Creating Sequences, Synonyms, and Indexes

Objectives

After completing this lesson, you should be able to:

- Create, maintain, and use sequences
- Create private and public synonyms
- Create and maintain indexes
- Query various data dictionary views to find information for sequences, synonyms, and indexes

In this lesson, you are introduced to the sequence, synonyms, and index objects. You learn the basics of creating and using sequences, synonyms and indexes.

Lesson Agenda

- Overview of sequences:
  - Creating, using, and modifying a sequence
  - Cache sequence values
  - `NEXTVAL` and `CURRVAL` pseudocolumns
  - SQL column defaulting using a sequence
- Overview of synonyms
  - Creating, dropping synonyms
- Overview of indexes
  - Creating indexes
  - Using the `CREATE TABLE` statement
  - Creating function-based indexes
  - Creating multiple indexes on the same set of columns
  - Removing indexes

Database Objects

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of data retrieval queries
Synonym	Gives alternative names to objects

There are several other objects in a database in addition to tables.

With views, you can present and hide data from the tables.

Many applications require the use of unique numbers as primary key values. You can either build code into the application to handle this requirement or use a sequence to generate unique numbers.

If you want to improve the performance of data retrieval queries, you should consider creating an index. You can also use indexes to enforce uniqueness on a column or a collection of columns.

You can provide alternative names for objects by using synonyms.

Schema

- A schema is a collection of logical structures of data or *schema objects*.
- A schema is owned by a database user and has the same name as that user. Each user owns a single schema.
- Schema objects can be created and manipulated with SQL and include tables, views, synonyms, sequences, stored procedures, indexes, clusters, and database links.
- If a table does not belong to the user, the owner's name must be prefixed to the table.
  - SELECT * FROM schema.tablename;

Referencing Another User's Tables

- Tables belonging to other users are not in the user's schema.
- You should use the owner's name as a prefix to those tables.

USERA

```
SELECT *
FROM userB.employees;
```

USERB

```
SELECT *
FROM userA.employees;
```


A schema is a collection of logical structures of data or *schema objects*. A schema is owned by a database user and has the same name as that user. Each user owns a single schema.

Schema objects can be created and manipulated with SQL and include tables, views, synonyms, sequences, stored procedures, indexes, clusters, and database links.

If a table does not belong to the user, the owner's name must be prefixed to the table. For example, if there are schemas named `USERA` and `USERB`, and both have an `EMPLOYEES` table, then if `USERA` wants to access the `EMPLOYEES` table that belongs to `USERB`, `USERA` must prefix the table name with the schema name:

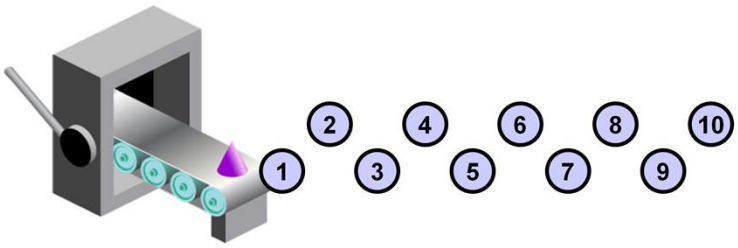
```
SELECT *
FROM   userb.employees;
```

If `USERB` wants to access the `EMPLOYEES` table that is owned by `USERA`, `USERB` must prefix the table name with the schema name:

```
SELECT *
FROM   usera.employees;
```

Sequences

- User-created database object that can automatically generate unique numbers
- Is shared by many users to generate integers
- Typical usage is to create a primary key value
- This can be a time-saving object, because it can reduce the amount of application code needed to write a sequence-generating routine.



ORACLE

A sequence is a user-created database object that can be shared by multiple users to generate integers.

You can define a sequence to generate unique values or to recycle and use the same numbers again.

A typical usage for sequences is to create a primary key value, which must be unique for each row. A sequence is generated and incremented (or decremented) by an internal Oracle routine. This can be a time-saving object, because it can reduce the amount of application code needed to write a sequence-generating routine.

Sequence numbers are stored and generated independent of tables. Therefore, the same sequence can be used for multiple tables.

