16-1 Working With Sequences

This lesson covers the following objectives:
• List at least three useful characteristics of a sequence
• Write and execute a SQL statement that creates a sequence
• Query the data dictionary using USER_SEQUENCES to confirm a sequence definition
• Apply the rules for using NEXTVAL to generate sequential unique numbers in a table
• List the advantages and disadvantages of caching sequence values
• Name three reasons why gaps can occur in a sequence
• The numbering process is handled through a database object called a SEQUENCE.

The Sequence Object

• You already know how to create two kinds of database objects, the TABLE and the VIEW.
• A third database object is the SEQUENCE.
• A SEQUENCE is a shareable object used to automatically generate unique numbers.
• Because it is a shareable object, multiple users can access it.
• Typically, sequences are used to create a primary-key value.
• As you'll recall, primary keys must be unique for each row. The sequence is generated and incremented (or decremented) by an internal Oracle routine.
• This object is a time-saver for you because it reduces the amount of code you need to write.
• Sequence numbers are stored and generated independently of tables.
• Therefore, the same sequence can be used for multiple tables.
• To create a SEQUENCE: Sequence Syntax

```
CREATE SEQUENCE sequence
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];
```

sequence	is the name of the sequence generator (object)
INCREMENT BY n	specifies the interval between sequence numbers where n is an integer (If this clause is omitted, the sequence increments by 1.)
START WITH n	specifies the first sequence number to be generated (If this clause is omitted, the sequence starts with 1.)
MAXVALUE n	specifies the maximum value the sequence can generate
NOMAXVALUE	specifies a maximum value of 10^27 for an ascending sequence and -1 for a descending sequence (default)
MINVALUE n	specifies the minimum sequence value
NOMINVALUE	specifies a minimum value of 1 for an ascending sequence and –(10^26) for a descending sequence (default)
CYCLE | NOCYCLE	specifies whether the sequence continues to generate values after reaching its maximum or minimum value (NOCYCLE is the default option.)
CACHE n | NOCACHE	specifies how many values the Oracle server pre-allocates and keeps in memory. (By default, the Oracle server caches 20 values.) If the system crashes, the values are lost.

Creating a Sequence

• In the SEQUENCE created for the London Marathon runners, the numbers will increment by 1, starting with the number 1.
• In this case, beginning the sequence with 1 is probably the best starting point.

```
CREATE SEQUENCE runner_id_seq
```

```
INCREMENT BY 1
START WITH 1
MAXVALUE 50000
NOCACHE
NOCYCLE;
```

• It is a tradition that the best runner in the elite group wears number 1.
• For other situations, such as department IDs and employee IDs, the starting number may be assigned differently.
• Because there will be at least 30,000 runners, the sequence's maximum value was set well above the expected number of runners.
• The NOCACHE option prevents values in the SEQUENCE from being cached in memory, which in the event of system failure prevents numbers pre-allocated and held in memory from being lost.
• The NOCYCLE option prevents the numbering from starting over at 1 if the value 50,000 is exceeded.
• Don't use the CYCLE option if the sequence is used to generate primary-key values unless there is a reliable mechanism that deletes old rows faster than new ones are added.

Confirming Sequences

• To verify that a sequence was created, query the USER_OBJECTS data dictionary.
• To see all of the SEQUENCE settings, query the USER_SEQUENCES data dictionary as shown below.
• List the value names in the SELECT statement as shown below.

```
SELECT sequence_name, min_value, max_value, increment_by, last_number
FROM user_sequences;
```

• If NOCACHE is specified, the last_number column in the query displays the next available sequence number.
• If CACHE is specified, the last_number column displays the next available number in the sequence which has not been cached into memory.

