

# **DATABASE SYSTEMS LAB**

<b>Course Code</b>	<b>: 30102422</b>
--------------------	-------------------

<b>Credit Hours</b>	<b>: 1</b>
---------------------	------------

<b>Prerequisite</b>	<b>: 30102421</b>
---------------------	-------------------



## Instructor Information

Name	:Dr. Jihad Nader				
Office No.	B17 F4 R8				
Tel (Ext)	N/A				
E-mail	jihadnader@bau.edu.jo				
Office Hours	Sun, Thurs ,Tues (9-10) mon, wed (10-11)				
Class Times	Building	Day	Start Time	End Time	Room No.
	-	Monday	14	17	مختبر المعالجات
	-	Wednesday	14	17	مختبر المعالجات



### 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 10g: SQL  
Fundamentals I**

**Volume I • Student Guide**

---

D17108GC11  
Edition 1.1  
August 2004  
D39786

**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	<b>Creating and Managing Tables:</b> <ul style="list-style-type: none"><li>- Database Objects</li><li>- Naming Conventions</li><li>- The Create Table Statement</li><li>- Creating a Table by Using a Subquery</li><li>- Querying the Data Dictionary</li><li>- The Alter Table Statement</li><li>- Truncating a Table</li><li>- Adding Comments to a Table</li></ul>	
Week 2	<b>Including Constraints</b> <ul style="list-style-type: none"><li>- Defining Constraints<ul style="list-style-type: none"><li>o The Not Null Constraint</li><li>o The Unique Constraint</li><li>o The Primary Key Constraint</li><li>o The Foreign Key Constraint</li><li>o The Check Constraint</li></ul></li><li>- Adding a Constraint</li><li>- Dropping a Constraint</li><li>- Enabling and Disabling Constraints</li><li>- Viewing Constraints</li></ul>	
Week 3	<b>Manipulating Data</b> <ul style="list-style-type: none"><li>- Data Manipulating Language.</li><li>- The Insert Statement</li><li>- Copying Rows from another Table</li><li>- The Update Statement</li><li>- The Delete Statement</li><li>- Database Transactions</li><li>- Commit and Rollback Statements</li></ul>	
Week 4	<b>Writing Basic SQL Statements</b> <ul style="list-style-type: none"><li>- Selecting Specific Columns</li><li>- Arithmetic Expressions</li><li>- Concatenation Operator</li><li>- Using Column Aliases</li><li>- Eliminating Duplicate Rows</li></ul>	

## COURSE SYLLABUS

Week	Course Topic	Notes
Week 5	Restricting and Sorting Data <ul style="list-style-type: none"><li>- Where Clause</li><li>- Comparison Operators</li><li>- Special Operators</li><li>- Logical Operator (And, Or, Not)</li><li>- Order By Clause</li></ul>	
Week 6	Displaying Data from Multiple Tables <ul style="list-style-type: none"><li>- Cartesian Product.</li><li>- Types of Joins</li><li>- Table Aliases.</li></ul>	
Week 7	Single-Row Functions <ul style="list-style-type: none"><li>- Character Functions.</li><li>- Number Functions</li><li>- Date Functions</li></ul>	
Week 8	Midterm Exam	Midterm Exam
Week 9	Project Proposal	
Week 10	Single-Row Functions <ul style="list-style-type: none"><li>- Conversion Functions</li><li>- General Functions</li></ul>	

# COURSE SYLLABUS

Week	Course Topic	Notes
Week 11	Aggregating Data using Group Functions <ul style="list-style-type: none"><li>- Types of Group Functions (AVG, SUM, MAX, MIN, COUNT).</li><li>- Creating Groups of data: Group By Clause.</li><li>- Excluding Group Results: Having Clause.</li><li>- Nested Group Functions</li></ul>	
Week 12	Subqueries <ul style="list-style-type: none"><li>- Types of Subqueries<ul style="list-style-type: none"><li>▪ Single-Row Subqueries</li><li>▪ Multiple-Row Subqueries</li></ul></li></ul>	
Week 13	Multiple-Column Subqueries <ul style="list-style-type: none"><li>- Column Comparisons</li><li>- Null Values in a subquery</li><li>- Using a subquery in the From Clause</li></ul>	
Week 14	Using the Set Operators <ul style="list-style-type: none"><li>- Union / Union All</li><li>- Intersect</li><li>- Minus</li></ul>	
Week 15	Project Discussion	
Week 16	Final Exam	Final Exam



