Betrifft	**WHERE Klausel Generierung mit .NET und Oracle**
Art der Info	**Technical Info**
Quelle	**Aus unserer Projekterfahrung und Architektur-Kurs**

Where ist the WHERE?

Der Artikel untersucht die Möglichkeiten, wie SQL WHERE-Klauseln dynamisch generiert werden. Untersucht werden Generierung mittels .NET und der Aufruf von Oracle REF CURSOR in Stored Procedures (Packages).

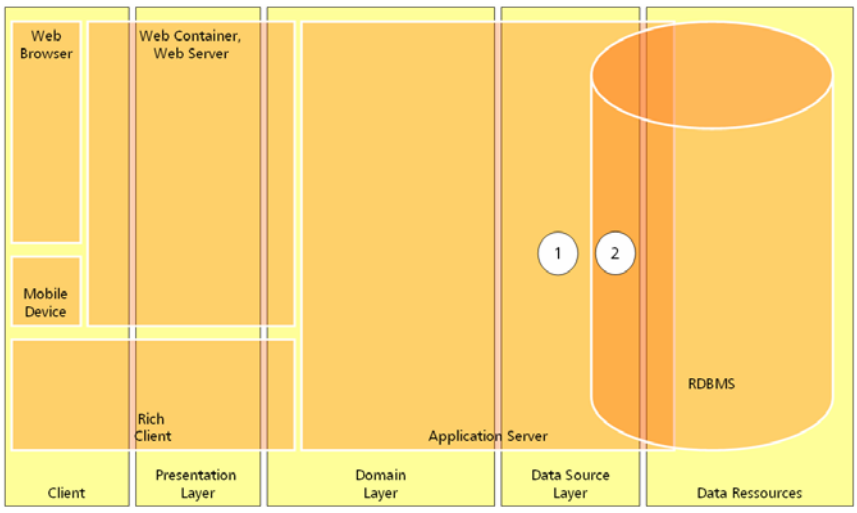
Der Leser sollte mit C# und PL/SQL Syntax vertraut sein.

Einführung

Die Generierung einer optimalen WHERE-Klausel kann für die Performance entscheidend sein.

Optimal bedeutet in diesem Zusammenhang die schnellst mögliche Reduktion der Resultatmenge in der Datenbank, sowie die best mögliche Verarbeitung durch den Optimizer.

In einer Mehrschichten-Architektur, gehört die Generierung der WHERE-Klausel in den „Data Source Layer". Dieser kann physisch jedoch im Middle-Tier (Application Server) oder in der Datenbank implementiert sein, zumindest in der von der Trivadis definierten Mehrschichten-Architektur für „Datenbanknahe Applikationen":



Im Prinzip gibt es also zwei Möglichkeiten:

1. Generierung der WHERE-Klausel im Middle-Tier (Application Server)

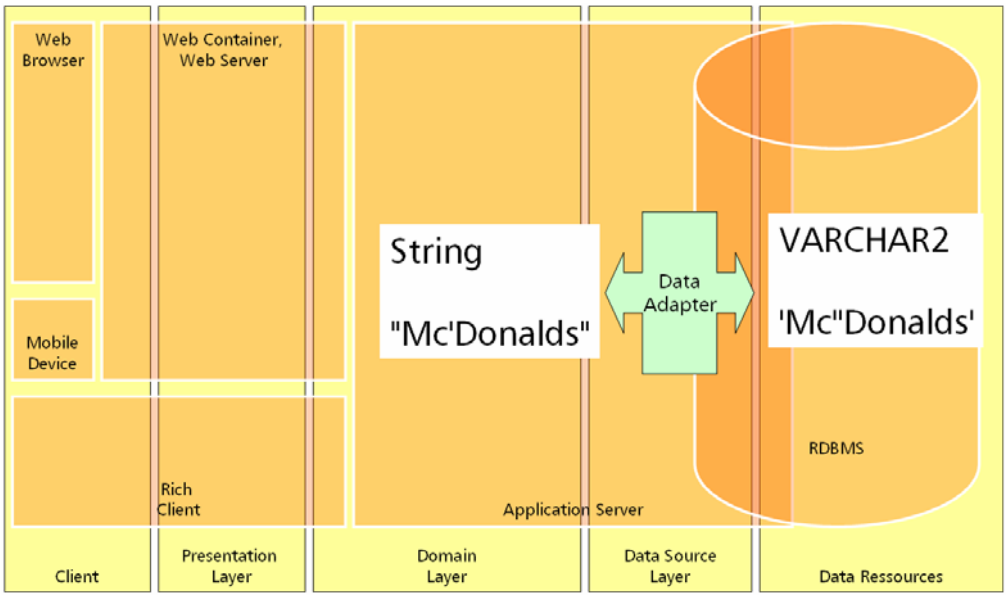
2. Generierung der WHERE-Klausel in der Datenbank in einer Stored Procedure. Gerade mit Oracle scheint diese Möglichkeit auf den ersten Blick attraktiv zu sein, da Oracle nebst normalen OUT Parameter auch REF CURSOR unterstützt.

