# Comparing DB2 materialized query tables and Oracle materialized views

## A technical overview

Burt L. Vialpando
Certified Consulting I/T Specialist
IBM

Vikram S. Khatri
Certified Consulting I/T Specialist
EMC

16 August 2007

Familiarize yourself with IBM® DB2® materialized query tables (MQTs) in a side-by-side comparison with Oracle materialized views. See, also, how the DB2 optimizer uses MQTs.

## Introduction

In DB2, a materialized query table (MQT) is a table whose definition is based on the result of a query. As such, the MQT typically contains precomputed results based on the data existing in the table or tables that its definition is based on. If the query compiler determines that a query will run more efficiently against an MQT than the base table or tables, the query executes against the MQT and you obtain the result faster than you otherwise would. This concept is identical to the materialized view concept in Oracle.

## The DB2 materialized query table in relation to the Oracle materialized view

In DB2, the DDL syntax to create an MQT is similar to that of creating an Oracle materialized view. There are syntactical differences though. Let's look at a real-world Oracle materialized view DDL converted to DB2 and compare the sections of the body (see Table 1):

### Table 1. A DB2 materialized query table converted from an Oracle materialized view

| DB2 materialized query table (MQT) | Oracle materialized view |
|---|---|
| `CREATE (1) TABLE`<br>`(2) PROD.DAILYSUM` | `CREATE (1) MATERIALIZED VIEW`<br>`(2) PROD.DAILYSUM` |

```
      (3) AS(
SELECT
       A.ORGUNT_SID         AS ORGUNT_SID,
       C.DPTCHRT_SID        AS DPTCHRT_SID,
       C.ANCDPT_SID         AS DPT_SID,
       B.TMFRAM_SID         AS TMFRAM_SID,
       B.STRTDT             AS STRTDT,
       SUM(A.ONSALEFLG)     AS ONSALEFLG,
       COUNT(A.ONSALEFLG)   AS ONSALEFLGCNT,
       SUM(A.SALES)         AS SALES,
       COUNT(A.SALES)       AS SALESCNT,
       SUM(A.SHRINK)        AS SHRINK,
       COUNT(A.SHRINK)      AS SHRNKCNT,
       SUM(A.SHRINKQTY)     AS SHRINKQTY,
       COUNT(A.SHRINKQTY)   AS SHRINKQTYCNT,
       COUNT(*)             AS RECORDCNT,
FROM   PROD.DAYTOT A,
       PROD.CALDTL B,
       PROD.DPTCHR C
WHERE  B.CAL_SID = 100
AND    B.TMFRAM_SID <> 10
AND    A.DT BETWEEN B.STRTDT AND B.ENDDT
AND    A.DPT_SID = C.RPTDPT_SID
GROUP  BY A.ORGUNT_SID, C.DPTCHRT_SID,
       C.ANCDPT_SID, B.TMFRAM_SID,
       B.STRTDT)
(4)  [INTIALLY DEFERRED]
(5)  REFRESH DEFERRED
(6)  [MAINTAINED BY SYSTEM]
(7)  [ENABLE QUERY OPTIMIZATION]
(8)  IN TABLESPACE1 INDEXES IN TABLESPACE2 ;
```

```
      (8) TABLESPACE TABLESPACE1
      (4) BUILD IMMEDIATE
      (5) REFRESH FAST ON DEMAND
      (3) AS
SELECT
       A.ORGUNT_SID         ORGUNT_SID,
       C.DPTCHRT_SID        DPTCHRT_SID,
       C.ANCDPT_SID         DPT_SID,
       B.TMFRAM_SID         TMFRAM_SID,
       B.STRTDT             STRTDT,
       SUM(A.ONSALEFLG)     ONSALEFLG,
       COUNT(A.ONSALEFLG)   ONSALEFLGCNT,
       SUM(A.SALES)         SALES,
       COUNT(A.SALES)       SALESCNT,
       SUM(A.SHRINK)        SHRINK,
       COUNT(A.SHRINK)      SHRNKCNT,
       SUM(A.SHRINKQTY)     SHRINKQTY,
       COUNT(A.SHRINKQTY)   SHRINKQTYCNT,
       COUNT(*)             RECORDCNT
FROM   PROD.DAYTOT A,
       PROD.CALDTL B,
       PROD.DPTCHR C
WHERE  B.CAL_SID = 100
AND    B.TMFRAM_SID != 10
AND    A.DT BETWEEN B.STRTDT AND B.ENDDT
AND    A.DPT_SID = C.RPTDPT_SID
GROUP  BY A.ORGUNT_SID, C.DPTCHRT_SID,
       C.ANCDPT_SID, B.TMFRAM_SID,
       B.STRTDT ;
```