CREATE SEQUENCE Statement: Syntax

Define a sequence to generate sequential numbers automatically:

```
CREATE SEQUENCE [ schema. ] sequence
  [ { INCREMENT BY | START WITH } integer
  | { MAXVALUE integer | NOMAXVALUE }
  | { MINVALUE integer | NOMINVALUE }
  | { CYCLE | NOCYCLE }
  | { CACHE integer | NOCACHE }
  | { ORDER | NOORDER }
];
```

ORACLE

Automatically generate sequential numbers by using the `CREATE SEQUENCE` statement.

In the syntax:

sequence	Is the name of the sequence generator
`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 (This is the default option.)
`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 (This is the default option.)

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

Creating a Sequence

- Create a sequence named DEPT_DEPTID_SEQ to be used for the primary key of the DEPARTMENTS table.
- Do not use the CYCLE option.

```
CREATE SEQUENCE dept_deptid_seq
                INCREMENT BY 10
                START WITH 280
                MAXVALUE 9999
                NOCACHE
                NOCYCLE;
```

```
sequence DEPT_DEPTID_SEQ created.
```

ORACLE

The example in the slide creates a sequence named DEPT_DEPTID_SEQ to be used for the DEPARTMENT_ID column of the DEPARTMENTS table. The sequence starts at 280, does not allow caching, and does not cycle.

Do not use the CYCLE option if the sequence is used to generate primary key values, unless you have a reliable mechanism that purges old rows faster than the sequence cycles.

For more information, see the "CREATE SEQUENCE" section in the *Oracle Database SQL Language Reference* for Oracle Database 19c.

Note: The sequence is not tied to a table. Generally, you should name the sequence after its intended use. However, the sequence can be used anywhere, regardless of its name.

NEXTVAL and CURRVAL Pseudocolumns

- Reference the sequence values by using the NEXTVAL and CURRVAL pseudocolumns.
- NEXTVAL returns the next available sequence value. It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for that sequence before CURRVAL contains a value.

ORACLE

After you create your sequence, it generates sequential numbers for use in your tables. Reference the sequence values by using the 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 CURRVAL pseudocolumn is used to refer to a sequence number that the current user has just generated. However, 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 you reference *sequence*.CURRVAL, the last value returned to that user's process is displayed.

Rules for Using NEXTVAL and CURRVAL

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

For more information, see the “Pseudocolumns” and “CREATE SEQUENCE” sections in *Oracle Database SQL Language Reference* for Oracle Database 19c.

Using a Sequence

- Insert a new department named “Support” in location ID 2500:

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

1 rows inserted

- View the current value for the DEPT_DEPTID_SEQ sequence:

```
SELECT   dept_deptid_seq.CURRVAL
FROM     dual;
```

ORACLE

The example in the slide inserts a new department in the DEPARTMENTS table. It uses the DEPT_DEPTID_SEQ sequence to generate a new department number as follows.

You can view the current value of the sequence using the *sequence_name*.CURRVAL, as shown in the second example in the slide. The output of the query is shown below:

	CURRVAL
1	280

Suppose that you now 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 EMPLOYEE_SEQ has already been created to generate new employee numbers.

SQL Column defaulting using a Sequence

- SQL syntax for column defaults allow `<sequence>.nextval`, `<sequence>.currval` as a SQL column defaulting expression for numeric columns, where `<sequence>` is an Oracle database sequence.
- The `DEFAULT` expression can include the sequence pseudocolumns `CURRVAL` and `NEXTVAL`, as long as the sequence exists and you have the privileges necessary to access it.

```
CREATE SEQUENCE s1 START WITH 1;
CREATE TABLE emp (a1 NUMBER DEFAULT s1.NEXTVAL NOT
NULL, a2 VARCHAR2(10));
INSERT INTO emp (a2) VALUES ('john');
INSERT INTO emp (a2) VALUES ('mark');
SELECT * FROM emp;
```

```
sequence S1 created.
table EMP created.
1 rows inserted.
1 rows inserted.
A1 A2
-- ----------
 1 john
 2 mark
```

ORACLE

SQL syntax for column defaults has been enhanced so that it allows `<sequence>.nextval, <sequence>.currval` as a SQL column defaulting expression for numeric columns, where `<sequence>` is an Oracle database sequence.

The `DEFAULT` expression can include the sequence pseudocolumns `CURRVAL` and `NEXTVAL`, as long as the sequence exists and you have the privileges necessary to access it. The user inserting into a table must have access privileges to the sequence. If the sequence is dropped, subsequent insert DMLs where *expr* is used for defaulting will result in a compilation error.

