Ingres 10.0

Release Summary

ING-10-RS-03

Contents

Chapter 1: DBMS Server Enhancements in Ingres 10

This section contains the following topics:

Ingres for 64-bit Windows

Ingres is ported to pure 64-bit Windows.

Multiversion Concurrency Control (MVCC)

MVCC provides concurrent access to the database without locking the data. This feature improves the performance of database applications in a multiuser environment. Applications will no longer hang because a read cannot acquire a lock.

MVCC provides each user connected to the database with a "snapshot" of the data to work with. The data is consistent with a point in time. Other users of the database see no changes until the transaction is committed. The snapshot can be taken at the start of a transaction, or at the start of each statement, as determined by the isolation level setting.

This release provides full MVCC support, in which readers do not block writers, and writers do not block readers.

The user invokes MVCC protocols for a session or table with the SQL statement:

```
SET LOCKMODE session | ON table_name WHERE LEVEL = MVCC
```

The alterdb command has two new options, -disable_mvcc and -enable_mvcc, which disable and enable MVCC, respectively. By default, MVCC is enabled for all existing and newly created databases.

Using MVCC is optional. Your existing applications that do not use MVCC will execute in the same manner they worked previously. The overhead of MVCC is the cost of maintaining multiple versions of database pages.

For the system administrator, MVCC may require additional buffer manager memory because Consistent Read pages occupy cache space that otherwise might be used by database pages.

The MVCC feature changes the format of many log records, which means that after running upgradedb, previous journals and checkpoints will be invalid.

For details about this feature, see the following:

- The chapters "Understanding the Locking System" and "Understanding Multiversion Concurrency Control" in the *Database Administrator Guide*
- The SET LOCKMODE and SET SESSION ISOLATION LEVEL statements in the *SQL Reference Guide*
- The alterdb command in the *Command Reference Guide*

Scalar Subqueries

A scalar subquery (or scalar subselect) is a subquery that selects only one column or expression and returns one row.

Ingres has supported scalar subqueries in "*expression comparison_op scalar subquery*" syntax in a WHERE, ON, or HAVING clause. This enhancement provides full scalar subquery support. It allows scalar subqueries to be used anywhere in an SQL query that a column or expression can be used. For example, it can appear in the select-list, in a WHERE or ON clause of a containing query, or as an operand in any expression.

A scalar subquery can be used in the following contexts:

- The select list of a query (that is, the expressions between the SELECT and FROM keywords)
- The JOIN clause of a query
- A WHERE clause that contains CASE, IF, COALESCE, and NULLIF expressions
- The source to an UPDATE statement when the subquery refers to more than the modified table
- A qualifier to a DELETE statement where the subquery identifies the rows to delete
- The VALUES clause of an INSERT statement

Scalar subqueries can be used to compute several different types of aggregations (max and avg) all in the same SQL statement. The following query uses both scalar subqueries and in-line views:

```
SELECT
  (SELECT MAX(salary) FROM emp) AS highest_salary,
  emp_name AS employee_name,
  (SELECT AVG(bonus) FROM commission) AS avg_comission,
  dept_name
FROM emp, (SELECT dept_name FROM dept WHERE dept = ‘finance’);
```

Scalar subqueries can also be used for inserting into tables, based on values from other tables. The following example uses scalar subquery to compute the maximum credit for Bill and insert this value into a max_credit table.

```
INSERT INTO max_credit (name,max_credit) VALUES (
   ‘Bill’,
   SELECT MAX(credit) FROM credit_table WHERE name = ‘Bill’
);

INSERT INTO emp_salary_summary
(sum_salaries, max_salary,min_salary, avg_salary)
VALUES (
   (SELECT SUM(salary) from emp),
   (SELECT MAX(salary) from emp),
   (SELECT MIN(salary) from emp),
   (SELECT AVG(salary) from emp));
```

To be a valid scalar subquery, the subquery must produce at most a single value. That is, the result should consist of zero or one row of one column. If more than one row results, a cardinality error occurs.

For more information, see these sections in the *SQL Reference Guide*:

- Scalar Subqueries
- Subqueries in the Target List, SET, and VALUES Clauses
- Example Query Using Derived Tables and Scalar Subquery
- Scalar Subquery Feature Can Change Behavior of Existing Subqueries

Data at Rest Encryption

Specific database table columns can be encrypted to enhance data security, ensure privacy, and protect media that contains database records holding sensitive information. This feature ensures that data in an Ingres database is stored on disk or other media in such a way that protected columns are unreadable without knowledge of the encryption passphrase. Ingres functions AES_ENCRYPT and AES_DECRYPT provide explicit encryption and decryption. Table-level encryption is not provided.

