IIAS Columnar Incremental Schema Backup and Restore Feature

Version: 1.0.22

Last modified: April 19, 2020

Terminology

This document discusses both cumulative and delta incremental schema backup. A reference to an 'incremental backup' in the document is, unless specifically mentioned, either a cumulative or delta incremental backup.

Introduction

This feature provides the ability to do cumulative incremental or delta incremental backup of a schema followed by full restore of the schema or table(s) within the schema.

In order to take advantage of this new feature you will need the following:

- A new schema created with the "ENABLE ROW MODIFICATION TRACKING" attribute. Permanent organized by column tables created within this schema will be enabled for row modification tracking, meaning they will contain 3 additional hidden columns for tracking incremental modifications to the rows between two backup's.
- Use of new options provided on the `db_backup` and `db_restore` commands for schema incremental backup.

Schema level backup (both full and incremental) of a schema enabled for row modification tracking will allow read access as well as concurrent insert, update, and delete.

What's new in 1.0.22

The following functionality was added:

- Improved support for migration of existing schemas with new option: `db_restore -enable-row-modification-tracking` to enable a restored schema for row modification tracking
- ADMIN_COPY_SCHEMA will copy ROWMODIFICATIONTRACKING attribute of the schema
- RENAME TABLE commands issued between full and incremental schema backups are captured and properly replayed during `db_restore -schema` and `db_restore -table` or `db_restore -tablefile`
- Improvements in backup and restore performance

Schema enabled for row modification tracking

CREATE SCHEMA syntax

```
>>-CREATE SCHEMA------------------------------------------------>

 >--+-schema-name------------------------------------+----------->
    +-AUTHORIZATION--authorization-name--------------+
    '-schema-name--AUTHORIZATION--authorization-name-'

 >--+---------------------------+------------------------------->
    '-DATA CAPTURE--+-NONE----+-'
                    '-CHANGES-'

 >--+-----------------------------------+------------------------>
    '-ENABLE ROW MODIFICATION TRACKING-'

 >--+--------------------------+------------------------------+-><
    | .----------------------. |
    | V                      | |
    '---schema-SQL-statement-+-'
```

Notice it is not the default and therefore to take advantage of incremental schema backup and restore explicit schema creation with the new attribute is necessary.

Catalog changes

A new column in SYSCAT.SCHEMATA called ROWMODIFICATIONTRACKING will indicate whether the schema is enabled for row modification tracking or not with a value of ‘N’ or ‘Y’.

db2 “DESCRIBE TABLE SYSCAT.SCHEMATA”

```
                                Data type                     Column
Column name                     schema    Data type name      Length     Scale Nulls
------------------------------- --------- ------------------- ---------- ----- ------
SCHEMANAME                      SYSIBM    VARCHAR                    128     0 No
OWNER                           SYSIBM    VARCHAR                    128     0 No
OWNERTYPE                       SYSIBM    CHARACTER                    1     0 No
DEFINER                         SYSIBM    VARCHAR                    128     0 No
DEFINERTYPE                     SYSIBM    CHARACTER                    1     0 No
CREATE_TIME                     SYSIBM    TIMESTAMP                   10     6 No
AUDITPOLICYID                   SYSIBM    INTEGER                      4     0 Yes
AUDITPOLICYNAME                 SYSIBM    VARCHAR                    128     0 Yes
AUDITEXCEPTIONENABLED           SYSIBM    CHARACTER                    1     0 No
DATACAPTURE                     SYSIBM    VARCHAR                      1     0 No
ROWMODIFICATIONTRACKING         SYSIBM    VARCHAR                      1     0 No
REMARKS                         SYSIBM    VARCHAR                    254     0 Yes
```

db2 “SELECT CAST(SCHEMANAME as CHAR(5)) as SCHEMANAME, ROWMODIFICATIONTRACKING FROM SYSCAT.SCHEMATA WHERE SCHEMANAME='S1'”

```
SCHEMANAME ROWMODIFICATIONTRACKING
---------- -----------------------
S1         Y

  1 record(s) selected.
```