In the slide example, sequence s1 is created, which starts from 1.

Caching Sequence Values

- You can cache sequences in memory to provide faster access to those 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.

ORACLE

You can cache sequences in memory to provide faster access to those 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.

Gaps in the Sequence

Although sequence generators issue sequential numbers without gaps, this action occurs independently of a commit or rollback. Therefore, if you roll back a statement containing a sequence, the number is lost.

Another event that can cause gaps in the sequence is a system crash. If the sequence caches values in memory, those values are lost if the system crashes.

Because sequences are not tied directly to tables, the same sequence can be used for multiple tables. However, if you do so, each table can contain gaps in the sequential numbers.

Gaps in the Sequence

- Although sequence generators issue sequential numbers without gaps, this action occurs independently of a commit or rollback. Therefore, if you roll back a statement containing a sequence, the number is lost.
- Another event that can cause gaps in the sequence is a system crash. If the sequence caches values in memory, those values are lost if the system crashes.
- Because sequences are not tied directly to tables, the same sequence can be used for multiple tables.
- However, if you do so, each table can contain gaps in the sequential numbers.

Modifying a Sequence

- If you reach the MAXVALUE limit for your sequence, no additional values from the sequence are allocated and you will receive an error indicating that the sequence exceeds the MAXVALUE.
- To continue to use the sequence, you can modify it by using the ALTER SEQUENCE statement.

```
ALTER SEQUENCE dept_deptid_seq
               INCREMENT BY 20
               MAXVALUE 999999
               NOCACHE
               NOCYCLE;
```

```
sequence DEPT_DEPTID_SEQ altered.
```

ORACLE

If you reach the MAXVALUE limit for your sequence, no additional values from the sequence are allocated and you will receive an error indicating that the sequence exceeds the MAXVALUE. To continue to use the sequence, you can modify it by using the ALTER SEQUENCE statement.

Syntax

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

In the syntax, *sequence* is the name of the sequence generator.

For more information, see the section on "ALTER SEQUENCE" in *Oracle Database SQL Language Reference* for Oracle Database 19c.

Guidelines for Modifying a Sequence

- You must be the owner or have the ALTER privilege for the sequence.
- Only future sequence numbers are affected.
- The sequence must be dropped and re-created to restart the sequence at a different number.
- Some validation is performed.
- To remove a sequence, use the DROP statement:

```
DROP SEQUENCE dept_deptid_seq;
sequence DEPT_DEPTID_SEQ dropped.
```

ORACLE

- You must be the owner or have the ALTER privilege for the sequence to modify it. You must be the owner or have the DROP ANY SEQUENCE privilege to remove 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 to restart the sequence at a different number.
- Some validation is performed. For example, a new MAXVALUE that is less than the current sequence number cannot be imposed.

```
ALTER SEQUENCE dept_deptid_seq
      INCREMENT BY 20
      MAXVALUE 90
      NOCACHE
      NOCYCLE;
```

- The error:

```
SQL Error: ORA-04009: MAXVALUE cannot be made to be less than the current value
04009. 00000 - "MAXVALUE cannot be made to be less than the current value"
*Cause:    the current value exceeds the given MAXVALUE
*Action:   make sure that the new MAXVALUE is larger than the current value
```

Sequence Information

- The USER_SEQUENCES view describes all sequences that you own.

```
DESCRIBE user_sequences
```

```
DESCRIBE user_sequences
Name            Null     Type
--------------- -------- -------------
SEQUENCE_NAME   NOT NULL VARCHAR2(128)
MIN_VALUE                NUMBER
MAX_VALUE                NUMBER
INCREMENT_BY    NOT NULL NUMBER
CYCLE_FLAG               VARCHAR2(1)
ORDER_FLAG               VARCHAR2(1)
CACHE_SIZE      NOT NULL NUMBER
LAST_NUMBER     NOT NULL NUMBER
PARTITION_COUNT          NUMBER
SESSION_FLAG             VARCHAR2(1)
KEEP_VALUE               VARCHAR2(1)
```

- Verify your sequence values in the USER_SEQUENCES data dictionary table.

