Add and detach data partitions from DB2 partitioned tables using InfoSphere DataStage

Ranjeeta Pegu (ranjepeg@in.ibm.com)  
Staff Software Engineer, B2B Analytics Services  
IBM  
15 January 2015

Table partitioning has benefits like the ability to roll in (including data in a table) and roll out (detaching data from a table) and improving query performances, particularly in data warehouse and decision support system environments. These activities are mostly performed manually by DBAs, but in a real-time data warehouse, where there is continuous flow of data via ETL tools, excessive manual interventions are unwanted and can impact the ETL running process. This tutorial discusses how the ETL process can be used for adding empty partitions to a partitioned table and removing unwanted partitions from the table without any manual or DBA intervention.

Introduction

The table partitioning feature of DB2 enhances query performance and improves performance when archiving or deleting data in large tables. In a partitioned table, the data is divided into multiple ranges or data partitions (storage objects) depending on the partition. So it is important to define the granularity of partition range, keeping in mind the data roll-in (attaching or adding new a partition to a partitioned table) and roll-out (detaching a partition from a portioned table) strategies. Each data partition has to have a unique name and defined boundaries.

The selection of partition ranges or key columns depends on what you want to achieve by partitioning. For example, if the table partitioning is aimed at achieving date-time roll-in/roll-out, the partition key column should be a date-time column. Partition key should make the data management task easy, but in reality, as large data trickles in, it requires more manual interventions and maintenance activities.

The tutorial discusses a process by which InfoSphere® DataStage® can be used for data management of a table with DB2® table partitioning on multiple key columns.

Create a DB2 partitioned table

DB2 provides a lot of flexibility for creating partitioned tables with a single date column, if you have a contract table with 10 years of data and you want to partition it by year. The following create table syntax illustrates how to do this easily.
Listing 1. Create partitioned table code by date

```sql
CREATE TABLE CONTRACTS (contract_id INT, contractor_id INT, Contract_Date DATE)
    PARTITION BY RANGE (contract_date)
    (STARTING '1/1/2004' ENDING '12/31/2014' EVERY 1 YEARS)
```

To check the partition details like data partition name, schema name of table, name of table to which partition belongs, data partition identifier ID, Low key value, high key value, status etc. Query the table SYSCAT.DATAPARTITIONS.

Listing 2. Get the details of partitions from partitioned table

```sql
SELECT D.DATAPARTITIONNAME, D.HIGHVALUE, D.LOWVALUE, D.TABNAME
FROM SYSCAT.DATAPARTITIONS D
WHERE D.TABNAME = 'CONTRACTS'
```

You can add new ranges and add one or more empty partitions. If you want to insert data for year 2015 in the contract table in the above example, you need to add a data partition for the year 2015. Listing 3 shows how to add empty data partitions to a partitioned table.

Listing 3. Adding empty partitions to the partitioned table

```sql
ALTER TABLE CONTRACTS
    ADD PARTITION PART11
    STARTING FROM ('2015-01-01') ENDING AT ('2015-12-31')
```

You can also attach tables that already contain data into a partitioned table. For example, contract details for year 2016 is stored in table CONTRACT_2016 and you want to merge the data for 2016 with the above table CONTRACTS. The syntax in Listing 4 will be used. The number of columns in both tables should be same.

Listing 4. Attaching data as new partition

```sql
ALTER TABLE CONTRACTS
    ATTACH PARTITION PART12 STARTING FROM ('2016-01-01') ENDING AT ('2016-12-31')
    FROM CONTRACTS_2016
```

The `attach partition` command will place the target table (CONTRACTS table in this case), into Set Integrity Pending State. To validate the integrity of the Set Integrity Pending State, execute the `SET INTEGRITY` statement with the IMMEDIATE CHECKED or IMMEDIATE UNCHECKED option on the table.

Listing 5. Set the integrity on CONTRACT table

```sql
SET INTEGRITY FOR CONTRACTS IMMEDIATE CHECKED
```

The existing partitions can be easily detached/removed from the partitioned table. For example, if the data for year 2004 is no longer relevant in the contract table above, it can be detached as shown above. The detached data is put into the TEMP_CONTRACT_2004. If data for 2004 is no longer required for any analysis, drop the table TEMP_CONTRACT_2004.
### Listing 6. Detaching data from partitioned table

```
ALTER TABLE CONTRACTS
DETACH PARTITION PART0
INTO TEMP_CONTRACT_2004
```

To know the detached table created but which are yet not accessible to you by a `DETACH PARTITION` command, use the command in Listing 7.

