DATABASE SYSTEMS LAB

Course Code	: 30102422
Credit Hours	: 1
Prerequisite	: 30102421

Instructor Information

Name	:Dr. Jihad Nader				
Office No.	**B17 F4 R8**				
Tel (Ext)	**N/A**				
E-mail	jihadnader@bau.edu.jo				
Office Hours	Thur ,Tues (10-11) mon, wed (1-2)				
Class Times	Building	Day	Start Time	End Time	Room No.
	-	Tuesday	11	2	Online

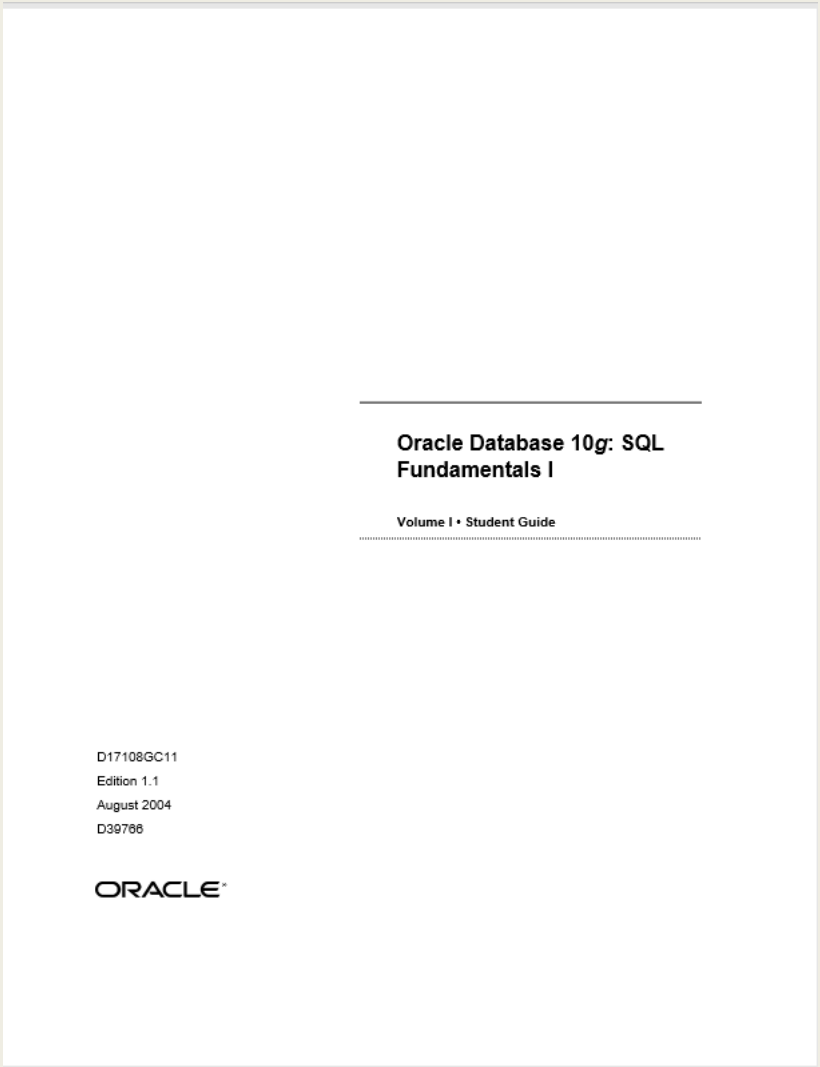
Course Description:

This Lab. practices the concepts introduced in the Database systems course using Oracle Database. The students are expected to implement a database project for some problem.

Course Title: Database Systems Lab **Credit Hour(1)**

[Pre-req. Course Code(30102421)]

Textbook: *Oracle Database 10g: SQL Fundamentals I, Volume I • Student Guide*



Oracle Database 10*g*: SQL Fundamentals I

Volume I • Student Guide

D17108GC11
Edition 1.1
August 2004
D39766

ORACLE

COURSE OBJECTIVES:

Upon completion of this course, students will have gained knowledge of the DBMS (Oracle) concepts and the ability to:

- Understand the concepts of relational databases and the Oracle Database 10g database technology.
- Use the powerful SQL programming language and its features.
- Identify features of Relational Database Management System (RDBMS).
- Categorize the main database objects
- Understand how constraints are created at the time of table creation
- Describe each data manipulation language (DML) statement
- List the capabilities of SQL SELECT statements
- Write SELECT statements to access data from more than one table using equijoins and non-equijoins
- Employ SQL functions to generate and retrieve customized data
- Identify when a subquery can help solve a question
- Write subqueries when a query is based on unknown values
- Use a set operator to combine multiple queries into a single query

COURSE SYLLABUS

Week	Course Topic	Notes
Week 1	Creating and Managing Tables: - Database Objects - Naming Conventions - The Create Table Statement - Creating a Table by Using a Subquery - Querying the Data Dictionary - The Alter Table Statement - Truncating a Table - Adding Comments to a Table	