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

ORACLE

The USER_SEQUENCES view describes all sequences that you own. When you create the sequence, you specify criteria that are stored in the USER_SEQUENCES view. The columns in this view are:

- SEQUENCE_NAME: Name of the sequence
- MIN_VALUE: Minimum value of the sequence
- MAX_VALUE: Maximum value of the sequence
- INCREMENT_BY: Value by which the sequence is incremented
- CYCLE_FLAG: Whether sequence wraps around on reaching the limit
- ORDER_FLAG: Whether sequence numbers are generated in order
- CACHE_SIZE: Number of sequence numbers to cache
- LAST_NUMBER: Last sequence number written to disk. If a sequence uses caching, the number written to disk is the last number placed in the sequence cache. This number is likely to be greater than the last sequence number that was used. The LAST_NUMBER column displays the next available sequence number if NOCACHE is specified.

After creating your sequence, it is documented in the data dictionary. Because a sequence is a database object, you can identify it in the USER_OBJECTS data dictionary table.

You can also confirm the settings of the sequence by selecting from the USER_SEQUENCES data dictionary view.

USER_SEQUENCES view

- SEQUENCE_NAME: Name of the sequence
- MIN_VALUE: Minimum value of the sequence
- MAX_VALUE: Maximum value of the sequence
- INCREMENT_BY: Value by which the sequence is incremented
- CYCLE_FLAG: Whether sequence wraps around on reaching the limit
- ORDER_FLAG: Whether sequence numbers are generated in order
- CACHE_SIZE: Number of sequence numbers to cache
- LAST_NUMBER: Last sequence number written to disk.

Lesson Agenda

- Overview of sequences:
  - Creating, using, and modifying a sequence
  - Cache sequence values
  - `NEXTVAL` and `CURRVAL` pseudocolumns
  - SQL column defaulting using a sequence
- Overview of synonyms
  - Creating, dropping synonyms
- Overview of indexes
  - Creating indexes
  - Using the `CREATE TABLE` statement
  - Creating function-based indexes
  - Creating multiple indexes on the same set of columns
  - Removing indexes

Synonyms

A synonym

- database object that enable you to call a table by another name.
- Can be created to give an alternative name to a table or to an other database object
- Because a synonym is simply an alias, requires no storage other than its definition in the data dictionary
- Is useful for hiding the identity and location of an underlying schema object

Synonyms are database object that enable you to call a table by another name.

You can create synonyms to give an alternative name to a table or to an other database object. For example, you can create a synonym for a table or view, sequence, PL/SQL program unit, user-defined object type, or another synonym.

Because a synonym is simply an alias, it requires no storage other than its definition in the data dictionary.

Synonyms can simplify SQL statements for database users. Synonyms are also useful for hiding the identity and location of an underlying schema object.

Creating a Synonym for an Object

- 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 objects.
- This method can be especially useful with lengthy object names, such as views.

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

`PUBLIC` Creates a synonym that is accessible to all users

To create a `PUBLIC` synonym, you must have the `CREATE PUBLIC SYNONYM` system privilege.

ORACLE

To refer to a table that is owned by another user, you need to prefix the table name with the name of the user who created it, followed by a period. 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 objects. This method can be especially useful with lengthy object names, such as views.

In the syntax:

`PUBLIC`	Creates a synonym that is accessible to all users
`synonym`	Is the name of the synonym to be created
`object`	Identifies the object for which the synonym is created

Guidelines

- The object cannot be contained in a package.
- A private synonym name must be distinct from all other objects that are owned by the same user.
- To create a `PUBLIC` synonym, you must have the `CREATE PUBLIC SYNONYM` system privilege.

For more information, see the section on "`CREATE SYNONYM`" in *Oracle Database SQL*

Language Reference for Oracle Database 19c.

Creating and Removing Synonyms

- Create a shortened name for the DEPT_SUM_VU view:

```
CREATE SYNONYM  d_sum
FOR  dept_sum_vu;
```

```
synonym D_SUM created.
```

```
DROP SYNONYM d_sum;
```

```
synonym D_SUM dropped.
```

- DBA can create a public synonym that is accessible to all users. The following example creates a public synonym named DEPT for Alice's DEPARTMENTS table:

```
CREATE PUBLIC SYNONYM  dept
FOR    alice.departments;
```

ORACLE

Creating a Synonym

The slide example creates a synonym for the DEPT_SUM_VU view for quicker reference.

