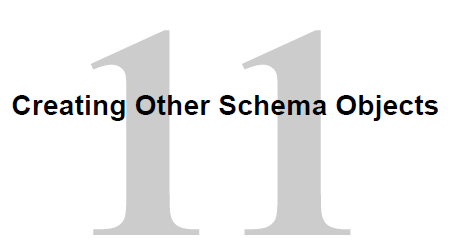
Les12-Index-Dictionary Views

Chapter 11



**Objectives**

After completing this lesson, you should be able to do the following:

• Create simple and complex views

• Retrieve data from views

• Create, maintain, and use sequences

• Create and maintain indexes

• Create private and public synonyms – maybe

VIEWS

11-3

**Lesson Agenda - VIEWS**

• Overview of views:

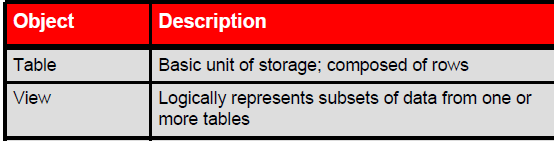
ADD, CHANGE DELETE views

– Creating, modifying, and retrieving data from a view

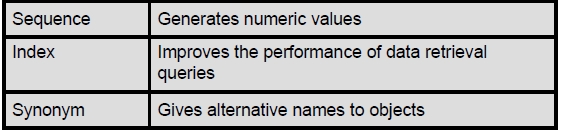
– Data manipulation language (DML) operations on a view

– Dropping a view

Database Objects



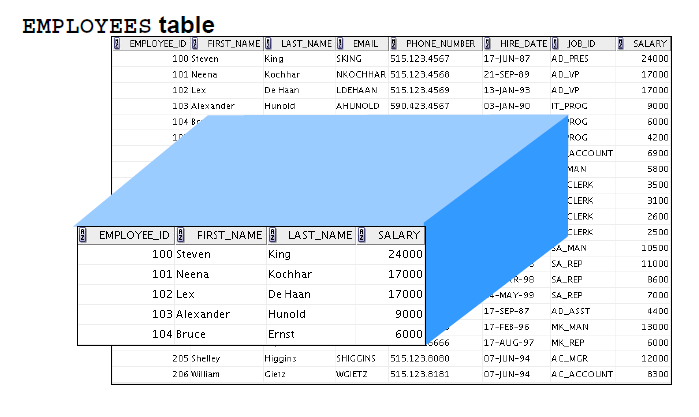
Other objects – covered later



What is a View

Already seen a view in DBS201

Original EMPLOYEES table has 8 columns



View has only 4 columns

and has 5 rows

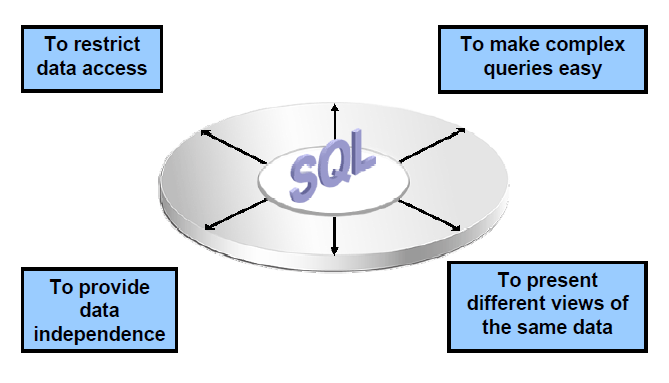
You can develop LOGICAL subsets of 1 or more tables by creating a view.

VIEW is a LOGICAL table and not a physical table with data. Therefore, has NO data

Views can be made also from other views. – Views of views

The view is stored as a SELECT statement in the data dictionary

Advantages to Views



payroll example

1 Restrict data access.

Payroll department needs access to employees pay records, but maybe the lower level payroll person should not see the President and other executive's compensation package. The view is accessible to the payroll clerk, but not the full tables.

2 Simplify queries \*\* major

The data can come from a number of tables via joins and subqueries. From that complex SQL a view is generated. There can be more than one view on the complex query and there can be sub views.

