Unleashing DB2 10 for Linux, UNIX, and Windows

Discover storage and availability features that enable faster business decisions

Simplify application development and portability

Explore real-world examples to launch your DB2 10 experience

Danny Arnold
Steve Diniro
Vivian Lee
Stan Musker
Jo-Ann Woods

Redbooks

ibm.com/redbooks

International Technical Support Organization

Unleashing DB2 10 for Linux, UNIX, and Windows

August 2012

SG24-8032-00

Note: Before using this information and the product it supports, read the information in "Notices" on page vii.

First Edition (August 2012)

This edition applies to Version 10, Release 1 of DB2 for Linux, UNIX, and Windows.

Contents

Notices

Preface

This IBM® Redbooks® publication provides a broad understanding of the key features in IBM DB2® 10 and how to use these features to get more value from business applications. It includes information about the following features:

- *Time Travel Query*, which you use to store and retrieve time-based data by using capability built into DB2 10, without needing to build your own solution
- *Adaptive compression*, an enhanced compression technology that adapts to changing data patterns, yielding extremely high compression ratios
- *Multi-temperature storage*, which you may use to optimize storage costs by identifying and managing data based on its "temperature" or access requirements
- *Row and column access control*, which offers security access enforcement on your data, at the row or column level, or both
- *Availability enhancements*, which provide different DB2 availability features for different enterprise needs; high availability disaster recovery (HADR) multiple-standby databases provide availability and data recovery in one technology, and the IBM DB2 pureScale® Feature provides continuous availability and scalability that is transparent to applications
- *Oracle compatibility*, which allows many applications written for Oracle to run on DB2, virtually unchanged
- *Ingest utility*, a feature-rich data movement utility that allows queries to run concurrently with minimal impact on data availability

This book is for database administrators and application developers who are considering DB2 10 for a new installation, an upgrade from a previous version of DB2, or a migration from a vendor product. It explains key features and illustrates concepts with real-world examples to provide a clear idea of how powerful these features are. Together, these features provide functionality that lower operational costs, ease development, and provide greater reliability.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Danny Arnold is an Executive IT Specialist in IBM Information Management with the Worldwide Enablement team. Danny has been working with the IBM technical sales community and customers to promote DB2 for Linux, UNIX, and Windows technologies since joining IBM in 1995. In addition to writing white papers and other technical documents, Danny has presented at local and regional DB2 User Group meetings worldwide on the benefits of DB2 technology from a business perspective. To complement his DB2 skills, Danny has achieved the Oracle Certified Professional (OCP) credential for Oracle Database, and Distinguished IT Specialist certification from The Open Group.

Steve Diniro is an Information Developer at the IBM Canada Lab in Toronto. He has been part of DB2 documentation team for over 10 years, and has written a wide variety of DB2 content, including context-sensitive help, monitoring, installation help, IBM DB2 Connect™, and reference information. Steve has industry experience as both a database administrator and an application developer.

Vivian Lee is a Senior Manager with the DB2 Information Development team in Toronto and Program Manager with the IBM Canada Lab. She has been in the DB2 organization for over 10 years, working in a number of areas, including product documentation and release management.

Stan Musker is an Information Developer working at the IBM Canada Lab in Toronto. He has been creating product documentation for 24 years, the last 14 years in Information Management. He is currently focused on the compatibility, temporal, and monitoring features available in DB2. In addition, he has helped develop product best practices, videos, tutorials, and eBooks.

Jo-Ann Woods is an Information Developer for DB2 for Linux, UNIX, and Windows, based at the IBM Canada Lab in Toronto. She has over nine years of experience with DB2 products. Her areas of expertise include product prerequisites, installation, licensing, and DB2 pureScale Feature.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  `http://www.facebook.com/IBMRedbooks`
- Follow us on Twitter:

  `http://twitter.com/ibmredbooks`
- Look for us on LinkedIn:

  `http://www.linkedin.com/groups?home=&gid=2130806`
- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  `https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm`
- Stay current on recent Redbooks publications with RSS Feeds:

  `http://www.redbooks.ibm.com/rss.html`

1

Introduction to DB2 10

In this chapter, we introduce key features of the DB2 10 database software product. We also provide an overview of the fictional company that is used throughout the book to illustrate how these key features are applied in a real-world business environment.

1.1 Purpose of this book

In this book, we use practical examples to highlight several new key features of DB2 10. The examples illustrate how you can apply and use the rich new functions of DB2 10 in a real-world business environment. The examples are syntactically correct and demonstrate how you can take advantage of key DB2 10 functionality in your own environment.

Each chapter of this book describes one of the following functions or features and shows how each can be applied in a business environment:

- Time Travel Query: Apply time-based state information to your data, so you can store and retrieve time-based data without having to build your own customized solution. Use functionality that is built into the database engine.
- Adaptive compression: Compress data at both the table and page level, but keep data available; page-level dictionaries automatically adjust to changes in data. You can still do offline-table reorganizations to regain optimal compression ratios, but this operation is required much less often, if at all, compared to previous versions of DB2.
- Multi-temperature storage: Optimize storage costs to match the access requirements of your data, so you can use faster, more expensive storage only for frequently accessed “hot” data, and dynamically move less frequently accessed “warm” or “cold” data to slower, less expensive storage.
- Row and column access control (RCAC): Control, at the row or column level, or both, who can access which parts of your data. This control is enforced at the data level, regardless of how the data is accessed.
- Availability enhancements: Extend your high availability disaster recovery (HADR) strategy with additional multiple standby support. Or reach an even higher level of availability with the DB2 pureScale Feature, which offers continuous availability, even under extreme circumstances such as sudden hardware or software failure.
- Oracle compatibility: Migrate your Oracle applications to use DB2 for Linux, UNIX, and Windows by exploiting increased levels of code compatibility. In almost all cases, you can take an existing Oracle application and run it on DB2 without having to make any source code changes.
- Ingest utility: Populate your data warehouse tables with large volumes of data without affecting availability of your data. Your warehouse continues to process queries while the data is streamed into it. You can ingest data from files or from pipes that feed from a variety of sources, including existing databases.

1.2 Introduction to DB2 10

DB2 10 introduces many new features that help you lower your administrative, application development, and storage costs, while maintaining highly reliable and available database environments.

Several features in this release allow you to take advantage of native capability built directly into DB2, minimizing development time and costs by avoiding the need for an application developer to build the capabilities into the application.

For example, with Time Travel Query functionality, an application can simply use SQL to determine the data in a row of a table at a particular point in time.

Also consider data security requirements imposed by company policies, or government or regulatory bodies. To help meet these requirements, you can use the robust data security features in DB2 10, which has been further expanded with row and column access control on data. With this feature, you can restrict what data gets returned to whom from a query, depending on the permissions and masks you implement. Your application does not need to be modified to take advantage of this feature.

DB2 10 also extends its SQL compatibility support with Oracle Database, first introduced in DB2 9.7. With this expanded support, which includes new scalar functions and expanded support for triggers and user-defined data types, migrating an Oracle application to DB2 for Linux, UNIX, and Windows is easier than ever.

For your data warehousing needs, DB2 10 offers the ingest utility for populating warehouse tables without downtime. There is minimal effect to existing query workloads or server resources; users can continue to query the warehouse while new data is continuously added. With this feature, real-time querying can be done against the warehouse, instead of having to run queries against the operational database when real-time querying is needed.

These are only several examples of administrative and application development savings that are available to you in DB2 10.

Beyond administrative and application development savings, you can also better manage your storage costs by taking advantage of multi-temperature storage capabilities. Use these capabilities to dynamically store more frequently accessed data (considered “hot” data) on faster storage, and move less frequently accessed data (considered “warm” or “cold” data) to slower storage. With this approach, you can optimize your storage costs by matching storage to data access requirements; use faster, more expensive storage for data that is more frequently accessed, and use slower, less expensive storage for less frequently accessed data.

You can also improve the efficiency of your storage by using the adaptive compression feature that is new to DB2 10. Adaptive compression expands on row compression technology in earlier releases of DB2 for Linux, UNIX, and Windows by providing page-level compression dictionaries in addition to previously available table-level dictionaries. Page-level dictionaries automatically adjust to data changes on a page. Your data remains available while page-level compression dictionaries automatically adapt to changing data values over time.

DB2 10 offers increased availability and reliability with multiple standby databases. You can maintain up to three HADR standby databases. This capability means that you can establish a principal standby database that is tightly synchronized with the primary database for high availability, and keep two auxiliary standby databases for disaster recovery. This HADR configuration can be achieved by solely using DB2 10 technology.

The DB2 pureScale Feature brings another level of availability, reliability, and scalability. With this feature you can rely on continuous availability, whether it is unplanned maintenance or an extreme circumstance. You can also transparently and seamlessly scale your system by adding more members to your clustered environment.

This book explores the powerful capabilities and enhancements that DB2 10 offers to lower your administrative, development, and storage costs, while giving you high availability and reliability.

1.3 Company overview: Great Toque Outdoors

To illustrate the practical application of key features of DB2 10 in a business environment, we base the examples in this book on a fictional company called *Great Toque Outdoors*.

This company is a global corporation that specializes in merchandise for the outdoor lifestyle, supporting leisure activities such as backpacking, fishing, and hiking. Great Toque Outdoors offers its products and services both online and in physical stores. Recently, the company decided to expand its business beyond merchandising to provide adventure travel services.

Great Toque Outdoors runs its inventory and internal business operations, and also its online store and travel services division, on a DB2 10 Advanced Enterprise Server Edition data warehouse.

Throughout this book, we demonstrate how the database administrators (DBAs) and application developers at Great Toque Outdoors can more easily and efficiently take advantage of several new features in DB2 10.

2

Time Travel Query

In this chapter, we describe Time Travel Query, which uses temporal tables within DB2 10 so that database transactions can view data as of a specific point in time: in the past, present, or future to satisfy business requirements. Great Toque Outdoors plans to use Time Travel Query within its adventure travel application.

2.1 Overview of Time Travel Query

A standard SELECT query against a table in a DB2 database retrieves the currently committed value of each data row within that table. In certain instances, an organization cannot use the currently committed value of the data, but wants to use the data results *as of* a specific point in time. In addition to allowing system time, which is automatically provided by DB2, to be used within temporal tables, DB2 10 provides a business time component or a combination of system time and business time. With system time, both past and present information can be queried; with business time, past, present, and future information can be queried. Although business time provides future time, it does not track row changes over time. Because some applications require both a business time component and the ability to track row changes, the bitemporal table in DB2 combines the business time and system time components in a single table. This wide range of time-period support and its associated data values is why this DB2 10 feature is named *Time Travel Query*.

2.2 Business value

Because the adventure travel division of its business is new, Great Toque Outdoors wants to ensure it can deliver outstanding customer service. It also wants to accurately provide customers with adventure travel trips when customers call a Great Toque Outdoors travel agent or view available trips online at Great Toque Outdoors adventure travel website. Because of frequent changes within the travel industry and the impact of cancelled or delayed trips to customers and their overall customer experience, strict adventure-travel trip control is required. Great Toque Outdoors wants the ability to track any change or cancellation of an adventure travel trip over time (data-change audit tracking). Upon receipt of a customer request for refund or a complaint, Great Toque Outdoors wants to be able to see the entire history for all changes, such as length of trip, duration of trip, lodging, location, and so on. This audit ability would provide the documentation and history for any trip change or cancellation and also the exact time of the change.

Great Toque Outdoors first thought about implementing this trip-change tracking capability through application changes and using database triggers to capture trip-change records. An estimation indicated that this custom implementation would take several months of application coding and testing to incorporate these changes. Great Toque Outdoors then heard about DB2 10 introducing temporal tables (specifically system-period temporal tables) and decided this feature would meet its application requirements with minimal effort and greatly reduce any application changes to implement the new trip-change history requirement.

Because the valid dates for a particular travel adventure are not related to the time that the trip is entered into the Great Toque Outdoors database, Great Toque Outdoors required a business time component (business-time-validity checking). Because DB2 10 allows both system-period and application-period (business-time based) temporal tables, Great Toque Outdoors could meet its second application requirement also, with a single implementation. Because Great Toque Outdoors had the requirement for system-period and application-period temporal tables, a “bitemporal” table (combining both system time and business time) in DB2 10 met their requirement.

With the information available in a bitemporal table, Great Toque Outdoors can now extract all valid trips as of a certain timestamp and can display the valid booking date ranges for each trip. This capability can be used for its Internet-based online application, for customers that use their travel agents by telephone, or in conjunction with the travel counter in each retail store. In addition to meeting the business-time requirements of Great Toque Outdoors, this solution provides data-change-audit tracking. This approach enables its customers to select an adventure travel trip that meets their travel availability dates with confidence. This solution improved customer satisfaction and met the expectations that Great Toque Outdoors had set for its adventure travel division.

In addition to meeting its immediate application requirements (data-change-audit tracking and business-time-validity checking), temporal tables in DB2 also provide Great Toque Outdoors a mechanism to recover data as a result of an application or user error, and provide a mechanism to design trips and enter these trips into the application at any time. This data entry can occur even if a particular trip is not eligible for booking until some future point in time (such as the 2016 Summer Olympic Games).

2.3 Business application examples

Using the example of Great Toque Outdoors, this chapter describes two types of temporal tables that are offered in DB2 10:

- System-period temporal tables
- Bitemporal (combination of system and business time) tables.

For each type of temporal table, examples are provided to illustrate the value of Time Travel Query to an organization in the organization’s day-to-day operations. Before we begin the discussion of Time Travel Query for Great Toque Outdoors, we provide some background about the Time Travel Query feature.

2.3.1 Time Travel Query description and usage

Time Travel Query is designed to support system time, business time, or a combination of both system time and business time (referred to as *bitemporal*). Before we show actual examples of Time Travel Query in use, we describe its basic components and how they work.

To begin, we first define system time and business time:

- *System time* is maintained by DB2 and is provided by the system clock on the database server at the time that the database transaction occurs. DB2 captures the valid start time, valid end time, and the transaction time as each update or delete transaction occurs in the database for those tables defined as using system time.
- *Business time* is either a timestamp or a date and provides the valid start and end times for a record in terms of the business and is unrelated to system time (when the transaction actually occurs within the database). Examples of business time are the start and end dates that an automobile insurance policy is valid, the check-in and check-out dates for a hotel reservation, and the valid booking dates for an event (such as a concert).

DB2 10 provides the following types of temporal tables:

- System-period temporal tables (STT)

  Use system time to keep track of rows that are updated or deleted in a separate history table. STT allows all modifications to a base table to be tracked when the table exists as an STT and has the ability to query data transparently at any point in time between the first usage of the STT and the present time.
- Application-period temporal tables (ATT)

  Use timestamps or dates to track the business time period for which each row within the table is valid. Unlike an STT, an ATT requires that application is aware of the ATT. Because business time is being tracked (which can be past, present, or future), DB2 cannot provide the business-time value. The business-time values (start and end time) must be provided by the application. If business time is specified as part of the update or delete statements, DB2 will automatically modify the data rows as required to maintain business time. ATTs allow rows within the table to be based on valid time periods for the business (independent of the system time) and can be used for past, present, or (unique to ATT) future time periods.
- Bitemporal tables

  Combine the characteristics of both STTs and ATTs. With the combination, data can support business time, while tracking all modifications to the table and allowing data access in the past, present, or future (depending on whether you are using business time or system time within the query).

The key difference between an STT and an ATT is that an STT has a separate history table to maintain all rows that are not current rows within the table. An ATT has no separate history table; all rows are maintained within the current table. Because an ATT has no history table, rows for time periods that are affected by delete operations are permanently deleted from the table, which is the same behavior as a regular DB2 table in terms of delete operations.

To track table data at any point in time over the course of a typical application usage of a table, DB2 10 Time Travel Query provides STTs. STTs are not visible to the application that is accessing the base table. To accomplish this transparency, base tables that intend to use system-period time must contain three columns (of TIMESTAMP data type with any precision):

- System-period start time

  This column is the system time when this row came into existence (the time of the original insert or update operation that created this version of the row).

- System-period end time

  This column is the system time when this row was modified with either a newer version of the row or deleted from the table. The setting of this timestamp to the following value indicates it is the current version of the row:

  ```
  9999-12-30-00:00.00
  ```

- Transaction start ID

  This column is the unique identifier for the transaction that modified or inserted the row; it can be used to find all records affected by the same transaction across multiple tables using Time Travel Query. A database can have many transactions occurring in a relatively short time span (milliseconds). To uniquely identify different transactions, a timestamp value provides subsecond granularity.

The system period is specified for a table with a clause in the CREATE TABLE or ALTER TABLE statements, as shown in Example 2-1.

Example 2-1 Clause for CREATE TABLE or ALTER TABLE to define system period

```
PERIOD SYSTEM_TIME(start_period_column, end_period_column)
```

In addition to the system-time-period specification, a history table must be built that has the same column order, column names, and column data types as the original base table where the STT is defined. After the history table is defined, the base table and history table become an STT, after the statement in Example 2-2 is issued.

Example 2-2 ALTER TABLE statement to combine base table with history table for STT

```
ALTER TABLE table_name ADD VERSIONING
   USE HISTORY TABLE history_table_name;
```

Now, we describe the application-period temporal table in more detail. An ATT must have two additional columns that define the business-period start time and business-period end time. Both column definitions must be either TIMESTAMP (any precision allowed) or DATE data type.

- Business-period start time: The timestamp or date (provided by the application) when this particular version of the data row becomes *valid*
- Business-period end time: The timestamp or date (provided by the application) when this particular version of the data row becomes *invalid*

An ATT is specified for a table with the clause in the CREATE TABLE or ALTER TABLE statements, as shown in Example 2-3.

Example 2-3 Clause for CREATE TABLE or ALTER TABLE to define application-period

```
PERIOD BUSINESS_TIME(bus_start_column, bus_end_column)
```

Because there is no history table in an ATT definition, no further action is required to activate the ATT.

Finally, the last type of temporal table, the bitemporal table is a combination of both the STT and ATT concepts. Therefore, the bitemporal table requires all five columns that were described for the STT and ATTs.

- System-period start time

  The system time when this row came into existence (the time of the original insert or update operation that created this version of the row).

- System-period end time

  The system time when this row was modified with either a newer version of the row or deleted. The setting of this timestamp to the following value indicates it is the current version of the row:

  ```
  9999-12-30-00:00.00
  ```

- Transaction start ID

  The unique identifier for the transaction that modified or inserted the row and can be used to find all records affected by the same transaction across multiple tables that use Time Travel Query.

- Business-period start time

  The timestamp or date (provided by the application) that this particular version of the data row becomes valid.
- Business-period end time

  The timestamp or date (provided by the application) that this particular version of the data row becomes invalid.

In addition, both periods (system and business) must be specified as part of the table definition with the clauses in the CREATE TABLE or ALTER TABLE statement shown, as shown in Example 2-4.

Example 2-4 System-period and application-period keywords for ATT definition

```
PERIOD SYSTEM_TIME(start_period_column, end_period_column)
PERIOD BUSINESS_TIME(bus_start_column, bus_end_column)
```

Finally, to complete the bitemporal table definition, you must create the history table (because the bitemporal table has STT characteristics) as an identical copy of the base table including the business time related columns. After the history table is defined, the base table and history table become the STT portion of the bitemporal table when the statement in Example 2-5 is issued.

Example 2-5 ALTER TABLE statement to start history table capture of table changes

```
ALTER TABLE table_name ADD VERSIONING
   USE HISTORY TABLE history_table_name;
```

One key benefit of DB2 10 STT and bitemporal tables is that the history table inherits its privileges from the base table. Therefore, there is no separate privilege or table access to grant; you basically grant the required application privileges on the base table and the application accesses the history table transparently.

Tip: Because the history tables for system-period temporal and bitemporal tables inherit their access privileges from the base table, a preferred practice is not to explicitly grant privileges on the history tables. This preference is especially true for INSERT, UPDATE, and DELETE privileges, because you do not want external applications outside of the DB2 10 database engine to modify the history tables. External history table modification can adversely affect the data-row-tracking mechanism of STT and bitemporal tables.

Another capability that is provided by Time Travel Query is changing the current time through the special registers, CURRENT TEMPORAL BUSINESS_TIME and CURRENT TEMPORAL SYSTEM_TIME, for a particular database session. With this capability, a database query can return data as of a past or future time, and is especially useful in situations where the SQL statements are part of a packaged application that does not allow statement modification.

2.3.2 Adding the Time Travel Query to a DB2 10 database

The following examples illustrate the advantages of temporal tables and how Great Toque Outdoors uses them to solve business problems.

System-period temporal table examples

For the first set of examples, we describe system-period temporal tables. As the name implies, a system-period temporal table uses system time, where the system-time information is fully and automatically maintained by the Time Travel Query feature of DB2 10.

As Example 2-6 shows, to create the TRAVEL table and make it a system-period temporal table, three columns (named SYS_START, SYS_END, and TX_START) must be defined in the table structure.

Example 2-6 CREATE TABLE statement for TOQUE.TRAVEL_STT table

```
CREATE TABLE TOQUE.TRAVEL_STT (
   TRIP_NAME CHAR(36) NOT NULL PRIMARY KEY,
   DESTINATION CHAR(18) NOT NULL,
   DEPARTURE_DATE DATE NOT NULL,
   PRICE DECIMAL(8,2) NOT NULL,
   SYS_START TIMESTAMP(12) NOT NULL GENERATED ALWAYS
      AS ROW BEGIN
      IMPLICITLY HIDDEN,
   SYS_END TIMESTAMP(12) NOT NULL GENERATED ALWAYS
      AS ROW END
      IMPLICITLY HIDDEN,
   TX_START TIMESTAMP(12) NOT NULL GENERATED ALWAYS
      AS TRANSACTION START ID
      IMPLICITLY HIDDEN,
   PERIOD SYSTEM_TIME (SYS_START, SYS_END))
 IN TOQUE_TRIPS;
```

Note: The three columns associated with STTs can be named with any valid DB2 column name. The important points for the STT definition are as follows:

- Each column must be defined as a TIMESTAMP data type.
- The start time and end time column names must be specified in the PERIOD SYSTEM_TIME(*start_time, end_time*) clause.

Each of these columns is defined to be of type `TIMESTAMP NOT NULL` (any precision is allowed) and they are generated columns (that are automatically maintained by DB2). The IMPLICITLY HIDDEN keyword means that the column is hidden by default and the trip-planning application that uses the TRAVEL table does not have to be aware of these columns; and a `SELECT *` query does not show any columns that are implicitly hidden. As the `CREATE TABLE` statement indicates, the table is identified as a system-period temporal table through the `PERIOD SYSTEM_TIME` keyword.

The next step is to create a history table for the system-period temporal table, as shown in Example 2-7.

Example 2-7 Creating the history table TOQUE.TRAVEL_HISTORY_STT

```
CREATE TABLE TOQUE.TRAVEL_HISTORY_STT LIKE TOQUE.TRAVEL_STT
   IN TOQUE_HISTORY;
[OPTIONAL] ALTER TABLE TOQUE.TRAVEL_HISTORY_STT APPEND ON;
```

The history table must be identical to the base table (TRAVEL in this example) with the same column names and data type definitions. The optional ALTER TABLE statement is to improve performance of row inserts to the history table.

Tip: The easiest and preferred method to ensure the history table matches the base table for a system-period temporal table is to use the following statement:

```
CREATE TABLE hist_table_name LIKE base_table_name ...
   WITH RESTRICT ON DROP
```

Using the LIKE option on CREATE TABLE prevents any discrepancy in the history table definition such as improper column order, misspelled column name, and other similar types of mistakes.

The WITH RESTRICT ON DROP clause prevents the history table from being implicitly dropped when the base table of the temporal table definition is dropped. This clause requires an explicit DROP TABLE statement to be executed against the history table to drop the table. An alternative method, to avoid dropping the history table when the base table is dropped, is to use the ALTER TABLE...DROP VERSIONING statement clause against the base table before the DROP TABLE statement.

For performance reasons, because the history table has new rows appended to the end of the table as rows are updated or deleted from the base table, a preferred practice is to use the following ALTER TABLE statement for history tables:

```
ALTER TABLE hist_table_name APPEND ON
```

To finalize the system-period temporal table process and activate the history collection portion, where all updates and delete activity on the base table cause rows to be written to the history table, the statement shown in Example 2-8 must be issued. Without this ALTER TABLE statement, the history table portion of the STT will not be activated.

Example 2-8   ALTER TABLE statement that activates the STT history collection

```
ALTER TABLE TOQUE.TRAVEL_STT
   ADD VERSIONING USE HISTORY TABLE TOQUE.TRAVEL_HISTORY_STT;
```

Now, we describe how system-period temporal tables maintain data history when data is updated or deleted within a base temporal table. Using the TRAVEL_STT and TRAVEL_HISTORY_STT tables that have already been created, some transactions that update existing data in the TRAVEL_STT table are described. Also, we show the content, before and after, of both the TRAVEL_STT and TRAVEL_HISTORY_STT tables, to illustrate how system-period temporal tables provide history tracking for the TRAVEL_STT table.

Example 2-9 shows the TRAVEL_STT table that contains the trips after the initial load of data, prior to any update operations.

Example 2-9 Initial contents of TRAVEL_STT table

```
TRIP                            COUNTRY          DEPARTURE     PRICE
------------------------------  ----------       ----------    ------
Sights of Sydney                Australia        12/14/2012    3200.00
Great Barrier Reef              Australia        01/04/2013    3400.00
Sound of Music Tour             Austria          06/14/2013    1975.00
Mozarts Birthplace Tour         Austria          06/21/2013    1925.00
Sights of Antwerp               Belgium          08/31/2012    2000.00
Sights of Belgium               Belgium          06/14/2013    1875.00
Sights of Rio De Janeiro        Brazil           01/11/2013    1900.00
Hockey Hall of Fame Tour        Canada           11/16/2012     825.00
Shanghai Shopping               China            09/14/2012    2100.00
The Great Wall Tour             China            06/07/2013    2200.00
Sights of Denmark               Denmark          08/10/2012    1675.00
Copenhagen Shopping             Denmark          06/21/2013    1375.00
Helsinki Tour                   Finland          08/03/2012    1425.00
Sights of Finland               Finland          08/10/2012    1550.00
Paris Museums Tour              France           08/24/2012    1700.00
Eiffel Tower Adventure          France           09/07/2012    2275.00
Neuschwanstein Castle           Germany          07/20/2012    1850.00
Munich Brewery Tour             Germany          07/20/2012    2000.00
Vatican City Tour               Italy            10/19/2012    2300.00
Romance of Venice Tour          Italy            06/28/2013    2550.00
Kyoto Tour                      Japan            04/12/2013    2400.00
Sights of Korea                 Korea            08/31/2012    1900.00
Sights of Seoul                 Korea            05/17/2013    2000.00
Mayan Ruins Adventure           Mexico           09/07/2012     975.00
Museums of Amsterdam            Netherlands      08/03/2012    1600.00
Amsterdam Canal Tour            Netherlands      08/24/2012    1200.00
Sentosa Island Vacation         Singapore        11/23/2012    4200.00
Orchard Road Shopping           Singapore        02/08/2013    3100.00
Don Quixote Adventure           Spain            08/31/2012    2775.00
Running wih the Bulls Tour      Spain            07/05/2013    2900.00
Sights of Sweden                Sweden           08/03/2012    1450.00
Old Stockholm Tour              Sweden           06/21/2013    1450.00
Alps Tour                       Switzerland      02/15/2013    2300.00
Zurich Tour                     Switzerland      06/14/2013    1800.00
Old London Tour                 United Kingdom   08/17/1012    1750.00
British Open 2013               United Kingdom   07/16/2013    3400.00
Grand Canyon Mule Tour          United States    05/31/2013    1000.00
```

At this point in time, the TRAVEL_HISTORY_STT table is empty (0 rows), because the only transactions against the TRAVEL_STT base table were INSERT operations. Because an INSERT operation does not change the existing value of a data row (it adds a new row), no action is taken by the STT table and no history is generated for an INSERT operation. For the initial data load, all of the records in the TRAVEL_STT table have a SYS_START value (date portion of timestamp shown) of 05/16/2012 and a SYS_END value (date value of timestamp shown) of 12/30/9999. The dates are in MM/DD/YYYY format.

The update statements, shown in Example 2-10, are executed against the TRAVEL_STT table on 05/18/2012 (two days after the original data-load) with the before and after results for each update statement. These results are also shown in separate examples for the TRAVEL_STT and TRAVEL_HISTORY_STT tables based on the columns used in the update statements, for readability.

Example 2-10 UPDATE statements executed against the TRAVEL_STT table

```
UPDATE statement #1:
UPDATE TOQUE.TRAVEL_STT SET DEPARTURE_DATE = '06-14-2013'
   WHERE DEPARTURE_DATE = '06-07-2013' AND DESTINATION = 'China';

UPDATE statement #2
UPDATE TOQUE.TRAVEL_STT SET TRIP_NAME = 'Australian Outback Tour'
   WHERE DEPARTURE_DATE = '12-14-2012' AND DESTINATION = 'Australia';

UPDATE statement #3:
UPDATE TOQUE.TRAVEL_STT SET PRICE = 1750.00
   WHERE DEPARTURE_DATE = '07-20-2012' AND DESTINATION = 'Germany';
```

For UPDATE statement #1 (Example 2-11), Table 2-1 on page 19 shows TRAVEL_STT before the update and Table 2-2 on page 19 shows TRAVEL_STT after the update.

Table 2-3 on page 19 shows TRAVEL_HISTORY_STT after the update, because the TRAVEL_HISTORY_STT table had 0 rows prior to the update. Notice that the original “The Great Wall Tour” record now has its SYS_END changed from 12/30/9999 to 05/16/2012 (date of update).

