B P Poddar Institute of Management and Technology

Department of Computer Science and Engineering

MASTER LAB MANUAL

Subject: DATABASE MANAGEMENT SYSTEM LAB

Subject Code: CS691

Degree: B.Tech

Branch: Computer Science and Engineering

Academic Year: 2017-2018

Semester: VI

Prepared by: Suvadeep Bhattacharjee

Departmental Mission, Vision, PEO, POs and PSOs

Vision

Developing competent professionals in Computer Science and Engineering, who can adapt to constantly evolving technologies through continuous learning.

Mission

1. Enrich students with sound knowledge in fundamentals and cutting edge technologies of Computer Science and Engineering to excel globally in challenging roles in industries and academics.
2. Emphasize quality teaching, learning and research to encourage creative thoughts through application of professional knowledge and skill.
3. Inspire leadership and entrepreneurship skills in evolving areas of Computer Science and Engineering with social and environmental awareness.
4. Instil moral and ethical values to attain the highest level of accomplishment and personal growth.

Program Educational Objective (PEO)

1. Graduates of Computer Science and Engineering program will have good knowledge in the core concepts of systems, software and tools for analysing problems and designing solutions addressing the dynamic requirements of the industry and society, while employed in Industries or work as entrepreneurs.
2. Graduates of Computer Science and Engineering program will opt for higher education and research in emerging fields of Computer Science & Engineering towards building a sustainable world.
3. Graduates of Computer Science and Engineering will have leadership skills, Communication Skills, ethical and moral values, team spirit and professionalism.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSO)

1. Students will have proficiency in fundamental engineering and computing techniques and knowledge in contemporary topics like Artificial Intelligence, Data science and Distributed computing towards development of optimized algorithmic solutions.

2. Students will have capabilities to participate in the development of software and embedded systems through synergized teams to cater to the dynamic needs of the industry and society at large

COURSE OUTCOMES OF DATA BASE MANAGEMENT SYSTEM LAB (CS 691)

Course Outcome	Description	Cognitive Level	PO Mapping	PSO Mapping
CS691.1	Design Tables and Views with Constraints	Create	PO1, PO2,PO3, PO4, PO5, PO8, PO9, PO10	PSO1, PSO2
CS691.2	Write Select and Project statements	Create	PO1, PO2,PO3, PO4, PO5, PO8, PO9, PO10	PSO1, PSO2
CS691.3	Design Nested Queries across Multiple Tables and use DDL DCL TCL Commands	Create	PO1, PO2,PO3, PO4, PO5, PO8, PO9, PO10	PSO1, PSO2
CS691.4	Write PL SQL procedures	Create	PO1, PO2,PO3, PO4, PO5, PO8, PO9, PO10	PSO1, PSO2
CS691.5	Create Cursors and Triggers	Create	PO1, PO2,PO3, PO4, PO5, PO8, PO9, PO10	PSO1, PSO2

General Guidelines and Rules

Students must read the lab module and make necessary preparation for each session PRIOR TO coming to the lab.

Do's and Don'ts

Do	1. Always shut down your computer properly before leaving the laboratory
	2. Always save your assignments/programs, etc. in the specified location/drive.
	3. Login to the computers using "Student" as login name
	4. In case of any technical problem, contact technical staff immediately
Do not	1. Do not use mobile phones in the laboratory
	2. Do not use pen drives/external hard disks in the computers of the laboratory without permission
	3. Do not touch the wires connected to the computers
	4. Do not save your files in the Desktop
	5. Do not switch off without performing proper "shutdown" of computer.
	6. Do not eat while inside the laboratory.
	7. Do not carry bags/personal belongings at your computer table during lab classes
	8. Do not perform any unauthorized experiments/ try to access unauthorized network locations.