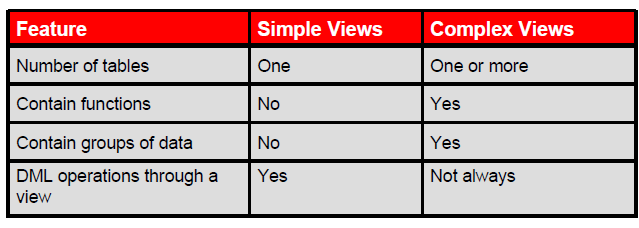
3 Views allow access to data based on their needs. Similar idea to #1

4 Present different views of same data

Views – Classification

Simple -- Complex

# Differences



**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 to the underlying tables

• 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

CREATE VIEW

Generic Syntax

Note that a subquery is embedded in the CREATE VIEW

**CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW *view***

**[(*alias* [, *alias*]...)]**

**AS *subquery***

**[WITH CHECK OPTION [CONSTRAINT *constraint*]]**

**[WITH READ ONLY [CONSTRAINT *constraint*]];**

(LIKE: CREATE TABLE …)

That subquery can be simple or contain a complex SELECT

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  Sometimes done during development of database and underlying tables may not have been created yet |
| NO FORCE | Default – creates view only if the tables exist |
| View | Name of the view |
| Alias | Alternate names for underlying expressions |
| subquery | SELECT etc… |
| 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.  No add etc… |

VIEW – Examples

One of the payroll people handles just employees in department 80. That person has need for a list of those employees and does not have a need for all the other.

**CREATE VIEW empvu80**

**AS SELECT employee\_id,**

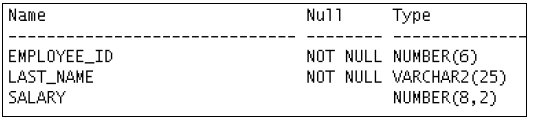
**last\_name,**

**salary**

**FROM employees**

**WHERE department\_id = 80;**

**DESCRIBE empvu80**



**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.

VIEW – Examples – with aliases

# Column aliases

**CREATE VIEW salvu50**

**AS SELECT employee\_id ID\_NUMBER,**

**last\_name NAME,**

**salary\*12 ANN\_SALARY**

**FROM employees**

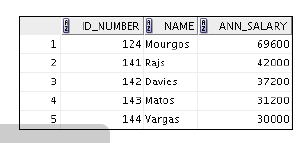
**WHERE department\_id = 50;**

View – Retrieving Data

**SELECT \***

**FROM salvu50;**

Results:



You can treat the view like a table. In this example only selecting 2 of the 3 columns in the view to display.

**SELECT ID\_NUMBER, NAME**

**FROM salvu50;**

MODIFY – CHANGE a VIEW

Requires

CREATE OR REPLACE 🡸 it saves deleting and creating – and re granting privileges

Example: Modify the previous empvu80 to add aliases

**CREATE OR REPLACE VIEW empvu80**

**(id\_number, name, sal, department\_id)**

**AS SELECT employee\_id,**

**first\_name || ' ' || last\_name,**

**salary,**

**department\_id**

**FROM employees**

**WHERE department\_id = 80;**

NOTE:

Alias is listed in SAME order as subquery

VIEWS – COMPLEX – Example

Why complex – uses group by and multiple tables

**CREATE OR REPLACE VIEW dept\_sum\_vu**

**( name, minsal, maxsal, avgsal)**

**AS SELECT d.department\_name,**

**MIN (e.salary),**

**MAX (e.salary),**

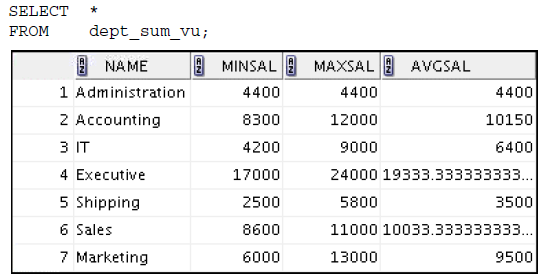
**AVG (e.salary)**

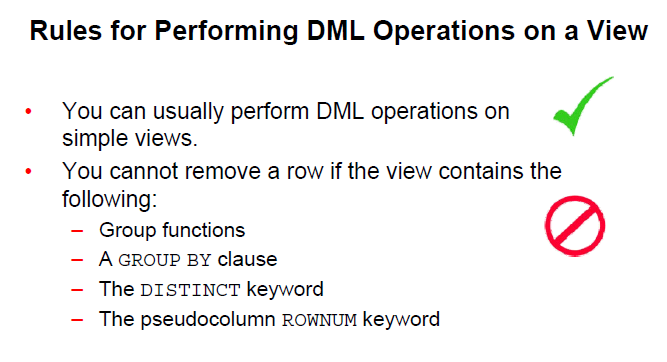
**FROM employees e JOIN departments d**

**ON ( e.department\_id = department\_id)**

**GROUP BY d.department\_name;**

Look at results:





Look at specifics

MODIFY DATA

You cannot modify data in a view if it contains:

• Group functions

• A GROUP BY clause

• The DISTINCT keyword

• The pseudo column ROWNUM keyword – covered later

• Columns defined by expressions

ADD DATA

• All the above

• NOT NULL columns in the base tables that are not selected by the view

- Why – NN requires data. If view doesn't have that column then constraint activates

When using a view need all columns to be MANDATORY in order for ADD to work

WITH CHECK OPTION

11-17

# Example:

**CREATE OR REPLACE VIEW empvu20**

**AS SELECT \***

**FROM employees**

**WHERE department\_id = 20**

**WITH CHECK OPTION CONSTRAINT empvu20\_ck ;**

This constraint stops

-- STOPS any insert of a row that does not use department 20

-- Any update to department number in the view

Example:

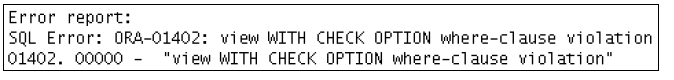
Can change salary as there is no constraint

UPDATE empvu20

SET department\_id = 10

WHERE employee\_id = 201;

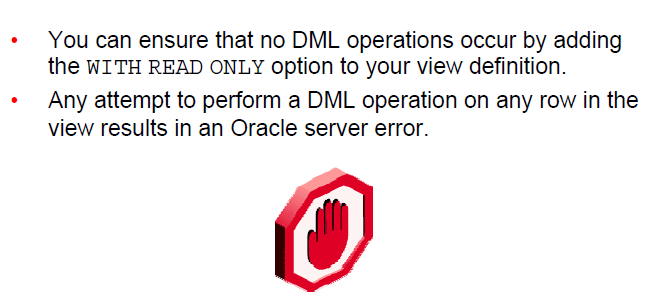
No rows are updated due to this error message



WHY: If you change the department number to 10 for a row in the view (the employee 201) then the view cannot see the employee as it is not department 20.

The WITH CHECK OPTION won't allow the change

STOPPING ALL DML



**CREATE OR REPLACE VIEW empvu10**

**(employee\_number, employee\_name, job\_title)**

**AS SELECT employee\_id, last\_name, job\_id**

**FROM employees**

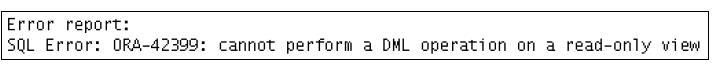
**WHERE department\_id = 10**

**WITH READ ONLY ;**

Example DML:

**DELETE FROM empvu10**

**WHERE employee\_number = 200;**



REMOVING A VIEW

11-20

Removing a view does not remove the data

**DROP VIEW empvu80;**

SEQUENCE

11-23

# Overview of sequences:

– Creating, using, and modifying a sequence

– Cache sequence values

– NEXTVAL and CURRVAL pseudo columns

A sequence is a database object that creates integer values.

You can create sequences and then use them to **generate numbers**.