Week 2	Including Constraints
	- Defining Constraints
	  o The Not Null Constraint
	  o The Unique Constraint
	  o The Primary Key Constraint
	  o The Foreign Key Constraint
	  o The Check Constraint
	- Adding a Constraint
	- Dropping a Constraint
	- Enabling and Disabling Constraints
	- Viewing Constraints	

Week 3	Manipulating Data - Data Manipulating Language. - The Insert Statement - Copying Rows from another Table - The Update Statement - The Delete Statement - Database Transactions - Commit and Rollback Statements	
Week 4	Writing Basic SQL Statements - Selecting Specific Columns - Arithmetic Expressions - Concatenation Operator - Using Column Aliases - Eliminating Duplicate Rows	

COURSE SYLLABUS

Week	Course Topic	Notes
Week 5	Restricting and Sorting Data - Where Clause - Comparison Operators - Special Operators - Logical Operator (And, Or, Not) - Order By Clause	
Week 6	Displaying Data from Multiple Tables - Cartesian Product. - Types of Joins - Table Aliases.	
Week 7	Single-Row Functions - Character Functions. - Number Functions - Date Functions ▪	
Week 8	Midterm Exam	Midterm Exam
Week 9	Project Proposal	
Week 10	Single-Row Functions - Conversion Functions - General Functions	

COURSE SYLLABUS

Week	Course Topic	Notes
Week 11	Aggregating Data using Group Functions - Types of Group Functions (AVG, SUM, MAX, MIN, COUNT). - Creating Groups of data: Group By Clause. - Excluding Group Results: Having Clause. - Nested Group Functions	
Week 12	Subqueries - Types of Subqueries ▪ Single-Row Subqueries ▪ Multiple-Row Subqueries	
Week 13	Multiple-Column Subqueries - Column Comparisons - Null Values in a subquery - Using a subquery in the From Clause	
Week 14	Using the Set Operators - Union / Union All - Intersect - Minus	
Week 15	Project Discussion	
Week 16	Final Exam	Final Exam

Week 6

Chapter 5:

Displaying Data from Multiple Tables

DISPLAYING DATA FROM MULTIPLE TABLES

Objectives

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

- *Write `SELECT` statements to access data from more than one table using equijoins and nonequijoins*
- *Join a table to itself by using a self-join*
- *View data that generally does not meet a join condition by using outer joins*
- *Generate a Cartesian product of all rows from two or more tables*

Obtaining Data from Multiple Tables

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
200	10	Administration
201	20	Marketing
202	20	Marketing
...		
102	90	Executive
205	110	Accounting
206	110	Accounting

Types of Joins

Joins that are compliant with the SQL:1999 standard include the following:

- *Cross joins*
- *Natural joins*
- *`USING` clause*
- *Full (or two-sided) outer joins*
- *Arbitrary join conditions for outer joins*

Joining Tables Using SQL:1999 Syntax

Use a join to query data from more than one table:

```
SELECT   table1.column, table2.column
FROM     table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON (table1.column_name = table2.column_name)]|
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)]|
[CROSS JOIN table2];
```

Creating Natural Joins

- *The `NATURAL JOIN` clause is based on all columns in the two tables that have the same name.*
- *It selects rows from the two tables that have equal values in all matched columns.*
- *If the columns having the same names have different data types, an error is returned.*

Retrieving Records with Natural Joins

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;
```

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
60	IT	1400	Southlake
50	Shipping	1500	South San Francisco
10	Administration	1700	Seattle
90	Executive	1700	Seattle
110	Accounting	1700	Seattle
190	Contracting	1700	Seattle
20	Marketing	1800	Toronto
80	Sales	2500	Oxford

8 rows selected.

Creating Joins with the `USING` Clause

- *If several columns have the same names but the data types do not match, the `NATURAL JOIN` clause can be modified with the `USING` clause to specify the columns that should be used for an equijoin.*
- *Use the `USING` clause to match only one column when more than one column matches.*
- *Do not use a table name or alias in the referenced columns.*
- *The `NATURAL JOIN` and `USING` clauses are mutually exclusive.*

Joining Column Names

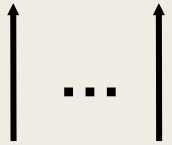
EMPLOYEES

EMPLOYEE_ID	DEPARTMENT_ID
200	10
201	20
202	20
124	50
141	50
142	50
143	50
144	50
103	60
104	60
107	60
149	80
174	80
176	80

...

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
60	IT
60	IT
60	IT
80	Sales
80	Sales
80	Sales



...

Foreign key **Primary key**

Retrieving Records with the USING Clause

