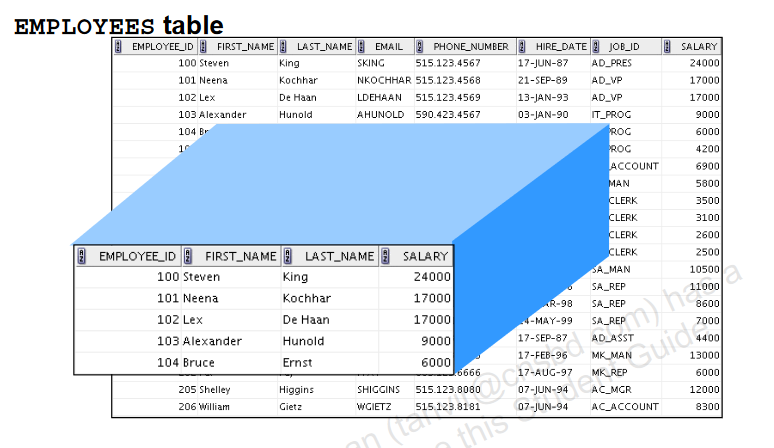
**View**

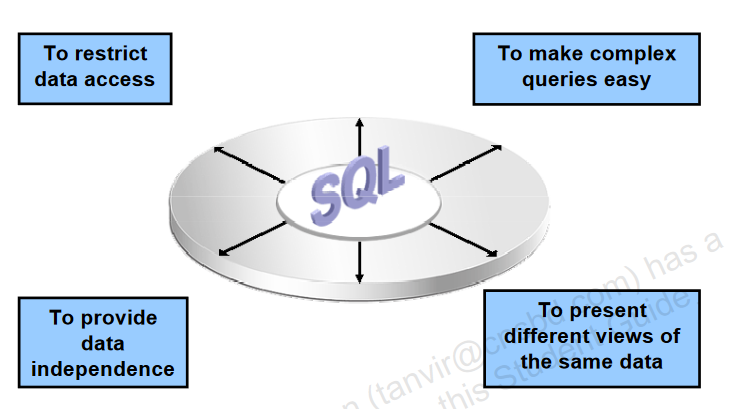
**What Is a View?**

* 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:**

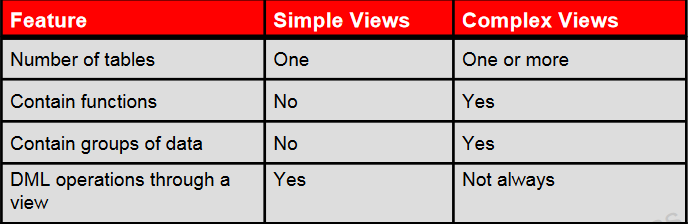
* Views restrict access to the data because it displays selected columns from the table.
* Views can be used to make simple queries to retrieve the results of complicated queries. For example, views can be used to query information from multiple tables without the user knowing how to write a join statement.
* 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.



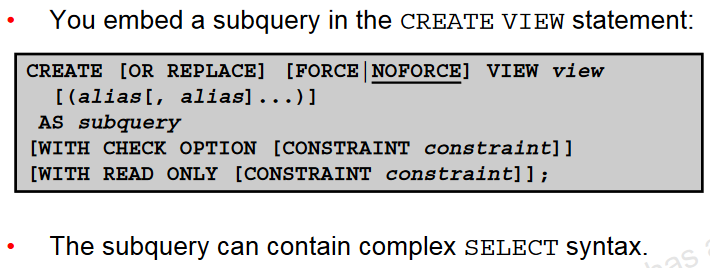
Simple Views and Complex Views

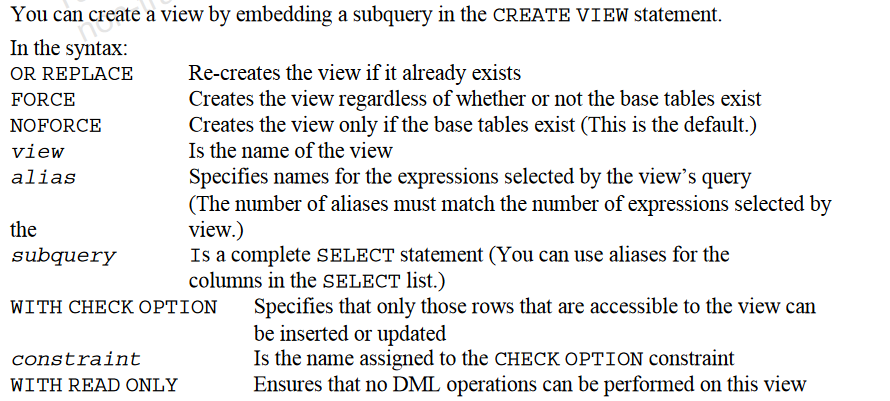
**There are two classifications for views:**

