



IBM Hippocratic Database Auditing

User Guide

Version 1.0

Introduction

The *IBM Hippocratic Database Auditing (HDBA)* system is an application used to track past disclosure from a database. An auditor can specify an audit expression (using SQL plus contextual information) whose result is the set of logged queries that accessed data tracked by the audit expression.

HDBA complements *IBM Hippocratic Database Active Enforcement*, a software system that operates as a middleware layer, ensuring that enterprise applications accessing a database adhere to fine-grained data disclosure policies.

HDBA requires data and metadata to perform audits:

Query log table: A table containing all the SQL queries of committed transactions that have been successfully executed by the database system.

Backlog tables: For each table in the database, the administrator can decide whether it is enabled for auditing. Whenever a table is enabled for auditing, a corresponding “backlog” table is created that stores all data updates (insert, deletes, updates) performed on the original table. Otherwise, disclosure from the table cannot be audited.

Metadata tables: Tables that store, for example, the audit queries that have been defined by the auditor using the HDBA audit query wizard.

An Audit Scenario

To understand the usage of HDBA, consider a fictional financial institution *FFI* that resulted from a merger between a fictional bank *FBank* and a fictional mortgage company *FMortgage*.

Claire Young is a 31-year old professional in Santa Monica, California who would like to open a platinum credit card account with FBank. She logs onto FBank’s website and completes an application, divulging a significant amount of personal information, including her address, telephone number, social security number, current employment specifics, and income history. Claire is provided with an online copy of FBank’s privacy policy and is allowed to specify her preferences regarding the disclosure of her personal information. She opts out of any disclosures of her financial information to companies unaffiliated with FBank, and requests that she not be contacted by telephone or mail with any product or service offers. She does not opt out of email contact from FBank.

Brian Jones is a mortgage broker with FFI’s Los Angeles office looking to develop additional home refinancing business. After the merger and database integration, Brian decides to search the mortgage database for leads on high-income bank credit card customers who might be interested in refinancing. Brian executes the following database query to request the contact information for all bank customers with a Los Angeles County area code and an annual income over \$100,000.

```
SELECT name, address, phone, email, ssn, income
FROM customers
WHERE county = 'LA' AND income > 100000
```

Because Claire opted out of disclosing her address, telephone number, and social security number, only her name and email address may be revealed to Brian.

Several months after Brian’s query, Claire receives a series of emails from FFI offering to refinance her existing home mortgage. Around the same time, Claire receives many telephone and mail solicitations from a variety of financial service companies not affiliated with

FMortgage. Claire suspects that FBank has sold her information to outside marketing companies and threatens to initiate legal action against FBank for these alleged privacy breaches. The company is unaware of any such disclosures between FBank and FMortgage, but is concerned about its exposure to liability.

Using the HDBA system, FFI's Chief Privacy Officer, requests an audit of all access to Claire's records since the date of her FBank credit card application on 12/17/2002. HDBA returns an audit trail of all queries that accessed Claire's records within the specified date range, including user, date, recipient, purpose, and actual information disclosed.

```
DURING '2002-12-17 00:00:00.0' AND CURRENT DATE
AUDIT *
FROM ffidb.customers
WHERE customer = 'Claire Young'
```

The results are displayed as follows:

User	Date	Customer	Purpose	Recipient	Information
Doug Allen	6-10-03	Claire Young	Cust. Service	None	SSN, telephone ...
.....
Eric Williams	8-03-03	Claire Young	A/R	None	balance, address ...

The auditor then narrows the request to reveal only the results of those queries that requested access to Claire's phone number, mailing address, or email, including aggregate queries.

```
DURING '2002-12-17 00:00:00.0' AND CURRENT DATE
AUDIT phone, address, email
FROM ffidb.customers
WHERE customer = 'Claire Young'
```

HDBA provides the following, more specific results.

User	Date	Customer	Phone	Address	Email
Brian Jones	09-10-03	Claire Young	-	-	cy@sbc.net
.....
Betty Watson	06-13-03	Claire Young	310-555-5483	38 Montana, SM	cy@sbc.net
Betty Watson	07-12-03	Claire Young	310-555-5483	38 Montana, SM	cy@sbc.net
Betty Watson	08-17-03	Claire Young	310-555-5483	38 Montana, SM	cy@sbc.net

The auditor determines that Brian's query accessed her email address and may have been the source of the email solicitations from FMortgage. However, the audit trail confirms that Brian's query did not return Claire's telephone number or mailing address. The audit trail also reveals several suspicious queries of Claire's record by Betty, an FBank account manager. It shows that Betty queried the database three times and accessed Claire's transaction history and contact information. However, the telephone number and mailing address revealed to Betty were for Claire's former residence. Since the time of Betty's last query, Claire has changed her contact information in the FBank database. Therefore, it appears that none of the FFI companies is responsible for an unauthorized disclosure of Claire's address and telephone number.

1. Working with HDBA

1.1 Installation

HDBA is an application that is accessible from the *IBM Hippocratic Database Control Center*, which also gives access to the policy editor for *IBM Hippocratic Database Active Enforcement*, which is described in a separate user guide [1]. The installation of HDBA happens during the installation of IBM Hippocratic Database Active Enforcement, which is described in [1].

1.2 System Requirements

Since HDBA is installed with IBM Hippocratic Database Control Center, HDBA runs on the Microsoft Windows operating system.

HDBA is a Java application that requires the Java runtime environment (JRE) version 1.4.2 or higher.

HDBA requires IBM DB2 version 8 or higher. The database system can run on any of the platforms Linux, UNIX, or Windows (LUW). When an auditor specifies and executes audit queries, HDBA automatically generates SQL92-compliant query expressions and calls the DB2 system.

It is further assumed that the query log created by the HDB Active Enforcement is available in each database that is audited. This log is created as part of the metadata installation of HDB Active Enforcement.

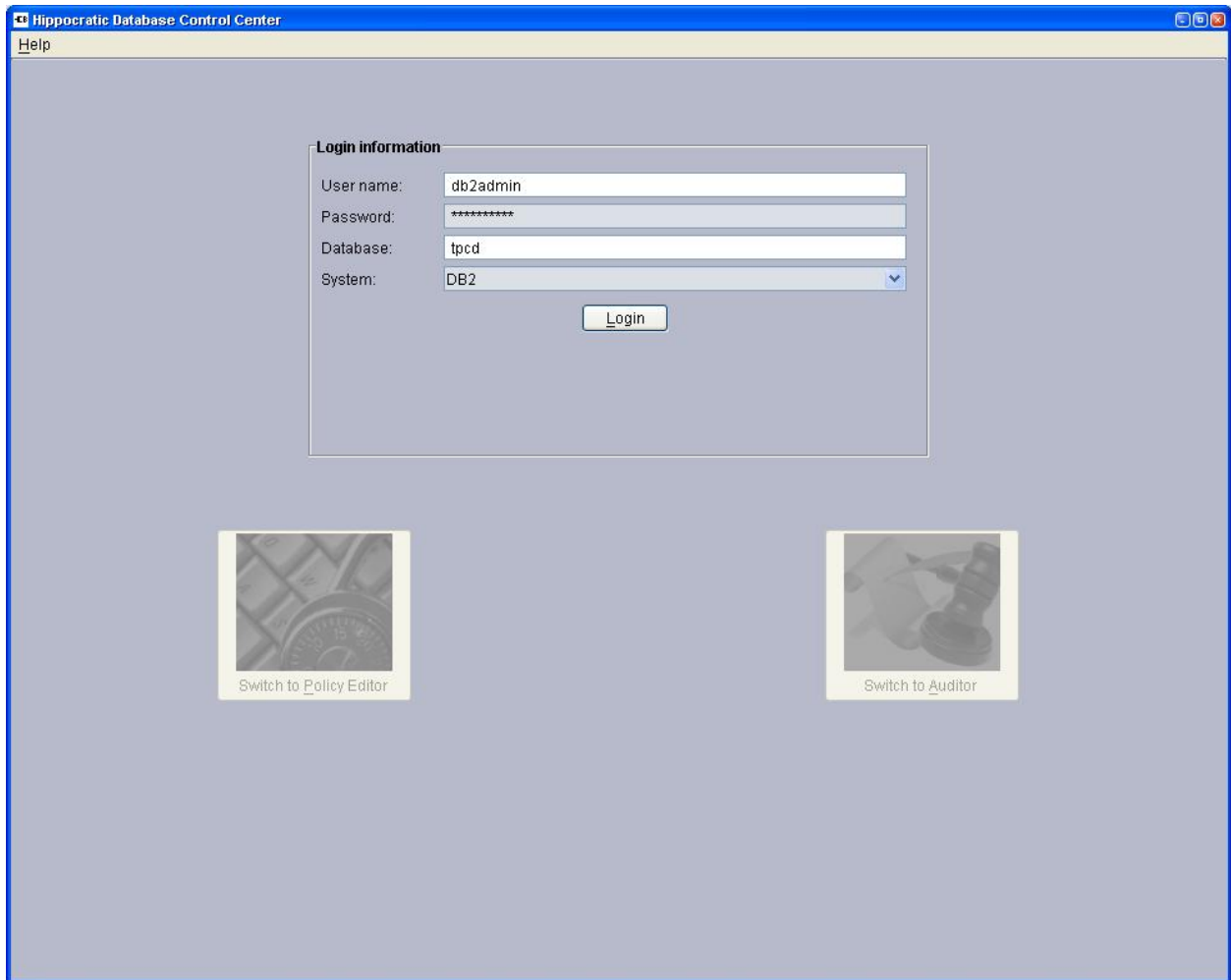
1.3 Starting the Program

HDBA is integrated into the Hippocratic Database ControlCenter, a Java application that can be started in a Windows console with the command

```
C:\hdb> java com.ibm.hdb.controlCenter.ControlCenter
```

