Paging data in Oracle, MS SQL Server and DB2 LUW using database code

---

White Paper

Last revised: June 2006

Table of Contents

Paging data in Oracle, MS SQL Server and DB2 LUW using database code

Abstract

For all the UI applications, it is a requirement to allow the end user to page through the record sets. So, instead of displaying say 10000 records that a selection criteria might qualify, the end user wants to see the first 25 records and then the next 25 and so on. Most of the time, the page size is configurable as well using a configuration option that is user based. There are many different schemes available to do paging of the record sets in the application. Some developers prefer to do it via client side coding (ASP.Net or Java), while some prefer to do it using database side code using SQL, T-SQL (MS SQL Server), PL/SQL (Oracle) or SQL/PL (DB2 LUW).

In this whitepaper, we will present the options that you can use to implement paging using database code. With the advent of SQL Server 2005, even it has a good set of analytic functions that make it very easy to implement paging using database code. Oracle and DB2 always had these options. If you want to select only the top n records from a query (or a range of records say 25 records from 25th to the 50th record), there are differences in the syntax and the logic used for the 3 major RDBMS that are used in enterprise applications → Oracle, MS SQL Server and DB2 LUW.

Sample Data to illustrate the paging of records

```
/*********************************************************************************************
 Create a dummy EMP_MASTER table populate it with some records for illustration.
 This is SQL Server Syntax.
*********************************************************************************************/
CREATE TABLE EMP_MASTER (EMP_ID INT IDENTITY PRIMARY KEY, EMP_NAME VARCHAR(10), SALARY
INT)
GO
INSERT INTO EMP_MASTER(EMP_NAME, SALARY) VALUES ('A', 100)
INSERT INTO EMP_MASTER(EMP_NAME, SALARY) VALUES ('B', 1000)
INSERT INTO EMP_MASTER(EMP_NAME, SALARY) VALUES ('C', 10000)
INSERT INTO EMP_MASTER(EMP_NAME, SALARY) VALUES ('D', 100000)
INSERT INTO EMP_MASTER(EMP_NAME, SALARY) VALUES ('E', 150000)
GO
```

Selecting top n records

SQL Server:

Option # 1: Using the TOP clause:

```
SELECT TOP 2 EMP_NAME, SALARY
FROM EMP_MASTER
ORDER BY SALARY DESC
GO
```

In the query above, the order by occurs first and then the top 2 records are presented to the end user.

Option # 2: Using ROWCOUNT:

```
SET ROWCOUNT 2
SELECT EMP_NAME, SALARY
FROM EMP_MASTER
ORDER BY SALARY DESC
SET ROWCOUNT 0
GO
```

In the above query, the set rowcount ensures that the processing of the records will stop after 2 records are returned back from the query. The last statement that re-sets the rowcount to 0 is used to ensure that the previous settings of rowcount are restored.

ORACLE

```
SELECT * FROM (SELECT EMP_NAME, SALARY FROM EMP_MASTER ORDER BY SALARY DESC)
WHERE ROWNUM < 3
/
```

In the case of Oracle, we use an inline view (the select statement in the from clause) and apply the rownum (pseudo column in Oracle) to filter off the records to return the top 2 records.

NOTE: Please do note that in the case of Oracle, you need to apply the rownum condition after the order by (order by will be in the inline view) else your results will be wrong since rownum gets assigned as the data is being ordered.

DB2 LUW

```
SELECT EMP_NAME, SALARY
FROM EMP_MASTER
ORDER BY SALARY DESC
FETCH FIRST 2 ROWS ONLY;
```

In the case of DB2 LUW, the selection of the top 2 records is achieved by using the "fetch first n rows only" clause as shown above.

Selecting a range of records

If you want to select records from x to y in a range of records say from 25 to 50, then you can do this (for these examples, assume that you have a table called user_master with lots of users in the table and you want to return records fro 25 – 50 and want to order by user_name in a descending order → this can easily be extended t queries that have join conditions, filter clauses, order by clauses etc.):

/**********

DB2 PAGING SQL CODE

```
**************************************************************/
SELECT * FROM
(SELECT ROWNUMBER() OVER() AS ROW_NUM, INNER_TABLE.*
 FROM (SELECT * FROM USER_MASTER order by USER_NAME DESC FETCH FIRST 50 ROWS ONLY) as
INNER_TABLE ) AS OUTER_TABLE
 WHERE OUTER_TABLE.ROW_NUM > 25
 OPTIMIZE FOR 50 ROWS;
```

As you can see from the query above, the rownumber() function and the windowing function is used to achieve the range of records that need to be returned back to the end user. The "optimize for" clause is optional and is used only to indicate to the optimizer to optimize the query for fast retrieval of the 50 records since the range that needs to be fetched in the end is from row 25 – row 50.

```
/**************************************************************
```

ORACLE PAGING SQL CODE

```
**************************************************************/
SELECT * FROM ( SELECT ROWNUM ROW_NUM, A.* FROM ( SELECT * FROM USER_MASTER ORDER BY
USER_NAME DESC) A
                    WHERE ROWNUM <= 50 ) WHERE ROW_NUM > 25
/
```

In the case of Oracle, using similar logic as in the case of DB2 LUW, one can have the inline view and then using the rownum pseudo-column twice, one can form the range. This method of paging is very effective and uses the COUNT(STOPKEY) optimization very effectively.

```
/**************************************************************
```