Tables created in a schema enabled for row modification tracking

Column-organized tables created in a schema enabled for row modification tracking will be enabled for row modification tracking themselves. In order to track incremental changes to the table data between a full and incremental backup, 3 implicitly hidden columns will be added for each table when it is created. A table enabled for row modification tracking will have the following 3 new columns:

Column Name	Datatype	Allow NULL	Description
SYSROWID	BIGINT	No	Column that uniquely identifies each row in the table. The ID is unique across all database partitions. The values for this column are generated by a SEQUENCE.
CREATEXID	BIGINT	No	Stores the ID of the transaction that added the row to the table.
DELETEXID	BIGINT	No	Stores the ID of the transaction that deleted the row. It will be zero if the row is not deleted.

After a row is inserted and assigned a value for SYSROWID, this value will not change even if the row is updated. Please be aware that the above column names will now be reserved by IBM for use in tables enabled for row modification tracking. Users attempting to use the same column name in the table will receive an error on create or alter table time.

Only permanent columnar tables created in a schema enabled for row modification tracking will be enabled for row modification tracking. The follow tables will not be enabled for row modification and thus will not contain the extra 3 columns:

- Temporary tables
- External tables
- Materialized query tables
- Tables not in the list above which are also not organized by column (i.e. organized by row, insert time, dimensions-clause, etc.)

The above tables can be created in a schema enabled for row modification, but there are couple caveats to be aware of:

- Temporary tables will not be captured by schema backup
- External tables, and Materialized query tables when captured by full or incremental schema backup will only capture the definition of the table and not the contents
- Tables not organized by column (excluding the 3 types mentioned above) which exist in a schema enabled for row modification tracking will cause the full and incremental schema

backup to fail. These tables must be stored elsewhere if you want to backup a schema enabled for row modification tracking.

You can see the extra columns for your table when doing a describe of the table:

```
db2 "DESCRIBE TABLE S1.T1"

                                Data type                     Column
Column name                     schema    Data type name      Length     Scale Nulls
------------------------------- --------- ------------------- ---------- ----- ------
SYSROWID                        SYSIBM    BIGINT                       8     0 No
CREATEXID                       SYSIBM    BIGINT                       8     0 No
DELETEXID                       SYSIBM    BIGINT                       8     0 No
C1                              SYSIBM    INTEGER                      4     0 Yes

  4 record(s) selected.
```

Since they are implicitly hidden columns with values internally generated you will not see them when doing a select * from the table for example, and you do not need to specify them when doing an insert, update or delete statement. Attempting to modify the columns (insert/update/delete/alter) will either be ignored or result in an error.

db_backup

Schema-level backup can be performed by passing the schema name to the db_backup utility using -schema <schema_name> option. If this option is omitted, the tool invokes db2 utilities to backup the entire database.

```
db_backup -type ONL|INC|DEL -path PATH -schema SCHEMANAME
          [-compress NO|GZIP|LZ4] [-format TEXT|BINARY]]

  -type onl|inc|del
     onl - Full online backup
     inc - Online Cumulative incremental backup
     del - Online Delta incremental backup (also known as a differential backup)

  -path PATH
    Target directory for backup image. Use ( ) or (,) or ( ,) to separate multi-path.

  -schema SCHEMANAME
    Schema to be backed up.

  -compress NO|GZIP|LZ4 (optional)
    Compress setting for the backup. By default, it is set to NO.

  -format TEXT|BINARY (optional)
    Format for the backup files. By default, it is set to BINARY.
```

Note The default format changed from TEXT in 1.0.20 and prior to BINARY in 1.0.21

General awareness for cross schema dependencies

Please be aware that schema backup will only capture objects in the schema to backup and thus any cross-schema dependencies could lead to problems during restore. For example, if a DISTINCT TYPE is created in schema S2 and then used in a table in schema S1, a schema backup of S1 followed by restore of S1 would depend on the existence of the type in S2 at the time of the restore or else restore will fail. Failure to create some types of non-table objects inside of S1 which depend on objects outside of the schema may be tolerated during restore, a warning will be issued during restore and the restore will return success (the list includes Functions/Procedures/Views/MQT's/Variables). Failure to create a table object for any reason will log an error during restore and fail the restore.

