SQL | SEQUENCES

Sequence is a set of integers 1, 2, 3, … that are generated and supported by some database systems to produce unique values on demand.

* A sequence is a user defined schema bound object that generates a sequence of numeric values.
* Sequences are frequently used in many databases because many applications require each row in a table to contain a unique value and sequences provides an easy way to generate them.
* The sequence of numeric values is generated in an a**scending or descending order** at defined intervals and can be configured to restart when exceeds max\_value.

**Syntax:**

CREATE SEQUENCE sequence\_name

START WITH initial\_value

INCREMENT BY increment\_value

MINVALUE minimum value

MAXVALUE maximum value

CYCLE|NOCYCLE ;

**sequence\_name:** Name of the sequence.

**initial\_value:** starting value from where the sequence starts.

Initial\_value should be greater than or equal

to minimum value and less than equal to maximum value.

**increment\_value:** Value by which sequence will increment itself.

Increment\_value can be positive or negative.

**minimum\_value:** Minimum value of the sequence.

**maximum\_value:** Maximum value of the sequence.

**cycle:** When sequence reaches its set\_limit

it starts from beginning.

**nocycle:** An exception will be thrown

if sequence exceeds its max\_value.

Following is the sequence query creating sequence in ascending order.

* **Example 1:**

CREATE SEQUENCE sequence\_1

start with 1

increment by 1

minvalue 0

maxvalue 100

cycle;

Above query will create a sequence named *sequence\_1*.Sequence will start from 1 and will be incremented by 1 having maximum value 100. Sequence will repeat itself from start value after exceeding 100.

* **Example 2:**  
  Following is the sequence query creating sequence in descending order.

CREATE SEQUENCE sequence\_2

start with 100

increment by -1

minvalue 1

maxvalue 100

cycle;

Above query will create a sequence named *sequence\_2*.Sequence will start from 100 and should be less than or equal to maximum value and will be incremented by -1 having minimum value 1.

* **Example to use sequence :** create a table named students with columns as id and name.

CREATE TABLE students

( ID number(10),

NAME char(20));

Now insert values into table

INSERT into students VALUES(sequence\_1.nextval,'Ramesh');

INSERT into students VALUES(sequence\_1.nextval,'Suresh');

where *sequence\_1.nextval* will insert id’s in id column in a sequence as defined in sequence\_1.  
**Output:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| ID | NAME |

------------------------

| 1 | Ramesh |

| 2 | Suresh |

----------------------

OR

# Oracle / PLSQL: Sequences (Autonumber)

This Oracle tutorial explains how to **create and drop sequences** in Oracle with syntax and examples.

Description

In Oracle, you can create an autonumber field by using sequences. A sequence is an object in Oracle that is used to generate a number sequence. This can be useful when you need to create a unique number to act as a primary key.

Create Sequence

You may wish to create a sequence in Oracle to handle an autonumber field.

### Syntax

The syntax to create a sequence in Oracle is:

CREATE SEQUENCE sequence\_name

MINVALUE value

MAXVALUE value

START WITH value

INCREMENT BY value

CACHE value;

**sequence\_name**

The name of the sequence that you wish to create.

### Example

Let's look at an example of how to create a sequence in Oracle.

For example:

CREATE SEQUENCE supplier\_seq

MINVALUE 1

MAXVALUE 999999999999999999999999999

START WITH 1

INCREMENT BY 1

CACHE 20;

This would create a sequence object called supplier\_seq. The first sequence number that it would use is 1 and each subsequent number would increment by 1 (ie: 2,3,4,...}. It will cache up to 20 values for performance.

If you omit the MAXVALUE option, your sequence will automatically default to:

MAXVALUE 999999999999999999999999999

So you can simplify your CREATE SEQUENCE command as follows:

CREATE SEQUENCE supplier\_seq

MINVALUE 1

START WITH 1

INCREMENT BY 1

CACHE 20;

Now that you've created a sequence object to simulate an autonumber field, we'll cover how to retrieve a value from this sequence object. To retrieve the next value in the sequence order, you need to use nextval.

