Rdb Journal
technical corner

Guide to Using SQL: Computed and Automatic Columns

A feature of Oracle Rdb

By Ian Smith
Oracle Rdb Relational Technology Group
Oracle Corporation

Guide to Using SQL: Computed and Automatic Columns

Computed columns are nothing new to Oracle Rdb and have been available since its first release in 1984. A special type of column - know as a **computed by** column - defines a calculation instead of a data type. This special column takes no space within the table but allows the programmer to fetch the value at run-time using the **select** statement, or via a cursor.

Oracle Rdb builds on this support in release V7.1 by adding a new type of computed column; automatic columns. These columns are stored with the row but its value is calculated at either **insert** or **update** time and like **computed by** columns they are read only.

This article reviews all types of computed columns available in Rdb.[1] The examples in the article use SQL language from Oracle Rdb V7.1 or later.

COMPUTED BY Columns

To create a virtual or **computed by** column use either the **create** table or **alter table … add column** clause. This special column computes a value when selected or when referenced in a WHERE clause. Consider this simple example:

```
SQL> create table PERSON (
cont> (employee_id integer,
cont>  employee_id_disp computed by
cont> SUBSTRING (CAST(employee_id + 100000 as VARCHAR (6)) from 2)
cont>     …);
```

The column EMPLOYEE_ID_DISP is used to display the employee id with leading zeros. The combination of CAST, SUBSTRING and addition is stored within the table definition and relieves the application program from performing this type of conversion.

[1] This is a revised version of an article that was first published in May 2002.

The **computed by** expression can be a simple value expression such as CURRENT_USER, a call to a user define function, or as complex as allowed by the full power of the SQL subquery syntax.

```
SQL> alter table DEPARTMENTS
cont> add column EMPLOYEE_COUNT
cont>  computed by
cont>    (select count (*)
cont>     from JOB_HISTORY jh
cont>     where job_end is NULL -- in the current job
cont>     and jh.department_code = departments.department_code);
```

The values are computed at select time therefore no space is required within the row, and the expression need not be evaluated during **insert** or **update** of the table unless it is referenced in a **where** clause, a constraint or a trigger. Neither the **insert** nor the **update** statement may assign values to this type of column.

A **computed by** column may reference other computed columns in the same table. Rdb expands each expression when referenced so that the correct value is returned.

Note: If many **computed by** columns use subquery syntax in the computed by expression, then Oracle recommends that these calculations be replaced with calls to SQL functions to limit the tables contexts used by these computed columns.

AUTOMATIC Columns

Rdb includes another type of read-only column called AUTOMATIC AS columns. Automatic columns are closely related to **computed by** columns; however, the computed values are evaluated at INSERT and UPDATE time and stored in the database.

The database designer can define an AUTOMATIC column to be computed and stored during INSERT, UPDATE or during both these statements. These columns can also be used as part of an index key, and referenced by constraints.

```
SQL> create table PERSON
cont>     person_key
cont>          automatic insert as GET_NEW_ID () primary key,
cont>     …);
SQL> create unique index PERSON_INDEX on PERSON (person_key);
```

This partial example shows the use a SQL function to calculate a unique value for the primary key field. The column PERSON_KEY will inherit its data type from the value expression – in this case the returned data type from the SQL function. The CAST operator can be used to adjust the result data type. For example, if the expression returns a DOUBLE PRECISION value you may prefer to have the column return a BIGINT(4) type instead.

As read-only columns they may not be targets for an UPDATE or INSERT statement and are therefore ideal for calculating auditing information that you do not want modified by unprivileged users.

The SQL syntax allow for three types of AUTOMATIC columns: **automatic insert as** that is calculated only during an INSERT statement, **automatic update as** that is calculated only during an UPDATE statement and **automatic as** that is calculated at both UPDATE and INSERT time.

Consider this simple example:

- When a row is inserted track who executed the statement (CURRENT_USER) and when this action occurred (CURRENT_TIMESTAMP).
- When a row is inserted or updated record the change timestamp.

```
SQL> create table PRODUCTS
cont>     product_id     integer primary key,
cont>     entered_by     automatic insert as current_user,
cont>     change_dt      automatic as current_timestamp,
cont>     …);
```

Here the ENTERED_BY column will contain the user name of the user who inserted the row, and the current timestamp will be written to CHANGE_DT. Rdb will revise the CHANGE_DT column automatically during subsequent UPDATE statements.

Note: in prior versions read-only columns were included in the default column list for INSERT, even though they could not be modified. Starting with Rdb V7.1 read-only columns (COMPUTED BY and AUTOMATIC) are excluded from the default column list for INSERT. This simplifies programming by allowing new COMPUTED BY and AUTOMATIC columns to be added without requiring changes to existing code.

Identity Columns

Rdb allows one column of a table to have the IDENTITY attribute. This attribute changes the column to an AUTOMATIC column with an associated sequence. Most of the comments concerning AUTOMATIC columns in this article also apply to IDENTITY columns.

```
SQL> create table PERSON
cont>     person_key
cont>          integer identity primary key,
cont>     …);
SQL> create unique index PERSON_INDEX on PERSON (person_key);
```

This is a very similar example to that shown above for Automatic columns. In this case we use **identity** rather than using a SQL function to compute the unique number. Adding **primary key** allows other tables to reference this column from **foreign key** definitions.