Example 2-11 Update statement for the China trip change

```
UPDATE TOQUE.TRAVEL_STT SET DEPARTURE_DATE = '06-14-2013'
   WHERE DEPARTURE_DATE = '06-07-2013' AND DESTINATION = 'China';
```

Table 2-1 TRAVEL_STT before UPDATE statement #1 for the China trips

Trip Name	Destination	Departure	SYS_START	SYS_END
Shanghai Shopping	China	09/14/2012	05/16/2012	12/30/9999
The Great Wall Tour	China	06/07/2013	05/16/2012	12/30/9999

Table 2-2 TRAVEL_STT after UPDATE statement #1 for the China trips

Trip Name	Destination	Departure	SYS_START	SYS_END
Shanghai Shopping	China	09/14/2012	05/16/2012	12/30/9999
The Great Wall Tour	China	06/14/2013	05/18/2012	12/30/9999

Table 2-3 TRAVEL_HISTORY_STT after UPDATE statement #1 for the China trips

Trip Name	Destination	Departure	SYS_START	SYS_END
The Great Wall Tour	China	06/07/2013	05/16/2012	05/18/2012

Next, we examine UPDATE statement #2 (Example 2-12), which changes the trip name for the 12/14/2012 trip with a destination of Australia. As in UPDATE statement #1, there are no prior history records for trips with a destination of “Australia” prior to UPDATE statement #2 executing. Therefore, we view the before-contents of the TRAVEL_STT table shown in Table 2-4 on page 20, the after-contents of the TRAVEL_STT table shown in Table 2-5 on page 20, and the after-contents of the TRAVEL _HISTORY_STT table shown in Table 2-6 on page 20.

Example 2-12 Update statement for the Australia trip change

```
UPDATE TOQUE.TRAVEL_STT SET TRIP_NAME = 'Australian Outback Tour'
   WHERE DEPARTURE_DATE = '12-14-2012' AND DESTINATION = 'Australia';
```

Table 2-4 TRAVEL_STT before UPDATE statement #2 for the Australia trips

Trip Name	**Destination**	**Departure**	**SYS_START**	**SYS_END**
Sights of Sydney	Australia	12/14/2012	05/16/2012	12/30/9999
Great Barrier Reef	Australia	01/04/2013	05/16/2012	12/30/9999

Table 2-5 TRAVEL_STT after UPDATE statement #2 for the Australia trips

Trip Name	**Destination**	**Departure**	**SYS_START**	**SYS_END**
Australian Outback Tour	Australia	12/14/2012	05/18/2012	12/30/9999
Great Barrier Reef	Australia	01/04/2013	05/16/2012	12/30/9999

Table 2-6 TRAVEL_HISTORY_STT after UPDATE statement #2 for the Australia trips

Trip Name	**Destination**	**Departure**	**SYS_START**	**SYS_END**
Sights of Sydney	Australia	12/14/2012	05/16/2012	05/18/2012

Finally, to conclude the examples using system-period temporal tables, we examine the results of UPDATE statement #3 (Example 2-13). This update lowers the price for the 07/20/2012 trips to Germany to try to generate more interest in this trip. As in the previous update statements, Table 2-7 on page 21 displays the contents of the TRAVEL_STT table before the update statement for the Germany trips, and Table 2-8 on page 21 displays the contents of the TRAVEL_STT table after UPDATE statement #3 for the Germany trips. Finally, Table 2-9 on page 21 shows the contents of the TRAVEL_HISTORY_STT table after UPDATE statement #3 has executed. Because both Germany trips have the same departure date, there are two entries in the history table after this update.

Example 2-13 Update statement for the Germany trip change

```
UPDATE TOQUE.TRAVEL_STT SET PRICE = 1750.00
   WHERE DEPARTURE_DATE = '07-20-2012' AND DESTINATION = 'Germany';
```

Table 2-7 TRAVEL_STT before UPDATE statement #3 for the Germany trips

Trip Name	Destination	Price	SYS_START	SYS_END
Neuschwanstein Castle	Germany	1850.00	05/16/2012	12/30/9999
Munich Brewery Tour	Germany	2000.00	05/16/2012	2/30/9999

Table 2-8 TRAVEL_STT after UPDATE statement #3 for the Germany trips

Trip Name	Destination	Price	SYS_START	SYS_END
Neuschwanstein Castle	Germany	1750.00	05/18/2012	12/30/9999
Munich Brewery Tour	Germany	1750.00	05/18/2012	12/30/9999

Table 2-9 TRAVEL_HISTORY_STT after UPDATE statement #3 for the Germany trips

Trip Name	Destination	Price	SYS_START	SYS_END
Neuschwanstein Castle	Germany	1850.00	05/16/2012	05/18/2012
Munich Brewery Tour	Germany	2000.00	05/16/2012	05/18/2012

Bitemporal temporal table examples

We examined STT tables; now we look at bitemporal tables, which are a type of temporal tables that Great Toque Outdoors will use for its application. Bitemporal tables can satisfy business-time-related queries (such as the valid booking dates for a particular trip) and track table changes over time for data tracking purposes.

For this set of examples, we use the same basic table structures as those used in the STT discussion, however now both the TRAVEL_BTT table and TRAVEL_HISTORY_BTT table have two additional columns to track the business time component that Great Toque Outdoors requires. This business time component consists of the valid booking start date and the valid booking end date (eligible date range in which a particular trip can be booked). Great Toque Outdoors policy is that the valid booking end date is always one day prior to the trip-departure date. Example 2-14 on page 22 shows the syntax for the creation and configuration of the TRAVEL_BTT table as a bitemporal table in DB2 10.

Example 2-14 Syntax for TRAVEL_BTT creation and setup including history

```
CREATE TABLE TOQUE.TRAVEL_BTT (
   TRIP_NAME CHAR(36) NOT NULL,
   DESTINATION CHAR(18) NOT NULL,
   DEPARTURE_DATE DATE NOT NULL,
   PRICE DECIMAL(8,2) NOT NULL,
   VALID_BOOKING_START DATE NOT NULL,
   VALID_BOOKING_END DATE NOT NULL,
   SYS_START TIMESTAMP(12) NOT NULL GENERATED ALWAYS
         AS ROW BEGIN IMPLICITLY HIDDEN,
   SYS_END TIMESTAMP(12) NOT NULL GENERATED ALWAYS
         AS ROW END IMPLICITLY HIDDEN,
   TX_START TIMESTAMP(12) NOT NULL GENERATED ALWAYS
         AS TRANSACTION START ID IMPLICITLY HIDDEN,
   PERIOD SYSTEM_TIME (SYS_START, SYS_END),
   PERIOD BUSINESS_TIME (VALID_BOOKING_START, VALID_BOOKING_END),
   PRIMARY KEY (TRIP_NAME, BUSINESS_TIME WITHOUT OVERLAPS))
IN TOQUE_TRIPS;

CREATE TABLE TOQUE.TRAVEL_HISTORY_BTT LIKE TOQUE.TRAVEL_BTT
         IN TOQUE_HISTORY;

ALTER TABLE TOQUE.TRAVEL_BTT
   ADD VERSIONING USE HISTORY TABLE TOQUE.TRAVEL_HISTORY_BTT;
```

Notice that, in Example 2-14, the key column definitions and keyword options to configure a bitemporal table are in bold type.

For the bitemporal table example, we display the valid booking start date and valid booking end date along with the trip name and destination in Example 2-15 on page 23. This example shows the initial data values in the TRAVEL_BTT table. To see the other columns not displayed, refer to the initial TRAVEL_STT table contents shown in Example 2-9 on page 17. The VALID_BOOKING_START and VALID_BOOKING_END columns are displayed as BOOK_ST and BOOK_END.

Example 2-15 Initial contents of TRAVEL_BTT table

```
TRIP                            COUNTRY          BOOK_ST     BOOK_END
------------------------------  ----------       ----------  ---------
Sights of Sydney                Australia        07/01/2012 12/13/2012
Great Barrier Reef              Australia        07/01/2012 01/03/2013
Sound of Music Tour             Austria          10/01/2012 06/13/2013
Mozarts Birthplace Tour         Austria          10/01/2012 06/20/2013
Sights of Antwerp               Belgium          07/01/2012 08/30/2012
Sights of Belgium               Belgium          10/01/2012 06/13/2013
Sights of Rio De Janeiro        Brazil           07/01/2012 01/10/2013
Hockey Hall of Fame Tour        Canada           07/01/2012 11/15/2012
Shanghai Shopping               China            07/01/2012 09/13/2012
The Great Wall Tour             China            07/01/2012 06/06/2013
Sights of Denmark               Denmark          07/01/2012 08/09/2012
Copenhagen Shopping             Denmark          07/01/2012 06/20/2013
Helsinki Tour                   Finland          07/01/2012 08/02/2012
Sights of Finland               Finland          07/01/2012 08/09/2012
Paris Museums Tour              France           07/01/2012 08/23/2012
Eiffel Tower Adventure          France           07/01/2012 09/06/2012
Neuschwanstein Castle           Germany          07/01/2012 07/19/2012
Munich Brewery Tour             Germany          07/01/2012 07/19/2012
Vatican City Tour               Italy            07/01/2012 10/18/2012
Romance of Venice Tour          Italy            10/01/2012 06/27/2013
Kyoto Tour                      Japan            10/01/2012 04/11/2013
Sights of Korea                 Korea            07/01/2012 08/30/2012
Sights of Seoul                 Korea            10/01/2012 05/16/2013
Mayan Ruins Adventure           Mexico           07/01/2012 09/06/2012
Museums of Amsterdam            Netherlands      07/01/2012 08/02/2012
Amsterdam Canal Tour            Netherlands      07/01/2012 08/23/2012
Sentosa Island Vacation         Singapore        07/01/2012 11/22/2012
Orchard Road Shopping           Singapore        07/01/2012 02/07/2013
Don Quixote Adventure           Spain            07/01/2012 08/30/2012
Running wih the Bulls Tour      Spain            10/01/2012 07/04/2013
Sights of Sweden                Sweden           07/01/2012 08/02/2012
Old Stockholm Tour              Sweden           10/01/2012 06/20/2013
Alps Tour                       Switzerland      07/01/2012 02/14/2013
Zurich Tour                     Switzerland      10/01/2012 06/13/2013
Old London Tour                 United Kingdom   07/01/2012 08/16/2012
British Open 2013               United Kingdom   10/16/2013 07/15/2013
Grand Canyon Mule Tour          United States    10/01/2012 05/30/2013
```

First, we examine the results of several update statements against the TRAVEL_BTT table. The current date is 05/18/2012 and some updates are now executed on the TRAVEL_BTT table. These update statements are shown in Example 2-16. For each UPDATE statement, the contents, before and after, of the TRAVEL_BTT table are displayed along with the after-contents of the TRAVEL_HISTORY_BTT table. Because they are the first updates after the initial data load in the TRAVEL_BTT table, the TRAVEL_HISTORY_BTT table is empty and contains no prior values for the trips involved in the UPDATE statements.

Example 2-16 UPDATE statements executed against the TRAVEL_BTT table

```
UPDATE statement #1:
UPDATE TOQUE.TRAVEL_BTT SET DEPARTURE_DATE = '05-17-2013',
   VALID_BOOKING_END = '05-16-2013'
   WHERE DEPARTURE_DATE = '04/12/2013' AND DESTINATION = 'Japan';

UPDATE statement #2:
UPDATE TOQUE.TRAVEL_BTT SET TRIP_NAME = 'Australian Outback Tour'
   WHERE DEPARTURE_DATE = '12-14-2012' AND DESTINATION = 'Australia';

UPDATE statement #3:
UPDATE TOQUE.TRAVEL_BTT FOR PORTION OF BUSINESS_TIME
   FROM '10/01/2012' TO '12/01/2012' SET PRICE = 3200.00
   WHERE TRIP_NAME = 'British Open 2013';
```

For UPDATE statement #1 (Example 2-17 on page 25), Table 2-10 on page 25 shows TRAVEL_BTT before the update and Table 2-11 on page 25 shows TRAVEL_BTT after the update. Notice that the booking start date and the departure date were both updated as result of the statement execution. The SYS_START timestamp was changed automatically by DB2 to reflect the timestamp of the row update (SYS_START). Because this is the current value of the row, the SYS_END value is 12/30/9999 (date portion of timestamp).

Table 2-12 on page 25 shows TRAVEL_HISTORY_BTT after UPDATE statement #1, because the TRAVEL_HISTORY_BTT table had 0 rows before the update. Notice that the original “Kyoto Tour” record now has its SYS_END changed from 12/30/9999 to 05/16/2012 (date of update).

Tip: For the BUSINESS_TIME columns of an application-period temporal or bitemporal table, the application is responsible for updating the business start and business end time columns (VALID_BOOKING_START and VALID_BOOKING_END in these examples). They are displayed as Booking Start and Booking End in the contents tables.

Example 2-17 Update statement for the Japan trip change

```
UPDATE TOQUE.TRAVEL_BTT SET DEPARTURE_DATE = '05-17-2013',
   VALID_BOOKING_END = '05-16-2013'
   WHERE DEPARTURE_DATE = '04/12/2013' AND DESTINATION = 'Japan';
```

Table 2-10 TRAVEL_BTT before UPDATE statement #1 for the Japan trip

Trip Name	Destination	Departure	Booking Start	Booking End	SYS_START	SYS_END
Kyoto Tour	Japan	04/12/2013	10/01/2012	04/11/2013	05/16/2012	12/30/9999

Table 2-11 TRAVEL_BTT after UPDATE statement #1 for the Japan trip

Trip Name	Destination	Departure	Booking Start	Booking End	SYS_START	SYS_END
Kyoto Tour	Japan	05/17/2013	10/01/2012	05/16/2013	05/18/2012	12/30/9999

Table 2-12 TRAVEL_HISTORY_BTT after UPDATE statement #1 for the Japan trip

Trip Name	Destination	Departure	Booking Start	Booking End	SYS_START	SYS_END
Kyoto Tour	Japan	04/12/2013	10/01/2012	04/11/2013	05/16/2012	05/18/2012

UPDATE statement #2 (Example 2-18) is similar to an update against a system-period temporal table, because it is not modifying any column that is associated with the BUSINESS_TIME component of the TRAVEL_BTT bitemporal table. For the TRAVEL_BTT table, the contents before the update (Table 2-13 on page 26) and the contents after the update (Table 2-14 on page 26) have no change to the Booking Start and Booking End values for the first Australian trip, whose name was changed. For this particular update, the changes to note are the SYS_START and SYS_END column values in both the TRAVEL_BTT after-contents and the TRAVEL_HISTORY_BTT after-contents, shown in Table 2-15 on page 26.

Tip: For the SYSTEM_TIME component of a temporal table, a system-end time (SYS_END in these examples) value of `12/30/99999:00:00.00` indicates the currently active row for a specific primary key value.

Example 2-18 Update statement for the Australia trip change

```
UPDATE TOQUE.TRAVEL_BTT SET TRIP_NAME = 'Australian Outback Tour'
   WHERE DEPARTURE_DATE = '12-14-2012' AND DESTINATION = 'Australia';
```

Table 2-13 TRAVEL_BTT before UPDATE statement #2 for the Australia trips

Trip Name	Destination	Departure	Booking Start	Booking End	SYS_START	SYS_END
Sights of Sydney	Australia	12/14/2012	07/01/2012	12/13/2012	05/16/2012	12/30/9999
Great Barrier Reef	Australia	01/04/2013	07/01/2012	01/03/2013	05/16/2012	12/30/9999

Table 2-14 TRAVEL_BTT after UPDATE statement #2 for the Australia trips

Trip Name	Destination	Departure	Booking Start	Booking End	SYS_START	SYS_END
Australian Outback Tour	Australia	12/14/2012	07/01/2012	12/13/2012	05/18/2012	12/30/9999
Great Barrier Reef	Australia	01/04/2013	07/01/2012	01/03/2013	05/16/2012	12/30/9999

Table 2-15 TRAVEL_HISTORY_BTT after UPDATE statement #2 for the Australia trips

Trip Name	Destination	Departure	Booking Start	Booking End	SYS_START	SYS_END
Sights of Sydney	Australia	12/14/2012	07/01/2012	12/13/2012	05/16/2012	05/18/2012

Finally, UPDATE statement #3 (Example 2-19 on page 27) is modifying the price of the “British Open 2013” trip and reducing it by $200.00 for the months of October and November 2012. This special pricing is to promote early bookings of the trip after it becomes active for booking on 10/01/2012. Because this update affects BUSINESS_TIME directly, we examine the same three results as the previous update statements.

The contents before the update of the trip row for TRAVEL_BTT are shown in Table 2-16 on page 27 and the contents after the update of the trip row for TRAVEL_BTT are shown in Table 2-17 on page 27. Table 2-18 on page 28 shows the SYSTEM_TIME changes in both the TRAVEL_BTT after-update contents and the TRAVEL_HISTORY_BTT after-update contents.

The after-update contents show two rows for the “British Open 2013” trip. The reason is because of the BUSINESS_TIME portion of the bitemporal table. Because the reduction of the trip price was only for the first two months of the booking period, DB2 automatically split “British Open 2013” trip into two rows:

- First row (with the reduced price) is valid from 10/01/2012 until 12/01/2012.
- Second row (with the original price) is valid from 12/01/2012 until 07/15/2013.

The SYSTEM_TIME is updated accordingly in the TRAVEL_BTT table:

- SYS_START timestamp is set to 05/18/2012.
- SYS_END timestamp in the TRAVEL_HISTORY_BTT table is set to 05/18/2012 (data value of timestamp only being displayed).

Tip: The BUSINESS_TIME period adjustment shown for the “British Open 2013” trip entry highlights the advantage of using DB2 10 Time Travel Query for business time management over a custom application solution. DB2 does the complex work of splitting time periods into multiple segments as required by the table updates.

Example 2-19 Update statement for the British Open 2013 trip change

```
UPDATE TOQUE.TRAVEL_BTT FOR PORTION OF BUSINESS_TIME
   FROM '10/01/2012' TO '12/01/2012' SET PRICE = 3200.00
   WHERE TRIP_NAME = 'British Open 2013';
```

Table 2-16 TRAVEL_BTT before UPDATE statement #3 for the British Open 2013 trip

Trip Name	**Destination**	**Departure**	**Price**	**Booking Start**	**Booking End**	**SYS_START**	**SYS_END**
British Open 2013	United Kingdom	07/16/2013	3400.00	10/01/2012	07/15/2013	05/16/2012	12/30/999

Table 2-17 TRAVEL_BTT after UPDATE statement #3 for the British Open 2013 trip

Trip Name	**Destination**	**Departure**	**Price**	**Booking Start**	**Booking End**	**SYS_START**	**SYS_END**
British Open 2013	United Kingdom	07/16/2013	3200.00	10/01/2012	12/01/2012	05/18/2012	12/30/999
British Open 2013	United Kingdom	07/16/2013	3400.00	12/01/2012	07/15/2013	05/18/2012	12/30/999

Table 2-18 TRAVEL_HISTORY_BTT after UPDATE statement #3 for the British Open 2013 trip

Trip Name	Destination	Departure	Price	Booking Start	Booking End	SYS_START	SYS_END
British Open 2013	United Kingdom	07/16/2013	3400.00	10/01/2012	07/15/2013	05/16/2012	05/18/2012

To continue with bitemporal tables and their behavior, we look at several delete operations against the TRAVEL_BTT table; we view the results of delete operations by looking at the TRAVEL_BTT and TRAVEL_HISTORY_BTT tables.

It is now 07/23/2012 and Great Toque Outdoors needs to delete some rows from the TRAVEL _BTT table for the following reasons:

- The Finland trips both depart in August 2012 but have no sales. Therefore, both trips must be cancelled to minimize the loss to Great Toque Outdoors in terms of fees that are due within one week of the departure date.
- The trip planning for the "British Open 2013" trip in terms of number of participants and bookings must be in place by 04/01/2013. Therefore the valid booking end date must be rescheduled to 03/30/2013 to accommodate the final planning changes.

The DELETE statements to be executed against the TRAVEL_BTT table are shown in Example 2-20.

Example 2-20 DELETE statements executed against the TRAVEL_BTT table

```
DELETE statement #1:
DELETE FROM TOQUE.TRAVEL_BTT WHERE DESTINATION = 'Finland';

DELETE statement #2:
DELETE FROM TOQUE.TRAVEL_BTT FOR PORTION OF BUSINESS_TIME
   FROM '04/01/2013' TO '07/30/2013'
   WHERE TRIP_NAME = 'British Open 2013';
```

As we did in previous examples, we examine the contents of both the TRAVEL_BTT and TRAVEL_HISTORY_BTT tables for the Finland trips. The TRAVEL_BTT table contents will be provided for before and after DELETE statement #1; the before-contents is shown in Table 2-19 on page 29 and the after-contents is in Table 2-20 on page 29. The TRAVEL_HISTORY_BTT table after-delete contents are shown in Table 2-21 on page 29. Because there were no changes to Finland destination trips before DELETE statement #1 is executed, the TRAVEL_HISTORY_BTT table contents before the deletion is not shown.

This DELETE statement #1 (Example 2-21 on page 29) in particular illustrates a key reason that Great Toque Outdoors chose a bitemporal table rather than an application-period temporal table (ATT) alone. If you notice the TRAVEL_BTT table contents after DELETE statement #1 for the Finland trips, there are no rows for Finland trips. Because ATTs do not have history tables associated with them, the data rows that are deleted are permanently removed from the table. Because the bitemporal table includes the system-period component and includes a history table to track all data changes (both updates and deletes), the deleted Finland trip data is preserved in the TRAVEL_HISTORY_BTT table.

Example 2-21 Delete statement to remove Finland trips

```
DELETE FROM TOQUE.TRAVEL_BTT WHERE DESTINATION = 'Finland';
```

Table 2-19 TRAVEL_BTT before DELETE statement #1 for the Finland trips

Trip Name	Departure	Price	Booking Start	Booking End	SYS_START	SYS_END
Helsinki Tour	08/03/2012	1425.00	07/01/2012	08/02/2012	05/16/2012	12/30/9999
Sights of Finland	08/10/2012	1550.00	07/01/2012	08/09/2012	05/16//2012	12/30/9999

Table 2-20 TRAVEL_BTT after DELETE statement #1 for the Finland trips

Trip Name	Departure	Price	Booking Start	Booking End	SYS_START	SYS_END
no rows returned						

Table 2-21 TRAVEL_HISTORY_BTT after DELETE statement #1 for the Finland trips

Trip Name	Departure	Price	Booking Start	Booking End	SYS_START	SYS_END
Helsinki Tour	08/03/2012	1425.00	07/01/2012	08/02/2012	05/16/2012	07/23/2012
Sights of Finland	08/10/2012	1550.00	07/01/2012	08/09/2012	05/16/2012	07/23/2012

Now, we examine the results of DELETE statement #2 (Example 2-22 on page 30) where the valid booking end date for the “British Open 2013” trip booking period is changed from 11/01/2012 to 03/30/2013. This example shows how delete operations affect the BUSINESS_TIME component of a bitemporal table. Because the “British Open 2013” trip had prior changes, history data is available for this trip before the execution of DELETE statement #2. Therefore, for the results of DELETE statement #2, we display the contents of both the

TRAVEL_BTT and TRAVEL_HISTORY_BTT tables before and after the delete operation. The TRAVEL_BTT table before-contents are shown in and the after-contents are shown in Table 2-23. The TRAVEL_HISTORY_BTT table before-contents are shown in Table 2-24 and the after-contents are shown in Table 2-25.

DELETE statement #2 changed the booking end date to 04/01/2013 for the second row of the "British Open 2013" trip, where the price was 3400.00. The SYSTEM_TIME component captures the actual date of the change in the SYS_END column in the TRAVEL_HISTORY_BTT table.

Example 2-22 Delete statement for the British Open 2013 trip change

```
DELETE FROM TOQUE.TRAVEL_BTT FOR PORTION OF BUSINESS_TIME
   FROM '04/01/2013' TO '07/30/2013'
   WHERE TRIP_NAME = 'British Open 2013';
```

Table 2-22 TRAVEL_BTT before DELETE #2 for the British Open 2013 trip

Trip Name	Departure	Price	Booking Start	Booking End	SYS_START	SYS_END
British Open 2013	07/16/2013	3200.00	10/01/2012	12/01/2012	05/18/2012	12/30/9999
British Open 2013	07/16/2013	3400.00	12/01/2012	07/15/2013	05/18/2012	12/30/9999

Table 2-23 TRAVEL_BTT after DELETE #2 for the British Open 2013 trip

Trip Name	Departure	Price	Booking Start	Booking End	SYS_START	SYS_END
British Open 2013	07/16/2013	3200.00	10/01/2012	12/01/2012	05/18/2012	12/30/9999
British Open 2013	07/16/2013	3400.00	12/01/2012	04/01/2013	07/23/2012	12/30/9999

Table 2-24 TRAVEL_HISTORY_BTT before DELETE #2 for the British Open 2013 trip

Trip Name	Departure	Price	Booking Start	Booking End	SYS_START	SYS_END
British Open 2013	07/16/2013	3400.00	10/01/2012	07/15/2013	05/16/2012	05/18/2012

Table 2-25 TRAVEL_HISTORY_BTT after DELETE #2 for the British Open 2013 trip

Trip Name	Departure	Price	Booking Start	Booking End	SYS_START	SYS_END
British Open 2013	07/16/2013	3400.00	10/01/2012	07/15/2013	05/16/2012	05/18/2012
British Open 2013	07/16/2013	3400.00	12/01/2012	07/15/2013	05/18/2012	07/23/2012

Tip: In the example description for DELETE statement #2, the VALID_BOOKING_END (displayed as Booking End in the example contents tables) for the “British Open 2013” needed to be changed to 03/30/2013. However, when you examine the TRAVEL_STT table after the delete operation, the second “British Open 2013” row has the Booking End set to 04/01/2013. The reason is that the time periods within a temporal table always use “inclusive-exclusive” for the start and end of a period for both SYSTEM_TIME and BUSINESS_TIME. *Inclusive-exclusive* means that the start of a period includes the start date that is specified (inclusive) and the period ends before the end time that is specified. So the Booking End date and SYS_END timestamp are not included in the period (exclusive). Because Booking End is a date and is excluded from the period, the last date prior to Booking End (04/01/2013) is 03/30/2013.

We examined the behavior of update and delete operations within bitemporal tables. Now, we execute several example SELECT statements to further illustrate the benefits of using SYSTEM_TIME and BUSINESS_TIME for daily queries against a table. See Example 2-23.

Example 2-23 Example queries against the TRAVEL_BTT bitemporal table

```
SELECT statement #1
SELECT STRIP(TRIP_NAME,TRAILING) AS TRIP, DESTINATION, DEPARTURE_DATE, PRICE,
DATE(SYS_START) AS SYSSTART, DATE(SYS_END) AS SYSEND, VALID_BOOKING_START AS
BUSSTART, VALID_BOOKING_END AS BUSEND
FROM TOQUE.TRAVEL_BTT
   FOR BUSINESS_TIME
   FROM '09-01-2012' TO '12-31-2012'
   ORDER BY DESTINATION, DEPARTURE_DATE;

SELECT statement #2:
SELECT STRIP(TRIP_NAME,TRAILING) AS TRIP, DESTINATION, DEPARTURE_DATE, PRICE,
DATE(SYS_START) AS SYS_START, DATE(SYS_END) AS SYS_END, VALID_BOOKING_START AS
BUSSTART, VALID_BOOKING_END AS BUSEND
FROM TOQUE.TRAVEL_BTT
   FOR SYSTEM_TIME
   FROM CURRENT TIMESTAMP - 6 MONTHS TO CURRENT TIMESTAMP
   ORDER BY DESTINATION,DEPARTURE_DATE;

SELECT statement #3:
SELECT STRIP(TRIP_NAME,TRAILING) AS TRIP, DESTINATION, DEPARTURE_DATE, PRICE,
DATE(SYS_START) AS SYSSTART, DATE(SYS_END) AS SYSEND, VALID_BOOKING_START AS
BUSSTART, VALID_BOOKING_END AS BUSEND FROM TOQUE.TRAVEL_BTT
   FOR BUSINESS_TIME
   AS OF '08-01-2012'
   ORDER BY DESTINATION, DEPARTURE_DATE;
```

Results for SELECT statement #1 are shown in Example 2-24 and provide all of the trips in the TRAVEL_BTT table that are valid for booking, between the dates of 09/01/2012 and 12/31/2012, using BUSINESS_TIME. To satisfy this query, the trip must have its valid booking period contain this date range. Twenty-five of the rows contained in the TRAVEL_BTT table meet this criteria.

Example 2-24 SELECT statement #1 results from the TRAVEL_BTT table

```
SELECT STRIP(TRIP_NAME,TRAILING) AS TRIP, DESTINATION, DEPARTURE_DATE, PRICE,
DATE(SYS_START) AS SYSSTART, DATE(SYS_END) AS SYSEND, VALID_BOOKING_START AS
BUSSTART, VALID_BOOKING_END AS BUSEND
FROM TOQUE.TRAVEL_BTT
   FOR BUSINESS_TIME
   FROM '09-01-2012' TO '12-31-2012'
   ORDER BY DESTINATION, DEPARTURE_DATE;

TRIP                       DESTINATION           BUSSTART     BUSEND
----------                 ----------------      ---------    --------
Australian Outback Tour    Australia             07/01/2012 12/13/2012
Great Barrier Reef         Australia             07/01/2012 01/03/2013
Sound of Music Tour        Austria               10/01/2012 06/13/2013
++++++++++ 19 rows omitted +++++++++++
British Open 2013          United Kingdom        12/01/2012 04/01/2013
British Open 2013          United Kingdom        10/01/2012 12/01/2012
Grand Canyon Mule Tour     United States         10/01/2012 05/30/2013
```

Results for SELECT statement #2 (Example 2-25 on page 33) illustrate that a query does not have to use fixed dates for Time Travel Query. In this example, the query is looking for results from six months before the current date until the current date based on the SYSTEM_TIME of the rows. Because this query uses SYSTEM_TIME, the TRAVEL_HISTORY_BTT table is also transparently accessed and the result is all rows in both the TRAVEL_BTT and TRAVEL_HISTORY_BTT tables (44 rows) are returned.

Tip: When viewing query results, you can determine whether the data row is from the base table or the history table by the SYS_END column value. If the timestamp is 12/30/9999..., the data row is the current row in the base table. Any other timestamp in the SYS_END column indicates that the row was retrieved from the history table.