Basisprinzip mit Adaptern

.NET liefert mit den ADO.NET Providern eine mächtige Basis-Infrastruktur. Jeder ADO.NET Provider muss einige Interface implementieren. Dazu gehören IDbConnection, IDbCommand, IDbDataAdapter und IDataReader.

Gerade IDbAdatapter ist ein starker Punkt in dieser Architektur. Wie der Name schön andeutet, ist dieses Interface eine Umsetzung des Adapter Patterns, welches jedoch auf „Daten“ spezialisiert ist.

Ein Adapter „übersetzt“ zwischen zwei Welten. Ähnlich wie Stromstecker-Adapter, wenn man z.B. einen Schweizer Stromstecker in Deutschland verwenden will. Der .NET Adapter „übersetzt“ z.B. Datentypen (**Oracle: VARCHAR2, .NET String**), oder kümmert sich um die richtige Umsetzung des Datumformats oder quoted Strings korrekt (**Oracle: 'Mc''Donalds', .NET "Mc'Donalds"**).



Das heisst, es gibt nur gerade einen richtigen Ort, die WHERE-Klausel zu generieren und das ist im DataAdapter und zwar über IDbCommand und der dazugehörenden IDataParameterCollection, weil so das ganze Datatype Handling vom .NET DataProvider übernommen wird.

Allerdings ist damit noch nicht klar, ob das Erstellen eines SQL-Strings im Middle-Tier (Application Server) oder der Aufruf einer Stored Procedure besser ist.

1. Erstellen eines SQL-Strings

Achtung:

Dynamische Generierung von WHERE-Klauseln als String im Middle-Tier (Application Server) oder in der Datenbank sollten auf gar keinen Fall gemacht werden, da sie keine optimale Umsetzung der Datentypen garantieren und grundsätzlich unsicher sind (z.B. SQL-Injection). Es gilt der Grundsatz, dass Parameter im Command-Object verwendet werden müssen.

Vorteile:

- Die WHERE-Klausel wird zur Laufzeit via Parameter-Objects geniert und enthält exakt die Bedingungen, die für den Optimizer relevant sind
- Verwendung von Bind-Variablen und minimaler Parse-Aufwand in der Datenbank

Nachteile:

- Kapselung der Datenbank nur über Views

Vereinfachtes Beispiel mit fixer WHERE-Klausel, um das Basisprinzip zu erläutern (txtJob und txtSal enthalten die Werte für die Parameter):

```
String tableName = "Emp";
try
{
    DataSet ds = new DataSet();
    OracleConnection connection = new OracleConnection("Data Source=DB1; User Id=scott; Password=tiger;");
    OracleCommand select = new OracleCommand("SELECT * FROM emp " +
                                             " WHERE job LIKE :job " +
                                             "   AND sal > :sal", connection);
    // Parameter JOB
    OracleParameter job = new OracleParameter("job",OracleDbType.Varchar2, 20, ParameterDirection.Input);
    job.Value = txtJob.Text.ToUpper();
    select.Parameters.Add(job);

    // Parameter SAL
    OracleParameter sal = new OracleParameter("sal",OracleDbType.Decimal, ParameterDirection.Input);
    sal.Value = Decimal.Parse(txtSal.Text);
    select.Parameters.Add(sal);

    // Fetch data
    OracleDataAdapter adapter = new OracleDataAdapter(select);
    adapter.Fill(ds, tableName);
```

Natürlich ist es nun das Ziel die WHERE-Klausel nicht fix, sondern eben dynamisch zu machen.