MAKAUT WB

Utech

Maulana Abul Kalam Azad University of

formerly known as West Bengal Uni

Database Management System Lab
Code: CS691
Contact: 3P
Credits: 2

Structured Query Language

1. **Creating Database**
   - Creating a Database
   - Creating a Table
   - Specifying Relational Data Types
   - Specifying Constraints
   - Creating Indexes
2. **Table and Record Handling**
   1. INSERT statement
   2. Using SELECT and INSERT together
   3. DELETE, UPDATE, TRUNCATE statements
   4. DROP, ALTER statements
3. **Retrieving Data from a Database**
   - The SELECT statement
   - Using the WHERE clause
   - Using Logical Operators in the WHERE clause
   - Using IN, BETWEEN, LIKE , ORDER BY, GROUP BY and HAVING Clause
   - Using Aggregate Functions
   - Combining Tables Using JOINS
   - Subqueries
4. **Database Management**
   - Creating Views
   - Creating Column Aliases
   - Creating Database Users
   - Using GRANT and REVOKE

Cursors in Oracle PL / SQL
Writing Oracle PL / SQL Stored Procedures

<table>
<tr><th>TOPIC</th><th>LIST OF EXPERIMENTS</th><th>CO</th><th>PO/ PSO</th></tr>
<tr>
<td>Using SQL Create table, Insert values and Use predicates with select and project</td>
<td>
1

a) Create the following table : <b>STUDENT</b>

<table>
<tr><th>Column Name</th><th>Data Type</th><th>Size</th><th>Constraints</th></tr>
<tr><td>RegNo</td><td>Varchar2</td><td>6</td><td>Not null</td></tr>
<tr><td>RollNo</td><td>Number</td><td>6</td><td>Not null</td></tr>
<tr><td>Name</td><td>Varchar2</td><td>10</td><td>Not null</td></tr>
<tr><td>Address</td><td>Varchar2</td><td>15</td><td>Not null</td></tr>
<tr><td>PhoneNo</td><td>Number</td><td>10</td><td></td></tr>
<tr><td>YearOfAdm</td><td>Number</td><td>4</td><td>Not null</td></tr>
<tr><td>DeptCode</td><td>Varchar2</td><td>4</td><td>Not null</td></tr>
<tr><td>Year</td><td>Number</td><td>1</td><td>Not null</td></tr>
<tr><td>BirthDate</td><td>Date</td><td></td><td>Not null</td></tr>
</table>

b) Insert the following data in the student table.

<table>
<tr><th>Reg No</th><th>Roll No</th><th>Name</th><th>Address</th><th>PhoneNo</th><th>YearOfAdm</th><th>Dept Code</th><th>Year</th><th>Birth Date</th></tr>
<tr><td>012301</td><td>123001</td><td>Ashish</td><td>Jadavpur</td><td>24761892</td><td>2003</td><td>CSE</td><td>3</td><td>01-Jun-81</td></tr>
<tr><td>012315</td><td>123015</td><td>Kamal</td><td>Kasba</td><td>24424987</td><td>2003</td><td>CSE</td><td>3</td><td>19-Sep-81</td></tr>
<tr><td>012424</td><td>124024</td><td>Ipsita</td><td>Kaikhali</td><td>25739608</td><td>2004</td><td>CSE</td><td>2</td><td>15-Aug-82</td></tr>
<tr><td>012250</td><td>122050</td><td>Anita</td><td>Hooghly</td><td>36719695</td><td>2002</td><td>IT</td><td>4</td><td>22-Dec-80</td></tr>
<tr><td>012344</td><td>123044</td><td>Biplab</td><td>Howrah</td><td></td><td>2003</td><td>IT</td><td>3</td><td>03-Jan-82</td></tr>
</table>
</td>
<td>CO1
CO2</td>
<td>PO1
PO2
PO3
PO4
PO5
PO8
PO9
PO10
PSO1</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>
<table>
<tr><td>012357</td><td>123057</td><td>Samik</td><td>Barasat</td><td>25426742</td><td>2003</td><td>IT</td><td>3</td><td>15-Jul-81</td></tr>
<tr><td>012419</td><td>124019</td><td>Srija</td><td>Garia</td><td>24755655</td><td>2004</td><td>EE</td><td>2</td><td>25-Oct-82</td></tr>
<tr><td>012427</td><td>124027</td><td>Saibal</td><td>Garia</td><td>24753306</td><td>2004</td><td>ECE</td><td>2</td><td>22-Mar-83</td></tr>
<tr><td>012236</td><td>122036</td><td>Santanu</td><td>Dum Dum</td><td></td><td>2002</td><td>ECE</td><td>4</td><td>11-Dec-80</td></tr>
<tr><td>012349</td><td>123049</td><td>Gita</td><td>Kasba</td><td>24428682</td><td>2003</td><td>MCA</td><td>3</td><td>14-Apr-81</td></tr>
</table>
c) Display all records
d) Display name, address and year of admission of each student
e) List the name and year of students who are in Computer Science.
f) List the names and departments of students belonging to 3rd year.
g) Display names of students with ‘a’ as the second letter in their names.
h) Display names of students in alphabetical order.
i) Display names and addresses of students who took admission in the year 2004.
j) List the names of students who do not have a phone number.
</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use of DML - select rows, delete rows and update table operations</td>
<td>Note : Tables created previously in lab exercises may be used if required
2.
a) Delete the name of a student whose roll no, year and department code is given.
b) Display the number of students in each department.
c) Change the address of a student whose roll no and name is given.
d) Add the college phone number (25739607) to each of these students.
e) Change the size of column Name to 15 characters.
f) Add a column MarksObtained (number) to the student table.
g) Insert values against marks column.
h) Drop column MarksObtained from table student.
i) Add constraint primary key to the column RegNo of table student.
j) Add check constraints to the column year of student table. (year should be entered within 1,2,3,4).</td>
<td>CO1
CO2</td>
<td>PO1
PO2
PO3
PO4
PO5
PO8
PO9
PO10
PSO1</td>
</tr>
<tr>
<td>Use of DDL - Alter Table Statement,</td>
<td>Note : Tables created previously in lab exercises may be used if required
3.
a. Create table DEPARTMENT
<table>
<tr><th>Column Name</th><th>Data Type</th><th>Size</th><th>Constraints</th></tr>
<tr><td>DeptCode</td><td>Varchar2</td><td>4</td><td>Not null, Primary key</td></tr>
</table>
</td>
<td>CO1
CO2</td>
<td>PO1
PO2
PO3
PO4</td>
</tr>
</table>