Handling for different table types under schema backup

- External Tables – The definition of the external table will be captured and restored but not the contents.
- Materialized query tables - The definition of the MQT will be captured and restored but not the contents. The user will need to refresh the MQT after restore.
- Temporary tables – These are not captured by schema backup.
- Any permanent tables not organized by column will cause schema backup of a schema enabled for row modification tracking to fail (since these tables would not be enabled for row modification tracking).

db_restore

Schema level restore can be performed by passing the schema name to the db_restore utility using -schema <schema_name> option. To restore a single or multiple tables, -table or -tablefile options can also be used respectively.

```
db_restore -type FRH|INC|DEL -path PATH -timestamp TIMESTAMP -schema SCHEMA
           [-table TABLE | -tablefile FILEPATH] [-drop-existing]
           [-enable-row-modification-tracking]
```

```
  -type frh|inc|del
     frh – Restore from a full backup
     inc – Full restore from a cumulative incremental backup
     del – Full restore from a delta incremental backup.

  -path PATH
    Path to the directory where backup exists. Use ( ) or (,) or ( ,) to separate multi-path

  -timestamp TIMESTAMP
    Timestamp of the backup.

  -schema SCHEMA
    Schema to be restored. Schema must not exist, or -drop-existing option must be specified.

  -table TABLE (optional)
    Table to be restored. Table must not exist, or -drop-existing option must be specified.
  -tablefile FILEPATH (optional)
    Full path to the file with tables to be restored.

  -drop-existing (optional)
    Drop existing schema and its table/non-table objects before restore. If used in conjunction
with -table option, only affected table is dropped.

  -enable-row-modification-tracking (optional)
    Restores a schema backup image taken from schema not enabled for row modification tracking
and enables target schema for row modification tracking.
```

Restore from an incremental backup

Restore of a cumulative or delta incremental backup always starts with restore of a full backup image first. All backup images required to restore the current cumulative or delta incremental backup image are automatically determined and restored in chronological order.

Using "-path"

If the location of the backup images has changed since the time of the backup then all relevant paths must be provided within the `-path` input parameter.

Using "-table"

Table name specified to the option applies to name that existed as of start of `db_backup` command that created the image. Any RENAME TABLE operations that happened after backup was taken are ignored. If the table was renamed after backup was taken, then after restore two copies of the table might exist: one that was restored and the renamed original.

Using "-drop-existing"

This option must be specified if target schema/table exist.

Note some schema objects can not be dropped via `db_restore` with the `-drop-existing option`. If you receive a failure during restore to drop any objects you will need to drop the objects manually and re-issue the restore.

Using "-enable-row-modification-tracking"

All tables that can be created as row modification tracking will be, and will have SYSROWID, CREATEXID, DELETEXID columns created and properly populated.

If a table cannot be enabled for row modification tracking, it will still be created and restored with data. These tables must be manually dropped from the schema or moved to another schema before incremental schema backup can be run.

Only schema backup images created on or after 1.0.22 can be used.

Migration considerations

Users may alter their schema to enable row modification tracking, however this does not modify any of the tables in the schema to enable row modification tracking.

The syntax for alter schema is the following:

```
>>-ALTER SCHEMA--schema-name------------------------------------>

    .--------------------------------------------------------------.
    V                                                              |
>----+-DATA CAPTURE--+-NONE----+--------------------------------+-+-><
     |               '-CHANGES-'                                |
     +-ENABLE ROW MODIFICATION TRACKING-------------------------+
```

Values stored in 3 new columns will require additional space, and therefore a table that is enabled for row modification tracking will consume more pages than exactly same table that is not.

Only new tables created in the schema will be enabled for row modification tracking. All columnar tables in the schema must be recreated after a schema is enabled for row modification tracking, before incremental schema backup will be possible.

Table migration

Recreating tables can be done through any means, for example using ADMIN_MOVE_TABLE, or CREATE TABLE … AS … WITH DATA (select * from one table into a new table in the same schema).