- **(1)**`CREATE TABLE`
  DB2 uses `CREATE TABLE` syntax rather than `CREATE MATERIALIZED VIEW` syntax that Oracle uses. DB2 calls MQTs "tables" rather than "views," thus it uses the `CREATE TABLE` syntax when creating these. DB2 knows this is a materialized query table rather than a regular table because, first, MQTs are always created from another table or set of tables and, second, the MQT creation DDL did *not* say at the end of it "`FOR DEFINITION ONLY`". If you want to create a regular (non-MQT) table, as defined from another table, then use the `FOR DEFINITION ONLY` keywords.
- **(2)**`TABLENAME.SCHEMANAME`
  DB2 then allows you to define the MQT using whatever schema qualifier and name you desire. As in Oracle, the `AS` precedes the entire `SELECT` clause that creates the MQT.
- **(3)**`AS SELECT`
  DB2 uses the `SELECT` clause to define the body of the MQT. Selecting subsets, full selects, and even joins of tables is permissible. This is identical in concept to Oracle. **Note:** There are many exceptions and caveats as to what can and cannot go into a DB2 MQT (just as in Oracle), although this article does not go into detail of these things. You can get clear rules of MQT creation in the DB2 documentation.
- **(4)**`DATA INTIALLY DEFERRED`
  In DB2, the default is to immediately place data into the new MQT upon its creation. If you do not want this to occur, you should use `DATA INITIALLY DEFERRED` subcommand. In Oracle, the `BUILD IMMEDIATE` is also the default, but you can say this specifically and is often used by Oracle DBAs. For the example in Table 1, data is to be built immediately upon MQT creation, so you can just leave the `DATA INTIALLY DEFERRED` option out of the MQT DDL. (The option is shown in brackets to indicate it is optional.)
- **(5)**`REFRESH [DEFERRED / IMMEDIATE]`

DB2 controls the way the MQT is refreshed with the `REFRESH [DEFERRED / IMMEDIATE]` options. The example use the `REFRESH DEFERRED` option because the source Oracle materialized view was defined with the `ON DEMAND` subcommand, which means the same thing. `REFRESH DEFERRED` in DB2 just means that you have to use a `REFRESH TABLE` statement in order to get the MQT have the latest changes to the data applied to it. (See the "`REFRESH TABLE` command" section for more information.) `REFRESH IMMEDIATE` means that DB2 changes the MQT every time the base table or tables change and does not require the use of the `REFRESH TABLE` statement.
**Note:** In Oracle, there is a `FAST` keyword, which is shown in the example. This particular feature uses special logs to accomplish a "delta" refresh, which can speed this process up. In order to get this "fast" refresh in DB2, you can use a staging table. (See the "Staging tables" section for more information.)

- **(6)**`MAINTAINED BY [SYSTEM / USER / FEDERATED TOOL]`
  DB2 MQTs can be "maintained" three different ways:
  1. `MAINTAINED BY SYSTEM` is the default, so it did not really have to be put in this output DDL example. It is shown in brackets to indicate that it is optional in this situation. This just means that only DB2 controls what is in the MQT by doing the `REFRESH` options (which are either `DEFERRED` or `IMMEDIATE`)
  2. `MAINTAINED BY USER` allows the DBA to perform direct `INSERT`, `UPDATES`, and `DELETES` to the MQT
  3. `MAINTAINED BY FEDERATED TOOL` indicates that the DB2 replication tool maintains the MQT
- **(7)**`ENABLE QUERY OPTIMIZATION`
  DB2 can now use the MQT in its query optimization steps by using `ENABLE QUERY OPTIMIZATION`. This is the default behavior and is shown in brackets to indicate that it was optional in this situation. There are some circumstances where you would `DISABLE QUERY OPTIMIZATION`, but this article does not cover that.
- **(8)**`IN [TABLE SPACE NAME] INDEXES IN [TABLE SPACE NAME]`
  Finally, DB2 places the MQT data and the MQT indexes in their designated table spaces. Oracle DBAs should note that in DB2, whether it is a base table or an MQT, you always define which table spaces the table data, indexes, and long objects (like lobs or XML) are assigned during *table creation time*.