Encryption "at rest" refers to data on physical media recorded in a persistent form. This includes stored database records, the transaction log, journals, and checkpoints. Encrypted columns are stored in the database files using 128-, 192-, or 256-bit Advanced Encryption Standard (AES) encryption.

When encryption has been enabled for a table that contains encrypted columns, the encryption is transparent to the applications accessing the data. Only minimal changes, if any, are required at the application level.

Note: Data is readable in DBMS buffers when a client is communicating with the DBMS Server. Data is also not protected when copied out of the database to flat files.

You specify encryption by using options on the CREATE TABLE statement. For example, the following statement creates a table with an encrypted column:

```
CREATE TABLE socsectab
(
    fname  CHAR(10),
    lname  CHAR(20),
    socsec CHAR(11) ENCRYPT
)
    WITH ENCRYPTION=AES128,
        PASSPHRASE='this is a secret';
```

At the column level, you can specify the encryption option NOSALT to override the default encryption, which is done with salt (extra random bits).

To enable access to data in an encrypted table, you must know the passphrase, which is specified on the MODIFY statement:

```
MODIFY socsectab ENCRYPT
    WITH PASSPHRASE='this is a secret';
```

To revoke access, specify an empty passphrase string using the MODIFY statement:

```
MODIFY socsectab ENCRYPT
    WITH PASSPHRASE='';
```

To change the passphrase, specify a new one on the MODIFY statement:

```
MODIFY socsectab ENCRYPT
    WITH PASSPHRASE='this is a secret',
        NEW_PASSPHRASE='we have a new secret';
```

Ingres servers that include this feature are compatible with other server versions.

Note: Because of changes to the standard catalogs, you must run upgradedb on existing installations.

For more information, see the *Security Guide* and *SQL Reference Guide*.

Long Identifiers

The maximum length of names for certain objects has increased from 32 to 256 bytes.

Names for the following objects can now be a maximum of 256 bytes:

- Table
- View
- Index
- Column
- Partition
- Procedure
- Procedure parameter
- Rule
- Sequence
- Synonym
- Object
- Constraint
- Integrity

The maximum length of names for the following objects remains at 32 bytes:

- Database
- Owner
- User
- Group
- Profile
- Role
- Schema
- Location
- Event
- Alarm
- Node

The maximum length of a collation sequence name is 64 bytes.

The maximum length for names of objects managed by Ingres tools such as Query-By-Forms, Report-By-Forms, Vision, and Visual Forms Editor remains at 32 bytes. (Examples of these objects are Forms, JoinDefs, QBFnames, Graphs, and Reports.) These tools may be unable to display and operate on objects (such as tables and columns) with long names.

The size of related columns in the standard catalog interface has also increased. For example, iitables.table_name, which is a view on iirelation.relid, has increased from 32 to 256.

The Long Identifiers feature changes the catalogs and the format of many log records, which means that after running upgradedb, previous journals and checkpoints will be invalid.

Pre-Ingres-10 databases must be upgraded with upgradedb before they can be accessed. Alternatively, you can unload your database before installing Ingres 10 and then reload the database after installing Ingres 10.

An older version (such as 9.2) client can access an Ingres 10 database if you limit names to a maximum of 32 bytes.

Migration Note: If an application is using dynamic SQL and is dynamically linked against libq, you must recompile the application to properly access the SQL Descriptor Area because of changes in the IISQLVAR structure for Long Identifiers.

Renaming Tables and Columns

Tables and columns can be renamed using the following statements new to Ingres 10:

```
[EXEC SQL] ALTER TABLE [schema.]table_name RENAME TO new_table_name

[EXEC SQL] ALTER TABLE [schema.]table_name RENAME [COLUMN] old_column_name TO
new_column_name

[EXEC SQL] RENAME TABLE [schema.]table_name TO new_table_name
```

These commands update the core catalogs to reflect the new table or column name. If a table name or column name is referred to in views, procedures, constraints, rules or other dependent objects, then the table name or column name is not allowed to be renamed; if renaming is attempted, appropriate error messages are given for each case. Ingres deals with each of the dependencies in the following ways:

- Indexes, grants, comments, synonyms, sequences, partitions, and extension tables are automatically transferred to the newly renamed table or column. For grants, query text is updated to reflect the new table and column names.
- A renaming operation on a table is not allowed if any views, procedures, rules, integrities, check or referential constraints, non-grant permits, or security alarms are dependent on the table being renamed. Dependencies for each case will be listed in the error log. Primary key and unique constraints will be transferred to the new table name.
- A renaming operation on a column is not allowed if any views, constraints, or integrities are dependent on the column being renamed or there are any procedures, rules, referential or check constraints, non-grant permits, or security alarms dependent on the table that contains the column being renamed. Dependencies on the column will be listed for each case in the error log.
- Any forms, join definitions, or reports that refer to the old table or column name will be invalidated; these must be recreated and reloaded.
- Any copydb (unloaddb) scripts generated before renaming the table or column will no longer be usable for reload operations.