The DB2 development team chose 12/30/9999 instead of the 12/31/9999 to ensure that any time zone manipulations that occur on timestamp and date values do not potentially roll over to the next century, but still represent the end of a given date format’s time.

Example 2-25 SELECT statement #2 results from the TRAVEL_BTT table

```
SELECT STRIP(TRIP_NAME,TRAILING) AS TRIP, DESTINATION, DEPARTURE_DATE, PRICE,
DATE(SYS_START) AS SYS_START, DATE(SYS_END) AS SYS_END, VALID_BOOKING_START AS
BUSSTART, VALID_BOOKING_END AS BUSEND
FROM TOQUE.TRAVEL_BTT
   FOR SYSTEM_TIME
   FROM CURRENT TIMESTAMP - 6 MONTHS TO CURRENT TIMESTAMP
   ORDER BY DESTINATION,DEPARTURE_DATE;

TRIP                        DESTINATION           SYS_START    SYS_END
----------                  ----------------      ---------    --------
Sights of Sydney            Australia             05/16/2012 05/18/2012
Australian Outback Tour     Australia             05/18/2012 12/30/9999
Great Barrier Reef          Australia             05/16/2012 12/30/9999 Sound
of Music Tour               Austria               05/16/2012 12/30/9999
Mozarts Birthplace Tour     Austria               05/16/2012 12/30/9999
Sights of Antwerp           Belgium               05/18/2012 12/30/9999
Sights of Antwerp           Belgium               05/16/2012 05/18/2012
Sights of Belgium           Belgium               05/16/2012 12/30/9999
++++++++++++++++ 36 rows omitted ++++++++++++++++
```

SELECT statement #3 shows the data in the TRAVEL_BTT table as of 08/01/2012. Therefore, any trip that is valid for booking on this date is returned as a query result. Results for SELECT statement #3 are shown in Example 2-26. There are 22 rows (out of 36 rows) returned for this query.

Example 2-26 SELECT statement #3 results from the TRAVEL_BTT table

```
SELECT STRIP(TRIP_NAME,TRAILING) AS TRIP, DESTINATION, DEPARTURE_DATE, PRICE,
DATE(SYS_START) AS SYSSTART, DATE(SYS_END) AS SYSEND, VALID_BOOKING_START AS
BUSSTART, VALID_BOOKING_END AS BUSEND FROM TOQUE.TRAVEL_BTT
   FOR BUSINESS_TIME
   AS OF '08-01-2012'
   ORDER BY DESTINATION, DEPARTURE_DATE;

TRIP                        DESTINATION           BUSSTART     BUSEND
----------                  ----------------      ---------    --------
Australian Outback Tour     Australia0            07/01/2012 12/13/2012
Great Barrier Reef          Australia             07/01/2012 01/03/2013
Sights of Antwerp           Belgium               07/01/2012 08/23/2012
Sights of Rio De Janeiro    Brazil                07/01/2012 01/10/2013
Hockey Hall of Fame Tour    Canada                07/01/2012 11/15/2012
Shanghai Shopping           China                 07/01/2012 09/13/2012
The Great Wall Tour         China                 07/01/2012 06/13/2013
++++++++++++++++++ 15 rows omitted +++++++++++++++++
```

2.4 Summary

DB2 10 Time Travel Query uses temporal tables to provide the mechanism to track either business time, system time, or both. Using system time and business time together in a bitemporal table enables time to be used for business logic requirements and to provide for data auditing to identify data changes over time. This combination provides an organization with flexible data analysis over past, present, and future time periods.

Time Travel Query provides special registers, CURRENT TEMPORAL BUSINESS_TIME and CURRENT TEMPORAL SYSTEM_TIME, that can be used to change the current time for a particular query session. This feature allows database queries as of a point-in-time in the past or future, even if the packaged application does not allow its SQL statement syntax to be modified.

Finally, to conclude the discussion of DB2 temporal tables, converting existing tables (for databases being upgraded from an earlier version of DB2 to DB2 10) into temporal tables can be easy to do. For business time, basically add the two new time columns (as either DATE or TIMESTAMP) to define the business start and business end periods, and define the BUSINESS_TIME period within the table. For system time, the process requires adding three new columns for system start, system end, and transaction ID (all defined as TIMESTAMP) and defining the SYSTEM_TIME period within the table. For system time, the history table must also be created and the linkage between the base table and history table established with the `ALTER TABLE... ADD VERSIONING` statement.

3

Adaptive compression

In this chapter, we describe compression enhancements that are available in DB2 10 and how you can achieve greater compression ratios than in previous versions of DB2, through compression technology that adapts to changing data patterns. After they are established, these extremely high compression ratios can be maintained *without* the need for classic table reorganizations.

3.1 Overview of adaptive compression

Adaptive compression continues the storage optimization work that began in DB2 9.1 to reduce database storage requirements, reduce the duration of maintenance tasks, and reduce the overall amount of storage required by DB2.

In previous versions of DB2, the data compression method (now known as *classic compression*) was based on a static compression dictionary that would identify repeating data patterns across an entire database table. Classic compression provided excellent storage savings across a variety of applications and industries. To retain the same storage savings, however, periodic classic table reorganizations needed to be performed.

DB2 10 introduces *adaptive compression*. Adaptive compression combines classic compression (at the *table-level*) with a new dynamic compression (at the *page-level*) that adapts over time as the data changes within a database table. With adaptive compression, you can achieve and retain storage savings beyond that of classic compression without taking your data offline to perform table reorganizations.

The result is an industry-leading compression solution, which requires less administration and whose storage savings remain high over time.

3.2 Business value

Great Toque Outdoors has benefitted from classic compression and the excellent compression ratios it yields. However, a table's compression dictionary is static under classic compression; after they are created, the compression dictionary does not change. As more rows are inserted or updated into Great Toque Outdoors' database tables, rows that contain recurring patterns will not be compressed if those patterns are not already defined in the compression dictionary. Consider the following examples:

- After Great Toque Outdoors expanded its business to include an adventure travel division, character strings for locales previously undefined in a table-level compression dictionary are being inserted uncompressed into tables.
- Constantly evolving inventory means new items inserted into a table will not be defined in the static compression dictionary, so recurring patterns for those items will not be compressed.

Data drift is the term used to describe the phenomenon of new data being inserted and updated without compression because the compression dictionary no longer fully represents subsequent recurring patterns.

Data drift causes the compression rate for classic compression to degrade slightly over time. The database administrators (DBAs) at Great Toque Outdoors attempt to restore higher compression ratios once a month by issuing a `REORG TABLE...RESETDICTIONARY` command. This command rebuilds the static table-level compression dictionary. However, this table reorganization requires the table to be offline. Although the DBAs have a planned database outage each month for such offline administration tasks, Great Toque Outdoors is a 24x7 international enterprise that requires its data to be highly available.

With adaptive compression, in addition to the static table-level compression dictionary, separate compression dictionaries are created at the page-level. For example, one table that spans 5000 pages has one static table-level compression dictionary and 5000 page-level compression dictionaries. Furthermore, the page-level compression dictionaries are *dynamic*. Unlike a static compression dictionary, a page-level compression dictionary is automatically and transparently rebuilt as a data page fills up. The reclaimed storage space frees up a significant amount of space on the page, so new rows can be inserted on the current page rather than starting a new page. The maintenance of optimum compression ratios is fully automatic, is transparent to the user, and requires no database administration.

Dynamically maintained page-level compression dictionaries are an excellent mechanism for addressing data drift. New and changed data that exhibit recurring patterns cannot be compressed by classic compression if those patterns are not already defined in the static table-level compression dictionary. However, with adaptive compression, page-level compression dictionaries get rebuilt automatically and transparently, such that new recurring patterns can be detected in inserted and changed data, and the data can be compressed.

In short, adaptive compression features both table-level compression (which stores globally recurring patterns) and page-level compression (which stores locally recurring patterns). These two mechanisms do more than complement each other. For example, if a table-level repeating pattern symbol exists as part of a repeating data pattern at the page-level, DB2 page-level compression will compress the data pattern that contains table-level symbols even further.

Together, table-level compression and page-level compression yield incredibly high compression ratios, which remain much more constant over time, compared to classic compression alone, as illustrated in Figure 3-1.

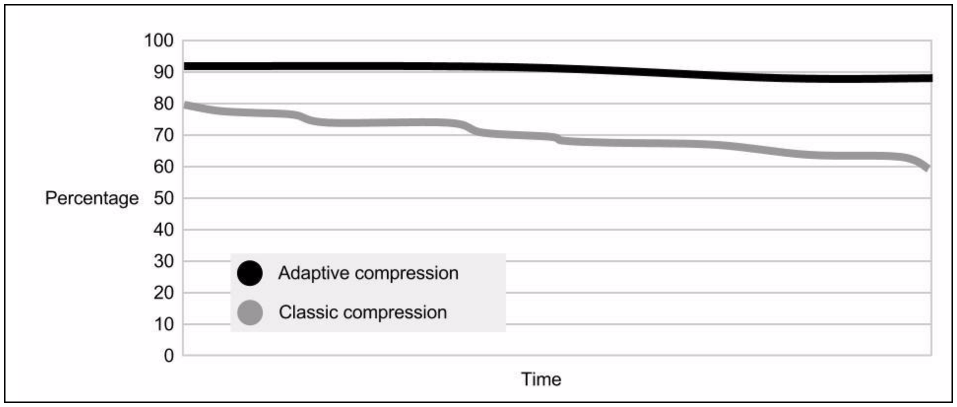


Figure 3-1 Percentage of storage space saved over time

The stable compression ratio exhibited by adaptive compression can also be illustrated by looking at the growth rate of a database table over time. In Figure 3-2, an uncompressed table grows at a linear rate. With classic compression, the table grows at a slower rate. However, with adaptive compression, the same table grows at the slowest rate.

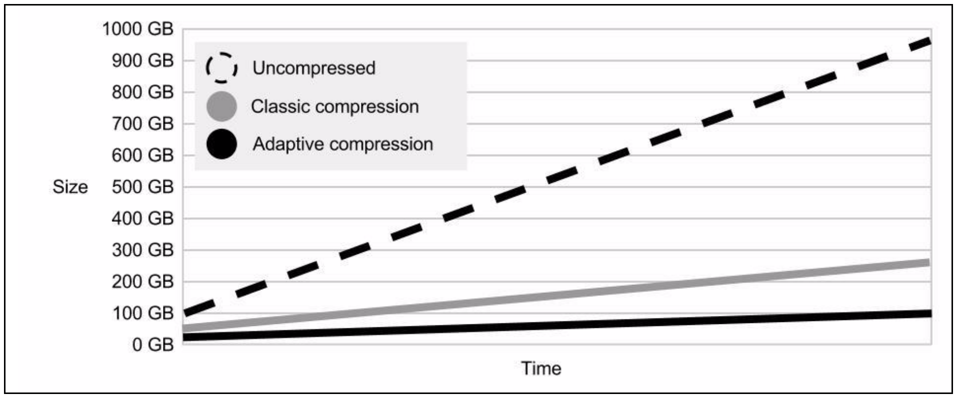


Figure 3-2 Table sizes over time

For Great Toque Outdoors, adaptive compression translates directly into a database that is highly available. In the past, DBAs performed monthly table reorganizations, rendering the tables unavailable for the duration of the reorganization. With adaptive compression, the compression ratios remain high without the need for classic table reorganization.

Great Toque Outdoors has a fixed schedule for hardware expenditure as its database grows. The increased compression ratios that adaptive compression yields lead directly to lowered storage costs, because its database requires less space. The result is less frequent hardware expenditure for storage.

Adaptive compression can also improve query performance, especially for applications that are I/O-bound (in contrast to processor-bound). An example of such data is likely found in a data warehouse, where decision-support workloads are composed of complex analytic queries that perform massive aggregations.

In such cases, row access is mostly sequential and less random. Typically, the data is locally clustered after batch-inserts of data that has already been sorted. Fewer physical I/O operations are required for reading the data in a compressed table, because the same number of rows is stored on fewer pages. Because there are more rows of data on a buffer pool page, the likelihood of required rows being in the buffer pool increases. For example, with data compressed at 50%, the same buffer pool memory holds twice as much data.

Fewer I/O operations also lead to a decrease in processor usage, because I/O access drives up processor usage in a server. Although some extra processing is required when fetched data requires uncompressing, for I/O-bound workloads, the processing savings with compression can more than offset the processing cost of compressing and uncompressing data.

Because adaptive compression can lead to increased query performance, further hardware expenditure savings can be realized with fewer frequent server upgrades.

Reducing hardware costs while increasing the availability of its database appeals to Great Toque Outdoors. Its DBAs prepare to adopt the adaptive compression feature, after doing some compression estimation tasks.

3.3 Business application examples

Great Toque Outdoors has many tables in its database, some of which use classic compression. This section shows how to determine which tables are good candidates for adaptive compression. It has examples to show how new tables are created with adaptive compression and existing tables are converted to adaptive compression. It also provides monitoring and tuning information.

3.3.1 Creating new tables with adaptive compression

The compression options in DB2 10 are available through new syntax in the CREATE TABLE statement shown in Example 3-1.

Example 3-1 Compression clause in the CREATE TABLE statement

```
      .-COMPRESS NO---------------.
>-----+---------------------------+---->
      |              .-ADAPTIVE-. |
      '-COMPRESS YES-+----------+-'
                     '-STATIC---'
```

For new tables that are created in Great Toque Outdoors' database, adaptive compression is applied by default when the following statement is issued:

```
CREATE TABLE … COMPRESS YES
```

This statement is the equivalent of explicitly issuing the following statement:

```
CREATE TABLE … COMPRESS YES ADAPTIVE
```

New tables can still be created with classic compression only, by issuing the following statement:

```
CREATE TABLE … COMPRESS YES STATIC
```

To create a table with no compression, issue the `CREATE TABLE` statement without a `COMPRESS` clause

Or, explicitly issue the following statement:

```
CREATE TABLE … COMPRESS NO
```

Figure 3-3 on page 41 compares data in an uncompressed table to the same data in tables with classic compression and adaptive compression. The table compressed with classic compression uses automatic dictionary compression (ADC), meaning the table-level compression dictionary is created automatically when the table has grown to a sufficient size. However, only newly inserted or updated rows are subject to compression; rows that existed *before* creating the compression dictionary are not compressed.

The size of the table that is compressed with classic compression by using ADC table is 21.2 GB. To achieve an optimal compression ratio on this table, a classic table reorganization must be done to compress the table further to 10.5 GB.

Notice, however, the size of the table compressed with adaptive compression is 8.4 GB *without* a classic table reorganization, which is even smaller than the same table with classic compression *plus* a classic table reorganization (10.5 GB).

If you *do* perform a table reorganization on the table with adaptive compression, its size shrinks even further from 8.4 GB to 6.4 GB.

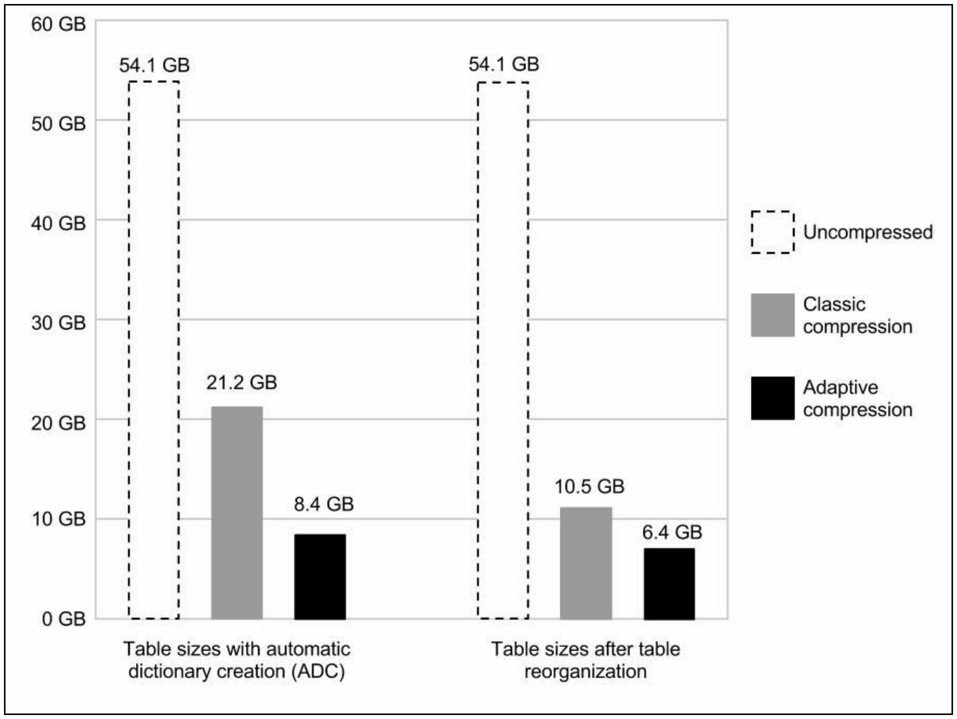


Figure 3-3 Table sizes and compression ratios

Tip: Adaptive compression assesses recurring patterns across an entire row of data, rather than individual columns. To maximize compression potential, when creating a table, consider arranging its columns so that recurring patterns are likely to occur. For example, if your table will have columns for city, state, and postal code, create the table so these columns are contiguous. This way sets up the table for recurring patterns better than city, telephone number, state, and postal code, for example.

3.3.2 Identifying candidate tables for adaptive compression

Before applying adaptive compression to existing tables, an important step is to first assess which tables are ideal candidates for compression: classic compression or adaptive compression. Ideal candidates for adaptive compression are tables that contain significant amounts of data or tables for which substantial growth is anticipated over time.

To begin, the DBAs at Great Toque Outdoors run the query in Example 3-2. The query uses the SQL administration function ADMIN_GET_TAB_INFO to return an ordered list of table names and table data object sizes for a particular schema.

To display the current compression settings and the row compression mode that are in use, the ROWCOMPMODE column from the SYSCAT.TABLES view is included through a JOIN clause.

Example 3-2 Identifying top space consumers

```
SELECT SUBSTR(T.TABSCHEMA, 1, 10) AS TABSCHEMA,
       SUBSTR(T.TABNAME, 1, 20) AS TABNAME,
       SUM(TI.DATA_OBJECT_P_SIZE)/1024/1024 AS STORAGESIZE_GB,
       T.ROWCOMPMODE AS ROWCOMPMODE
FROM TABLE (SYSPROC.ADMIN_GET_TAB_INFO('TOQUE', '')) TI
       JOIN SYSCAT.TABLES T ON T.TABSCHEMA = TI.TABSCHEMA AND
                               T.TABNAME = TI.TABNAME
GROUP BY T.TABSCHEMA, T.TABNAME, T.ROWCOMPMODE
ORDER BY STORAGESIZE_GB DESC;

TABSCHEMA  TABNAME                   STORAGESIZE_GB ROWCOMPMODE
---------- ------------------------- -------------- -----------
TOQUE      ORDER_DETAILS                         14           S
TOQUE      SALES_TARGET                          12           S
TOQUE      PRODUCT_FORECAST                      12           S
TOQUE      ORDER_HEADER                          10
TOQUE      INVENTORY_LEVELS                       9           S
TOQUE      PRODUCT_NAME_LOOKUP                    7           S

  6 record(s) selected.
```

The query in Example 3-2 identifies the top-space consumers, in the Great Toque Outdoors database, by listing the tables from biggest to smallest. This way helps the DBAs identify which tables to start working with. In this example, the ORDER_HEADER table is uncompressed; the remaining tables have classic compression.

With candidate tables identified, the DBAs at Great Toque Outdoors can estimate space savings with the ADMIN_GET_TAB_COMPRESS_INFO administrative table function, as shown in Example 3-3. The percentages returned are based on an assumption that the following command will be run:

```
TABLE REORG … RESETDICTIONARY
```

Note: The SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO function computes the estimates for all tables in a particular schema if a table name is not specified in the second parameter. The query in Example 3-3 calls the function in this manner for illustrative purposes; in practical uses, however, a table should be specified because the run time is rather long when querying an entire schema.

Example 3-3 Estimating saved space

```
SELECT SUBSTR(TABNAME, 1, 20) AS TABNAME,
   ROWCOMPMODE,
   PCTPAGESSAVED_CURRENT AS PCT_CURR,
   PCTPAGESSAVED_STATIC AS PCT_STATIC,
   PCTPAGESSAVED_ADAPTIVE AS PCT_ADAPTIVE
FROM TABLE(SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO('TOQUE', ''));

TABNAME              ROWCOMPMODE PCT_CURR PCT_STATIC PCT_ADAPTIVE
-------------------- ----------- -------- ---------- ------------
ORDER_DETAILS        S                 67         68           72
SALES_TARGET         S                 78         83           90
PRODUCT_FORECAST     S                 80         82           90
ORDER_HEADER                            0         11           17
INVENTORY_LEVELS     S                 69         70           73
PRODUCT_NAME_LOOKUP  S                 87         87           91

  6 record(s) selected.
```

The results include values that are derived from the following columns that are returned by the SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO function:

- ROWCOMPMODE

  This column is the current compression mode of the table, displayed as one of the following values:

S	Classic row compression
A	Adaptive row compression
Blank	No row compression

- PCTPAGESSAVED_CURRENT

  This column is the current compression savings.

- PCTPAGESSAVED_STATIC

  This column is the estimated compression savings that can be achieved with classic row compression.

- PCTPAGESSAVED_ADAPTIVE

  This column is the estimated compression savings that can be achieved with adaptive row compression.

The results of the PRODUCT_FORECAST table, in Example 3-3 on page 43, show an estimated space savings of 82% with classic compression, compared to a space savings of 90% with adaptive compression. This comparison indicates that if a 1 GB table was compressed with classic compression, its size would be about 184 MB; the table compressed with adaptive compression would be reduced to about 102 MB.

Of the six tables for which the compression estimates were computed, five show good compression potential:

- For the ORDER_DETAILS, SALES_TARGET and PRODUCT_FORECAST tables, the Great Toque Outdoors DBAs decide to apply adaptive compression because of the significant storage space savings.
- For the INVENTORY_LEVELS and PRODUCT_NAME_LOOKUP tables, the Great Toque Outdoors DBAs consider continuing with classic compression (or at least set a lower priority for altering the table for adaptive compression), unless the DBAs expect the data characteristics to change significantly or expect much new data to be inserted. The DBAs notice however, that the compression ratio can be improved even by merely performing a classic table reorganization.
- The ORDER_HEADER table has rather poor compression estimates. The storage space savings relative to the amount of processing overhead incurred with uncompressing data (albeit small) might be too low to justify enabling compression on this particular table.

3.3.3 Converting existing tables to use adaptive compression

Existing tables that are either uncompressed or compressed with classic compression remain that way after upgrading to DB2 10. However, existing tables can be converted to use adaptive compression with the following statement:

```
ALTER TABLE … COMPRESS YES ADAPTIVE
```

Tip: For a table in a regular table space, compression might cause the average row size to be short. For a table in a regular table space, each page can hold a maximum of 255 rows. If your average row size is near or below that ratio, consider converting the regular table space to a large table space with the `ALTER TABLESPACE...CONVERT TO LARGE` statement. Alternatively, use the ADMIN_MOVE_TABLE procedure to move your table to a large table space.

The DBAs at Great Toque Outdoors have decided to first convert the TOQUE.PRODUCT_FORECAST table to adaptive compression, so they issue the ALTER TABLE statement shown in Example 3-4.

Example 3-4 Altering a table to use adaptive compression

```
ALTER TABLE TOQUE.PRODUCT_FORECAST COMPRESS YES ADAPTIVE;
```

This ALTER TABLE statement does not apply adaptive compression to any data that already exists in the table. Adaptive compression is applied as new data is inserted and existing data is updated. In this case, the benefits of adaptive compression will materialize over time. However, the DBAs can achieve immediate high space savings by applying adaptive compression to *all* data by rebuilding the compression dictionaries with one of the following methods:

- The ADMIN_MOVE_TABLE procedure, as shown in Example 3-5. The table remains online when this procedure is called.
- The REORG command, as shown in Example 3-6. The table is offline when this command is run.

Example 3-5 Calling the ADMIN_MOVE_TABLE procedure

```
CALL SYSPROC.ADMIN_MOVE_TABLE('TOQUE','PRODUCT_FORECAST',
                         '','','','','','','','','MOVE');
```

Example 3-6 Performing a classic table reorganization

```
REORG TABLE TOQUE.PRODUCT_FORECAST RESETDICTIONARY;
```

3.3.4 Monitoring compression ratios and optionally performing a table reorganization

Because the PRODUCT_FORECAST table already has a table-level compression dictionary, it already contains compressed rows. As the existing table data is manipulated and new data is inserted into the table, page-level compression dictionaries are created and dynamically updated as needed. However, the existing table-level compression dictionary is static; over time, the table-level compression ratio might gradually decline. Therefore, the DBAs monitor the compression ratios of the table over time. They can choose one of the following methods to monitor compression ratios:

- Running the same SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO function that was used to estimate space savings, as shown in Example 3-3 on page 43. The advantage of this method is that the current compression savings can be determined with a single function call.
- Running the RUNSTATS command for a given table, then querying the PCTPAGESSAVED column of the SYSCAT.TABLES catalog view. The advantage of this second method is that execution time might be less than the SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO function.

Regardless of which method is used, Great Toque Outdoors DBAs occasionally monitor the compression ratio of a table over time, as opposed to a single snapshot. With multiple snapshots of compression ratios, DBAs can observe the trending of a table's compression.

To make the tracking of compression ratios easier, the DBAs first set up a table to record compression ratios, as shown in Example 3-7.

Example 3-7 Creating a compression history table

```
CREATE TABLE ADMIN.COMPRESSION_HISTORY (TABSCHEMA       VARCHAR(128),
                                        TABNAME         VARCHAR(128),
                                        PCTPAGESSAVED   SMALLINT,
                                        MEASUREMENT_DATE DATE);
```

Next, DBAs run the RUNSTATS command against their PRODUCT_FORECAST table, as shown in Example 3-8.

Example 3-8 Running the RUNSTATS command

```
RUNSTATS ON TABLE TOQUE.PRODUCT_FORECAST
```

Each time the RUNSTATS command is run against the PRODUCT_FORECAST table, the DBAs store the date and percentage of space saved in the compression history table that they created in Example 3-7 on page 46. This INSERT operation is shown in Example 3-9.

Example 3-9   Storing the date and percentage of space saved

```
INSERT INTO ADMIN.COMPRESSION_HISTORY
   SELECT TABSCHEMA, TABNAME, PCTPAGESSAVED, CURRENT DATE
   FROM SYSCAT.TABLES
   WHERE TABSCHEMA NOT LIKE 'SYS%';
```

After storing the date and percentage of space that is saved each time RUNSTATS is run against the PRODUCT_FORECAST table, the DBAs can query the compression history table that they created in Example 3-7 on page 46. This query and its results are shown in Example 3-10.

Example 3-10   Querying the compression history table

```
SELECT PCTPAGESSAVED, MEASUREMENT_DATE
   FROM ADMIN.COMPRESSION_HISTORY
   WHERE TABSCHEMA = 'TOQUE' AND TABNAME = 'PRODUCT_FORECAST'
   ORDER BY MEASUREMENT_DATE ASC;

PCTPAGESSAVED MEASUREMENT_DATE
------------- ----------------
           89 03/18/2012
           78 05/25/2012
           71 08/10/2012
           66 10/06/2012
           59 11/10/2012

  5 record(s) selected.
```

When the DBAs believe the PRODUCT_FORECAST table exhibits too much data drift, they schedule a table reorganization to coincide with the next database maintenance window. At that time, they do a classic table reorganization to rebuild the table-level compression dictionary and page-level compression dictionaries, by using the command shown in Example 3-6 on page 45.

3.4 Summary

With the DB2 10 adaptive compression feature, you can achieve extremely high compression rates and reduce current and future storage space requirements.

Adaptive compression is dynamic and automatic. Compression rates remain high as your tables grow and change, yet adaptive compression requires less administration than classic compression. With adaptive compression, less monitoring is necessary, and in most cases, there is no need for routine table reorganizations.

4

Multi-temperature storage

In this chapter, we describe multi-temperature storage to optimize the use of the various types of storage available for an application environment. Multi-temperature storage reserves faster, more expensive storage for critical data, and slower, less expensive storage for all other data. Using this multi-temperature storage approach helps to balance storage purchases across solid-state drive (SSD), serial-attached SCSI (SAS), and Serial Advanced Technology Attachment (SATA) storage, and to optimize storage budgets.

4.1 Overview of multi-temperature storage

In today's business environment, where enterprises are continually seeking methods to reduce IT operating costs, database storage is a key component of operating-cost reduction discussions. Critical data requires the fastest possible disk performance to achieve service-level performance requirements. However, as data ages, its importance relative to the business is reduced also. Therefore, as data ages, storage costs can be reduced if the data is migrated to lower cost storage. DB2 10 provides multi-temperature storage through the automatic storage component of the DB2 database to identify database storage paths as hot, warm, or cold:

- Hot: Defined for mission-critical data that requires the fastest access.
- Warm: Defined for data that is still queried but does not have the requirement for fastest access.
- Cold: Defined for data that might need to remain in the database for regulatory or audit purposes but is rarely accessed and data access speed is not critical.

With the multi-temperature storage component of DB2 10, database storage can be purchased and allocated based on the access and storage requirements for the types of data (hot, warm, or cold) to allow an informed storage acquisition process. In this way, an enterprise can build a realistic storage budget based on its application and data-aging requirements, and purchase the exact storage needed for each storage type.

4.2 Business value

DB2 10 multi-temperature storage provides the following business benefits:

- Allows application service-level agreements to be tied to storage tiers based on data access priority by using DB2 table partitioning to separate a table into easily manageable pieces, and distributing those table partitions among the multi-temperature storage groups within a database based on data priority. As data ages and its priority diminishes, the table space for that table partition can easily be moved to a lower priority storage group. Note that the process of moving table spaces to new storage groups is an online process.
- Allows a storage type to be targeted to a specific storage group based on the performance requirements for a particular table partition. In this way, storage purchases can be divided into separate classes of storage based on data access requirements, versus always buying the fastest (most expensive) storage. Storage purchases and capital expenditures are tailored to actual

data access speed requirements to optimize storage expenditures based on business requirements for data access.