## `REFRESH TABLE` command

For refresh deferred MQTs, you have to tell DB2 when and how you want them to be refreshed because DB2 does not do this automatically. You can specify this with the `REFRESH TABLE` command. The details of this command are shown in Listing 1, below:

## Listing 1. `REFRESH TABLE` command

```
               .-,-----------------------------------------------------------.
               V                                                             |
>>-REFRESH TABLE----table-name--| online-options |--| query-optimization-options |-+-->

>--+-----------------+-----------------------------------------><
    +-INCREMENTAL-----+
   '-NOT INCREMENTAL-'

online-options:

   .-ALLOW NO ACCESS----.
|--+--------------------+--------------------------------------|
   +-ALLOW READ ACCESS--+
   '-ALLOW WRITE ACCESS-'

query-optimization-options:

|--+-------------------------------------------------------------------------------+--|
| .-ALLOW QUERY OPTIMIZATION-.                                .-WITH REFRESH AGE ANY-. |
'-+--------------------------+--USING REFRESH DEFERRED TABLES--+----------------------+-'
```

## Keywords `INCREMENTAL` / `NON INCREMENTAL`

- `INCREMENTAL` specifies an incremental refresh for the table by considering only the delta portion (if any) of its underlying tables or the content of an associated staging table (if one exists and its contents are consistent). If such a request cannot be satisfied (that is, the system detects that the materialized query table definition needs to be fully recomputed), an error (SQLSTATE 55019) is returned
- `NON INCREMENTAL` does a full refresh on the MQT

## Keywords `ALLOW [NO ACCESS` / `READ ACCESS` / `WRITE ACCESS]`

These keywords indicate the degree to which others can use the MQT during its refresh:

- `ALLOW NO ACCESS` is the default and provides the fastest refresh of the MQT. Its description is self explanatory
- `ALLOW READ ACCESS` is the next fastest refresh of the MQT. You can read the MQT during the refresh itself, but you cannot do a user-maintained DML operation against the MQT
- `ALLOW WRITE ACCESS` is the slowest refresh. It allows complete DML against an MQT during its refresh

## How to do DBMS_MVIEW.REFRESH in DB2

The DB2 `REFRESH TABLE` statement is similar to using the Oracle package `DBMS_MVIEW`, procedure `REFRESH` as illustrated below.

A full refresh example:

- Oracle:
  ```
  EXEC DBMS_MVIEW.REFRESH('PROD.DAILYSUM','c');
  ```
- DB2:
  ```
  REFRESH TABLE PROD.DAILYSUM NON INCREMENTAL;
  ```

An incremental refresh example:

- Oracle:
```
EXEC DBMS_MVIEW.REFRESH('PROD.DAILYSUM');
```
- DB2:
```
REFRESH TABLE PROD.DAILYSUM INCREMENTAL;
```

### CURRENT REFRESH AGE register for DEFERRED MQTs

The CURRENT REFRESH AGE register specifies a timestamp duration value with a data type of DECIMAL(20,6). It is the maximum duration since a particular timestamped event occurred to a cached data object (for example, a REFRESH TABLE statement processed on a system-maintained refresh deferred materialized query table), such that the cached data object can be used to optimize the processing of a query. If CURRENT REFRESH AGE has a value of 99 999 999 999 999, and the query optimization class is 5 or more, the types of tables specified in CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION are considered when optimizing the processing of a dynamic SQL query.

**SET CURRENT REFRESH AGE**

When you use DB2 command db2 SET CURRENT REFRESH AGE ANY, it actually sets the refresh age register with a value 99 999 999 999 999.