Week 3



# Chapter 2:

# Data Manipulating Language

# Objectives

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

- *Describe each data manipulation language (DML) statement*
- *Insert rows into a table*
- *Update rows in a table*
- *Delete rows from a table*
- *Control transactions*

# Data Manipulation Language

- *A DML statement is executed when you:*
  - Add new rows to a table
  - Modify existing rows in a table
  - Remove existing rows from a table
- *A transaction consists of a collection of DML statements that form a logical unit of work.*

# Adding a New Row to a Table

**DEPARTMENTS**

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

70	Public Relations	100	1700
----	------------------	-----	------

**New  
row**

**Insert new row  
into the  
DEPARTMENTS table**

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700
70	Public Relations	100	1700

# INSERT Statement Syntax

- Add new rows to a table by using the *INSERT* statement:

```
INSERT INTO  table [(column [, column...])]  
VALUES      (value [, value...]);
```

- With this syntax, only one row is inserted at a time.

# Inserting New Rows

- *Insert a new row containing values for each column.*
- *List values in the default order of the columns in the table.*
- *Optionally, list the columns in the `INSERT` clause.*

```
INSERT INTO departments(department_id,  
                        department_name, manager_id, location_id)  
VALUES (70, 'Public Relations', 100, 1700);  
1 row created.
```

- *Enclose character and date values in single quotation marks.*

# Inserting Rows with Null Values

- *Implicit method: Omit the column from the column list.*

```
INSERT INTO departments (department_id,  
                           department_name )  
VALUES      (30, 'Purchasing');  
1 row created.
```

- **Explicit method: Specify the NULL keyword in the VALUES clause.**

```
INSERT INTO departments  
VALUES      (100, 'Finance', NULL, NULL);  
1 row created.
```



# Inserting Special Values

The `SYSDATE` function records the current date and time.

```
INSERT INTO employees (employee_id,  
                        first_name, last_name,  
                        email, phone_number,  
                        hire_date, job_id, salary,  
                        commission_pct, manager_id,  
                        department_id)  
VALUES                (113,  
                        'Louis', 'Popp',  
                        'LPOPP', '515.124.4567',  
                        SYSDATE, 'AC_ACCOUNT', 6900,  
                        NULL, 205, 100);
```

1 row created.

# Inserting Specific Date Values

- Add a new employee.

```
INSERT INTO employees
VALUES      (114,
            'Den', 'Raphealy',
            'DRAPHEAL', '515.127.4561',
            TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
            'AC_ACCOUNT', 11000, NULL, 100, 30);
```

1 row created.

- Verify your addition.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_P
114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-99	AC_ACCOUNT	11000	

- The *DD-MON-YY* format is usually used to insert a date value. With this format, recall that the century defaults to the current century. Because the date also contains time information, the default time is midnight (00:00:00).
- If a date must be entered in a format other than the default format (for example, with another century or a specific time), you must use the
- The example in the slide records information for employee *EMPLOYEES* table. It sets the *HIRE DATE* column to be *F* use the following statement instead of the one shown in the slide. The hire date is interpreted as 2099.

```
INSERT INTO employees
VALUES      (114,
            'Den', 'Raphealy',
            'DRAPHEAL', '515.127.4561',
            '03-FEB-99',
            'AC_ACCOUNT', 11000, NULL, 100, 30);
```

# Creating a Script

- Use `:` substitution in a SQL statement to prompt for values.
- `:` is a placeholder for the variable value.

```
INSERT INTO departments
      (department_id, department_name, location_id)
VALUES (:department_id, :department_name, :location);
```

:DEPARTMENT\_ID   
DEPARTMENT\_NAME   
:LOCATION

Submit

1 row created.