Notes:

- The new SQL renaming syntax works only with Ingres 10.
- The ALTER TABLE RENAME or RENAME TABLE operation can be rolled back.
- Procedures are not permitted to execute ALTER TABLE RENAME or RENAME TABLE statements.

- Ingres Star does not support ALTER TABLE RENAME or RENAME TABLE statements for renaming tables and columns.

For more information:

- About statement syntax and examples, and complete rules and restrictions for renaming tables and columns, see the *SQL Reference Guide*.
- About registering renamed tables or columns with Ingres Star, see Register Table as Link Statement—Define Table to Ingres Star and Register Tables with StarView in the *Star User Guide*.
- About updating the dd_regist_columns and dd_regist_tables tables with updated table and column names, see those sections and Reregistering Renamed Tables and Tables with Renamed Columns in the *Replicator User Guide*.

Bulk Loading Improvements

Several improvements make it easier to bulk load data:

- The CSV and SSV delimiters allow the COPY statement to read and write files with comma-separated values (CSV).
- COPY FROM an ordinary text data file is up to two times faster.
- A COPY FROM that loads a partitioned table can now perform bulk-load into partitions that satisfy the standard bulk-load criteria.

A related feature, Batch Query Execution (see page 14), improves the efficiency of loading data from a client interface.

For more information, see these sections in the *SQL Reference Guide*:

- Delimiters in the Data File
- CSV and SSV Delimiters

Batch Query Execution

Queries now can be executed in batch. Batch statement execution improves communication performance between the client and the DBMS Server. Batched statements that are run against an Ingres Star server may see less performance benefit.

A group of statements can be sent to the server, where they are executed, and then the correct number of responses are sent back to the client. This feature is restricted to queries that return either a response or a row count, but no data.

Statements allowed in batched queries include INSERT, DELETE, and statements that return no data, such as CREATE TABLE. SELECT statements and row-producing procedures are not allowed in batched queries.

To take advantage of batch query execution, Java programmers can use the addBatch and executeBatch methods, which are supported in Ingres JDBC. OpenAPI programmers can use the new function, IIapi_batch().

Batched statements that use repeated dynamic INSERT statements (for example, through Java-based Extract Transfer and Load tools) are specially optimized to improve performance.

If the DBMS does not support batch processing, the Ingres JDBC driver detects it and automatically executes the statements individually.

Even for servers that support batch processing, if you discover a difference in behavior between individual statement execution and batch execution and you want to force individual statement execution, you can use the ingres.jdbc.batch.enabled system property to disable batch query execution. For more information, see the ingres.jdbc.batch.enabled property in the *Connectivity Guide*.

We recommend not using autocommit with batch execution. If batch execution is used with autocommit, and you cancel the batch execution, it is impossible to tell which statements were committed.

Note: Embedded SQL does not yet support batch query execution.

For more information, including guidelines on how to boost performance using this feature, see the *Connectivity Guide* and *OpenAPI User Guide*.

BOOLEAN Data Type

BOOLEAN can be used as a data type when defining a column in a table or a variable in a database procedure. Support for the BOOLEAN data type helps migrations from other database products.

Boolean columns accept as input the SQL literals FALSE and TRUE. In addition, due to automatic coercion rules, the strings 'FALSE' and 'TRUE' and the integers 0 and 1 are also acceptable for use in a Boolean column or variable. Input is not case sensitive.

The IS Boolean operator can be used in expressions. IS TRUE is true for a BOOLEAN TRUE, IS FALSE is true for a BOOLEAN FALSE, IS UNKNOWN is true for an Unknown (NULL) value. IS UNKNOWN is a synonym for IS NULL when dealing with Boolean values.

ORDER BY BOOLEAN results in grouping rows in this order: FALSE, TRUE, NULL.

The CREATE INDEX statement allows an index to be created on BOOLEAN columns.

Terminal Monitor output for a BOOLEAN column shows the literals FALSE and TRUE as unquoted strings.

CASE expressions can be used with BOOLEAN columns or literals. For example:

```
CASE expr WHEN cond1 THEN expr2
```

and

```
CASE WHEN search_cond1 THEN expr1
```

accept FALSE or TRUE in *condN* or *search_condN* or part thereof, and *exprN* can include BOOLEAN columns or literals.

