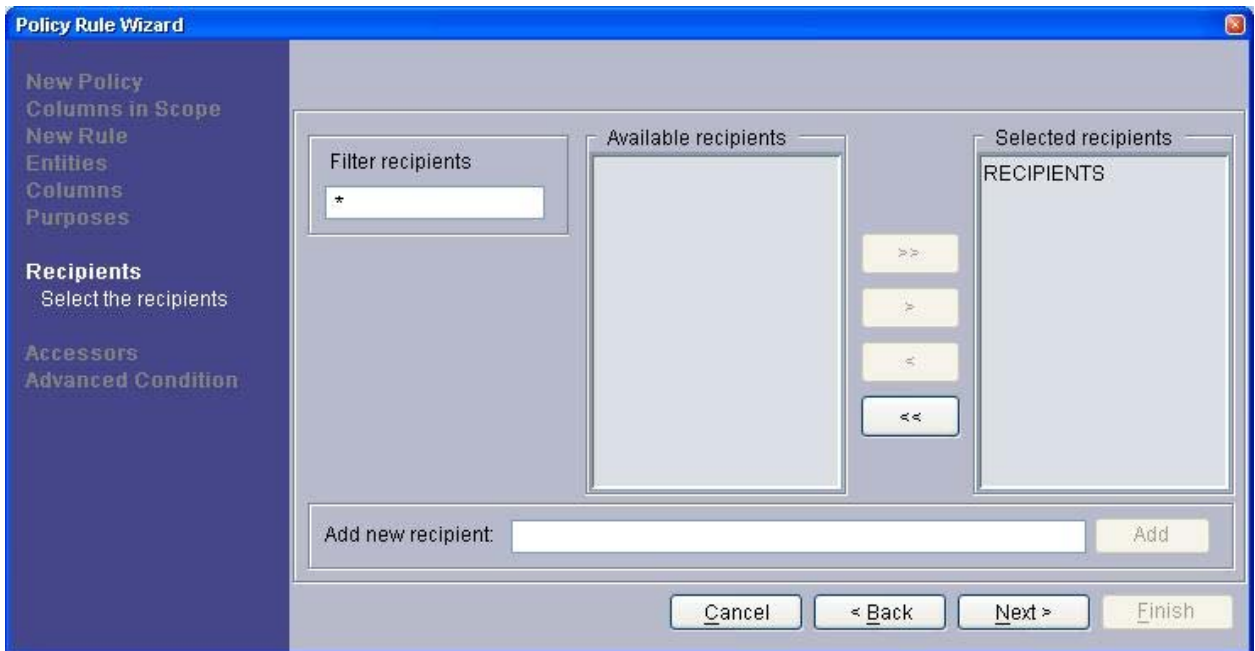


Figure 10. Recipient Selection Screen

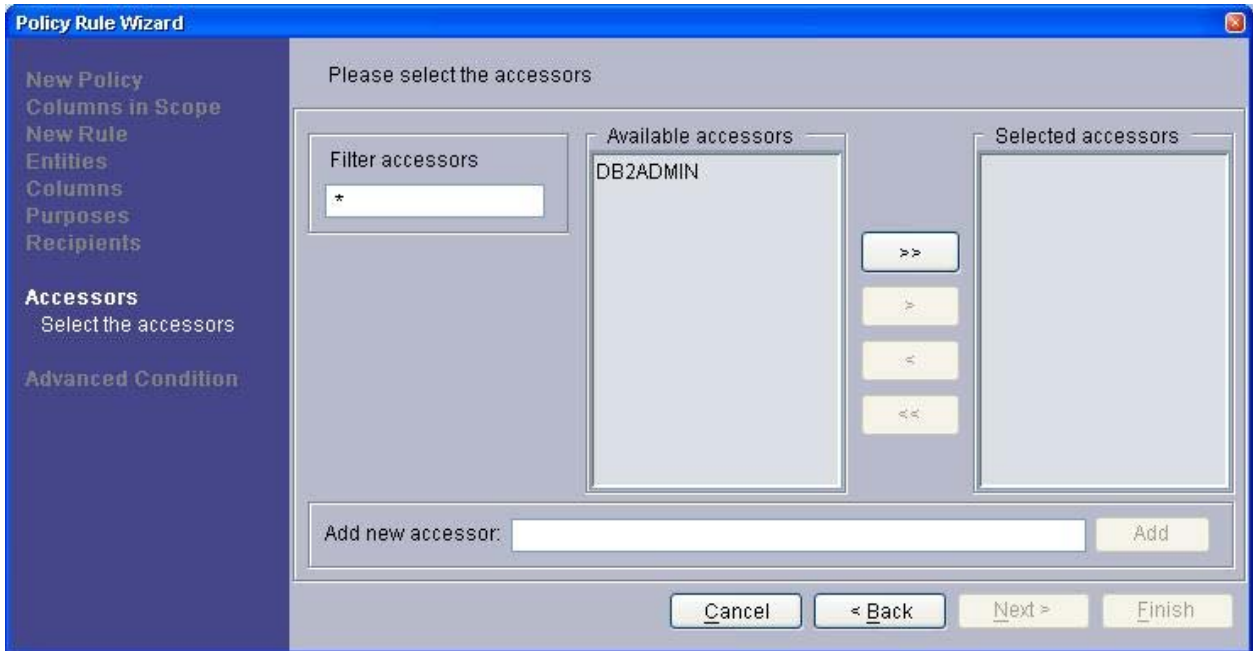


Figure 11. Accessor Selection Screen

13. **Advanced Condition** The user may now click “Finish” to close the Disclosure Rule Wizard and create the new policy rules. Alternatively, the user may append extra conditions to the disclosure rule (beyond the basic purposes / recipients / accessors combination). In the “Advanced Condition” screen (Figure 13), the user may define necessary boolean expressions based on database columns. The user is responsible for the validity of the resulting expressions, which are appended to the disclosure rule. As a rule of thumb, if conditions apply, all selected columns in the wizard should belong to the same table, and predicates are created only on columns of those tables. If other tables are required in a condition, subqueries will need to be formulated in order to reference these tables. Since subqueries are beyond the capabilities of the condition editor, the user will have to directly write and edit the subquery predicate into “compound condition” window. --- If complex versioning is in effect for the policy, the system will