The optimizer of IBM® DB2® for Linux®, UNIX®, and Windows® relies on a cost-based design. Using an SQL statement input by a user or application, the optimizer selects an execution plan with the lowest cost to achieve the best performance. If the performance still doesn't meet requirements after you have used all the best practices for tuning the SQL statement, you can use an optimization profile to guide DB2 to create the execution plan that the user expects. This article explains how to use optimization profiles within DB2 stored procedures.

Introduction

Most major relational database management systems, including DB2 for Linux, UNIX, and Windows, rely on a cost-based optimizer design. The optimizer estimates the cost based on several conditions, including external conditions such as CPU parallelism and speed, I/O storage characteristics, and communication bandwidth, and internal conditions such as DB2 registry variables, DB2 optimization level, statistics information, and so on. Moreover, many of these conditions are changing all the time during the runtime of the system, so the process of picking up the best execution plan is a very complex procedure for any database system. The DB2 optimization profile is an important complement to the optimizer. For SQL statements that don't meet performance requirements after you have implemented other tuning practices, you can use this function to modify the default execution plan. It if most useful for application debug and SQL statements performance optimization.

Stored procedures are another useful function of database systems. Using stored procedures, the database can implement functions that exist in the advanced programming language, such as variable definition, condition statement, control statement and so on. Stored procedures in DB2 are written in DB2 SQL Procedure Language (SQL PL). SQL PL is a subset of the SQL Persistent Storage Module language standard. This standard combines the convenience of accessing data by SQL and the stream control function of a programming language.

The SQL statements in stored procedures often have more complex logic and are very hard to debug or tune, and in some cases, the access plan selected by DB2 is not the one you want to
use, especially in an application debug situation. If, after using all the best practices, you are still not able to get the expected performance from the stored procedure, you can use an optimization profile to assist. The SQL statements in stored procedure can be a little different than SQL statements elsewhere in that they often includes some input (output) variables. When you want to use an optimization profile with these kind of statements, you need to take some additional steps to achieve the expected results. In this article, you will learn how to use an optimization profile to modify the execution plan of SQL statements in stored procedure with an example.

Introducing optimization profiles

An optimization profile is an XML document that contains optimization guidelines for one or more data manipulation language (DML) statements. An optimization profile can contain global guidelines, which apply to all DML statements that are executed while the profile is in effect, and it can contain specific guidelines that apply to individual DML statements in a package. Optimization profiles can be used to modify the access plan of SQL statements, but that doesn't mean that you can specify an access plan arbitrarily for a statement.

Special attention should be given to the following two issues.

1. The optimization class overrides the optimization profiles. That is, the optimizer will use the guidelines only if they follow the rules of the current optimization class. For example, you cannot use merge join or hash join at optimization level 0.
2. An optimization guideline will be picked up only if it is one of the plans evaluated by the optimizer. If for some reason the optimizer did not evaluate the plan specified in the optimization profile, it will not use it.

The basic format of the optimization profile is showed in Listing 1.

Listing 1. Example of optimization profile file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.1.0.0">
  <!-- Global optimization guidelines section. Optional but at most one. -->
  <OPTGUIDELINES>
    Here is for the global guidelines
  </OPTGUIDELINES>