### Listing 7. List tables created by detach partition command

```
SELECT TABSCHEMA, TABNAME, TYPE
FROM SYSCATE.TABLE WHERE TYPE = 'L'
```

You can use the `SET INTEGRITY` command to make the table accessible.

If your requirement is to partition the table on two or more columns (e.g., contractor ID and contract date) and there is continuous flow of data into the table via ETL tool, creating a DB2 partitioned table and adding partitions is not a simple or straightforward affair, requiring continuous manual interventions. Listing 8 shows how to create a partitioned table with two columns.

### Listing 8. Creating a table with multiple columns as partition key

```
CREATE TABLE Contracts (id INT, contractor_id INT, Contract_Date timestamp )
PARTITION BY RANGE (contractor_id, Contract_date)
(PARTITION PART0 STARTING FROM (1000,'2004-01-01') ENDING AT (1000,'2004-12-31') ,
  PARTITION PART1 STARTING FROM (1000,'2005-01-01') ENDING AT (1000,'2005-12-31'),
  -----------------------------------
  PARTITION PART10 STARTING FROM (1000,'2014-01-01') ENDING AT (1000,'2014-12-31') ,
  ----------------------------------
```

Specify the value of the range — all unique combinations of contractor ID and contract date or you may end up getting frequent job failures with error SQL0327N.

**Note:** The row cannot be inserted into table table-name because it is outside the bounds of the defined data partition ranges.

To handle this problem, you need to add new partitions to the partitioned table as and when new partition key combinations come from the source table.

**Adding empty data partitions**

To explain the process, the CONTRACTS table created in Listing 8 will be used as the target table and CONTRACT_SOURCE as a source table. The contract table is partitioned on contractor IDs and contract date column.

### Table 1. CONTRACT_SOURCE table for ETL processing

<table>
<thead>
<tr>
<th>Contract_ID</th>
<th>Contractor_ID</th>
<th>Contract_Date</th>
<th>Contract_Amount</th>
</tr>
</thead>
</table>

Add and detach data partitions from DB2 partitioned tables using InfoSphere DataStage
When there is continuous flow of data into a partitioned table, the ETL should be designed in such a way that it first adds all the partitions that are not available in the target table and inserts records into the partitioned table. This is achieved by creating stored procedure that will add partition to the target table.

**Listing 9. DB2 stored procedure to add empty partitions to the table**

```sql
CREATE PROCEDURE SP_ADD_EMPTY_PARTITION
(IN VHIGHVALUE VARCHAR (20),
 IN VLOWVALUE VARCHAR (20),
 IN v1 VARCHAR (5),
 IN v2 VARCHAR (8),
 IN TABNAME VARCHAR (100))
LANGUAGE SQL
BEGIN
 DECLARE SQL_STMT VARCHAR (2000);
 SET SQL_STMT = 'ALTER TABLE ' ||TABNAME ||' ADD PARTITION P_'||v1||'_'||v2||' STARTING FROM (' || VLOWVALUE || ' )ENDING AT (' || VHIGHVALUE || ')';
 EXECUTE IMMEDIATE SQL_STMT;
END
```

Explanation of the stored procedure:

- **VHIGHVALUE** and **VLOWVALUE** passes the upper and lower limit of the partition range.
- **TABSCHEMA** passes the database schema that contains the table.
- **V1** and **V2** passes the columns that makes the partition key. In the above example, **V1** is the contractor ID, and **V2** is the year. Say **V1** = 1001 and **V2** = 2015, the name of the new data partition will be **P_1001_2015**. It is important to name the data partitions properly in the partitioned table, as it will make the detach process simpler as explain the later section.
- **TABNAME** passes the partitioned table name.

Executing the DB2 command call `SP_ADD_EMPTY_PARTITION ('1000, 12/31/2014', '1000, 01/01/2014', 1000, 2014, CONTRACTS)` will add partition data **P_1000_2014** to the target table **contract** with upper limit range of **(1000, '12/31/2014')** and low limit of **(1000, '01/01/2014')**.

The next step is to create the ETL job that will call and execute the above stored procedure before inserting that data into the target table. The ETL job will have source (a DB connector stage or flat files), which will be pass values of partition key for which no data partition exists in the table.
Figure 1. ETL job for adding empty partition into the table

The ETL job uses a DB2 connector stage and a stored procedure stage. Note that instead of DB2 connector stage, you can use ODBC or DB2 Enterprise stage or sequential file stage. To fetch the unique partition keys for which data partitions need to be added, use the SQL query in Listing 10. This query may change as per the source data and business logic.