generate additional predicates that check that the entity subscribes to this policy and version. These extra predicates are not visible in the “compound condition” window shown in Figure 13. However, the complete policy rule condition will be visible in the “policy rule details” window shown in Figure 14. If, for example, a rule is being defined for a column belonging to a table related to the entity table by foreign-key relationship, but no foreign-keys have been explicitly specified on the table, a warning message will appear. No subscription condition will be generated for these columns and the user will have to explicitly add the entity subscription conditions by editing the rule.

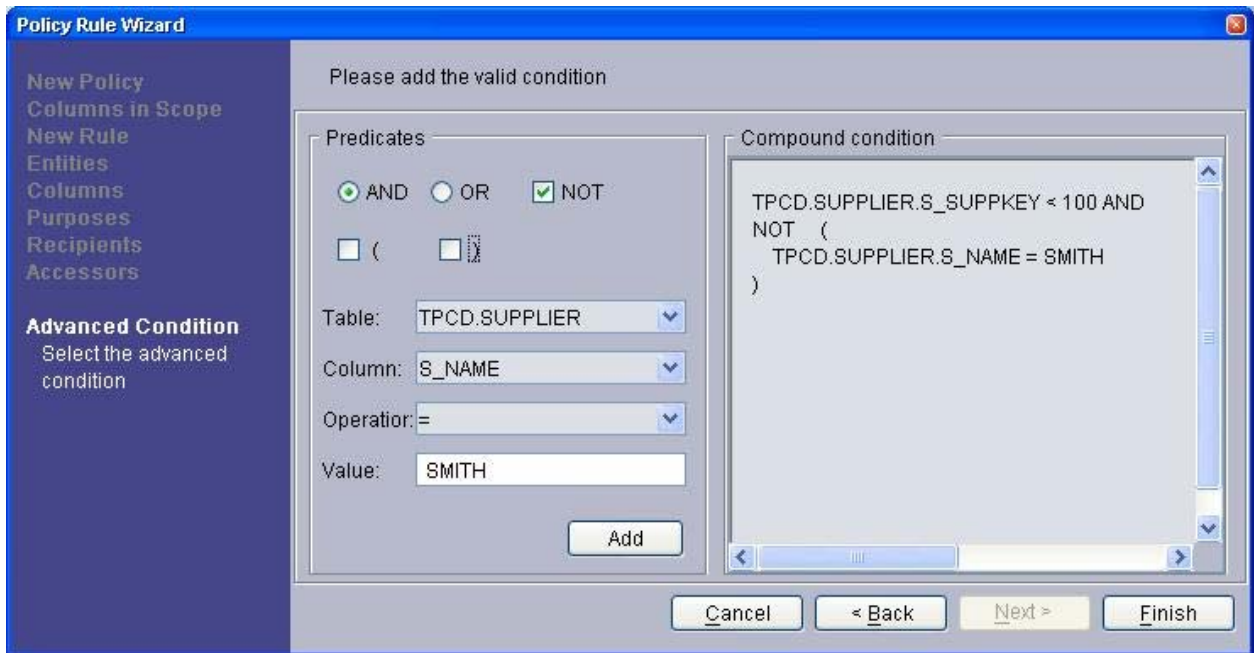


Figure 13. Advanced Conditions Screen

14. **Rule Details** After the disclosure rule is generated, the user may view the results on the right side of the main panel in the Control Center (Figure 14), and may edit or remove individual rules. Note that if a disclosure rule covers multiple columns / purposes / recipients / accessors, multiple rows are generated in the result panel. One row represents each column / purpose / recipient / accessor combination.