  <!-- Statement profile section. Zero or more. -->
  <STMTPROFILE ID="profile id">
    <STMTKEY>
      Here is for the statement that we want to apply this statement level optimization guidelines to
    </STMTKEY>
    <OPTGUIDELINES>
      Here is for the optimization guidelines for the statement defined in the <STMTKEY> element
    </OPTGUIDELINES>
  </STMTPROFILE>
</OPTPROFILE>
```
OPTPROFILE

An optimization profile begins with the OPTPROFILE element. This element consists of an attribute called VERSION which specifies the edition of the XML schema that this file keeps to. One optimization profile must include and can only include one OPTPROFILE element.

Global optimization guidelines

An optimization profile can have at most one global optimization guidelines section. While the optimization profile is in effect, it will apply to all the statements that are being executed. Global optimization guidelines are defined in the OPTGUIDELINES element. For example, you can specify which MQT to use, the optimization level, the concurrent degree of the query and so on.

Statement profile section

An optimization profile can have zero or more statement profile sections. While the optimization profile is in effect, it will only apply to the exactly matched SQL statement. Statement optimization guidelines are defined within STMTPROFILE element. It consists of one STMTKEY element and one OPTGUIDELINES element.

The STMTKEY element defines the SQL statement that you will apply the statement optimization guidelines to. DB2 uses the content defined within the STMTKEY element to match the SQL statement. If matched successfully, the optimization guidelines related to this STMTKEY will be applied to this SQL statement. The statement defined within STMTKEY element must exactly match the statement whose access plan is to be affected, and this match is case sensitive. It allows redundant whitespace and control characters, such as line feeds character. However, it does not allow the use of a wildcard character to match statement group. Each statement that needs to be affected should only have one separate STMTPROFILE section. If there are more than one STMTPROFILE matching the executing statement, we will only choose and apply the first one.

Within the OPTGUIDELINES element, you can specify the access method of DB2 to some table (table scan or index scan), modify the sequence of join operation and join method to use, specify the rules of query re-write, and so on. When the executing SQL statement exactly matches the content in STMTKEY element, all the optimization guidelines in the related OPTGUIDELINES element will be applied to the creation of the access plan for this SQL statement.

Modifying the access plan of SQL statements in a stored procedure

SQL statements that are used inside a stored procedure can be special in form. For example, they often contain some input (output) variables. This kind of SQL statement can't be used directly by the STMTKEY element. When DB2 compiles these statements, it will replace these variables with the internal form, and then use this internal form as the final version. If you use the original SQL statement as the STMTKEY, while DB2 uses the internal form to do the match, it will not be successful. So, you need to find out the internal form of these SQL statements at first and then use them as the STMTKEY to create the optimization profile. The following sections show the required steps.
Test environment

The following environment is used for the testing and examples in this article.

- Operating system: AIX 6.1
- DB2 for Linux, UNIX, and Windows Version 9.7
- Database: SAMPLE database

All of the examples in this article are based on DB2 V9.7 running on AIX 6.1 operating system. For other operating systems, the implementation should be the same.

Preparing the test database

As shown in Figure 1, the SAMPLE database is a small database that is provided by DB2. If you didn't install it during setup, you can go to the sqlib/bin directory of current instance and run the db2samp1 command to create it automatically.

Figure 1. Create test database

```
bash-3.2$ db2samp1
Creating database "SAMPLE"...
Connecting to database "SAMPLE"...
Creating tables and data in schema "DB2INST1"...
Creating tables with XML columns and XML data in schema "DB2INST1"...
'db2samp1' processing complete.
bash-3.2$
```

Creating the explain tables

You need to use the DB2 Explain tool to see if the optimization guidelines are chosen, and if the access plans of the SQL statements have been modified. The output of the Explain tool will show the name of the optimization profile and the valid guidelines that are used. Therefore, you need to create the tables that the Explain tool needs. By default, DB2 will not create these tables.

As shown in Listing 2, from the sqlib/misc directory of the current instance owner, execute the file EXPLAIN.DDL to finish the creation work of all the Explain tables.

Listing 2. Script to create Explain table

