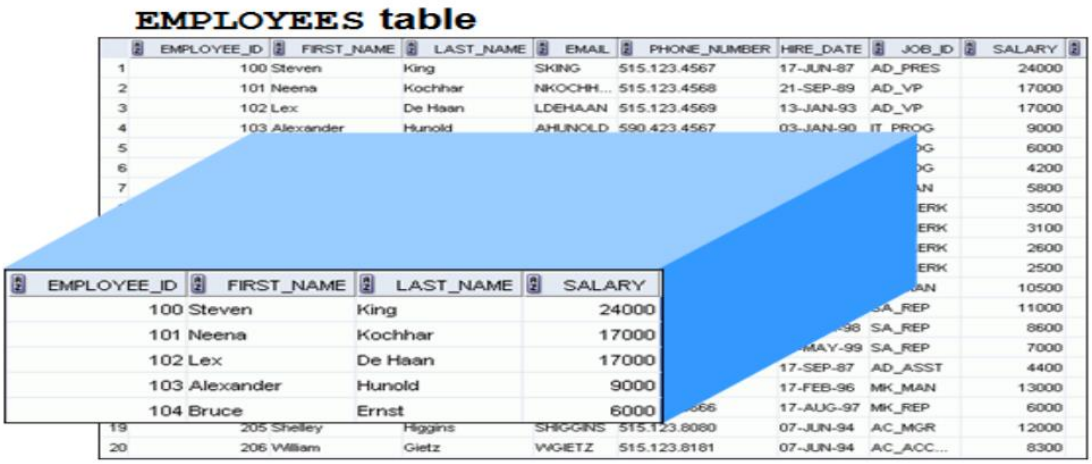
DB Lab 08: Views, Indexes, Set operations



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. The syntax of view is as follows.

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
 AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

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

Example:

Create the EMPVU80 view, which contains details of the employees in department 80:

```
CREATE VIEW empvu80
 AS SELECT employee_id, last_name, salary
   FROM   employees
   WHERE  department_id = 80;
```

Output:

```
CREATE VIEW succeeded.
```

This example creates a view that contains the employee number, last name, and salary for each employee in department 80.

You can display the structure of the view by using the DESCRIBE command.

DESCRIBE empvu80;

Example:

Create a view by using column aliases in the subquery

```
CREATE VIEW salvu50
 AS SELECT  employee_id ID_NUMBER, last_name NAME,
        salary*12 ANN_SALARY
   FROM    employees
   WHERE   department_id = 50;
```

Output:

```
CREATE VIEW succeeded.
```

You can control the column names by including column aliases in the subquery.

The above example 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.

Retrieving Data from a View

You can retrieve data from a view as you would from any table. You can display either the contents of the entire view or just specific rows and columns.

Example:

```
SELECT *
FROM   salvu50;
```

Output:

	ID_NUMBER	NAME	ANN_SALARY
1	124	Mourgos	69600
2	141	Rajs	42000
3	142	Davies	37200
4	143	Matos	31200
5	144	Vargas	30000

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.

Example:

```
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;
```

Output:

```
CREATE OR REPLACE VIEW succeeded.
```

Creating a Complex View

The example creates a complex view of department names, minimum salaries, maximum salaries, and the average salaries by department. Note that alternative names have been specified for the view. This is a requirement if any column of the view is derived from a function or an expression.

Example:

```
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 = d.department_id)
  GROUP BY d.department_name;
```

You can view the structure of the view by using the DESCRIBE command. Display the contents of the view by issuing a SELECT statement.

Rules for Performing DML Operations on a View

- You can perform DML operations on data through a view if those operations follow certain rules.
- You can remove a row from a view unless it contains any of the following:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM keyword
  - You cannot modify data in a view if it contains:
    - Group functions
    - A GROUP BY clause
    - The DISTINCT keyword
    - The pseudocolumn ROWNUM keyword
    - Columns defined by expressions
  - You cannot add data through a view if the view includes:
    - Group functions
    - A GROUP BY clause
    - The DISTINCT keyword
    - The pseudocolumn ROWNUM keyword
    - Columns defined by expressions
    - NOT NULL columns in the base tables that are not selected by the view

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.

Example:

```
CREATE OR REPLACE VIEW empvu20
AS SELECT      *
  FROM    employees
  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 constraint. Like consider the following query.