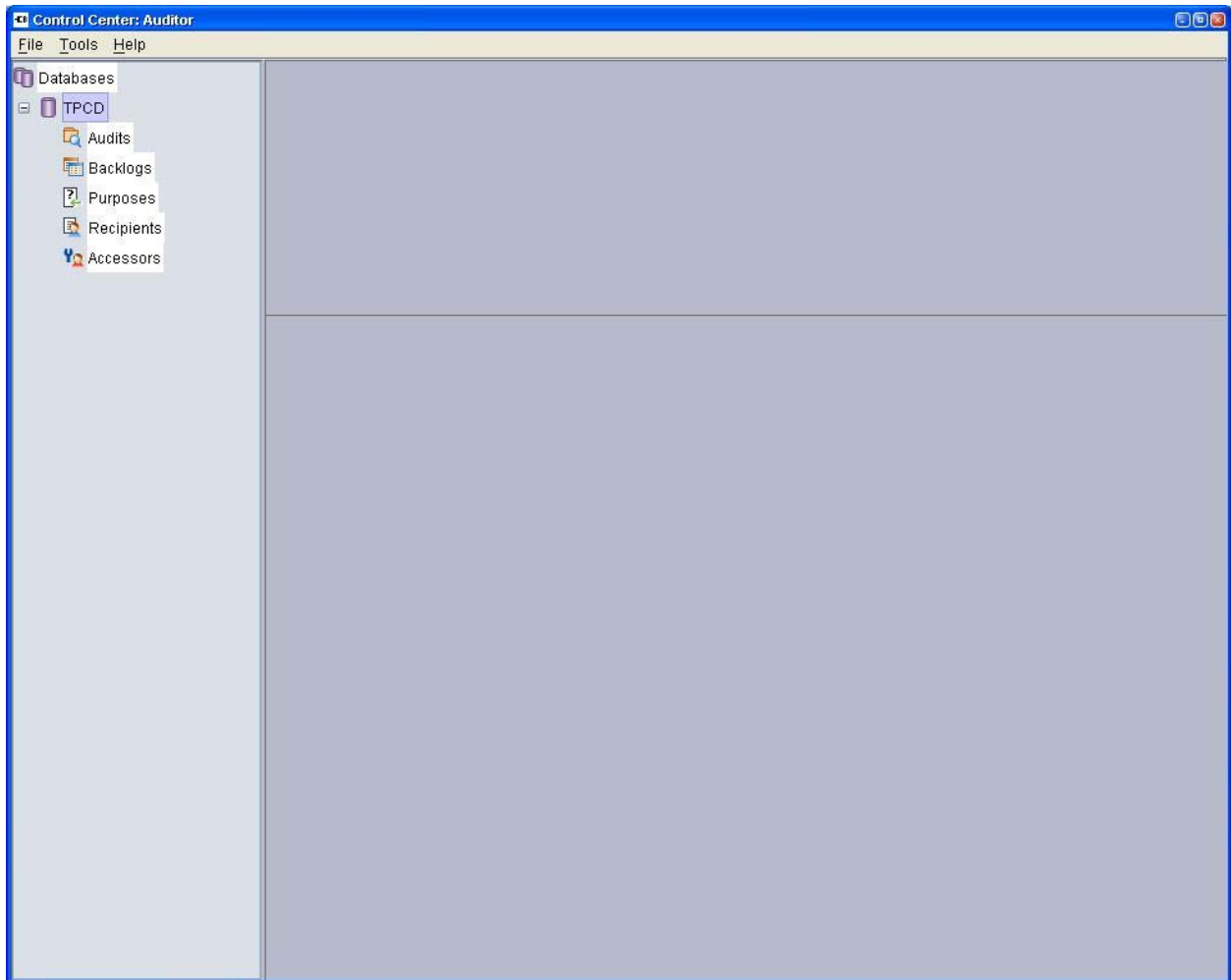
1.4 Login

A Login panel appears that asks for the user credentials, the database to which a connection is to be established, and the type of database system that hosts the database. Currently, the only database system supported is DB2.



When the user has logged in successfully, two buttons are activated: one for the policy editor, the other for HDBA.

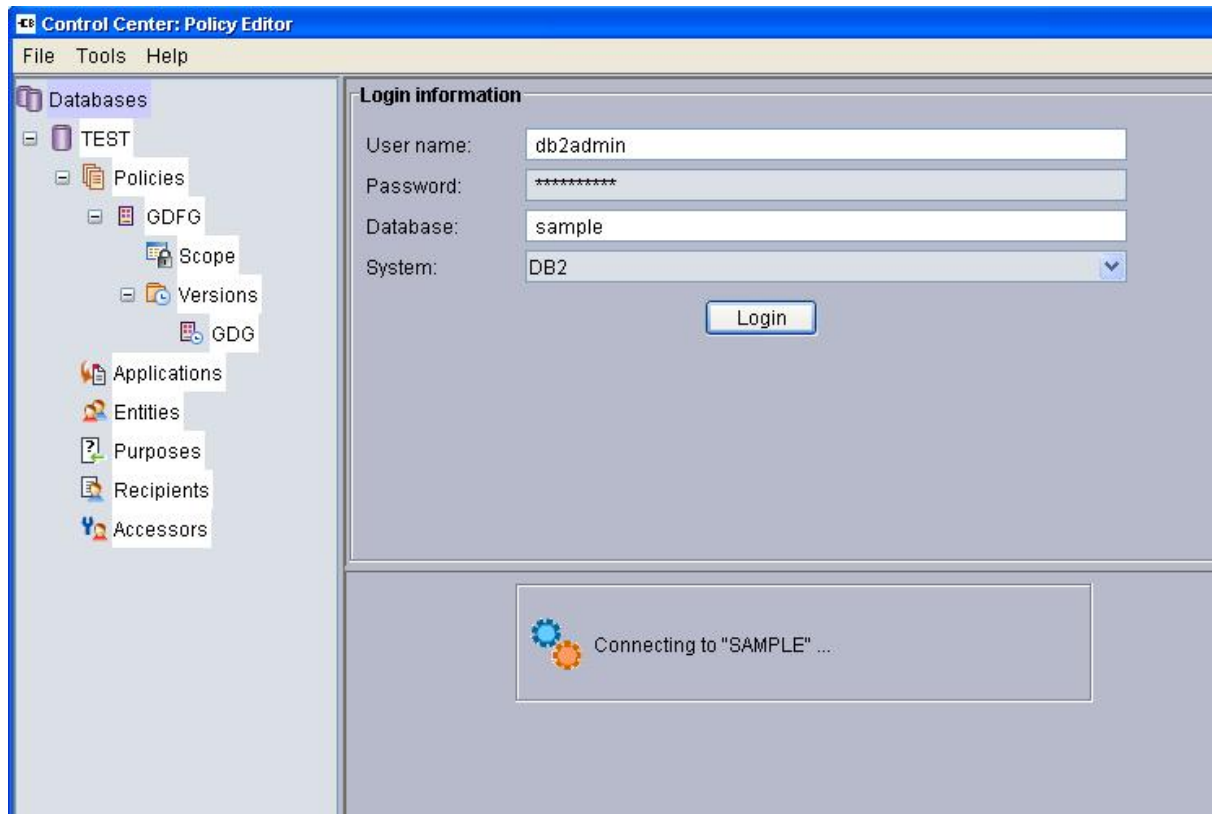
After clicking on the button “Switch to Auditor,” the window shows two parts: the audit browse tree on the left and an initially empty panel on the right.



The audit tree remains in the left panel. The content in the right panel changes according to the actions performed on the tree. The root of the tree is the node “Databases,” under which the list of databases is shown to which a connection is currently established.

1.5 Connect to Database

To connect to another database (or to reconnect to a database after disconnecting), the user clicks on the node “Database”, right clicks and selects “Connect to another database.” A login dialog is displayed in the right panel.



1.6 Disconnect from Database

After selecting the database to disconnect from at Database→<database>, the user can right click and select “Disconnect from database.”

1.7 Expand or Collapse Audit Browse Tree

If the user wants to see the fully expanded audit tree, user selects node “Database,” right clicks and selects “Expand tree.” The entire tree can be contracted to the root node by selecting “Collapse tree” Option.