The CAST operator supports casting BOOLEAN to and from character types and from the integer values 0 and 1. For example:

- CAST (BOOLEAN AS *character_type*) is allowed.
- CAST(*character_type* AS BOOLEAN) is accepted if the character type is the string 'FALSE' or 'TRUE', regardless of case.
- CAST(*integer_constant* AS BOOLEAN) is accepted for values 0 and 1.
- CAST(*integer_expression* AS BOOLEAN) is accepted if the integer expression has the value 0 or 1.

(Casting from 0 and 1 is an Ingres extension to the SQL standard, which does not allow it.)

For casting to strings, the data type must be of sufficient length (for example, CHAR(5) for FALSE) or silent truncation occurs (unless -string_truncation=fail is used at connect time). The shortcut CHAR(*expr*) returns a single character (that is, 'F' or 'T') because it is interpreted as CAST(*expr* AS CHAR(1)).

Internally, the BOOLEAN type is stored as a single-byte integer that can take only the values 0 and 1.

This feature adds to or changes the syntax of many statements, including ALTER TABLE, COPY TABLE, CREATE INTEGRITY, CREATE TABLE, CREATE TABLE…AS SELECT, DECLARE GLOBAL TEMPORARY TABLE, INSERT INTO, REGISTER TABLE, SELECT, UPDATE, WHERE clause of SELECT, DELETE, UPDATE, and JOIN source ON *search_condition*.

The BOOLEAN data type is supported in Ingres Star, Ingres Replicator, OpenAPI, embedded SQL, and by Ingres connectivity drivers.

Note: A pre-10.0 client will get an error if it tries to retrieve a result that includes a BOOLEAN column from a 10.0 or higher server.

Here are examples of using the BOOLEAN data type when creating a table or procedure:

```
CREATE TABLE example (column1 BOOLEAN NOT NULL);

CREATE PROCEDURE example_proc (flag BOOLEAN NOT NULL) AS
DECLARE
   var1 BOOLEAN;
BEGIN
    ...
END;
```

Here is an example of using the literals FALSE and TRUE in an SQL context:

```
INSERT INTO example VALUES (FALSE);
UPDATE example SET column1 = TRUE;
SELECT * FROM example WHERE column1 IS TRUE;
 ...
var1 = TRUE;
WHILE var1 IS NOT FALSE
 ...
```

For more information, see these sections in the *SQL Reference Guide*:

- BOOLEAN Data Type
- Storage Formats of Data Types
- BOOLEAN Literals
- Data Type Conversion Functions
- IS TRUE, IS FALSE, IS UNKNOWN Predicates
- Default Clause

New SQL Functions

New SQL functions provide compatibility with and ease migration from other database products. The new functions are as follows:

NVL

Specifies a value other than a null that is returned to your application when a null is encountered. For example in NVL(a,b), if 'a' is NULL then return 'b' else return 'a'.

This function is an alias for the IFNULL function.

NVL2

Returns different values based on whether the input value is NULL. For example, in NVL2(a,b,c), if 'a' is not null then return 'b' else return 'c'.

GREATEST

Returns the greatest of values in v1 through vN. Return type is that of the first parameter. If all of v1 through vN are NULL, then it returns NULL.

LEAST

Returns the least of values v1 through vN. Return type is that of the first parameter. If all of the values v1 through vN are NULL, then it returns NULL.

GREATER

Returns the greatest of values in v1 through vN. Return type is that of the first parameter. If any of v1 through vN are NULL, then it returns NULL.

LESSER

Returns the least of values v1 through vN. Return type is that of the first parameter. If any of the values v1 through vN are NULL, then it returns NULL.

For details, see the following sections in the *SQL Reference Guide*:

- NVL Function
- NVL2 Function
- IFNULL, NVL, NVL2 Result Data Types
- GREATEST, GREATER, LEAST, LESSER Functions

Terminal Monitor Silent Mode

The line-based terminal monitor for SQL can be run in silent mode. Silent mode shows only query output; it suppresses header and footer text, column titles, separators, and row counts.

In addition, the terminal monitor has new commands, which include:

\[no]silent

Switches silent mode on and off.

\[no]titles

Switches column titles on and off.

\[no]trim

Trims or does not trim spaces around column data.

\vdelimiter

Specifies the vertical separator character or resets it to the default. The character can also be set to SPACE, TAB, or NONE.

This feature allows simple reports to be created as SQL scripts and then run without having to edit the output.