SQL SERVER PAGING SQL CODE

```
**************************************************************/
```

In SQL Server 2000, since there were no analytic functions, one had to make use of rowcount and temp tables (or table variables) to achieve the range set retrieval of records. if you want to write up a generic paging code, then you need to write up a stored procedure that takes in the string that needs to be executed (application needs to take care of the SQL injection issues in this case), the number of records to be returned, the starting number and two output parameters. Here is one simple proc. That would do that for you:

```
CREATE PROC USP_PAGING
   @SQLSTRING VARCHAR(7900),
   @NUMBER_OF_RECORDS INT,
   @START_RECORD VARCHAR(10),
   @TOTAL_RECORDS INT = 0 OUTPUT,
   @RETURN_CODE INT = 0 OUTPUT, --this is success
   @COUNT_FLAG BIT = 1
AS
BEGIN
 --SET THE NOCOUNT FLAG ON
--Set the other options that need to be set (like for indexed views etc.)

   SET NOCOUNT ON
   SET ANSI_NULL_DFLT_ON ON
   SET CONCAT_NULL_YIELDS_NULL OFF
   SET DATEFORMAT mdy
```

```
  SET ANSI_NULLS ON
  SET ANSI_PADDING ON
  SET ANSI_WARNINGS ON
  SET ARITHABORT ON
  SET CONCAT_NULL_YIELDS_NULL ON
  SET QUOTED_IDENTIFIER ON
  SET NUMERIC_ROUNDABORT OFF

  DECLARE @TOTALSET INT
  DECLARE @ISQLERROR INT
  SELECT @ISQLERROR = 0            --indicating success
  SELECT @TOTALSET = (@NUMBER_OF_RECORDS - 1) + CONVERT(INT, @START_RECORD)
  SET ROWCOUNT @TOTALSET       --Setting the rowcount to the total number of records
  DECLARE @COMMAND VARCHAR(8000), @SQLSTRING2 VARCHAR(7900)
  /*****************************************************************
  Get the position information for the first FROM clause
  We need to replace this with the temp table clause.
  *****************************************************************/
  DECLARE @COUNT INT
  SELECT @COUNT = CHARINDEX (' FROM ', @SQLSTRING)
  DECLARE @SELECT INT, @SELECT2 INT
  SELECT @SELECT = CHARINDEX ('SELECT ' , @SQLSTRING)
  SELECT @SELECT2 = CHARINDEX (' DISTINCT ', SUBSTRING(@SQLSTRING, 1, 17))
  IF @SELECT2 > 0
  BEGIN
  SELECT @SQLSTRING2 = STUFF(@SQLSTRING, @SELECT, 16, 'SELECT DISTINCT TOP 100 PERCENT ')
  END
  ELSE
  BEGIN
  SELECT @SQLSTRING2 = STUFF(@SQLSTRING, @SELECT, 7, 'SELECT TOP 100 PERCENT ')
  END
  /*************************************************************************
  Use an in-line view in which we do the order by and hence the sorting
  operation occurs and then the data gets inserted into the temp table.
  That way, the order of the identity values does not get messed up because
  of the data ordering.
  *************************************************************************/

  SELECT @COMMAND = 'SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED; SELECT
IDENTITY(INT, 1, 1) AS ROW_NUM, A.* INTO #TEMP FROM (' + @SQLSTRING2 + ') AS A OPTION (FAST 1);
SELECT * FROM #TEMP WHERE ROW_NUM >= '+''+ @START_RECORD +''+' ORDER BY ROW_NUM'
  EXEC (@COMMAND)
  SELECT @ISQLERROR = @@ERROR          --IF SUCCESS, @@ERROR = 0
  SET ROWCOUNT 0          --Setting the rowcount back to 0
  --Now, go and get the total count of the records
 IF @COUNT_FLAG = 1
 BEGIN
  DECLARE @COMMAND4 VARCHAR(7900)
  SELECT @COMMAND4 = STUFF(@SQLSTRING, @COUNT, 6 , ' INTO #TEMPTABLE3 FROM ')
  DECLARE @COMMAND5 VARCHAR(100)
  SELECT @COMMAND5 = ' ;SELECT COUNT(*) FROM #TEMPTABLE3'
  DECLARE @COMMAND6 VARCHAR(8000)
--THE COMMAND STRING FOR GETTING THE TOTAL COUNT
  SELECT @COMMAND6 = @COMMAND4 + @COMMAND5
  CREATE TABLE #TOTALCOUNT(TOTAL_COUNT INT)
  INSERT INTO #TOTALCOUNT EXEC (@COMMAND6)
  SELECT @TOTAL_RECORDS = TOTAL_COUNT FROM #TOTALCOUNT
  SELECT @RETURN_CODE = @ISQLERROR
 END
 ELSE
 BEGIN
  SELECT @TOTAL_RECORDS = 0
```

```
   SELECT @RETURN_CODE = @ISQLERROR
  END
   --SET THE NOCOUNT FLAG OFF
   SET NOCOUNT OFF
END

GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO
```

In SQL Server 2005, one can make use of the row_number() analytic function in order to do paging. Here is an example:

```
SELECT * FROM
(
SELECT ROW_NUMBER() OVER (ORDER BY ROWNUM) AS ROW_NUM, *
FROM
(           SELECT col1, col2, col3...,
                        ROW_NUMBER() OVER
                        (ORDER BY col4 desc) AS ROWNUM
            FROM <your joined tables and filter criteria here> a
) AS DT1
WHERE DT1.ROWNUM <  @rowNumberEnd) AS DT2
WHERE DT2.ROW_NUM >= @rowNumberStart
```

This is on the same lines as the code that was shown above for DB2 LUW and Oracle.

Summary

So, this is how paging can be achieved on the database side in Oracle, MS SQL Server and DB2 LUW. There are other ways of doing paging as well like using the client side (ASP.Net or Java) code to do the paging but in our experience, the server side (database code) works out the best and is most performant of all the options available. Of course, you need to ensure that you have good index design to cover the queries that you are using. We hope that you will find this whitepaper useful for your work.