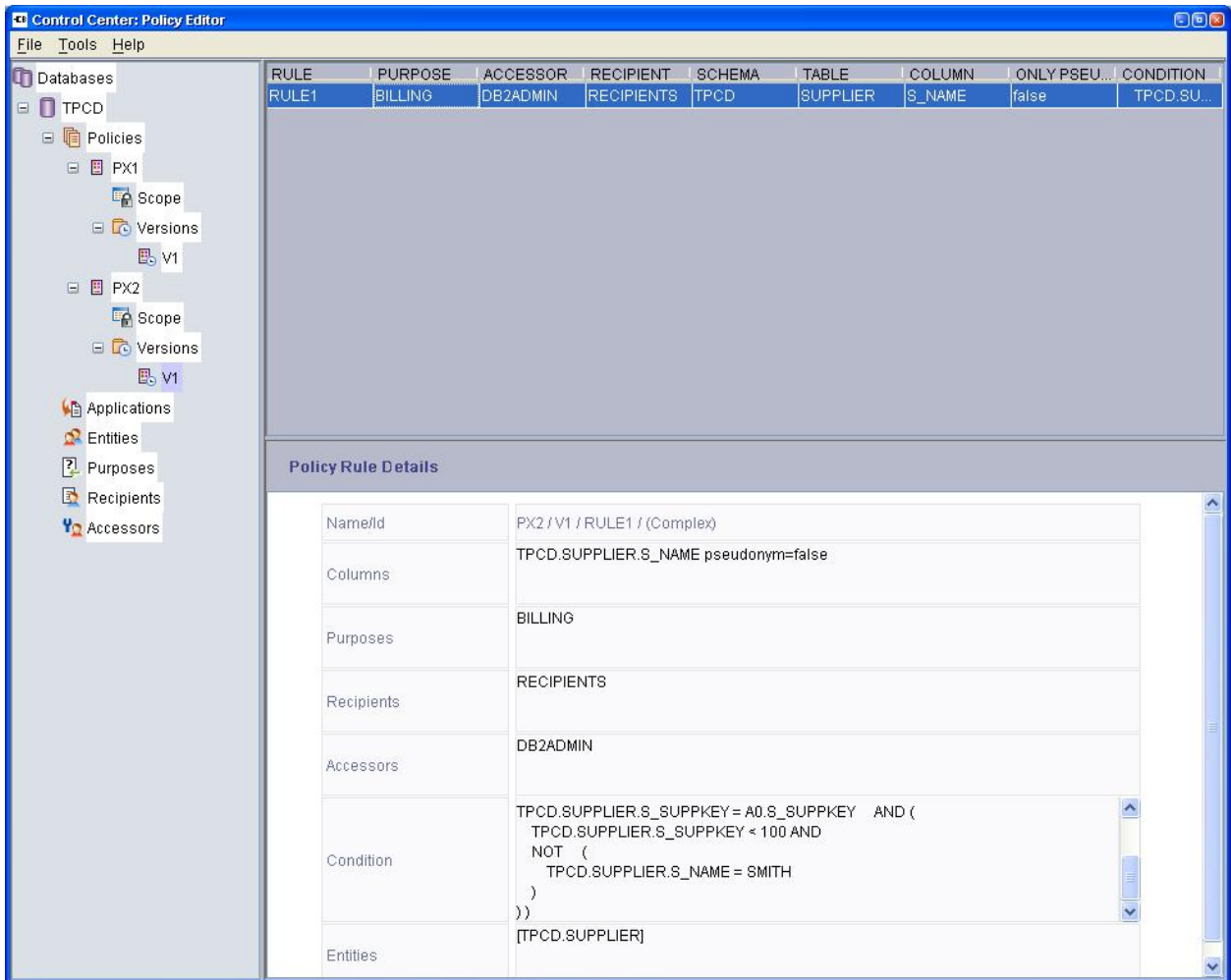


Figure 12. HDB Control Center Main Screen with Rule Display

15. Application Usage After policies and rules are defined, the user clicks on the database name node, right clicks and selects “Define Application Usage” (Figure 15). Application usage information is used to determine the purposes, and recipients for incoming queries (refer to Section 4.2). This information will be used when connecting to a database using the HDB JDBC driver in order to infer the purpose and recipient from the application and accessor. The HDB JDBC url extends the database name to identify the application name. For example, jdbc:hdb2:sample#APP1 is used to create a JDBC connection to database sample using HDB and application APP1. HDB will infer that queries on that connection will be for the purpose of BILLING and recipient RECIPIENTS.