Ein besseres Beispiel mit dynamischer WHERE-Klausel, welches in der Praxis noch durch eine Collection und einem **`loop`**, anstatt einem **`if`** für jeden Parameter ersetzt wird.

```
String tableName = "Emp";
String condition = " WHERE ";
OracleConnection connection = new OracleConnection(
                                  "Data Source=DB1; User Id=scott; Password=tiger;");
StringBuilder commandText = new StringBuilder();
OracleCommand select = new OracleCommand();;
OracleParameter job = null;
OracleParameter sal = null;

try
{
    DataSet ds = new DataSet();
    commandText.Append("SELECT * FROM emp ");

    // Parameter JOB
    if (txtJob.Text != String.Empty)
    {
        commandText.Append(condition);
        commandText.Append(" job LIKE :job ");
        job = new OracleParameter("job",OracleDbType.Varchar2, 20, ParameterDirection.Input);
        job.Value = txtJob.Text.ToUpper();
        condition = " AND ";
    }

    // Parameter SAL
    if (txtSal.Text != String.Empty)
    {
        commandText.Append(condition);
        commandText.Append(" sal > :sal ");
        sal = new OracleParameter("sal",OracleDbType.Decimal, ParameterDirection.Input);
        sal.Value = Decimal.Parse(txtSal.Text);
        condition = " AND ";
    }

    select = new OracleCommand(commandText.ToString(), connection);
    // can add parameters only AFTER commandText is fully known
    if (job != null) {select.Parameters.Add(job);}
    if (sal != null) {select.Parameters.Add(sal);}

    // Fetch data
    OracleDataAdapter adapter = new OracleDataAdapter(select);
    adapter.Fill(ds, tableName);
```

2. Aufruf einer Stored Procedure

Vorteile:

- Kapselung in der Datenbank

Nachteile:

- Die Stored Procedure muss Parameter für alle Attribute in der WHERE-Klausel definieren, auch dann wenn sie für die aktuelle Abfrage nicht verwendet werden
- Generierung einer optimalen WHERE-Klausel nur über dynamisches SQL möglich. Ansonsten komplexe Abfragen, ob ein Wert für einen bestimmten Parameter übergeben wurde (z.B. WHERE attr = param OR param IS NULL)

Definition des Oracle Package:

```
PACKAGE EMP_DAL IS
  TYPE emp_ref_cursor IS REF CURSOR RETURN emp%ROWTYPE;

  PROCEDURE get_emps(p_job IN VARCHAR2,
                     p_sal IN NUMBER,
                     p_emp_ref_cursor OUT emp_ref_cursor);
END EMP_DAL;
```

Die Procedure weiss nicht, welche Parameter „gefüllt“ und welche nicht. Deshalb ist die OR-Logik nötig, welche gerade dem Oracle Optimizer bei komplexeren Abfragen Mühe bereitet.

```
PACKAGE BODY EMP_DAL  IS
  PROCEDURE get_emps(p_job IN VARCHAR2,
                     p_sal IN NUMBER,
                     p_emp_ref_cursor OUT emp_ref_cursor) IS
    BEGIN
    OPEN p_emp_ref_cursor FOR
         SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno
           FROM emp
          WHERE (job LIKE p_job OR p_job IS NULL)
            AND (sal > p_sal OR p_sal IS NULL);
    END;
END EMP_DAL;
```

Der Aufrufer übergibt nun ständig alle Parameter, da die Stored Procedure die Generierung der WHERE-Klausel übernimmt.

```
String tableName = "Emp";
OracleConnection connection = new OracleConnection("Data Source=DB1; User Id=scott; Password=tiger;");
OracleCommand select = new OracleCommand();
OracleParameter job = new OracleParameter("p_job", OracleDbType.Varchar2, 20, ParameterDirection.Input);
OracleParameter sal = new OracleParameter("p_sal", OracleDbType.Decimal, ParameterDirection.Input);
OracleParameter refCursor = new OracleParameter("p_emp_ref_cursor", OracleDbType.RefCursor, ParameterDirection.Output);
try
{
    DataSet ds = new DataSet();
    select.Connection = connection;
    select.CommandText = "EMP_DAL.GET_EMPS";
    select.CommandType = CommandType.StoredProcedure;

    if (txtJob.Text == String.Empty)
    {
        job.Value = DBNull.Value;
    }
    else
    {
        job.Value = txtJob.Text.ToUpper();
    }

    if (txtSal.Text == String.Empty)
    {
        sal.Value = DBNull.Value;
    }
    else
    {
        sal.Value = Decimal.Parse(txtSal.Text);
    }

    select.Parameters.Add(job);
    select.Parameters.Add(sal);
    select.Parameters.Add(refCursor);

    OracleDataAdapter adapter = new OracleDataAdapter(select);
    adapter.Fill(ds, tableName);
```