# Copying Rows from Another Table

- Write your *INSERT* statement with a subquery:

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM   employees
WHERE  job_id LIKE '%REP%';
```

4 rows created.

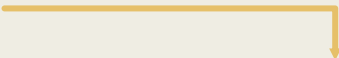
- Do not use the *VALUES* clause.
- Match the number of columns in the *INSERT* clause to those in the subquery.

# Changing Data in a Table

## EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSION_F
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	60	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	60	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	60	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

Update rows in the **EMPLOYEES** table:



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSIO
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	30	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	30	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	30	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

# UPDATE Statement Syntax

- Modify existing rows with the *UPDATE* statement:

```
UPDATE    table  
SET       column = value [, column = value, ...]  
[WHERE    condition];
```

- Update more than one row at a time (if required).

# Updating Rows in a Table

- Specific row or rows are modified if you specify the *WHERE* clause:

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
1 row updated.
```

- All rows in the table are modified if you omit the *WHERE* clause:

```
UPDATE    copy_emp
SET       department_id = 110;
22 rows updated.
```

# Updating Two Columns with a Subquery

Update employee 114's job and salary to match that of employee 205.

```
UPDATE    employees
SET       job_id  = (SELECT  job_id
                     FROM    employees
                     WHERE    employee_id = 205),
          salary  = (SELECT  salary
                     FROM    employees
                     WHERE    employee_id = 205)
WHERE     employee_id = 114;
1 row updated.
```



# Updating Rows Based on Another Table

Use subqueries in `UPDATE` statements to update rows in a table based on values from another table:

```
UPDATE copy_emp
SET    department_id = (SELECT department_id
                        FROM employees
                        WHERE employee_id = 100)
WHERE  job_id         = (SELECT job_id
                        FROM employees
                        WHERE employee_id = 200);

1 row updated.
```

# Removing a Row from a Table

## DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
100	Finance		
50	Shipping	124	1500
60	IT	103	1400

**Delete a row from the DEPARTMENTS table:**

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400

# DELETE Statement

You can remove existing rows from a table by using the DELETE statement:

```
DELETE [FROM]    table  
[WHERE          condition] ;
```

# Deleting Rows from a Table

- *Specific rows are deleted if you specify the `WHERE` clause:*

```
DELETE FROM departments
WHERE department_name = 'Finance';
1 row deleted.
```

- *All rows in the table are deleted if you omit the `WHERE` clause:*

```
DELETE FROM copy_emp;
22 rows deleted.
```

# Deleting Rows Based on Another Table

Use subqueries in `DELETE` statements to remove rows from a table based on values from another table:

```
DELETE FROM employees
WHERE department_id =
    (SELECT department_id
     FROM departments
     WHERE department_name
           LIKE '%Public%');

1 row deleted.
```

# Database Transactions

A database transaction consists of one of the following:

*DML statements that constitute one consistent change to the data* –

*One DDL statement* –

*One data control language (DCL) statement* –

# Database Transactions

*Begin when the first DML SQL statement is executed* –

*End with one of the following events:* –

A COMMIT or ROLLBACK statement is issued. ■

A DDL or DCL statement executes (automatic commit). ■

The user exits *SQL\*Plus*. ■

The system crashes. ■

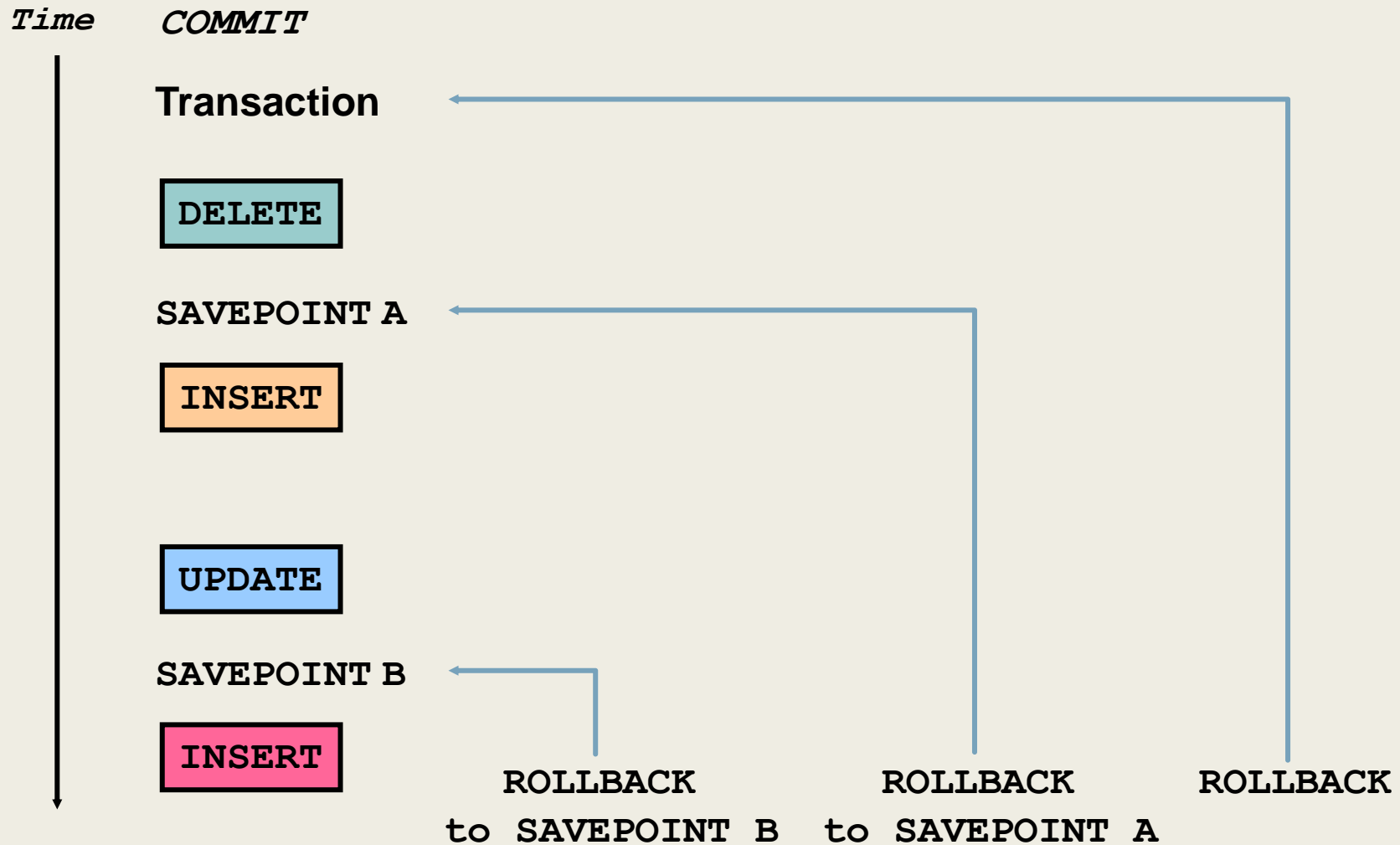
# Advantages of COMMIT and ROLLBACK Statements

With COMMIT and ROLLBACK statements, you can:

- Ensure data consistency* –
- Preview data changes before making changes permanent* –
- Group logically related operations* –



# Controlling Transactions



# Rolling Back Changes to a Marker

Create a marker in a current transaction by using the *SAVEPOINT* statement. –

Roll back to that marker by using the *ROLLBACK TO SAVEPOINT* statement. –

```
UPDATE...  
SAVEPOINT update_done;  
Savepoint created.  
INSERT...  
ROLLBACK TO update_done;  
Rollback complete.
```

# Implicit Transaction Processing

*An automatic commit occurs under the following circumstances:* –

- DDL statement is issued ■
- DCL statement is issued ■
- Normal exit from iSQL\*Plus, without explicitly issuing COMMIT or ROLLBACK statements ■

*An automatic rollback occurs under an abnormal termination of iSQL\*Plus or a system failure.* –