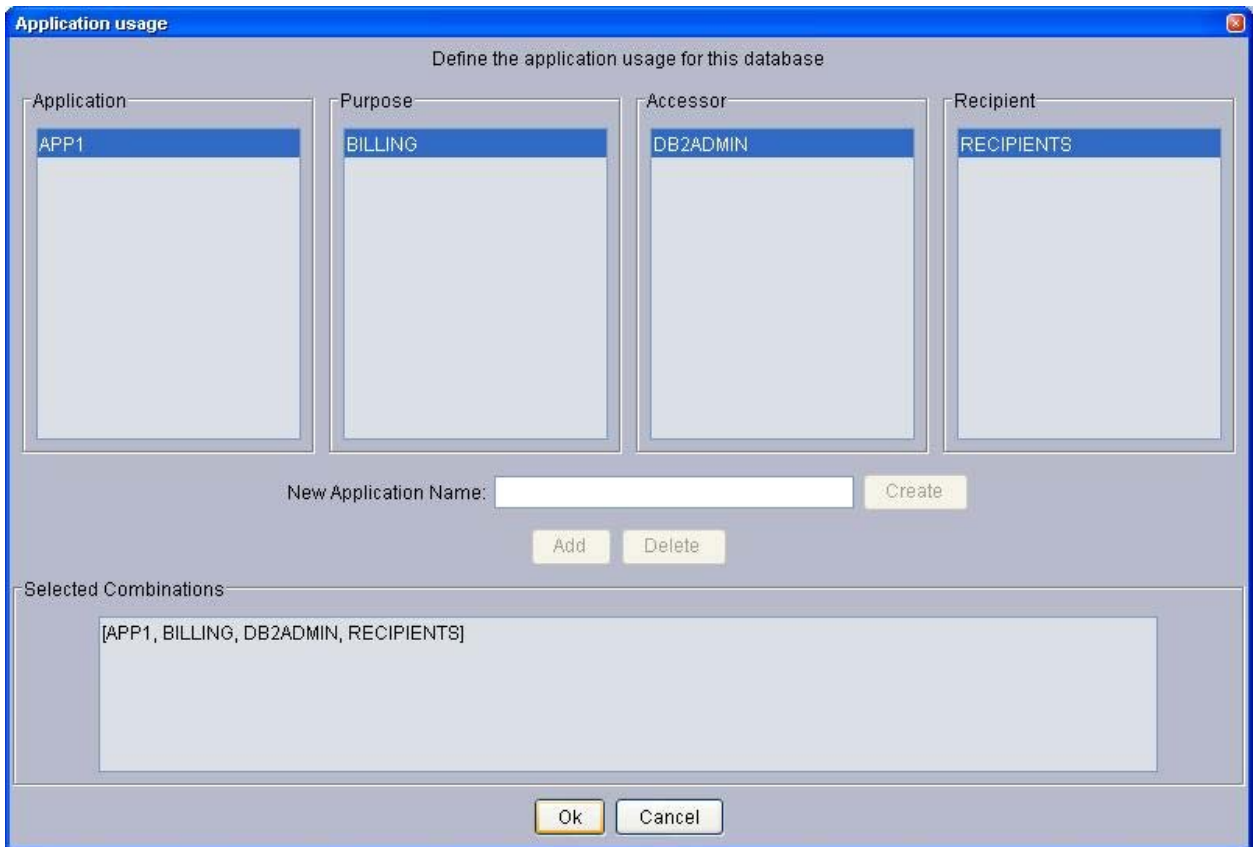


Figure 15. Application Usage Definition Screen

- 16. Import Rules** While defining new version user can import rules from the previous versions or later user can import rules for a version from another, "Import rules" wizard is implemented for the same.