Natürlich kann in der Stored Procedure auch mit dynamischem SQL gearbeitet werden. Die einfachste Möglichkeit ist das SQL Statement dynamisch zu erstellen. Dies erstellt allerdings keine optimale SQL Statements, da keine Bind-Variablen mehr verwendet werden. Zudem müssen weak REF CURSORs eingesetzt werden.

```
PACKAGE EMP_DAL IS
  TYPE emp_ref_cursor IS REF CURSOR;

  PROCEDURE get_emps(p_job IN VARCHAR2,
                     p_sal IN NUMBER,
                     p_emp_ref_cursor OUT emp_ref_cursor);
END EMP_DAL;
```

```
PACKAGE BODY EMP_DAL  IS
  PROCEDURE get_emps(p_job IN VARCHAR2,
                     p_sal IN NUMBER,
                     p_emp_ref_cursor OUT emp_ref_cursor) IS

    condition VARCHAR2(7) := ' WHERE ';
    stmt VARCHAR2(2000) :=
                'SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno' ||
                '  FROM emp ';
    BEGIN
    IF (p_job IS NOT NULL) THEN
      stmt := stmt || condition || ' job LIKE '''||p_job||'''';
      condition := ' AND ';
    END IF;

    IF (p_sal IS NOT NULL) THEN
      stmt := stmt || condition || ' sal > ' || p_sal;
      condition := ' AND ';
    END IF;

    OPEN p_emp_ref_cursor FOR stmt;

    END;
END EMP_DAL;
```

Diesen Code optimal mit Bind Variablen zu schreiben wird nun mit PL/SQL zu einer Herausforderung, da auch die USING-Klausel dynamisch sein muss.

z.B. mit einem Parameter gefüllt

```
OPEN p_emp_ref_cursor
  FOR SELECT empno, ename
        FROM emp
       WHERE job LIKE :job
  USING p_job
```

und wenn 2 Parameter gefüllt sind

```
OPEN p_emp_ref_cursor
  FOR SELECT empno, ename
        FROM emp
       WHERE job LIKE :job
         AND sal > :sal
  USING p_job, p_sal
```

Einen dynamischen REF CURSOR mit DBMS_SQL zu programmieren überlasse ich dem Leser, ich habe es gar nicht versucht. Die dynamische WHERE-Klausel im Middle-Tier (Application Server) mit .NET ist garantiert einfacher und schlussendlich effizienter, da optimal.

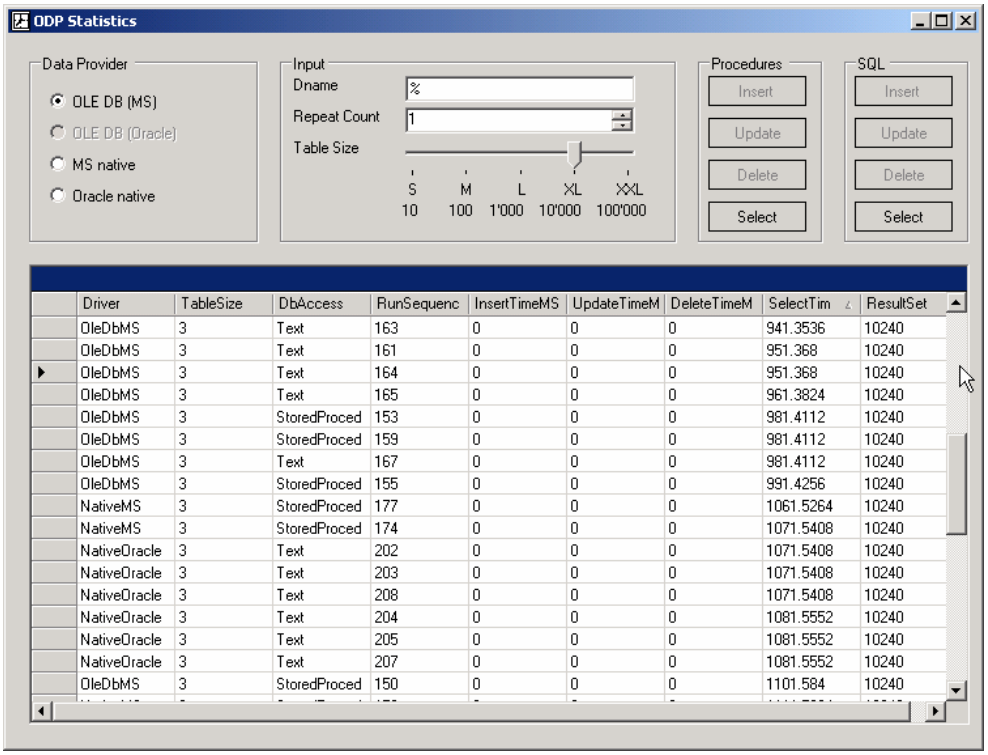
Bleibt die Frage, der Performance. Stored Procedures müssten doch wenigstens schneller sein!

