Rdb Journal
technical corner

# Guide to Using SQL: Identity Columns

***A feature of Oracle Rdb***

By Ian Smith
Oracle Rdb Relational Technology Group
Oracle Corporation

The Rdb Technical Corner is a regular feature of the Oracle Rdb Web Journal. The examples in this article use SQL language from Oracle Rdb V7.1 and later versions.

## Guide to Using SQL: Identity Columns

There have been many requests for Oracle Rdb to generate unique numbers for use as PRIMARY KEY values. In Rdb 7.1 we chose to implement two models that capture the functionality of most SQL database systems on the market and reflect the current planning for the draft SQL Database Language Standard.

### *CREATE SEQUENCE and AUTOMATIC columns*

The identity feature described here is closely related to two other features of Rdb V7.1: the **automatic** columns, and the **create sequence** statement. Identity combines these two features into a simple package for creating table specific unique values.

The current draft SQL database language standard, informally called SQL:200x, defines identity as an *internal sequence generator*. This implies that the associated sequence is not stored in the system tables and has no private name. That is, it is not an explicitly named schema object such as that created using **create sequence**, instead a sequence is implicitly created when the **identity** clause is applied to one column of a table.

The IDENTITY syntax is currently quite simple.

```
IDENTITY [ (start-value [, increment]) ]
```

Note: In the future it is likely that Rdb will extend this syntax when the final SQL:200x standard is published.

The CREATE TABLE or ALTER TABLE … ADD COLUMN statement can be used to specify a column with the IDENTITY attribute. IDENTITY informs Rdb that this column is to be specially treated and deliver unique identification for all rows inserted into the table. Such columns are always read-only and cannot be assigned values in an **insert** or **update** statement.

The IDENTITY keyword can be followed by an optional parameter list that contains the starting value – the equivalent to the MINVALUE clause in a create sequence statement, and the increment value – the equivalent to the INCREMENT BY clause in a create sequence statement. If either is omitted they default to one (1) resulting in a simple sequence similar to this created sequence:

```
SQL> create sequence seqname start with 1 increment by 1;
```

The following example illustrates the use of identity columns in multiple tables. As you will see we can apply constraints to these columns and reference them in queries.

Firstly we define some domains to allow consistent usage of types in the tables and routines.

```
SQL> create domain MONEY as INTEGER (2);
SQL> create domain CUSTOMER_IDENT as INTEGER;
SQL> create domain PRODUCT_IDENT as INTEGER;
SQL> create domain ORDER_IDENT as INTEGER;
SQL> create domain PRODUCT_NAME as char (20);
SQL> create domain CUSTOMER_NAME as char (20);
```

Next we define the tables for a very simple Customer/Orders database. The products being ordered are described in the PRODUCTS table, we assign a unique number to each product using an IDENTITY column. This will be referenced from other tables as a **foreign key**. In a similar way we also define a CUSTOMERS table assigning unique values using **identity**[1].

```
SQL> create table PRODUCTS
cont>     (product_id       PRODUCT_IDENT identity
cont>                       primary key,
cont>      product_name     PRODUCT_NAME,
cont>      unit_price       MONEY,
cont>      unit_name        char (10));
```

[1] This table is not shown in this article.

The ORDERS table also uses an identity. For display purposes we will start the values at 10000 so that order will have five digits. This is achieved by including the START WITH value (also the MINVALUE) in the IDENTITY clause[2].

```
SQL> create table ORDERS
cont>     (order_id         ORDER_IDENT identity (10000)
cont>                       primary key,
cont>      order_date       automatic insert as current_timestamp,
cont>      customer_id      CUSTOMER_IDENT
cont>                       references CUSTOMERS not deferrable
cont>                       not null not deferrable);
```

Each order is a collection of components or lines. As each line is inserted into the database we will reference the ORDERS sequence so we can also store the unique value assigned to the order.

```
SQL> create table ORDER_LINES
cont>     (order_id         ORDER_IDENT
cont>                       references ORDERS,
cont>      line_number      integer
cont>                       check (line_number > 0) not deferrable
cont>                       not null not deferrable,
cont>      product_id       PRODUCT_IDENT
cont>                       references PRODUCTS not deferrable
cont>                       not null not deferrable,
cont>      quantity         integer,
cont>      discount         float default 0.0E0
cont>                      check (discount between 0.0E0 and 100.0e0)
cont>                       not deferrable);
```

[2] It is currently not possible to alter the START WITH value for a sequence, so plan accordingly.