Frequently Asked Questions about AUTOMATIC columns

What if the updated or inserted data is wrong, how can I fix it? A privileged user who has the database privilege DBADM can use SET FLAGS 'AUTO_OVERRIDE' statement to disable the AUTOMATIC column for new queries in the current session. The columns are temporarily treated as read-write columns and can be updated, or new rows inserted.

This is a common requirement when reloading data during database restructuring. Use the RDMS$SET_FLAGS logical or use the SET FLAGS statement with the keyword AUTO_OVERRIDE prior to running update queries that reference the AUTOMATIC or IDENTITY columns for update or insert.

The SQL IMPORT statement enables this flag automatically when importing tables with AUTOMATIC columns so that previously recorded values are not replaced when loading the new database.

I need to reload the current table for database restructuring. How do I retain the current values for the automatic and identity columns? If the table is being reorganized it may be necessary to unload the data and reload it after the storage map and indexes for the table are re-created, yet the old data must remain the same.

Normally, RMU Unload does not unload columns marked as AUTOMATIC, you must use the /VIRTUAL_FIELD qualifier with the keyword AUTOMATIC to request this action.

```
$ rmu/unload-
/virtual_fields=(automatic) payroll_db people people.unl
```

Following the restructure of the database, the data can be reloaded. If the target columns are also defined as AUTOMATIC then RMU Load will not, by default, write to those columns. Therefore, you must use the /VIRTUAL_FIELD qualifier with the keyword AUTOMATIC to request that the **automatic insert** action be suppressed.

```
$ rmu/load/virtual_fields=(automatic) payroll_db people people.unl
```

Can I modify an existing column to an AUTOMATIC column? Yes. Supported modifications include:

- ALTER COLUMN may change an AUTOMATIC column to a normal updatable base column even if there exists constraints and indices, as long as the data types are the same
- A non-computed base column can be altered to be an AUTOMATIC column. The old data is retained and the column is made read-only.
- A non-computed base column can be altered to a COMPUTED BY column. The old data is will not be accessible (a warning is issued for interactive SQL) and references to that column will evaluate the COMPUTED BY expression. If indices or constraints reference this column then the ALTER TABLE statement will fail. Note that altering the column back to a base, or automatic column will allow older versions of the row data to be visible, any rows inserted while the column was a COMPUTED BY column will return NULL.
- If a column has a DEFAULT (base column or AUTOMATIC UPDATE AS column) and it is converted to a COMPUTED BY, AUTOMATIC AS or an AUTOMATIC INSERT AS column then the DEFAULT value is removed (as these types of columns are incompatible with DEFAULT).

Can I provide a DEFAULT for an AUTOMATIC column? A DEFAULT clause is only required when **automatic update as** is used. In this case during INSERT the default value will be used, or possibly NULL if none were provided. For **automatic as** or **automatic insert as** the computed expression will be applied to the new row, and so a **default** would redundant.

What type of constraints can be defined for AUTOMATIC columns? Prior to V7.1.0.3 only NOT NULL, and CHECK constraints were permitted. This restriction was lifted and all types of constraints are now allowed.

Can I use sequences in an automatic column? Yes, this is an ideal use of AUTOMATIC columns. Use the expression seq.NEXTVAL as the expression. The default data type will be BIGINT but you can add a CAST expression to use a numeric type acceptable to your application.

The following example shows the use of a sequence to provide PRIMARY KEY values to the department table.

```
SQL> set dialect 'sql99';
SQL> set display no row counter;
SQL> create sequence dept_id;
SQL> create table departments
cont> (dept_id automatic insert as dept_id.nextval primary key,
cont>  dept_name char(20));
SQL> insert into departments values ('Admin');
SQL> insert into departments values ('Engineering');
SQL> insert into departments values ('Accounting');
SQL> insert into departments values ('Marketing');
SQL> select * from departments;
     DEPT_ID   DEPT_NAME
          1    Admin
          2    Engineering
          3    Accounting
          4    Marketing
4 rows selected
```

The IDENTITY clause implicitly creates an AUTOMATIC column to deliver the results from the tables private sequence in just this manner.

What happens when a table is altered and an automatic insert column is added? In this case the automatic expression is used much like a **default** value defined for a column. Each existing row of the table is updated during **alter table** storing the result of the expression. If you plan to add several automatic columns to a table do so within a single **alter table** statement so that just a single scan of the table will be performed for the update.

Can an automatic column reference itself? I want to record the accesses to a row by automatically incrementing an integer column value. Yes this can be done, but requires two steps. Create the column and give it a data type, and then execute an **alter table … alter column** statement to adjust the computed expression. The two steps are required to provide a data type for the column.

```
SQL> alter table employees
cont> add column reference_count integer default 0;
SQL> alter table employees
cont> alter column reference_count
cont>     automatic update as reference_count + 1;
```

Oracle Rdb
Guide to Using SQL: Automatic Columns
September 2008 (originally May 2002)

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

Oracle Corporation provides the software
that powers the Internet.