Performance Tests

Trivadis hat einige Tests gemacht, wo in der Applikation die Datenmenge wie folgt bestimmt werden kann:

Small	=	10 Rows
Medium	=	100 Rows
Large	=	1'000 Rows
Extra Large	=	10’000 Rows
Very Extra Large	=	100’000 Rows

Für interaktive Applikationen dürften vermutlich nur gerade Werte < 10'000 Rows interessant sein.



Driver	TableSize	DbAccess	RunSequenc	InsertTimeMS	UpdateTimeM	DeleteTimeM	SelectTim	ResultSet
OleDbMS	3	Text	163	0	0	0	941.3536	10240
OleDbMS	3	Text	161	0	0	0	951.368	10240
OleDbMS	3	Text	164	0	0	0	951.368	10240
OleDbMS	3	Text	165	0	0	0	961.3824	10240
OleDbMS	3	StoredProced	153	0	0	0	981.4112	10240
OleDbMS	3	StoredProced	159	0	0	0	981.4112	10240
OleDbMS	3	Text	167	0	0	0	981.4112	10240
OleDbMS	3	StoredProced	155	0	0	0	991.4256	10240
NativeMS	3	StoredProced	177	0	0	0	1061.5264	10240
NativeMS	3	StoredProced	174	0	0	0	1071.5408	10240
NativeOracle	3	Text	202	0	0	0	1071.5408	10240
NativeOracle	3	Text	203	0	0	0	1071.5408	10240
NativeOracle	3	Text	208	0	0	0	1071.5408	10240
NativeOracle	3	Text	204	0	0	0	1081.5552	10240
NativeOracle	3	Text	205	0	0	0	1081.5552	10240
NativeOracle	3	Text	207	0	0	0	1081.5552	10240
OleDbMS	3	StoredProced	150	0	0	0	1101.584	10240

Getestet wurden die beiden .NET Provider für Oracle. Einerseits von Microsoft und anderseits von Oracle mit .NET Version 1.1 und einer Oracle9i Release 2 Datenbank mit SQL im Middle-Tier (Application Server) generiert und mit REF CURSOR in Stored Procedures.

Testresultate

Beide Provider bewegen sich in ähnlichen Grössenordnungen betreffend der Performance. Der Microsoft Provider ist bei kleineren Datenmengen, der Oracle Provider bei grösseren Datenmengen leicht schneller.

Erstaunlicherweise ist die Generierung von SQL mittels Parameter im Command-Object schneller als der Aufruf von Stored Procedures. Bei 1'000 Rows liegen die Zugriffszeiten bei ca. 100 ms.

Wie bereits erklärt, ist die Generierung von SQL mittels Parameter im Command-Object auch flexibler und einfacher, auch für den Optimizer. Somit ist klar, dass der DataAdapter für SELECT-Befehle via SQL aus dem Middle-Tier (Application Server) auf Views (Kapselung!) in der Datenbank zugreifen soll.

Details der Tests können wir aus unterschiedlichen Gründen nicht publizieren. Gerne bin ich bereit schriftliche Anfragen zu beantworten.

Fazit

Der Artikel zeigt, dass die Generierung von SELECT-Befehlen im Middle-Tier (Application Server) zusammen mit dem Command-Object und den dazugehörigen Parametern nicht nur konzeptionell richtig, sondern auch flexibler und einfacher ist. Erstaunlicherweise ist es meistens auch schneller, wobei die Performance-Unterschiede zu klein sind, um damit einen Architekturentscheid herbei zu führen.

Trivadis hat eine Basis-Architektur für „Datenbanknahe Applikationen“ entwickelt, welche genau dieses Pattern umsetzt. Stored Procedures haben darin durchaus Platz, werden aber nicht zum Lesen der Daten empfohlen.

.NET zusammen mit Oracle ist bei vielen Firmen ein beliebtes und erfolgreiches Gespann. Die vorstellten Konzepte helfen, die optimale Abfrage Performance zu erreichen.

Die .NET Architektur für „Datenbanknahe Applikationen“ ist auch in einem 3-tägigen Trivadis Kurs vollständig dokumentiert und erlernbar.

Greetings byebye = new Greetings("Viel Erfolg und gute Performance“);

Urs Meier
Trivadis AG
Kanalstrasse 5
8152 Glattbrugg

Mail: urs.meier@trivadis.com
Tel: +41 808 70 20
Fax: +41 808 70 21
Internet: http://www.trivadis.com