Sequences

Tuesday, October 17, 2023 6:55 PM

Sequences:

- **Sequence is a DB Object.**
- **Sequence is used to generate sequential integers.**

TransactionID	OrderID	studentid
123456	5678	1001
123457	5679	1002
123458	5680	1003
	5681	1004

Syntax to create the sequence:

```
CREATE SEQUENCE <sequence_name>
[START WITH <value>]
[INCREMENT BY <value>]
[MINVALUE <value>]
[MAXVALUE <value>]
[CYCLE / NOCYCLE]
[CACHE <size> / NOCACHE];
```

Example:

```
CREATE SEQUENCE s1;
Output:
sequence created
```

Clause	Default value
START WITH	1
INCREMENT BY	1
MINVALUE	1
MAXVALUE	10 power 28
CYCLE	NOCYCLE
CACHE	20

user_Sequences:

- **it is a system table.**
- **It maintains all sequences information.**

```
SELECT * FROM user_sequences;
```

Sequence pseudo columns:

Sequence pseudo columns. they are:

- ○ **NEXTVAL**
- ○ **CURRVAL**

NEXTVAL	**returns next value in the sequence**
CURRVAL	**returns current value in the sequence**

Syntax:

sequence_name.sequence_pseudo_coulmn

Example:

```
s1.nextval
s1.currval
```

Note:

to give permission to the user for creating sequence:

```
login as dba:
    GRANT create sequence
    TO c##batch6pm;
```

```
SELECT s1.nextval FROM dual;
Output: 1

SQL> SELECT s1.nextval FROM dual;
Output: 2

SQL> SELECT s1.nextval FROM dual;
```

Output: 3

Example:
create employee table with following structure.
generate empnos automatically using sequence:

EMPLOYEE

EMPNO	ENAME	SAL
1	A	5000
2	B	3000
3	C	7000
4	D	6000
5	E	9000

```
CREATE SEQUENCE s2
START WITH 1
INCREMENT BY 1
MINVALUE 1
MAXVALUE 5;

CREATE TABLE employee
(
empno NUMBER(4),
ename VARCHAR2(10),
sal NUMBER(8,2)
);

INSERT INTO employee VALUES(s2.nextval,  '&ename', &sal);
```

Output:

```
enter value for ename: A
enter value for sal: 5000

/
enter value for ename: B
enter value for sal: ..

/
enter value for ename: C
enter value for sal: ..

/
enter value for ename: D
enter value for sal: ..

/
enter value for ename: E
enter value for sal: 5000

/
enter value for ename: F
enter value for sal: ..
ERROR: reached max value
```

Assignment:

create course table with following structure.
generate course ids using sequence:

COURSE

CID	CNAME
10	JAVA
20	PYTHON
30	C#
40	HTML
..	
100	ORACLE

```
CREATE SEQUENCE s3
START WITH 10
INCREMENT BY 10
MINVALUE 10
MAXVALUE 100;
```

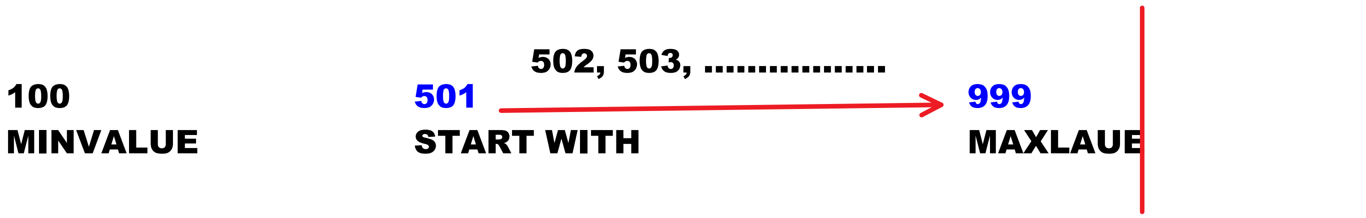
START WITH	is used to specify starting value in sequence
INCREMENT BY	is used to specify step value
MINVALUE	is used to specify min value in sequence
MAXVALUE	is used to specify max value in the sequence