```
db2 connect sample
db2 -tvf EXPLAIN.DDL
db2 connect reset
```

After running the previous command, reconnect to the database and run db2 list tables. You can see all the newly created tables in the system catalog with prefix EXPLAIN, as shown in Figure 2. All these tables are used by Explain tool to store information.
Figure 2. Newly created Explain tables

<table>
<thead>
<tr>
<th>Table/View</th>
<th>Schema</th>
<th>Type</th>
<th>Creation time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-20:53.45.070587</td>
</tr>
<tr>
<td>ADEFUSR</td>
<td>DB2INST1</td>
<td>S</td>
<td>2010-07-03-20:53.48.416843</td>
</tr>
<tr>
<td>ADVISE_INDEX</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.34.522535</td>
</tr>
<tr>
<td>ADVISE_INDEX</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.34.406936</td>
</tr>
<tr>
<td>ADVISE_MGT</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.34.718701</td>
</tr>
<tr>
<td>ADVISE_PARTITION</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.34.914101</td>
</tr>
<tr>
<td>ADVISE_TABLE</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.34.927471</td>
</tr>
<tr>
<td>ADVISE_WORKLOAD</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.34.627794</td>
</tr>
<tr>
<td>CATLOG</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-20:53.50.564595</td>
</tr>
<tr>
<td>CL_SCHEND</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-20:53.43.551144</td>
</tr>
<tr>
<td>CUSTOMER</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-20:53.50.578325</td>
</tr>
<tr>
<td>DEPARTMENT</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-20:53.43.907462</td>
</tr>
<tr>
<td>DEPT</td>
<td>DB2INST1</td>
<td>A</td>
<td>2010-07-03-20:53.44.217081</td>
</tr>
<tr>
<td>EMP</td>
<td>DB2INST1</td>
<td>A</td>
<td>2010-07-03-20:53.44.381851</td>
</tr>
<tr>
<td>EMPACT</td>
<td>DB2INST1</td>
<td>A</td>
<td>2010-07-03-20:53.45.068155</td>
</tr>
<tr>
<td>EMPLOYEE</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-20:53.44.237914</td>
</tr>
<tr>
<td>EMPMDC</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-20:53.48.842604</td>
</tr>
<tr>
<td>EMPPROJECT</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-20:53.45.017207</td>
</tr>
<tr>
<td>EMP_ACT</td>
<td>DB2INST1</td>
<td>A</td>
<td>2010-07-03-20:53.45.062549</td>
</tr>
<tr>
<td>EMP_PHOTO</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-20:53.44.362547</td>
</tr>
<tr>
<td>EMP_RESUME</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-20:53.44.607792</td>
</tr>
<tr>
<td>EXPLAIN_ACTUALS</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.35.383064</td>
</tr>
<tr>
<td>EXPLAIN_ARGUMENT</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.03.748301</td>
</tr>
<tr>
<td>EXPLAIN_DIAGNOSTIC</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.34.229490</td>
</tr>
<tr>
<td>EXPLAIN_DIAGNOSTIC_DATA</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.34.312593</td>
</tr>
<tr>
<td>EXPLAIN_INSTANCE</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.32.378111</td>
</tr>
<tr>
<td>EXPLAIN_OBJECT</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.33.859136</td>
</tr>
<tr>
<td>EXPLAIN_OPERATOR</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.33.910971</td>
</tr>
<tr>
<td>EXPLAIN_PREDICATE</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.34.006038</td>
</tr>
<tr>
<td>EXPLAIN_STATEMENT</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.33.435636</td>
</tr>
<tr>
<td>EXPLAIN_STREAM</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-21:00.34.128567</td>
</tr>
<tr>
<td>INVENTORY</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-20:53.50.470966</td>
</tr>
<tr>
<td>IN_TRAY</td>
<td>DB2INST1</td>
<td>T</td>
<td>2010-07-03-20:53.45.162281</td>
</tr>
</tbody>
</table>

Create SYSTOOLS.OPT_PROFILE table

As shown in Listing 3, all of the optimization profiles that you have defined will be stored in the OPT_PROFILE table under schema SYSTOOLS. By default, DB2 will not create this table. If you want to use the optimization profile to modify the access plan of SQL statement, you need to create it yourself.

Listing 3. SQL to create SYSTOOLS.OPT_PROFILE table