The database administrator can create a public synonym that is accessible to all users. The following example creates a public synonym named DEPT for Alice's DEPARTMENTS table:

```
CREATE PUBLIC SYNONYM  dept
FOR    alice.departments;
```

```
public synonym DEPT created.
```

Removing a Synonym

To remove a synonym, use the DROP SYNONYM statement. Only the database administrator can drop a public synonym.

```
DROP PUBLIC SYNONYM  dept;
```

For more information, see the section on "DROP SYNONYM" in *Oracle Database SQL Language Reference* for Oracle Database 19c.

Synonym Information

```
DESCRIBE user_synonyms
```

```
DESCRIBE user_synonyms
Name         Null     Type
------------ -------- -------------
SYNONYM_NAME NOT NULL VARCHAR2(128)
TABLE_OWNER           VARCHAR2(128)
TABLE_NAME   NOT NULL VARCHAR2(128)
DB_LINK               VARCHAR2(128)
```

```
SELECT *
FROM   user_synonyms;
```

	SYNONYM_NAME	TABLE_OWNER	TABLE_NAME	DB_LINK
1	D_SUM	ORA21	DEPT_SUM_VU	(null)

- SYNONYM_NAME: Name of the synonym
- TABLE_OWNER: Owner of the object that is referenced by the synonym
- TABLE_NAME: Name of the table or view that is referenced by the synonym
- DB_LINK: Name of the database link reference (if any)

The USER_SYNONYMS dictionary view describes private synonyms (synonyms that you own).

You can query this view to find your synonyms. You can query ALL_SYNONYMS to find out the name of all the synonyms that are available to you and the objects on which these synonyms apply.

The columns in this view are:

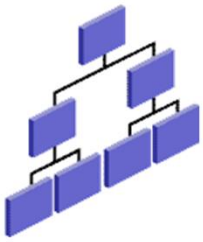
- SYNONYM_NAME: Name of the synonym
- TABLE_OWNER: Owner of the object that is referenced by the synonym
- TABLE_NAME: Name of the table or view that is referenced by the synonym
- DB_LINK: Name of the database link reference (if any)

Lesson Agenda

- Overview of sequences:
  - Creating, using, and modifying a sequence
  - Cache sequence values
  - `NEXTVAL` and `CURRVAL` pseudocolumns
  - SQL column defaulting using a sequence
- Overview of synonyms
  - Creating, dropping synonyms
- Overview of indexes
  - Creating indexes
  - Using the `CREATE TABLE` statement
  - Creating function-based indexes
  - Creating multiple indexes on the same set of columns
  - Removing indexes

Indexes

- An Oracle Server index is a schema object
- Can be used by the Oracle Server to speed up the retrieval of rows by using a pointer and improves the performance of some queries.
- Its purpose is to reduce disk input/output (I/O) by using a indexed path to locate data quickly.
- After an index is created, no direct activity is required by the user.
- Indexes are logically and physically independent of the data in the objects with which they are associated. This means that they can be created or dropped at any time.



ORACLE

An Oracle Server index is a schema object that can speed up the retrieval of rows by using a pointer and improves the performance of some queries. Indexes can be created explicitly or automatically. If you do not have an index on the column, a full table scan occurs.

An index provides direct and fast access to rows in a table. Its purpose is to reduce the disk I/O by using an indexed path to locate data quickly. An index is used and maintained automatically by the Oracle Server. After an index is created, no direct activity is required by the user.

Indexes are logically and physically independent of the data in the objects with which they are associated. This means that they can be created or dropped at any time, and have no effect on the base tables or other indexes.

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

For more information, see the section on “Schema Objects: Indexes” in *Oracle Database Concepts* for Oracle Database 19c.

How Are Indexes Created?

- Automatically: A unique index is created automatically when you define a `PRIMARY KEY` or `UNIQUE` constraint in a table definition.

- Manually: You can create unique or nonunique index on columns to speed up access to the rows.

ORACLE

You can create two types of indexes.

- **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` constraint. The name of the index is the name that is given to the constraint.
- **Nonunique index:** This is an index that a user can create. For example, you can create the `FOREIGN KEY` column index for a join in a query to improve the speed of retrieval.

Note: You can manually create a unique index, but it is recommended that you create a unique constraint, which implicitly creates a unique index.