NEXTVAL and CURRVAL Pseudocolumns

• The NEXTVAL pseudocolumn is used to extract successive sequence numbers from a specified sequence.
• You must qualify NEXTVAL with the sequence name.
• When you reference sequence.NEXTVAL, a new sequence number is generated and the current sequence number is placed in CURRVAL.
• The example below inserts a new department in the DEPARTMENTS table.
• It uses the DEPARTMENTS_SEQ sequence for generating a new department number as follows:

```
INSERT INTO departments (department_id, department_name, location_id)
VALUES (departments_seq.NEXTVAL, 'Support', 2500);
```

• Suppose now you want to hire employees to staff the new department.
• The INSERT statement to be executed for all new employees can include the following code:

```
INSERT INTO employees(employee_id, department_id, ...)
VALUES (employees_seq.NEXTVAL, dept_deptid_seq .CURRVAL, ...);
```

•Note: The preceding example assumes that a sequence called EMPLOYEES_SEQ has already been created for generating new employee numbers.

• The CURRVAL pseudocolumn in the example below is used to refer to a sequence number that the current user has just generated.
• NEXTVAL must be used to generate a sequence number in the current user's session before CURRVAL can be referenced.
• You must qualify CURRVAL with the sequence name.

• When sequence.CURRVAL is referenced, the last value generated by that user's process is returned.

```
INSERT INTO employees (employee_id, department_id, ...)
VALUES (employees_seq.NEXTVAL, dept_deptid_seq.CURRVAL, ...);
```

Using a Sequence

• After you create a sequence, it generates sequential numbers for use in your tables. Reference the sequence values by using the NEXTVAL and CURRVAL pseudocolumns.
• You can use NEXTVAL and CURRVAL in the following contexts:
- The SELECT list of a SELECT statement that is not part of a subquery
- The SELECT list of a subquery in an INSERT statement
- The VALUES clause of an INSERT statement
- The SET clause of an UPDATE statement

• You cannot use NEXTVAL and CURRVAL in the following contexts:
- The SELECT list of a view
- A SELECT statement with the DISTINCT keyword
- A SELECT statement with GROUP BY, HAVING, or ORDER BY clauses
- A subquery in a SELECT, DELETE, or UPDATE statement
- The DEFAULT expression in a CREATE TABLE or ALTER TABLE statement

To continue our London Marathon example, a table was created for the runners:

```
CREATE TABLE runners
        (runner_id NUMBER(6,0) CONSTRAINT runners_id_pk PRIMARY KEY,
        first_name VARCHAR2(30),
        last_name VARCHAR2(30));
```

• We then create the sequence that will generate values for the runner_id primary key column.

```
CREATE SEQUENCE runner_id_seq
INCREMENT BY 1
START WITH 1
MAXVALUE 50000
NOCACHE
NOCYCLE;
```

• Using the following syntax would allow new participants to be inserted into the runners table.
• The runner's identification number would be generated by retrieving the NEXTVAL from the sequence.

```
INSERT INTO runners
        (runner_id, first_name, last_name)
VALUES (runner_id_seq.NEXTVAL, 'Joanne', 'Everely');
        INSERT INTO runners
        (runner_id, first_name, last_name)
VALUES (runner_id_seq.NEXTVAL, 'Adam', 'Curtis');
```

• To confirm the sequence worked correctly, we query the table:

```
SELECT runner_id, first_name, last_name
FROM runners;
```

RUNNER_ID	FIRST_NAME	LAST_NAME
1	Joanne	Everely
2	Adam	Curtis

• To view the current value for the runners_id_seq, CURRVAL is used.
• Note the use of the DUAL table in this example.
• Oracle Application Express will not execute this query, but you should understand how this works.

SELECT runner_id_seq.CURRVAL
FROM dual;
• Cache sequences in memory provide faster access to sequence values.
• The cache is populated the first time you refer to the sequence.
• Each request for the next sequence value is retrieved from the cached sequence.
• After the last sequence value is used, the next request for the sequence pulls another cache of sequences into memory.
• 20 is the default number of sequence numbers cached.

Nonsequential Numbers
• Although sequence generators issue sequential numbers without gaps, this action occurs independently of a database commit or rollback.
• Gaps (nonsequential numbers) can be generated by:
  – Rolling back a statement containing a sequence, the number is lost.
  – A system crash. If the sequence caches values into the memory and the system crashes, those values are lost.
  – The same sequence being used for multiple tables. If you do so, each table can contain gaps in the sequential numbers.