The following simple stored procedure demonstrates how this is done. We use the name of the table as the prefix for the CURRVAL pseudo column. The identity column definition implicitly defines a sequence with the same name as the table allowing an implicit relationship between the table and its identity sequence.

```
SQL> create module NEW_ORDERS
cont>     procedure ADD_ORDER_LINE
cont>               (in :pn PRODUCT_NAME,
cont>                in :qt integer,
cont>                in :ord integer default ORDERS.currval)
cont>     comment is  'Insert one row into the ORDER_LINES table for the '
cont>     /         'specified order.  We will DEFAULT to the current '
cont>     /         'active order';
cont>     insert into ORDER_LINES
cont>         values (:ord,
cont>                   1 + (select count (*)
cont>                        from ORDER_LINES
cont>                        where order_id = :ord),
cont>                   (select product_id from PRODUCTS
cont>                     where product_name = :pn),
cont>                   :qt, DEFAULT);
cont> end module;
```

Now that **primary key** and **foreign key** constraints are defined on these **identity** columns we will need to include indices for fast constraint evaluation and join access methods. Identity columns can be referenced in sorted or hashed indices and this example database could use either.

## *Frequently Asked Questions:*

***What happens when I alter an existing table and add a new IDENTITY column?*** The IDENTITY sequence is used to provide unique values for all those previously inserted rows. Each row will be implicitly updated with this value. This is an important point to remember, since the ALTER TABLE statement may perform a lot of I/O and the updated rows might fragment if there is not sufficient space on the page to accommodate the new value in each row.

***Can I have more than one IDENTITY column per table?*** No, only one IDENTITY column is permitted per table. Rdb enforces this by creating a sequence with the same name as the table. However, you can use **create sequence** to create other sequences which can be applied to other columns. These assignments can be made implicit using the **automatic** column feature, or explicitly using the sequence within **insert** and **update** statements.

***Can I alter an existing column to have an IDENTITY attribute?*** This is not possible, only new columns can have the IDENTITY attribute.

***Can I update the identity column value if an error occurred during insert?*** Yes, these special columns act just like automatic columns. The database administrator can use the SET FLAGS 'AUTO_OVERRIDE' command to make these columns read-write for this session, then an **update** statement can be used to correct the values.

You can also reload rows (such as after a table/database restructuring) using your own application code. Use **insert** for the saved data, and include values for the **identity** column. To do this you should define RDMS$SET_FLAGS to "AUTO_OVERRIDE" prior to running your reload program. If you are using RMU Load then this logical name definition is not required, but you must use the /VIRTUAL=AUTOMATIC qualifier on Unload and Load to inform RMU that these normally read only columns should be saved and reloaded[3].

***What happens to the identity column when I use TRUNCATE TABLE?*** When all the rows are removed from the table the special identity sequence is reset to the START WITH value so that new **insert** statements will start using identity values from the start. Note that other sequences do not behave in this way, only **identity** columns.

[3] This ability is available in RMU Load only in Rdb V7.1.2 and later versions.

***Can I use NEXTVAL with an identity sequence?*** No, the NEXTVAL operation is reserved for implicit use by the INSERT statement. There is no need to reference NEXTVAL for these special sequences.

***Can I use CURRVAL with an identity sequence?*** Yes, it is useful for applications to use CURRVAL for the table's identity value. After an INSERT into the table the current value of the identity column can be fetched and used for foreign key assignments. Simply use the name of the table to reference the CURRVAL pseudo column.

***What operations can I perform on the IDENTITY sequence?*** GRANT, REVOKE, SHOW, SEQUENCE, ALTER SEQUENCE can all be performed on the identity sequence. RENAME and DROP SEQUENCE are not permitted on the identity sequence. Instead you must rename the table, or drop the identity column[4].

Oracle recommends that any user with access to the table for INSERT also be provided with SELECT privilege to the sequence created for the identity sequence.

***Can I get RMU Load to allocate unique values for the table rows?*** IDENTITY is implemented as an **automatic insert** column therefore any tool used to load data will cause the identity column to allocate new values.

***Can I create constraints and indices on the IDENTITY column?*** Yes, this is possible and desirable. If the **identity** column is a PRIMARY KEY then efficient access will be required for validation of FOREIGN KEY constraints.

***Does the SET FLAGS option SEQ_CACHE work with identity columns?*** Yes, identity columns are based on the existing sequence feature for Rdb and all the tuning and monitoring suggestions can be applied to **identity** columns.

***If the identity column is read-only do I now have to enumerate all column names in the insert statement?*** The **insert** statement allows the programmer to omit all columns names and just provide values in the **values** clause; one for each column. In Rdb 7.1 read-only columns (**automatic**, **identity** and **computed by** columns) are no longer considered part of the default column list of the insert statement.

[4] Dropping the table will implicitly drop the associated identity sequence.

***How can I tell which tables have identity attributes?*** You can perform a query on RDB$RELATIONS which describes all tables and views and join it with RDB$SEQUENCES which describes sequences, both those created with **create sequence**, and **identity**. If the RDB$RELATION_NAME is equal to any RDB$SEQUENCE_NAME then this table has an **identity** column.

```
SQL> select rdb$sequence_name
cont> from rdb$relations inner join rdb$sequences
cont> on (rdb$relation_name = rdb$sequence_name);
 RDB$SEQUENCES.RDB$SEQUENCE_NAME
 CUSTOMERS
 ORDERS
 PRODUCTS
3 rows selected
```

**Oracle Rdb**
**Guide to Using SQL: Identity Columns**
**May 2003**

**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.**