The value of CURRENT REFRESH AGE must be 0 or 99 999 999 999 999. The initial value is 0. You can invoke the SET CURRENT REFRESH AGE statement to change the value.

### Staging tables

When you define the MQT, you can define a second table that will be the staging table for that MQT. A staging table allows incremental maintenance support for the deferred MQT. The staging table collects changes that need to be applied to the MQT to synchronize it with the contents of underlying tables. Using staging tables eliminates the high lock contention caused by immediate maintenance content when an immediate refresh of the MQT is requested. Also, the MQTs no longer need to be entirely regenerated whenever a REFRESH TABLE is performed.

## How MQTs are used by the DB2 optimizer

The Oracle DBA already knows how materialized views work, so let's see how an MQT works in DB2. From a DB2 "beginners" perspective with little experience in using DB2 and its tools, let's look at the Control Center, the Command Editor, as well as the text and visual explain utilities.

### Setting up the MQT example

1. First, launch the DB2 Control Center. Select **Start** > **All Programs** > **IBM DB2 General Administration Tools** > **Control Center**.
   You can also launch the DB2 Control Center using a Windows command prompt. Enter db2cc.
2. Right-click on the **SAMPLE database** > **Query**. This opens the Command Editor and connects to the SAMPLE database.
3. Copy and paste in the Command Editor's command window the SQL from Listing 2:
   **Listing 2. Creating MQT example base tables**
```
--#SET TERMINATOR !
```

```
DROP TABLE SHIPPING.BILL !
DROP TABLE SHIPPING.CHARGE_BILL !

CREATE TABLE "SHIPPING"."BILL"
("BILL_GK" INTEGER NOT NULL ,
 "TD_IND"   DOUBLE ,
 "CREATION_TS" TIMESTAMP NOT NULL  ) !

CREATE UNIQUE INDEX "SHIPPING"."IX_BILL"
ON "SHIPPING"."BILL"
("BILL_GK" ASC)
ALLOW REVERSE SCANS !

CREATE TABLE "SHIPPING"."CHARGE_BILL"
(  "BILL_GK"  INTEGER NOT NULL ,
   "BILL_CHARGE_IND"  INTEGER NOT NULL ,
   "RATE_PRICE"  DOUBLE ) !

ALTER TABLE "SHIPPING"."CHARGE_BILL"
ADD CONSTRAINT "PK_CHARGEBILL" PRIMARY KEY
("BILL_GK", "BILL_CHARGE_IND") !
```

This creates two tables (with appropriate indexing) for you to work out the MQT example. Note that the command `--#SET TERMINATOR` can change what the SQL delimiter is in the SQL scripting, in this case an exclamation point (!). This is necessary to do when you are creating stored procedures or UDFs in DB2. You can also set this in the Command Editor at the bottom where it says “Statement termination character”, as shown in Figure 1:

### Figure 1. Statement terminator character


4. Make sure your cursor is at the very beginning of the SQL in the Command Editor’s command window and click on the green arrow to run it. It should say "The SQL command completed successfully" for each statement.:

### Figure 2. `Run` command

## Create random data in tables

Notice the `INSERT` statement in this example that inserts random data in each table. If you look at it carefully, it is a single `INSERT` statement, inserting 25,000 rows using a temporary table using a `WITH` construct. You can use this type of recursive SQL in DB2 to create sample data.

5. Copy and paste the SQL from Listing 3, which inserts data into the tables and collects statistics on them. **Notes:**
   - To clear the previous command, right-click anywhere in the Command Editor, then select **Select all** Use your delete key to clear everything selected.
   - To clear the results pane, right-click anywhere in the Command Editor and select **Clear results**

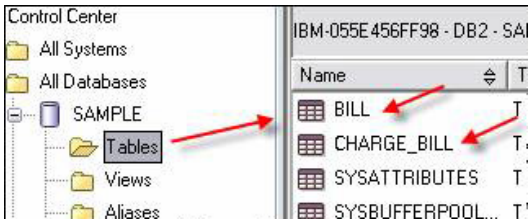
Run this SQL.