* **simple and**
* **complex.**
* **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**





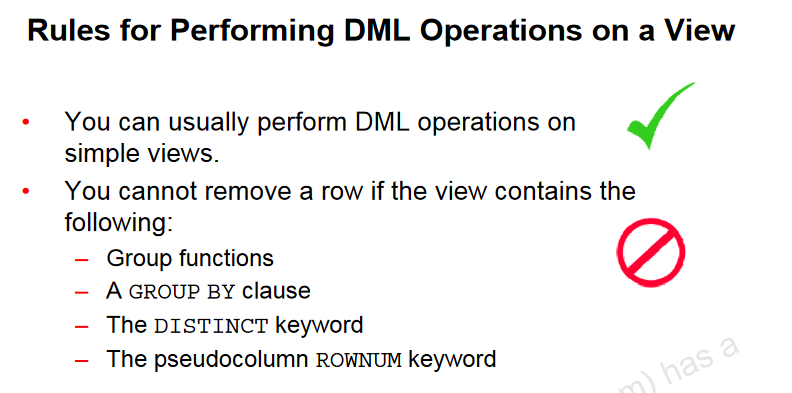
|  |
| --- |
| **EXAMPLE-1:**   * **Create the EMPVU80 view, which contains details of the employees in department 80:**   SELECT employee\_id, last\_name, salary  FROM employees  WHERE department\_id = 80;    CREATE VIEW empvu80  AS SELECT employee\_id, last\_name, salary  FROM employees  WHERE department\_id = 80;     * **Describe the structure of the view by using the SQL\*Plus DESCRIBE command:**   DESCRIBE empvu80;     * **Retrieving Data from a View**   SELECT \* FROM empvu80;    **EXAMPLE-2:**   * **Create a view by using column aliases in the subquery :**   SELECT employee\_id ID\_NUMBER, last\_name NAME,  salary\*12 ANN\_SALARY  FROM employees  WHERE department\_id = 50;    CREATE VIEW salvu50  AS SELECT employee\_id ID\_NUMBER, last\_name NAME,  salary\*12 ANN\_SALARY  FROM employees  WHERE department\_id = 50;     * **Describe the structure of the view by using the SQL\*Plus DESCRIBE command:**   DESCRIBE salvu50;     * **Retrieving Data from a View**   SELECT \* FROM salvu50; |