<table>
<tr>
<td>Check Constraints, Foreign Key constraints in SQL</td>
<td>
<table>
<tr><td>DeptName</td><td>Varchar2</td><td>15</td><td>Not null</td></tr>
<tr><td>HOD</td><td>Varchar2</td><td>4</td><td>Not null</td></tr>
</table>
<b>FACULTY</b>
<table>
<tr><th>Column Name</th><th>Data Type</th><th>Size</th><th>Constraints</th></tr>
<tr><td>FacultyCode</td><td>Varchar2</td><td>4</td><td>Not null, Primary key, Starts with ‘F’</td></tr>
<tr><td>FacultyName</td><td>Varchar2</td><td>15</td><td>Not null</td></tr>
<tr><td>DateOfJoin</td><td>Date</td><td></td><td>Not null</td></tr>
<tr><td>DeptCode</td><td>Varchar2</td><td>4</td><td>Must be either CSE,IT, CA, CHEM, MTHS, PHYS, HUM, BBA</td></tr>
</table>
b. Insert appropriate values in the above table.
c. Add constraint : DeptCode of Faculty is foreign key and references DeptCode in Department
d. Find the names of faculties of CSE Department.
e. Find the number of faculties in the Computer application department
f. Show the names of the heads of departments with department name.
g. Find the number of faculties who joined in August.
h. Add an extra attribute to the faculty table - Salary Number(8,2)
i. Insert values into the corresponding field Salary Number(8,2).
j. Find the name and salary of the faculty who earn more than 8000.
k. Find the name, department of the faculties who earn between 8000 and 12000.
</td>
<td></td>
<td>PO5
PO8
PO9
PO10
PSO1
PSO2</td>
</tr>
<tr>
<td>Join Operations Cartesian Product, Natural Join, Outer Join</td>
<td>
Note : Tables created previously in lab exercises may be used if required
4.
a. Create table SUBJECT and insert appropriate values.
<table>
<tr><th>Column Name</th><th>Data Type</th><th>Size</th><th>Constraints</th></tr>
<tr><td>SubjectCode</td><td>Varchar2</td><td>4</td><td>Not null, Primary key</td></tr>
<tr><td>SubjectName</td><td>Varchar2</td><td>15</td><td>Not null</td></tr>
<tr><td>Faculty</td><td>Varchar2</td><td>4</td><td>Foreign key references FacultyCode of table FACULTY</td></tr>
</table>
b. Find the number of faculties in each department with their department
</td>
<td>CO1
CO2</td>
<td>PO1
PO2
PO3
PO4
PO5
PO8
PO9
PO10
PSO1
PSO2</td>
</tr>
</table>