- Allows easy movement of table data between temperature ranges within the storage infrastructure for the database as the data ages and the data access requirements change.

Figure 4-1 shows an overview of DB2 10 and multi-temperature storage, describing automatic storage and the new concept of storage groups that make multi-temperature storage within DB2 possible.

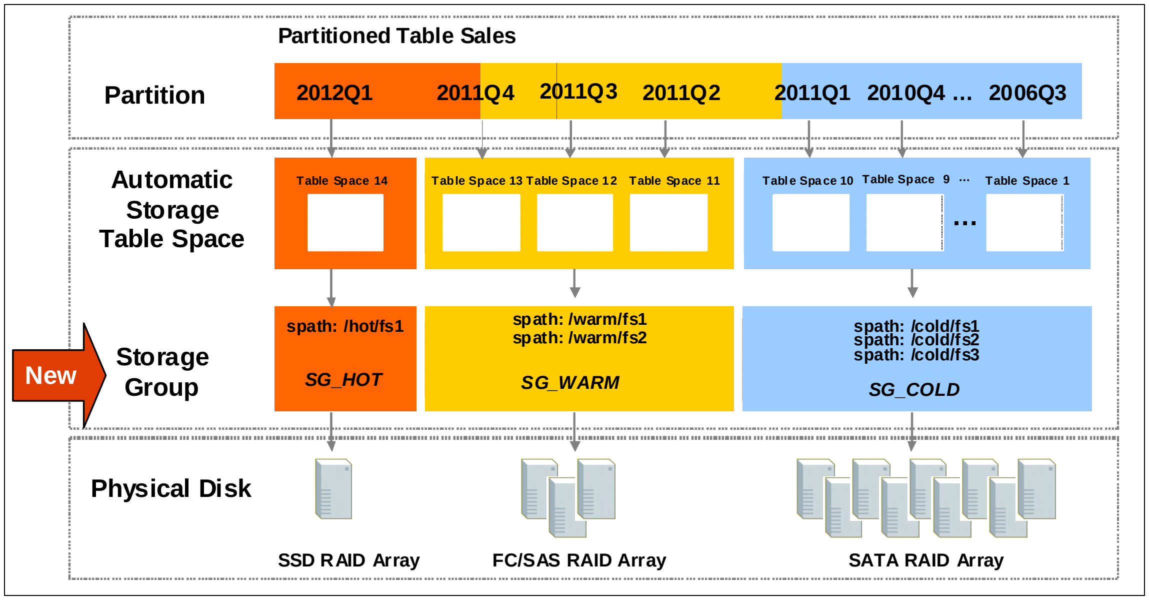


Figure 4-1 Concept of storage groups within DB2 10

The storage group layer within DB2 10 automatic storage provides the layer of abstraction between the table space and physical disk; it allows table spaces to be moved as an online operation from one storage group to another. As the diagram depicts, a database storage system can consist of various storage devices such as solid-state drives (SSD), storage area networks (SAN, such as FC/SAS RAID Array), and older, slower disks (such as SATA RAID Array). By assigning table spaces to specific storage groups, the disk configuration for the database can be optimized for the data access patterns of the application. Critical, high activity data can be placed on faster, higher temperature storage and less critical, infrequently accessed data can be placed on slower, less expensive storage without adversely impacting application performance.

4.3 Business application examples

Like many organizations in IT environments of today, a large portion of Great Toque Outdoors annual operating budget is targeted for new storage purchases. The new storage would help the company keep more data online for analysis purposes, and meet corporate standards for data retention. The current data growth rates would have Great Toque Outdoors purchasing 200 terabytes (TB) of storage annually at the corporate level.

For its corporate sales data warehouse, Great Toque Outdoors maintains approximately 3 TB of data with data access patterns as follows:

- Sixty percent of the queries and analysis accesses the most recent month of sales data (which accounts for approximately 80 GB of data).
- Thirty percent of the queries and analysis accesses the current quarter, the most recent three months, of data (which accounts for approximately 240 GB of data).
- Ten percent of the queries accesses data older than three months (which accounts for approximately 2.8 TB of data).

Further discussions with Great Toque Outdoors about service-level agreements and workload priority highlighted that the data priority for the current month was high, the current quarter was medium, and all other data was low priority in terms of access requirements. Great Toque Outdoors decided to create three storage groups for its DB2 10 database: (TOQUE_HOT, TOQUE_WARM, and TOQUE_COLD.

Upgrade tip: Multi-temperature storage requires DB2 automatic storage and its new storage group definition to be able to configure multi-temperature storage for the table spaces within a database. DB2 10 allows a non-automatic storage table space to be converted to automatic storage as an online process. Clients that upgrade their DB2 environments to DB2 10 can take advantage of multi-temperature storage and automatic storage within DB2 by this conversion, with minimal effort.

4.3.1 Multi-temperature storage description and usage

Prior to DB2 10, automatic storage within DB2 used a single storage container (or *group* in DB2 10 terminology). DBAs could specify multiple storage paths when defining automatic storage, but all table spaces within the database used all paths within that single storage container (group). DB2 10 has added a new component to the automatic storage concept, the *storage group*. Automatic storage now consists of one (default) or more storage groups, and each storage group can have multiple storage paths. Table spaces can now be assigned to specific storage groups within automatic storage or if no storage group is specified, the table space is assigned to the default storage group (IBMSTOGROUP).

DB2 limit increase: With the addition of storage groups within automatic storage, the limit for storage paths has been increased from 128 storage paths to 128 storage paths per storage group. DB2 10 allows 256 storage groups to be defined per database instance.

Because DB2 table spaces allow the disk characteristics to be specified (overhead and transfer rate) for DB2 optimizer plan selection during statement compilation, these disk characteristics have been added to the storage group definition in automatic storage (identified as overhead and disk read rate at the storage group level). This way allows the disk performance characteristics at the table space level to be inherited from the storage group, which eliminates the need for DBAs to intervene when a table space is moved from a higher temperature storage group to a lower temperature storage group (for example, hot to warm).

When you are first introduced to multi-temperature storage in DB2 10, an important concept to understand is the difference between multi-temperature storage within automatic storage and simply moving a table from one table space to a new table space through an online table-move or another method. Multi-temperature storage has the following key differentiators:

- Ease of use

  Ease of use is achieved through the addition of the storage group concept within DB2 automatic storage. To implement multi-temperature storage, you basically define a storage group within automatic storage. When this is defined, a table space is assigned to the storage group using either the CREATE TABLESPACE or ALTER TABLESPACE statements. If an ALTER TABLESPACE statement is executed, the table space contents will be moved from the existing containers or storage group to the new storage group specified.

- Flexibility

  Flexibility allows the storage groups to be defined to meet your business requirements. A database instance provides up to 256 storage groups that can be defined and the storage group names can be tailored to match naming standards in your organization. Multi-temperature storage is not limited to one storage group per data temperature; DBAs may define multi-temperature storage groups for each application. This approach allows an organization to define various data priorities and storage media types for a particular data "temperature" based on individual application requirements. For a mission-critical application, the "hot" storage group might be SSD storage, but a less critical application might have a "warm" storage group as its highest temperature.

- Data priority identification

  Data priority identification allows a data tag to be defined at both the storage group and table space levels. This data tag is an integer value in the range of 1 - 9. It is used by DB2 Workload Manager (WLM) to assign workload priority based on the data tag (with the high and low priority number defined by the DBAs, based on the organization's preference as to whether high or low values are high priority). Unlike previous workload definitions in earlier versions of DB2, which determined priority based on an organizational group or application, DB2 10 WLM with multi-temperature storage now allows workload priority based on the data priority. In this way, queries can be prioritized through the data accessed, rather than the group or application running the query.

- Availability

  Availability provides movement of table spaces between storage groups as an online operation to maximize application availability. To minimize application impact during the rebalance operation, with DB2 10 the rebalance utility can be throttled to restrict the impact of the utility to applications executing against the database.

4.3.2 Adding multi-temperature storage to a DB2 10 database

When a new database is created in DB2 10, a single storage group is created. Adding or dropping storage paths, changing storage performance parameters (overhead and device-read rate), or changing the data tag value within the default storage group (IBMSTOGROUP) is done through the ALTER STOGROUP statement. To define additional storage groups within the database, the CREATE STOGROUP statement is used. After storage groups are created, they are assigned to table spaces through the CREATE TABLESPACE or ALTER TABLESPACE statements. Alternatively, you can also add storage groups to a database as part of a redirected-restore operation. This latter method might be the preferable way to add storage groups if you are moving the database to a new storage system as part of the DB2 10 upgrade process.

Although DB2 10 allows the storage performance-related parameters to be specified at either the storage group or table space level, the preferred practice is to define the performance values (OVERHEAD and DISK READ RATE) at the storage group level. This way eliminates storage parameter changes at the table space level, because table spaces move between storage groups.

Tip: Define the storage performance parameters (OVERHEAD and DISK READ RATE) at the storage group level and set OVERHEAD and TRANSFERRATE at the table space level (through the CREATE TABLESPACE or ALTER TABLESPACE statements) to INHERIT. This step allows a table space to be moved among various storage group temperatures without you having to modify the table space parameters.

Great Toque Outdoors created its initial DB2 10 database for the SALES database with the default storage group by using the CREATE DATABASE statement, as shown in Example 4-1.

Example 4-1 Great Toque Outdoors SALES database creation

```
CREATE DATABASE sales ON ‘/home/db2inst1’;
```

After the SALES database are created, the three multi-temperature storage paths are defined by using the statements shown in Example 4-2.

Example 4-2 Great Toque Outdoors multi-temperature storage groups

```
CREATE STOGROUP toque_hot ON ‘/gto_sales_datahot’ OVERHEAD 0.75
   DEVICE READ RATE 500;
CREATE STOGROUP toque_warm ON ‘/gto_sales_datawarm’ OVERHEAD 3.4
   DEVICE READ RATE 112;
CREATE STOGROUP toque_cold ON ‘/gto_sales_datacold’ OVERHEAD 4.2
   DEVICE READ RATE 100;
```

After the storage groups are created within the SALES database, the Great Toque Outdoors database administrators (DBAs) create the range-partitioned ORDERS table for data since January 2011. Based on data access patterns for the application, Great Toque Outdoors decided to place data older than three months (the current “rolling quarter” of data) in the cold storage group. Data for the two months before the current month is placed in the warm storage group and the most active and critical data of the current month is placed in the hot storage group. This storage temperature pattern will be maintained as each new month of data is “attached” (in DB2 terminology) to the table. The initial ORDERS table definition included monthly partitions from January 2011 through May 2012. As each new month is added to the table, the month prior to the added partition will be moved from hot to warm storage and the oldest month of the prior rolling quarter of data will be moved from warm storage to cold storage.

Tip: DB2 10 provides a table function to view storage groups within a database. To view all of the storage groups in the currently connected database, issue the following statement:

```
SELECT VARCHAR(STORAGE_GROUP_NAME, 30) AS STOGROUP,
   VARCHAR(DB_STORAGE_PATH, 40) AS STORAGE_PATH FROM
   TABLE(ADMIN_GET_STORAGE_PATHS('',-1)) AS T
```

Issuing the DB2 table function ADMIN_GET_STORAGE_PATH against Great Toque Outdoors database shows the storage groups in Example 4-3.

Example 4-3 Storage group results from ADMIN_GET_STORAGE_PATH

```
STOGROUP                       STORAGE_PATH
------------------------------ ----------------------------------------
IBMSTOGROUP                    /home/db2inst1
TOQUE_HOT                      /gto_sales_data_hot
TOQUE_WARM                     /gto_sales_data_warm
TOQUE_COLD                     /gto_sales_data_cold
```

The first step to the table creation process for Great Toque Outdoors is the table space definition with assignment to the three storage groups. Because movement between storage group temperatures is at the table space level, a table space will be defined for each table partition within the ORDERS table. This step allows the partitions for the ORDERS table to be moved easily between storage groups as needed. Sample table space definitions for the three storage groups (hot, warm, and cold) are shown in Example 4-4.

Example 4-4 Great Toque Outdoors SALES database table space definition

```
CREATE TABLESPACE jan2012 MANAGED BY AUTOMATIC STORAGE
   USING STOGROUP toque_cold INITIALSIZE 50M INCREASESIZE 5M OVERHEAD
   INHERIT TRANSFERRATE INHERIT;
CREATE TABLESPACE feb2012 MANAGED BY AUTOMATIC STORAGE
   USING STOGROUP toque_cold INITIALSIZE 50M INCREASESIZE 5M OVERHEAD
   INHERIT TRANSFERRATE INHERIT;
CREATE TABLESPACE mar2012 MANAGED BY AUTOMATIC STORAGE
   USING STOGROUP toque_warm INITIALSIZE 50M INCREASESIZE 5M OVERHEAD
   INHERIT TRANSFERRATE INHERIT;
CREATE TABLESPACE apr2012 MANAGED BY AUTOMATIC STORAGE
   USING STOGROUP toque_warm INITIALSIZE 50M INCREASESIZE 5M OVERHEAD
   INHERIT TRANSFERRATE INHERIT;
CREATE TABLESPACE may2012 MANAGED BY AUTOMATIC STORAGE
   USING STOGROUP toque_hot INITIALSIZE 50M INCREASESIZE 5M OVERHEAD    INHERIT
TRANSFERRATE INHERIT;
```

After the table spaces are created in the specified storage groups for Great Toque Outdoors, the ORDERS table are created as a partitioned table, as shown in Example 4-5.

Example 4-5 CREATE TABLE syntax for ORDERS table

```
CREATE TABLE TOQUE.ORDERS  (
   ORDER_NUMBER INTEGER NOT NULL,
   ORDER_DATE DATE,
   PRODUCT_NUMBER INTEGER NOT NULL,
   CUSTOMER_NUMBER INTEGER,
   SHIP_DATE DATE,
   COMMENTS VARCHAR(100))
PARTITION BY RANGE(ORDER_DATE) (
   ...earlier partitions omitted
   PARTITION ord_jan_2012 STARTING FROM ('01/01/2012') INCLUSIVE ENDING
      AT ('02/01/2012') EXCLUSIVE IN toque_cold,
   PARTITION ord_feb_2012 STARTING FROM ('02/01/2012') INCLUSIVE ENDING
      AT ('03/01/2012') EXCLUSIVE IN toque_cold,
   PARTITION ord_mar_2012 STARTING FROM ('03/01/2012') INCLUSIVE ENDING
      AT ('04/01/2012') EXCLUSIVE IN toque_warm,
   PARTITION ord_apr_2012 STARTING FROM ('04/01/2012') INCLUSIVE ENDING
      AT ('05/01/2012') EXCLUSIVE IN toque_warm,
   PARTITION ord_may_2012 STARTING FROM ('05/01/2012') INCLUSIVE ENDING
      AT ('06/01/2012') EXCLUSIVE IN toque_hot)
INDEX IN toque_index;
```

The data is then loaded into the ORDERS table and normal application activity takes place until June 2012. At this point in time, a NEWORDERS table is created to load the June 2012 orders data to prepare for the DB2 ATTACH process to the ORDERS table. When preparing for the DB2 ATTACH operation, the table containing the new table partition must have the same column and data type information (column names, order of columns, data type definition and length, and identical indexes).

As the syntax in Example 4-6 illustrates, the NEWORDERS table definition matches the ORDERS table.

Example 4-6   NEWORDERS table space and table definition

```
CREATE TABLESPACE jun2012 MANAGED BY AUTOMATIC STORAGE
   USING STOGROUP toque_hot INITIALSIZE 50M INCREASESIZE 5M
   OVERHEAD INHERIT TRANSFERRATE INHERIT;

COMMIT WORK;
CREATE TABLE TOQUE.NEWORDERS  (
   ORDER_NUMBER INTEGER NOT NULL,
   ORDER_DATE DATE NOT NULL,
   PRODUCT_NUMBER INTEGER NOT NULL,
   CUSTOMER_NUMBER INTEGER,
   SHIP_DATE DATE,
   COMMENTS VARCHAR(100))
IN jun2012 INDEX IN toque_index;
CREATE INDEX TOQUE.NEWORDERS_IDX ON TOQUE.NEWORDERS(ORDER_DATE);
```

Tip: When adding a partition to an existing partitioned table, create the table partition within the desired target storage pool (through the table partition table space definition) prior to the ATTACH operation. This way prevents a post ATTACH rebalance operation.

After the NEWORDERS table is created and loaded, the new JUN2012 range partition is added to the ORDERS table with the syntax shown in Example 4-7.

Example 4-7   NEWORDERS table (jun2012) partition is added to ORDERS table

```
ALTER TABLE TOQUE.ORDERS
   ATTACH PARTITION ord_jun_2012 STARTING FROM ('06/01/2012') INCLUSIVE
   ENDING AT ('07/01/2012') EXCLUSIVE
   FROM TABLE TOQUE.NEWORDERS;
```

Now that the new table partition (jun2012) is added to the ORDERS table, a query against the SYSCAT.TABLESPACES system view is executed to identify the table space names and their associated storage groups prior to the movement of the MAR2012 and MAY2012 table partitions. Actually, the underlying table spaces moved to new storage groups; there is no change to the table partition definitions themselves.

The results of this query are shown in Example 4-8.

Example 4-8 Query against SYSCAT.TABLESPACES to display storage groups

```
SELECT TBSPACE, SGNAME FROM SYSCAT.TABLESPACES
   WHERE SGNAME LIKE 'TOQUE_%'

Excerpt of query results displaying 2012 table partitions ONLY
TABLESPACE    STORAGE_GROUP
----------------------------
JAN2012       TOQUE_COLD
FEB2012       TOQUE_COLD
MAR2012       TOQUE_WARM
APR2012       TOQUE_WARM
MAY2012       TOQUE_HOT
JUN2012       TOQUE_HOT
```

The ALTER TABLESPACE statements are now issued by the Great Toque Outdoors DBAs to move the MAR2012 table partition to the TOQUE_COLD storage group, and to move the MAY2012 table partition to the TOQUE_WARM storage group. The actual statements are shown in Example 4-9.

Example 4-9 ALTER TABLESPACE commands to identify new storage groups

```
ALTER TABLESPACE may2012 USING STOGROUP toque_warm;
ALTER TABLESPACE mar2012 USING STOGROUP toque_cold;
```

During the storage group rebalance operation, where the MAY2012 and MAR2012 table spaces are moved to new storage groups, the rebalance process can be monitored through the MON_GET_REBALANCE_STATUS table function, as shown in Example 4-10.

Example 4-10 MON_GET_REBALANCE _STATUS table function syntax

```
SELECT VARCHAR(TBSP_NAME, 30) AS TABLESPACE, REBALANCER_MODE AS MODE,
   REBALANCER_STATUS AS STATUS,
   REBALANCER_EXTENTS_REMAINING AS EXT_REM,
   REBALANCER_EXTENTS_PROCESSED AS EXT_PROC
   FROM TABLE(MON_GET_REBALANCE_STATUS(NULL,-2)) AS T;
```

Several sample results of the MON_GET_REBALANCE_STATUS table function are shown next to illustrate the steps in the rebalance operation that moves table spaces between storage groups. The actual status provided by the rebalance operation is shown in Example 4-11.

Example 4-11 MON_GET_REBALANCE_STATUS results

```
TABLESPACE  MODE                STATUS   EXT_REM     EXT_PROC
---------   -----               ------   --------    --------
MAY2012     FWD_REBAL           ACTIVE     0           0
========================================================================

TABLESPACE  MODE                STATUS   EXT_REM     EXT_PROC
---------   -----               ------   --------    --------
MAY2012     FWD_REBAL_OF_2PASS  ACTIVE     13          1
========================================================================

TABLESPACE  MODE                STATUS   EXT_REM     EXT_PROC
---------   -----               ------   --------    --------
MAR2012     FWD_REBAL_OF_2PASS  ACTIVE     13          1
MAY2012     REV_REBAL_OF_2PASS  ACTIVE     4           10
========================================================================

TABLESPACE  MODE                STATUS   EXT_REM     EXT_PROC
---------   -----               ------   --------    --------
MAR2012     FWD_REBAL_OF_2PASS  ACTIVE     12          2
MAY2012     REV_REBAL_OF_2PASS  ACTIVE     4           10
========================================================================

TABLESPACE  MODE                STATUS   EXT_REM     EXT_PROC
---------   -----               ------   --------    --------
MAR2012     FWD_REBAL_OF_2PASS  ACTIVE     11          3
MAY2012     REV_REBAL_OF_2PASS  ACTIVE     4           10
========================================================================

TABLESPACE  MODE                STATUS   EXT_REM     EXT_PROC
---------   -----               ------   --------    --------
MAR2012     FWD_REBAL_OF_2PASS  ACTIVE     11          3
MAY2012     REV_REBAL_OF_2PASS  ACTIVE     3           11
========================================================================
TABLESPACE  MODE                STATUS   EXT_REM    EXT_PROC
---------   -----               ------   --------   --------
MAR2012     FWD_REBAL_OF_2PASS  ACTIVE     11          3
MAY2012     REV_REBAL_OF_2PASS  ACTIVE     2           12
========================================================================

TABLESPACE  MODE                STATUS   EXT_REM    EXT_PROC
---------   -----               ------   --------   --------
MAR2012     FWD_REBAL_OF_2PASS  ACTIVE     10          4
MAY2012     REV_REBAL_OF_2PASS  ACTIVE     1           13
========================================================================

TABLESPACE  MODE                STATUS   EXT_REM    EXT_PROC
---------   -----               ------   --------   --------
MAR2012     FWD_REBAL_OF_2PASS  ACTIVE     9           5
MAY2012     REV_REBAL_OF_2PASS  ACTIVE     1           13
========================================================================

TABLESPACE  MODE                STATUS   EXT_REM    EXT_PROC
---------   -----               ------   --------   --------
MAR2012     FWD_REBAL_OF_2PASS  ACTIVE     10          4
MAY2012     REV_REBAL_OF_2PASS  ACTIVE     1           13
========================================================================

TABLESPACE  MODE                STATUS   EXT_REM    EXT_PROC
---------   -----               ------   --------   --------
MAR2012     FWD_REBAL_OF_2PASS  ACTIVE     9           5
MAY2012     REV_REBAL_OF_2PASS  ACTIVE     1           13
========================================================================

TABLESPACE  MODE                STATUS   EXT_REM    EXT_PROC
---------   -----               ------   --------   --------
MAR2012     REV_REBAL_OF_2PASS  ACTIVE     5           9
========================================================================

TABLESPACE  MODE                STATUS   EXT_REM    EXT_PROC
---------   -----               ------   --------   --------
MAR2012     REV_REBAL_OF_2PASS  ACTIVE     4           10

========================================================================
TABLESPACE  MODE                STATUS   EXT_REM     EXT_PROC
---------   -----               ------   --------    --------
MAR2012     REV_REBAL_OF_2PASS  ACTIVE      3           11
=====================================================================

TABLESPACE  MODE                STATUS   EXT_REM     EXT_PROC
---------   -----               ------   --------    --------
MAR2012     REV_REBAL_OF_2PASS  ACTIVE      2           12

=====================================================================
TABLESPACE  MODE                STATUS   EXT_REM     EXT_PROC
---------   -----               ------   --------    --------
MAR2012     REV_REBAL_OF_2PASS  ACTIVE      1           13

=====================================================================
After this point, the REBALANCE operation was completed prior to the
next execution of the MON_GET_REBALANCE_STATUS table function query.
=====================================================================
```

Tip: When you review the MON_GET_REBALANCE_STATUS output, consider the following MODE definitions:

- FWD_REBAL

  Occurs when new containers are added or the size of existing containers increase. In a forward-rebalancing operation, data movement starts with the first extent in the table space and ends with the high watermark extent.

- REV_REBAL

  Occurs when containers are removed or reduced in size and data must move out of the space that is being freed. In a reverse rebalancing operation, data movement starts at the high watermark extent and moves in reverse order through the table space, ending with the first extent in the table space.

- FWD_REBAL_OF_2PASS or REV_REBAL_OF_2PASS

  A two-pass rebalance is a forward rebalance followed by a reverse rebalance. The status shows which phase of the two-pass rebalance is in progress at a particular point in time.

After the rebalance operation is completed, the query against the SYSCAT.TABLESPACES system view is rerun to validate that the storage groups for the MAR2012 and MAY2012 table partition table spaces are now in the correct storage groups (TOQUE_COLD and TOQUE_WARM). The results of the query against the SYSCAT.TABLESPACES view are shown in Example 4-12.

Example 4-12 SYSCAT.TABLESPACES query results validating storage groups

```
SELECT TBSPACE, SGNAME FROM SYSCAT.TABLESPACES
   WHERE SGNAME LIKE 'TOQUE_%'

Excerpt of query results displaying 2012 table partitions ONLY
TABLESPACE      STORAGE_GROUP
---------------------------
JAN2012         TOQUE_COLD
FEB2012         TOQUE_COLD
MAR2012         TOQUE_COLD
APR2012         TOQUE_WARM
MAY2012         TOQUE_WARM
JUN2012         TOQUE_HOT
```

As the query results demonstrate, after rebalancing is complete, the table spaces that are being moved (MAR2012 and MAY2012) are located in their new storage groups (TOQUE_COLD and TOQUE_WARM).

DB2 10 adds two new ALTER TABLESPACE clauses that are related to storage groups. Example 4-13 shows these new clauses.

Example 4-13 REBALANCE SUSPEND clause (ALTER TABLESPACE statement)

```
ALTER TABLESPACE tablespace_name REBALANCE SUSPEND;
```

The REBALANCE SUSPEND clause (Example 4-14) will cause the table space rebalance operation (which is relocating a table space from the existing storage group to a new storage group as an online process) to be suspended until either of the following operations are performed:

- A resume rebalance statement is issued.
- The database is reactivated after being brought down with a table space rebalance in a suspended state.

Example 4-14 REBALANCE RESUME clause (ALTER TABLESPACE statement)

```
ALTER TABLESPACE tablespace_name REBALANCE RESUME;
```

The REBALANCE RESUME clause resumes the rebalance operation for a table space that previously had its rebalance operation suspended.

4.4 Summary

Multi-temperature storage provides an easy, online mechanism to move table spaces through various storage temperatures as application access requirements change over time. With this mechanism, an organization can tailor its annual storage purchases to the specific needs of the applications.

With the multiple storage group capability of DB2 10 automatic storage, an enterprise can optimize its storage purchases. This feature allows more effective usage of the storage budget to provide superior performance for critical data and lower cost storage for less critical data that is infrequently accessed.

As this chapter described, DB2 10 added ADMIN_GET_STORAGE_PATHS and MON_GET_REBALANCE_STATUS table functions to examine storage groups.

There is also a new catalog view, SYSCAT.STOGROUPS.

With the ease-of-use of multi-temperature storage, administration costs are minimized along with continued availability of the application during the storage group rebalance process.

5

Row and column access control

In this chapter, we describe the row and column access control (RCAC) capability of DB2 10. RCAC provides a mechanism to control data at a granular level within a table that is not possible by using standard relational database privileges. With RCAC, an organization can restrict individuals from accessing data that is not required for their job. This type of partial table data access is achieved through the row permissions portion of RCAC. In addition, specific column data can be masked with nulls (null characters), a user-defined mask, or a partial column mask to restrict sensitive data within a column from being viewed by unauthorized personnel. Another benefit of RCAC is that it restricts any user with DATAACCESS authority, including database administrators (DBAs), from accessing the data, preventing these individuals from viewing restricted or sensitive data. Other database vendors cannot provide this level of security from database privileged users without purchasing an additional product. With DB2 10 and RCAC, you can restrict privileged users as part of the database engine security mechanism.

5.1 Overview of RCAC

For some applications, the table-level access privileges that are provided by a relational database are not granular enough to provide the security required. In previous versions of DB2, granular data access privilege required either the creation of specialized views or the implementation of label-based access control (LBAC). Views create management overhead and might require application changes (because the view must be accessed instead of the table). LBAC requires labels to be created on the tables and columns that require access restrictions and then credentials on those tables must be granted to individual users or groups of users.

With the DB2 10 release, RCAC now allows the security administrator to provide permissions at the row level to provide granular row access within a table and masks at the column level to restrict the ability to view data within sensitive columns. These permissions and masks can be used together or singularly to provide the level of table access control that an application requires.

5.2 Business value

Over the past several years, a number of laws and industry-related compliance regulations were created to govern the availability of certain types of information within a database. One example of this new information governance is the Payment Card Institute (PCI) regulation that governs the use of credit card numbers by requiring this information to be encrypted when at rest within a database. Database encryption provides adequate protection of data at rest on the storage disk device. However, if a user or group is authorized to access the database and the tables that are stored within the database, the data is displayed in an unencrypted format. This requires another level of data protection as the data is accessed within a database. RCAC, available in DB2 10 with its permissions and masks, represents a second layer of security that complements the current table-privileges security model. Permissions within RCAC control specific rows within a table that a specific group of users can access. Masks within RCAC control how much data of a particular column can be viewed: the entire column, a part of the column, or none of the column. By using both permissions and masks, an organization can comply with information governance by allowing only specific groups of individuals to access rows of data within a table, and whether that group can view columns containing sensitive data.

The Great Toque Outdoors sales application stores customer information in the following tables:

- CUSTOMERS table contains permanent account information for any customer shopping with Great Toque Outdoors.
- CUST_CRDT_CARD table contains the credit card information about each customer that uses a credit card when shopping with Great Toque Outdoors for either retail sales or sales within the adventure travel division.