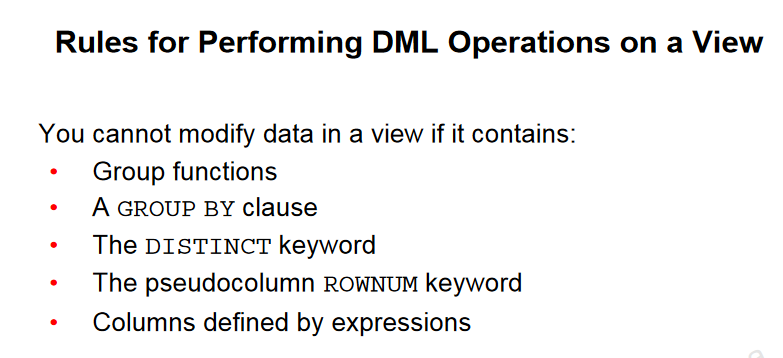
**Modifying a View**

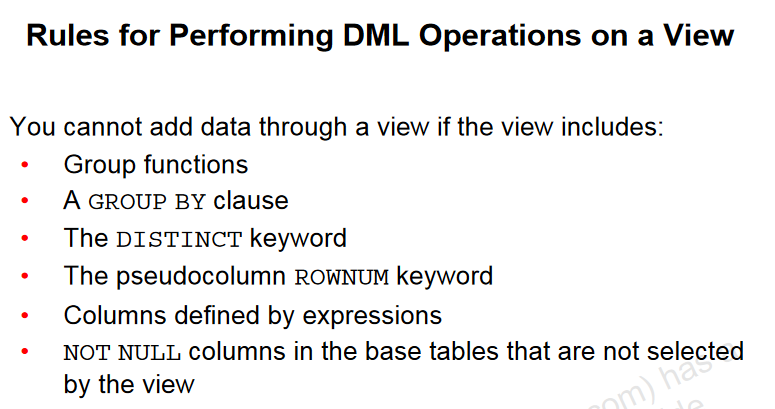
|  |
| --- |
| With the OR REPLACE option, a view can be created even if one exists with this name already, thus replacing the old version of the view for its owner. This means that the view can be altered without dropping, re-creating, and regranting object privileges.  Note: When assigning column aliases in the CREATE OR REPLACE VIEW clause, remember that the aliases are listed in the same order as the columns in the subquery.   * Modify the EMPVU80view by using a CREATE OR REPLACE VIEW clause. Add an alias for each column name:   DROP table emp;  CREATE table emp as SELECT \* FROM employees WHERE 1=2;  SELECT \* FROM emp;  INSERT INTO emp  SELECT \* FROM employees;  SELECT \* FROM emp;  CREATE OR REPLACE VIEW empvu80  (id\_number, name, sal, department\_id)  AS SELECT employee\_id, first\_name || ' '  || last\_name, salary, department\_id  FROM emp  WHERE department\_id = 80;     * Column aliases in the CREATE OR REPLACE VIEW clause are listed in the same order as the columns in the subquery.   SELECT \* FROM empvu80; |

**Creating a Complex View**

|  |
| --- |
| * Create a complex view that contains group functions to display values from two tables:   DROP table emp;  CREATE table emp as SELECT \* FROM employees WHERE 1=2;  SELECT \* FROM emp;  DESC emp;  INSERT INTO emp  SELECT \* FROM employees;  DROP table dept;  CREATE table dept as SELECT \* FROM departments WHERE 1=2;  SELECT \* FROM dept;  DESC dept;  INSERT INTO dept  SELECT \* FROM departments;  CREATE OR REPLACE VIEW dept\_sum\_vu  (name, minsal, maxsal, avgsal)  AS SELECT d.department\_name, MIN(e.salary),  MAX(e.salary),AVG(e.salary)  FROM emp e JOIN dept d  ON (e.department\_id = d.department\_id)  GROUP BY d.department\_name;    SELECT \* FROM dept\_sum\_vu; |







**Using the WITH CHECK OPTION Clause**

|  |
| --- |
| * You can ensure that DML operations performed on the view stay in the domain of the view by using the WITH CHECK OPTION clause:   DROP table emp;  CREATE table emp as SELECT \* FROM employees WHERE 1=2;  SELECT \* FROM emp;  DESC emp;  INSERT INTO emp  SELECT \* FROM employees;  CREATE OR REPLACE VIEW empvu20  AS SELECT \*  FROM emp  WHERE department\_id = 20  WITH CHECK OPTION CONSTRAINT empvu20\_ck ;     * Any attempt to INSERT a row with a department\_id other than 20, or to UPDATE the department number for any row in the view fails because it violates the WITH CHECK OPTION constrain.   UPDATE empvu20  SET department\_id = 10  WHERE employee\_id = 201; |

**Denying DML Operations**

|  |
| --- |
| * You can ensure that no DML operations occur by adding the WITH READ ONLY option to your view definition * Any attempt to perform a DML operation on any row in the view results in an Oracle server error.     DROP table emp;  CREATE table emp as SELECT \* FROM employees WHERE 1=2;  SELECT \* FROM emp;  DESC emp;  INSERT INTO emp  SELECT \* FROM employees;  CREATE OR REPLACE VIEW empvu10  (employee\_number, employee\_name, job\_title)  AS SELECT employee\_id, last\_name, job\_id  FROM emp  WHERE department\_id = 10  WITH READ ONLY ;    DELETE FROM empvu10  WHERE employee\_number = 200; |

**Removing a View**

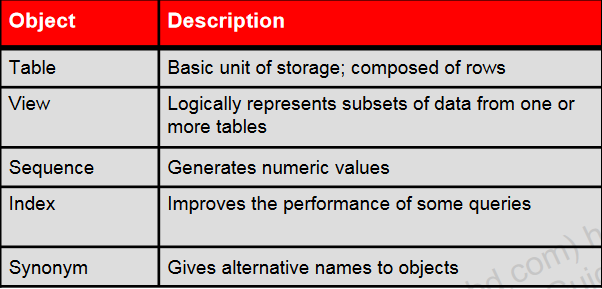
|  |
| --- |
| * You can remove a view without losing data because a view is based on underlying tables in the database.   DROP VIEW empvu80; |