The target table will be created as enabled for row modification tracking if the target schema is enabled for row modification tracking and target table type supports row modification tracking.

Whether the source table is enabled for row modification tracking or not has no impact on what will be chosen for the target table. If the user attempts to select any of the new hidden columns from the source table where the target is created in a schema enabled for row modification, this will be treated as if the user has attempted to create a table with a user column of the same name and result in an error (column name already exists). Values for the new hidden columns will not be carried over from the source to the target during a CREATE TABLE … AS *(fullselect)* WITH DATA operation.

Schema migration

Migrating schemas can be done through ADMIN_COPY_SCHEMA or `db_restore -enable-row-modification-tracking`.

To use ADMIN_COPY_SCHEMA source schema must be enabled for row modification tracking via `ALTER SCHEMA … ENABLE ROW MODIFICATION TRACKING`.

Both operations will copy/restore all tables, but only columnar tables will be enabled for row modification tracking. Other tables must be manually dropped or moved to another schema before incremental schema backup will be possible.

ADMIN_COPY_SCHEMA with a *copymode* of 'COPY' or 'COPYNO' will fail if at least one table is defined as DISTRIBUTE BY RANDOM or contains GENERATED ALWAYS AS IDENTITY column.

DB2 scope of use for objects in schema enabled for row modification tracking

- db2load into a table enabled for row modification tracking which also contains a user defined identity column will fail.
- db2load into a table enabled for row modification tracking which is also defined as DISTRIBUTE BY RANDOM will fail.
- Exception tables for db2load and INGEST cannot be columnar and thus the recommended approach to use an exception table for db2load or INGEST into a table enabled for row modification tracking is to create a row organized table in another schema (different schema because the row organized table will cause incremental schema backup to fail). In addition to that, the exception table must have a similar definition including the hidden columns defined for row modification tracking except the SYSROWID can not be an identity column since these are not supported in exception tables.
    - For example:
        - CREATE SCHEMA S1 ENABLE ROW MODIFICATION TRACKING
        - CREATE SCHEMA S2
        - CREATE TABLE S1.T1 (C1 INT)
        - CREATE TABLE S2.LOADEXTBL (SYSROWID BIGINT NOT NULL, CREATEXID BIGINT NOT NULL, DELETEXID BIGINT NOT NULL, C1 INT)
        - LOAD FROM <file> OF DEL INSERT INTO S1.T1 FOR EXCEPTION S2.LOADEXTBL
- The maximum number of user defined columns in a table enabled for row modification tracking is reduced to 1009.
- IMPORT using CREATE or REPLACE_CREATE options into a table enabled for row modification tracking may fail.
- ADMIN_COPY_SCHEMA with a *copymode* of 'COPY' or 'COPYNO' will fail if at least one table is defined as DISTRIBUTE BY RANDOM or contains GENERATED ALWAYS AS IDENTITY column.
- No support for alter of a table enabled for row modification tracking to be a materialized query table.
- None of the 3 new hidden columns (SYSROWID, CREATEXID, DELETEXID) are supported within the *distribution key* of the table (i.e. not allowed in the DISTRIBUTE BY HASH column list).
- No support for Q Replication of tables in schemas enabled for row modification tracking. Do not subscribe these schemas/tables for replication.
- No support for db2move if the source or target is a table in a schema enabled for row modification tracking.
- None of the 3 new hidden columns can be included as part of an index key or in an enforced constraint

- No support for System-period temporal tables or Bitemporal-period temporal tables with row modification tracking enabled, however Application-period temporal table will support row modification tracking.
- No support for CREATE VIEW … WITH ROW MOVEMENT with row modification tracking enabled.

db_backup scope of use

Scope of use for schema backup (any schema):