1.8 Metadata Installation

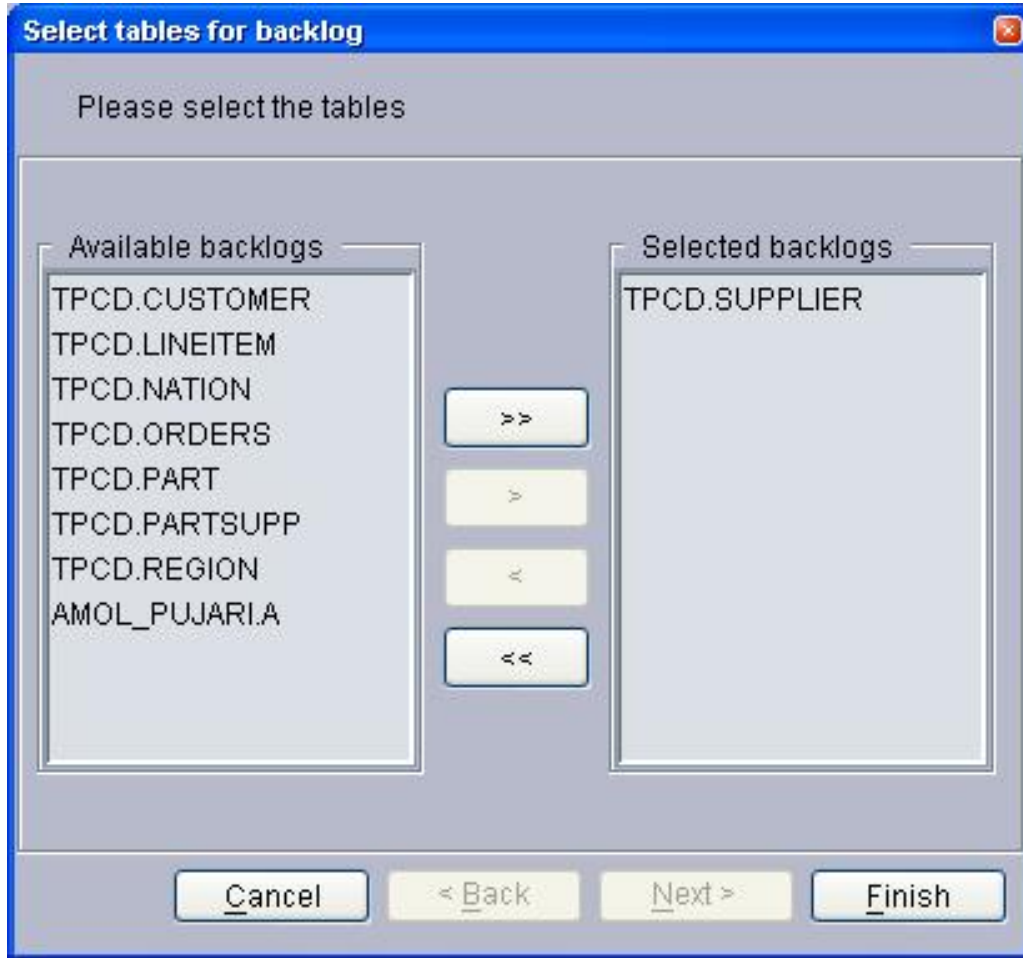
Auditing requires the installation of tables specifically for the audit system. After selecting the database, the administrator can right click and select to “Install audit metadata.” The Control Center will generate and execute appropriate SQL scripts to create the metadata tables.

In the same way, audit metadata can be removed. The reason why these menu items are presented to the administrator (instead of installing the metadata transparently during system installation) is that we want to allow the user to dispose all database objects required for auditing easily. The database will then appear as if auditing was never enabled.

1.9 Backlog Table Installation

The administrator can select the specific tables to be made auditable. The administrator may decide to enable all tables of a database or only some of them. The latter may be appropriate if

some of the tables will never contain sensitive information or if there is simply no need to audit the disclosure of data from these tables.



For each table that is enabled for auditing, a so-called *backlog table* is created, which captures all data modifications (insert, update, delete) to the table. For any schema change to an auditable table, a new backlog table will need to be created for the original table. The details about backlog tables are not described in this document and are not important to the user.

1.10 Select Backlog Tables

After selecting the database, the administrator can right click and select “Define backlog tables.” A menu appears showing the tables with and without backlog tables. Tables with a backlog table are enabled for auditing. Note that when a table is disabled for auditing (moved from the right to the left list in the menu), and the database changes are committed, it is not possible to audit disclosures from these tables anymore. All information about data updates to the table is removed and cannot be recovered.

1.11 Add Example Data

Once the user creates policy metadata tables and defines the backlog tables, she can add the example data. This will insert example or dummy data into the Query log table as well as into the Purposes, Accessors, and Recipients tables that were created by installing policy metadata.

1.12 Create an Audit

An audit is a named collection of audit queries. An auditor may want to organize the audit queries into different audits. Select <database>→Audits, right click and select “Add audit.” The user is asked to enter a unique name. The audit then appears as a new node under “Audits.”

1.13 Delete all Audits

All audits can be removed as follows: Select <database>→Audits, right click and select “Delete all audits.” Audit Queries

An audit query is a specification of the disclosures from logged queries that the auditor is interested in. The components of an audit query are explained in Section 2.

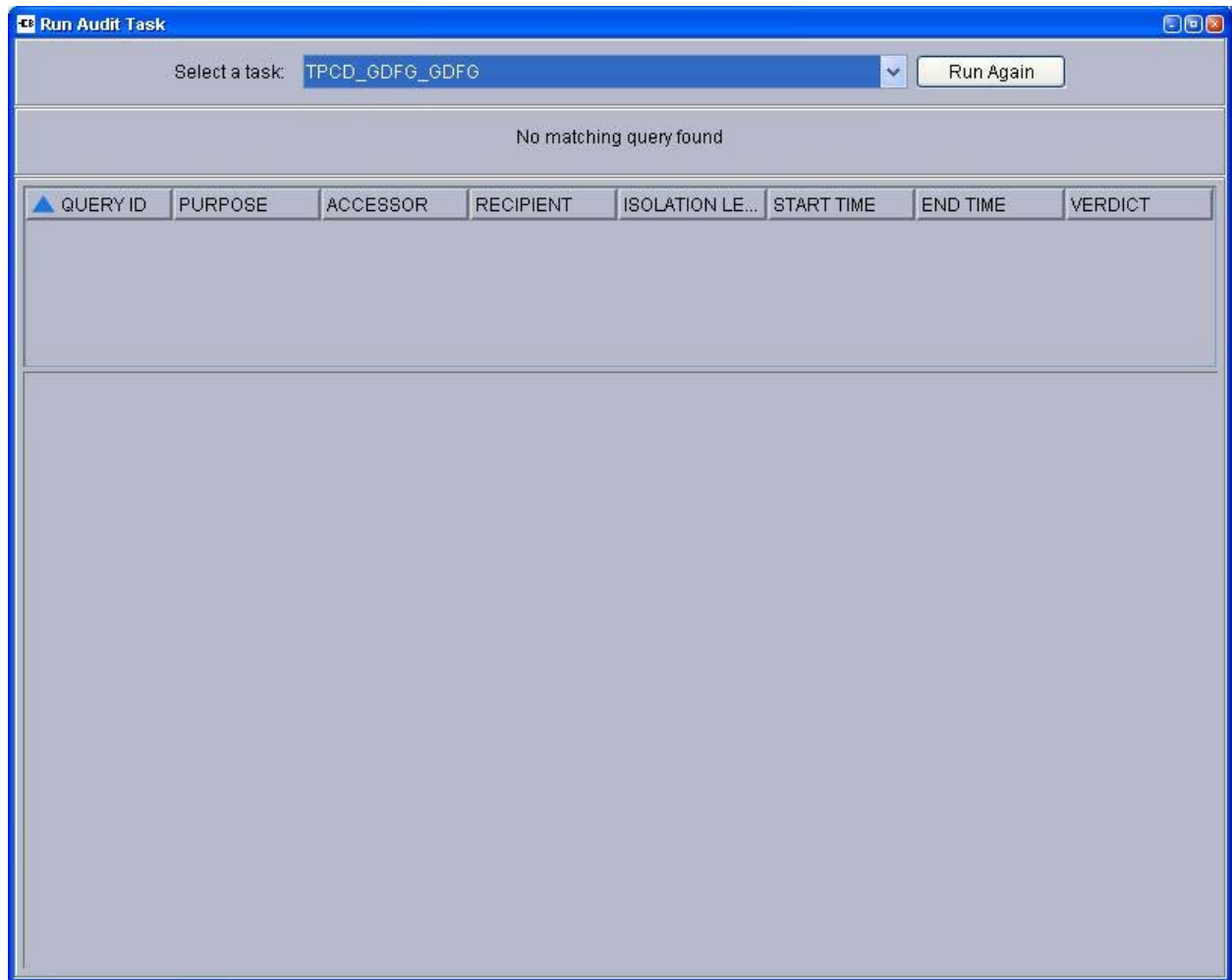
1.14 Create an Audit Query

To create a new audit query, select <database>→Audits→<audit>, right click and select “Add audit query.” The already existing and the newly created audit query are displayed in the table panel on the right.

1.15 Run an Audit Query

An audit query displayed in the audit query table can be executed as follows: Select the audit query in the table, right click, and select “Run.”

The result of the audit will show all the suspicious as well as candidate queries along with the query id, start and end date with the time. It will also show purpose, recipient and accessor (user) for each query. The result is shown using Run Audit Wizard, that also displays the status of the running task. If a logged query accesses tables that do not have a backlog table, a warning message will appear in the status window indicating a logged query parsing error. The error log can then be viewed to isolate these and other errors with logged queries.



1.16 Display Audit Query Details

An audit query displayed in the audit query table can be inspected follows: Select the audit query in the table, right click, and select “Show details.” A window appears that provides all information stored about the audit query. The user may see the SQL query generated by the audit system for this audit specification by checking the box “Show SQL.”

1.17 Edit an Audit Query

An audit query, which is displayed in the table panel on the right after the user selects <database>→Audits→<audit>, can be changed. Every detail, including the name of the audit query can be modified using the audit query wizard as follows: Select the audit query in the table, right click and select “Edit.” The audit query wizard is described in Section 2.