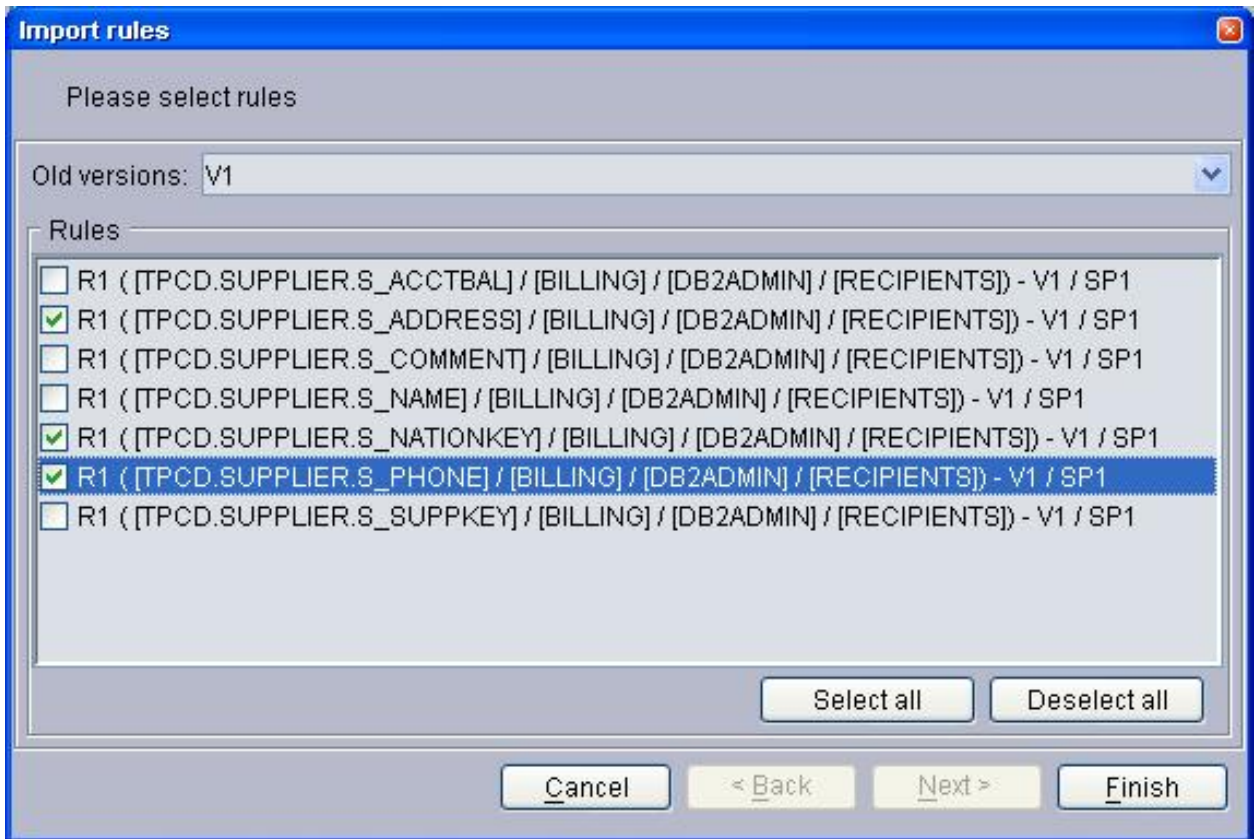


Figure 13. Import Rule Wizard Screen

17. **Simple Policy Versioning** The default policy versioning type is “Complex”. However, while creating policies, users can select versioning type as “Complex” or “Simple”. With simple policy versioning type, policies get applied without user subscriptions. Simple policies can have multiple versions out of which only one will be activate at a time