Viewing the Next Value
• If the sequence was created with NOCACHE, it is possible to view the next available sequence value without incrementing it by querying the USER_SEQUENCES table.

SELECT sequence_name, min_value, max_value, last_number AS "Next number"
FROM USER_SEQUENCES
WHERE sequence_name = 'RUNNER_ID_SEQ';

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	Next number
RUNNER_ID_SEQ	1	50000	3

Modifying a Sequence
• As with the other database objects you've created, a SEQUENCE can also be changed using the ALTER SEQUENCE statement.
• What if the London Marathon exceeded the 50,000 runner registrations and you needed to add more numbers?
• The sequence could be changed to increase the MAXVALUE without changing the existing number order.

```
ALTER SEQUENCE runner_id_seq
INCREMENT BY 1
MAXVALUE 999999
NOCACHE
NOCYCLE;
```

• Some validation is performed when you alter a sequence.
• For example, a new MAXVALUE that is less than the current sequence number cannot be executed.

```
ALTER SEQUENCE runner_id_seq
INCREMENT BY 1
MAXVALUE 90
NOCACHE
NOCYCLE;
ERROR at line 1:
ORA-04009: MAXVALUE cannot be made to be less than the current value
```

ALTER SEQUENCE Guidelines
• A few guidelines apply when executing an ALTER SEQUENCE statement.
• They are:

– You must be the owner or have the ALTER privilege for the sequence in order to modify it.
– Only future sequence numbers are affected by the ALTER SEQUENCE statement.
– The START WITH option cannot be changed using ALTER SEQUENCE. The sequence must be dropped and re-created in order to restart the sequence at a different number.

Removing a Sequence
• To remove a sequence from the data dictionary, use the DROP SEQUENCE statement.
• You must be the owner of the sequence or have DROP ANY SEQUENCE privileges to remove it.
• Once removed, the sequence can no longer be referenced.

DROP SEQUENCE runner_id_seq;

16-2 Indexes and Synonyms

This lesson covers the following objectives:
• Define an index and its use as a schema object
• Name the conditions that cause an index to be created automatically
• Create and execute a CREATE INDEX and DROP INDEX statement
• Create and execute a function-based index
• Create private and public synonyms
Oracle uses an index to speed up the retrieval of rows.

Indexes
• An Oracle Server index is a schema object that can speed up the retrieval of rows by using a pointer. Indexes can be created explicitly or automatically.
• If you do not have an index on the column you're selecting, then a full table scan occurs.
• An index provides direct and fast access to rows in a table.
• Its purpose is to reduce the necessity of disk I/O (input/output) by using an indexed path to locate data quickly.
• The index is used and maintained automatically by the Oracle Server. Once an index is created, no direct activity is required by the user.
• A ROWID is a base 64 string representation of the row address containing block identifier, row location in the block, and the database file identifier.
• Indexes use ROWID's because they are the fastest way to access any particular row.
• Indexes are logically and physically independent of the table they index.
• This means that they can be created or dropped at any time and have no effect on the base tables or other indexes.

COUNTRY_ID	COUNTRY_NAME	CAPITOL	REGION_ID
1	United States of America	Washington, DC	21
2	Canada	Ottawa	21
3	Republic of Kazakhstan	Astana	143
7	Russian Federation	Moscow	151
12	Coral Sea Islands Territory	-	9
13	Cook Islands	Avarua	9
15	Europa Island	-	18
20	Arab Republic of Egypt	Cairo	15
…	…	…	…

• Note: When you drop a table, corresponding indexes are also dropped.

Types of Indexes
• Two types of indexes can be created:
– Unique index: The Oracle Server automatically creates this index when you define a column in a table to have a PRIMARY KEY or a UNIQUE KEY constraint.
• The name of the index is the name given to the constraint.
• Although you can manually create a unique index, it is recommended that you create a unique constraint in the table, which implicitly creates a unique index.
– Nonunique index: This is an index that a user can create to speed up access to the rows.
• For example, to optimize joins, you can create an index on the FOREIGN KEY column, which speeds up the search to match rows to the PRIMARY KEY column.