Cycle / NoCycle:

- default one is "NoCycle"

NOCYCLE:

```
CREATE SEQUENCE s4
START WITH 501
INCREMENT BY 1
MINVALUE 100
MAXVALUE 999
NOCYCLE;
```



stops generating next value after reaching max value

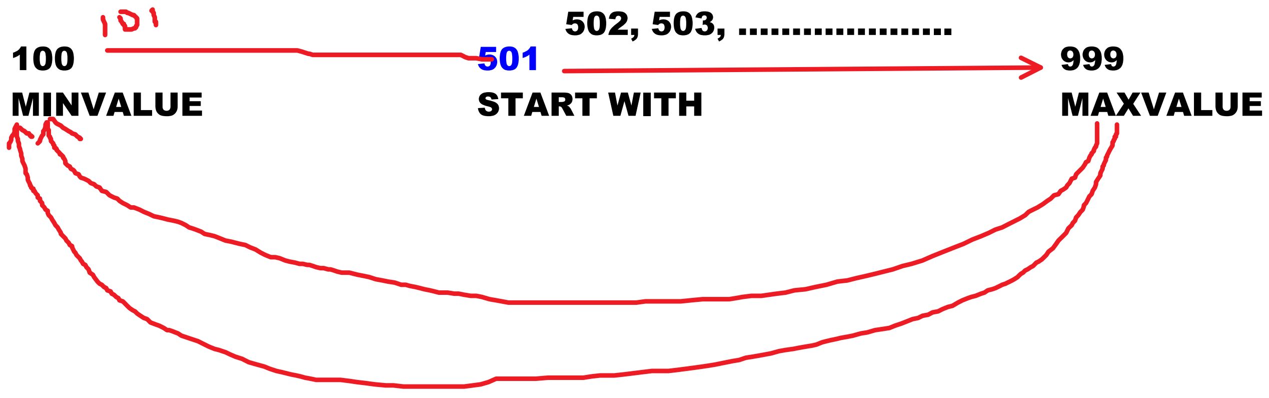
If sequence is created with NOCYCLE,

- sequence starts from START WITH value.
- generates next value up to max value.
- After reaching max value, it will be stopped.

CYCLE:

```
CREATE SEQUENCE s4
START WITH 501
INCREMENT BY 1
MINVALUE 100
MAXVALUE 999
```

CYCLE;



If sequence is created with CYCLE,

- sequence starts from START WITH value.
- generates next value up to max value.
- After reaching max value, it will be reset to minvalue.

```
CREATE SEQUENCE s5
START WITH 25
INCREMENT BY 1
MINVALUE 1
MAXVALUE 30
NOCYCLE;
```

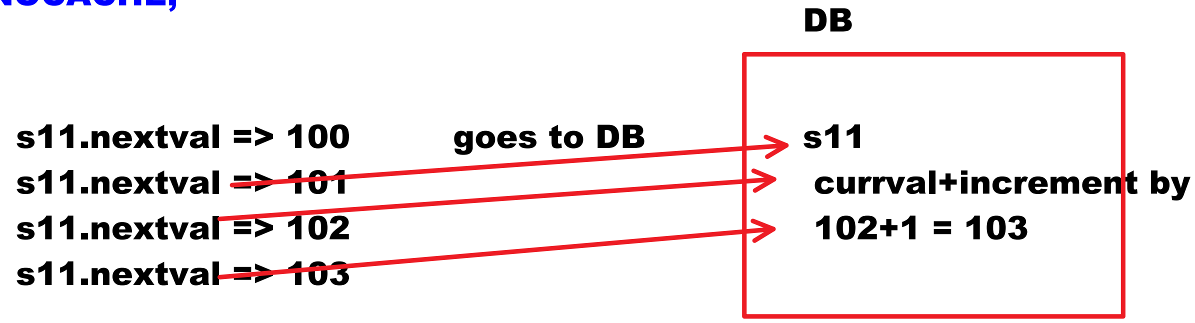
```
CREATE SEQUENCE s6
START WITH 25
INCREMENT BY 1
MINVALUE 1
MAXVALUE 30
CYCLE;
```