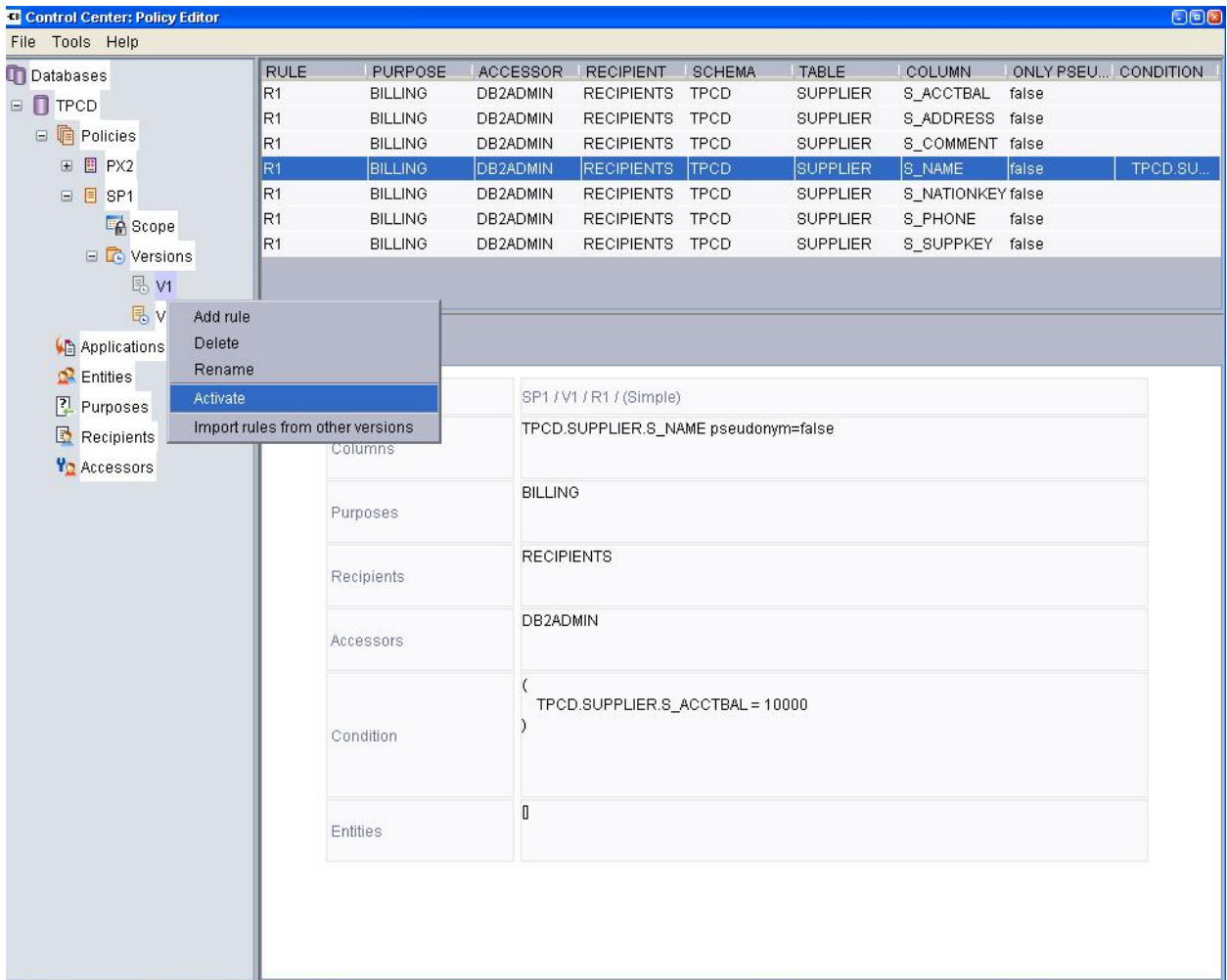


Figure 16. Activating simple policy versioning

4. HDB JDBC Driver Installation

Every database application that is to be enabled for active enforcement has to have installed an HDB driver that manages all interactions between the application and the database system(s). This section describes the actions required to use the drivers offered by the HDB system. The HDB JDBC drivers are for the *IBM DB2 UDB* and *Oracle* database systems, and for both target systems, JDBC type 2 and 4 drivers are implemented. However, only the IBM DB2 UDB type 2 driver is available at this time. Other drivers can nevertheless be made available if requested by customers. The HDB JDBC drivers are used for applications written in Java.

The drivers are a middleware layer that ensures that statements written in SQL (SELECT, INSERT, DELETE, and UPDATE statements) adhere to the data disclosure policies that have been defined for the database to which the application is connected. An HDB JDBC driver automatically rewrites queries and returns only data that is consistent with these policies. This rewriting is transparent to the application, only the appropriate HDB JDBC driver has to be installed and be used to create the database connections.

4.1 Packages Required for Compilation and Program Execution

When a JDBC application is compiled using a Java compiler, the classpath has to include the package file `hdb.zip` provided by the HDB system to be able to use the appropriate HDB driver. An example compilation and execution in a Windows environment might look like this:

```
set HDB="c:\program files\hdb\java"  
javac -classpath %HDB%\hdbClasses.zip HelloWorld.java  
java -classpath %HDB%\hdbClasses.zip HelloWorld
```

4.2 Database Application Code Impact

In the following subsections, we explain the mechanisms for passing contextual information from Java applications that use a JDBC driver to the database.

4.2.1 HDB JDBC Driver for DB2

The HDB JDBC driver for DB2 can be used similar to any other JDBC driver provided with IBM DB2 UDB. A Java application that uses a type 2 JDBC for DB2 contains statements like the following:

```
// Load the JDBC driver.  
Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");  
  
// Establish connection to the database "our_db".  
Connection c = DriverManager.getConnection("jdbc:db2:our_db", "admin",  
"password123");
```

To enable active enforcement, one simply has to replace the above code by the following statements:

```
// Load the JDBC driver.
Class.forName("com.ibm.db2.jdbc.app.HdbDriver");

// Establish connection to the database "our_db".
Connection c =
DriverManager.getConnection("jdbc:hdb2:our_db#accounting#pseudonympass123",
"admin", "password123");
```

In the above statement, “accounting” is used as the application identifier to retrieve the privacy semantics from the database, and pseudonympass123 will be used as the encryption key to generate pseudonyms.

In general, the following drivers and URLs have to be used by the JDBC applications:

- DB2 type 2 driver:
 - Class: com.ibm.hdb.jdbc.app.db2.HdbDriver
 - URL: jdbc:hdb2:<database>#<application>#
- DB2 type 4 driver:
 - Class: com.ibm.hdb.jdbc.jcc.db2.HdbDriver
 - URL: jdbc:hdb2://<host>:<port>/<database>#<application>#

4.2.2 HDB JDBC Driver for Oracle

The HDB JDBC driver to be used for applications connecting to an Oracle database can be used in the same way as a JDBC driver provided by Oracle. For a Java application that uses a type 2 or type 4 Oracle JDBC driver, do the following:

- Oracle type 2 driver:
 - Class: com.ibm.hdb.jdbc.app.oracle.HdbDriver
 - URL: jdbc:oracle:oci8:@<databaseService>#<application>#
- Oracle type 4 driver:
 - Class: com.ibm.hdb.jdbc.jcc.oracle.HdbDriver
 - URL: jdbc:oracle:thin:@//<host>:<port>#<application>#

4.2.3 HDB JDBC Driver for J2EE Connection Pools for DB2 and Oracle

When a J2EE application uses a database connection pool, the following HDB drivers have to be used instead:

- DB2: com.ibm.hdb.jdbc.jcc.db2.HdbConnectionPoolDataSource
- Oracle: com.ibm.hdb.jdbc.thin.oracle.HdbConnectionPoolDataSource

4.3 Limitations

HDB drivers have the following limitations:

1. When a prepared statement is executed using `PreparedStatement.execute()` and the query is of the type “SELECT *” and the schema of the table is changed during the program execution then the schema changes may not be reflected in consecutive executions of the prepared statement.