	name. c. Increment the salary of each faculty by Rs 500. d. Find the names of students and faculties whose name start with ‘S’. e. Find the students who stay in Kaikhali f. Find the names of faculties who take classes in the IT department. g. Find the names of all faculties whose HOD is given.		
Queries using aggregate functions (count,sum, avg,max,min) and group by, having	Note : Tables created previously in lab exercises may be used if required 5. a. Add extra attribute to the Subject table - department varchar2 (4), year varchar2 (1) b. Insert values into the fields - department, year. c. Find the maximum salary among the faculties. d. Find the names of faculties who earn more than the average of all faculties. e. List the names of faculties of CSE department who earn more than the average salary of the department. f. Find the maximum and minimum salaries among faculties. g. Find the second maximum salary among all faculties. h. Find the names of faculties who are not the HOD’s of any department. i. Find the names of subjects for students of CSE 3rd year.	CO1 CO2	PO1 PO2 PO3 PO4 PO5 PO8 PO9 PO10 PSO1 PSO2
Creation and Dropping of Views	Note : Tables created previously in lab exercises may be used if required 6. a. Name the departments having highest number of faculties and display the names of faculties b. Create a view on the STUDENT table named V_STD selecting all the columns. Run the following queries on the view. i. Display all data from the view. ii. Insert a new row into the view with the following data – 012363 123011 Bishakh Salt Lake 23371987 2005 IT iii. Display data from student table to verify that the row has been inserted into the Table. iv. Update the address of Bishakh to “SectorV” & verify the change in the table. c. Create a view on student table snamed V_STD_2 selecting the columns – RegNo, Name, Year, Deptcode. i. Display data from the view. ii. Try to insert data into table through view. iii. Update the Deptcode of ‘Kamal’ to ‘IT’ through view. iv. Delete records of students of 4th year through view. d. Create a view named V_FACULTY consisting of columns	CO1 CO2	PO1 PO2 PO3 PO4 PO5 PO8 PO9 PO10 PSO1 PSO2

<table>
<tr>
<td></td>
<td>FacultyName, DeptCode from FACULTY table and HOD from Department table.

i. Display data from V_FACULTY
ii. Try to insert a new row into this view V_FACULTY.
iii. Try to update the DeptCode of a CSE faculty to IT.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nested Queries using any, all in, exist, not exists, unique, intersect constraints</td>
<td>Note : Tables created previously in lab exercises may be used if required
7.
Considering -
Branch Schema <branch-name, branch-city, assets>
Customer Schema <customer-name, customer-street, customer-city>
Loan Schema <loan-number, branch-name, amount>
Borrower Schema <customer-name, loan-number>
Account Scheme <account-number, branch-name, balance>
Depositor Scheme <customer-name, account-number>

BRANCH TABLE
<table>
<tr><td>Branch Name</td><td>Branch City</td><td>Assets</td></tr>
<tr><td>Brighton</td><td>Brooklyn</td><td>7100000</td></tr>
<tr><td>Downtown</td><td>Brooklyn</td><td>9000000</td></tr>
<tr><td>Mianus</td><td>Horseneck</td><td>400000</td></tr>
<tr><td>North Town</td><td>Rye</td><td>3700000</td></tr>
<tr><td>Perryridge</td><td>Horseneck</td><td>1700000</td></tr>
<tr><td>Pownal</td><td>Bennington</td><td>300000</td></tr>
<tr><td>Redwood</td><td>Palo Alto</td><td>2100000</td></tr>
<tr><td>Round Hill</td><td>Horseneck</td><td>800000</td></tr>
</table>