CACHE <size> / NOCACHE:

NOCACHE:

```
CREATE SEQUENCE s11
START WITH 100
INCREMENT BY 1
MINVALUE 100
```

```
MAXVALUE 999
NOCACHE;
```



DB

s11.nextval => 100 goes to DB
s11.nextval => 101
s11.nextval => 102
s11.nextval => 103

s11
currval+increment by
102+1 = 103

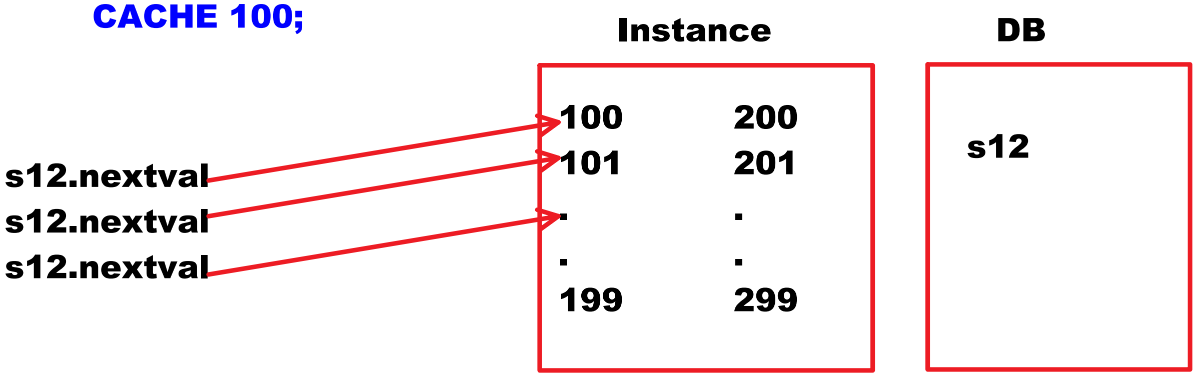
If sequence is created with NOCACHE,

- for every sequence call, it goes to DB, identifies current value, adds increment by value and returns the value to sequence call.

- If no of travels to DB are increased then performance will be degrade. To improve the performance we use CACHE.

CACHE <size>:

```
CREATE SEQUENCE s12
START WITH 100
INCREMENT BY 1
MINVALUE 100
MAXVALUE 999
CACHE 100;
```



Instance

DB

s12.nextval
s12.nextval
s12.nextval

100 200
101 201
. .
. .
199 299

s12

If sequence is CACHE size 100,

- When sequence is created 100 values will be preloaded

into the INSTANCE.

- **For every sequence call it collects sequential number from INSTANCE.**
- **Every time it will not go to DB, every time it will not calculate. So, performance will be improved.**

NOTE:

- **CACHE purpose is improving performance of generating sequential numbers.**
- **In case of CYCLE,**
  Always cache size must be less than 1 cycle.

```
CREATE SEQUENCE s13
START WITH 1
MINVALUE 1
MAXVALUE 10
NOCYCLE;
```

Output:
Sequence created.

```
CREATE SEQUENCE s14
START WITH 1
MINVALUE 1
MAXVALUE 10
CYCLE;
```

Output:
ERROR:
CACHE size is > 1 cycle

default cache size
20 > 10 => ERROR

```
CREATE SEQUENCE s14
START WITH 1
MINVALUE 1
MAXVALUE 10
CYCLE
CACHE 10;
```

Output:
Sequence created

Can we call a sequence from UPDATE command?
YES.

Make empnos as sequential numbers in emp table.
start numbering from 1001:

EMP

EMPNO	ENAME
~~7369~~ 1001	SMITH
~~7499~~ 1002	ALLEN
~~7521~~ 1003	WARD

```
CREATE SEQUENCE s15
START WITH 1001
INCREMENT BY 1
MINVALUE 1001
MAXVALUE 9999;
```

```
UPDATE emp
SET empno=s15.nextval;
```

Can we call a sequence from CREATE command?
YES.
From ORACLE 12C version onwards we can call a sequence from CREATE command.