To start the Terminal Monitor in silent mode use the –S flag on the sql command. For example:

```
sql –S demodb
```

To invoke silent mode while in a script use \silent. For example:

```
sql demodb
\silent
```

Each command issues a confirmation message when in non-silent mode.

For details, see the Terminal Monitor appendix in the *SQL Reference Guide*.

Daitch-Mokotoff Soundex Function

The SOUNDEX_DM function uses the Daitch-Mokotoff phonetic algorithm for finding similar sounding strings.

The Daitch-Mokotoff soundex function can return multiple soundex codes for a name. It returns one or more six-character codes in a comma-separated varchar string, up to a maximum of 16 codes. For example:

Name	**Codes Returned**
Moskowitz	645740
Peterson	739460,734600
Jackson	154600,454600,145460,445460

The function syntax is:

```
SOUNDEX_DM('string')
```

For details, see the SOUNDEX_DM section under String Functions in the *SQL Reference Guide*.

Check Digit Functions

Two SQL functions are used to generate and validate a check digit.

The function GENERATE_DIGIT generates a check digit for a specified string. The check digit can be used to help determine if a string, such as a credit card number, has been entered correctly.

The function VALIDATE_DIGIT tests if the check digit is valid.

Syntax for these functions is:

```
GENERATE_DIGIT('scheme', 'string')
```

which returns the check digit as a character.

```
VALIDATE_DIGIT('scheme', 'string')
```

which returns 1 for valid, 0 for invalid.

The *scheme* can be one of the following check digit schemes:

- LUHN or LUHN_A
- VERHOEFF or VERHOEFFNR
- ISBN
- ISBN_13
- ISSN
- UPC, UPC_A, or EAN_12
- UPC_E
- EAN, EAN_13, GTIN_13, or JAN
- EAN_8 or GTIN_8

For details, see the GENERATE_DIGIT and VALIDATE_DIGIT sections under String Functions in the *SQL Reference Guide*.

Recovery Server Error Handling

New options are provided for handling errors during recovery redo and undo processing.

Offline redo/undo processing occurs during installation startup if the installation previously shut down abnormally. Before other servers start, the Recovery Server processes the log file. It first re-does all updates in the log from the last consistency point, then un-does any open transactions.

Online undo processing occurs when the Recovery Server is performing pass-abort of transaction (after a rollback failure by a DBMS Server).

Previously, if an error occurred during this processing, recovery of the database was stopped, the database was marked inconsistent, and the Recovery Server startup was halted. The only option left to the DBA in such a situation was to roll forward the database. This feature provides additional options for handling such errors.

Two new Recovery Server parameters in CBF allow additional options for handling Recovery Server redo/undo errors:

- offline_error_action
- online_error_action

Note: You should use the new configuration parameters instead of the unsupported environment variable II_DMFRCP_STOP_ON_INCONS_DB. If II_DMFRCP_STOP_ON_INCONS_DB is set, the configuration parameters are ignored.

For more information about these configuration options, see the following sections in the *Database Administrator Guide:*

- Recovery Server Offline Error Handling
- Recovery Server Online Error Handling

Hash Join and Hash Aggregation Improvements

Improvements to hash join and hash aggregation enable faster queries and better concurrent query capability. Environments that will benefit are those with moderate to heavy concurrency (where memory usage is a potential problem), or in which the hash join or hash aggregation cannot fit in memory and is therefore spilled to disk before completing.

This feature adds five new configuration parameters to the DBMS Server component:

qef_hash_rbsize

The size in bytes of a read buffer for a hash operation (join or aggregation). There is one read buffer per hash operation, so we recommend relatively large values.

Default: 128 KB

qef_hash_wbsize

The size in bytes of a write buffer for a hash operation (join or aggregation). There are many write buffers per hash operation, and in the hash join case, the write buffer size interacts with the number of hash "buckets." Larger buffers can mean fewer (and therefore larger) buckets. Smaller buffers mean less room for rows in memory and more spillage. Installations must balance spill write efficiency (large qef_hash_wbsize) against memory usage and the ability to fit many buckets in memory (small qef_hash_wbsize).

Default: 16 KB

qef_hash_cmp_threshold

The minimum size of the compressible (non-key) part of a row to be considered for run-length compression. Run-length compression allows more hash-join rows to fit in memory, and reduces the size of hash join or hash aggregation spill to disk, at the expense of some extra CPU overhead.

Default: 128 bytes

qef_hashjoin_min

The minimum allocation, in bytes, for a hash join. Most installations do not need to change qef_hashjoin_min unless the optimizer underestimates the size of hash joins, causing excessive spillage.