Creating an Index
• Create an index on one or more columns by issuing the CREATE INDEX statement:

CREATE INDEX index_name ON table_name(column...,column)

•To create an index in your schema, you must have the CREATE TABLE privilege.
•To create an index in any schema, you need the CREATE ANY INDEX privilege or the CREATE TABLE privilege on the table on which you are creating the index. Null values are not included in the index.
• For example, to improve the speed of query access to the REGION_ID column in the WF_COUNTRIES table:
CREATE INDEX wf_cont_reg_id_idx ON wf_countries(region_id);

When to Create an Index
• An index should be created only if:
  – The column contains a wide range of values
  – A column contains a large number of null values
  – One or more columns are frequently used together in a WHERE clause or a join condition
  – The table is large and most queries are expected to retrieve less than 2-4% of the rows.

When Not to Create an Index
• When deciding whether or not to create an index, more is not always better.
• Each DML operation (INSERT, UPDATE, DELETE) that is performed on a table with indexes means that the indexes must be updated.
• The more indexes you have associated with a table, the more effort it takes to update all the indexes after the DML operation.
• It is usually not worth creating an index if:
  – The table is small
  – The columns are not often used as a condition in the query
  – Most queries are expected to retrieve more than 2-4 % of the rows in the table
  – The table is updated frequently
  – The indexed columns are referenced as part of an expression

Composite Index
• A composite index (also called a "concatenated" index) is an index that you create on multiple columns in a table.
• Columns in a composite index can appear in any order and need not be adjacent in the table.
• Composite indexes can speed retrieval of data for SELECT statements in which the WHERE clause references all or the leading portion of the columns in the composite index.

CREATE INDEX emps_name_idx ON employees(first_name, last_name);

• Null values are not included in the composite index.
• To optimize joins, you can create an index on the FOREIGN KEY column, which speeds up the search to match rows to the PRIMARY KEY column.
• The optimizer does not use an index if the WHERE clause contains the IS NULL expression.

Confirming Indexes
• Confirm the existence of indexes from the USER_INDEXES data dictionary view.
• You can also check the columns involved in an index by querying the USER_IND_COLUMNS view.
• The query shown on the next slide is a join between the USER_INDEXES table (names of the indexes and their uniqueness) and the USER_IND_COLUMNS (names of the indexes, table names, and column names) table.

```
SELECT DISTINCT ic.index_name, ic.column_name, ic.column_position, id.uniqueness
FROM user_indexes id, user_ind_columns ic
WHERE id.table_name = ic.table_name
AND ic.table_name = 'EMPLOYEES';
```

INDEX NAME	COLUMN NAME	COL POS	UNIQUENESS
EMP_EMAIL_UK	EMAIL	1	UNIQUE
EMP_EMP_ID_PK	EMPLOYEE_ID	1	UNIQUE
EMP_DEPARTMENT_IX	DEPARTMENT_ID	1	NONUNIQUE
EMP_JOB_IX	JOB_ID	1	NONUNIQUE
EMP_MANAGER_IX	MANAGER_ID	1	NONUNIQUE
EMP_NAME_IX	LAST_NAME	1	NONUNIQUE
EMP_NAME_IX	FIRST_NAME	2	NONUNIQUE

Function-based Indexes

• A function-based index stores the indexed values and uses the index based on a SELECT statement to retrieve the data.
• A function-based index is an index based on expressions.
• The index expression is built from table columns, constants, SQL functions, and user-defined functions.
• Function-based indexes are useful when you don't know in what case the data was stored in the database.
• For example, you can create a function-based index that can be used with a SELECT statement using UPPER in the WHERE clause.
• The index will be used in this search.

```
CREATE INDEX upper_last_name_idx ON employees (UPPER(last_name));

SELECT * FROM employees
WHERE UPPER(last_name) = 'KING';
```

• Function-based indexes defined with the UPPER(column_name) or LOWER(column_name) keywords allow case-insensitive searches.
• If you don't know how the employee last names were entered into the database, you could still use the index by entering uppercase in the SELECT statement.
• When a query is modified using an expression in the WHERE clause, the index won't use it unless you create a function-based index to match the expression.
• For example, the following statement allows for case-insensitive searches using the index:

```
CREATE INDEX upper_last_name_idx ON employees (UPPER(last_name));

SELECT * FROM employees
WHERE UPPER(last_name) LIKE 'KIN%';
```