Create student table. generate sequential numbers using sequence. call the sequence from create command:

STUDENT

sid	sname

1	A
2	B
3	C

```
CREATE SEQUENCE s16
START WITH 1
INCREMENT BY 1
MINVALUE 1
MAXVALUE 9999;

CREATE TABLE student
(
sid NUMBER(4) DEFAULT s16.nextval,
sname VARCHAR2(10)
);

INSERT INTO student(sname) VALUES('A');
INSERT INTO student(sname) VALUES('B');
INSERT INTO student(sname) VALUES('C');
COMMIT;

SELECT * FROM student;
```

Output:

```
sid
-----
1
2
3
```

From ORACLE 12C version onwards,
sequential numbers can be generated using 2 ways.
they are:

- **using SEQUENCE**
- **using IDENTITY**

generating sequential numbers using identity:

Syntax:

```
<column_name> <data_type> generated always as
identity(<sequence_options>)
```

Example:

```
CREATE TABLE student1
(
sid number(4) generated always as identity,
sname VARCHAR2(10)
);

INSERT INTO student1(sname) VALUES('A');
INSERT INTO student1(sname) VALUES('B');
INSERT INTO student1(sname) VALUES('C');
COMMIT;

SELECT * FROM student1;
Output:
sid
----
1
2
3
```

what is the difference between sequence and identity?

- **no difference is there.**
- **when we use IDENTITY, implicitly sequence will be created.**

```
SELECT * FROM user_sequences;
```

Example:

COURSE

CID	CNAME
10	C
20	JAVA

30	C#
40	PYTHON
..	
100	ORACLE

```
CREATE TABLE course
(
cid NUMBER(2) generated always as identity(START WITH
10 INCREMENT BY 10 MINVALUE 10 MAXVALUE 100),
cname VARCHAR2(10)
);

SQL> insert into course(cname) values('C');

1 row created.

SQL> insert into course(cname) values('C++');

1 row created.

SQL> insert into course(cname) values('JAVA');

1 row created.

SQL> commit;

Commit complete.

SQL> select * from course;

       CID CNAME
---------- ----------
        10 C
        20 C++
        30 JAVA
```

Altering Sequence:

```
CREATE SEQUENCE s17
```

```
START WITH 1
INCREMENT BY 1
MINVALUE 1
MAXVALUE 5;

SELECT s17.nextval FROM dual;  --1
SELECT s17.nextval FROM dual;  --2
SELECT s17.nextval FROM dual;  --3
SELECT s17.nextval FROM dual;  --4
SELECT s17.nextval FROM dual;  --5
SELECT s17.nextval FROM dual;  --ERROR


ALTER SEQUENCE s17 MAXVALUE 10;

SELECT s17.nextval FROM dual;  --6
SELECT s17.nextval FROM dual;  --7
SELECT s17.nextval FROM dual;  --8
SELECT s17.nextval FROM dual;  --9
SELECT s17.nextval FROM dual;  --10
SELECT s17.nextval FROM dual;  --ERROR

ALTER SEQUENCE s17 MAXVALUE 20
INCREMENT BY 2;
```

generate sequential numbers in descending order from 50 to 1:

```
50
49
48
47
.
.
1
```

```
CREATE SEQUENCE s18
START WITH 50
INCREMENT BY -1
MINVALUE 1
MAXVALUE 50;

SELECT s18.nextval from dual; --50
SELECT s18.nextval from dual; --49
SELECT s18.nextval from dual; --48
```

VIEWS

Wednesday, October 18, 2023 7:26 PM

Virtual => not real

VIEWS:

- **VIEW is a DB Object.**
- **VIEW is virtual table. Virtual Table means, It does not contain physical data. It does not occupy memory.**
- **A VIEW holds SELECT query.**
- **When we retrieve data through VIEW, it runs SELECT query which is in that VIEW.**
- **A view will be created based on table.**
- **The table on which view is created is called "Base Table".**
- **If we make any DMLs through VIEW, these will be applied to base table. Because, VIEW does not contain any physical data.**