```
SELECT employees.employee_id, employees.last_name,
       departments.location_id, department_id
FROM   employees JOIN departments
USING (department_id) ;
```

EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
200	Whalen	1700	10
201	Hartstein	1800	20
202	Fay	1800	20
124	Mourgos	1500	50
141	Rajs	1500	50
142	Davies	1500	50
144	Vargas	1500	50
143	Matos	1500	50

...

19 rows selected.

Qualifying Ambiguous Column Names

- *Use table prefixes to qualify column names that are in multiple tables.*
- *Use table prefixes to improve performance.*
- *Use column aliases to distinguish columns that have identical names but reside in different tables.*
- *Do not use aliases on columns that are identified in the `USING` clause and listed elsewhere in the SQL statement.*

Using Table Aliases

- *Use table aliases to simplify queries.*
- *Use table aliases to improve performance.*

```
SELECT e.employee_id, e.last_name,
       d.location_id, department_id
FROM   employees e JOIN departments d
USING (department_id) ;
```

Creating Joins with the `ON` Clause

- *The join condition for the natural join is basically an equijoin of all columns with the same name.*
- *Use the `ON` clause to specify arbitrary conditions or specify columns to join.*
- *The join condition is separated from other search conditions.*
- *The `ON` clause makes code easy to understand.*

Retrieving Records with the ON Clause

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id);
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500

...

19 rows selected.

Self-Joins Using the ON Clause

EMPLOYEES (WORKER)

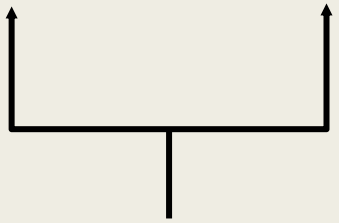
EMPLOYEE_ID	LAST_NAME	MANAGER_ID
100	King	
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103
107	Lorentz	103
124	Mourgos	100

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos

...



MANAGER_ID in the WORKER table is equal to EMPLOYEE_ID in the MANAGER table.

Self-Joins Using the ON Clause

```
SELECT e.last_name emp, m.last_name mgr
FROM   employees e JOIN employees m
ON    (e.manager_id = m.employee_id);
```

EMP	MGR
Hartstein	King
Zlotkey	King
Mourgos	King
De Haan	King
Kochhar	King
...	

19 rows selected.

Applying Additional Conditions to a Join

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
AND    e.manager_id = 149 ;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
174	Abel	80	80	2500
176	Taylor	80	80	2500

Creating Three-Way Joins with the ON Clause

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
103	Southlake	IT
104	Southlake	IT
107	Southlake	IT
124	South San Francisco	Shipping
141	South San Francisco	Shipping
142	South San Francisco	Shipping
143	South San Francisco	Shipping
144	South San Francisco	Shipping

...

19 rows selected.

Nonequijoins

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600

...

20 rows selected.

JOB_GRADES

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

Salary in the EMPLOYEES table must be between lowest salary and highest salary in the JOB_GRADES table.

Retrieving Records with Nonequijoins

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON     e.salary
       BETWEEN j.lowest_sal AND j.highest_sal;
```

LAST_NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Whalen	4400	B
Hunold	9000	C
Ernst	6000	C

...

20 rows selected.

Outer Joins

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

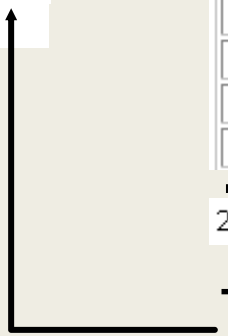
8 rows selected.

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
60	Hunold
60	Ernst
60	Lorentz
50	Mourgos
50	Rajs
50	Davies
50	Matos
50	Vargas
80	Zlotkey

...

20 rows selected.



There are no employees in department 190.

INNER Versus OUTER Joins

- *In SQL:1999, the join of two tables returning only matched rows is called an inner join.*
- *A join between two tables that returns the results of the inner join as well as the unmatched rows from the left (or right) tables is called a left (or right) outer join.*
- *A join between two tables that returns the results of an inner join as well as the results of a left and right join is a full outer join.*

LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e LEFT OUTER JOIN departments d
ON   (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		

20 rows selected.

RIGHT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
Davies	50	Shipping
...		
Kochhar	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
	190	Contracting

20 rows selected.

FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON   (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		
	190	Contracting

21 rows selected.

Cartesian Products

- *A Cartesian product is formed when:*
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- *To avoid a Cartesian product, always include a valid join condition.*

Generating a Cartesian Product

EMPLOYEES (20 rows)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

20 rows selected.

DEPARTMENTS (8 rows)

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

8 rows selected.

Cartesian product:
20 x 8 = 160 rows

EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
100	90	1700
101	90	1700
102	90	1700
103	60	1700
104	60	1700
107	60	1700
...		

160 rows selected.

Creating Cross Joins

- *The `CROSS JOIN` clause produces the cross-product of two tables.*
- *This is also called a Cartesian product between the two tables.*

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments ;
```

LAST_NAME	DEPARTMENT_NAME
King	Administration
Kochhar	Administration
De Haan	Administration
Hunold	Administration

...

160 rows selected.

Summary

In this lesson, you should have learned how to use joins to display data from multiple tables by using:

- *Equijoins*
- *Nonequijoins*
- *Outer joins*
- *Self-joins*
- *Cross joins*
- *Natural joins*
- *Full (or two-sided) outer joins*