1.18 Delete an Audit Query

An audit query displayed in the audit query table can be deleted as follows: Select the audit query in the table, right click, and select “Delete.”

2. Audit Query Definition Wizard

An audit query is specified by

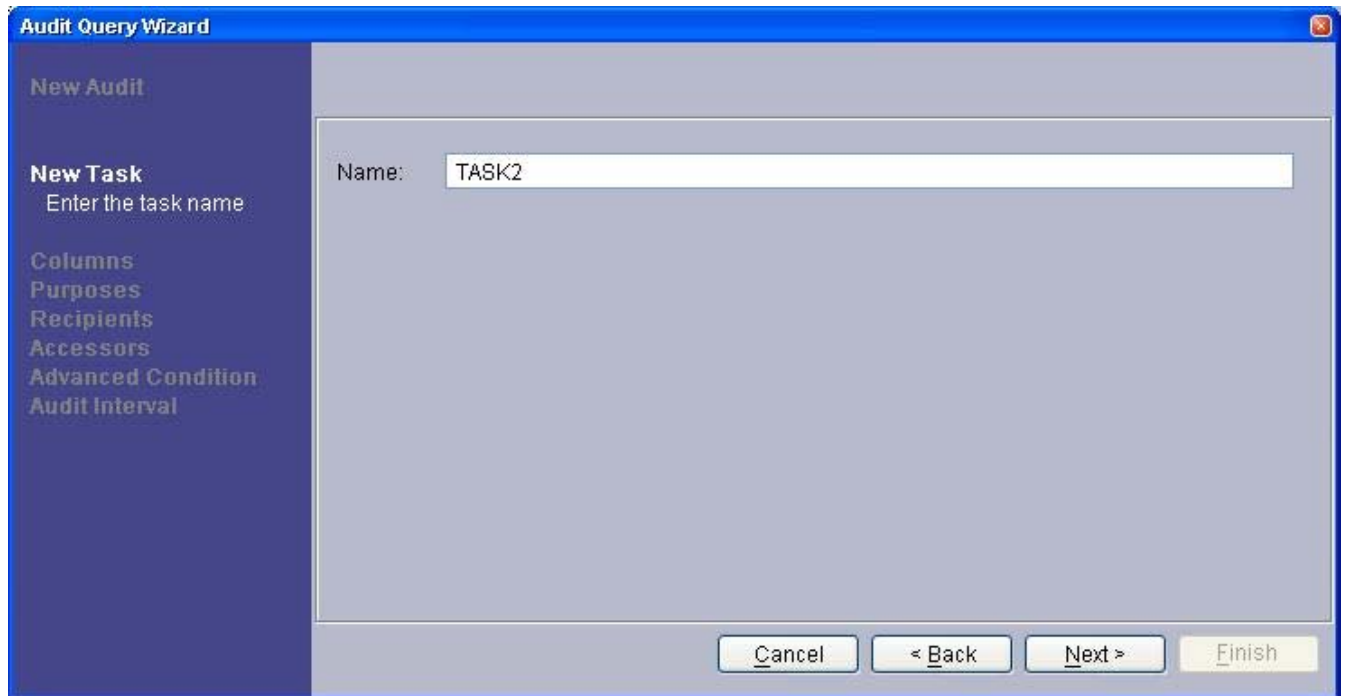
- a name
- a set of columns that have to appear in the SELECT clause or WHERE clause of the user query,
- a set of purposes,
- a set of accessors of queries,
- a set of recipients of query results,
- a condition that specifies the data whose disclosure the auditor is tracking, and
- a time interval.

An audit query is a SQL query together with context information. Since the design of an audit query depends on many pieces of information, a wizard leads the user through the query definition.

The audit query definition wizard can be started from the audit tree by selecting <database>→Audits→<audit>, right click, select “Add audit query” or in the audit query table by selecting an audit query, right click and select “Edit.” The wizard guides the user through the query specification process. Its result is an audit query object displayed in the audit query table panel. The following sections describe each step of the wizard in detail.

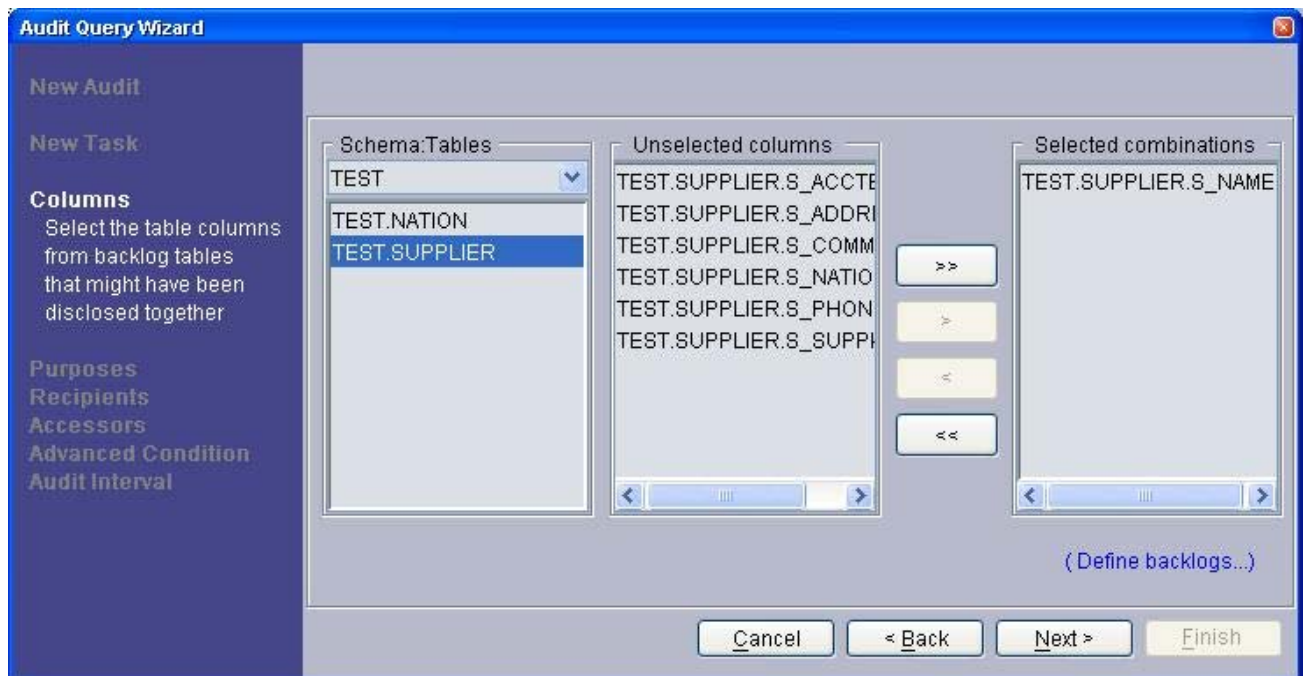
2.1 Name

The name entered for the audit task (query) has to be unique. This is automatically enforced by the GUI.



2.2 Columns

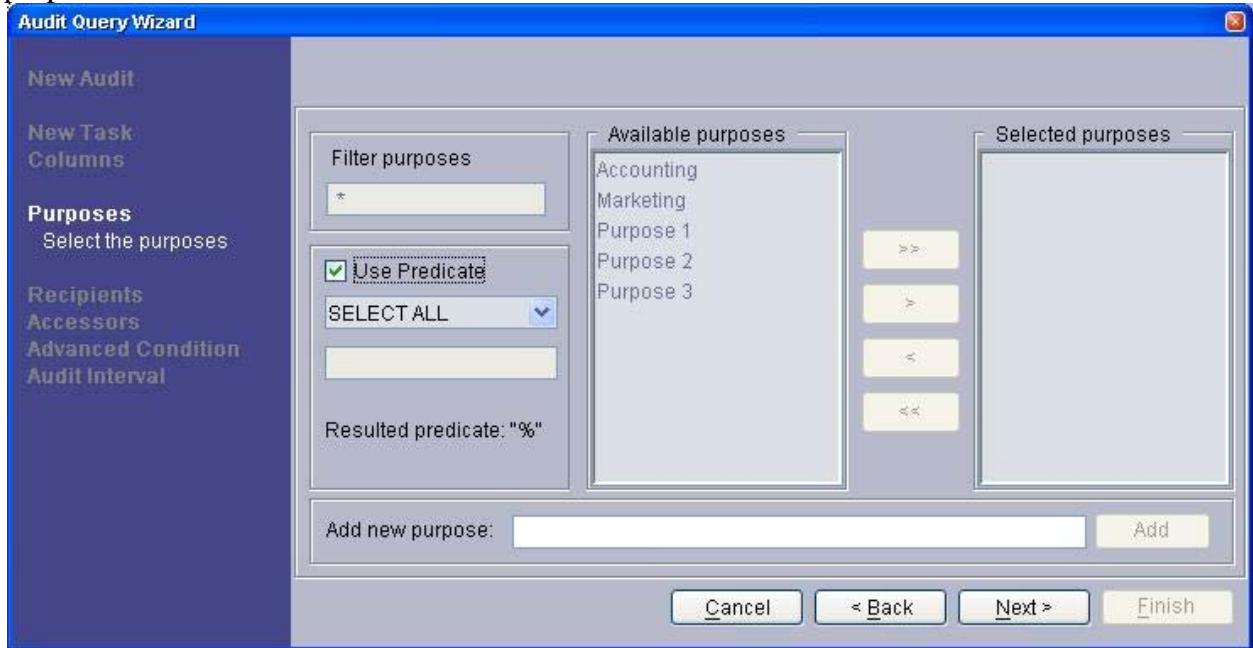
These are the columns that logged queries need to reference (e.g., in SELECT or WHERE clauses) to be candidates for auditing.