To meet the various corporate information governance policies, Great Toque Outdoors decided to restrict the credit card data to the accounting and sales departments. For general customer information, Great Toque Outdoors decided to mask sensitive personal information (such as marital status, age, and email contact information) and limit this information to the accounting and sales departments. To further restrict credit card information within the accounting department, each customer is assigned to a customer group, and Great Toque Outdoors accounting personnel are assigned to specific customer groups. By using these customer groups, only accounting personnel who are assigned to the same group as a customer can access credit card information about that particular customer. To further strengthen SALES database security in conjunction with using RCAC, Great Toque Outdoors is using the following additional security capabilities within DB2 to restrict DBAs:

- Data access (ability to view data) to all tables
- Access control (ability to GRANT database privileges) to all tables

In addition, no user of the SALES database can have implicit database connection capability through the use of PUBLIC. All database connection access must be explicitly granted to either a user ID, group, or database role.

After looking at the database capabilities of DB2 10, Great Toque Outdoors decided to use the row permissions and column masking capabilities of RCAC within DB2 to meet its information governance corporate rules.

5.3 Business application examples

Great Toque Outdoors corporate security read the latest “2010 Annual Study: Global Cost of a Data Breach” released by Symantec and the Ponemon Institute:

`http://www.symantec.com/about/news/resources/press_kits/detail.jsp?pkid=ponemon`

This study indicates that the worldwide average cost of a data breach to a company is four million US dollars ($4M), up 18% from the previous study in 2009. Both the cost of a breach and the indicators, signifying that frequency and costs of breaches would increase in the future, caused Great Toque to realize they needed to implement more granular data privileges for its sensitive customer information.

5.3.1 Row and column access control description and usage

Standard relational database security at the table level provides the ability to control only read and write operations against the data contained within the table. Although this level of security is adequate for some applications, it does not meet the data security needs that are required by industry and government regulations in existence today. Before DB2 10 and the availability of RCAC, the only method for more granular security within DB2 was label-based access control (LBAC). LBAC provides granular row and column access to data, but requires the creation of labels on rows and columns along with the granting of security credentials and is targeted more for government types of applications. RCAC provides fine grained access control of table data and is implemented by creating rules and then activating those rules against the table. The simpler implementation and ability to protect rows, columns, or both within a table make RCAC ideal for many applications that require security above what standard relational database privileges provide.

RCAC is defined at the individual table level within a database and provides two components to control data access. The first component is the permission that controls access to individual rows within a database table. Along with providing row level protection through permissions, the second component of RCAC is its ability to mask data for sensitive columns. RCAC has additional DB2 functions to allow data access to be controlled at the group (set up within the underlying operating system on the database server), database roles (created and granted within DB2 by the security administrator), or through a trusted context role that is assigned upon connection to the DB2 database. However, RCAC is not limited to restricting data access or masking data using these new functions. RCAC is flexible and allows other methods of controlling data access, such as global variables within DB2. Several examples of types of RCAC permissions that use other criteria are provided next.

The first example uses the CURRENT TIME global variable to restrict access to the PAYROLL table during normal business hours, as shown in Example 5-1.

Example 5-1 RCAC permission that restricts PAYROLL table access to normal business hours

```
CREATE PERMISSION payrollp
   ON PAYROLL
   FOR ROWS WHERE CURRENT TIME BETWEEN ‘8:00’ AND ’17:00’
   ENFORCED FOR ALL ACCESS ENABLE;
```

Now we extend the previous example to highlight a more complex access rule, where the access to the PAYROLL table is restricted to normal business hours, and must be using the Human Resources (HR) application identified by the stored procedure named HRPROCS.PROC1, as shown in Example 5-2.

Example 5-2 RCAC permission that restricts PAYROLL table access to normal business hours and only when accessed through the HR application stored procedure

```
CREATE PERMISSION payroll-table-rules
   ON PAYROLL
   FOR ROWS WHERE CURRENT TIME BETWEEN ‘8:00’ AND ’17:00’
      AND SYSIBM.ROUTINE_SPECIFIC_NAME = ‘PROC1’
      AND SYSIBM.ROUTINE_SCHEMA = ‘HRPROCS’
      AND SYSIBM.ROUTINE_TYPE = ‘P’
   ENFORCED FOR ALL ACCESS ENABLE;
```

RCAC allows fine-grained access control to be placed on specific rows within a table to restrict access to certain groups of users. RCAC also provides the ability to mask column data for sensitive columns when the data row is returned to the application. Furthermore, a combination of row and column access control is possible for applications requiring both types of data access control. In RCAC terminology, row level control is defined as a permission; column level control is defined as a mask. Within RCAC, permissions and masks are activated or deactivated on a particular table through a clause on the ALTER TABLE statement. One of the advantages of RCAC in DB2 10 is that RCAC permissions and masks apply to all users within the database, even those with DBADM authority. Therefore, RCAC can be used to restrict data access and prevent access to sensitive data even for privileged database users. To provide a better understanding of RCAC and how it works, in the next section, we show an example RCAC implementation for Great Toque Outdoors.

5.3.2 Adding row and column access control to a DB2 10 database

To meet these new corporate information governance guidelines, DB2 10 RCAC was implemented as described in the remainder of this section.

For the RCAC examples using the SALES database, the CUSTOMERS table definition is shown in Example 5-3; the CUST_CRDT_CARD table definition is shown in Example 5-4.

Example 5-3 CUSTOMER table definition

```
CREATE TABLE TOQUE.CUSTOMERS (
         CUST_CODE  INTEGER NOT NULL ,
         CUST_FIRST_NAME  VARCHAR(75) ,
         CUST_LAST_NAME  VARCHAR(90) ,
         CUST_ADDRESS1  VARCHAR(120) ,
         CUST_CITY  VARCHAR(90) ,
         CUST_PROV_STATE  VARCHAR(90) ,
         CUST_PROV_STATE_CODE  VARCHAR(30) ,
         CUST_POSTAL_ZONE  VARCHAR(30) ,
         CUST_COUNTRY_CODE  INTEGER ,
         CUST_EMAIL  VARCHAR(120) ,
         GENDER_CODE  INTEGER ,
         CUST_AGE  INTEGER ,
         MARITAL_STATUS_CODE  INTEGER,
         CUST_GROUP CHAR(10) NOT NULL WITH DEFAULT 'CUSTGROUPA')
IN TOQUE_TRAVEL INDEX IN TOQUE_INDEX2;

ALTER TABLE TOQUE.CUSTOMERS
   ADD PRIMARY KEY
      (CUST_CODE);
```

Example 5-4 CUST_CRDT_CARD table definition

```
CREATE TABLE TOQUE.CUST_CRDT_CARD  (
         CUST_CC_ID  INTEGER NOT NULL ,
         CUST_CODE  INTEGER ,
         CRDT_METHOD_CODE  INTEGER ,
         CUST_CC_NUMBER  CHAR(57) ,
         CUST_CC_SERV_CODE  INTEGER ,
         CUST_CC_EXP_DATE  TIMESTAMP,
         CUST_GROUP CHAR(10) NOT NULL WITH DEFAULT 'CUSTGROUPA')
       IN TOQUE_TRAVEL INDEX IN TOQUE_INDEX2;

ALTER TABLE TOQUE.CUST_CRDT_CARD
   ADD PRIMARY KEY
      ( CUST_CC_ID );
```

In the Great Toque Outdoors SALES database environment are three classifications of user groups. Figure 5-1 shows the groups and their members.

- Database administrators (DBAs)
- Sales department
- Accounting department

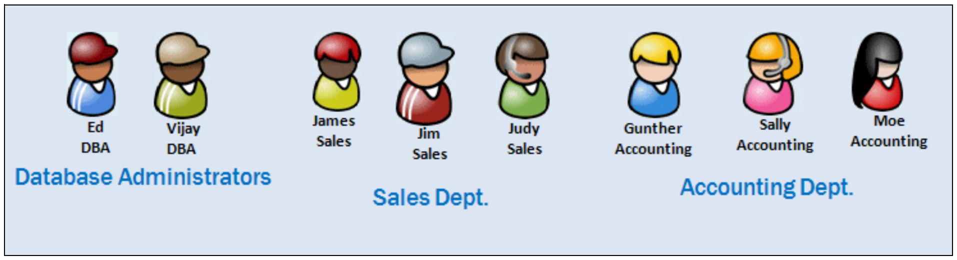


Figure 5-1 User IDs and roles within Great Toque Outdoors SALES database

Great Toque Outdoors decided on the following data access policy for the CUSTOMERS and CUST_CRDT_CARD tables:

- CUSTOMERS row level access: Only the Sales and Accounting department personnel are allowed access to the CUSTOMER table. DBAs are not allowed to access the table.
- CUST_CRDT_CARD row level access: Only the Sales department has global access to the CUST_CRDT_CARD table. For the Accounting department, each member of the Accounting department will be assigned a subset of customers. Only customers within the Accounting department member's customer subset can be accessed. DBAs are not allowed to access the table.
- CUSTOMERS column level access: No one has access to the Customer Group column, because this column is used to control row level access to the CUST_CRDT_CARD table. Only the Accounting department can access the remainder of the table columns data. Customer-sensitive data, age, and marital status are masked to members of the Sales department. Because DBAs cannot access the table rows, providing masks for this group is unnecessary.
- CUST_CRDT_CARD column level access: No one has access to the Customer Group column, because this column is used to control row level access to the CUST_CRDT_CARD table. Sales and Accounting departments can access all credit card information. In the case of the Accounting department, the table level permissions restrict the CUST_CRDT_CARD rows to those rows in the same CUST_GROUP as the role granted to the Accounting department team member. Because DBAs cannot access the table rows, there is no need to provide masks for this group.

To implement the requirement of Great Toque Outdoors that the Accounting department members can access customer data only for customers who are assigned to their customer group, both an operating system (OS) group and a database role are defined for the Accounting department. DB2 database security allows users to be assigned to both OS groups and database roles; this combination provides the granularity that is required to implement the required security policy of Great Toque Outdoors for the CUSTOMERS and CUST_CRDT_CARD tables.

First, an OS group named ACCOUNTING is created at the OS security level and all user IDs within the Accounting department (for example, Gunther, Sally, and Moe) are granted membership within this OS group. The next step is to create the database roles (DBA, SALES, CUSTGROUPA, CUSTGROUPB, and CUSTGROUPC). The OS group and database role assignments by user ID are shown in Table 5-1.

Table 5-1 SALES database group (OS level) and role (Database Level) assignments

User ID	ACCOUNTING (group)	CUSTGROUPA (role)	CUSTGROUPB (role)	CUSTGROUPC (role)	SALES (role)	DBA (role)
Ed						✓
Vijay						✓
James					✓	
Jim					✓	
Judy					✓	
Gunther	✓	✓				
Sally	✓		✓			
Moe	✓			✓		

Now that the OS group and database role assignments are in place, and also the additional CUST_GROUP column in both the CUSTOMERS and CUST_CRDT_CARD tables, the RCAC table permissions and column masks are defined and activated.

The syntax to implement Great Toque Outdoors table level permissions to control row access for the CUSTOMERS table is shown in Example 5-5. The syntax to control row access for the CUST_CRDT_CARD table is shown in Example 5-6. After the permissions are created, the permissions are activated for the table using the ALTER TABLE statement.

Example 5-5 Row level permissions for the CUSTOMERS table

```
CREATE PERMISSION custa ON TOQUE.CUSTOMERS
   FOR ROWS WHERE
      (VERIFY_ROLE_FOR_USER(SESSION_USER,'SALES')=1)
      OR
      (VERIFY_GROUP_FOR_USER(SESSION_USER,'ACCOUNTING')=1)
   ENFORCED FOR ALL ACCESS ENABLE;

ALTER TABLE TOQUE.CUSTOMERS ACTIVATE ROW ACCESS CONTROL;
```

Example 5-6 Row level permissions for the CUST_CRDT_CARD table

```
CREATE PERMISSION crdta ON TOQUE.CUST_CRDT_CARD
   FOR ROWS WHERE
      (VERIFY_ROLE_FOR_USER(SESSION_USER,'SALES')=1)
      OR
      (VERIFY_GROUP_FOR_USER(SESSION_USER,'ACCOUNTING')=1 AND
         VERIFY_ROLE_FOR_USER(SESSION_USER,CUST_GROUP)=1)
   ENFORCED FOR ALL ACCESS ENABLE;

ALTER TABLE TOQUE.CUST_CRDT_CARD ACTIVATE ROW ACCESS CONTROL;
```

In Example 5-6, within the CREATE PERMISSION statement, the combination of group and role assignments for Accounting department members is used to allow row access to Accounting department members, if their individual Customer Group assignment (CUSTGROUPA, CUSTGROUPB, or CUSTGROUPC) matches the CUST_GROUP value of the data row.

After the table permissions are activated, the table rows are protected. If the various groups that are allowed to access the rows are entitled to see all columns within the table, no further RCAC rules are required.

To illustrate the difference between row level permission only and the combination of a row permission and a column mask, see Example 5-7. That example shows the results of a query by Gunther of the Accounting department against the CUSTOMERS and CUST_CRDT_CARD tables with only row level permission enabled. This result is compared with combined table permission and mask configuration later in this chapter (Example 5-13 on page 81).

Example 5-7 Query results against CUSTOMERS and CUST_CRDT_CARD with table permission (row level) only for Gunther in Accounting department

```
SQL statement syntax (returns all customers whose last name begins with the letter
“T” and have a credit card entry in CUST_CRDT_CARD table):

SELECT B.CUST_FIRST_NAME AS FIRST_NAME, B.CUST_LAST_NAME AS LAST_NAME,
   A.CUST_CC_NUMBER AS CC_NUMBER,A.CUST_CC_EXP_DATE AS EXP_DATE,       A.CUST_GROUP
AS GROUP
   FROM TOQUE.CUST_CRDT_CARD A, TOQUE.CUSTOMERS B
   WHERE A.CUST_CODE = B.CUST_CODE AND B.CUST_LAST_NAME LIKE 'T%';

Query results for Gunther in the Accounting Department with CUST_GROUP=CUSTGROUPA
(EXP_DATE displayed as date for readability):

FIRST_NAME    LAST_NAME     CC_NUMBER         EXP_DATE   GROUP
------------  ------------- -------------     ---------- ------------
Mila          Thornton      4998765422607287  2009-10-01 CUSTGROUPA
Melissa       Torval        5198765468918809 2009-01-01 CUSTGROUPA
Ranulfo       Taveres Ifran 9998765401974186 2012-06-01 CUSTGROUPA
Sakura        Tamura        4998765494706199 2012-06-01 CUSTGROUPA
Akane         Tsuji         9998765410988961 2009-07-01 CUSTGROUPA
-------remaining 354 records not shown------------------------------
```

The next step in the RCAC implementation for the CUSTOMERS and CUST_CRDT_CARD tables is the creation of the table masks for each table to protect the sensitive columns within each table. Example 5-8 shows the CREATE MASK syntax for the CUSTOMERS table. Four masks are defined on the CUSTOMERS table.

Example 5-8 CREATE MASK syntax for the CUSTOMERS table

```
CREATE MASK cust_group_mask ON TOQUE.CUSTOMERS
   FOR COLUMN cust_group RETURN
      CASE WHEN (VERIFY_ROLE_FOR_USER(SESSION_USER,'SECURITY')=1)
           THEN cust_group
         ELSE 'NONE'
      END
   ENABLE;

CREATE MASK cust_age_mask ON TOQUE.CUSTOMERS
   FOR COLUMN cust_age RETURN
      CASE WHEN (VERIFY_GROUP_FOR_USER(SESSION_USER,'ACCOUNTING')=1)
           THEN cust_age
        WHEN (VERIFY_ROLE_FOR_USER(SESSION_USER,'SALES')=1)
           THEN 21
        ELSE NULL
      END
   ENABLE;

CREATE MASK cust_email_mask ON TOQUE.CUSTOMERS
   FOR COLUMN cust_email RETURN
      CASE WHEN (VERIFY_GROUP_FOR_USER(SESSION_USER,'ACCOUNTING')=1)
           THEN cust_email
        WHEN (VERIFY_ROLE_FOR_USER(SESSION_USER,'SALES')=1)
           THEN cust_email
        ELSE NULL
      END
   ENABLE;

CREATE MASK marital_status_mask ON TOQUE.CUSTOMERS
   FOR COLUMN marital_status_code RETURN
      CASE WHEN (VERIFY_GROUP_FOR_USER(SESSION_USER,'ACCOUNTING')=1)
           THEN marital_status_code
         ELSE NULL
      END
   ENABLE;

ALTER TABLE TOQUE.CUSTOMERS ACTIVATE COLUMN ACCESS CONTROL;
```

Tips:

- RCAC masks must conform to the underlying data type of the column being masked. However, DB2 does not flag an improper data type mask as an error on the CREATE MASK statement itself. As an example, if the CUST_AGE column mask in the CUST_AGE_MASK in Example 5-8 on page 75 had its value set to “ADULT” instead of the integer value of 21, the CREATE MASK statement will succeed without error. However, subsequent queries against the CUSTOMERS table by anyone in the SALES role will return an error because ADULT is not an integer value.
- Although permissions and masks are created on individual database tables, the CREATE PERMISSION and CREATE MASK names must be unique in the database. Therefore, the preferred practice is to include elements of both the column name and table name to ensure the permission and mask names are unique within the database. If you create a mask called “LAST_NAME_MASK” for the LAST_NAME column and the same column name exists in more than one table, the new CREATE MASK definition will fail. Always check the CREATE PERMISSION and CREATE MASK statement return codes to ensure all permissions and masks were created successfully.
- If a NULL value is being returned for a column value as part of a CREATE MASK statement (as Example 5-8 shows with MARITAL_STATUS_MASK), the underlying column (MARITAL_STATUS_CODE) must allow NULL values. Otherwise, the CREATE MASK statement will fail. If a column does not allow NULL values, the return value must be a character string or number value. In Example 5-8 on page 75 the CREATE MASK statement for `cust_group_mask` returns a value of `'NONE'` (the CUST_GROUP column is defined as NOT NULL).

Example 5-9 shows the CREATE MASK syntax for the CUST_CRDT_CARD table. This example highlights one of the advantages of RCAC over the capabilities of some database competitors: flexible column masking. The CUST_GROUP column is masked to display a constant value, `'NONE'` during any access to the CUST_CRDT_CARD table by any user that is not a member of the SECURITY database role.

Example 5-9 CREATE MASK syntax for the CUST_CRDT_CARD table

```
CREATE MASK cust_cc_group_mask ON TOQUE.CUST_CRDT_CARD
   FOR COLUMN cust_group RETURN
      CASE WHEN (VERIFY_ROLE_FOR_USER(SESSION_USER,'SECURITY')=1)
           THEN cust_group
        ELSE 'NONE'
      END
   ENABLE;

ALTER TABLE TOQUE.CUST_CRDT_CARD ACTIVATE COLUMN ACCESS CONTROL;
```

Now that the row permissions and column masks are defined for both the CUSTOMERS and CUST_CRDT_CARD tables, several sample queries are executed for various users within Great Toque Outdoors to show behavior of RCAC.

Before we show the query results for various users within Great Toque Outdoors, Example 5-10 shows the queries that are executed for each user.

Example 5-10 The SQL statements to test the RCAC permissions and masks created

```
Query #1 - CUSTOMERS table query that returns all customers that live
in a city that begins with the letter “P”:
SELECT CUST_FIRST_NAME AS FIRST_NAME, CUST_LAST_NAME AS LAST_NAME,
   CUST_CITY AS CITY, CUST_AGE AS AGE, CUST_GROUP AS GROUP
   FROM TOQUE.CUSTOMERS WHERE CUST_CITY LIKE 'P%';

Query #2 - A join of the CUSTOMERS and CUST_CRDT_CARD tables that
returns all customers whose last name begin with the letter “T”:
SELECT B.CUST_FIRST_NAME AS FIRST_NAME, B.CUST_LAST_NAME AS LAST_NAME,
   A.CUST_CC_NUMBER AS CC_NUMBER,A.CUST_CC_EXP_DATE AS EXP_DATE,
      A.CUST_GROUP AS GROUP
   FROM TOQUE.CUST_CRDT_CARD A, TOQUE.CUSTOMERS B
   WHERE A.CUST_CODE = B.CUST_CODE AND B.CUST_LAST_NAME LIKE 'T%';
```

The queries referenced in Example 5-10 on page 77 will now be executed against the SALES database for the user IDs listed in Table 5-2.

Table 5-2 User IDs with roles and groups used for test query executions

User ID	Role	OS group
ED	DBA	None
JAMES	SALES	None
GUNTHER	CUSTGROUPA	ACCOUNTING
SALLY	CUSTGROUPB	ACCOUNTING
MOE	CUSTGROUPC	ACCOUNTING

We now execute the test queries for the user population that is specified. We provide the expected outcome and then the actual query results to validate that the RCAC permissions and masks were implemented correctly and meet the application requirements of Great Toque Outdoors for the CUSTOMERS and CUST_CRDT_CARD tables.

The RCAC queries begin with execution by Ed, who is a member of the DBA role within Great Toque Outdoors. Based on the application requirements and the RCAC rules implemented, we expect to see Ed's queries return no rows for either query. The reason is because the DBA role was granted "no access" through the RCAC rules on the CUSTOMERS and CUST_CRDT_CARD tables.
Example 5-11 on page 79 shows the results of Query #1 and Query #2 for Ed, whose role is DBA.

Note: The GROUP column listed in the output for the query results in the remainder of this chapter for Query #1 and Query #2 refers to the CUST_GROUP column in the TOQUE.CUSTOMERS and TOQUE.CUST_CRDT_CARD tables, not an OS group. Accounting department members have a DB2 role assignment that matches this CUST_GROUP column.

Example 5-11 Query results for Ed with the DBA role assignment

```
SELECT CUST_FIRST_NAME AS FIRST_NAME, CUST_LAST_NAME AS LAST_NAME,
   CUST_CITY AS CITY, CUST_AGE AS AGE, CUST_GROUP AS GROUP
   FROM TOQUE.CUSTOMERS
   WHERE CUST_CITY LIKE 'P%';

Results:
FIRST_NAME    LAST_NAME        CITY             AGE      GROUP
------------  -------------    -------------    ------   ----------
0 record(s) selected.

SELECT B.CUST_FIRST_NAME AS FIRST_NAME, B.CUST_LAST_NAME AS LAST_NAME,
   A.CUST_CC_NUMBER AS CC_NUMBER,A.CUST_CC_EXP_DATE AS EXP_DATE,
   A.CUST_GROUP AS GROUP
   FROM TOQUE.CUST_CRDT_CARD A, TOQUE.CUSTOMERS B
   WHERE A.CUST_CODE = B.CUST_CODE AND B.CUST_LAST_NAME LIKE 'T%';

Results (NOTE: EXP_DATE displayed as date for readability):
FIRST_NAME    LAST_NAME     CC_NUMBER        EXP_DATE   GROUP
------------  ------------- -------------    ---------- -------
0 record(s) selected.
```

The results of the queries executed by Ed, as shown in Example 5-11, are as expected; no rows returned for either query.

Because James is a member of the SALES role, he has access to all data rows and columns, except the CUST_GROUP column ("GROUP" in the query results) and the CUST_AGE column ("AGE" in the query results) in Example 5-12 on page 80. Therefore the CUST_GROUP values are displayed as "NONE", which was the mask defined for this column. In addition, the CUST_AGE value is set to 21, because this was the mask value constant defined.

Example 5-12 Query results for James with the SALES role assignment

```
SELECT CUST_FIRST_NAME AS FIRST_NAME, CUST_LAST_NAME AS LAST_NAME,
   CUST_CITY AS CITY, CUST_AGE AS AGE, CUST_GROUP AS GROUP
   FROM TOQUE.CUSTOMERS WHERE CUST_CITY LIKE 'P%';

Results:
FIRST_NAME    LAST_NAME        CITY             AGE     GROUP
------------  -------------    -------------    ------  ----------
Silvio        Benzi            Perugia          21      NONE
Flora         Alegre           Porto Alegre     21      NONE
Iliana        Hermida          Puerto Vallarta  21      NONE
Brianca       Baade            Puebla           21      NONE
Fausto        Soto Peña        Puebla           21      NONE
-------remaining 1759 records not shown-----------------------

SELECT B.CUST_FIRST_NAME AS FIRST_NAME, B.CUST_LAST_NAME AS LAST_NAME,
   A.CUST_CC_NUMBER AS CC_NUMBER,A.CUST_CC_EXP_DATE AS EXP_DATE,
   A.CUST_GROUP AS GROUP
   FROM TOQUE.CUST_CRDT_CARD A, TOQUE.CUSTOMERS B
   WHERE A.CUST_CODE = B.CUST_CODE AND B.CUST_LAST_NAME LIKE 'T%';

Results (NOTE: EXP_DATE displayed as date for readability):
FIRST_NAME    LAST_NAME     CC_NUMBER         EXP_DATE   GROUP
------------  ------------- -------------     ---------- -------
Mila          Thornton      4998765422607287  2009-10-01  NONE
Melissa       Torval        5198765468918809 2009-01-01  NONE
Ranulfo       Taveres Ifran 9998765401974186 2012-06-01  NONE
Sakura        Tamura        4998765494706199 2012-06-01  NONE
Akane         Tsuji         9998765410988961 2009-07-01  NONE
----remaining 1064 records not shown--------------------------
```

Gunther is in the ACCOUNTING group and is a member of the CUSTGROUPA role. Therefore, according to the permissions and masks defined on the CUSTOMERS and CUST_CRDT_CARD tables, he can see all columns except for the CUST_GROUP column value in both tables. As Example 5-13 on page 81 shows, because of the CUST_CRDT_CARD table permissions, the Accounting department members can see only rows with the same CUST_GROUP value as their DB2 role name (CUSTGROUPA in the case of Gunther). Therefore, for Query #2, Gunther's query results returned only a total of 359 rows versus the 1069 rows that were returned for the same query by James of the Sales department. The difference between the SALES role and the ACCOUNTING group is that Gunther can view the actual CUST_AGE (AGE in the results) value in Query #1, because the CUSTOMERS table mask definitions allow ACCOUNTING group access to this column.

Example 5-13 Query results for Gunther with the ACCOUNTING group and CUSTGROUPA role assignments

```
SELECT CUST_FIRST_NAME AS FIRST_NAME, CUST_LAST_NAME AS LAST_NAME,
   CUST_CITY AS CITY, CUST_AGE AS AGE, CUST_GROUP AS GROUP
   FROM TOQUE.CUSTOMERS
   WHERE CUST_CITY LIKE 'P%';

Results:
FIRST_NAME    LAST_NAME       CITY            AGE     GROUP
------------  -------------   -------------   ------  ----------
Silvio        Benzi           Perugia         22      NONE
Flora         Alegre          Porto Alegre    39      NONE
Iliana        Hermida         Puerto Vallarta 40      NONE
Brianca       Baade           Puebla          29      NONE
Fausto        Soto Peña       Puebla          29      NONE
-------remaining 1759 records not shown-----------------------

SELECT B.CUST_FIRST_NAME AS FIRST_NAME, B.CUST_LAST_NAME AS LAST_NAME,
   A.CUST_CC_NUMBER AS CC_NUMBER,A.CUST_CC_EXP_DATE AS EXP_DATE,
   A.CUST_GROUP AS GROUP
   FROM TOQUE.CUST_CRDT_CARD A, TOQUE.CUSTOMERS B
   WHERE A.CUST_CODE = B.CUST_CODE AND B.CUST_LAST_NAME LIKE 'T%';

Results (NOTE: EXP_DATE displayed as date for readability):
FIRST_NAME    LAST_NAME     CC_NUMBER        EXP_DATE   GROUP
------------  ------------- -------------    ---------- -------
Mila          Thornton      4998765422607287 2009-10-01  NONE
Melissa       Torval        5198765468918809 2009-01-01  NONE
Ranulfo       Taveres Ifran 9998765401974186 2012-06-01  NONE
Sakura        Tamura        4998765494706199 2012-06-01  NONE
Akane         Tsuji         9998765410988961 2009-07-01  NONE
-------remaining 354 records not shown-----------------------
```

To better illustrate the difference between row permissions and column masks, we compare the results in Example 5-7 on page 74 (with row permissions only) and the results in Example 5-13 (with both row permissions and column masks). As you can see from the results for Gunther for Query #2 (Example 5-14 on page 82), there were 359 rows returned in both cases. Column level masks do not affect the rows that are returned for a query; their only effect is how the columns are displayed to the user executing the query. In the case of Gunther and other members of the Accounting department, the only column mask that affects the result is the mask on the CUST_GROUP column. For the Accounting department group, this column mask returns “NONE” for the CUST_GROUP column because the mask is a constant value mask. The first row that is returned for each query is shown in Example 5-14, to indicate the effect of the CUST_GROUP mask for the second time Gunther runs Query #2.

Example 5-14 Gunther Query #2 for Result #1 and Result #2

```
SELECT B.CUST_FIRST_NAME AS FIRST_NAME, B.CUST_LAST_NAME AS LAST_NAME,
   A.CUST_CC_NUMBER AS CC_NUMBER,A.CUST_CC_EXP_DATE AS EXP_DATE,
   A.CUST_GROUP AS GROUP
   FROM TOQUE.CUST_CRDT_CARD A, TOQUE.CUSTOMERS B
   WHERE A.CUST_CODE = B.CUST_CODE AND B.CUST_LAST_NAME LIKE 'T%';

Result #1 (table permission only):
FIRST_NAME    LAST_NAME     CC_NUMBER         EXP_DATE   GROUP
------------  ------------- -------------     ---------- ------------
Mila          Thornton      4998765422607287  2009-10-01 CUSTGROUPA

Result #2 (table permission and masks):
FIRST_NAME    LAST_NAME     CC_NUMBER         EXP_DATE   GROUP
------------  ------------- -------------     ---------- -------
Mila          Thornton      4998765422607287 2009-10-01  NONE
```

Sally is in the Accounting department with Gunther. Sally has a role assignment of CUSTGROUPB; Gunther has a role assignment of CUSTGROUPA. As expected, the Query #1 results are identical with respect to the customers returned (all located in cities starting with the letter “P”). See Example 5-15 on page 83.