```
UPDATE empvu20
SET   department_id = 10
WHERE  employee_id = 201;
```

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

You can ensure that no DML operations occur on your view by creating it with the WITH READ ONLY option.

Example:

```
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 ;
```

Output:

```
CREATE OR REPLACE VIEW succeeded.
```

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. On the other hand, 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.

Syntax:

```
DROP VIEW view;
```

DROP VIEW empvu80;

Output:

```
DROP VIEW empvu80 succeeded.
```

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, then 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.

Syntax:

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

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

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.

Syntax:

```
DROP INDEX index;
```

Example:

DROP INDEX emp_last_name_idx;

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

Synonyms

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.

Syntax:

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

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.

Example:

CREATE SYNONYM d_sum

FOR dept_sum_vu;

Output:

```
CREATE SYNONYM succeeded.
```

Removing a Synonym

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

Example:

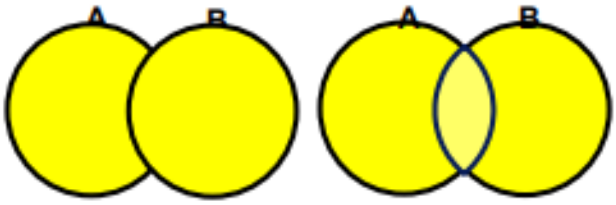
DROP SYNONYM d_sum;

Output:

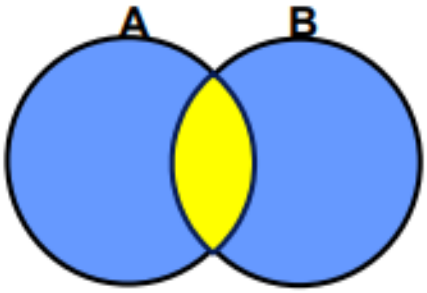
```
DROP SYNONYM d_sum succeeded.
```

Set Operators

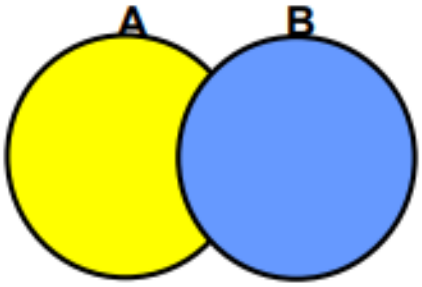
Set operators combine the results of two or more component queries into one result. Queries containing set operators are called *compound queries*.



UNION/UNION ALL



INTERSECT



MINUS

Set Operator Guidelines

- The expressions in the SELECT lists of the queries must match in number and data type. Queries that use UNION, UNION ALL, INTERSECT, and MINUS operators in their WHERE clause must have the same number and data type of columns in their SELECT list. The data type of the columns in SELECT list of the queries in the compound query may not be exactly the same. The column in second query must be in the same data type group (such as numeric or character) as the corresponding column in the first query.
- Set operators can be used in subqueries.
- You should use parentheses to specify the order of evaluation in queries that use the INTERSECT operator with other set operators. This ensures compliance with emerging SQL standards that will give the INTERSECT operator greater precedence than the other set operators.

The Oracle Server and Set Operators

When a query uses set operators, the Oracle server eliminates duplicate rows automatically except in the case of the UNION ALL operator. The column names in the output are decided by the column list in the first SELECT statement. By default, the output is sorted in ascending order of the first column of the SELECT clause.

The corresponding expressions in the SELECT lists of the component queries of a compound query must match in number and data type. If component queries select character data, the data type of the return values is determined as follows.

- If both queries select values of CHAR data type, of equal length, then the returned values have the CHAR data type of that length. If the queries select values of CHAR with different lengths, then the returned value is VARCHAR2 with the length of the larger CHAR value.
- If either or both of the queries select values of VARCHAR2 data type, then the returned values have the VARCHAR2 data type.

If component queries select numeric data, then the data type of the return values is determined by numeric precedence. If all queries select values of the NUMBER type, then the returned values have the NUMBER data type. In queries using set operators, the Oracle server does not perform implicit conversion across data type groups. Therefore, if the corresponding expressions of component queries resolve to both character data and numeric data, the Oracle server returns an error.

Tables Used in This Lab

Two tables are used in this lab. They are the EMPLOYEES table and the JOB_HISTORY table.

You are already familiar with the EMPLOYEES table that stores employee details such as a unique identification number, email address, job identification (such as ST_CLERK, SA_REP, and so on), salary, manager and so on.

Some of the employees have been with the company for a long time and have switched to different jobs. This is monitored using the JOB_HISTORY table. When an employee switches jobs, the details of the start date and end date of the former job, the job_id (such as ST_CLERK, SA_REP, and so on), and the department are recorded in the JOB_HISTORY table.

UNION Operator

The UNION operator returns all rows that are selected by either query. Use the UNION operator to return all rows from multiple tables and eliminate any duplicate rows.

Guidelines

- The number of columns being selected must be the same.
- The data types of the columns being selected must be in the same data type group (such as numeric or character).
- The names of the columns need not be identical.
- UNION operates over all of the columns being selected.
- NULL values are not ignored during duplicate checking.
- By default, the output is sorted in ascending order of the columns of the SELECT clause.

The UNION operator eliminates any duplicate records. If records that occur in both the EMPLOYEES and the JOB_HISTORY tables are identical, the records are displayed only once.

Example:

```
SELECT employee_id, job_id
FROM   employees
UNION
SELECT employee_id, job_id
FROM   job_history;
```

Output:

	EMPLOYEE_ID	JOB_ID
1	100	AD_PRES
2	101	AC_ACCOUNT

...

22	200	AC_ACCOUNT
23	200	AD_ASST
24	201	MK_MAN

...

Observe in the output shown in the example that the record for the employee with the EMPLOYEE_ID 200 appears twice because the JOB_ID is different in each row.

UNION ALL Operator

Use the UNION ALL operator to return all rows from multiple queries.

Guidelines

The guidelines for UNION and UNION ALL are the same, with the following two exceptions that pertain to UNION ALL: Unlike UNION, duplicate rows are not eliminated and the output is not sorted by default.

Example:

```
SELECT employee_id, job_id, department_id
FROM   employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM   job_history
ORDER BY employee_id;
```

Output:

	EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	100	AD_PRES	90
...			
16	144	ST_CLERK	50
17	149	SA_MAN	80
18	174	SA_REP	80
19	176	SA_REP	80
20	176	SA_MAN	80
21	176	SA_REP	80
22	178	SA_REP	(null)
...			
30	206	AC_ACCOUNT	110

In the example, 30 rows are selected. The combination of the two tables totals to 30 rows. The UNION ALL operator does not eliminate duplicate rows. UNION returns all distinct rows selected by either query. UNION ALL returns all rows selected by either query, including all duplicates.

INTERSECT Operator

Use the INTERSECT operator to return all rows that are common to multiple queries.

Guidelines

- The number of columns and the data types of the columns being selected by the SELECT statements in the queries must be identical in all the SELECT statements used in the query. The names of the columns, however, need not be identical.
- Reversing the order of the intersected tables does not alter the result.
- INTERSECT does not ignore NULL values.

Example:

```
SELECT employee_id, job_id
FROM   employees
INTERSECT
SELECT employee_id, job_id
FROM   job_history;
```

Output:

	EMPLOYEE_ID	JOB_ID
1	176	SA_REP
2	200	AD_ASST

In the above example, the query returns only those records that have the same values in the selected columns in both tables.

MINUS Operator

Use the MINUS operator to return all distinct rows selected by the first query, but not present in the second query result set (the first SELECT statement MINUS the second SELECT statement).

Example:

```
SELECT employee_id
FROM   employees
MINUS
SELECT employee_id
FROM   job_history;
```

Output:

	EMPLOYEE_ID
1	100
2	103
3	104
4	107
5	124
...	
14	205
15	206

Matching the SELECT Statement:

Example:

The EMPLOYEES and JOB_HISTORY tables have several columns in common (for example, EMPLOYEE_ID, JOB_ID, and DEPARTMENT_ID). But what if you want the query to display the employee ID, job ID, and salary using the UNION operator, knowing that the salary exists only in the EMPLOYEES table?

The code example in the slide matches the EMPLOYEE_ID and JOB_ID columns in the EMPLOYEES and JOB_HISTORY tables. A literal value of 0 is added to the

JOB_HISTORY SELECT statement to match the numeric SALARY column in the EMPLOYEES SELECT statement.