For example:

supplier\_seq.NEXTVAL;

This would retrieve the next value from supplier\_seq. The nextval statement needs to be used in a SQL statement. For example:

INSERT INTO suppliers

(supplier\_id, supplier\_name)

VALUES

(supplier\_seq.NEXTVAL, 'Kraft Foods');

This insert statement would insert a new record into the suppliers table. The supplier\_id field would be assigned the next number from the supplier\_seqsequence. The supplier\_name field would be set to Kraft Foods.

## Drop Sequence

Once you have created your sequence in Oracle, you might find that you need to remove it from the database.

### Syntax

The syntax to a drop a sequence in Oracle is:

DROP SEQUENCE sequence\_name;

**sequence\_name**

The name of the sequence that you wish to drop.

### Example

Let's look at an example of how to drop a sequence in Oracle.

For example:

DROP SEQUENCE supplier\_seq;

This example would drop the sequence called supplier\_seq.

## Frequently Asked Questions

One common question about sequences is:

**Question:** While creating a sequence, what does cache and nocacheoptions mean? For example, you could create a sequence with a cache of 20 as follows:

CREATE SEQUENCE supplier\_seq

MINVALUE 1

START WITH 1

INCREMENT BY 1

CACHE 20;

Or you could create the same sequence with the nocache option:

CREATE SEQUENCE supplier\_seq

MINVALUE 1

START WITH 1

INCREMENT BY 1

NOCACHE;

**Answer:** With respect to a sequence, the cache option specifies how many sequence values will be stored in memory for faster access.

The downside of creating a sequence with a cache is that if a system failure occurs, all cached sequence values that have **not** be used, will be "lost". This results in a "gap" in the assigned sequence values. When the system comes back up, Oracle will cache new numbers from where it left off in the sequence, ignoring the so called "lost" sequence values.

**TIP:** To recover the lost sequence values, you can always execute an ALTER SEQUENCE command to reset the counter to the correct value.

Nocache means that none of the sequence values are stored in memory. This option may sacrifice some performance, however, you should not encounter a gap in the assigned sequence values.

**Question:** How do we set the LASTVALUE value in an Oracle Sequence?

**Answer:** You can change the LASTVALUE for an Oracle sequence, by executing an ALTER SEQUENCE command.

For example, if the last value used by the Oracle sequence was 100 and you would like to reset the sequence to serve 225 as the next value. You would execute the following commands.

ALTER SEQUENCE seq\_name

INCREMENT BY 124;

SELECT seq\_name.nextval FROM dual;

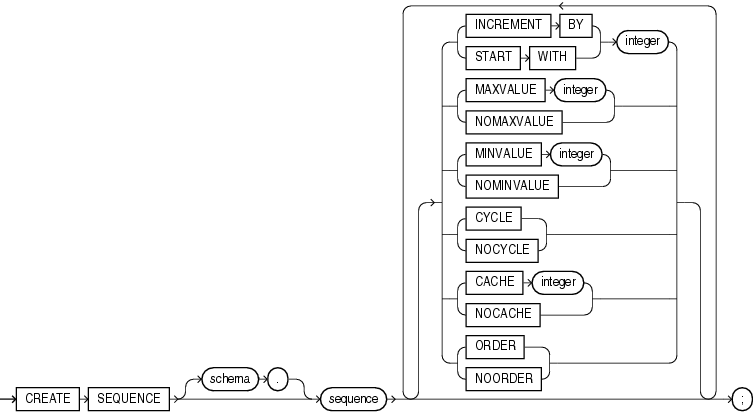
ALTER SEQUENCE seq\_name

INCREMENT BY 1;

Now, the next value to be served by the sequence will be 225.