Because the CUST_CRDT_CARD table has a permission that is based on the group and role assignments for the Accounting department (as shown in Example 5-15 on page 83), the fact that Gunther and Sally are assigned different roles does matter for Query #2. The row permissions in effect on the TOQUE.CUST_CRDT_CARD table will cause only the rows, where the CUST_GROUP column is equal to the role defined for members of the Accounting department, to be returned. In addition, the column masks prevent the CUST_GROUP column values from being displayed (returned as NONE for all Accounting employees). Query #2 returns 359 rows when Gunther

(CUSTGROUPA) executes the query. When Sally (CUSTGROUPB) executes the same query, 326 rows are returned because of the different customer groups.

Example 5-15 Query results for Sally with the ACCOUNTING group and CUSTGROUPB role assignments

```
SELECT CUST_FIRST_NAME AS FIRST_NAME, CUST_LAST_NAME AS LAST_NAME,
   CUST_CITY AS CITY, CUST_AGE AS AGE, CUST_GROUP AS GROUP
   FROM TOQUE.CUSTOMERS WHERE CUST_CITY LIKE 'P%';

Results:
FIRST_NAME    LAST_NAME        CITY             AGE     GROUP
------------  -------------    -------------    ------  ----------
Silvio        Benzi            Perugia          22      NONE
Flora         Alegre           Porto Alegre     39      NONE
Iliana        Hermida          Puerto Vallarta  40      NONE
Brianca       Baade            Puebla           29      NONE
Fausto        Soto Peña        Puebla           29      NONE
-------remaining 1759 records not shown-----------------------

SELECT B.CUST_FIRST_NAME AS FIRST_NAME, B.CUST_LAST_NAME AS LAST_NAME,
   A.CUST_CC_NUMBER AS CC_NUMBER,A.CUST_CC_EXP_DATE AS EXP_DATE,
   A.CUST_GROUP AS GROUP
   FROM TOQUE.CUST_CRDT_CARD A, TOQUE.CUSTOMERS B
   WHERE A.CUST_CODE = B.CUST_CODE AND B.CUST_LAST_NAME LIKE 'T%';

Results (NOTE: EXP_DATE displayed as date for readability):
FIRST_NAME    LAST_NAME     CC_NUMBER        EXP_DATE   GROUP
------------  ------------- -------------    ---------- -------
Heena         Thackrey      5398765488174076 2009-04-01  NONE
Marc          Talbot        4998765434869081 2012-01-01  NONE
David         Tejada        4998765407318868 2010-02-01  NONE
Fumie         Tsuru         5498765469905307 2010-01-01  NONE
Randall       Turner        5398765463008687 2012-04-01  NONE
-------remaining 321 records not shown-----------------------
```

Now we look at Moe, also an Accounting department member, to see what the results are for the same queries with the role set to CUSTGROUPC. These results are in Example 5-16.

Example 5-16 Query results for Moe with the ACCOUNTING group and CUSTGROUPB role assignments

```
SELECT CUST_FIRST_NAME AS FIRST_NAME, CUST_LAST_NAME AS LAST_NAME,
   CUST_CITY AS CITY, CUST_AGE AS AGE, CUST_GROUP AS GROUP
   FROM TOQUE.CUSTOMERS
   WHERE CUST_CITY LIKE 'P%';

Results:
FIRST_NAME    LAST_NAME        CITY             AGE     GROUP
------------  -------------    -------------    ------  ----------
Silvio        Benzi            Perugia          22      NONE
Flora         Alegre           Porto Alegre     39      NONE
Iliana        Hermida          Puerto Vallarta  40      NONE
Brianca       Baade            Puebla           29      NONE
Fausto        Soto Peña        Puebla           29      NONE
-------remaining 1759 records not shown------------------------

SELECT B.CUST_FIRST_NAME AS FIRST_NAME, B.CUST_LAST_NAME AS LAST_NAME,
   A.CUST_CC_NUMBER AS CC_NUMBER,A.CUST_CC_EXP_DATE AS EXP_DATE,
   A.CUST_GROUP AS GROUP
   FROM TOQUE.CUST_CRDT_CARD A, TOQUE.CUSTOMERS B
   WHERE A.CUST_CODE = B.CUST_CODE AND B.CUST_LAST_NAME LIKE 'T%';

Results (NOTE: EXP_DATE displayed as date for readability):
FIRST_NAME    LAST_NAME     CC_NUMBER         EXP_DATE   GROUP
------------  ------------- -------------     ---------- -------
Yuta          Takiguchi     379876546597531   2009-07-01  NONE
Pernille      Trier         5498765447558376 2012-02-01  NONE
Samuel        Tupã Gabizo   4998765488452677 2013-10-01  NONE
Kazuki        Tsumura       9998765432353038  2012-11-01  NONE
Daichi        Tsuji         9998765440971623  2009-05-01  NONE
-------remaining 379 records not shown------------------------
```

As Example 5-16 shows, Moe receives the same results for Query #1 as other members of the Accounting department. For Query #2, the role assignment (CUSTGROUPC for Moe) results in 384 rows returned, compared to 326 rows for Sally and 359 rows for Gunther.

5.4 Summary

The examples in this chapter show how RCAC can be used to both restrict access to data rows and to mask sensitive column data for a single column or multiple columns within the tables of a DB2 database. RCAC is a powerful new security mechanism that is straightforward to implement in a DB2 10 environment, and provides the granular row and column access capabilities that are required to meet the application data security requirements of today.

DB2 10 has the following new functions for RCAC:

- VERIFY_ROLE_FOR_USER
- VERIFY_GROUP_FOR_USER
- VERIFY_TRUSTED_CONTEXT_FOR_USER

With the flexibility of using the new functions combined with system global variables, RCAC provides the data access granularity and security that regulatory compliance requirements demand today.

6

Availability enhancements

In this chapter, we describe the improved availability features and enhancements to disaster recovery.

The demand to have your system available 24x7 is always increasing. DB2 10 continues the availability offerings of prior DB2 versions in the areas of high availability, disaster recovery, and continuous availability, and meets this challenge in two ways:

- High availability disaster recovery (HADR): multiple standby databases
- Continuous availability: DB2 pureScale Feature

DB2 10 delivers on an ongoing dedication to providing database availability, even during a region-wide disaster. Support of multiple standby databases helps provide near-continuous availability and data recovery in one technology. This technology eliminates the need to configure a replication-based solution that was required in earlier HADR implementations to support both HA and DR.

The DB2 pureScale Feature provides continuous availability and scalability as required, and is transparent to applications.

Great Toque Outdoors has various needs for its various business divisions. They have retail stores located in several North American, South American, and European cities. Its adventure travel division is primarily an Internet presence. The divisions are run separately and are therefore capable of exploiting various DB2 functions.

6.1 HADR multiple standby databases

With high availability, the database is almost always able to service requests. With disaster recovery, data is protected in case of a catastrophic outage.

The high availability disaster recovery (HADR) component of earlier DB2 versions is enhanced in DB2 10 to provide multiple standby databases, and the ability to include a time-delay on log-replay to allow a recovery window from application or user error. This time-delay eliminates the need to resort to a full or partial database-restore operation. With multiple standby databases, up to three standbys are supported:

- One principal standby database
- Two auxiliary standby databases

Having multiple standby databases eliminates the need to configure a replication-based solution that was required in earlier HADR implementations to support both high availability (HA) and disaster recovery (DR). With multiple standby support, both HA and DR are provided in a single technology, transparent to applications.

- High availability requirements are met by having one standby database (the principal standby database) located at the same site (or a nearby location), running in close synchronization with the database for timely, automated ultra-fast failover in the event of an outage.
- Disaster recovery requirements are met by having one or two additional standby databases (the auxiliary standby databases) situated in a remote site, providing protection from total loss at the primary site.

With the DB2 10 time-delayed replay feature, you can keep an auxiliary standby log replay running behind the primary database, providing time to recover from application errors that might cause data loss on the primary database. In addition, all standby databases support the HADR *reads on standby* feature, and support both forced and unforced takeovers.

6.1.1 Business value

Although the retail stores and the adventure travel division of Great Toque Outdoors both require 24x7 availability, their tolerance for availability delays differs. With worldwide retail stores, there is always someone buying something somewhere, but the retail stores can tolerate an outage of a few minutes. This means that the functionality that is provided by multiple standby databases (with the HADR delayed replay feature) will be implemented by the retail stores.

The current model of Great Toque Outdoors retail stores poses several immediate problems:

- With retail stores located all over North America, South America, and Europe, there is no such thing as off-peak hours. There is always a store somewhere in the world making sales and affecting inventory. However, a brief break in availability can be tolerated.
- Although a brief break in availability can be tolerated, the retail stores must also be safeguarded against an outage or disaster, even one that is region-wide. Great Toque Outdoors retail stores must have disaster recovery protection in case there is a total loss of the primary database.
- The retail stores require protection against application errors, and must avoid data loss caused by an errant transaction. Although Great Toque Outdoors retail stores need both high availability and disaster recovery protection, they cannot afford the additional workload required to recover from an errant transaction. For example, accidentally deleting valuable data, which would require the database administrators (DBAs) to restore an earlier backup image to a point in time that is immediately before the errant transaction occurred. Costly, both in administration time and system down time. Great Toque Outdoors retail stores need to be protected against extraneous transactions.

When assessing DB2 10, Great Toque Outdoors soon realized that the functionality of multiple standby databases could address the company's problems and protect its data, while also keeping it highly available. With multiple standbys, the company can guard against a scenario in which a region-wide outage or disaster brings down both the primary and a standby database. Great Toque Outdoors can set up a multiple standby system with a primary database and a principal standby database in the same physical location, and two additional auxiliary standby databases at separate physical locations a long distance away, as shown in Figure 6-1 on page 90.

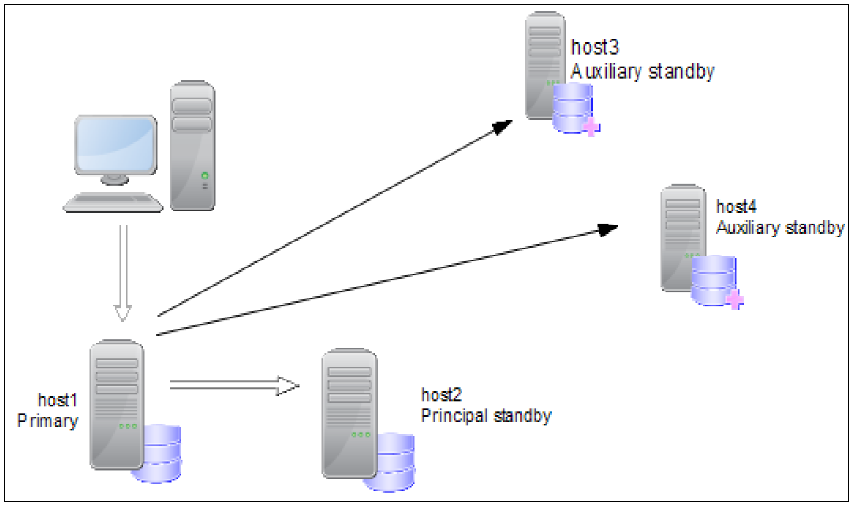


Figure 6-1 Sample multiple HADR setup with the maximum three standby databases

In addition, specifying a delayed replay on one of the auxiliary standby databases provides protection against application errors. The data is protected by specifying a delay for applying changes on the standby database. This delays replaying of the logs, intentionally keeping the standby database at a point in time that is earlier than that of the primary database. This approach guarantees that transactions are not committed on the standby database until the replay delay has elapsed. As a result, the DBAs have time to detect and handle the problem, and a window of recoverability from errant transactions on the primary database is provided. The DBAs can also choose to enable log-spooling, which allows the standby database to receive more logs and provides greater protection against data loss in case of failure on the primary database. However, the standby database chosen to have delayed replay cannot take over as the primary database until delayed replay is disabled.

When assessing DB2 10, the company also realized that other enhancements provide better utilization of standby-specific features:

- The *reads on standby* feature (introduced in DB2 9.7 Fix Pack 1) directs read-only workloads to a standby database. Read-only workloads are non-logged operations. Multiple standby databases can support a larger read-only workload, which increases performance.
- With HADR rolling updates, you can do updating without downtime. With multiple standby databases, availability is maintained as the updates roll through the standby databases.

6.1.2 Business application examples

For its retail stores, high availability is crucial for Great Toque Outdoors. But the company also requires a disaster recovery strategy, and the ability to avoid data loss caused by errant transactions. HADR multiple standby databases meets both of these requirements.

Great Toque Outdoors retail stores need both high availability and disaster recovery protection, so they decide to use the maximum number of standby databases (one principal standby, and two auxiliary standbys). The principal standby database will be in close synchronization with the primary database. To ensure protection in the case of a disaster, the two auxiliary standby databases will be in remote locations. The additional cost of these databases is justified by the data protection provided by setting up one to use the time-delayed replay feature, and the other is set up to use the *reads on standby* feature.

The company headquarters and warehouse are located in Boston, Massachusetts, US, so the primary database and principal standby database will be in Boston. To protect against an outage or disaster affecting only the Boston area, an auxiliary standby database will be located 700 kilometers away in Toronto, Canada. To protect against an outage or significant disaster affecting a larger area, a second auxiliary standby database will be 4000 kilometers from Boston, in Seattle, Washington, US. The auxiliary standby database in Seattle will also be the database with time-delayed replay enabled.

The setup for the company is shown in Figure 6-2 on page 92, and has the following hosts:

- *host1* is the primary database at headquarters in Boston
- *host2* is the principal standby database at the warehouse in Boston
- *host3* is the auxiliary standby database in Toronto
- *host4* is the auxiliary standby database in Seattle

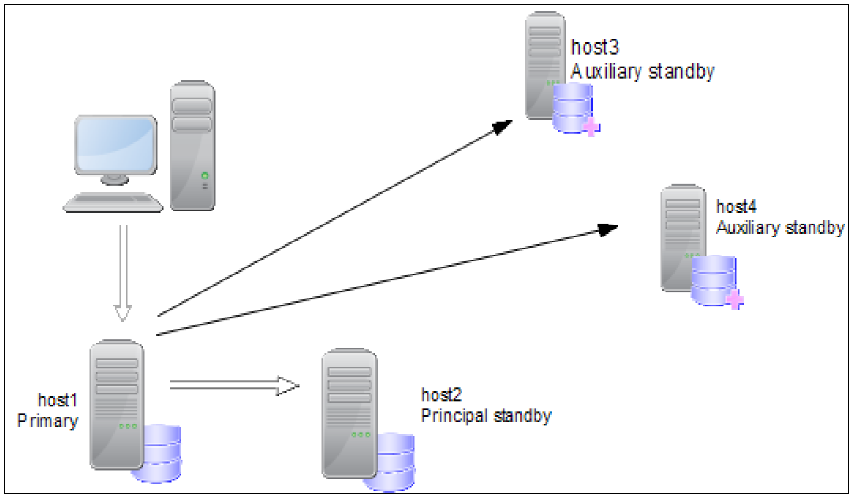


Figure 6-2 Great Toque Outdoors multiple standby database setup

After deciding to incorporate functionality of the multiple standby databases into its retails stores, Great Toque Outdoors implements this in the following phases:

1. Establish values for the HADR configuration parameters.
2. Set up the multiple standby system.
3. Configure delayed replay.

Establishing values for HADR configuration parameters

To set up a multiple standby environment, Great Toque Outdoors must first establish values for seven HADR configuration parameters. The following values must be established for all four databases (the primary database and the three standby databases):

- hadr_local_host
- hadr_local_svc
- hadr_remote_host
- hadr_remote_svc
- hadr_remote_inst
- hadr_syncmode
- hadr_target_list

DBAs use the following steps:

1. They establish the local parameter values for all four databases. The local parameter values help establish the remote parameter values; “remote” as used here means in terms of one of the other databases.
2. They choose the synchronization mode.
3. They derive the target list (`hadr_target_list`) by concatenating values from other configuration parameters.

To begin, the DBAs must determine the following values for all four databases in the HADR setup (Table 6-1):

- ► Host name
- ► Port number
- ► Instance name

Table 6-1 Host name, port number, and instance name for all four databases

Intended role	Host name (`hadr_local_host`)	Port number (`hadr_local_svc`)	Instance name	Notes
Primary	boston1	10	dbinst1	Primary database is located at company headquarters in Boston, Massachusetts, US.
Principal standby	boston2	40	dbinst2	Principal standby database is located in inventory warehouse in another area of Boston.
Auxiliary standby	toronto	41	dbinst3	Auxiliary standby database is located 700 km away in Toronto, Ontario, Canada.
Auxiliary standby	seattle	42	dbinst4	Auxiliary standby database located 4000 km away in Seattle, Washington, US. This auxiliary standby database will also have delayed replay enabled.

From this table, the local configuration parameters are already defined:

- ► Host name correlates to `hadr_local_host`
- ► Port number correlates to `hadr_local_svc`

The parameter values for the standby databases are also derived from Table 6-1 on page 93.

With the local configuration parameters established, the remote configuration parameters can be established.

On the primary database, the remote parameters are established from the HADR configuration parameters that correspond to the host name, port number, and instance name settings of the principal standby database, as shown in Table 6-2.

Table 6-2 Configuration parameters on the primary database

hadr_remote_host	hadr_remote_svc	hadr_remote_inst
boston2	40	dbinst2

On the three standby databases (principal and the two auxiliary), the remote parameters are established from the configuration parameters corresponding to the host name, port number, and instance name settings on the primary database, as shown in Table 6-3.

Table 6-3 Configuration parameters on the principal standby and two auxiliary standby databases

hadr_remote_host	hadr_remote_svc	hadr_remote_inst
boston1	10	dbinst1

Note: The values of these remote configuration parameters are important during takeover. During takeover, DB2 automatically makes the configuration changes and the standby databases are automatically redirected to the new primary database to reflect the new topology and their new roles.

Synchronization values (`hadr_syncmode`) must be decided. For Great Toque Outdoors to achieve the closest possible synchronization between the primary database and principal standby database, the `hadr_syncmode` configuration parameter is set to SYNC on both on the primary database and the principal standby database. SYNC mode ensures zero data loss in case of an outage. On the two auxiliary standby databases, the `hadr_syncmode` configuration parameter is set to SUPERASYNC. (This is done automatically and is the only SYNC mode supported for auxiliary standby databases.)

The `hadr_target_list` configuration parameter can now be established. This configuration parameter specifies the databases that will be accepted as a standby database. The primary and standby databases must include each other in their `hadr_target_list`. This way ensures that, in the case of switched roles, the original primary remains in the system as a standby of the new primary database.

This parameter uses the format host:port.

- Host name (`hadr_local_host`)
- Port number (`hadr_local_svc`)

The host:port value of each database is separated with the pipe character (vertical bar, “|”) as a delimiter.

The `hadr_target_list` configuration parameters are established as follows:

- On the primary database

  The `hadr_target_list` contains the information of the three standby databases. The first entry specifies the principal standby database, and then the two auxiliary standby databases:

  ```
  boston2:40|toronto:41|seattle:42
  ```

- On the principal standby database

  The `hadr_target_list` indicates the first entry is the primary database, and then specifies the two auxiliary standby databases:

  ```
  boston1:10|toronto:41|seattle:42
  ```

- On the two auxiliary standby databases

  The `hadr_target_list` first indicates the principal standby database, next indicates the primary database, and then indicates the other auxiliary standby database:

  ```
  boston2:40|boston1:10|seattle:42
  boston2:40|boston1:10|toronto:41
  ```

The values of the HADR configuration parameters are now established, as listed in Table 6-4.

Table 6-4 Summary of the 7 HADR configuration parameters

Configuration parameter	**Primary**	**Principal**	**Auxiliary**	**Auxiliary**
hadr_local_host	boston1	boston2	toronto	seattle
hadr_local_svc	10	40	41	42
hadr_remote_host	boston2	boston1	boston1	boston1
hadr_remote_svc	40	10	10	10
hadr_remote_inst	dbinst2	dbinst1	dbinst1	dbinst1
hadr_syncmode	SYNC	SYNC	SUPERASYNC	SUPERASYNC
hadr_target_list	boston2:40|toronto:41|seattle:42	boston1:10|toronto:41|seattle:42	boston2:40|boston1:10|seattle:42	boston2:40|boston1:10|toronto:41

Setting up a multiple standby system

After the values of HADR configuration parameters are established, the multiple standby system can be set up. Use the following steps to set up a multiple standby environment:

1. Create the standby database by restoring a backup image based on the existing database that is to be the primary database.
2. Issue the following command at the primary database:

   ```
   BACKUP DB sales TO /nfs1/backups/db2/sales;
   ```

3. Issue the following command on each standby database:

   ```
   RESTORE DB sales FROM /nfs1/backups/db2/sales REPLACE HISTORY FILE;
   ```

4. Configure each standby database with the values planned previously.

   On the principal standby database (boston2), remote host, remote service, and remote instance parameters point to the primary database (the first value in the `hadr_target_list`):

   ```
   DB2 "UPDATE DB CFG FOR sales USING
        HADR_TARGET_LIST  boston1:10|toronto:41|seattle:42
        HADR_REMOTE_HOST  boston1
        HADR_REMOTE_SVC   10
        HADR_LOCAL_HOST   boston2
        HADR_LOCAL_SVC    40
        HADR_SYNCMODE     sync
        HADR_REMOTE_INST  db2inst1";
   ```

On the first auxiliary standby database (toronto):

```
DB2 "UPDATE DB CFG FOR sales USING
     HADR_TARGET_LIST  boston2:40|boston1:10|seattle:42
     HADR_REMOTE_HOST  boston1
     HADR_REMOTE_SVC   10
     HADR_LOCAL_HOST   toronto
     HADR_LOCAL_SVC    41
     HADR_SYNCMODE     superasync
     HADR_REMOTE_INST  db2inst1";
```

On the second auxiliary standby database (seattle):

```
DB2 "UPDATE DB CFG FOR sales USING
     HADR_TARGET_LIST  boston2:40|boston1:10|toronto:41
     HADR_REMOTE_HOST  boston1
     HADR_REMOTE_SVC   10
     HADR_LOCAL_HOST   seattle
     HADR_LOCAL_SVC    42
     HADR_SYNCMODE     superasync
     HADR_REMOTE_INST  db2inst1;
```

5. Configure the primary database with the values planned previously. On the primary database (boston1), the principal standby database is boston2 and the remote host, remote service, and remote instance parameters point to the principal standby (the first value in the hadr_target_list):

```
DB2 "UPDATE DB CFG FOR sales USING
     HADR_TARGET_LIST  boston2:40|toronto:41|seattle:42
     HADR_REMOTE_HOST  boston2
     HADR_REMOTE_SVC   40
     HADR_LOCAL_HOST   boston1
     HADR_LOCAL_SVC    10
     HADR_SYNCMODE     sync
     HADR_REMOTE_INST  db2inst2";
```

6. Start the three standby databases by using the following command:

```
START HADR ON DB sales AS STANDBY;
```

7. Start the primary database by using the following command:

```
START HADR ON DB sales AS PRIMARY;
```

All standby databases connect to the primary database within seconds. The DBAs can monitor the standby databases using the **db2pd** command.

Configuring delayed log replay

To configure a standby system for delayed log replay, the DBAs must set the `hadr_replay_delay` database configuration parameter. This configuration parameter is specified in units of seconds. The DBAs decide a replay delay of two hours is appropriate. The DBAs set the replay delay to two hours, which is 7200 seconds, as shown in Example 6-1.

Example 6-1 Setting the replay delay

```
UPDATE DB CFG FOR DB sales USING hadr_replay_delay 7200;
DEACTIVATE DB sales;
ACTIVATE DB sales;
```

To enable log spooling with unlimited spooling, the DBAs issue the command in Example 6-2.

Example 6-2 Enabling log spooling

```
UPDATE DB CFG FOR DB sales USING hadr_spool_limit -1;
```

For the value to take effect, the DBAs must restart HADR.

6.1.3 Summary

Great Toque Outdoors retail stores require availability and errant transaction protection. The multiple standby databases functionality that is provided by DB2 10 addresses its problems and protects its data while also keeping it highly available.

6.2 Continuous availability with DB2 pureScale Feature

With an increasing focus on 24x7 availability, companies must be prepared to handle both planned (maintenance) and unplanned outages. In the area of continuous availability, the DB2 pureScale Feature is enhanced to better reflect its capabilities as a clustered database solution.

The DB2 pureScale Feature was first introduced in DB2 9.8, which was a DB2 pureScale-only release. DB2 10 builds on DB2 pureScale Feature support, reliability, and performance.

In DB2 10, the DB2 pureScale Feature is included in certain DB2 editions and can be installed as a native component. Improvements to supported networks and various performance improvements save DBAs time and resources.

The DB2 pureScale Feature provides continuous availability, scalability, and application transparency across a cluster of database servers that share a common set of disks. The DB2 pureScale technology is based on the proven DB2 for IBM z/OS® Parallel Sysplex® architecture, recognized as a gold industry standard for maintaining high availability and scalability.

A DB2 pureScale instance consists of members and the cluster caching facility (CF) servers. A typical DB2 pureScale environment has four members and two CFs: a primary CF and a secondary CF.

DB2 Cluster Services (CS) is a set of subcomponents and services that provide built-in failure detection, recovery automation, and a cluster file system for shared access. The CS includes other industry-leading IBM software:

- General Parallel File System (IBM GPFS™)
- Reliable Services Clustering Technology (RSCT)
- IBM Tivoli® Systems Automation for Multiplatforms (SA MP)

Each member and CF has a CS. The CS constantly monitors the members and CFs and automatically initiates recovery processing if required. Several of the resources that the CS monitors are as follows:

- Access to paths and file systems
- Cluster caching facility server processes
- DB2 processes
- Host computers in the cluster
- Network adapters

6.2.1 Business value

Great Toque Outdoors adventure travel division, forever one user-click away from losing a potential customer, requires continuous availability, with zero downtime. The adventure travel division is an Internet presence and must meet the needs of potential customers who are browsing for vacation items that are related to travel.

The current model of Great Toque Outdoors adventure travel division poses the following immediate problems:

- Has the constant potential to lose a customer purchase when customer query does not receive an immediate response; continuous availability with no downtime is necessary.
- Requires the ability to scale out for a yearly inventory sale, and also periodic short-term travel specials.

When assessing DB2 10, the company realized that the DB2 pureScale Feature could address the problems and provide the continuous availability and scalability required.

Another apparent benefit is that the continuing enhancements to DB2 pureScale Feature functionality can maximize system efficiency and throughput. DB2 10 includes support of the following in a DB2 pureScale environment:

- Support for 10GE network on IBM AIX® servers, and Red Hat Enterprise Linux (RHEL) 6.1
- DB2 Workload Manager (DB2 WLM)
- Table partitioning for DB2 pureScale tables. The ability to partition tables in a DB2 pureScale environment also includes the following items:
  - Table space-level backup and recovery operations
  - The new CURRENT MEMBER clause in the CREATE TABLE and ALTER TABLE statements, which can be used to partition a table or index to reduce the level of active sharing between members.

6.2.2 Business application examples

A critical requirement for Great Toque Outdoors adventure travel division is 24x7 availability with no downtime. The adventure travel division also must be able to scale out for peak sales times. The DB2 pureScale Feature meets these needs.

Separate networks in one environment

DB2 10 provides an alternative to the existing InfiniBand support. Support for remote direct memory access (RDMA) over Converged Ethernet (RoCE) networks provides more choice for the Great Toque Outdoors DBAs who are deploying the DB2 pureScale Feature. By using existing Ethernet network infrastructure to help reduce deployment costs, the DBAs do not have to adopt a new network medium. The DBAs can take advantage of a less expensive test environment on their Red Hat Enterprise Linux (RHEL) 6.1 platforms. The addition of 10 Gigabit Ethernet (10GE) network support provides the DBAs the flexibility to run a test environment on a 10GE network, while running the production environment on an InfiniBand network, allowing two separate networks in one environment. The DBAs can run DB2 pureScale environment tests on the more readily available 10GE network that they already have in place. This task can be done at a much lower cost than the alternative (and more costly) InfiniBand network.

Improved performance and availability

With DB2 10, the Great Toque Outdoors DBAs can use DB2 Workload Manager (DB2 WLM) to define, manage, and monitor workloads in a DB2 pureScale environment.

For example, with the ability to assign workload and prioritize data by using DB2 WLM, the DBAs can also use the features of multi-temperature storage to manage different classes of work and assign priority to workload, based on data temperature. As described in Chapter 4, “Multi-temperature storage” on page 49, hot data is given a higher priority and accessed faster. As a result, the DBAs can assign priorities so that update requests to the travel table are given higher priority over query requests to the table.

With the addition of table partitioning in a DB2 pureScale environment, the DBAs can put each table partition into a separate table space. In this way, DBAs can divide table objects into multiple table spaces for better performance. Because only the required partitions are accessed during query processing, performance increases. The DBAs can also, for example, use this approach to take advantage of the features of multi-temperature storage, by moving table spaces between storage groups. Availability is improved during planned maintenance because the DBAs have better control over partition-level maintenance.

Table-space-level backup and recovery reduces the amount of backup time and space that are required by allowing the DBAs to back up only the subset of table partitions that are actively being updated. The DBAs can segment the table space backup to a specific partition. Great Toque Outdoors keeps a backup of the past three years. Instead of backing up all data, only the most recent quarter (three months) is backed up. This way dramatically reduces both time and storage. For example Chapter 4, “Multi-temperature storage” on page 49 shows that the ORDERS table is created as a partitioned table. Instead of backing up the entire table, the DBAs back up only the past three months: April, May, and June 2012. Availability is improved during planned maintenance because the DBAs have better control over backup and recovery operations.

6.2.3 Summary

Great Toque Outdoors adventure travel division requires zero downtime, and scalability for a yearly inventory sale and other short-term travel specials. The DB2 pureScale Feature provides the 24x7 continuous availability, scalability, and application transparency that meets the needs of potential customers browsing for vacations and travel related items. Additional DB2 10 enhancements to DB2 pureScale Feature functionality help maximize system efficiency and throughput, and also availability during planned maintenance.