- Incremental schema backup (`-type inc` and `-type del`) is only supported for schemas which are enabled for row modification tracking.
- Concurrent Select/Insert/Update/Delete operations are supported against the row modification tracking schema being backed up. Some DDL will remain blocked. Full access to other schemas not under backup will remain unchanged (i.e. no restrictions).
- No multi-schema backup support for schemas enabled for row modification tracking. If any one of the schemas specified in a multi-schema backup are enabled for row modification tracking, the backup will fail.
- Backup of a schema enabled for row modification tracking will fail if Row and column access control (RCAC) is in effect for any data inside the schema.
- Can only backup entire schema and not a single or select number of tables.
- Cannot backup a schema that contains a period (.) in its name.
- The `DB2 LIST HISTORY` command will not show schema level backups or restores. Use `db_backup`/`db_restore -history` instead.
- Point in time roll forward is not supported.
- Label-based access control (LBAC) information is not supported on columnar tables and not scoped by a schema so this information will not be included in a schema backup.
- Typed tables and views not supported.
- JAR objects stored in the database (via SQLJ.INSTALL_JAR) for use in procedures will not be captured by the schema backup. This is no different than the behavior for any other external procedure like a C routine. It is recommended customers save and restore these themselves.
- Cannot backup a schema that has tables with single/double quotes in the name.
- If schema specified via `–schema` option has double quotes in the name or is case-sensitive, it must be enclosed with single and double quotes (e.g. Schema"1" must be specified as `–schema '"Schema"1""'`)

Scope of use for schema backup of schemas enabled for row modification:

If your schema contains any of the following, then schema backup (full or incremental) will fail:

- Tables that are row organized. Not supported
- Tables with geospatial data. Not supported.
- Tables with LOBs columns defined >= 64K or DBCLOB >= 32k. Not supported.

- An index in your schema to backup is defined on a table in a different schema, or a table in your schema to backup has an index defined in a different schema. This would create a cross schema dependency that is very likely to cause restore to fail and thus is blocked at backup time.
- Methods: Created via CREATE METHOD
- Triggers: Created via CREATE TRIGGER
- Reference types
- Anchor data types
- Function templates: Any functions defined with CREATE FUNCTION … AS TEMPLATE

If any of the following DDL was issued between the previous schema backup and the current schema incremental backup, the incremental backup will fail (the response here is to instead run a full schema backup):

- DROP/RENAME TABLESPACE. Note this applies to any tablespace and not just tablespaces used by a given schema to backup.
- DROP SCHEMA
- ALTER TABLE … ADD FOREIGN KEY

The following index operations will not be captured by an incremental backup. Indexes will be captured in a full backup, but any index change will not be captured by an incremental (it will be ignored by incremental backup and thus will not be part of the schema after restore):

- CREATE/DROP/ALTER/RENAME INDEX

If any of the following DDL was issued between the previous schema backup and the current schema incremental backup it will result in a full backup of the table referenced in the DDL (other tables in the schema will be unaffected, i.e. only incremental changes captured):

- CREATE TABLE
- TRUNCATE TABLE
- LOAD using REPLACE option
- IMPORT using REPLACE option
- ALTER TABLE ... ACTIVATE NOT LOGGED INITIALLY WITH EMPTY TABLE

Special considerations for tablespace restore and rollforward when used in conjunction with schema backup and restore. It is recommended not to mix these types of backup and restore since some inconsistencies could arise, for example:

- CREATE SCHEMA S1 ENABLE ROW MODIFICATION TRACKING
- CREATE TABLESPACE TBSP1 …
- CREATE TABLE S1.T1 (C1 INT) IN TBSP1
- Db2 tablespace backup of TBSP1
- FULL schema backup of S1
- Insert into table S1.T1 values (1), commit
- Insert into table S1.T1 values (2), commit
- DELTA incremental schema backup of S1

- Db2 tablespace restore and rollforward (of TBSP1) to just after insert of 1, and prior to insert of 2.
- Insert into table S1.T1 values (3), commit
- DELTA schema backup of S1
- Cumulative INCREMENTAL schema backup of S1

Currently S1.T1 has rows with values 1 and 3. However, if the user does a restore of the final DELTA schema backup captured above S1.T1 would have rows with values 1, 2, and 3. Restore of the final cumulative incremental schema backup would result in S1.T1 having rows with values 1 and 3.