The window allows specifying a set of columns where each column may belong to a different table of the database to be audited. At least one column has to be specified.

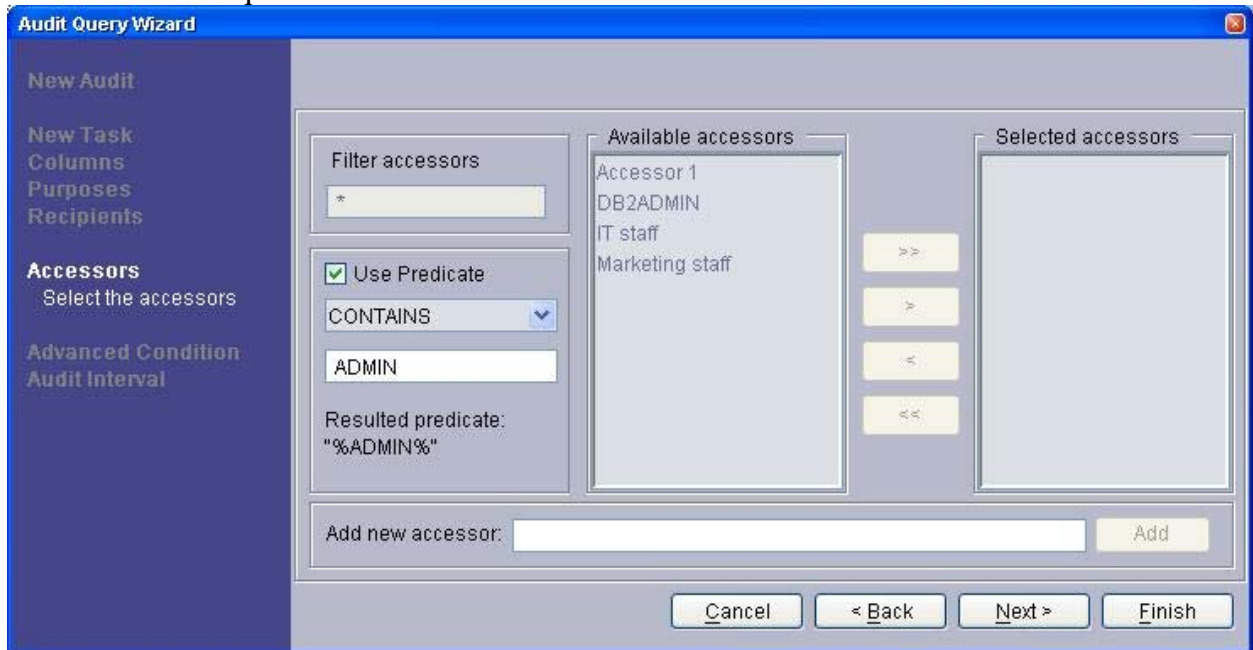
2.3 Purposes

These are the purposes (according to policies) for which queries have been executed. By choosing “SELECT ALL”, the auditor wants to examine whether there are queries that ran for *any* purpose.



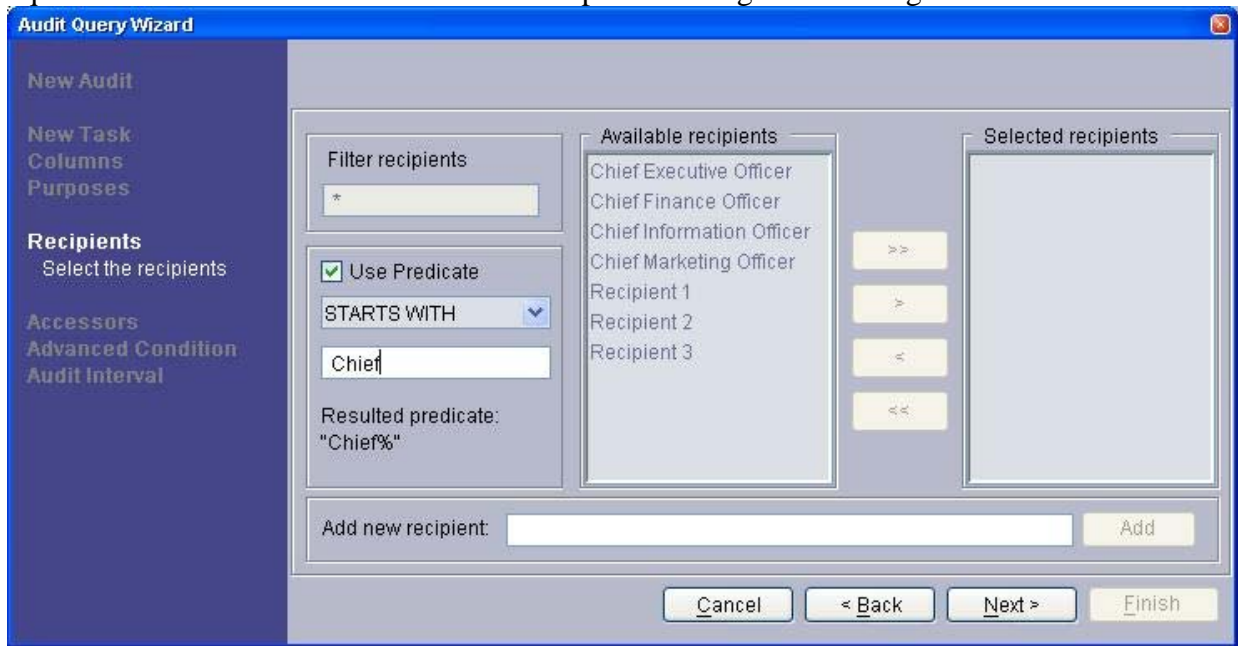
2.4 Accessors

The accessors are the authorized database users who run queries against the database. The auditor wants to track queries from accessors whose ID's contain the name ADMIN.



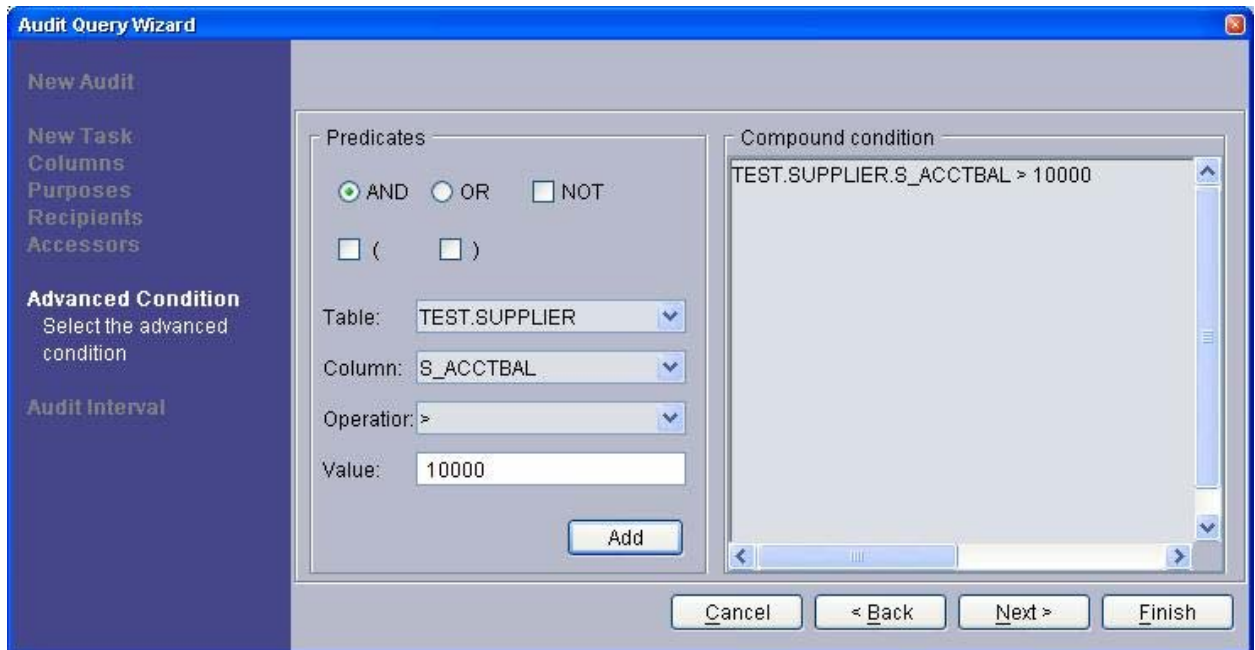
2.5 Recipients

These recipients are the intended final consumers of query results. The auditor wants to track queries whose results were disclosed to recipients having ID's starting with "Chief".



2.6 Condition

The condition is used to specify the data whose disclosure is tracked by the audit. For example, an auditor may only be interested in queries that disclosed information about customers who (1) are female and (2) come from California; or some Suppliers whose account balance is greater than 10000.