Syntax to create the view:

```
CREATE VIEW <view_name>
AS
<SELECT query>;
```

Note:

to give permission to create the view:

login as DBA:
username: system
password: naresh

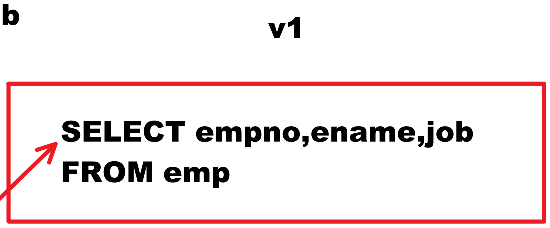
```
GRANT create view TO c##batch6pm;
```

Example:

Login as USER:

```
CREATE VIEW v1
AS
SELECT empno,ename,job
FROM emp;
```

Output:
View created.



v1

```
SELECT empno,ename,job
FROM emp
```

```
SELECT * FROM v1;    --this query will be rewritten as following:
```

Output:

empno	ename	job
..	..	..
..	...	..

```
SELECT * FROM (SELECT empno,ename,job
FROM emp)
```

Advantages:

- provides security.
- reduces complexity and simplifies the queries.

Security can be implemented at 3 levels

DB level security	SCHEMA [user]
Table Level Security	GRANT, REVOKE
Data Level Security	VIEW

VIEW can be used to implement Data Level Security.

Data Level Security can be applied at 2 levels. They are:

- Column Level Security
- Row Level Security

Implementing Column Level Security:

EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO

V2

EMPNO	ENAME	JOB

on this view v2 we give permission to another user

c#batch6pm:

```
CREATE VIEW v2
AS
SELECT empno,ename,job
FROM emp;

GRANT all ON v2 TO c##userA;
```

c##userA:

```
SELECT * FROM c##batch6pm.v2;

empno     ename      job
---------------------------------------
```

Here for remaining 5 columns we are providing security

c##batch6pm	c##userA

CREATE VIEW v2
AS
SELECT empno,ename,job
FROM emp;	

GRANT all ON v2 TO c##userA;	
	SELECT * FROM c##batch6pm.v2; Output: empno ename job -------------------------------------

	INSERT INTO c##batch6pm.v2
	VAUES(1001,'A','CLERK');
	Output:
	1 row created.

	SELECT * FROM c##batch6pm.v2; Output: empno ename job ---------------------------------- 1001 A CLERK
SELECT * FROM v2; --it will not display 1001 --because not committed	
	COMMIT;
SELECT * FROM v2; --it will display 1001	

Implementing Row Level Security:

EMP

EMPNO	ENAME	JOB	SAL	DEPTNO
1001	A			10
1002	B			10
1003	C			10
1004	D			20
1005	E			20
1006	F			30
1007	G			30

```
CREATE VIEW v3
AS
SELECT * FROM emp
WHERE deptno=10;
```

give permission
on v3
to
10th dept manager

```
CREATE VIEW v3
AS
SELECT * FROM emp
WHERE deptno=10;
```

v3

```
SELECT * FROM emp
WHERE deptno=10;
```

```
SELECT * FROM v3;
--displays 10th dept records only
--this view cannot display other than 10 dept records.
```

--But, we can insert other dept records through this view. this is problem. To avoid this problem we use **"WITH CHECK OPTION" clause.**

```
INSERT INTO v3(empno,ename,deptno)
VALUES(9002,'BB',30);
```

Output:
1 row created.

```
SELECT * FROM v3;
--does not display 30th dept record 9002
--it displays 10th dept records only
```

WITH CHECK OPTION clause:

```
CREATE VIEW v4
```

```
AS
SELECT * FROM emp
WHERE deptno=10 WITH CHECK OPTION;

INSERT INTO v4(empno,ename,deptno)
VALUES(9003,'C',30);
```

Output:

```
ERROR: view WITH CHECK OPTION where-clause violation
```

- **WITH CHECK OPTION clause is used to restrict the user from entering the records which cannot be displayed by the view later.**