2. Function LENGTH(), which computes the length of a text string with leading and trailing whitespaces, is not supported. An exception is thrown for statements using this function.
3. Query audit log generation is only supported with the HDB Type 2 JDBC driver for DB2.

Appendix A. Query Enforcement Details

To enforce these data disclosure policies, Active Enforcement defines an HJDBC driver that encapsulates the SQL query and rewrites the query with privacy semantics. The HJDBC driver is wrapped around existing JDBC driver functions, which incorporate disclosure enforcement code into these wrapper functions. The solution is database and application agnostic because the enforcement code resides in the JDBC layer. Accordingly, HDB cell-level enforcement is possible for any company with JDBC-compatible applications and a data source that uses SQL. Another SQL interface, such as ODBC, can be used in place of JDBC. Figure 1 shows graphical representation of query execution flow.

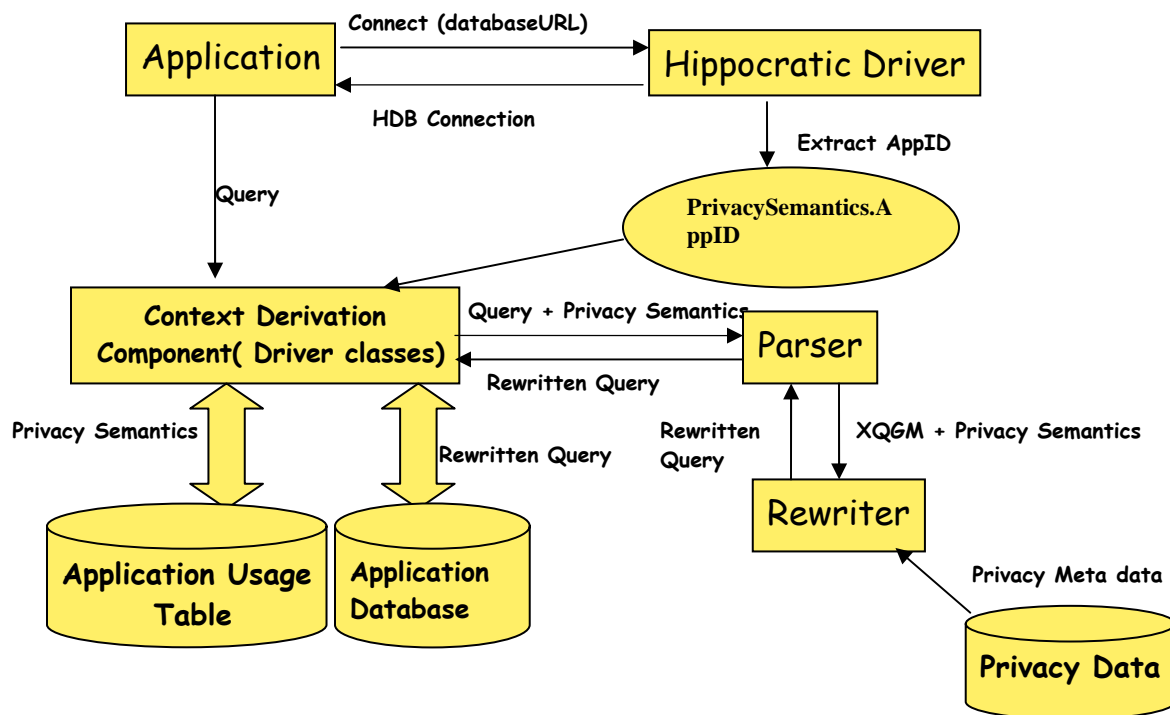


Figure 18. HDB Query Execution Flow

Whenever an application wants to execute any SQL query, it should have a connection object to the underlying database. To connect to the database, the application issues a connect request to the drivers. This request contains the connection parameters (database url) which also includes the application name. When any query is sent to the HJDBC drivers for execution, the context derivation component extracts the privacy semantics from the Application Usage table, tags the

query with these semantics, and sends the query to the parser. The Query is then parsed into HDBQGM (a derivative of the DB2 QGM), which provides graphic representations of SQL queries. The HDBQGM form of the query and privacy semantics are then passed to rewriter, which modifies the query to include the privacy semantics.

Appendix B. SQL Support provided by HDB Driver

HDB Driver uses the Derby parser for parsing the SQL statements for privacy enforcement. The level of SQL supported by the HDB Driver is SQL-92 features supported by Derby with some deviations. The Derby support for SQL features can be found at the following link:

<http://incubator.apache.org/derby/docs/10.0/manuals/reference/sqlj151.html#HDRSII-SQL92-41891>

The list of deviations from this specification is:

1. Alter table drop column - not supported in DB2
2. Predicates: OVERLAPS – not supported in DB2
3. Explicit defaults – Assigning a column default value in INSERT or UPDATE statement is not supported in HDB.
4. Constraint Tables – HDB Driver over DB2 supports the DB2 system catalogs and not derby catalogs. The DB2 catalog can be found at