The condition panel allows conditions using pull-down menus. The GUI assists in choosing only valid schema, table, and column names. However, it is the user's responsibility to choose valid constant values such that the condition is a valid SQL92 condition since the syntax of constants differs for different data types. For example, if an attribute is a text string (CHAR, VARCHAR, etc.), the constant must be enclosed by single quotes. Similarly, date and time constants take a valid string representation. The details about expressions can be found in the manuals for DB2 [2]. The syntax for conditions in the automatic edit mode is defined in EBNF as follows:

```

<expression> ::= <term> |
                `NOT' <term> |
                `(' <term> `)' |
                <expression> <logicalOp> <term>
<term> ::= <attribute> <comparisonOp> <constant>
<logicalOp> ::= `AND' | `OR'
<comparisonOp> ::= `<' | `<=' | `=' | `>=' | `>' | `<>'

```

In other words, it is the set of simple Boolean expressions that can be bracketed and whose terms are attribute-constant comparisons.

An empty condition is considered as a “true” predicate. Example expressions that can be formulated in the automatic mode are

- customer = 'Claire Young'
- (customer = 'Claire Young' AND zip <> 'CA') OR NOT zip = 'NY'
- birthdate = DATE('1960-07-04')

Although many typical conditions can be expressed in the automatic edit mode, only the manual mode allows formulating also join or other more complex conditions. In manual mode, the auditor types in the condition as text directly in the edit window. The following condition, a join, can be edited in the manual edit mode only:

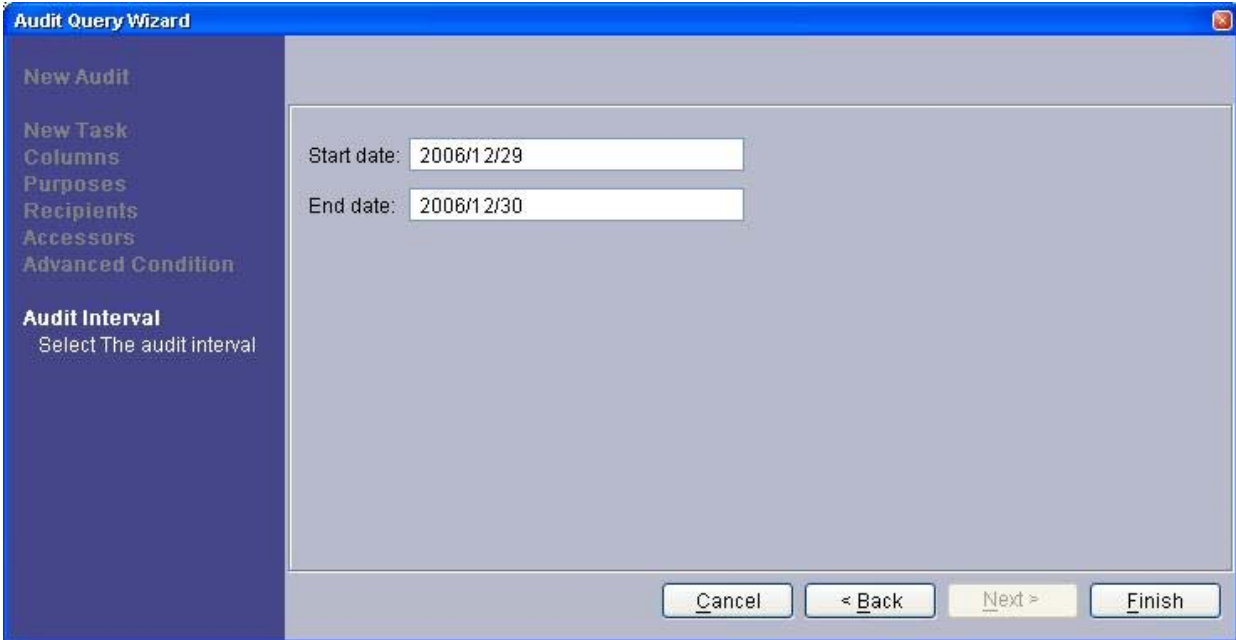
- ffidb.customers.address = ffidb.employees.homeaddr

The same is true for more complex conditions, like

- ffidb.customers.address IN (SELECT homeaddr FROM ffidb.employees WHERE gender = 'F')

Time Interval

User queries are associated with a timestamp denoting the successful execution by the database system.



The screenshot shows the 'Audit Query Wizard' window. On the left is a dark blue sidebar with a list of steps: 'New Audit', 'New Task', 'Columns', 'Purposes', 'Recipients', 'Accessors', 'Advanced Condition', and 'Audit Interval'. The 'Audit Interval' step is selected and highlighted. Below the sidebar, the main area contains two text input fields: 'Start date:' with the value '2006/12/29' and 'End date:' with the value '2006/12/30'. At the bottom right of the window are four buttons: 'Cancel', '< Back', 'Next >', and 'Finish'.

The timestamp of logged queries qualifying for the audit query must lie within the specified interval boundaries (inclusive). The GUI automatically enforces that the interval is valid.

3. Database Schema Evolution

During the time the audit system is in use, the database schema may change several times. For example, new tables are added to the schema, columns of a table are dropped, two tables are joined into one table, tables are normalized, columns are renamed, and so forth. Audit queries may no longer be valid because the database objects (tables/columns) referenced in the query string changed. The query log may contain queries that were formulated against previous schemas and hence it is a challenge to audit these queries with an audit expressed against the current schema of the database.

3.1 Schema Change Detection

When the auditor switches to the audit tool, the system checks whether the current audit metadata is up to date in the connected databases, that is, whether there was a schema change in any of the databases and these changes have not been recorded in the audit metadata. If there is such a situation, a window informs about the tables in the current database schema for which there is either no backlog table in the audit metadata (a new table was created) or the schema of the table differs from the schema of the backlog table (for example, a column has been added to the source table).

The auditor can also manually check whether the audit metadata is up to date by selecting Database→<database> in the audit tree, right click, select “Check for schema change.” This should be done immediately following a change to the database schema in order to correctly audit queries after a schema change.

3.2 Reconciliation of a Table Schema with Clio

When a schema change to a table has been detected, the auditor can then select the table that she wishes to update in the audit metadata. We call the process that tells the system how to map (stale) tables to the new backlog table “reconciliation.” When a table is selected, the user is informed about the columns available in the audit metadata for the table and the actual list of columns of the current table. The user can then start a schema mapping tool called “Clio” that assists in the reconciliation. The schema reconciliation process works as follows:

The user selects a table from the list of tables that need to be reconciled and clicks “OK.”

A window named “Reconciliation SQL from Clio” appears. The user clicks on the button “Start Clio.”

After short time, the Clio tool appears and the panel “Schema view” is displayed. The user can perform the mapping of the attributes from the stale/old backlog table to the attributes of the current table in the database. The result of this mapping is a view definition for the new backlog table.

When the user has completed the mapping (by connecting source attributes on the left with target attributes on the right), she can switch to the panel “Query.”

In this panel, she selects the radio button “SQL-92.” The text panel then displays the view definition for the new version of the backlog table.

She selects the mapping query text in this text area by clicking into the text area, typing “Control-c” to copy the text into a paste buffer of the operating system.

She clicks once into the empty text area “SQL backlog reconciliation subquery” of the Wizard window, where Clio was started, and types “Control-v.” The view definition now is

pasted into the text area. The pasted subquery is a part of a SQL query that can be viewed by checking the box “Show full SQL query.”

By clicking on the button “OK,” the user has successfully defined the table schema reconciliation.

4. Limitations

HDBA has some limitations, listed below.

The only language currently supported in the GUI is English.

The length of unqualified table names should be at least 3 characters less than the maximum size of tables names in DB2, which is 128 in IBM DB2 UDB version 8.¹ The reason is that backlog tables use prefixes like “Bx”, where “x” is an integer number starting with 1. For example, the 3rd backlog version of table `employee` has the unqualified name `B3_employee`. For versions that require several digits in a decimal representation, which is highly unlikely in practice, shorter table names should be used.²

Appendix A. Description of HDBADMIN Database Tables

Below is a description of the HDBADMIN tables created by the HDB Control Center for auditing services (all columns do not admit null values, i.e., they are “NOT NULL”):

HDBADMIN.AUDIT (“AUDITID” VARCHAR(32), “TASKID” VARCHAR(32), “VERSION” VARCHAR(32), “CONDITION” CLOB (32627), “BEGIN” TIMESTAMP, “END” TIMESTAMP)

- Audit specifications are stored in this table. These specifications are created by wizards and can be reused. The AUDITID, TASKID, VERSION combination uniquely identifies an audit specification. The condition is an audit expression as an SQL query. The BEGIN and END values are the time interval for selecting logged queries subject to the audit. A logged query must have run between this interval to qualify for the audit.

HDBADMIN.AUDIT_PURPOSES (“AUDITID” VARCHAR(32), “TASKID” VARCHAR(32), “VERSION” VARCHAR(32), “PURPOSE” VARCHAR(32))

- This table lists purposes for selecting logged queries subject to an audit. The combination of AUDITID, TASKID, VERSION uniquely identifies an audit. Each admissible purpose appears as a separate tuple in this table. Purposes can include the SQL wildcard character ‘%’ for partial matching.

HDBADMIN.AUDIT_RECIPIENTS (“AUDITID” VARCHAR(32), TASKID VARCHAR(32), “VERSION” VARCHAR(32), “RECIPIENT” VARCHAR(32))

- This table lists recipients for selecting logged queries subject to an audit. The combination of AUDITID, TASKID, VERSION uniquely identifies an audit. Each admissible recipient appears as a separate tuple in this table. Recipients can include the SQL wildcard character ‘%’ for partial matching.