Use of database and schema backup and restore in conjunction

It is not recommended to mix these types of backup and restore. Restoring a database level image taken at a timestamp invalidates all schema level backup images taken after that timestamp. Schema backup does not contain information about db-level objects (like TABLESPACE) and schema restore does not ensure they are re-created if not present. In addition to that, database restore and rollforward will overwrite (put back in time) the history table IBM_SAILFISH.GRANULAR_BACKUP. See "History table considerations for full database restore" section below.

db_restore scope of use

General scope of use:

- Concurrent read/write access to the schema or table undergoing restore is not blocked. See section "Concurrent access to the schema during db_restore below.
- Cannot restore a single schema or table(s) from a database-level backup.
- Cannot restore a database from a schema-level backup.
- Cannot restore a schema with a leading space in the name.
- Cannot restore a table with a leading space in the name of primary key.
- Table names specified via `–table` option are case-insensitive
- If table specified via `–table` option is case-sensitive, it must be enclosed with single and double quotes (e.g. Table1 must be specified as `–table '"Table1"'`)
- Table names specified in file via `–tablefile` option are case sensitive
- `GRANT … ON TABLE` permissions are not restored with `–table` or `–tablefile` options

Concurrent access to the schema during db_restore

Read and Write operations to a schema or table undergoing a restore are not fully blocked during the operation. A concurrent application accessing the schema during restore can see incorrect results or even modify data leaving the schema in a logically inconsistent state after restore. It is even possible to cause restore to fail if concurrent application interferes with restore process (e.g. creates/locks a table).

It is highly recommended users ensure no concurrent access is allowed during restore either via application control, or via QUIESCE DATABASE IMMEDIATE prior to schema or table level restore. After successful restore access can be resumed via UNQUIESCE DATABASE command. Please consult documentation for QUIESCE DATABASE IMMEDIATE operation prior to use – it will force all active connections for all schemas in the database and prevent new ones until UNQUIESCE DATABASE.

While a database is quiesced `apissues` may report Major alert “351: Database availability issue” until the database is unquiesced. This is expected behaviour. It is possible to disable this alert via console config update. Please note that when the issue is reported, console may not be able to detect that database is available after UNQUIESCE until all “351” events are deleted.

History table considerations after full database restore

Full restore of database will also restore (put back in time) data stored in IBM_SAILFISH.GRANULAR_BACKUP table. Information in that table can be used to determine path/location of all schema backup images. Users must preserve history table explicitly, and then restore relevant information after restoring the database to facilitate removal of schema backup images invalidated by database restore:

1. Store data from history table
   ```
   db2 "create external table '/scratch/history.bkp' as (SELECT *
   from IBM_SAILFISH.GRANULAR_BACKUP)"
   ```
2. Restore database
   ```
   db_restore -type frc -path <path> -timestamp <timestampA>
   ```
3. Restore history table data
   ```
   db2 "insert into IBM_SAILFISH.GRANULAR_BACKUP select * from
   external '/scratch/history.bkp' where TIMESTAMP > <timestampA>"
   ```

Examples:

```
# Create schema and populate
CREATE SCHEMA S1 ENABLE ROW MODIFICATION TRACKING
CREATE TABLE S1.T1 (c1 int)
INSERT INTO S1.T1 values (1)

# Capture a full online schema backup
db_backup -type onl -schema S1 -compress lz4 -format binary -path p1

# Insert some more data, then capture a cumulative incremental schema backup
INSERT INTO S1.T1 values (2)
db_backup -type inc -schema S1 -compress lz4 -format binary -path p1
```

```
# Insert more data, then do restore
INSERT INTO S1.T1 values (3)
db_restore -type inc -timestamp <timestamp of INC backup> -drop-existing -schema S1 -path p1
```

The above restore will first drop the schema and its contents, followed by creation of S1 again (enabled for row modification tracking), followed by restore of the ONL backup, followed by restore of the INC backup. The resulting schema S1 will contain table T1 with two rows in it (values 1 and 2 for the rows). Notice that request to restore the incremental backup will restore all dependant backup images first, followed by restore of the requested image.