<http://publib.boulder.ibm.com/infocenter/db2help/index.jsp?topic=/com.ibm.db2.udb.doc/admin/c0004932.htm>

5. Character set definition – DB2 character set (codepages) definition supported. More information can be found at:

<http://publib.boulder.ibm.com/infocenter/db2help/index.jsp?topic=/com.ibm.db2.udb.doc/admin/r0000284.htm>

6. Temporary tables – Not supported by HDB. The Hippocratic driver recognizes only stored tables and views and not temporary tables.
7. Collation – The collating sequences valid for HDB are the DB2 Collating sequences. More information can be found at:

<http://publib.boulder.ibm.com/infocenter/db2help/index.jsp?topic=/com.ibm.db2.udb.doc/ad/c0006812.htm>

Appendix C. Description of HDBADMIN Database Tables

Below is a description of the HDB admin tables included in the distribution as created by the HDB Control Center:

"HDBADMIN"."SCOPE" ("POLICYID" VARCHAR(32) NOT NULL , "VERSION" VARCHAR(32) NOT NULL , "SCHEMA" VARCHAR(32) NOT NULL , "TABLE" VARCHAR(32) NOT NULL , "COLUMN" VARCHAR(32) NOT NULL)

- This table stores the database columns that are governed by HDB active enforcement (AE). If a column name is mentioned in the scope table, then its disclosure is subjected to the policy rules (see below) of policies identified by POLICYID. If VERSION is blank, the complex versioning is in effect for this policy. Otherwise, if a version is specified, the simple versioning is in effect for this policy and the activated policy version is identified by the column. By default, if a column name is included in the scope table but is not mentioned in any policy rules, then no disclosure is allowed under any circumstances. On the contrary, for those columns that are not mentioned in the scope table, their disclosures are not governed by HDB AE.

"HDBADMIN"."POLICY" ("POLICYID" VARCHAR(32) NOT NULL , "RULEID" VARCHAR(32) NOT NULL , "VERSION" VARCHAR(32) NOT NULL , "PURPOSE" VARCHAR(32) NOT NULL , "ACCESSOR" VARCHAR(32) NOT NULL , "RECIPIENT" VARCHAR(32) NOT NULL , "SCHEMA" VARCHAR(32) NOT NULL , "TABLE" VARCHAR(32) NOT NULL , "COLUMN" VARCHAR(32) NOT NULL , "CONDITION" VARCHAR(1024) NOT NULL , "PSEUDONYM" INTEGER NOT NULL)

- This is the table where policy rules are stored. Each policy has an ID (policyid) and version number. Each policy contains a number of rules, each one has another ID (ruleid). A rule is a description of the condition(s) under which column data will be disclosed. For a given rule, the (schema, table, column) fields specify the column that the rule applies to. (purpose, accessor, recipient) refers to the purpose of the query, the initiator of the query, and the recipient of query results respectively to which the specified data column will be disclosed and condition is any extra boolean conditions that can be appended to a rule (e.g. the age attribute of patient must be > 18 before data from the specified column will be disclosed). If pseudonym is 0, column values are returned as plain text, if pseudonym is 1, column values are returned as ciphertext represented using a base64 string of digits.

"HDBADMIN"."APPLICATION_USAGE" ("APPID" VARCHAR(32) NOT NULL , "PURPOSE" VARCHAR(32) NOT NULL , "ACCESSOR" VARCHAR(32) NOT NULL , "RECIPIENT" VARCHAR(32) NOT NULL)

- In the current implementation of HDB AE, the user does not specify (purpose, recipient) themselves in queries. Thus, we identify those values through application ID, i.e. the application name through which the connection was issued from, and this table stores the mapping from appid, accessor to the other values.

As an example, here are two policy rules and their verbal descriptions (both are from create.sql):

```
insert into hdbadmin.POLICY values('1', '1', '1', 'BILLING', 'BOB', 'OURS', SAMPLE, 'PATIENTINFO',
'NAME', 'exists(select * from sample.patientinfo where patientid=1)')
```

- This is rule 1 in version 1 of policy 1. It says that for the purpose is billing, if Bob is issuing a query that requests data from SAMPLE.PATIENTINFO.NAME for recipient "ours," then data from the NAME column will be disclosed only if the patient has patientid 1, i.e. only patient 1's, but no others, NAME will be disclosed.

```
insert into hdbadmin.POLICY values('1', '13', '1', 'PSYCHIATRIC', 'PAUL', 'OURS', 'SAMPLE',
'TESTRESULTS', 'CODE', 'exists (select 1 from sample.patientinfo_association a where
sample.testresults.patientid = a.patientid and a.policyid = "1" and a.version = "1")');
```

- This is rule 13 in version 1 of policy 1. It says that for purpose of psychiatric, if Paul is issuing a query that requests data from SAMPLE.TESTRESULTS.CODE for recipient "ours," then data from the CODE column will be disclosed only if the patient has subscribed to version 1 of policy 1 (this is accomplished by the join in the condition field of the testresults and association tables).