CUSTOMER TABLE
<table>
<tr><td>Customer Name</td><td>Customer Street</td><td>Customer City</td></tr>
<tr><td>Adams</td><td>Spring</td><td>Pittsfield</td></tr>
<tr><td>Brooks</td><td>Senator</td><td>Brooklyn</td></tr>
<tr><td>Curry</td><td>North</td><td>Rye</td></tr>
<tr><td>Glenn</td><td>Sand Hill</td><td>Woodside</td></tr>
<tr><td>Green</td><td>Walnut</td><td>Stamford</td></tr>
<tr><td>Hayes</td><td>Main</td><td>Harrison</td></tr>
<tr><td>Johnson</td><td>Alma</td><td>Palo Alto</td></tr>
<tr><td>Jones</td><td>Main</td><td>Harrison</td></tr>
<tr><td>Lindsay</td><td>Park</td><td>Pittsfield</td></tr>
<tr><td>Smith</td><td>North</td><td>Rye</td></tr>
<tr><td>Turner</td><td>Putnam</td><td>Stamford</td></tr>
<tr><td>Williams</td><td>Nassau</td><td>Princeton</td></tr>
</table>

BORROWER TABLE</td>
<td>CO3</td>
<td>PO1
PO2
PO3
PO4
PO5
PO8
PO9
PO10
PSO1
PSO2</td>
</tr>
</table>

Customer Name	Loan Number
Adams	l-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

ACCOUNT TABLE

Account Number	Branch Name	Balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

a. To find all customers having a loan, an account or both at the bank, without duplicates.
b. To find all customers having a loan, an account or both at the bank, with duplicates.
c. To find all customers having both a loan and an account at the bank, without duplicates.
d. To find all customers having a loan, an account or both at the bank, with duplicates.
e. To find all customers who have an account but no loan at the bank, without duplicates.
f. To find all customers who have an account but no loan at the bank, with duplicates.
g. Find the number of depositors for each branch where average account balance is more than Rs 1200.
h. Find all customers who have both an account and a loan at the Perryridge branch.
i. Find the names of all branches that have assets greater than that of each branch located in Brooklyn.
j. Find all customers who have an account at all the branches located in Brooklyn.
k. Find all customers who have at most one account at the Perryridge branch.
l. Find all customers who have at least two accounts at the Perryridge

	branch. m. Find the all customers who have an account but no loan at the bank. n. Find the all customers who have either an account or a loan (but not both) at the bank.		
DDL DCL TCL Commands	Note : Tables created previously in lab exercises may be used if required 8. Consider the following tables namely "DEPARTMENTS" & "EMPLOYEES" Their schemas are as follows - Departments (dept _no , dept_ name , dept_location); Employees (emp_id , emp_name , emp_salary); a. Develop a query to grant all privileges of employees table into departments table b. Develop a query to grant some privileges of employees table into departments table c. Develop a query to revoke all privileges of employees table from departments table d. Develop a query to revoke some privileges of employees table from departments table e. Write a query to implement the save point f. Write a query to implement the rollback g. Write a query to implement the commit	CO3	PO1 PO2 PO3 PO4 PO5 PO8 PO9 PO10 PSO1 PSO2
PL/Sql Basic	9. a. Write a PL/SQL code, EX_INVNO.SQL, block for inverting a number using all forms of loops. b. Write a PL/SQL code, EX_SUMNO.SQL that prints the sum of 'n' natural numbers. c. Write a PL/SQL program to print all the prime numbers between 100 and 400 d. Write a PL/SQL program to print n terms of fibonacci series. e. Write a PL/SQL program to calculate HCF of two numbers. f. Write a PL/SQL code, EX_AREA.SQL, of block to calculate the area of the circle for the values of radius varying from 3 to 7. Store the radius and the corresponding values of calculated area in the table AREA_VALUES.	CO4	PO1 PO2 PO3 PO4 PO5 PO8 PO9 PO10 PSO1 PSO2
Procedures and cursors using PL/SQL	10. a. Create a PL/SQL program using cursors, to retrieve first tuple from the department relation. b. (use table dept(dno, dname, loc)) c. Create a PL/SQL program using cursors, to retrieve each tuple from the department relation. d. (use table dept(dno, dname, loc)) e. Create a PL/SQL program using cursors, to display the number, name, salary of the three highest paid employees. f. (use table emp(empno, ename,sal)) g. Create a PL/SQL program using cursors, to delete the employees	CO4 CO5	PO1 PO2 PO3 PO4 PO5 PO8 PO9