OR

***create\_sequence*::=**

  
[Description of the illustration create\_sequence.gif](https://docs.oracle.com/cd/B28359_01/server.111/b28286/img_text/create_sequence.htm)

**Semantics**

***schema***

Specify the schema to contain the sequence. If you omit *schema*, then Oracle Database creates the sequence in your own schema.

***sequence***

Specify the name of the sequence to be created.

If you specify none of the following clauses, then you create an ascending sequence that starts with 1 and increases by 1 with no upper limit. Specifying only INCREMENT BY -1 creates a descending sequence that starts with -1 and decreases with no lower limit.

* To create a sequence that increments without bound, for ascending sequences, omit the MAXVALUEparameter or specify NOMAXVALUE. For descending sequences, omit the MINVALUE parameter or specify the NOMINVALUE.
* To create a sequence that stops at a predefined limit, for an ascending sequence, specify a value for the MAXVALUE parameter. For a descending sequence, specify a value for the MINVALUE parameter. Also specify NOCYCLE. Any attempt to generate a sequence number once the sequence has reached its limit results in an error.
* To create a sequence that restarts after reaching a predefined limit, specify values for both the MAXVALUEand MINVALUE parameters. Also specify CYCLE. If you do not specify MINVALUE, then it defaults to NOMINVALUE, which is the value 1.

**INCREMENT BY**Specify the interval between sequence numbers. This integer value can be any positive or negative integer, but it cannot be 0. This value can have 28 or fewer digits. The absolute of this value must be less than the difference of MAXVALUE and MINVALUE. If this value is negative, then the sequence descends. If the value is positive, then the sequence ascends. If you omit this clause, then the interval defaults to 1.

**START WITH**Specify the first sequence number to be generated. Use this clause to start an ascending sequence at a value greater than its minimum or to start a descending sequence at a value less than its maximum. For ascending sequences, the default value is the minimum value of the sequence. For descending sequences, the default value is the maximum value of the sequence. This integer value can have 28 or fewer digits.

**Note:**

This value is not necessarily the value to which an ascending cycling sequence cycles after reaching its maximum or minimum value.

**MAXVALUE**Specify the maximum value the sequence can generate. This integer value can have 28 or fewer digits. MAXVALUE must be equal to or greater than START WITH and must be greater than MINVALUE.

**NOMAXVALUE**Specify NOMAXVALUE to indicate a maximum value of 1027 for an ascending sequence or -1 for a descending sequence. This is the default.

**MINVALUE**Specify the minimum value of the sequence. This integer value can have 28 or fewer digits. MINVALUE must be less than or equal to START WITH and must be less than MAXVALUE.

**NOMINVALUE**Specify NOMINVALUE to indicate a minimum value of 1 for an ascending sequence or -1026 for a descending sequence. This is the default.

**CYCLE**Specify CYCLE to indicate that the sequence continues to generate values after reaching either its maximum or minimum value. After an ascending sequence reaches its maximum value, it generates its minimum value. After a descending sequence reaches its minimum, it generates its maximum value.

**NOCYCLE**Specify NOCYCLE to indicate that the sequence cannot generate more values after reaching its maximum or minimum value. This is the default.

**CACHE**Specify how many values of the sequence the database preallocates and keeps in memory for faster access. This integer value can have 28 or fewer digits. The minimum value for this parameter is 2. For sequences that cycle, this value must be less than the number of values in the cycle. You cannot cache more values than will fit in a given cycle of sequence numbers. Therefore, the maximum value allowed for CACHE must be less than the value determined by the following formula:

(CEIL (MAXVALUE - MINVALUE)) / ABS (INCREMENT)

If a system failure occurs, then all cached sequence values that have not been used in committed DML statements are lost. The potential number of lost values is equal to the value of the CACHE parameter.

**Note:**

Oracle recommends using the CACHE setting to enhance performance if you are using sequences in an Oracle Real Application Clusters environment.

**NOCACHE**Specify NOCACHE to indicate that values of the sequence are not preallocated. If you omit both CACHE and NOCACHE, then the database caches 20 sequence numbers by default.