7

Oracle compatibility

In this chapter, we describe DB2 support for the Oracle SQL and Procedural Language/Structured Query Language (PL/SQL) dialects. With this support, many applications that are written for Oracle can execute on DB2 virtually unchanged. We describe the language features that are supported by DB2 and provide examples of their usage.

7.1 Overview of Oracle compatibility

SQL is a programming language for the manipulation and retrieval of data in relational database management systems (RDBMS). SQL became a standard of the American Standards Institute (ANSI) in 1986, and of the International Organization for Standards (ISO) in 1987. Since its inception, the SQL standard has proven to be a dynamic document that has been enhanced with various features over the years. Although all major RDBMS products use SQL, significant issues exist when trying to migrate code between products. These issues are often attributed to the large size and dynamic nature of the standard, which has led to a variety of interpretations and implementations of its content. Additionally, most products have added their own proprietary extensions that are usually only available on their systems. These proprietary extensions often result in *vendor lock-in*, which causes a customer to be dependant on that vendor for products and services. Vendor lock-in limits flexibility and limits the ability of a customer to switch to another vendor RDBMS product without substantial migration costs.

One of the highlights of DB2 is its Oracle compatibility feature. Started in DB2 9.7 and continuing in DB2 10, this feature enables Oracle applications to run natively on DB2 without incurring the time, cost, and risks typically associated with database migrations. Native support means that there is no need for an emulator or emulation layer. In almost all cases, this gives you the ability to take an existing Oracle application and run it on DB2 without having to make any source code changes. During DB2 10 testing, clients easily migrated data from expensive Oracle Database to IBM DB2 software with 98% code compatibility that did not require changing the data or retraining staff.[1]

Being able to transition from Oracle to DB2 with relatively little cost or effort helps you break down many of the barriers that are associated with vendor lock-in.

7.2 Business value

As part of its move into the adventure travel business, Great Toque Outdoors recently acquired the (fictional company) Janben Excursions Group. Janben Excursions are specialists in wilderness adventure vacations, such as white-water rafting and extreme rock climbing. Great Toque Outdoors believes that adding this travel category to its adventure travel division will have a strong appeal to its current customer base.

[1] See *Made in IBM Labs: New IBM Software Accelerates Decision Making in the Era of Big Data*, available at: http://www.ibm.com/press/us/en/pressrelease/37379.wss

Janben Excursions currently uses Oracle Database Standard Edition One as its RDBMS. Before being purchased by Great Toque Outdoors, Janben Excursions saw a need to improve its database management and transaction processing capabilities. In its environment, adding such features would have required a costly upgrade to Oracle Database Enterprise Edition because those desirable enhancements are not available in Oracle Database Standard Edition One.

Now with its acquisition by Great Toque Outdoors, Janben Excursions does not need to invest in an upgrade because all the functions that they want to add are available to them in DB2 10. In addition, with the DB2 compatibility feature, Janben Excursions can continue to run its existing Oracle applications natively on DB2.

7.3 Business application examples

Prior to DB2 9.7, migrating an Oracle Database application to use DB2 required a complete conversion effort because Oracle has proprietary data types, proprietary functions, and the PL/SQL procedural language. With the addition of Oracle compatibility started in DB2 9.7, Oracle Database applications can be migrated to DB2 for Linux, UNIX, and Windows without the need to change data types, function calls, or rewrite PL/SQL code in DB2 SQL PL. In this way, the Janben transition into the Great Toque Outdoors DB2 environment is simple. The ability to run its Oracle applications natively on DB2 means that the company can expect the same speed and efficiency as applications that are written specifically for DB2.

The following sections describe several Oracle syntax constructs that are supported on DB2. These are the compatibility features that reduce the time and complexity of enabling Janben applications to run in the Great Toque Outdoors DB2 environment.

7.3.1 Concurrency control

In the past, locking semantics or concurrency control was one of the major differences between Oracle and DB2.

Isolation level is the term used to describe the degree to which data that is being accessed by one application or process is locked or isolated from any other concurrently executing process. Before DB2 9.7, the behavior of the default isolation level for each database product was significantly different and therefore was incompatible.

Oracle enforces statement-level isolation. This means that all the data returned by a query comes from a single point in time, that is, the time that the query started. As a result, a query never sees changes made by other transactions during its execution. Only data committed before the query began is available to the query. Changes made by other transactions that commit during query execution are invisible. As illustrated in Table 7-1, the phrase often associated with this behavior is “readers do not block writers and writers do not block readers.”

Cursor stability is the default isolation level for DB2. Traditionally, the cursor stability isolation level locks any row being accessed during a transaction while the cursor is positioned on that row. This lock remains in effect until the next row is fetched or the transaction terminates. However, if any data in the row was changed, the lock is held until the change is committed. This means that a new transaction will wait for the outcome of a pending concurrent transaction, so that the transaction is always working with the most recent data. As a result, “writers can block readers and readers can occasionally block writers” (see Table 7-1).

Table 7-1 Isolation levels

Pending transaction	New transaction	Oracle behavior	Old DB2 behavior	New DB2 behavior
Reader	Reader	Does not block	Does not block	Does not block
Reader	Writer	Does not block	Occasionally blocks	Does not block
Writer	Reader	Does not block	Blocks	Does not block
Writer	Writer	Blocks	Blocks	Blocks

Starting in Version 9.7, DB2 supports locking mechanisms that are familiar to Oracle users. Now with cursor stability and currently committed semantics in effect, if DB2 encounters an uncommitted changed row, it simply retrieves the currently committed version of the locked row from the log and eliminates any potential wait situations. Even if the row has been written out and been overwritten in the log buffer, DB2 finds it and brings the required version into the buffer pool.

7.3.2 Data types

The heart of any database is its data. The data type of any given piece of data determines its range of values and the set of operators and functions that apply to it. Any instances of mismatched types or semantics can seriously affect the ability to enable an application to run on another RDBMS. To allow Oracle applications to run on DB2, DB2 must support Oracle nonstandard types.

The following DB2 data types are added to support Oracle applications.

- BOOLEAN
- DATE
- INDEX BY
- NCHAR
- NCLOB
- NVARCHAR
- NUMBER
- REF CURSOR
- ROW TYPE
- TIMESTAMP(n)
- VARCHAR2
- VARRAY

7.3.3 Implicit casting

Implicit casting is the automatic conversion of data of one data type to data of another data type based on an implied set of conversion rules. This automatic conversion occurs in support of weak typing.

Before version 9.7, DB2 used strong typing during comparisons and assignments. Strong typing requires matching data types, which means that you must explicitly convert one or both data types to a common data type before doing comparisons or assignments. Strong typing adheres to the SQL Standard, and so earlier versions of DB2 took the approach that type mismatches were likely caused by coding mistakes. When an Oracle application used weak typing in its SQL, that application failed to compile against these earlier versions of DB2.

Starting in 9.7, the rules used during comparisons and assignments are relaxed. If two objects have mismatched types, implicit casting is used to do comparisons or assignments if a reasonable interpretation of the data types can be made. Implicit casting is also supported during function resolution. When the data types of the arguments of a function being invoked cannot be promoted to the data types of the parameters of the selected function, the data types of the arguments are implicitly cast to the data types of the parameters.

Implicit casting reduces the number of SQL statements that you must modify when enabling applications that run on non-DB2 data servers to run on DB2. In many cases, you no longer have to explicitly cast data types when comparing or assigning values with mismatched data types.

Additionally, DB2 includes another enhancement that enables you to use untyped parameter markers and untyped NULL keywords almost anywhere in an SQL statement where you can use an expression. With deferred-prepare, DB2 does not resolve the type of a parameter marker until it has seen the first actual value.

7.3.4 Built-in functions

All RDBMS products provide libraries of functions that operate on their data, and even though in many cases, similar functionality is being provided, each product attaches a unique name to its version of that built-in function. This "different names for the same thing" approach is yet another roadblock to database compatibility.

DB2 has its own traditional set of functions, but it also supports a library of compatible Oracle functions. This overlap of supported built-in functions eliminates the need to change function calls, or rewrite PL/SQL code when migrating Oracle applications to DB2. Table 7-2 lists some of these compatible Oracle functions and identifies any similar traditional DB2 functions.

Table 7-2 Functions

Supported Oracle function	Description	Equivalent DB2 function
Conversion and cast functions		
TO_CHAR	Returns a character representation of an input expression that has been formatted by using a character template.	VARCHAR_FORMAT
TO_CLOB	Returns a CLOB representation of a character string type.	CLOB
TO_DATE	Returns a timestamp that is based on the interpretation of the input string by using the specified format.	TIMESTAMP_FORMAT
TO_NUMBER	Returns a DECFLOAT(34) value that is based on the interpretation of the input string by using the specified format.	DECFLOAT_FORMAT
TO_SINGLE_BYTE	Returns a string in which multi-byte characters are converted to the equivalent single-byte character where an equivalent character exists.	None

Supported Oracle function	Description	Equivalent DB2 function
TO_TIMESTAMP	Returns a timestamp that is based on the interpretation of the input string using the specified format.	TIMESTAMP_FORMAT
Date arithmetic		
ADD_MONTHS	Returns a datetime value that represents expression plus a specified number of months.	None
EXTRACT	Returns a portion of a date or timestamp based on its arguments.	None
MONTHS_BETWEEN	Returns an estimate of the number of months between the first argument and the second argument.	None
NEXT_DAY	Returns a datetime value that represents the first weekday (identified by the second argument) that is later than the date in the first argument.	None
String manipulation		
INITCAP	Returns a string with the first character of each word converted to uppercase, using the UPPER function semantics, and the other characters converted to lowercase, using the LOWER function semantics.	None
INSTR	Returns the starting position of a string (identified by the second argument) within another string (identified by the first argument).	LOCATE_IN_STRING
INSTRB	Returns the starting position, in bytes, of a string within another string.	None
LISTAGG	Aggregates a set of string elements into one string by concatenating the strings.	None
LPAD	Returns a string representing the first argument that is padded on the left, with pad or blanks	None
RPAD	Returns a string representing the first argument that is padded on the right, with pad or blanks.	None

Supported Oracle function	Description	Equivalent DB2 function
SUBSTR	Returns a substring of a string.	None
SUBSTR2	Returns a substring from a string starting at a specified position in the string and continuing for a specified or default length.	None
SUBSTRB	Returns a substring of a string, beginning at a specified position in the string (length calculated in bytes).	None
Miscellaneous		
ARRAY_AGG	Aggregates a set of elements into an array.	None
BITAND	Returns results as a corresponding base 10 integer value in a data type based on the data type of the input arguments.	None
DECODE	Performs equality comparisons between arguments determining which argument to return as the result.	None
GREATEST	Returns the maximum value in a set of values.	None
HEXTORAW	Returns a bit string representation of a hexadecimal character string.	VARCHAR_BIT_FORMAT
LEAST	Returns the minimum value in a set of values.	None
NVL	Returns the first argument that is not null.	COALESCE
NVL2	Returns the second argument when the first argument is not NULL. If the first argument is NULL, the third argument is returned.	None
RATIO_TO_REPORT	Returns the ratio of an argument to the sum of the arguments in an OLAP partition.	None
UNNEST	Returns a result table that includes a row for each element of the specified array.	None

DB2 support for Oracle built-in functions means that Janben can continue to use Oracle-based scalar functions and run them against the Great Toque Outdoors DB2 database. As part of a travel-now-pay-later promotion, Janben wants to email existing Great Toque Outdoors customers who do not typically use credit when making their purchases. Example 7-1 on page 111 shows how Janben can query the customer and credit card tables (`cust_customer` and `cust_crdt_card`) and use the NVL function in the query to return a specific message when there is no customer credit card information available.

Example 7-1 Query using the NVL function

```
SELECT c.cust_code, c.cust_email, d.cust_cc_serv_code,
   NVL(d.cust_cc_number, 'never') as been_used
FROM
   toque_ct.cust_customer c, toque_ct.cust_crdt_card d
WHERE
   c.cust_code =d.cust_code and c.cust_code BETWEEN 105000 and 105005;


CUST_CODE   CUST_EMAIL           CUST_CC_SERV_CODE BEEN_USED
---------- -------------------- ----------------- -----------------
     105000 JJones@su1003.FIC                  730 349876547075088
     105001 ABoileau@tu1003.FIC               2187 never
     105002 WStockma@rg6002.FIC               8043 5198765424351327
     105003 IWillis@tu1003.FIC                7968 5298765407380243
     105004 JTorval@tu1003.FIC                1314 4998765413329545
     105005 WMaier@bg6002.FIC                  454 4998765404960118

  6 record(s) selected.
```

7.3.5 Oracle SQL support

The complexity and size of the SQL Standard means that most database products do not support the entire standard. This issue can result in situations where, theoretically, something that you should be able to do is not possible in reality because your product does not let you, which can be frustrating.

Equally as aggravating, or perhaps even more so, are situations where one database product vendor has created or implemented their own feature or function, that other products do not offer. The frustration is compounded when that unique feature or function is a key contributor to your productivity. When product-specific features are popular and used extensively, they limit the ability to run common SQL against more than one product. This incompatibility can result in many language tweaks, both small and large.

Table 7-3 lists several popular Oracle features that are supported by DB2.

Table 7-3 Support for Oracle SQL

Feature	**Description**
CONNECT BY recursion	A hierarchical query containing a CONNECT BY clause that defines the join conditions between parent and child elements. Connect-by recursion uses the same subquery for the seed (START WITH clause) and the recursive step (CONNECT BY clause). This combination provides a concise method of representing recursions such as bill of materials, report-to chains, or email threads.
DUAL table	DB2 resolves any unqualified reference to the DUAL table as a built-in view that returns one row and one column that is named DUMMY, whose value is 'X'.
MINUS	MINUS is an SQL operation that derives a result by returning the rows from an initial query that are not returned in a second query. DB2 treats MINUS as a synonym for EXCEPT.
Outer join operator (+)	Support for the outer join operator (+), means that queries can use this operator as alternative syntax within predicates of the WHERE clause.
PUBLIC synonym	Using a PUBLIC synonym or alias, an object can be referenced independent of the current SQL path or CURRENT SCHEMA setting by its simpler, one-part name. You can define public aliases for all of the objects that you can define private aliases for, namely, another alias (private or public), a nickname, a module, a sequence, a PL/SQL package, a table, or a view.
ROWNUM pseudocolumn	Numbers the records in a result set. The first record that meets the WHERE clause criteria in a SELECT statement is given a row number of 1, and every subsequent record meeting that same criteria increases the row number. ROWNUM is allowed in the WHERE clause of a subselect and is useful for restricting the size of a result set.
SELECT INTO FOR UPDATE	Specifies that the selected row from the underlying table will be locked to facilitate updating the row later on in the transaction.
TRUNCATE TABLE	Enables alternative semantics for the TRUNCATE statement, under which IMMEDIATE is an optional keyword that is assumed to be the default if not specified.

DB2 support for Oracle features means that Janben can continue to use its Oracle-based applications and run them against the Great Toque Outdoors DB2 database. Example 7-2 shows how Janben can continue to use ROWNUM in a query to get customer credit card information from a Great Toque Outdoors table. The example also shows how an Oracle statement running in DB2 returns the same results as DB2 native syntax for the same task (FETCH FIRST).

Example 7-2 Using ROWNUM on a DB2 table

```
SELECT cust_code, cust_cc_number FROM toque_ct.cust_crdt_card WHERE
   ROWNUM < 3;

CUST_CODE   CUST_CC_NUMBER
----------- -------------------------------------------------------
     131072 5298765461884536
     131073 5598765426519067

  2 record(s) selected.


SELECT cust_code, cust_cc_number FROM toque_ct.cust_crdt_card FETCH
   FIRST 2 ROWS ONLY;

CUST_CODE   CUST_CC_NUMBER
----------- -------------------------------------------------------
     131072 5298765461884536
     131073 5598765426519067

  2 record(s) selected.
```

7.3.6 PL/SQL support

SQL is part data manipulation language (DML) for accessing and manipulating data, and part data definition language (DDL) for managing database structures like tables and indexes. But SQL does not have the procedural logic necessary for creating applications. Most database products offer extensions to standard SQL that consist of statements and language elements that can be used to implement procedural logic in SQL statements. Both DB2 SQL PL (SQL Procedural Language) and Oracle PL/SQL provide statements for declaring variables and condition handlers, assigning values to variables, and for implementing procedural logic.

Typically, when an application is migrated from one database product to another, the SQL and procedural language is translated from one product dialect to another. As first described in 7.3.5, “Oracle SQL support” on page 111, translation is a complex task.

- The resulting code can be convoluted because of mismatches between the source and target dialects.
- The application developers porting the code might not be familiar with the target dialect. This can make it difficult for them to debug their porting efforts.
- For packaged applications, translation must be repeated with every new release of the application.

The expression “lost in translation” has much truth to it. DB2 avoids these translation issues by being multilingual.

The DB2 engine includes a PL/SQL compiler next to its SQL PL compiler. As shown in Figure 7-1, both compilers produce virtual machine code for the DB2 SQL Unified Runtime Engine. Also note that monitoring and development tools are linked into DB2 at the runtime engine level.

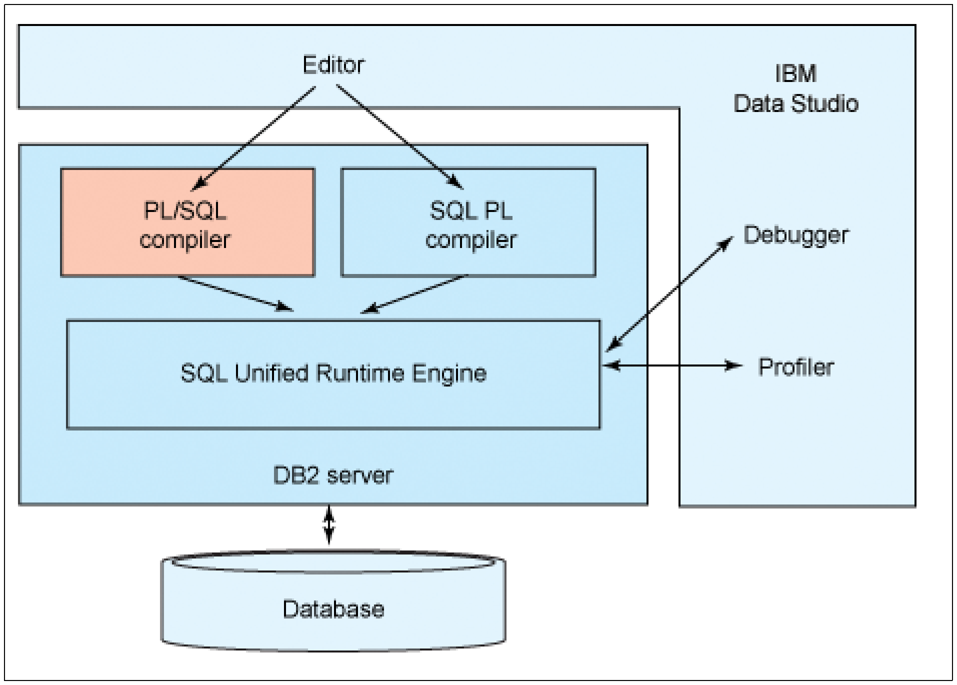


Figure 7-1 Compiler support

The integration of the PL/SQL compiler into DB2 has the following benefits:

- No translation is required. The source code remains as it is in the system catalog.
- Developers can continue working in the language they are familiar with.
- Logic does not need to be moved to the DB2 dialect, even if new logic is written in SQL PL. Routines using PL/SQL and SQL PL can call each other.
- Vendors of packaged applications can use one set of source code for both Oracle and DB2.
- PL/SQL and SQL PL perform at the same speed because they both produce the same virtual machine code for the DB2 SQL Unified Runtime Engine.

DB2 supports all of the common constructs for PL/SQL. Several examples of this syntax support are listed in Table 7-4.

Table 7-4 PL/SQL syntax support

Function	**Description**
Syntax	
:= assignments	Sets a previously-declared variable or formal OUT or IN OUT parameter to the value of an expression.
%ROWTYPE attribute	Declares PL/SQL variables of type record with fields that correspond to the columns of a table or view. Each field in a PL/SQL record assumes the data type of the corresponding column in the table.
%TYPE attribute	Use of this attribute ensures that type compatibility between table columns and PL/SQL variables is maintained.
$IF...$THEN... $ELSE...$END conditional compilation directives	Allows the compilation of different sections of PL/SQL based on context. This feature can be used to minimize any DB2 specific code within a shared PL/SQL code base.
Exception handling	PRAGMA syntax can be used immediately after the definition of an exception, specifying the Oracle sqlcode or DB2 sqlstate that corresponds to a user-defined exception.
Control structures	Standard statements like IF, WHILE, LOOP, and FOR.
Procedures and scalar functions	
IN, OUT, and INOUT parameters	Parameters can be input, output, or both input and output.

Function	Description
Syntax	
Named parameter invocation	Invoke procedures and functions by associating arguments to parameters by name using the following notation, rather than relying on positional association: `=>`
Parameter defaulting	Create procedures and specify default values for parameters. When calling procedures, arguments can be assigned to parameters by name, allowing you to pass named arguments in any order.
BEFORE and AFTER triggers	Specify whether the associated triggered action is applied before or after any changes caused by the actual update of the subject table are applied to the database.
Multiple-event triggers	Creating a trigger with UPDATE, DELETE, and INSERT operations together in a single clause means that the trigger is activated by the occurrence of any of the specified events.
FOR EACH ROW and FOR EACH STATEMENT triggers	Create triggers that fire once for each row of the subject table that is affected by the triggering SQL operation or only one time per statement irrespective of the number of rows affected.
Blocks	
Anonymous blocks	An executable statement that can contain PL/SQL control statements and SQL statements. It can be used to implement procedural logic in a scripting language. In PL/SQL contexts, this statement can be compiled and executed by DB2.
Packages	
PL/SQL package support	PL/SQL package definitions are supported by DB2. A PL/SQL package is a named collection of functions, procedures, variables, cursors, and user-defined types that are referenced using a common qualifier, the package name. **Note:** PL/SQL packages are not the same as DB2 packages.

7.3.7 Built-in packages

Certain Oracle applications use packages that are provided by their RDBMS. These built-in modules provide an easy-to-use programmatic interface for performing a variety of useful operations. To aid in enabling applications using such packages, DB2 provides the packages listed in Table 7-5.

Table 7-5 Built-in packages

Function	Description
DBMS_ALERT	Provides a set of procedures for registering for alerts, sending alerts, and receiving alerts.
DBMS_DDL	Provides the capability to obfuscate DDL objects such as routines, triggers, views, or PL/SQL packages. Obfuscation allows the deployment of SQL objects to a database without exposing the procedural logic.
DBMS_JOB	Provides procedures for the creation, scheduling, and managing of jobs.
DBMS_LOB	Provides an API for LOB processing that echoes DB2's built-in LOB functions.
DBMS_OUTPUT	Provides a set of procedures for putting messages (lines of text) in a message buffer and getting messages from the message buffer.
DBMS_PIPE	Provides a set of routines for sending messages through a pipe within or between sessions that are connected to databases within the same DB2 instance.
DBMS_SQL	Provides a set of procedures for executing dynamic SQL, and therefore supports various data manipulation language (DML) or data definition language (DDL) statements.
DBMS_UTILITY	Provides various utility programs.
UTL_FILE	Provides a set of routines for reading from and writing to files on the DB2 server.
UTL_MAIL	Provides the capability to send email notifications from SQL.
UTL_SMTP	Provides the capability to send email over the Simple Mail Transfer Protocol (SMTP).

7.4 Enabling compatibility

The Oracle compatibility features that are provided by DB2 are in two categories: enabled by default and disabled by default.

Certain features, including the following features, are enabled by default.

- Implicit casting or weak typing that reduces statement modifications (see "Implicit casting" on page 107).
- Synonyms that let you use nonstandard syntax and features like MINUS (see "Oracle SQL support" on page 111).
- Support for DB2 functions and compatible Oracle functions like TO_TIMESTAMP (see "Built-in functions" on page 108).
- Support for built-in modules for performing a variety of useful functions (see 7.3.7, "Built-in packages" on page 117).

Other compatibility features are disabled by default and can be enabled by setting the DB2_COMPATIBILITY_VECTOR registry variable. You can take full advantage of the DB2 compatibility features for Oracle applications by using the recommended setting for the DB2_COMPATIBILITY_VECTOR which is ORA.

Setting the registry variable to enable all of the supported Oracle compatibility features is shown in Example 7-3. Only databases created after the registry variable has been set will be Oracle compatible.

Example 7-3 Enabling all compatibility feature

```
db2set DB2_COMPATIBILITY_VECTOR=ORA
db2stop
db2start
```

Important: If you enable DB2 compatibility features, some SQL behavior changes from what is documented in the SQL reference information.

7.5 Summary

In summary, DB2 does not approach RDBMS compatibility as a classic program conversion task. Most vendors provide tooling that translates one product’s source language to another product’s language. This translation almost always requires manual intervention. Certain vendors choose to emulate the source language in the target language. These approaches often result in a loss of precision, convoluted code, more demands on system space, and a loss of speed.

For DB2, compatibility is built directly in its engine. Rather than viewing differences in dialect as a limitation that must be worked around, DB2 is extended to natively support the types, features, and functions available in Oracle. DB2 is not concerned whether you pronounce SQL as “es queue el” or “sequel,” it provides supports, whatever your dialect is.

8 Ingest utility

In this chapter, we describe the ingest utility, a new feature-rich data movement utility which allows queries to run concurrently with minimal effect on data availability.

8.1 Overview of the ingest utility

To make mission-critical decisions in the business environment of today, data warehouses must have data that is current and available 24x7. DB2 10 features the ingest utility, a new data movement utility that meets these demanding requirements and can be a key component of your extract, transform, and load (ETL) operations.

The ingest utility is a high-speed, configurable, client-side DB2 utility. With its multi-threaded architecture, the ingest utility can pump massive amounts of data quickly and continuously into DB2 tables from files, or pipes, with minimal effect on concurrent user workload and data server resources.

The ingest utility differs from other data movement utilities in several key areas:

- With ingest utility, queries can run concurrently against the DB2 tables that are being loaded with minimal effect on data availability. In other words, you no longer need to choose between data concurrency and availability.
- The ingest utility features high fault tolerance and recoverability.
- Unlike other data movement utilities, the ingest utility supports SQL expressions (including predicates and casting), so in addition to doing the load stage of your ETL process, the ingest utility can also participate in the transform stage.
- The ingest utility supports a range of DML statements, including INSERT, UPDATE, DELETE, MERGE, and REPLACE.
- Not only is the ingest utility compatible with IBM InfoSphere® Warehouse, but it is "database-partition-aware," which means the utility routes inserted rows directly to the appropriate partition (rather than going through a coordinator). This approach contributes to its speed in a partitioned database environment.

8.2 Business value

Great Toque Outdoors currently has an OLTP database for transactional processing and a data warehouse for business intelligence. Its present ETL strategy involves regular, although infrequent, processes by which the data is exported from its OLTP database, transformed with scripts, and imported into its data warehouse with batch load operations.

This current model poses several immediate problems:

- Batch load operations require a window of time when exclusive access to the database is required, locking out other queries and database operations. Because Great Toque Outdoors is a global enterprise, there is no such thing as off-peak hours; there is always a region somewhere in the world that cannot query the data warehouse while the load process is running in another geography.
- As a result of carefully distributing ETL processes throughout the week to minimize application impact, Great Toque Outdoors is left with stale data in its data warehouse. Sometimes its newest data is already one week old, leaving its business analysts at a disadvantage when having to making mission-critical decisions.
- Great Toque Outdoors already committed a large portion of its IT budget to the server and storage hardware that hosts its OLTP database and its data warehouse. Ongoing administration is another expense of moving and transforming the data. Expanding the business to include an adventure travel division will add strain to its resources. The expansion might cause the company to re-evaluate its data warehouse operations, especially considering that data is currently refresh only *periodically*. Furthermore, when they *do* refresh their data, the warehouse is not always available.

Great Toque Outdoors considered an adjustment of its ETL process by implementing a staging server with some middleware. However, this approach was rejected because it increased hardware and software costs, and did not address the issues of stale data and data that is unavailable during the periodic refresh process.

When assessing DB2 10, the company realized that the ingest utility could address immediate problems of Great Toque Outdoors, and also provide other benefits:

- Support for SQL expressions, including predicates and casting, allows column values to be derived from more than one data field, or from an expression at run time. This feature contributes to decreased costs in the transform step, because fewer scripts would need to be maintained to manipulate data after being exported from the OLTP database.
- Unlike Great Toque Outdoors' batch loaders, which did not support loading from a continuous data stream, the INGEST command can load data continuously when called from a scheduled script.
- Also unlike Great Toque Outdoors' batch loader, which supported only a few basic SQL statements, the ingest utility supports INSERT, UPDATE, MERGE, REPLACE, and DELETE statements.
- The ingest utility features high fault tolerance and recoverability. The existing load process used by Great Toque Outdoors allows for restarts of failed jobs, but the recoverability is not as robust as the ingest utility. Because the ingest utility issues SQL statements just like any other DB2 application, changes are logged and its statements can be recovered if necessary. Furthermore, the ingest utility commits data frequently, and the commit interval can be configured by time or number of rows. Rejected rows can be discarded or placed into a file or table. Redirecting the rejected rows allows Great Toque Outdoors DBAs to determine why the record was rejected, such that similar instances can be avoided in future iterations of the process. Redirecting the rejected rows also provides an opportunity for the DBAs to resubmit the file after fixing it.

Not having to balance concurrency with availability, and having fresh data in its data warehouse were key factors for Great Toque Outdoors to choose the ingest utility in DB2 10. It became the obvious choice to help the company reduce cost and improve functionality when compared to its current batch load process.

8.3 Business application examples

Great Toque Outdoors wants its data warehouse to be updated within 30 minutes of a sale. To enable this function, files containing sales transaction records arrive in a continuous stream: 100 - 1000 files an hour.