	whose salary is more than 3000. h. Create a PL/SQL program using cursors, to update the salary of each employee by the avg salary if their salary is less than avg salary. i. Create a PL/SQL program using cursors, to insert into a table, NEWEMP, the record of ALL MANAGERS. Also DISPLAY on the screen the NO, NAME, JOIN_DATE. Handle any user defined exceptions. j. (use table emp(emp_no, emp_name, join_date, desig))		PO10 PSO1 PSO2
Additional Experiments			
Creation and usage of trigger	Note : Tables created previously in lab exercises may be used if required 11. Considering - Empa Schema<id number, name, dname, age, income, expence, savings> Emp Schema<institute name, employee id, salary> Sal <institute name, total employee, total salary> a. For every insert or delete or update in Empa table create trigger to display the message TABLE IS INSERTED or TABLE IS DELETED or TABLE IS UPDATED b. Define trigger to force all department names to uppercase. c. Create a Trigger to check the age valid or not using message after every insert or delete or update in Trig table d. Create a Trigger to check the age valid and Raise appropriate error code and error message. e. A trigger restricting updates that allows changes to Empa records only on Mondays through Fridays, and only during the hours of 8:00am to 5:00pm. f. Create a Trigger for Emp table it will update another table Sal while inserting values.	CO5	PO1 PO2 PO3 PO4 PO5 PO8 PO9 PO10 PSO1 PSO2

B. P. Poddar Institute of Management and Technology
Department of Computer Science and Engineering
Academic Year : 2017-18 [Even Semester]

Lab Assignment 1

Topic : Using SQL Create table, Insert values and Use predicates with select and project

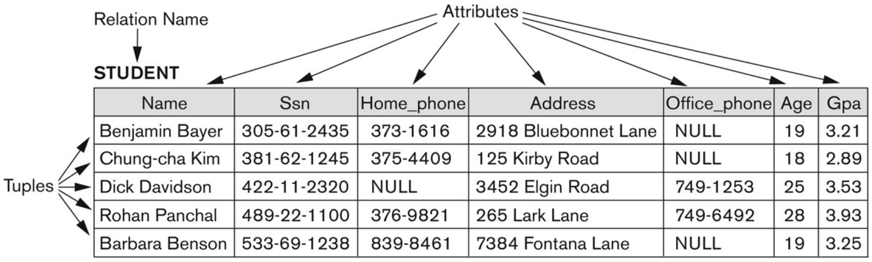
Readings:

Structured Query Language (SQL)

- SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.
- SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

Mapping of Terms in SQL to Relational Model

- Table, Row, Column Header, Column Type :: Relation, Tuple, Attribute, Domain

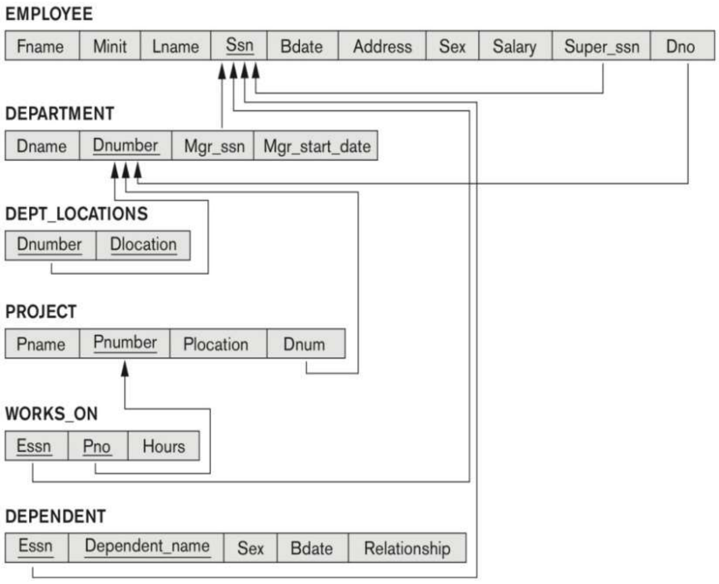


STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 5.1
The attributes and tuples of a relation STUDENT.