```sql
CREATE TABLE SYSTOOLS.OPT_PROFILE (
    SCHEMA VARCHAR(128) NOT NULL,
    NAME VARCHAR(128) NOT NULL,
    PROFILE BLOB (2M) NOT NULL,
    PRIMARY KEY ( SCHEMA, NAME )
);
```

The following three columns are in this table.

- Column SCHEMA refers to the schema name of the optimization profile.
- Column NAME refers to the name of the optimization profile.
- Column PROFILE stores the content of the optimization profile.

SCHEMA.NAME can be used to uniquely identify one optimization profile in the database. As shown in Listing 4, save the previous script into file SYSTOOLS.OPT_PROFILE.DDL and reconnect to the database.
Listing 4. Script to create SYSTOOLS.OPT_PROFILE table

db2 connect to sample
db2 -tvf SYSTOOLS.OPT_PROFILE.DDL
db2 connect reset

After running the script, you will finish the creation work of this table as shown in Figure 3.

Figure 3. Creating SYSTOOLS.OPT_PROFILE table

Creating the stored procedure

For the detailed syntax of the stored procedure, you can refer to the stored procedure section in the IBM DB2 9.7 Information Center, see the Related topics section. As shown in Listing 5, you will create a simple stored procedure with the name GET_EMP_NUM to finish all of your examples. This stored procedure uses DEPARTMENT table and EMPLOYEE table in the sample database. It has two parameters, one input parameter called DEPT_NO (the department ID), and one output parameter called EMP_NUM (the number of employees in this department). The function of this stored procedure is to calculate the number for employees in a department based on the ID of the department that the user inputs.
Listing 5. Script to define the stored procedure

```sql
CONNECT TO SAMPLE%

CALL SYSPROC.SET_ROUTINE_OPTS('EXPLAIN ALL')%

CREATE PROCEDURE GET_EMP_NUM(
    IN DEPT_NO CHAR(50),
    OUT EMP_NUM INTEGER)
LANGUAGE SQL
BEGIN
    SELECT COUNT(*) INTO EMP_NUM
    FROM DEPARTMENT, EMPLOYEE
    WHERE DEPARTMENT.DEPTNO = EMPLOYEE.WORKDEPT
    AND DEPARTMENT.DEPTNAME = DEPT_NO;
END%
CONNECT RESET%
```

Call the SYSPROC.SET_ROUTINE_OPTS stored procedure to set the pre-compile and binding options before the definition of GET_EMP_NUM stored procedure. You can also implement the same function by modifying the DB2_SQLROUTINE_PREPOPTS registry variable. If you call the SYSPROC.SET_ROUTINE_OPTS stored procedure, it will override the value of this registry variable. Here, you pass the parameter EXPLAIN ALL to the SYSPROC.SET_ROUTINE_OPTS stored procedure. It means that during the creation of stored procedure, the access plans of all the SQL statements in it will be saved in Explain tables.

Save the previous script into file create_procedure.ddl as shown in Listing 6, and then run the commands shown in Figure 4 to finish the creation work of the stored procedure. Note that you should execute RUNSTATS on these two tables before creating the stored procedure so that DB2 will use the latest statistics information to generate more efficient access plans.

Listing 6. Script to create stored procedure

```sql
db2 connect to sample
db2 'runstats on table db2inst1.department and indexes all'
db2 'runstats on table db2inst1.employee and indexes all'
db2 connect to reset
db2 -td% -vf create_procedure.ddl
```
Figure 4. Create `GET_EMP_NUM` stored procedure

```bash
bash-3.2S db2 -td4 -vf create_procedure.dli
CONNECT TO SAMPLE

  Database Connection Information
Database server = DB2/AIX64 9.7.1
SQL authorization ID = DB2INST1
Local database alias = SAMPLE

CALL SYSPROC.SET_ROUTINE_OPTS('EXPLAIN ALL')
  Return Status = 0
CREATE PROCEDURE GET_EMP_NUM(IN DEPT_NO CHAR(50), OUT EMP_NUM INTEGER)
LANGUAGE SQL
BEGIN
  SELECT COUNT(*) INTO EMP_NUM FROM DEPARTMENT, EMPLOYEE WHERE DEPARTMENT.DEPTNO = EMPLOYEE.WORKDEPT AND DEPARTMENT.DEPTNAME = DEPT_NAME;
END
DB20000I The SQL command completed successfully.
CONNECT RESET
DB20000I The SQL command completed successfully.
```

View the internal form of SQL statements defined in stored procedure

For these SQL statements defined in stored procedure, DB2 will replace the input (output) variable with its internal form. You can use the SQL statement shown in Listing 7 to view the internal form of these statements.

**Listing 7. SQL to view the internal form of SQL statements defined in stored procedure**