Creating an Index

- Create an index on one or more columns:

```
CREATE [UNIQUE][BITMAP]INDEX index
ON table (column[, column]...);
```

- Improve the speed of query access to the LAST_NAME column in the EMPLOYEES table:

```
CREATE INDEX  emp_last_name_idx
ON employees(last_name);
index EMP_LAST_NAME_IDX created.
```

- Specify BITMAP to indicate that the index is to be created with a bitmap for each distinct key, rather than indexing each row separately. Bitmap indexes store the rowids associated with a key value as a bitmap.

Create an index on one or more columns by issuing the CREATE INDEX statement.

In the syntax:

- index Is the name of the index
- table Is the name of the table
- Column Is the name of the column in the table to be indexed

Specify UNIQUE to indicate that the value of the column (or columns) upon which the index is based must be unique. Specify BITMAP to indicate that the index is to be created with a bitmap for each distinct key, rather than indexing each row separately. Bitmap indexes store the rowids associated with a key value as a bitmap.

For more information, see the section on "CREATE INDEX" in *Oracle Database SQL Language Reference* for Oracle Database 19c.

CREATE INDEX with the CREATE TABLE Statement

```
CREATE TABLE NEW_EMP
(employee_id NUMBER(6)
             PRIMARY KEY USING INDEX
             (CREATE INDEX emp_id_idx ON
             NEW_EMP(employee_id)),
first_name   VARCHAR2(20),
last_name    VARCHAR2(25));
```

```
table NEW_EMP created.
```

```
SELECT INDEX_NAME, TABLE_NAME
FROM   USER_INDEXES
WHERE  TABLE_NAME = 'NEW_EMP';
```

	INDEX_NAME	TABLE_NAME
1	EMP_ID_IDX	NEW_EMP

ORACLE

In the example in the slide, the CREATE INDEX clause is used with the CREATE TABLE statement to create a PRIMARY KEY index explicitly. You can name your indexes at the time of PRIMARY KEY creation to be different from the name of the PRIMARY KEY constraint.

You can query the USER_INDEXES data dictionary view for information about your indexes.

The following example illustrates the database behavior if the index is not explicitly named:

```
CREATE TABLE EMP_UNNAMED_INDEX
   (employee_id NUMBER(6) PRIMARY KEY ,
    first_name VARCHAR2(20),
    last_name VARCHAR2(25));
```

```
table EMP_UNNAMED_INDEX created.
```

```
SELECT INDEX_NAME, TABLE_NAME
FROM   USER_INDEXES
WHERE  TABLE_NAME = 'EMP_UNNAMED_INDEX';
```

	INDEX_NAME	TABLE_NAME
1	SYS_C0010972	EMP_UNNAMED_INDEX

Observe that the Oracle Server gives a generic name to the index that is created for the PRIMARY KEY column.

You can also use an existing index for your PRIMARY KEY column—for example, when you are expecting a large data load and want to speed up the operation. You may want to disable the constraints while performing the load and then enable them, in which case having a unique index on the PRIMARY KEY will still cause the data to be verified during the load. Therefore, you can first create a nonunique index on the column designated as PRIMARY KEY, and then create the PRIMARY KEY column and specify that it should use the existing index. The following examples illustrate this process:

Step 1: Create the table:

```
CREATE TABLE NEW_EMP2
 (employee_id NUMBER(6),
 first_name  VARCHAR2(20),
 last_name   VARCHAR2(25)
 );
```

Step 2: Create the index:

```
CREATE INDEX emp_id_idx2 ON
  new_emp2(employee_id);
```

Step 3: Create the PRIMARY KEY:

```
ALTER TABLE new_emp2 ADD PRIMARY KEY (employee_id) USING INDEX
 emp_id_idx2;
```

Function-Based Indexes

- Function-based indexes defined with the `UPPER(col_name)` keywords allow non-case-sensitive searches.
- The Oracle Server uses the index only when that particular function is used in a query.
- A full table scan is performed without a WHERE clause.

```
CREATE INDEX upper_dept_name_idx
ON dept2(UPPER(department_name));
```

```
index UPPER_DEPT_NAME_IDX created.
```

```
SELECT *
FROM   dept2
WHERE  UPPER(department_name) = 'SALES';
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	80	Sales	145	2500

ORACLE

Function-based indexes defined with the `UPPER(column_name)` or `LOWER(column_name)` keywords allow non-case-sensitive searches. For example, consider the following index:

```
CREATE INDEX upper_last_name_idx ON emp2 (UPPER(last_name));
```

This facilitates processing queries such as:

```
SELECT * FROM emp2 WHERE UPPER(last_name) = 'KING';
```

The Oracle Server uses the index only when that particular function is used in a query. For example, the following statement may 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);
```

Note: For creating a function-based index, you need the `QUERY REWRITE` system privilege. The `QUERY_REWRITE_ENABLED` initialization parameter must be set to `TRUE` for a function-based index to be used.