### Listing 3. Inserting data into the MQT example base tables

```
--#SET TERMINATOR !

INSERT INTO SHIPPING.CHARGE_BILL
(BILL_GK, BILL_CHARGE_IND, RATE_PRICE)
WITH TEMP1 (R1,R2,R3) AS
( VALUES (0, RAND(2), RAND()+(RAND()/1E5) )
UNION ALL
SELECT R1+1, RAND(2), RAND()+(RAND()/1E5)
FROM TEMP1
WHERE R1 < 25000
)
SELECT R1, R2*23+1, DECIMAL(R3*99999,7,2)
FROM TEMP1 !

INSERT INTO SHIPPING.BILL (BILL_GK, TD_IND, CREATION_TS)
WITH TEMP1 (R1,R2,R3) AS
( VALUES (0,RAND(), CURRENT TIMESTAMP )
UNION ALL
SELECT R1 +1,RAND(),CURRENT TIMESTAMP
FROM TEMP1
WHERE R1 < 25000
)
SELECT R1, R2*23+1, CURRENT TIMESTAMP
FROM TEMP1 !

RUNSTATS ON TABLE SHIPPING.BILL  ON ALL COLUMNS WITH DISTRIBUTION
ON ALL COLUMNS AND DETAILED INDEXES ALL !

RUNSTATS ON TABLE SHIPPING.CHARGE_BILL  ON ALL COLUMNS WITH DISTRIBUTION
ON ALL COLUMNS AND DETAILED INDEXES ALL !
```

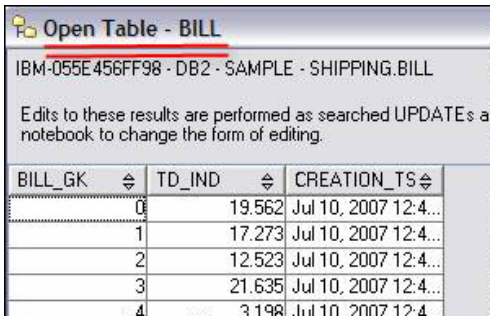
6. From the Control Center, you can check out what these tables and the data looks like. Click on the **Tables** directory, then on the column **schema** to sort by schema. You should be able to find the two new tables, BILL and CHARGE_BILL, under schema SHIPPING.

### Figure 3. How tables look



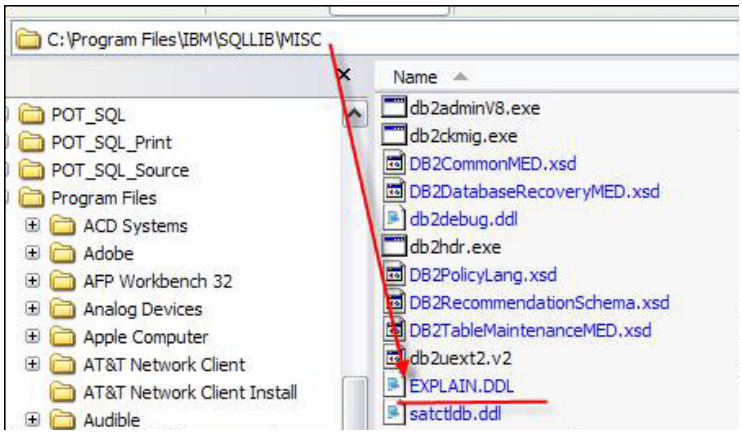

7. Double-click on either table to see the data in it.

### Figure 4. Open table "Bill"




8. Next, you need to create the explain tables in your schema in order to work with the DB2 explain facilities. Find the file in this path: C:\Program Files\IBM\SQLLIB\MISC\EXPLAIN.DDL.