```sql
SELECT PKGNAME, S.TEXT
FROM SYSCAT.STATEMENTS AS S,
     SYSCAT.ROUTINEDEP AS D,
     SYSCAT.ROUTINES AS R
WHERE PKGSCHEMA = BSCHEMA
  AND PKGNAME = BNAME
  AND BTYPE = 'K'
  AND R.SPECIFICNAME = D.SPECIFICNAME
  AND R.ROUTINESCHEMA = D.ROUTINESCHEMA
  AND R.ROUTINENAME = 'GET_EMP_NUM'
  AND R.ROUTINESCHEMA = 'DB2INST1'
ORDER BY STMTNO;
```

You should replace the value of `R.ROUTINESCHEMA` with the schema name of the stored procedure that you use (in this example it is `DB2INST1`), and replace the value of `R.ROUTINENAME` with the name of the stored procedure (in this example it is `GET_EMP_NUM`). After that, save the script into file `get_routine_sqls.sql`, and connect to the database to execute the script, as shown in Listing 8.
Listing 8. Script to view the internal form of SQL statements defined in stored procedure

```sql
db2 connect to sample
db2 -tvf get_routine_sqls.sql
db2 connect reset
```

The output of the SQL above includes two columns. One is the package name related to the stored procedure, and the other is the internal form of the SQL statement(s). After running the script, the output is shown in Figure 5.

**Figure 5. Query the internal form of SQL statements defined in stored procedure**

```
P1513856
SELECT COUNT(*) INTO :HV00009
:HI00009 FROM DEPARTMENT, EMPLOYEE WHERE DEPARTMENT.DEPTNO = EMPLOYEE.WORKDEPT
AND DEPARTMENT.DEPTNAME = :HV00008 :HI00008
```

You can see that the output parameter EMP_NUM has been replaced by :HV00009 :HI00009, and the input parameter DEPT_NO has been replaced by :HV00008 :HI00008. It is quite different from the SQL statement originally defined in the stored procedure. As this new form is the one that DB2 will use during the compile time and execute time, this is the one that you will use in the STMTKEY element. Besides, the first column of the result is the package name related to this stored procedure (in this example it is **P1513856**). You need to use this information in a later query to retrieve the access plan of this stored procedure.

**View the default access plan**

To verify that the optimization profile will modify the default access plan of DB2, you need to first get the current access plan of SQL statement. Using the db2exfmt tool, the –n option relates to the package name. For the detailed information of each option of this tool, you can refer to the related section in the IBM DB2 9.7 Information Center, see the Related topics section. As shown in Listing 9 and Figure 6, you save the original access plan in file orig_plan.out.

**Listing 9. Script to view the original access plan of SQL statement**

```bash
db2exfmt -d sample -e db2inst1 -g -l -n 'P1513856' -s db2inst1
-o orig_plan.out -w -l -# @ -v %
```
Figure 6. Original access plan

Original Statement:
-------------------
SELECT COUNT(*) INTO :HV00009 :H100009
FROM DEPARTMENT, EMPLOYEE
WHERE DEPARTMENT.DEPTNO = EMPLOYEE.WORKDEPT AND DEPARTMENT.DEPTNAME = :HV00008 :H100008

Optimized Statement:
-------------------
SELECT 4.0C0
FROM
  (SELECT COUNT(*)
  FROM
    (SELECT 9$IDS
    FROM DB2INST1.EMPLOYEE AS Q1, DB2INST1.DEPARTMENT AS Q2
    WHERE Q2.DEPTNAME = :HV00008 AND (Q2.DEPTNO = Q1.WORKDEPT)) AS Q3) AS

Access Plan:
-------------
Total Cost: 7.6335
Query Degree: 1

<table>
<thead>
<tr>
<th>ROWS</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td>1.0</td>
</tr>
<tr>
<td>I/O</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>GFSBY</td>
<td>2.0</td>
</tr>
<tr>
<td>7.63304</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>3.0</td>
</tr>
<tr>
<td>NLJOIN</td>
<td>3.0</td>
</tr>
<tr>
<td>7.63252</td>
<td>1.0</td>
</tr>
</tbody>
</table>

From the previous access plan, you can see that DB2 chose a nested loop join to execute this statement by default. DEPARTMENT is picked up as the outer table (using table scan method), and EMPLOYEE is picked up as the inner table (using index scan method, XEMP2 is one index of table EMPLOYEE). In the following examples, you will use the optimization profile to modify this join sequence (letting EMPLOYEE be the outer table and DEPARTMENT be the inner table). The purpose of doing this is not for better performance, but only to show that the optimization profile has been applied to the SQL statement.

Creating the optimization profile

As shown in Listing 10, use the internal form of the SQL statements defined in stored procedure to create the optimization profile file.
Listing 10. Script of optimization profile