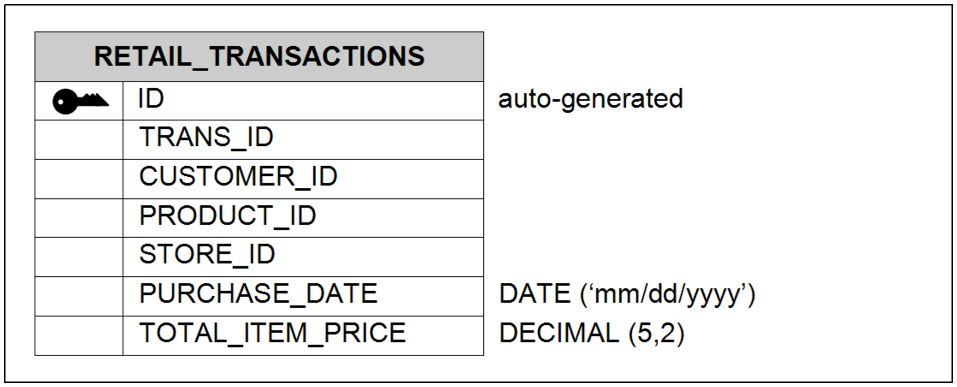
After the company decided to incorporate the ingest utility into the latter stages of its ETL strategy, Great Toque Outdoors implemented the ingest utility in the following phases:

1. Preparation: Design and test a basic INGEST command
2. Preparation: Install a client and prepare the environment for restart
3. Implementation: Set up a continuous ingest process on the production ETL system
4. Post-implementation: Monitor ingest operations and optionally tune performance parameters

The details for these phases are described in the following sections.

8.3.1 Preparation: Design and test INGEST command

One of the tables in the Great Toque Outdoors data warehouse is the RETAIL_TRANSACTIONS table, shown in Figure 8-1.



	RETAIL_TRANSACTIONS	
🔑	ID	auto-generated
	TRANS_ID	
	CUSTOMER_ID	
	PRODUCT_ID	
	STORE_ID	
	PURCHASE_DATE	DATE ('mm/dd/yyyy')
	TOTAL_ITEM_PRICE	DECIMAL (5,2)

Figure 8-1 RETAIL_TRANSACTIONS table (data warehouse)

The data for the RETAIL_TRANSACTIONS table comes from the TRANSACTION_HEADER and TRANSACTION_DETAILS tables in the company's OLTP database, as shown in Figure 8-2.

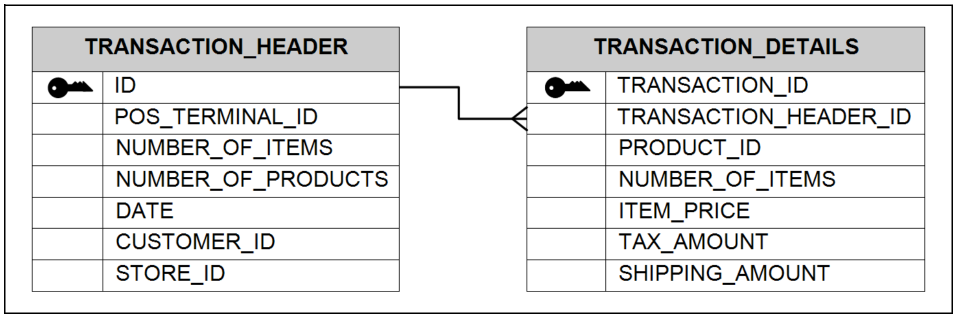


Figure 8-2   TRANSACTION tables (OLTP database)

Before issuing their first INGEST command, the DBAs at Great Toque Outdoors need to generate a data file by exporting data from their OLTP database. Although their existing export process uses scripts with CALL statements to the ADMIN_CMD procedure, for the design phase of the implementation, they issue the EXPORT command from the command line processor (CLP), shown in Example 8-1, to get a small sampling from the OLTP database.

Example 8-1   Exporting sample data from the OLTP database

```
EXPORT TO /home/db2inst1/data/data.del OF DEL
   SELECT
      NEXT VALUE FOR TOQUE_DW.RET_TRANS_SEQ,
      D.TRANSACTION_ID,
      H.CUSTOMER_ID,
      D.PRODUCT_ID,
      H.STORE_ID,
      H.DATE
      D.ITEM_PRICE,
      D.TAX_AMOUNT,
      D.SHIPPING_AMOUNT,
   FROM TOQUE.TRANSACTION_HEADER AS H, TOQUE.TRANSACTION_DETAILS AS D
   WHERE H.ID < 100
```

To ingest the exported `data.del` file, a basic INGEST command is issued from the CLP. The command in Example 8-2 includes the following optional clauses:

- `MESSAGES` *`file`*: The purpose of this clause is so the DBAs can view details about any rows that did not get ingested because of errors.
- `RESTART OFF`: The purpose of this clause is to skip the process of writing failed INGEST commands to a restart table, because the restart table is not created yet and is not required during the design phase. The task of creating a restart table is discussed in 8.3.2, "Preparation: Install a client and prepare the environment" on page 129.

Example 8-2 Basic INGEST command

```
INGEST FROM FILE /home/db2inst1/data/data.del
   FORMAT DELIMITED
   (
      $field1 INTEGER EXTERNAL,
      $field2 INTEGER EXTERNAL,
      $field3 INTEGER EXTERNAL,
      $field4 INTEGER EXTERNAL,
      $field5 INTEGER EXTERNAL,
      $field6 DATE 'mm/dd/yyyy',
      $field7 DECIMAL(5,2) EXTERNAL,
      $field8 DECIMAL(4,2) EXTERNAL,
      $field9 DECIMAL(4,2) EXTERNAL
   )
   MESSAGES /home/db2inst1/logs/messages.txt
   RESTART OFF
   INSERT INTO TOQUE_DW.RETAIL_TRANSACTIONS
      VALUES($field1, $field2, $field3, $field4, $field5,
             $field6, $field7 + $field8 + $field9)
```

The bold text in Example 8-2 demonstrates the ability of the ingest utility to use expressions. The final column in the data warehouse RETAIL_TRANSACTIONS table is TOTAL_ITEM_PRICE. However, the OLTP database stores the values for base price, taxes, and shipping individually. They are exported as-is, and the ingest utility calculates the total price at run time.

The primary key for the RETAIL_TRANSACTIONS data warehouse table is automatically generated. Although their existing export process generates data files for new data only, the DBAs at Great Toque Outdoors also experimented with another feature of the ingest utility: using the MERGE keyword to *update* table rows whose primary key fields match existing table rows, and *add* rows that are new to the table.

To export a superset of the data already exported from the transaction tables in the OLTP database, the command in Example 8-3 is issued.

Example 8-3 Exporting a superset of data

```
EXPORT TO /home/db2inst1/data/data.del OF DEL
   SELECT
      next value for TOQUE_DW.RET_TRANS_SEQ,
      D.TRANSACTION_ID,
      H.CUSTOMER_ID,
      D.PRODUCT_ID,
      H.STORE_ID,
      H.DATE
      D.ITEM_PRICE,
      D.TAX_AMOUNT,
      D.SHIPPING_AMOUNT,
   FROM TOQUE.TRANSACTION_HEADER AS H, TOQUE.TRANSACTION_DETAILS AS D
   WHERE H.ID < 200
```

To ingest the updated data file, the command in Example 8-4 is issued.

Example 8-4 Ingesting data, including a subset which has already been ingested

```
INGEST FROM FILE /home/db2inst1/data/data.del
   FORMAT DELIMITED
   (
      $field1 INTEGER EXTERNAL,
      $field2 INTEGER EXTERNAL,
      $field3 INTEGER EXTERNAL,
      $field4 INTEGER EXTERNAL,
      $field5 INTEGER EXTERNAL,
      $field6 DATE 'mm/dd/yyyy',
      $field7 DECIMAL(5,2) EXTERNAL,
      $field8 DECIMAL(4,2) EXTERNAL,
      $field9 DECIMAL(4,2) EXTERNAL
   )
   MESSAGES /home/db2inst1/msgs.txt
   RESTART OFF
   MERGE INTO TOQUE_DW.RETAIL_TRANSACTIONS
      ON (ID = $field1)
         WHEN MATCHED THEN
            UPDATE SET (TRANS_ID, CUSTOMER_ID, PRODUCT_ID,
                        STORE_ID, PURCHASE_DATE, TOTAL_ITEM_PRICE) =
                       ($field2, $field3, $field4, $field5,
                        $field6, $field7 + $field8 + $field9)
         WHEN NOT MATCHED THEN
            INSERT VALUES($field1, $field2, $field3, $field4, $field5,
                          $field6, $field7 + $field8 + $field9)
```

After deciding to create an INGEST command by using the INSERT keyword (and letting the export process isolate the new rows), the DBAs enhance their INGEST command to include fault tolerance.

At the core of their strategy is the INGEST command, shown in Example 8-5. The command includes clauses to log messages, and, in the event of failure, collect information so the job can be restarted from the point of the last commit.

Example 8-5 Great Toque Outdoors' INGEST command

```
INGEST FROM FILE /home/db2inst1/data/data.del
   FORMAT DELIMITED
   (
      $field1 INTEGER EXTERNAL,
      $field2 INTEGER EXTERNAL,
      $field3 INTEGER EXTERNAL,
      $field4 INTEGER EXTERNAL,
      $field5 INTEGER EXTERNAL,
      $field6 DATE 'mm/dd/yyyy',
      $field7 DECIMAL(5,2) EXTERNAL,
      $field8 DECIMAL(4,2) EXTERNAL,
      $field9 DECIMAL(4,2) EXTERNAL
   )
   MESSAGES /home/db2inst1/logs/messages.txt
   RESTART NEW 'ingestjob001'
   INSERT INTO TOQUE_DW.RETAIL_TRANSACTIONS
      VALUES($field1, $field2, $field3, $field4, $field5,
             $field6, $field7 + $field8 + $field9)
```

8.3.2 Preparation: Install a client and prepare the environment

The DB2 Data Server Runtime Client or the DB2 Data Server Client must be installed on the hardware that runs the ingest utility.

Great Toque Outdoors has several options of where the script can run, including the following locations:

- The DB2 coordinator partition
- The existing ETL server
- A new server that is also hosting an additional DB2 coordinator partition that is dedicated to the ingest utility
- A new server that is dedicated to running the ingest utility

Great Toque Outdoors anticipates moving large volumes of data, especially after it expanded its business to include an adventure travel division. The company therefore decided to run the ingest utility on a dedicated server, an option that yields the highest performance benefit.

The final step of preparation is to create a restart table. You can restart failed INGEST commands from the last commit point; however, a restart table must exist to store the information that is needed to resume an INGEST command. The DBAs perform the one-time task of creating a restart table by calling the SYSPROC.SYSINSTALLOBJECTS stored procedure, as shown in Example 8-6, after connecting to the target database.

Example 8-6 Creating a restart table

```
db2 "CALL SYSPROC.SYSINSTALLOBJECTS('INGEST','C','TABLESPACE1',NULL)"
```

The SYSINSTALLOBJECTS procedure grants SELECT, INSERT, UPDATE, and DELETE privileges to PUBLIC, so explicit GRANT statements are not required.

8.3.3 Implementation

After designing their INGEST command, and preparing their ingest environment, the next phase for the DBAs is to prepare an environment where the steady stream of exported files can be ingested continuously.

Their solution is to create a bash shell script and store it on their production ETL system. The full script is in Appendix A, "Ingest utility script" on page 135.

The script polls a directory for exported files and processes them until the directory is empty. The script is added to a scheduler in anticipation of a scenario where the ingest script exhausts the data files before the next set of data files are exported. The entry shown in Example 8-7 is placed in **`crontab`**, which runs the script every 30 minutes.

Example 8-7 Scheduling the ingest script to run regularly

```
30 * * * * $HOME/bin/ingest_files.sh
```

8.3.4 Post-implementation

After placing the ingest script on their production ETL server and scheduling a **`cron`** job, the DBAs start the maintenance phase of the project. This phase involves monitoring ingest operations and, if needed, tuning performance parameters.

Monitoring

Several methods are involved for monitoring an ingest operation.

Method 1: INGEST LIST command

The INGEST LIST command, as shown in Example 8-8, lists basic information about INGEST commands currently running under the authorization ID that is connected to the database. A separate CLP session is required to successfully invoke this command. It must be run on the same client that the INGEST command is running on.

Example 8-8 Running the INGEST LIST command

```
=> INGEST LIST

Ingest job ID       = DB21000:20101116.123456.234567:34567:45678
Ingest temp job ID  = 1
Database Name       = TOQUEDWDB
Input type          = FILE
Target table        = TOQUE_DW.RETAIL_TRANSACTIONS
Start Time          = 04/10/2010 11:54:45.773215
Running Time        = 01:02:03
Number of records processed = 30,000
```

The `Ingest temp job ID` is an integer you can use on the INGEST GET STATS command, described next.

Method 2: INGEST GET STATS command

The INGEST GET STATS command displays statistics about one or more INGEST commands that are currently running under the authorization ID that is connected to the database. A separate CLP session is required to successfully invoke this command. It must be run on the same client that the INGEST command is running on.

The INGEST GET STATS command can be run to display all ingest operations that are running under the same authorization ID. It can also be run for a specific ingest operation, by specifying an `Ingest job ID` or an `Ingest temp job ID`; these values can be determined by first running the INGEST LIST command. Example 8-9 on page 132 is displaying statistics of an ingest operation based on an `Ingest temp job ID`.

Example 8-9 Running the INGEST GET STATS command for a specific job

```
=> INGEST GET STATS FOR 1

Ingest job ID = DB21000:20101116.123456.234567:34567:4567
Database Name = TOQUEDWDB
Target table  = TOQUE_DW.RETAIL_TRANSACTIONS

Overall          Overall           Current          Current
ingest rate      write rate        ingest rate      write rate
(records/sec)    (writes/sec)      (records/sec)    (writes/sec) Total records
---------------- ----------------- --------------- ---------------- ----------------
54321            65432             76543            87654            98765
```

Method 3: Monitoring routines and elements

Monitoring routines, such as MON_GET_CONNECTION, can also be used to retrieve the following monitor elements that return information about ingest operations:

- `client_acctng`
- `client_applname`
- `appl_name`
- `client_userid`
- `client_wrkstnname`

The statement in Example 8-10 returns the number of rows modified and the number of commits.

Example 8-10 Determining the number of rows modified and the number of commits

```
=> SELECT client_acctng AS "Job ID",
      SUM(rows_modified) AS "Total rows modified",
      SUM(total_app_commits) AS "Total commits"
   FROM TABLE(MON_GET_CONNECTION(NULL, NULL))
   WHERE application_name = 'DB2_INGEST'
   GROUP BY client_acctng
   ORDER BY 1;

Job ID                                          Total rows modified Total commits
----------------------------------------------- ------------------- -------------
DB21000:20101116.123456.234567:34567:45678                       92            52
DB21000:20101116.987654.234567:34567:45678                      172           132

  2 record(s) selected.
```

Tuning

By monitoring ingest operations, the DBAs at Great Toque Outdoors can optionally tune several ingest utility configuration parameters:

- `commit_count`
  - Description: Specifies the number of rows each flusher writes in a single transaction before issuing a commit. The flushers issue the SQL statements to perform the operations on the DB2 tables.
  - Default value: 0
  - Range: 0 to maximum 32-bit signed integer
- `commit_period`
  - Description: Specifies the number of seconds between committed transactions.
  - Default value: 1
  - Range: 0 to 2,678,400 (31 days)
- `num_flushers_per_partition`
  - Description: Specifies the number of flushers to allocate for each database partition.
  - Default value:

    `max(1,((number of logical CPUs)/2)/(number of partitions))`
  - Range: 0 to *x*, where *x* depends on system resources
- `num_formatters`
  - Description: Specifies the number of formatters to allocate. The formatters parse each record, convert the data into the format that DB2 database systems require, and put each formatted record on one of the flusher queues.
  - Default value:

    `max(1,((number of logical CPUs)/2))`
  - Range: 1 to *x*, where *x* depends on system resources
- `pipe_timeout`
  - Description: Specifies the maximum number of seconds to wait for data when the input source is a pipe.
  - Default value: 600 seconds (10 minutes)
  - Range: 0 to 2,678,400 (31 days)

- `retry_count`
  - Description: Specifies the number of times to retry a failed, but recoverable, transaction.
  - Default value: 0
  - Range: 0 to 1000
- `retry_period`
  - Description: Specifies the number of seconds to wait before retrying a failed, but recoverable, transaction.
  - Default value: 0
  - Range: 0 to 2,678,400 (31 days)
- `shm_max_size`
  - Description: Specifies the maximum size of Inter Process Communication (IPC) shared memory in bytes. Because the ingest utility runs on the client, this memory is allocated on the client machine.
  - Default value: 1 GB (1,073,741,824)
  - Range: *x* to available memory

For example, by default, the INGEST command commits transactions every 1 second. If the DBAs at Great Toque Outdoors decided to change the behavior to commit every 1,000 rows (instead of an amount of time), they would issue the commands in Example 8-11:

Example 8-11 Setting ingest configuration parameters

```
INGEST SET commit_count 1000;
INGEST SET commit_period 0;
```

8.4 Summary

With the DB2 10 ingest utility, you can pump massive amounts of data into your data warehouse without sacrificing availability. The ingest utility features high fault tolerance and recoverability. It also supports SQL expressions and a range of DML statements, and a number of monitoring options. With its rich set of features, the ingest utility can be a key component of your ETL system.

A

Ingest utility script

This appendix provides a script that ingests data into a data warehouse on a continuous basis.

Introduction to the ingest utility script

The script in Example A-1 demonstrates the ongoing processing of exported files as they are written to the `/home/db2inst1/data` directory and ingesting that data into a target table in the data warehouse. The data files are moved to a `SUCCESS` directory if the ingest was successful; otherwise, the files are moved to a *`FAILED`* directory. The script also creates two log files:

- The `ingest_files.log` file is a global log that stores information messages for all operations. Messages for successful operations are preceded by `[INFO]`; error messages for failed operations are preceded by `[ERROR]`.
- The `failed_ingest.log` file stores information about the failed INGEST commands.

The script, `ingest_files.sh`, requires a user name as the first parameter, and a password as the second parameter. It does not output to standard output; instead, the DBAs check the *<SCRIPT_PATH>*`/logs/ingest_files.log` file after the script completes.

Notes:

- Long lines use the standard line continuation character (backslash)
- Script logic that is not directly related to the ingest utility is displayed in comment boxes with pseudo-code:

```
########################################################
# check if the source directory exists                 #
########################################################
```

The ingest_files.sh script

Example A-1 shows the ingest_files.sh script.

Example A-1 ingest_files.sh

```
# =================== SECTION 1: HANDLE COMMAND LINE ARGUMENTS ===================

# check the command line arguments
if [ $# -ne 2 ]
then
    echo -e "INVALID number of arguments"
    echo -e "Enter two arguments: USER_NAME and PASSWORD"
    exit 1
fi
```

```
# assign the command line arguments to shell variables
USER_NAME=$1
PASSWORD=$2

# check if user name and password is non-blank
if [ ! -n "$USER_NAME" -o ! -n "$PASSWORD" ]
then
    echo -e "username and password cannot be blank"
    exit 1
fi

# ============================ SECTION 2: INPUT VALUES ============================

# path to search for the data files
INPUT_FILES_DIRECTORY="/home/db2inst1/data"

# store the name of the data warehouse database
DATABASE_NAME="TOQUEDWDB"

# store the schema name of the data warehouse database
SCHEMA_NAME="TOQUE_DW"

# store the name of the data warehouse's target table
TABLE_NAME="RETAIL_TRANSACTIONS"

# store the user name of the user to be notified about failed INGEST commands
MAIL_USER_NAME=${USER_NAME}

# get the script path
SCRIPT_PATH="$( dirname "$( which "$0" )" )"

# ======================== SECTION 3: INGEST COMMAND DETAILS ========================

# break the ingest command elements into two parts, assigning each to a variable

# part 1
INGEST_COMMAND_DETAILS_PART1="FORMAT DELIMITED \
(\$field1 INTEGER EXTERNAL, \$field2 INTEGER EXTERNAL, \
\$field3 INTEGER EXTERNAL, \$field4 INTEGER EXTERNAL, \
\$field5 INTEGER EXTERNAL, \$field6 DATE 'mm/dd/yyyy', \
\$field7 DECIMAL(5,2) EXTERNAL, \$field8 DECIMAL(4,2) EXTERNAL, \
\$field9 DECIMAL(4,2) EXTERNAL) \
MESSAGES ${SCRIPT_PATH}/logs/messages.txt RESTART NEW"

# part 2
INGEST_COMMAND_DETAILS_PART2="INSERT INTO ${SCHEMA_NAME}.${TABLE_NAME} \
VALUES (\$field1, \$field2, \$field3, \$field4, \$field5, \
\$field6, \$field7 + \$field8 + \$field9)"
```

```
# ======= SECTION 4: SET UP ENVIRONMENT, LOGS, SUCCESS AND FAILED DIRECTORIES =======

# create the 'logs' subdirectory if it does not exist in the SCRIPT_PATH directory
if [ ! -d ${SCRIPT_PATH}/logs ]
then
    mkdir ${SCRIPT_PATH}/logs
    # if mkdir fails then echo to standard out and exit
    if [ $? -ne 0 ]
    then
   echo "unable to create logs directory at $SCRIPT_PATH"
   exit 1
    fi
fi

# store the name of the file that logs all operations performed by the ingest
LOGFILE=${SCRIPT_PATH}/logs/ingest_files.log

########################################################################
# create ingest_files.log if it does not exist in the 'logs' directory  #
########################################################################

# store the name of the file that logs failed operations attempted by the ingest
FAILEDINGESTLOG=${SCRIPT_PATH}/logs/failed_ingest.log

########################################################################
# create failed_ingest.log if it does not exist in the 'logs' directory #
########################################################################

########################################################################
# check if INPUT_FILES_DIRECTORY exists and if it is a valid directory  #
########################################################################

echo -e $time"  [INFO]  ${INPUT_FILES_DIRECTORY} found" >>$LOGFILE

# a 'success' subdirectory holds all of the files that were successfully ingested
SUCCESS_DIRECTORY=${INPUT_FILES_DIRECTORY}/success

#################################################################################
# create SUCCESS_DIRECTORY if it does not already exist in INPUT_FILES_DIRECTORY #
#################################################################################

# a 'failed' subdirectory holds of all the files that were not successfully ingested
FAILED_DIRECTORY=${INPUT_FILES_DIRECTORY}/failed


################################################################################
# create FAILED_DIRECTORY if it does not already exist in INPUT_FILES_DIRECTORY #
################################################################################
```

```
# === SECTION 5: FIND FILES, CONNECT TO THE DATABASE, EXECUTE THE INGEST COMMAND ===

# loop through the files in the INPUT_FILES_DIRECTORY
while true
do

  FILES='find ${INPUT_FILES_DIRECTORY} \( -name success -prune \) -o \
  \( -name failed -prune \) \
  -o -name "*.*" -type f -print'

  if [ ! -n "$FILES" ]
  then
    echo -e " " >>$LOGFILE
    time='date'
    echo -e $time" [ERROR] no files found at $INPUT_FILES_DIRECTORY" >>$LOGFILE
    echo -e " " >>$LOGFILE
    exit 1
  else

    # connect to DATABASE_NAME as USER_NAME
    time='date'
    echo -e $time"  [INFO]  attempting to connect to $DATABASE_NAME \
    as $USER..." >>$LOGFILE

    time='date'
    echo -e $time"  [INFO]  " >>$LOGFILE
    db2 connect to $DATABASE_NAME USER $USER_NAME USING $PASSWORD >> $LOGFILE 2>&1

    # exit if connection to the DATABASE_NAME AS USER_NAME fails
    if [ $? -ne 0 ]
    then
      time='date'
      echo -e $time"  [ERROR]  invalid database_name, \
      user_name or password" >>$LOGFILE
      # exit if database_name, user_name, or password is invalid
      exit 1
    fi

    for FILE in $FILES
    do

      # generate a separate INGEST command for each file found

      time='date'
      echo -e $time"  [INFO]  ${FILE} found at $INPUT_FILES_DIRECTORY" >>$LOGFILE

      time='date'
      echo -e $time" [INFO] invoking INGEST command for $FILE found at \
      $INPUT_FILES_DIRECTORY..." >>$LOGFILE
```

```
# store a unique job ID in case the ingest command needs to be restarted
timestamp='date +%d%m%y%H%M%S'
jobId="${SCHEMA_NAME}.${TABLE_NAME}_${timestamp}"

#################################################################
# write the jobId, time, source file, target db & table,        #
# and userID to the global log file                             #
#################################################################

# run the INGEST command
db2 -xv "INGEST FROM FILE ${FILE} ${INGEST_COMMAND_DETAILS_PART1} '${jobId}' \
${INGEST_COMMAND_DETAILS_PART2}" >>$LOGFILE 2>&1

# capture the return status of INGEST (0 = success; 4 = failure)
INGEST_RET_CODE=$?

#################################################################
# write the jobId, time, and return code to the global log file #
#################################################################

# check the return status of the INGEST command
if [ $INGEST_RET_CODE -lt 4 ]
then
  # INGEST was successful - move data file to SUCCESS_DIRECTORY
  time='date'
  echo -e $time"  [INFO]  $FILE successfully ingested" >>$LOGFILE

  time='date'
  echo -e $time"  [INFO]  moving $FILE to success directory at \
  ${SUCCESS_DIRECTORY}..." >>$LOGFILE

  time='date'
  mv_cmd_success='mv ${FILE} ${SUCCESS_DIRECTORY}/ 2>&1'



  #################################################################
  # error handling if file was not moved successfully             #
  #################################################################

else
  # INGEST failed - move data file to FAILED_DIRECTORY
  time='date'
  echo -e $time"  [ERROR]  ingesting $FILE failed" >>$LOGFILE

  time='date'
  echo -e $time"  [INFO]  moving $FILE to failed directory at \
  ${FAILED_DIRECTORY}..." >>$LOGFILE
```

```
        time='date'
        mv_cmd_error='mv ${FILE} ${FAILED_DIRECTORY}/ 2>&1'

        ########################################################################
        # error handling if file was not moved successfully, send email to USER #
        ########################################################################

        ########################################################################
        # write the jobId, time, source file, target db & table, userID to the  #
        # failure log file                                                      #
        ########################################################################

        # in failure log file, write ingest command string to rerun the failed job
        INGEST_COMMAND="INGEST FROM FILE ${FILE} \
        ${INGEST_COMMAND_DETAILS_PART1} '${jobId}' \
        ${INGEST_COMMAND_DETAILS_PART2}"

        echo -e $INGEST_COMMAND >>$FAILEDINGESTLOG

        ################################################################
        # Write end time, return code, and duration to failure log file #
        # Also concatenate contents of ${SCRIPT_PATH}/messages.txt      #
        ################################################################

      fi
    done
  fi
done
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks publications

The following IBM Redbooks publication has additional information about the topic in this document.

- *Oracle to DB2 Conversion Guide: Compatibility Made Easy*, SG24-7736

  `http://www.redbooks.ibm.com/abstracts/sg247736.html`

You can search for, view, download or order documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Other publications

These publications are also relevant as further information sources:

- Zikopoulos, et al., *DB2 pureScale: Risk Free Agile Scaling*, New York, The McGraw-Hill Companies, 2011. 978-0-07-175240-4

  `http://public.dhe.ibm.com/common/ssi/ecm/en/imm14079usen/IMM14079USEN.PDF`

- Zikopoulos, et al., *Warp Speed, Time Travel, Big Data, and More, DB2 10 for Linux, UNIX, and Windows New Features,* New York, The McGraw-Hill Companies, 2012. 978-0-07-180296-3

  `http://public.dhe.ibm.com/common/ssi/ecm/en/imm14091usen/IMM14091USEN.PDF`

Online resources

The following websites are also relevant as further information sources:

- IBM DB2 10.1 Information Center for Linux, UNIX, and Windows

  http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp
- Introduction to the IBM DB2 pureScale Feature

  http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=/com.ibm.db2.luw.licensing.doc/doc/c0057442.html
- Information about HADR multiple standby databases

  http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=/com.ibm.db2.luw.admin.ha.doc/doc/c0059994.html
- DB2 10: Run Oracle applications on DB2 10 for Linux, UNIX, and Windows

  http://www.ibm.com/developerworks/data/library/techarticle/dm-0907oracleappsondb2/

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM Redbooks

Unleashing DB2 10 for Linux, UNIX, and Windows

(0.2”spine)
0.17”<->0.473”
90<->249 pages

Unleashing DB2 10 for Linux, UNIX, and Windows

Discover storage and availability features that enable faster business decisions

Simplify application development and portability

Explore real-world examples to launch your DB2 10 experience

This IBM Redbooks publication provides a broad understanding of the key features in IBM DB2 10 and how to use these features to get more value from business applications. It includes information about the following features:

- Time Travel Query, which you use to store and retrieve time-based data by using capability built into DB2 10, without needing to build your own solution
- Adaptive compression, an enhanced compression technology that adapts to changing data patterns, yielding extremely high compression ratios
- Multi-temperature storage, which you may use to optimize storage costs by identifying and managing data based on its “temperature” or access requirements
- Row and column access control, which offers security access enforcement on your data, at the row or column level, or both
- Availability enhancements, which provide different DB2 availability features for different enterprise needs; high availability disaster recovery (HADR) multiple-standby databases provide availability and data recovery in one technology, and the IBM DB2 pureScale Feature provides continuous availability and scalability that is transparent to applications
- Oracle compatibility, which allows many applications written for Oracle to run on DB2, virtually unchanged
- Ingest utility, a feature-rich data movement utility that allows queries to run concurrently with minimal impact on data availability

This book is for database administrators and application developers who are considering DB2 10 for a new installation, an upgrade from a previous version of DB2, or a migration from a vendor product. It explains key features and illustrates concepts with real-world examples to provide a clear idea of how powerful these features are. Together, these features provide functionality that lower operational costs, ease development, and provide greater reliability.

SG24-8032-00 ISBN 0738437107