```
SELECT employee_id, job_id,salary
FROM   employees
UNION
SELECT employee_id, job_id,0
FROM   job_history;
```

Output:

	EMPLOYEE_ID	JOB_ID	SALARY
1	100	AD_PRES	24000
2	101	AC_ACCOUNT	0
3	101	AC_MGR	0
4	101	AD_VP	17000
5	102	AD_VP	17000

...

29	205	AC_MGR	12000
30	206	AC_ACCOUNT	8300

The above example matches the EMPLOYEE_ID and JOB_ID columns in the EMPLOYEES and JOB_HISTORY tables. A literal value of 0 is added to the JOB_HISTORY SELECT statement to match the numeric SALARY column in the EMPLOYEES SELECT statement.

In the results shown, each row in the output that corresponds to a record from the JOB_HISTORY table contains a 0 in the SALARY column.

Using the ORDER BY Clause in Set Operations

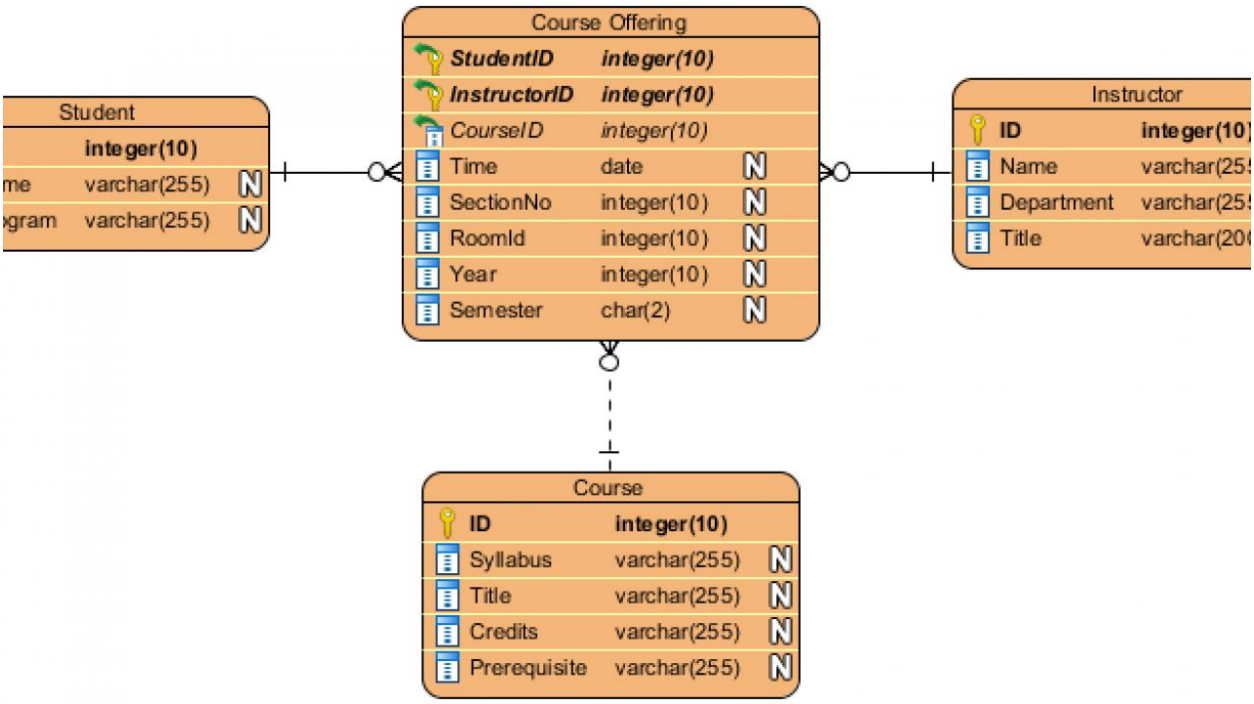
The ORDER BY clause can be used only once in a compound query. If used, the ORDER BY clause must be placed at the end of the query. The ORDER BY clause accepts the column name or an alias. By default, the output is sorted in ascending order in the first column of the first SELECT query.

Note: The ORDER BY clause does not recognize the column names of the second SELECT query. To avoid confusion over column names, it is a common practice to ORDER BY column positions.

Lab Tasks:

Task 01: (10 marks)

Create DDL and DML language for the ERD below. Insert at least 3 records in each table after applying all possible constraints. Create a save point SAVEDDL after creation of table and 2nd save point SAVEDML after insertion of records. Save all the changes permanent.



Task 2: Create a view to store list of department IDs for departments that do not contain the job ID ST_CLERK.

Task 3: The HR department needs a list of countries that have no departments located in them. Display the country ID and the name of the countries. Use the set operators to create this report.