```xml
<?xml version='1.0' encoding='UTF-8'?>
<OPTPROFILE VERSION='9.1.0.0'>
  <STMTPROFILE ID='example profile'>
    <STMTKEY>
      <![CDATA[SELECT COUNT(*) INTO :HV00009 :HI00009 FROM DEPARTMENT, EMPLOYEE WHERE DEPARTMENT.DEPTNO = EMPLOYEE.WORKDEPT AND DEPARTMENT.DEPTNAME = :HV00008 :HI00008]]>
    </STMTKEY>
    <OPTGUIDELINES>
      <NLJOIN>
        <ACCESS TABLE='EMPLOYEE'/>
        <ACCESS TABLE='DEPARTMENT'/>
      </NLJOIN>
    </OPTGUIDELINES>
  </STMTPROFILE>
</OPTPROFILE>
```

The following points from the previous script should be paid special attention to.

- **ID attribute of STMTPROFILE element**: the value of ID attribute is the unique identify of this statement level optimization guideline in this optimization profile. If this optimization guideline is applies to some SQL statement, you can see this identify from the access plan of the statement.
- **STMTKEY element**: you must use the internal form of SQL statement as the STMTKEY, otherwise, it can't be matched successfully. In common, you always use `<![CDATA[[]]>` to enclose the statement.
- **NLJOIN element**: this element specifies the sequence of join. Here, you use EMPLOYEE as the outer table and DEPARTMENT as the inner table.

**Insert the optimization profile into the database**

After defining the optimization profile, save it into file `test_profile.prof`. Then load this data into table SYSTOOLS.OPT_PROFILE, then use the import command to finish this operation.

First, define the data source file for import with the name `profile_file.load`. As shown in Listing 11, this file specifies the schema name of the optimization profile as Test, the name of the optimization profile is OPTPROF, and the detailed content of the optimization profile is contained in the `test_profile.prof` file.

**Listing 11. data source file of import**

```
"TEST", "OPTPROF", "test_profile.prof"
```

Then run the import command, as shown in Listing 12 and Figure 7, to load the data into the database.

**Listing 12. Script of import command**

```
db2 "IMPORT FROM profile_file.load OF DEL MODIFIED BY LOBSINFILE INSERT_UPDATE INTO SYSTOOLS.OPT_PROFILE"
```
Modify the definition file of stored procedure to use this optimization profile

If you want to use an optimization profile in your stored procedure, beyond setting the previous registry variable to enable this function, you also need to call SYSPROC.SET_ROUTINE_OPTS system stored procedure to set the schema name, and then name of the optimization profile that will be used during pre-compile and binding time, as shown in Listing 13. You need to specify the OPTPROFILE option in the parameter of SYSPROC.SET_ROUTINE_OPTS, and its value is TEST.OPTPROF. So the optimization profile you previously defined will be used in the access plans creation for the SQL statements in this stored procedure. Of course, you can get the same results by setting the DB2_SQLROUTINE_PREPOPTS registry variable.

Listing 13. Modified stored procedure