- **WITH CHECK OPTION's WHERE condition violated records cannot be accepted.**

Types of Views:

2 Types:

- **Simple View**
- **Complex View**

Simple View:

- **If view created based on one table then it is called "Simple View".**

- **Simple View can be also called as "Updatable View".**

- **We can perform DML operations through Simple View.**

Complex View:

- **If view created based on multiple tables then it is called "Complex View".**

- **If view created based on joins / sub queries / GROUP BY / HAVING / Aggregate Functions / Expressions / Set Operators then it is called "Complex View".**

- **It can be also called as "Read-Only View".**

- **We cannot perform DML operations through Complex View.**

Example:

```
CREATE VIEW v6
AS
SELECT deptno, sum(sal) AS sum_of_Sal
FROM emp
GROUP BY deptno;

SELECT * FROM v6;
--displays dept wise sum of salaries


CREATE VIEW v7
AS
SELECT <10 tables column list>
FROm <10 table names>
WHERE 9 join conditions;

SELECT * FROM v7;


CREATE VIEW v8
AS
SELECT e.ename, d.dname
FROM emp e, dept d
WHERE e.deptno=d.deptno;

SELECT * FROM v8;
```

user_views:

- it is a system table
- it maintains all vies information.

to see all views list:

```
SELECT view_name, text
FROM user_views;
```

dropping views:

Syntax:

```
DROP VIEW <view_name>;
```

Example:

```
DROP VIEW v7;
```

Can we create a view based on another view?
YES.

```
CREATE VIEW v8
AS
SELECT empno,ename,job,sal FROM emp;

CREATE VIEW v9
AS
SELECT ename,sal FROM v8;
```

Can we create a view without base table?
NO.

But, there is one concept. that is: FORCE VIEW

FORCE VIEW will be created without base table.
This view will not work until base table is created.

creating view first, base table next then go for FORCE VIEW.

```
CREATE VIEW v10
AS
SELECT * FROM nareshit;
Output:
View created with compilation errors

SELECT * FROM v10;
Output:
View has some errors

CREATE TABLE nareshit(f1 int);

INSERT INTO nareshit VALUES(1001);
INSERT INTO nareshit VALUES(1002);
```

```
COMMIT;

SELECT FROM v10;
Output:
f1
------
1001
1002
```

VIEW:

virtual table => no physical data

advantages:

- security
- reduces complexity

2 types:

simple view   => based on 1 table
DMLs can be performed through simple view

complex view => based on multiple tables
DMLs cannot be performed through simple view

INDEXES

Thursday, October 19, 2023 7:25 PM

INDEXES:

INDEX GOAL:
improving performance of data retrieval

sal>10000

BOOK
INDEX

CHAPTER NAME	PGNO
DDL commands	10
DML commands	20
CLAUSES	50
JOINS	80

ORACLE INDEX

VALUE	ROWID
9000	*
10000	* *
12000	*
13000	* * *

INDEXES:

- **INDEX is a DB Object.**
- **INDEX is used to improve the performance of data retrieval.**
- **With BOOK INDEX we can refer a chapter quickly.**
  In the same way, with ORACLE INDEX we can retrieve the record quickly.
- **INDEX will be created on columns which we use frequently in WHERE clause.**

  Syntax:
  ```
  CREATE INDEX <indexname> ON <table_name>(<column_list>);
  ```

  When we submit a SELECT command to ORACLE,
  it may perform TABLE SCAN or INDEX SCAN.

  If INDEX is not created, it performs TABLE SCAN

If INDEX is created, it performs INDEX SCAN

Example:

```
SET AUTOTRACE ON EXPLAIN  -- to see execution plan

SELECT ename,sal FROM emp WHERE sal>2500;   --Table Scan

CREATE INDEX i1 ON emp(sal);
Output: index created

SELECT ename,sal FROM emp WHERE sal>2500;   --Index Scan
```

Note:
Index scan is faster than Table Scan

Types of Indexes:

2 types:

- **B-Tree Index**
  - **Simple Index**
  - **Composite Index**
  - **Function-Based Index**
  - **Unique Index**
- **Bitmap Index**