- Automatic number like order numbers

A sequence:

• Can automatically generate unique numbers

• Is a shareable object – probably wouldn't see this

• Can be used to create a primary key value

• Replaces application code

• Speeds up the efficiency of accessing sequence values

- When cached in memory

CREATE SEQUENCE Statement

Generic syntax

**CREATE SEQUENCE *sequence 🡨 name of sequence***

**[INCREMENT BY *n*] 🡨 specifies increment value**

**[START WITH *n*] 🡨 Starting (default 1 if omitted**

**[{MAXVALUE *n* | NOMAXVALUE}] 🡨 maximum value – default is nomax**

**[{MINVALUE *n* | NOMINVALUE}] 🡨 this is default if not stated**

**[{CYCLE | NOCYCLE}] 🡨 allows recycling of numbers–reuse**

**[{CACHE *n* | NOCACHE}]; 🡨 allows caching x values-faster**

\* Never use CYCLE if using it to generate Primary Keys

Here is a sequence to use as PK

**CREATE SEQUENCE dept\_deptid\_seq 🡨 note the naming convention**

**INCREMENT BY 10**

**START WITH 120**

**MAXVALUE 9999**

**NOCACHE**

**NOCYCLE;**

NEXTVAL and CURRVAL Pseudo columns

• NEXTVAL

- used to extract successive sequence number

- returns the next available sequence value.

It returns a unique value every time it is referenced, even for different users.

Specify NEXTVAL and the sequence name

• CURRVAL obtains the current sequence value.

• NEXTVAL must be issued for that sequence before CURRVAL contains a value.

SEE EXAMPLES

Using a Sequence

**INSERT INTO departments**

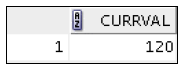
**(department\_id, department\_name, location\_id)**

**VALUES (dept\_deptid\_seq.NEXTVAL, 'Support', 2500);**

View current value of sequence

**SELECT dept\_deptid\_seq.CURRVAL**

**FROM dual;**



**C**aching Sequence Values

• Caching sequence values in memory gives faster access to those values.

- Less going out to disk to retrieve the latest number an update it

• Gaps in sequence values can occur when:

Often an exam question

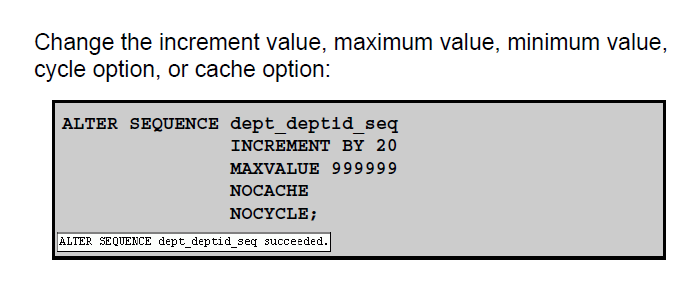
– A rollback occurs

– The system crashes

– A sequence is used in another table (rare)

**\* What is in cache is lost**

Modifying a Sequence



Guidelines:

- 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

**DROP SEQUENCE dept\_deptid\_seq;**

INDEX

**Is used by the Oracle server to speed up the retrieval of rows by using a pointer.**

If you do not have an index on the column then a FULL table scan is required.

**Can reduce I/O**

The purpose of an index is to reduce I/O in accessing required data from disk

**Independent of the table**

By being independent of the underlying table the index can be dropped or created with no effect on the data in the table or other indexes.

NOTE: Dropping a table will also drop any corresponding indexes.

**Maintained automatically by Oracle Server**

No programmer or user activity is required to maintain the index once it is created.

DBA usually creates and decides on index

# Which columns to index?

EXAMPLE:

Consider EMPLOYEE table

Would you index last name YES

job\_id not likely as it repeats a lot

email maybe – don't know

Depends on if it is searched a lot

salary only if use it as a search condition a lot

Create and DROP index

**How Are Indexes Created?**

Automatically:

A unique index is created automatically when you define a

- PRIMARY KEY or

- UNIQUE constraint in a table definition.

Oracle handles the indexing on primary and unique constraints

Manually:

Developers can create nonunique indexes on other columns to speed up access to rows.