```sql
CONNECT TO SAMPLE%

CALL SYSPROC.SET_ROUTINE_OPTS('EXPLAIN ALL OPTPROFILE TEST.OPTPROF')%

CREATE PROCEDURE GET_EMP_NUM(
   IN DEPT_NO CHAR(50),
   OUT EMP_NUM INTEGER)
LANGUAGE SQL
BEGIN
   SELECT COUNT(*) INTO EMP_NUM
   FROM DEPARTMENT, EMPLOYEE
   WHERE DEPARTMENT.DEPTNO = EMPLOYEE.WORKDEPT
   AND DEPARTMENT.DEPTNAME = DEPT_NO;
END%
CONNECT RESET%
```

As shown in Listing 14, drop the stored procedure that has been created, and then re-create it.
Listing 14. re-create stored procedure

db2 connect to sample
db2 drop procedure GET_EMP_NUM
db2 -td% -vf create_procedure.ddl

View the new access plan to check the optimization profile is used

To check that the optimization profile is successfully used, you need to compare the original access plan with the new access plan.

First, as the stored procedure is re-created, DB2 will allocate a new package for it. Run the command shown in Listing 15 and Figure 8 to get the related package name for this stored procedure.

Listing 15. Get the package name of the stored procedure

db2 connect to sample
db2 -tvf get_routine_sqls.sql

Figure 8. New package name for the re-created stored procedure

---

P2270199
SELECT COUNT(*) INTO :HV00009
:HV00009 FROM DEPARTMENT, EMPLOYEES WHERE DEPARTMENT.DEPTNO = EMPLOYEE.WORKDEPT
AND DEPARTMENT.DEPTNAME = :HV00008 :HV00008

Then use the previous package name to view the new access plan of the stored procedure and save the result in file curr_plan.out, as shown in Listing 16.

Listing 16. Get new access plan

db2exfmt -d sample -e db2inst1 -g -l -n 'P2270199' -s db2inst1
-o curr_plan.out -w -l -# 0 -v %

From the new access plan, you can see there is an additional Profile Information part, as shown in Figure 9. It includes the schema name, and the name of the optimization profile that is used to generate the access plan of this SQL statement (in this example, it is TEST.OPTPROF), and the ID of the statement level optimization guideline that matches the statement. This ID is specified when you define the optimization profile.
Figure 9. New access plan of stored procedure_1

------------------- STATEMENT 7 SECTION 1 -------------------
QUERYNO: 7
QUERYTAG:
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Profile Information:
---------------------
OPT_PROF: (Optimization Profile Name)
  TEST.OPTPROF
STMT_PROF: (Statement Profile Name)
  example profile

Original Statement:
--------------------
SELECT COUNT(*) INTO :HV00009 :HL00009
FROM DEPARTMENT, EMPLOYEE
WHERE DEPARTMENT.DEPTNO = EMPLOYEE.WORKDEPT AND DEPARTMENT.DEPTNAME = :HV00009 :HL00009

Optimized Statement:
---------------------
SELECT Q4.$C0
FROM
  (SELECT COUNT(*)
  FROM
    (SELECT S4IDS
     FROM DB2INST1.EMPLOYEE AS Q1, DB2INST1.DEPARTMENT AS Q2
     WHERE (Q2.DEPTNAME = :HV00009) AND (Q2.DEPTNO = Q1.WORKDEPT)) AS Q3) AS Q4

At the same time, from the new access plan you can see that the sequence of the join is changed, as shown in Figure 10. Now, the outer table is EMPLOYEE table and the inner table is DEPARTMENT table. So you can prove that the optimization profile is used successfully to access plans generation of the SQL statements in stored procedure.
Conclusion

Stored procedures are a very useful function of DB2. SQL statements defined within a stored procedure are often very complex and hard to debug and tune. In some situations, the access plan selected by DB2 may not be the one you want to use, especially when you are debugging performance issues. By using an optimization profile, you can affect the access plans of SQL statements without changing the application and database settings. It is a very efficient tool for debugging your application, and optimizing your SQL.
Related topics

- Visit IBM DB2 9.7 for Linux, UNIX and Windows Information Center to get more detailed information about optimization profiles.
- Visit the developerWorks resource page for DB2 for Linux, UNIX, and Windows to read articles and tutorials and connect to other resources to expand your DB2 skills.
- Now you can use DB2 for free. Download DB2 Express-C, a no-charge version of DB2 Express Edition for the community that offers the same core data features as DB2 Express Edition and provides a solid base to build and deploy applications.
- 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.