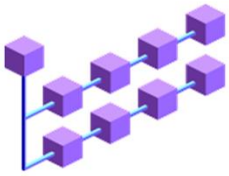
The Oracle Server treats indexes with columns marked `DESC` as function-based indexes. The columns marked `DESC` are sorted in descending order.

Function-Based Indexes

- **Note:** For creating a function-based index, you need the `QUERY REWRITE` system privilege.
- The `QUERY_REWRITE_ENABLED` initialization parameter must be set to `TRUE` for a function-based index to be used.

Creating Multiple Indexes on the Same Set of Columns

- You can create multiple indexes on the same set of columns.
- Multiple indexes can be created on the same set of columns if:
  - The indexes are of different types
  - The indexes uses different partitioning
  - The indexes have different uniqueness properties



You can create multiple indexes on the same set of columns if the indexes are of different types, use different partitioning, or have different uniqueness properties. For example, you can create a B-tree index and a bitmap index on the same set of columns.

Similarly, you can create both a unique and non-unique index on the same set of columns

When you have multiple indexes on the same set of columns, only one of these indexes can be visible at a time.

Note: Invisible Index – An invisible index is maintained by DML operations. To create an invisible index, you can use the `CREATE INDEX` statement with the `INVISIBLE` keyword.

Example of Creating Multiple Indexes on the Same Set Of Columns

```
CREATE INDEX emp_id_name_ix1
ON employees(employee_id, first_name);
```

```
index EMP_ID_NAME_IX1 created.
```

```
ALTER INDEX emp_id_name_ix1 INVISIBLE;
```

```
index EMP_ID_NAME_IX1 altered.
```

```
CREATE BITMAP INDEX emp_id_name_ix2
ON employees(employee_id, first_name);
```

```
bitmap index EMP_ID_NAME_IX2 created.
```


The code example shows the creation of a B-tree index, `emp_id_name_ix1`, on the `employee_id` and `first_name` column of the employees table in the `HR` schema. After the creation of the index, it is altered to make it invisible. Then a bitmap index is created on the `employee_id` and `first_name` column of the `employees` table in the `HR` schema. The bitmap index, `emp_id_name_ix2`, is visible by default.

Index Creation Guidelines

	Create an index when:
✓	A 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% to 4% of the rows in the table
	Do not create an index when:
✗	The columns are not often used as a condition in the query
✗	The table is small or most queries are expected to retrieve more than 2% to 4% of the rows in the table
✗	The table is updated frequently
✗	The indexed columns are referenced as part of an expression

ORACLE

More Is Not Always Better

Having more indexes on a table does not produce faster queries. Each DML operation that is committed on a table with indexes means that the indexes must be updated. The more indexes that you have associated with a table, the more effort the Oracle Server must make to update all the indexes after a DML operation.

When to Create an Index

Therefore, you should create indexes only if:

- The column contains a wide range of values
- The column contains a large number of null values
- One or more columns are frequently used together in a `WHERE` clause or join condition
- The table is large and most queries are expected to retrieve less than 2% to 4% of the rows

Remember that if you want to enforce uniqueness, you should define a unique constraint in the table definition. A unique index is then created automatically.

Index Information

- `USER_INDEXES` provides information about your indexes.
- `USER_IND_COLUMNS` describes columns of indexes owned by you and columns of indexes on your tables.

```
DESCRIBE user_indexes
```

```
DESCRIBE user_indexes
Name                  Null     Type
--------------------- -------- --------------
INDEX_NAME            NOT NULL VARCHAR2(128)
INDEX_TYPE                     VARCHAR2(27)
TABLE_OWNER           NOT NULL VARCHAR2(128)
TABLE_NAME            NOT NULL VARCHAR2(128)
TABLE_TYPE                     VARCHAR2(11)
UNIQUENESS                     VARCHAR2(9)
```

...