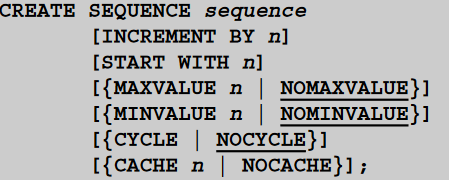
**Sequences**

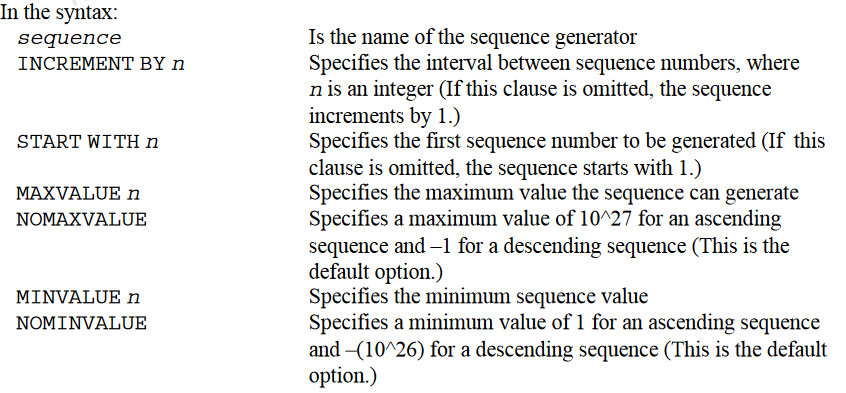
A sequence is a database object that creates integer values.

A sequence:

* Can automatically generate unique numbers
* Is a shareable object
* Can be used to create a primary key value
* Replaces application code
* Speeds up the efficiency of accessing sequence values when cached in memory







**Creating a Sequence**

|  |
| --- |
| * Create a sequence named DEPT\_DEPTID\_SEQ to be used for the primary key of the DEPT table. * Do not use the CYCLE option.   DROP table dept;  CREATE table dept as SELECT \* FROM departments WHERE 1=2;  SELECT \* FROM dept;  DESC dept;  INSERT INTO dept  SELECT \* FROM departments;  SELECT department\_id FROM dept;  CREATE SEQUENCE dept\_deptid\_seq  INCREMENT BY 10  START WITH 120  MAXVALUE 9999  NOCACHE  NOCYCLE; |

**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.

**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.

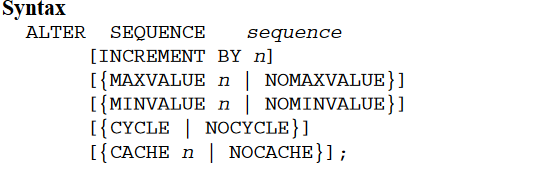
**Using a Sequence**

|  |
| --- |
| * Insert a new department named “Support” in location ID 2500:   DROP table dept;  CREATE table dept as SELECT \* FROM departments WHERE 1=2;  SELECT \* FROM dept;  DESC dept;  INSERT INTO dept  SELECT \* FROM departments;  SELECT department\_id FROM dept;  CREATE SEQUENCE dept\_deptid\_seq  INCREMENT BY 10  START WITH 120  MAXVALUE 9999  NOCACHE  NOCYCLE;  INSERT INTO dept(department\_id,  department\_name, location\_id)  VALUES (dept\_deptid\_seq.NEXTVAL,  'Support', 2500);     * View the current value for the DEPT\_DEPTID\_SEQ sequence:   SELECT dept\_deptid\_seq.CURRVAL  FROM dual; |

**Caching Sequence Values**

* Caching sequence values in memory gives faster access to those values.
* Gaps in sequence values can occur when:
  + - A rollback occurs
    - The system crashes
    - A sequence is used in another table

**Modifying a Sequence**



|  |
| --- |
| * Change the increment value, maximum value, minimum value, cycle option, or cache option:   ALTER SEQUENCE dept\_deptid\_seq  INCREMENT BY 20  MAXVALUE 999999  NOCACHE  NOCYCLE; |