### Figure 5. Create explain tables




9. Copy the contents of the file EXPLAIN.DDL into the Command Editor and run it. The explain tables are now in your user's schema and ready for use.
10. Copy and paste the SQL from Listing 4, and run it:

### Listing 4. Create a stored procedure to test your tables

```
--#SET TERMINATOR !

CALL SYSPROC.SET_ROUTINE_OPTS('EXPLAIN YES EXPLSNAP YES') !

DROP PROCEDURE SHIPPING.GET_CHARGE_COUNT !

CREATE PROCEDURE SHIPPING.GET_CHARGE_COUNT(OUT v_count INTEGER)
LANGUAGE SQL
BEGIN
  SET v_COUNT = (SELECT COUNT(*)
                 FROM (
                       SELECT DISTINCT
                              B.RATE_PRICE
                       FROM   SHIPPING.CHARGE_BILL B,
                              SHIPPING.BILL        C
                       WHERE  B.BILL_GK = C.BILL_GK
                      ) AS TEMP
    );
END !
```

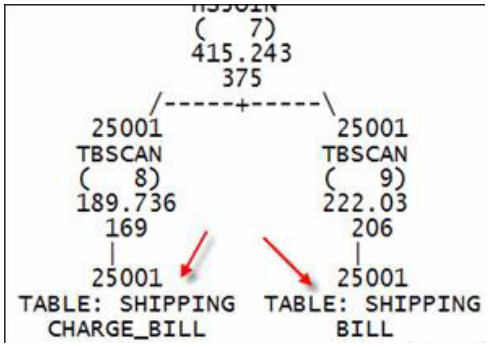
11. Notice that this stored procedure is counting all distinct values for `RATE_PRICE` that are in common between the two tables. Test the stored procedure with the following SQL command:
```
CALL SHIPPING.GET_CHARGE_COUNT(?)
```
12. So, the stored procedure SHIPPING.GET_CHARGE_COUNT is now using your tables and has an access plan stored in DB2 in what is called a DB2 "package". There are a couple of ways to find out how this access plan looks in DB2 using the various explain facilities. The first explain facility is a line command explain tool called `db2exfmt`, and the second explain is a visual explain.

**Line command explain:**

13. Enter the following DB2 utility command in a Windows command prompt anywhere on your C: drive:
```
db2exfmt -D SAMPLE -O MQT_EXFMT_BEFORE.TXT -1
```

    **Note:** This is *not* a Command Editor command, but rather is a DB2 utility executed from the OS.
14. This DB2 explain utility command produces a Windows text file called beforemqt_exfmt.txt. You can review the explain output from this file to see how DB2 is accessing the data in the query. Figure 6 includes an excerpt from this file's output, illustrating that the stored procedure is using the two individual tables from the query in that stored procedure code.
    Take note of the very top number in the stored procedure, which is the total timerons, or DB2 cost, for that query. You will reduce it later using an MQT.

    ### Figure 6. Text explain plan output

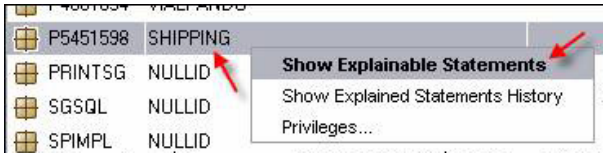



**Visual explain:**

15. Now let's review a visual explain method of seeing the same information as you saw in the text explain output. From the DB2 Control Center, find and expand the directory tree **Application Objects**. Then find and expand **Packages**.
16. You now see all the packages in DB2 that have bound access plans to the SAMPLE database. You want to find the one for your stored procedure you just created. In this case, since it is the only one that is created by the owner "SHIPPING," you can easily find it. So go to the only package that is in the "SHIPPING" creator's schema, as shown in Figure 7, below.
    **Note:** The package name is generated by DB2 when the stored procedure was created, so it will most likely not match the one you have in your database.

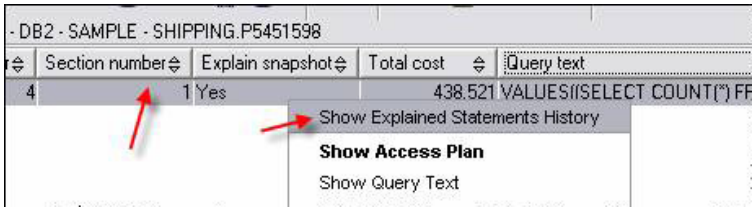
17. Right-click on this package, and select **Show Explainable Statements**.