¹ See the following Web page for SQL limits in IBM DB2 UDB version 8:

<http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.doc/admin/r0011049.htm>

² This limitation should be removed in the future by using a different, appropriate representation of unqualified backlog table names. It is also possible to use a different base for the number representation (enumerate Unicode letters in a certain range—which would cover several orders of magnitude of backlog versions while using only a single “digit” = character).

HDBADMIN.AUDIT_ACCESSORS (“AUDITID” VARCHAR(32), TASKID VARCHAR(32), “VERSION” VARCHAR(32), “ACCESSOR” VARCHAR(32))

- This table lists accessors (or database users) for selecting logged queries subject to an audit. The combination of AUDITID, TASKID, VERSION uniquely identifies an audit. Each admissible accessor appears as a separate tuple in this table. Accessors can include the SQL wildcard character ‘%’ for partial matching.

HDBADMIN.AUDIT_PROJECTION_COLUMNS (“AUDITID” VARCHAR(32), “TASKID” VARCHAR(32), “VERSION” VARCHAR(32), “SCHEMA” VARCHAR(128), “TABLE” VARCHAR(128), “COLUMN” VARCHAR(30))

- This table lists the database columns that must appear in a logged query for it to be considered a candidate for the audit. The combination of AUDITID, TASKID, VERSION uniquely identify an audit. The combination of SCHEMA, TABLE, COLUMN specify the columns whose information disclosure are being tracked by the audit. Each individual column is represented as a tuple in this table.

HDBADMIN.QUERYLOG (“QUERY” CLOB(32627), “TIM1” TIMESTAMP, “TIM2” TIMESTAMP, “ISOLATION” INT, “USR” VARCHAR(128), “PURPOSE” VARCHAR(128), “RECIPIENT” VARCHAR(128), “SPEC_RECIP” VARCHAR(128))

- This table is the log of individual queries submitted to the database overtime. The log only contains queries of committed transactions. The log is automatically populated by the HDB JDBC driver by adding a tuple to the log with each query submitted to the database. The QUERY attribute represents the query string. TIM1 is the timestamp the query started or was submitted to the database, and TIM2 is the timestamp when the cursor over the query’s result was closed. These two attributes represent the time interval during which the query was evaluated and accessed the database. These intervals are then used at the time of audit to recover the snapshot of the database that the query accessed; to assess the information accessed and disclosed by the query. The ISOLATION attribute represents the transaction isolation level under which the query ran. These integer values are obtained by the JDBC getTransactionIsolation method on the HDB JDBC connection object used to run the user’s query. Knowing the isolation level with which a query ran is necessary in order to determine the data it accessed. The USR, PURPOSE, RECIPIENT, SPEC_RECIP attributes describe the authenticated user (or accessor) running the query, the purpose, recipient, and spec_recip for which the query was run.

HDBADMIN.ABV (“TSHEMA” VARCHAR(128), “TNAME” VARCHAR(128), “BSHEMA” VARCHAR(128), “BNAME” VARCHAR(128), “TIM” TIMESTAMP, “ABV” CLOB(32627), “IDKEY” BIGINT)

- This table describes the mapping from source tables referenced in queries and audits to backlog tables that store full temporal snapshots of source tables. TSHEMA, TNAME represent the schema name-table name of a source table while BSHEMA, BNAME give the schema name-table name of a backlog table that stores multiple versions of tuples from the source table. If a source table evolves over time, there can be multiple backlog tables, each with a different schema representing the schema of the source table at an earlier point in time. When a schema change is detected, a new backlog table is created to match the new schema of the table. This fact is recorded as a tuple in this table. The creation time of the backlog table is recorded in the TIM attribute. The ABV attribute is a SQL expression used to map the current schema of a backlog

table to previous schemata. This is needed to audit queries that ran over a previous schema of the source table. IDKEY uniquely identifies each tuple in the table.

HDBADMIN.DROPPEDCOLUMNS (“TSHEMA” VARCHAR(128), “TNAME” VARCHAR(128), “TCOLUMN” VARCHAR(128), “DATATYPE” SMALLINT, “TYPENAME” VARCHAR(128), “COLSIZE” INTEGER, “SCALE” INTEGER, “POS” INTEGER, “ISNULLABLE” VARCHAR(4), “TIM” TIMESTAMP)

- When auditing in the presence of evolving schemata, columns that existed in past schemata may no longer exist. However, these columns should nevertheless remain visible for the purposes of auditing past disclosure of information they may have contained. This table stores dropped columns that no longer exist in the source schema of a table or its backlog. Attributes TSHEMA, TNAME identify a table using its schema-table name. TCOLUMN is the name of a dropped column; DATATYPE, TYPENAME, COLSIZE, SCALE, POS, ISNULLABLE are metadata associated with the column when it existed. TIM identifies when the column was dropped from the table.

Appendix B. Sample Metadata with Schema Evolution

B.1 Adding/Dropping Columns

We show metadata for simple schema evolution using the TPCD.NATION table that has four attributes (N_NATIONKEY, N_NAME, N_REGIONKEY, N_COMMENT). At some time, an additional attribute N_NEW was added to the table, and at a later time, it was removed. This is shown in the three consecutive backlog tables that we created for these changes; B1_NATION was the original backlog table created for NATION, B2_NATION has the extra N_NEW ATTRIBUTE, followed by B3_NATION which no longer has the N_NEW attribute.

```
create table TPCD.B1_NATION (  
    N_NATIONKEY      integer not null,  
    N_NAME           char(25) not null,  
    N_REGIONKEY     integer not null,  
    N_COMMENT       varchar(152),  
    opr             char(1) not null,  
    tim             timestamp not null,  
    primary key (N_NATIONKEY, tim)  
);
```

```
create table TPCD.B2_NATION (  
    N_NATIONKEY     integer not null,  
    N_NAME          char(25) not null,  
    N_REGIONKEY    integer not null,  
    N_COMMENT      varchar(152),  
    N_NEW          integer not null,  
    opr           char(1) not null,  
    tim          timestamp not null,  
    primary key (N_NATIONKEY, tim)  
);
```

```
create table TPCD.B3_NATION (  
    N_NATIONKEY     integer not null,  
    N_NAME          char(25) not null,  
    N_REGIONKEY    integer not null,  
    N_COMMENT      varchar(152),  
    opr           char(1) not null,  
    tim          timestamp not null,  
    primary key (N_NATIONKEY, tim)  
);
```

```

N_NATIONKEY      integer not null,
N_NAME           char(25) not null,
N_REGIONKEY      integer not null,
N_COMMENT        varchar(152),
opr              char(1) not null,
tim              timestamp not null,
primary key (N_NATIONKEY, tim)
);

```

The following insert statement created metadata for the dropped column N_NEW after the system detected a schema change which resulted in the creation of table B3_NATION. The timestamp shows the time when the missing attribute was detected, and before the creation of B3_NATION.

```

insert into HDBADMIN.DROPPEDCOLUMNS values(
  'TPCD', 'NATION3', 'N_NEW', 4,
  'INTEGER', 10, 0, 1, 'NO',
  '2004-01-14-17.05.36.312001');

```

The three following tuples were inserted into the HDBADMIN.ABV table as metadata describing the mappings for table TPCD.NATION through schema evolution for auditing. The 1st tuple was inserted when a backlog table was first created for NATION. Therefore, the mapping query for this table is a simple select since B1_NATION is the only backlog table that ever existed for NATION. The 2nd insert statement was issued after the system detected that N_NEW was added to the schema of NATION and backlog table B2_NATION was created. The mapping query is a SQL union that maps the two backlog tables into the schema of the new backlog table. Since N_NEW didn't exist in B1_NATION, its value is mapped to a null value. The 3rd insert statement was issued after N_NEW was dropped from table NATION and a new backlog table B3_NATION was created. Its mapping query reconciles the schemas of B1_NATION, B2_NATION, B3_NATION to the schema of B3_NATION. B3_NATION no longer has attribute N_NEW, but N_NEW should still be visible for auditing; N_NEW appears as the 1st column that is mapped in the query. Dropped columns are mapped first followed by remaining columns.

```

insert into HDBADMIN.ABV values(
  'TPCD',
  'NATION',
  'TPCD',
  'B1_NATION',
  '2004-01-12-17.05.36.312001',
  select    N_NATIONKEY, N_NAME, N_REGIONKEY, N_COMMENT, opr, tim
  from      tpcd.bl_nation',
  default);

```

```

insert into HDBADMIN.ABV values(
  'TPCD',
  'NATION',
  'TPCD',
  'B2_NATION',
  '2004-01-13-17.05.36.312001',
  select    N_NATIONKEY, N_NAME, N_REGIONKEY, N_COMMENT,
  cast(null as integer), opr, tim
  from      tpcd.bl_nation q

```

```

union
select  N_NATIONKEY, N_NAME, N_REGIONKEY, N_COMMENT, N_NEW,
        opr, tim
from    tpcd.b2_nation',
default);

```

```

insert into HDBADMIN.ABV values(
  'TPCD',
  'NATION',
  'TPCD',
  'B3_NATION',
  '2004-01-14-17.05.36.312001',
  'select  cast(null as integer) as N_NEW,
          N_NATIONKEY as N_NATIONKEY,
          N_NAME as N_NAME,
          N_REGIONKEY as N_REGIONKEY,
          N_COMMENT as N_COMMENT, opr, tim
from      tpcd.b1_nation

union
select    N_NEW, N_NATIONKEY, N_NAME, N_REGIONKEY, N_COMMENT,
          opr, tim
from      tpcd.b2_nation

union
select    cast(null as integer) as N_NEW,
          N_NATIONKEY, N_NAME, N_REGIONKEY, N_COMMENT, opr, tim
from      tpcd.b3_nation',
default);