COMPANY relational database schema (Fig. 5.7)



Basic form of the CREATE TABLE Statement

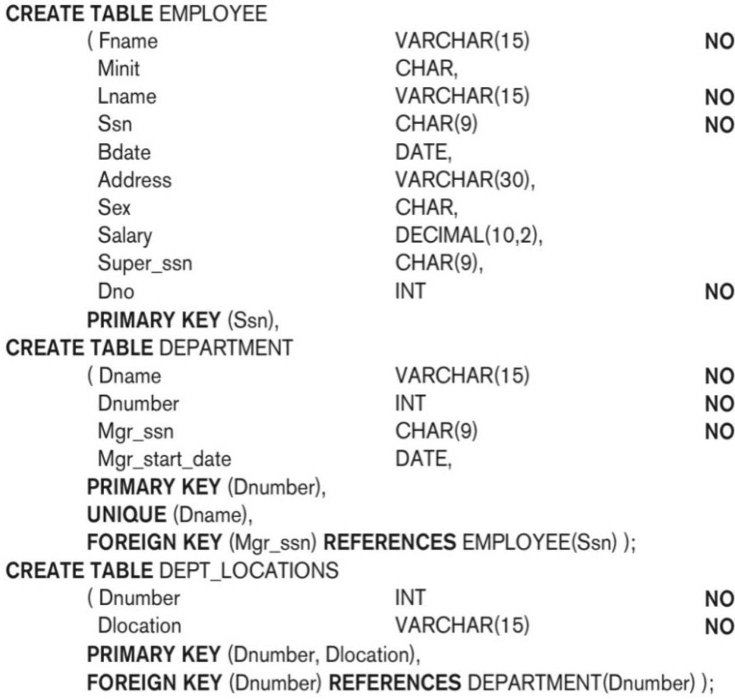
```
CREATE TABLE tablename (
  column_name  datatype  [default x]  [constraint(s)],
  column_name  datatype  [default x]  [constraint(s)],
  …
  table_constraint,
  table_constraint
  …
);
```

- Used to create Base tables (base relations)
- Relation and its tuples are actually created and stored as a file by the DBMS

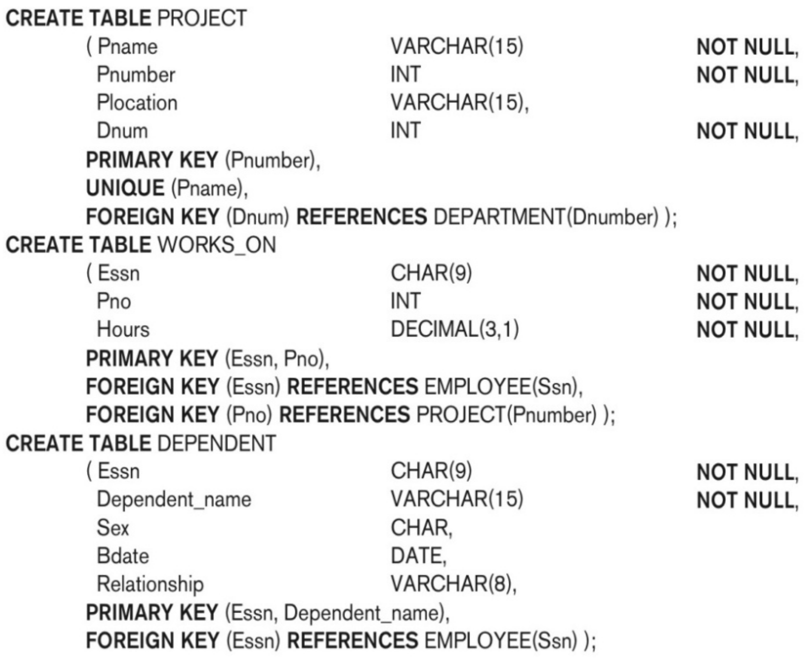
Attribute Data Types and Domains in SQL

Data Type	**Representation**
Numeric	INTEGER, INT, SMALLINT, FLOAT or REAL, and DOUBLE PRECISION
Character-string	CHAR(*n*), CHARACTER(*n*) VARCHAR(*n*), CHAR VARYING(*n*), CHARACTER VARYING(*n*)
Bit-string	BIT(*n*), BIT VARYING(*n*)
Boolean	Values of TRUE or FALSE or NULL
DATE	Components are YEAR, MONTH, and DAY in the form YYYY-MM-DD