Listing 10. SQL to fetch the new partitioned key values

```
SELECT HIGHVALUE,
       LOWVALUE,
       SUBSTR (HIGHVALUE,1, LOCATE (',',HIGHVALUE)-1) AS V1,
       SUBSTR (HIGHVALUE, LOCATE( ',',HIGHVALUE )+ 2,4 )AS V2
FROM
(

SELECT CS.CONTRACTOR_ID ||
     ','||DATE(LAST_DAY (CURRENT_DATE) +(12 – MONTH(CURRENT_DATE) )MONTHS)||'''' AS HIGHVALUE,
     CS.CONTRACTOR_ID ||
     ','||DATE(LAST_DAY(CURRENT_DATE) – MONTH(CURRENT_DATE) MONTHS + 1 DAY)|| '''' AS LOWVALUE,
     'CONTRACT' AS TABNAME
FROM CONTRACT_SOURCE CS
MINUS
SELECT HIGHVALUE, LOWVALUE, D.TABNAME
FROM SYSCAT.DATAPARTITIONS D
WHERE D.TABNAME='CONTRACT'
)
```

The above query returns all those combinations for which a partition does not exist in the table, but these data partitions are required before inserting the incoming source data into the target table. Use the above query in the DB2 connector stage, as explained in Figure 2.
Now you configure the stored procedure stage as shown in Figure 3. In the syntax tab in the properties of the stored procedure stage, enter the procedure name.

Save the job as J_Add_Partition, compile it and run to add data partitions. You must place the job in the ETL sequence such that it always runs before the jobs inserting data into the target table as shown in Figure 4.
Figure 4. ETL job sequence to add the empty data partitions before inserting the data in the partitioned table

Detaching partitions from the partitioned table

Like adding data partitions, you can detach partitions from a partitioned table by using a stored procedure in the ETL job. The stored procedure in Listing 11 gives the sample code.

Listing 11. DB2 stored procedure to detach partitions to the table

```
CREATE PROCEDURE SP_DETACH
(IN V_PART VARCHAR (50),
 IN V_TABNAME VARCHAR (50))
LANGUAGE SQL
BEGIN
 DECLARE SQL_STMT  VARCHAR(1024);
 IF
 (SELECT COUNT(1) FROM SYSCAT.DATAPARTITIONS S
  WHERE S.TABNAME =V_TABNAME AND S.DATAPARTITIONNAME=V_PART
 ) > 0) THEN
 SET  SQL_STMT = 'ALTER TABLE '||V_TABNAME||' DETACH PARTITION '|| V_PART || ' INTO TABLE ' || V_TABNAME
 ||'_'||V_PART ; EXECUTE IMMEDIATE SQL_STMT ;
 END IF;
 END
```

Explanation of the stored procedure:

- V_PART is the partition name that is no longer relevant. You can get this value from SYSCAT.DATAPARTITIONS. As mentioned, that's why it is important to name the data partitions properly. In the example CONTRACT table used above, if the data for year 2005 and contractor ID 1001 is no longer required and needs to be removed from the table, V_PART is P_1001_2014.
• V_TABNAME is the partitioned table name.

The procedure checks whether the data partition exists and if so, it is detached from the target table. If the data partition no longer exists in the partitioned table, it does nothing. The detached data is moved to a new Table_PartitionName table (above, it is CONTRACT_P_1001_2014).

Create ETL job J_Data_Partion_Detach, as shown in Figure 5. You can use DB2 connector, DB2 Enterprise Stage, or a flat file. For example, if the requirement is to remove all the data partitions older than 2006, use contractor ID 1001 in the contract table.

**Listing 12. Query to fetch data partition names older than 2006 and contractor ID 1001 for contract table**

```sql
SELECT
    D.DATAPARTITIONNAME,
    D.TABNAME
FROM  SYSCAT.DATAPARTITIONS D
WHERE D.TABNAME = 'CONTRACTS'
    AND SUBSTR(LOWVALUE , 1, LOCATE(',',LOWVALUE)-1)=1001
    AND SUBSTR(LOWVALUE ,LOCATE(',',LOWVALUE)+2,4) > 2005
```

**Figure 5. ETL job for detaching partition**

Maintain the record of all the data partitions detached from the table in a Detached file text file. Use a copy stage to copy and propagate the data into stored procedure stage and sequential file stage, as shown in Figure 6. The Detached file is used to drop all the tables created by the detached process.
When the executed ETL job detaches unwanted partitions from the partitioned table using the stored procedure `SP_DETACH` and maintains a Detach file text file.

The detach process creates tables where the data from detached partitions are stored. The tables can be dropped if data is no longer required or re-attached to the partitioned table. These tables cannot be dropped unless the asynchronous background process (ABP) on the partitioned table completes.