Default: 0 (no minimum)

qef_hashjoin_max

The maximum allocation, in bytes, for a hash join. Most installations do not need to change qef_hashjoin_max unless the optimizer frequently over-estimates the size of hash joins, taking memory away from needier hash joins.

Default: 0 (which means use qef_hash_mem)

The qef_hash_rbsize and qef_hash_wbsize parameters may need to be changed to suit your environment. The qef_hashjoin_min and qef_hashjoin_max parameters will rarely need to be changed, since they are for preventing problems in extreme situations.

The Ingres upgrade process adds these parameters to an existing config.dat. If for some reason any parameters are missing from config.dat, the default values are used.

For details about the parameters, see the online help for Configuration-By-Forms (or the equivalent visual tool).

Query Execution Improvements Related to Partitioned Tables

Enhancements to the optimizer improve execution times for queries against partitioned tables, especially queries used in data warehousing applications.

Query optimization involving partitioned tables previously included partition pruning against constant predicates, and partition-compatible joining. The enhancements include:

- More complete analysis of partition pruning opportunities
- Join-time partition pruning
- Nested partition-compatible joining
- Partition-compatible aggregation

Direct I/O Improvements (UNIX)

Performance improvements were made to Ingres direct I/O capability in UNIX environments, including Linux.

Control of direct I/O is now configured at the system rather than DBMS level. Administrators can select direct I/O independently for tables, the transaction log, and build files (tables being loaded). Direct I/O is beneficial in large data warehouse environments, when the OS file system cache is ineffective (because of large sequential scans). OLTP-oriented installations may run better with direct I/O OFF, making use of the OS file system cache.

New direct I/O configuration parameters were added to Ingres:

ii.$.config.direct_io

Enables direct I/O for tables.

Default: OFF

ii.$.config.direct_io_log

Enables direct I/O for the transaction log.

Default: OFF

ii.$.config.direct_io_load

Enables direct I/O when writing bulk-load files for (bulk-load) COPY FROM, CREATE TABLE AS SELECT, MODIFY, and CREATE INDEX. Typically, direct_io_load ON is recommended only in cases where normal mode floods the operating system file cache.

Default: OFF

Note: If the platform or file system does not support direct I/O, the direct I/O parameters are ignored.

The configuration parameters dbms.*.direct_io and recovery.direct_io are now obsolete. During upgrade, if dbms.*.direct_io is ON in the old version, then config.direct_io and config.direct_io_log are set to ON.

You can set the new direct I/O parameters using the iisetres command.

File Allocation Improvements (UNIX)

Ingres can now take advantage of the "fallocate" function of certain file systems, which reserves (pre-allocates) disk space as files are written. When applied to table files, pre-allocation may slow creation of the table somewhat, but subsequent reading of the table may be faster, since the table is more likely to be allocated sequentially on disk. The improvement is most likely to be seen when table scans are common, such as in data warehousing installations with large tables.

To take advantage of pre-allocation, the table should be created with the ALLOCATION= and EXTEND= parameters set to reflect the expected table size. The default allocation and extend values are too small to be useful for pre-allocation.

This improvement adds a new configuration parameter to the DBMS Server component:

fallocate

Enables use of file pre-allocation through fallocate. If the file system does not support fallocate or an equivalent, the parameter setting does not matter (is OFF).

Default: OFF

Standard Compression Performance Improvements

Internal changes to the DMF data structures improve the performance of standard compression, especially for tables with many columns. (Standard compression is specified by using the WITH COMPRESSION=DATA clause on the MODIFY or the DECLARE GLOBAL TEMPORARY TABLE statement.)

Compression makes rows smaller and is useful in I/O intensive applications, such as data warehousing.

Miscellaneous Changes

The following miscellaneous changes are included in Ingres 10.

New Way to Specify Unicode Literals

To specify a Unicode literal value within a non-Unicode command string compatible with Microsoft SQL server, the following syntax is now available:

```
N'unicode_string'
```

For more information, see Unicode Literals in the *SQL Reference Guide*.

New Terminal Type Added

The PuTTY terminal type is added. For more information, see Terminal Names in the *Character-based Querying and Reporting Tools User Guide*.

ingstop -f Command Parameter Updated

The –f (force immediate shutdown) parameter of the ingstop command (to shut down an Ingres instance) is updated to be more effective. This parameter typically is used with the –k (kill instance) parameter.

For more information about the ingstop –f parameter, see the ingstop command description in the *Command Reference Guide*.

IPM Lock List Display More_Info Screen Updated

The Interactive Performance Monitor Lock List Display More_Info screen now displays the following new information for a selected lock list:

- Internal session ID number
- External operating system process ID number

For more information about the updates to this screen, see Lock List Display More_Info Screen in the *Interactive Performance Monitor User Guide*.