**ORDER**Specify ORDER to guarantee that sequence numbers are generated in order of request. This clause is useful if you are using the sequence numbers as timestamps. Guaranteeing order is usually not important for sequences used to generate primary keys.

ORDER is necessary only to guarantee ordered generation if you are using Oracle Real Application Clusters. If you are using exclusive mode, then sequence numbers are always generated in order.

**NOORDER**Specify NOORDER if you do not want to guarantee sequence numbers are generated in order of request. This is the default.

**Example**

**Creating a Sequence: Example**The following statement creates the sequence customers\_seq in the sample schema oe. This sequence could be used to provide customer ID numbers when rows are added to the customers table.

CREATE SEQUENCE customers\_seq

START WITH 1000

INCREMENT BY 1

NOCACHE

NOCYCLE;

The first reference to customers\_seq.nextval returns 1000. The second returns 1001. Each subsequent reference will return a value 1 greater than the previous reference.

# Managing Sequences

This section describes aspects of managing sequences, and contains the following topics:

* [About Sequences](https://docs.oracle.com/cd/B28359_01/server.111/b28310/views002.htm#i1107995)
* [Creating Sequences](https://docs.oracle.com/cd/B28359_01/server.111/b28310/views002.htm#i1006575)
* [Altering Sequences](https://docs.oracle.com/cd/B28359_01/server.111/b28310/views002.htm#i1006623)
* [Using Sequences](https://docs.oracle.com/cd/B28359_01/server.111/b28310/views002.htm#i1007824)
* [Dropping Sequences](https://docs.oracle.com/cd/B28359_01/server.111/b28310/views002.htm#i1006644)

## About Sequences

**Sequences** are database objects from which multiple users can generate unique integers. The sequence generator generates sequential numbers, which can help to generate unique primary keys automatically, and to coordinate keys across multiple rows or tables.

Without sequences, sequential values can only be produced programmatically. A new primary key value can be obtained by selecting the most recently produced value and incrementing it. This method requires a lock during the transaction and causes multiple users to wait for the next value of the primary key; this waiting is known as **serialization**. If developers have such constructs in applications, then you should encourage the developers to replace them with access to sequences. Sequences eliminate serialization and improve the concurrency of an application.

**See Also:**

[*Oracle Database Concepts*](https://docs.oracle.com/cd/B28359_01/server.111/b28318/schema.htm#CNCPT611) for an overview of sequences

## Creating Sequences

To create a sequence in your schema, you must have the CREATE SEQUENCE system privilege. To create a sequence in another user's schema, you must have the CREATE ANY SEQUENCE privilege.

Create a sequence using the CREATE SEQUENCE statement. For example, the following statement creates a sequence used to generate employee numbers for the empno column of the emp table:

CREATE SEQUENCE emp\_sequence

INCREMENT BY 1

START WITH 1

NOMAXVALUE

NOCYCLE

CACHE 10;

Notice that several parameters can be specified to control the function of sequences. You can use these parameters to indicate whether the sequence is ascending or descending, the starting point of the sequence, the minimum and maximum values, and the interval between sequence values. The NOCYCLE option indicates that the sequence cannot generate more values after reaching its maximum or minimum value.

The CACHE clause preallocates a set of sequence numbers and keeps them in memory so that sequence numbers can be accessed faster. When the last of the sequence numbers in the cache has been used, the database reads another set of numbers into the cache.

The database might skip sequence numbers if you choose to cache a set of sequence numbers. For example, when an instance abnormally shuts down (for example, when an instance failure occurs or a SHUTDOWN ABORTstatement is issued), sequence numbers that have been cached but not used are lost. Also, sequence numbers that have been used but not saved are lost as well. The database might also skip cached sequence numbers after an export and import. See [*Oracle Database Utilities*](https://docs.oracle.com/cd/B28359_01/server.111/b28319/toc.htm) for details.