```

B.2 Horizontal Table Splitting/Merging

We show metadata for horizontal partitioning schema evolution using the TPCD.NATION table that has four attributes (N_NATIONKEY, N_NAME, N_REGIONKEY, N_COMMENT). The initial backlog table B1_NATION was created to match the initial schema of the NATION table. Later in time, the NATION table was split into two tables NATIONA and NATIONB, each having a disjoint subset of tuples in NATION. Tuples with N_NATIONKEY <= 12 were placed into NATIONA, the remaining tuples were placed into NATIONB. Backlog tables for these tables were, respectively, B2_NATIONA, B2_NATIONB. Thereafter, the two tables NATIONA and NATIONB were merged back into NATION with backlog table B3_NATION. However, the schema of the tables remained the same throughout these changes.

```

create table TPCD.B1_NATION (
  N_NATIONKEY  integer not null,
  N_NAME       char(25) not null,
  N_REGIONKEY  integer not null,

```

```

N_COMMENT      varchar(152),
opr            char(1) not null,
tim            timestamp not null,
primary key (N_NATIONKEY, tim)
);

```

```

create table TPCD.B2_NATIONA (
N_NATIONKEY    integer not null,
N_NAME         char(25) not null,
N_REGIONKEY    integer not null,
N_COMMENT      varchar(152),
opr            char(1) not null,
tim            timestamp not null,
primary key (N_NATIONKEY, tim)
);

```

```

create table TPCD.B2_NATIONB (
N_NATIONKEY    integer not null,
N_NAME         char(25) not null,
N_REGIONKEY    integer not null,
N_COMMENT      varchar(152),
opr            char(1) not null,
tim            timestamp not null,
primary key (N_NATIONKEY, tim)
);

```

```

create table TPCD.B3_NATION (
N_NATIONKEY    integer not null,
N_NAME         char(25) not null,
N_REGIONKEY    integer not null,
N_COMMENT      varchar(152),
opr            char(1) not null,
tim            timestamp not null,
primary key (N_NATIONKEY, tim)
);

```

The first insert statement (below) followed the creation of the backlog table B1_NATION and describes the mapping as a simple select from the backlog table. However, when the backlog table for NATIONA was created, the mapping needed to link previous tuples in B1_NATION with tuples in B1_NATIONA. The mapping query for NATIONA selects the tuples of B1_NATION that would have appeared in B2_NATIONA; this is achieved using the predicate `n_nationkey <= 12` over B1_NATION. These tuples are union'ed with those in B2_NATIONA to form the complete set of tuples in NATIONA through schema evolution. Since NATIONA is evolved from NATION, we assume that no updates to B1_NATION occur after the creation of B2_NATIONA. Otherwise, audits will not function correctly. B2_NATIONB is organized similarly to B1_NATIONA. For table partitioning schema evolution like this, all partitions must have the same timestamp creation time for auditing to function correctly. This is the case for NATIONA, and NATIONB and their respective backlogs which have a timestamp creation time of 2004-01-13-17.05.36.312001 in this sample. The final mapping of NATIONA, NATIONB back into NATION is shown as the last insert statement in the list; all partitions and previous backlogs are union'ed to form the complete set of tuples in NATION over evolving schemata.

```

insert into HDBADMIN.ABV values(
    'TPCD',
    'NATION',
    'TPCD',
    'B1_NATION',
    '2004-01-12-17.05.36.312001',
    'select * from tpcd.b1_nation',
    default);

```

```

insert into HDBADMIN.ABV values(
    'TPCD',
    'NATIONA',
    'TPCD',
    'B2_NATIONA',
    '2004-01-13-17.05.36.312001',
    'select *
     from tpcd.b1_nation
     where n_nationkey <= 12

     union
     select *
     from tpcd.b2_nationa',
    default);

```

```

insert into HDBADMIN.ABV values(
    'TPCD',
    'NATIONB',
    'TPCD',
    'B2_NATIONB',
    '2004-01-13-17.05.36.312001',
    'select *
     from tpcd.b1_nation
     where n_nationkey > 12

     union
     select *
     from tpcd.b2_nationb',
    default);

```

```

insert into HDBADMIN.ABV values(
    'TPCD',
    'NATION',
    'TPCD',
    'B3_NATION',
    '2004-01-14-17.05.36.312001',
    'select *
     from tpcd.b1_nation

     union
     select *
     from tpcd.b2_nationa

```

```

union
select *
from   tpcd.b2_nationb

union
select *
from   tpcd.b3_nation',
default);

```

B.3 Vertical Table Splitting/Merging

We show metadata for vertical partitioning schema evolution using the TPCD.NATION table that has four attributes (N_NATIONKEY, N_NAME, N_REGIONKEY, N_COMMENT). Backlog table B1_NATION is created for the initial schema of NATION. Thereafter, the table gets partitioned into two tables NATIONA, NATIONB, each table has all the tuples of the previous backlog, but a disjoint set of attributes with the exception of the primary key attribute N_NATIONKEY; NATIONA has attributes N_NAME, and N_REGIONKEY while NATIONB has attribute N_COMMENT. Finally, the partitions are re-united into a single table NATION with all attributes. The following statements show the creation of backlog tables for the evolving schemata.

```

create table TPCD.B1_NATION (
  N_NATIONKEY      integer not null,
  N_NAME           char(25) not null,
  N_REGIONKEY      integer not null,
  N_COMMENT        varchar(152),
  opr             char(1) not null,
  tim             timestamp not null,
  primary key (N_NATIONKEY, tim)
);

```

```

create table TPCD.B2_NATIONA (
  N_NATIONKEY      integer not null,
  N_NAME           char(25) not null,
  N_REGIONKEY      integer not null,
  opr             char(1) not null,
  tim             timestamp not null,
  primary key (N_NATIONKEY, tim)
);

```

```

create table TPCD.B2_NATIONB (
  N_NATIONKEY      integer not null,
  N_COMMENT        varchar(152),
  opr             char(1) not null,
  tim             timestamp not null,
  primary key (N_NATIONKEY, tim)
);

```

```

create table TPCD.B3_NATION (
  N_NATIONKEY      integer not null,

```

```

N_NAME          char(25) not null,
N_REGIONKEY     integer not null,
N_COMMENT       varchar(152),
opr             char(1) not null,
tim             timestamp not null,
primary key (N_NATIONKEY, tim)
);

```

The insert statements below define the mapping between previous schemata and the current schema. The 1st statement was issued when backlog table B1_NATION was created, and its mapping query is a simple select from the backlog table. The following two insert statements show the mapping from the initial backlog to the partitioned schemata. In each case, the columns belonging to each partition are projected from the initial backlog and union'ed with the partitioned backlog. As for horizontal partitioning, it is necessary that the partitioned backlogs have an identical creation timestamp; in the sample below, each has 2004-01-13-17.05.36.312001. Once the partitioned schema is created, further updates to the initial schema and its backlog B1_NATION should not occur. The last insert statement shows the mapping from the partitioned schemata to a unified schema. The 1st subquery is a left outer join that selects tuples of B2_NATIONA with or without matching tuples in B2_NATIONB. The next subquery specifically selects tuples in B2_NATIONB that did not match any tuples in B2_NATIONA using an existential subquery. These two subqueries might have been substituted with a single full outer join operation. However, the derby parser that we are currently using with HDB does not support full outer join query syntax. The last two subqueries are simple selects over the remaining schemata having merged partitions.

```

insert into HDBADMIN.ABV values(
  'TPCD',
  'NATION',
  'TPCD',
  'B1_NATION',
  '2004-01-12-17.05.36.312001',
  'select * from tpcd.bl_nation',
default);

```

```

insert into HDBADMIN.ABV values(
  'TPCD',
  'NATIONA',
  'TPCD',
  'B2_NATIONA',
  '2004-01-13-17.05.36.312001',
  'select N_NATIONKEY, N_NAME, N_REGIONKEY, opr, tim
  from tpcd.bl_nation

  union
  select * from tpcd.b2a_nation',
default);

```

```

insert into HDBADMIN.ABV values(
  'TPCD',
  'NATIONB',
  'TPCD',
  'B2_NATIONB',

```

```

'2004-01-13-17.05.36.312001',
'select N_NATIONKEY, N_COMMENT, opr, tim
from tpcd.bl_nation

union
select * from tpcd.b2b_nation',
default);

insert into HDBADMIN.ABV values(
'TPCD',
'NATION',
'TPCD',
'B3_NATION',
'2004-01-14-17.05.36.312001',
'select x.n_nationkey, x.n_name, x.n_regionkey,
y.n_comment, x.opr , x.tim
from tpcd.b2a_nation x left outer join tpcd.b2b_nation y
on (x.n_nationkey=y.n_nationkey)

union
select y.n_nationkey, cast(null as char(25)),
cast(null as integer), y.n_comment, y.opr , y.tim
from tpcd.b2b_nation y
where not exists (select 1
from tpcd.b2a_nation x
where x.n_nationkey=y.n_nationkey)

union
select *
from tpcd.bl_nation

union
select *
from tpcd.b3_nation',
default);

```

5. References

- [1] IBM Hippocratic Database Active Enforcement: User Guide, Version 1, 2006.
- [2] IBM DB2 Universal Database: SQL Reference Volume 1, Version 8, 2002.
Similar information available online: <http://publib.boulder.ibm.com/infocenter/db2luw/v8/>