CBF and VCBF Log Writer Update

In Configuration-By-Forms or Visual Configuration-By-Forms, the Recovery Server will accept a configuration of log_writer=0. A useful setup to ensure that there is only one log writer in the installation for log write optimization configures the DBMS log_writer=1 and the Recovery Server log_writer=0.

Type and Function Names Logged in II_DBMS_LOG

If the DBMS Server is linked with user-defined types or functions, the type or function names are logged in the DBMS log (II_DBMS_LOG).

dmf_tcb_limit Parameter Extended

The dmf_tcb_limit parameter in config.dat extends to all Table Control Blocks (TCBs), including partition TCBs. The default dmf_tcb_limit is now 10,000.

This is documented in Partitioning Schemes in the *SQL Reference Guide*.

Chapter 2: Connectivity Enhancements in Ingres 10

This section contains the following topics:

JDBC 4.0 Support

The Ingres JDBC driver is JDBC 4.0 compliant and requires Java Runtime Environment 1.6.

For more information about JDBC 4.0 support, see the *Connectivity Guide*.

64-bit ODBC Driver

Ingres 10 provides a 64-bit ODBC driver as part of the Ingres port to 64-bit Windows.

ODBC Driver Names

The Ingres instance ID, rather than the Ingres version, is now used to identify Ingres ODBC drivers. The Ingres ODBC driver is installed with the driver names "Ingres" and "Ingres xx," where "xx" is the value of II_INSTALLATION.

This naming convention allows any number of Ingres instances to include ODBC driver names specific to the instance. If you upgrade to a new release of Ingres in the same installation path, the ODBC driver names are unaffected and require no modifications to ODBC configuration or ODBC connection strings.

For more information, see ODBC Driver Names in the chapter "Understanding ODBC Connectivity" in the *Connectivity Guide*.

IngresType.IngresDate Parameter Type in .NET Data Provider

The Ingres .NET Data Provider sends .NET DateTime parameter data to the Data Access Server and the DBMS Server. A new IngresType.IngresDate parameter type allows the application to specify that the DateTime parameter data be sent to the DBMS with an INGRESDATE data type and format.

As of Ingres 9.1 (also know as Ingres 2006 Release 2), the Ingres .NET Data Provider sends DateTime data as ANSI TIMESTAMP_WITH_TIMEZONE type, rather than INGRESDATE type. In some cases, the change in data type can produce unwanted effects after upgrading to new releases of Ingres.

The new parameter type directs the data provider to send the parameter data as an INGRESDATE data type and format. This feature should be used only in applications that access only INGRESDATE columns. Applications that access ANSI DateTime columns may experience side effects of this feature due to loss of fractional second information and time zone format differences when the DateTime parameter data is sent as INGRESDATE.

Example:

```
IngresConnection conn = new IngresConnection();
IngresCommand cmd;
cmd = new IngresCommand();
cmd.CommandText = "INSERT INTO my_table VALUES (?)";
cmd.Connection = conn;
cmd.Parameters.Add(new IngresParameter("mydateparm", IngresType.IngresDate));
cmd.Parameters[0].Value = DateTime.Now;
```

For details about the IngresType.IngresDate parameter type, see the *Connectivity Guide*.

Named Parameters in Parameterized Queries in .NET Data Provider

The Ingres .NET Data Provider now supports named parameters in queries. Named parameters allow the parameter values to be set regardless of the order that they appear in the SQL statement.

A named parameter marker consists of an initial character of question mark (?), at-sign (@), or colon (:) followed by a name. The name is an alphanumeric identifier or an integer. Although the recommended parameter marker character is the question mark, the Ingres .NET Data Provider supports all three characters to ease migrations from other database products.

Prior releases supported the unnamed (positional) parameter marker placeholder syntax using the question mark (?), similar to ODBC and JDBC.

Using named and unnamed parameters in the same SQL statement is not allowed.

For more information, see the *Connectivity Guide*.

Positional Parameter Support in Drivers

The Ingres JDBC Driver, ODBC Driver, and .NET Data Provider support positional parameters in database procedure invocation.

The drivers use positional parameter support in the DBMS, when available. This provides better performance when executing database procedures because the driver does not need to query the database catalog to determine parameter names and map ordinal position to named parameters.

In ODBC, no application changes are required.

TCP/IP for Local Communications on Windows

TCP/IP (Ingres protocol tcp_ip) can now be used for local inter-process communications (IPC) on Windows as an alternative to named pipes, which is the default.