**Figure 7. Show explain plan**




18. There is only one explainable statement in this package. Right-click on it, and select **Show Explained Statement History**:

**Figure 8. Show explain statements history**

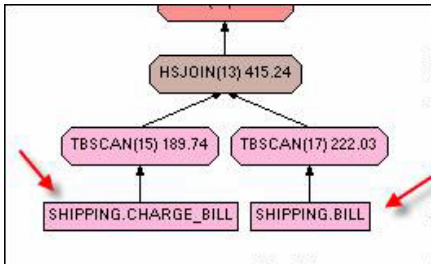



19. There is only one version of this statement in the package. Right-click on it, and select **Show Access Plan**:

**Figure 9. Show access plan**


20. This gives you a visual explain output of the SQL statement in the package for this stored procedure. Notice it tells you the same as the text output tells you, which is that the base tables are being used for the stored procedure's SQL statement.

**Figure 10. Explain plan graph**




## How MQTs work: Bringing the example home

So, what does all of this have to do with MQTs? Let's take a look at that now.

21. Copy and paste the SQL from Listing 5 into your Command Editor window, and run it. This creates an MQT for your two tables, previously shown in the example above.

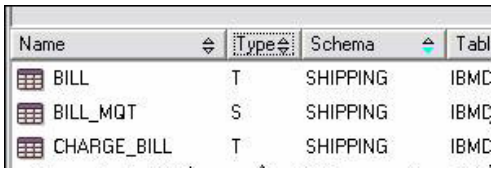
Note the `FROM` clause in the MQT DDL. Earlier in this article, you learned what each part of this DDL means. So now you will see it in action.

### Listing 5. Creating the MQT

```
--#SET TERMINATOR !

DROP TABLE SHIPPING.BILL_MQT !

CREATE TABLE SHIPPING.BILL_MQT AS
(
  SELECT B.RATE_PRICE,
         COUNT(*) CNT
  FROM SHIPPING.CHARGE_BILL B,
       SHIPPING.BILL        C
  WHERE B.BILL_GK = C.BILL_GK
  GROUP BY B.RATE_PRICE
)
DATA INITIALLY DEFERRED REFRESH IMMEDIATE
ENABLE QUERY OPTIMIZATION
MAINTAINED BY SYSTEM !

REFRESH TABLE SHIPPING.BILL_MQT !

RUNSTATS ON TABLE SHIPPING.BILL_MQT ON ALL COLUMNS WITH DISTRIBUTION
ON ALL COLUMNS AND DETAILED INDEXES ALL !
```

22. You now have three tables: two base tables and one MQT for a summary join of both of them. Figure 11 illustrates how these three tables look in the Control Center:

### Figure 11. MQT




Note that the MQT has a DB2 type of "S," which means it is a summary table. You can double-click on the MQT table to view the data if you like.

23. The DB2 optimizer will now use this MQT any time it knows that it can, rather than use the base tables automatically. Your query does not have to change to use the MQT.
To prove this, let's simply rebind the package for the stored procedure that you already created. You won't change the SQL in that package, just tell DB2 to give the access path another look now that you have an MQT based on your two tables.
Execute the following command from the Command Editor to first clear out the explain tables:

```
--#SET TERMINATOR !

DELETE FROM EXPLAIN_INSTANCE !
COMMIT !
```

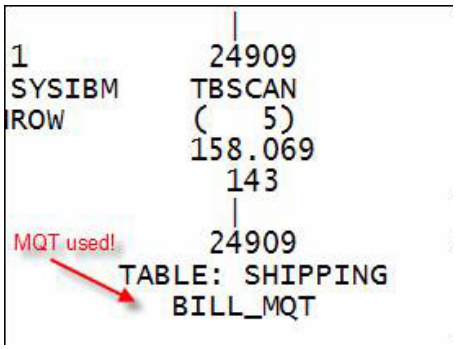
**Note:** This is an optional step. If you do not do it, DB2 keeps both versions of your access path in your package, but will uses the latest one from the last rebind.