* [*Oracle Database SQL Language Reference*](https://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_6015.htm#SQLRF01314) for the CREATE SEQUENCE statement syntax
* [*Oracle Real Application Clusters Deployment and Performance Guide*](http://www.oracle.com/pls/topic/lookup?ctx=db111&id=RACDP) for information about how caching sequence numbers improves performance in an Oracle Real Application Clusters environment

## Altering Sequences

To alter a sequence, your schema must contain the sequence, or you must have the ALTER ANY SEQUENCEsystem privilege. You can alter a sequence to change any of the parameters that define how it generates sequence numbers except the sequence starting number. To change the starting point of a sequence, drop the sequence and then re-create it.

Alter a sequence using the ALTER SEQUENCE statement. For example, the following statement alters the emp\_sequence:

ALTER SEQUENCE emp\_sequence

INCREMENT BY 10

MAXVALUE 10000

CYCLE

CACHE 20;

**See Also:**

[*Oracle Database SQL Language Reference*](https://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_2011.htm#SQLRF00817) for syntax and additional information about the ALTER SEQUENCE statement

## Using Sequences

To use a sequence, your schema must contain the sequence or you must have been granted the SELECT object privilege for another user's sequence. Once a sequence is defined, it can be accessed and incremented by multiple users (who have SELECT object privilege for the sequence containing the sequence) with no waiting. The database does not wait for a transaction that has incremented a sequence to complete before that sequence can be incremented again.

The examples outlined in the following sections show how sequences can be used in master/detail table relationships. Assume an order entry system is partially comprised of two tables, orders\_tab (master table) and line\_items\_tab (detail table), that hold information about customer orders. A sequence named order\_seq is defined by the following statement:

CREATE SEQUENCE Order\_seq

START WITH 1

INCREMENT BY 1

NOMAXVALUE

NOCYCLE

CACHE 20;

### Referencing a Sequence

A sequence is referenced in SQL statements with the NEXTVAL and CURRVAL pseudocolumns; each new sequence number is generated by a reference to the sequence pseudocolumn NEXTVAL, while the current sequence number can be repeatedly referenced using the pseudo-column CURRVAL.

NEXTVAL and CURRVAL are not reserved words or keywords and can be used as pseudocolumn names in SQL statements such as SELECT, INSERT, or UPDATE.

#### Generating Sequence Numbers with NEXTVAL

To generate and use a sequence number, reference *seq\_name*.NEXTVAL. For example, assume a customer places an order. The sequence number can be referenced in a values list. For example:

INSERT INTO Orders\_tab (Orderno, Custno)

VALUES (Order\_seq.NEXTVAL, 1032);

Or, the sequence number can be referenced in the SET clause of an UPDATE statement. For example:

UPDATE Orders\_tab

SET Orderno = Order\_seq.NEXTVAL

WHERE Orderno = 10112;

The sequence number can also be referenced outermost SELECT of a query or subquery. For example:

SELECT Order\_seq.NEXTVAL FROM dual;

As defined, the first reference to order\_seq.NEXTVAL returns the value 1. Each subsequent statement that references order\_seq.NEXTVAL generates the next sequence number (2, 3, 4,. . .). The pseudo-column NEXTVAL can be used to generate as many new sequence numbers as necessary. However, only a single sequence number can be generated for each row. In other words, if NEXTVAL is referenced more than once in a single statement, then the first reference generates the next number, and all subsequent references in the statement return the same number.

Once a sequence number is generated, the sequence number is available only to the session that generated the number. Independent of transactions committing or rolling back, other users referencing order\_seq.NEXTVALobtain unique values. If two users are accessing the same sequence concurrently, then the sequence numbers each user receives might have gaps because sequence numbers are also being generated by the other user.

