**Database 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.

**What Is a View?**

You can present logical subsets or combinations of data by creating views of tables. A view is a logical table based on a table or another view. A view contains no data of its own, but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called *base tables*. The view is stored as a SELECT statement in the data dictionary.

**Advantages of Views**

* + - Views restrict access to the data because it displays selected columns from the table.
    - Views can be used to make simple queries to retrieve the results of complicated queries. For example, views can be used to query information from multiple tables without the user knowing how to write a join statement.
    - Views provide data independence for ad hoc users and application programs. One view can be used to retrieve data from several tables.
    - Views provide groups of users access to data according to their particular criteria.

**Simple Views and Complex Views**

There are two classifications for views: simple and complex. The basic difference is related to the DML (INSERT, UPDATE, and DELETE) operations.

* + - A simple view is one that:
      * Derives data from only one table
      * Contains no functions or groups of data
      * Can perform DML operations through the view
    - A complex view is one that:
      * Derives data from many tables
      * Contains functions or groups of data
      * Does not always allow DML operations through the view
* **Creating a View**

You can create a view by embedding a subquery in the CREATE VIEW statement.

**In the syntax:**

OR REPLACE Re-creates the view if it already exists

FORCE Creates the view regardless of whether or not the base tables exist

NOFORCE Creates the view only if the base tables exist (This is the default.)

*view* Is the name of the view

*alias* Specifies names for the expressions selected by the view’s query (The number of aliases must match the number of expressions selected by the view.)

*subquery* Is a complete SELECT statement (You can use aliases for the columns in the SELECT list.)

WITH CHECK OPTION Specifies that only those rows that are accessible to the view can be inserted or updated

*constraint* Is the name assigned to the CHECK OPTION constraint

WITH READ ONLY Ensures that no DML operations can be performed on this view

**Note:** In SQL Developer, click the Run Script icon or press [F5] to run the data definition language (DDL) statements. The feedback messages will be shown on the Script Output tabbed page.

**Guidelines**

* + - The subquery that defines a view can contain complex SELECT syntax, including joins, groups, and subqueries.
    - If you do not specify a constraint name for the view created with the WITH CHECK OPTION, the system assigns a default name in the SYS\_C*n* format.
    - You can use the OR REPLACE option to change the definition of the view without dropping and re-creating it, or regranting the object privileges previously granted on it.
* You can control the column names by including column aliases in the subquery.
* The example in the slide creates a view containing the employee number (EMPLOYEE\_ID) with the alias ID\_NUMBER, name (LAST\_NAME) with the alias NAME, and annual salary (SALARY) with the alias ANN\_SALARY for every employee in department 50.
* Alternatively, you can use an alias after the CREATE statement and before the SELECT subquery. The number of aliases listed must match the number of expressions selected in the subquery.
* CREATE OR REPLACE VIEW salvu50 (ID\_NUMBER, NAME, ANN\_SALARY)

AS SELECT employee\_id, last\_name, salary\*12

FROM employees

WHERE department\_id = 50;

**Modifying a View**

With the OR REPLACE option, a view can be created even if one exists with this name already, thus replacing the old version of the view for its owner. This means that the view can be altered without dropping, re-creating, and regranting object privileges.

**Note:** When assigning column aliases in the CREATE OR REPLACE VIEW clause, remember that the aliases are listed in the same order as the columns in the subquery.

**Using the WITH CHECK OPTION Clause**

It is possible to perform referential integrity checks through views. You can also enforce constraints at the database level. The view can be used to protect data integrity, but the use is very limited.

The WITH CHECK OPTION clause specifies that INSERTs and UPDATEs performed through the view cannot create rows that the view cannot select. Therefore, it enables integrity constraints and data validation checks to be enforced on data being inserted or updated. If there is an attempt to perform DML operations on rows that the view has not selected, an error is displayed, along with the constraint name if that has been specified.

UPDATE empvu20

SET department\_id = 10

WHERE employee\_id = 201;

causes:

**Note:** No rows are updated because, if the department number were to change to 10, the view would no longer be able to see that employee. With the WITH CHECK OPTION clause, therefore, the view can see only the employees in department 20 and does not allow the department number for those employees to be changed through the view.

**Denying DML Operations**

Any attempt to remove a row from a view with a read-only constraint results in an error:

DELETE FROM empvu10

WHERE employee\_number = 200;

Similarly, any attempt to insert a row or modify a row using the view with a read-only constraint results in the same error.

**Removing a View**

You use the DROP VIEW statement to remove a view. The statement removes the view definition from the database. However, dropping views has no effect on the tables on which the view was based. Alternatively, views or other applications based on the deleted views become invalid. Only the creator or a user with the DROP ANY VIEW privilege can remove a view.

In the syntax, *view* is the name of the view.

**Sequences**

A sequence is a database object that creates integer values. You can create sequences and then use them to generate numbers.

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**

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.)

**Creating a Sequence (continued)**

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 preallocates and keeps in memory (By default, the Oracle server caches 20 values.)

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 120, 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 section on “CREATE SEQUENCE” in the *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

**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**

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
    - The DEFAULT expression in a CREATE TABLE or ALTER TABLE statement
* **Using a Sequence**
* 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:
* 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.
* **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.
* **Gaps in the Sequence**
* Although sequence generators issue sequential numbers without gaps, this action occurs independent 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.
* **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.

**Guidelines for Modifying a Sequence**

* + - 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:
* **Indexes**
* Indexes are database objects that you can create to improve the performance of some queries. Indexes can also be created automatically by the server when you create a primary key or a unique constraint.
* 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, 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 table that 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.
* **Note:** When you drop a table, the corresponding indexes are also dropped.

**How Are Indexes Created?**

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 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.

**Index Creation Guidelines**

**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.

**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.

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

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

**Synonyms**

Synonyms are database objects that enable you to call a table by another name. You can create synonyms to give an alternative name to a table.

**Creating a Synonym for an Object**

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.
* **Creating and Removing Synonyms**
* **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;
* **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;