• To ensure that the Oracle Server uses the index rather than performing a full table scan, be sure that the value of the function is not null in subsequent queries.
• For example, the following statement is guaranteed to use the index, but without the WHERE clause the Oracle Server may perform a full table scan:

```
SELECT * FROM employees
WHERE UPPER (last_name) IS NOT NULL
ORDER BY UPPER (last_name);
```

•The Oracle Server treats indexes with columns marked DESC as function-based indexes.
•The columns marked DESC are sorted in descending order.
• All of these examples use the UPPER and LOWER functions, but it is worth noticing that while these two are very frequently used in Function Based Indexes, the Oracle database is not limited to them.
• Any valid Oracle Built-in function, for example TO_CHAR can be used.
• Another example of Function Based Indexes is shown here.
• The employees table is queried to find any employees hired in 1987.

```
SELECT first_name, last_name, hire_date
FROM employees
WHERE TO_CHAR(hire_date, 'yyyy') = '1987'
```

•This query results in a Full Table Scan, which can be a very expensive operation if the table is big.
•Even if the hire_date column is indexed, the index is not used due to the TO_CHAR expression.
• Once we create the following Function Based Index, we can run the same query, but this time avoid the expensive Full Table Scan.

```
CREATE INDEX emp_hire_year_idx ON employees (TO_CHAR(hire_date, 'yyyy'));
SELECT first_name, last_name, hire_date
FROM employees
WHERE TO_CHAR(hire_date, 'yyyy') = '1987'
```

FIRST_NAME	LAST_NAME	HIRE_DATE
Steven	King	17-Jun-1987
Jennifer	Whalen	17-Sep-1987

•Now, Oracle can use the index on the hire_date column.

Removing an Index

• You cannot modify indexes.
• To change an index, you must drop it and then re-create it.
• Remove an index definition from the data dictionary by issuing the DROP INDEX statement.
• To drop an index, you must be the owner of the index or have the DROP ANY INDEX privilege.
• If you drop a table, indexes and constraints are automatically dropped, but views and sequences remain.

```
DROP INDEX upper_last_name_idx;
DROP INDEX emps_name_idx;
DROP INDEX emp_hire_year_idx;
```

SYNONYM

• In SQL, as in language, a synonym is a word or expression that is an accepted substitute for another word.
• Synonyms are used to simplify access to objects by creating another name for the object.
• Synonyms can make referring to a table owned by another user easier and shorten lengthy object names.
• For example, to refer to the amy_copy_employees table in your classmate's schema, you can prefix the table name with the name of the user who created it followed by a period and then the table name, as in USMA_SBHS_SQL01_S04. amy_copy_employees.
• Creating a synonym eliminates the need to qualify the object name with the schema and provides you with an alternative name for a table, view, sequence, procedure, or other object.
• This method can be especially useful with lengthy object names, such as views.
• The database administrator can create a public synonym accessible to all users and can specifically grant the CREATE PUBLIC SYNONYM privilege to any user, and that user can create public synonyms.
• Syntax:

```
CREATE [PUBLIC] SYNONYM synonym
FOR object;
```

•PUBLIC: creates a synonym accessible to all users
•synonym: is the name of the synonym to be created
•object: identifies the object for which the synonym is created

```
CREATE SYNONYM amy_emps
FOR amy_copy_employees;
```

SYNONYM Guidelines

• Guidelines:
  – The object cannot be contained in a package.
  – A private synonym name must be distinct from all other objects owned by the same user.
• To remove a synonym:

```
DROP [PUBLIC] SYNONYM name_of_synonym
DROP SYNONYM amy_emps;
```

Confirming a SYNONYM

• The existence of synonyms can be confirmed by querying the USER_SYNONYMS data dictionary view.

Column Name	Contents
Synonym_name	Name of the synonym.
Table_name	Owner of the object referenced by the synonym.
Table_owner	Name of the object referenced by the synonym.
Db_link	Database link referenced in a remote synonym.