24. Now, let's rebind the package for the stored procedure. Issue the following command from the Command Editor:

```
CALL REBIND_ROUTINE_PACKAGE('P','SHIPPING.GET_CHARGE_COUNT','ANY')
```

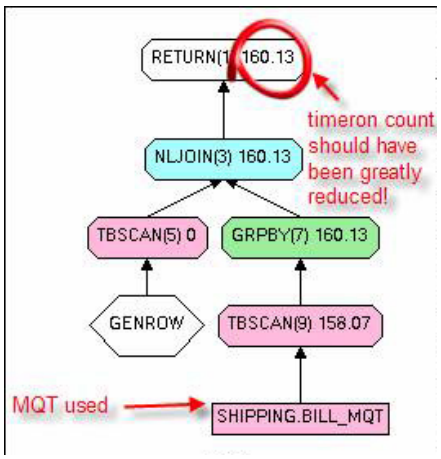
25. If you got a 0 return code from this last command, the rebind is successful. DB2 has now changed your SQL in the package to utilize the MQT.
26. So, how do you check this all out? You can run the following text explain utility from a Windows command prompt (*not* the Command Editor) and produce another text output file with your new access path:
```
db2exfmt -D SAMPLE -O MQT_EXFMT_AFTER.TXT -1
```
27. Figure 12 depicts an excerpt from the output. Notice the MQT is now being used by the package used by the stored procedure:

**Figure 12. MQT used by optimizer**




28. You can alternatively use the visual explain method shown previously. (Remember, it is a package in the Application Objects tree in the Control Center.) Figure 13 shows how the output now looks:

**Figure 13. Show explain plan**




Note the timeron count again. It should be greatly reduced because DB2 is now utilizing the MQT and not the individual tables. This is, of course, just a small example of what savings you can get in a real world.

## Conclusion

The DB2 materialized query table is exactly the same in concept as the Oracle materialized view. In this article, you have learned what the practical differences are between the two and how MQTs work in DB2 so that you can build your new DB2 database with confidence in using DB2 MQTs.

To learn more about materialized query tables, or for more detailed information about any of the topics covered in this article, see the DB2 Information Center.

# Downloads

| Description | Name | Size |
| --- | --- | --- |
| Sample DB2 scripts to create MQT | DB2MQT.zip | 10KB |

# Resources

## Learn

- "Use materialized query tables to speed up queries in DB2" (developerWorks, August 2002): Learn how to use MQTs to speed up queries.
- "An introduction to materialized query tables" (developerWorks, September 2005): Gain an understanding of MQTs, summary tables, and staging tables. By way of working examples, see how to get up and running with materialized query tables
- "Maximize the performance of WebSphere Information Integrator with materialized query tables" (developerWorks, May 2006): Learn about how the use of MQTs can help improve performance of federated system.
- "Improve DB2 query performance in a business intelligence environment" (developerWorks, May 2006): Improve performance of queries in a business intelligence environment. Walk through various methods, step-by-step, then experiment on your own system.
- DB2 Information Center: Find the information that you need to use the DB2 family of products and features, as well as related WebSphere Information Integration products and features.
- db2ude: Vikram Khatri's blog: Stay tuned to learn about tips and techniques with DB2 9.
- developerWorks Information Management zone: Learn more about Information Management. Find technical documentation, how-to articles, education, downloads, product information, and more.
- Stay current with developerWorks technical events and webcasts.
- Technology bookstore: Browse for books on these and other technical topics.

## Get products and technologies

- DB2 9 Enterprise Edition: Download a trial version.
- DB2 9 Express-C: Download a free license.
- Build your next development project with IBM trial software, available for download directly from developerWorks.

## Discuss

- Participate in the discussion forum for this content.
- Participate in developerWorks blogs and get involved in the developerWorks community.

## About the authors

**Burt L. Vialpando**

Burt Vialpando has been an IBM employee since 1998, with professional database experience since 1984. Burt is a Senior Certified IT specialist currently working for the DB2 migration team performing pre-sales Oracle to DB2 migration support. He holds numerous DB2, Oracle, and other certifications and currently serves on the IT Certification Board.

---

**Vikram S. Khatri**

Vikram Khatri works for IBM in the Sales and Distribution Division and is a member of DB2 Migration team. Vikram has 21 years of IT experience and specializes in migrating non-DB2 databases to DB2. Vikram supports the DB2 technical sales organization by assisting with complex database migration projects as well as with database performance benchmark testing.