You query the `USER_INDEXES` view to find out the names of your indexes, the table name on which the index is created, and whether the index is unique.

Note: For a complete listing and description of the columns in the `USER_INDEXES` view, see "`USER_INDEXES`" in the *Oracle Database Reference* for Oracle Database 19c.

USER_INDEXES: Examples

a

```
SELECT index_name, table_name, uniqueness
FROM   user_indexes
WHERE  table_name = 'EMPLOYEES';
```

	INDEX_NAME	TABLE_NAME	UNIQUENESS
1	EMP_NAME_IX	EMPLOYEES	NONUNIQUE
2	EMP_MANAGER_IX	EMPLOYEES	NONUNIQUE
3	EMP_JOB_IX	EMPLOYEES	NONUNIQUE
4	EMP_DEPARTMENT_IX	EMPLOYEES	NONUNIQUE
5	EMP_EMP_ID_PK	EMPLOYEES	UNIQUE
6	EMP_EMAIL_UK	EMPLOYEES	UNIQUE

...

b

```
SELECT index_name, table_name
FROM   user_indexes
WHERE  table_name = 'EMP_LIB';
```

	INDEX_NAME	TABLE_NAME
1	SYS_C0010979	EMP_LIB

ORACLE

In slide example **a**, the `USER_INDEXES` view is queried to find the name of the index, name of the table on which the index is created, and whether the index is unique.

In slide example **b**, observe that the Oracle Server gives a generic name to the index that is created for the `PRIMARY KEY` column. The `EMP_LIB` table is created by using the following code:

```
CREATE TABLE emp_lib
   (book_id NUMBER(6) PRIMARY KEY,
    title VARCHAR2(25),
    category VARCHAR2(20));

table EMP_LIB created.
```

Querying USER_IND_COLUMNS

```
DESCRIBE user_ind_columns
```

```
DESCRIBE user_ind_columns
Name             Null Type
---------------- ---- --------------
INDEX_NAME            VARCHAR2(128)
TABLE_NAME            VARCHAR2(128)
COLUMN_NAME           VARCHAR2(4000)
COLUMN_POSITION       NUMBER
COLUMN_LENGTH         NUMBER
CHAR_LENGTH           NUMBER
DESCEND               VARCHAR2(4)
```

```
SELECT index_name, column_name,table_name
FROM   user_ind_columns
WHERE  index_name = 'LNAME_IDX';
```

	INDEX_NAME	COLUMN_NAME	TABLE_NAME
1	LNAME_IDX	LAST_NAME	EMP_TEST

The USER_IND_COLUMNS dictionary view provides information such as the name of the index, name of the indexed table, name of a column within the index, and the column's position within the index.

For the slide example, the emp_test table and LNAME_IDX index are created by using the following code:

```
CREATE TABLE emp_test AS SELECT * FROM employees;
CREATE INDEX lname_idx ON emp_test(last_name);
```

Removing an Index

- Remove an index from the data dictionary by using the `DROP INDEX` command:

```
DROP INDEX index;
```

- Remove the `emp_last_name_idx` index from the data dictionary:

```
DROP INDEX emp_last_name_idx;
index EMP_LAST_NAME_IDX dropped.
```

- To drop an index, you must be the owner of the index or have the `DROP ANY INDEX` privilege.

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.

In the syntax, *`index`* is the name of the index.

You can drop an index using the `ONLINE` keyword.

```
DROP INDEX emp_indx ONLINE;
```

`ONLINE`: Specify `ONLINE` to indicate that DML operations on the table are allowed while dropping the index.

Note: If you drop a table, indexes and constraints are automatically dropped but views remain.

Quiz

Indexes must be created manually and serve to speed up access to rows in a table.

a. True
b. False

Answer: b

Note: Indexes are designed to speed up query performance. However, not all indexes are created manually. The Oracle server automatically creates an index when you define a column in a table to have a `PRIMARY KEY` or a `UNIQUE` constraint.

Summary

In this lesson, you should have learned how to:

- Automatically generate sequence numbers by using a sequence generator
- Use synonyms to provide alternative names for objects
- Create indexes to improve the speed of query retrieval
- Find information about your objects through the following dictionary views:
  - `USER_VIEWS`
  - `USER_SEQUENCES`
  - `USER_SYNONYMS`

In this lesson, you should have learned about database objects such as sequences, indexes, and synonyms.