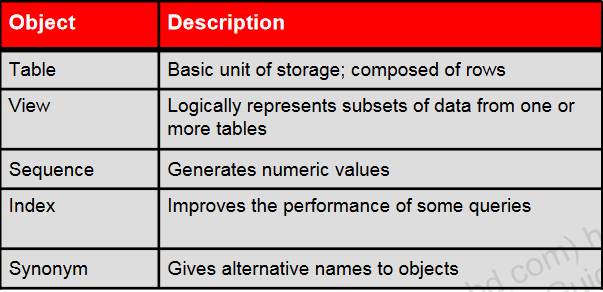
Guidelines for Modifying a Sequence

* 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;    ALTER SEQUENCE dept\_deptid\_seq  INCREMENT BY 20  MAXVALUE 999999  NOCACHE  NOCYCLE; |

**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 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.

An index:

* Is a schema object
* Can be used by the Oracle server to speed up the retrieval of rows by using a pointer
* Can reduce disk input/output (I/O) by using a rapid path access method to locate data quickly
* Is independent of the table that it indexes
* Is used and maintained automatically by the Oracle server

**How Are Indexes Created?**

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



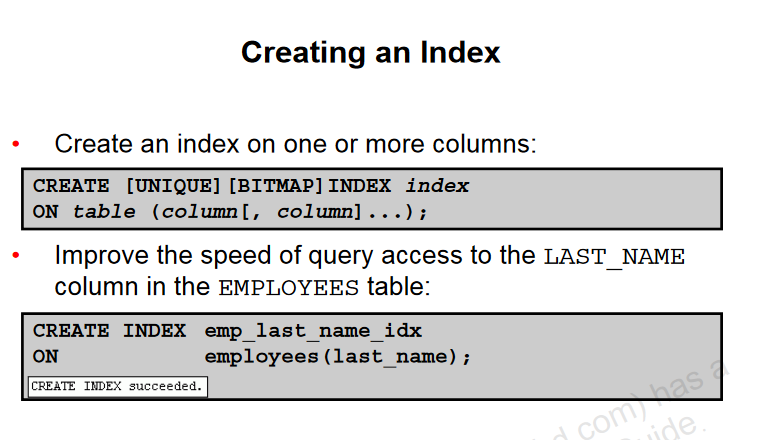
* Manually: Users can create nonunique indexes on columns to speed up access to the rows.

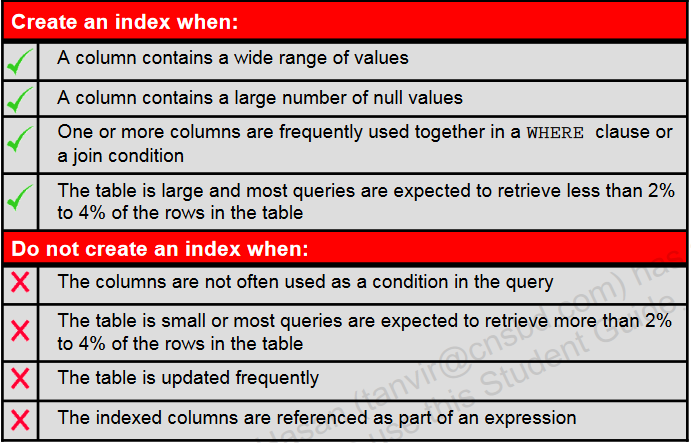


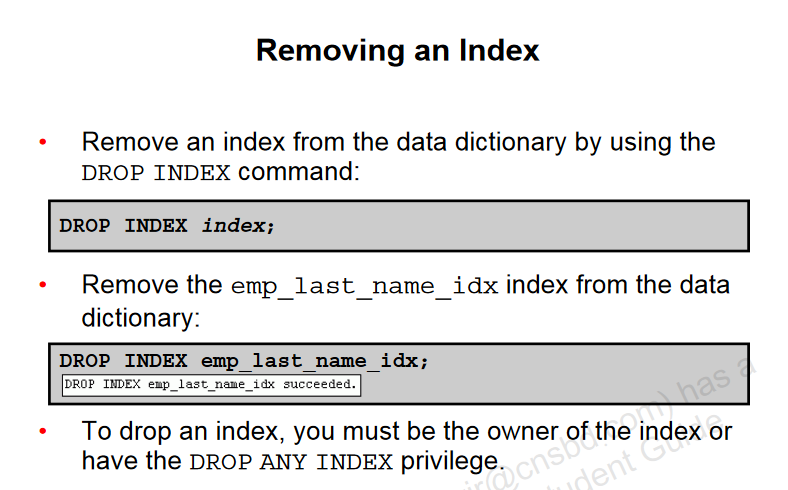
TYPE OF INDEX   
  
You can create two types of indexes.

1. 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.
2. 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.







**Synonyms**