CREATE INDEX

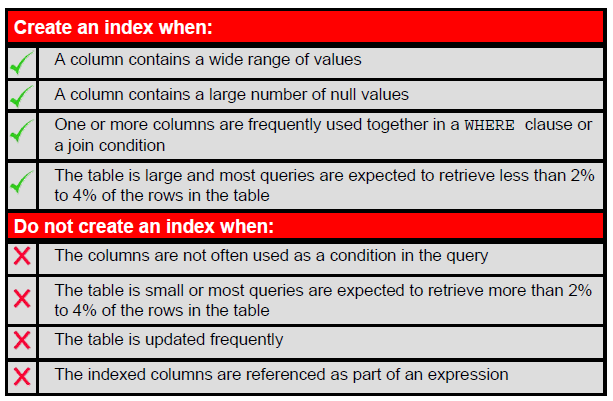
**CREATE INDEX emp\_last\_name\_idx 🡨 note naming convention**

**ON employees (last\_name);**

GUIDELINES:

Last name

Gets to values quicker



**SOME RULES ON INDEXES**

##### **1 The column is used often in a where clause and the table is large.**

Example 1:

In the lost or stolen credit card department, many callers do not have the credit card number handy. The company uses the name to seek your information. With millions of credit card users an index would be faster.

Example 2:

A customer phones to place an order for product. The customer number is your company’s reference number and often is unknown to the caller. Again a search by name is often used.

##### **2 The table is very large and most retrievals display a small amount of data.**

The above example and reporting/queries that retrieve 1 to 5% of the data.

###### **TRADE-OFF**

More indices do no speed the processing overall.

-- For every index there is overhead activities to maintain the index.

Null values are not included in the index

REMOVE INDEX

**DROP INDEX emp\_last\_name\_idx;**

**Can't be modified**

**Must drop and add**

Must be owner to be able to DROP

Or have privilege

What happens to the data - NOTHING

What happens to an application - NOTHING – may run slower

What is effected – just the speed of searches

DROP SEQUENCE -- Data not effected

-- Code might be if it uses the sequence

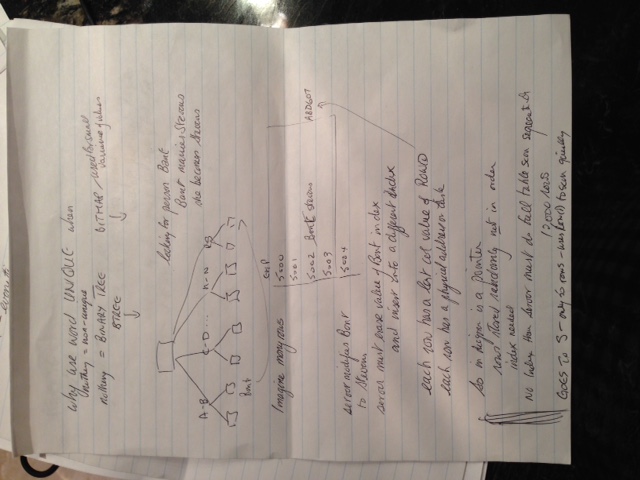
Code never has an index in it

NOTE:

Drop a table, indexes and sequence automatically dropped

Views remain

Investigate: Applying 2 indices on Last name and first name. Several methods.



REVERSE

used only on binary tree

if have order numbers

10000019

10000020

10000021

When searching as a human you soon twig to looking at the end to speed up processing

So indexing on the end numbers or in reverse order improves search

91000001

02000001

12000001

If searching order numbers you usually search last few months and not several years

SYNONYM

# PURPOSE

**1 To shorten lengthy object names**

**2 Refer to table owned by another user – really the same as 1**

# CREATING SYNONYM

**CREATE SYNONYM d\_sum**

**FOR dept\_sum\_vu;**

# REMOVING VIEW

**DROP SYNONYM d\_sum;**

# PUBLIC SYNONYM – created by DBA

**CREATE PUBLIC SYNONYM STUDLIST**

**FOR registration.STUDENT;**

Allows access to table STUDENT owned by user REGISTRATION**.**