#### Using Sequence Numbers with CURRVAL

To use or refer to the current sequence value of your session, reference *seq\_name*.CURRVAL. CURRVAL can only be used if *seq\_name*.NEXTVAL has been referenced in the current user session (in the current or a previous transaction). CURRVAL can be referenced as many times as necessary, including multiple times within the same statement. The next sequence number is not generated until NEXTVAL is referenced. Continuing with the previous example, you would finish placing the customer's order by inserting the line items for the order:

INSERT INTO Line\_items\_tab (Orderno, Partno, Quantity)

VALUES (Order\_seq.CURRVAL, 20321, 3);

INSERT INTO Line\_items\_tab (Orderno, Partno, Quantity)

VALUES (Order\_seq.CURRVAL, 29374, 1);

Assuming the INSERT statement given in the previous section generated a new sequence number of 347, both rows inserted by the statements in this section insert rows with order numbers of 347.

#### Uses and Restrictions of NEXTVAL and CURRVAL

CURRVAL and NEXTVAL can be used in the following places:

* VALUES clause of INSERT statements
* The SELECT list of a SELECT statement
* The SET clause of an UPDATE statement

CURRVAL and NEXTVAL cannot be used in these places:

* A subquery
* A view query or materialized view query
* A SELECT statement with the DISTINCT operator
* A SELECT statement with a GROUP BY or ORDER BY clause
* A SELECT statement that is combined with another SELECT statement with the UNION, INTERSECT, or MINUS set operator
* The WHERE clause of a SELECT statement
* DEFAULT value of a column in a CREATE TABLE or ALTER TABLE statement
* The condition of a CHECK constraint

### Caching Sequence Numbers

Sequence numbers can be kept in the sequence cache in the System Global Area (SGA). Sequence numbers can be accessed more quickly in the sequence cache than they can be read from disk.

The sequence cache consists of entries. Each entry can hold many sequence numbers for a single sequence.

Follow these guidelines for fast access to all sequence numbers:

* Be sure the sequence cache can hold all the sequences used concurrently by your applications.
* Increase the number of values for each sequence held in the sequence cache.

#### The Number of Entries in the Sequence Cache

When an application accesses a sequence in the sequence cache, the sequence numbers are read quickly. However, if an application accesses a sequence that is not in the cache, then the sequence must be read from disk to the cache before the sequence numbers are used.

If your applications use many sequences concurrently, then your sequence cache might not be large enough to hold all the sequences. In this case, access to sequence numbers might often require disk reads. For fast access to all sequences, be sure your cache has enough entries to hold all the sequences used concurrently by your applications.

#### The Number of Values in Each Sequence Cache Entry

When a sequence is read into the sequence cache, sequence values are generated and stored in a cache entry. These values can then be accessed quickly. The number of sequence values stored in the cache is determined by the CACHE parameter in the CREATE SEQUENCE statement. The default value for this parameter is 20.

This CREATE SEQUENCE statement creates the seq2 sequence so that 50 values of the sequence are stored in the SEQUENCE cache:

CREATE SEQUENCE seq2

CACHE 50**;**

The first 50 values of seq2 can then be read from the cache. When the 51st value is accessed, the next 50 values will be read from disk.

Choosing a high value for CACHE lets you access more successive sequence numbers with fewer reads from disk to the sequence cache. However, if there is an instance failure, then all sequence values in the cache are lost. Cached sequence numbers also could be skipped after an export and import if transactions continue to access the sequence numbers while the export is running.

If you use the NOCACHE option in the CREATE SEQUENCE statement, then the values of the sequence are not stored in the sequence cache. In this case, every access to the sequence requires a disk read. Such disk reads slow access to the sequence. This CREATE SEQUENCE statement creates the SEQ3 sequence so that its values are never stored in the cache:

CREATE SEQUENCE seq3

NOCACHE;

## Dropping Sequences

You can drop any sequence in your schema. To drop a sequence in another schema, you must have the DROP ANY SEQUENCE system privilege. If a sequence is no longer required, you can drop the sequence using the DROP SEQUENCE statement. For example, the following statement drops the order\_seq sequence:

DROP SEQUENCE order\_seq;

When a sequence is dropped, its definition is removed from the data dictionary. Any synonyms for the sequence remain, but return an error when referenced.