Use the command in Listing 13 to check if any asynchronous process for the partitioned table is running.

Listing 13. Command to check the utilities running in the db

```
"Db2 list utilities show detail"
```

You need to capture this information into a file using a script similar to that of Listing 14 and if no asynchronous process is running on the partitioned table, only then can you drop the tables.

Listing 14. Shell script to list the DB2 utilities running on the database

```
#!/Bin/bash
Check_asynchronous_part_detach()
{
  Dbname=$1
  Usr=$2
  Pswd=$3
  Schema=$4
  Db2 connect to "$dbname" user "$usr" using "$pswd" > /dev/null
  Db2 list utilities show detail > asynchronous_detach.txt
  Db2 connect reset
}
Check_asynchronous_part_detach $1 $2 $3 $4
```
Execute this script only in the DB2 server and share or transfer the file asynchronous_detach.txt to the DataStage server before running the jobs to drop the tables created in the detaching process.

To check if an asynchronous process is running, use the shell script shown in Listing 15. This shell script will be in DataStage server and it uses the input file asynchronous_detach.txt. Use the command `stage` to execute this script, as shown in Figure 7.

**Listing 15. Shell script to verify whether any asynchronous process running on the database**

```bash
#!/bin/bash
# Check if any ASYNCHRONOUS PARTITION DETACH is running
filename=$1
n=$(grep -c ASYNCHRONOUS PARTITION DETACH "$filename")
echo $n
```

If the output of the above script is 0, drop the tables storing the data from the detached data partitions using the shell script in Listing 16. The input parameters for the script are database name, user, password, schema name, and the data partition name, which is stored in the Detached file file created by the ETL job J_Data_Partition_Detach above.

**Listing 16. Script to drop the detached tables**

```bash
Drop_Temp_Partition()
{
DBNAME=$1
USER=$2
PWD=$3
SCHEMA=$4
PART=$5
db2 connect to "$DBNAME" USER "$USER" USING "$PWD"
while read temp
do
  a=`echo $temp | cut -d, -f1`
b=`echo $temp | cut -d, -f2`
c=`echo $temp | cut -d, -f3`
d=$b'_'$a
db2 drop table $SCHEMA.$d
done < $PART
db2 connect reset
}
drop_temp_partition $1 $2 $3 $4 $5
```

The shell scripts in listings 15 and 16 are executed via the DataStage job sequence, as shown in the Figure 7.
Add and detach data partitions from DB2 partitioned tables using InfoSphere DataStage

Figure 7. Dropping the detached tables from the DB

Use the execute command stage to execute the shell scripts. Check_ABP_Detach checks for any asynchronous process in running in the DB. If the script returns zero only, execute the DropTempPartition. To do this in the properties, select the Triggers tab, select Custom-(conditional) for expression type, and in the Expression field, enter Field(Check_ABP_Detach.$CommandOutput,@FM,1) = 0. Save the job sequence and place the sequence in ETL flow such that you can easily maintain the partitioned tables with little manual intervention.

Conclusion

Using an ETL tool like IBM DataStage and simple DB2 stored procedures to attach data partitions and remove data partitions from DB2 partitioned tables drastically reduces the manual interventions by the DBA and it is particularly beneficial in data warehousing environments, where there is continuous data flow.
Resources

Learn

- Read the developerWorks article "Introducing DB2 9, Part 2: Table partitioning in DB2 9."
- Refer to the IBM Knowledge Center for a better understanding of partitioned tables.
- Read the IBM Data Magazine for information about Creating and Using Partitioned Tables.
- Read the Advantages of using table partitioning.
- Stay current with developerWorks technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Attend a free developerWorks Live! briefing to get up-to-speed quickly on IBM products and tools as well as IT industry trends.
- Follow developerWorks on Twitter.
- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.

Get products and technologies

- Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, use a product in a cloud environment, or spend a few hours in the SOA Sandbox learning how to implement Service Oriented Architecture efficiently.

Discuss

- Get involved in the My developerWorks community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.
About the author

Ranjeeta Pegu

Ranjeeta Pegu has nine years of experience on data warehousing and data analytics, currently based at the Sterling Commerce Lab in Bangalore, India. She is an IBM-certified DataStage professional. Her core areas of interest are ETL designing and modeling, ETL job tuning, and DB2 query tuning. She frequently contributes to technical blogs and forums. She holds a master’s degree from the Indian Institute of Technology, Kharagpur (IIT-KGP), and a bachelor’s degree from North Eastern Regional Institute of Science and Technology, Itanagar.