The default provides the best overall performance both locally and remotely with Ingres Net. However, when the DBMS is accessed primarily from remote clients that are “direct connect”-compatible with the server, using TCP/IP as the local IPC and “direct connect” (vnode attribute connection_type = direct) can improve response time.

To configure TCP/IP as the local IPC

1. Shut down Ingres.
2. Set Ingres environment variable II_GC_PROT:

   ```
   ingsetenv II_GC_PROT tcp_ip
   ```
3. Start Ingres.

Note: Only protocol tcp_ip, not wintcp, is supported as a local IPC. If II_GC_PROT is not set, then named pipes is used.

With named pipes, “direct connect” is supported only across Windows machines. With the tcp_ip local IPC, “direct connect” is also supported to and from Linux and UNIX on Intel-based machines.

Note: The Ingres environment variable II_GC_REMOTE must now be configured on Windows servers (previously only required on UNIX and Linux) to allow direct access. For configuration and compatibility requirements for “direct connect,” see the *Connectivity Guide*.

Performance Improvements to Communications Protocol Drivers on Windows

Various performance improvements have been made to the local and remote Ingres protocol drivers (named pipes and tcp_ip). All changes are internal to the drivers and require no configuration changes. The result should be slightly improved response times for both local and remote access.

Chapter 3: Supportability Enhancements in Ingres 10

This section contains the following topics:

Process ID (PID) for GCF Servers Displayed in iimonitor

The iimonitor "show server" command output for the Communications Server (GCC), Data Access Server (GCD), and Name Server (GCN) now displays the process ID information. Also, the GCC and GCD servers display the actual symbolic port address in the addr: field of the Protocols section.

Here is an example of the iimonitor show server command output, with new fields shown in bold:

```
IIMONITOR> show server
    Server: 32874
    Class: COMSVR
    Object: *
    Pid: 25373

Connected Sessions ( includes system and admin sessions )
    Current: 3

Active Sessions ( includes user sessions only )
    Inbound sessions:
      Current: 0
      Limit: 64
    Outbound sessions:
      Current: 0
      Limit: 64

Protocols
    TCP_IP host:ingnet02 addr:MB3 port:23059
Registry Protocols
    Not Enabled
```

For more information about the iimonitor "show server" command, see the iimonitor command description in the *Command Reference Guide*.

OpenAPI Writing of DBMS Trace Messages to File

OpenAPI can redirect trace messages returned from the DBMS to a disk file.

The II_API_SET environment variable can be set to a value of PRINTTRACE to enable writing of GCA_TRACE messages to the default output file iiprttrc.log. Output can be directed to a specific file by adding the TRACEFILE argument.

Here are examples in the UNIX environment:

1. Enable OpenAPI tracing of GCA_TRACE messages to default output file iiprttrc.c:

   ```
   setenv II_API_SET printtrace
   ```

2. Enable OpenAPI tracing of GCA_TRACE messages to output file tracing.log:

   ```
   setenv II_API_SET "printtrace; tracefile tracing.log"
   ```

Setting II_API_SET does not affect passing of trace information to the application by OpenAPI.

Chapter 4: Usability Enhancements in Ingres 10

This section contains the following topics:

Improved Visual Tools Dialogs

Certain dialogs and menus in Visual DBA, Ingres Import Assistant, and Ingres Network Utility were redesigned to make these tools more intuitive and user friendly. For example, the advanced features on many dialogs are now hidden.

The changes are designed to help new users more easily perform basic tasks such as establishing a remote connection, creating a database, creating a table, populating a table, generating statistics, backing up and restoring a database, and creating a user.

Terminal Monitor Usability Enhancements

The following command:

```
ingres dbname
```

now connects the user to the named database and starts the line-based Terminal Monitor for SQL. (In previous releases, it opened a QUEL session.)

When the terminal monitor session is started, a simple message displays usage information helpful to new users (shown in bold here):

```
$ sql demodb
INGRES TERMINAL MONITOR Copyright 2010 Ingres Corporation
Ingres Linux Version II 10.0.0 (int.lnx/00)NPTL login
Thu Apr 22 06:24:54 2010
Enter \g to execute commands, "help help\g" for help, \q to quit

continue
*
```

In addition, the history recall feature is now enabled by default in Linux and UNIX environments. You can recall text previously entered in your terminal monitor session by using the Up and Down arrow keys.

To disable this feature for the duration of the terminal monitor session, start the terminal monitor with the -nohistory_recall flag.

You can enable or disable this feature for the instance by setting the new configuration parameter ii.*.tm.history_recall.

Note: The -nohistory_recall flag is not applicable on Windows.

Index