SQL CREATE TABLE DDL statements for defining the COMPANY schema from Fig. 5.7 (Fig. 6.1)



```
CREATE TABLE EMPLOYEE
       ( Fname              VARCHAR(15)        NOT NULL,
         Minit              CHAR,
         Lname              VARCHAR(15)        NOT NULL,
         Ssn                CHAR(9)            NOT NULL,
         Bdate              DATE,
         Address            VARCHAR(30),
         Sex                CHAR,
         Salary             DECIMAL(10,2),
         Super_ssn          CHAR(9),
         Dno                INT                NOT NULL,
       PRIMARY KEY (Ssn),
CREATE TABLE DEPARTMENT
       ( Dname              VARCHAR(15)        NOT NULL,
         Dnumber            INT                NOT NULL,
         Mgr_ssn            CHAR(9)            NOT NULL,
         Mgr_start_date     DATE,
       PRIMARY KEY (Dnumber),
       UNIQUE (Dname),
       FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
CREATE TABLE DEPT_LOCATIONS
       ( Dnumber            INT                NOT NULL,
         Dlocation          VARCHAR(15)        NOT NULL,
       PRIMARY KEY (Dnumber, Dlocation),
       FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
```



```
CREATE TABLE PROJECT
       ( Pname                  VARCHAR(15)          NOT NULL,
         Pnumber                INT                  NOT NULL,
         Plocation              VARCHAR(15),
         Dnum                   INT                  NOT NULL,
       PRIMARY KEY (Pnumber),
       UNIQUE (Pname),
       FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
       ( Essn                   CHAR(9)              NOT NULL,
         Pno                    INT                  NOT NULL,
         Hours                  DECIMAL(3,1)         NOT NULL,
       PRIMARY KEY (Essn, Pno),
       FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
       FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
       ( Essn                   CHAR(9)              NOT NULL,
         Dependent_name         VARCHAR(15)          NOT NULL,
         Sex                    CHAR,
         Bdate                  DATE,
         Relationship           VARCHAR(8),
       PRIMARY KEY (Essn, Dependent_name),
       FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

NOTE: Some foreign keys may cause errors, specified either via:

- Circular references
- Or because they refer to a table that has not yet been created

Specifying Constraints in SQL

- Basic constraints:
  - Relational Model has 3 basic constraint types that are supported in SQL:
    - Key constraint: A primary key value cannot be duplicated
    - Entity Integrity Constraint: A primary key value cannot be null
    - Referential integrity constraints : The "foreign key " must have a value that is already present as a primary key, or may be null.

Specifying Attribute Constraints

- Other Restrictions on attribute domains:
  - Default value of an attribute
    - DEFAULT <value>
    - NULL is not permitted for a particular attribute (NOT NULL)
  - CHECK clause
    - Dnumber INT NOT NULL CHECK (Dnumber > 0 AND Dnumber < 21);

Specifying Key and Referential Integrity Constraints

- PRIMARY KEY clause
  - Specifies one or more attributes that make up the primary key of a relation
  - Dnumber INT PRIMARY KEY;

- UNIQUE clause
  - Specifies alternate (secondary) keys (called CANDIDATE keys in the relational model).
  - Dname VARCHAR(15) UNIQUE;

- FOREIGN KEY clause
  - Default operation: reject update on violation *
  - Attach referential triggered action clause
    - Options include SET NULL, CASCADE, and SET DEFAULT
    - Action taken by the DBMS for SET NULL or SET DEFAULT is the same for both ON DELETE and ON UPDATE
    - CASCADE option suitable for "relationship" relations

* Possible violations for each operation

- INSERT may violate any of the constraints:
  - Domain constraint:
    - if one of the attribute values provided for the new tuple is not of the specified attribute domain
  - Key constraint:
    - if the value of a key attribute in the new tuple already exists in another tuple in the relation
  - Referential integrity:
    - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation

      For example - inserting a tuple in table EMPLOYEE (department no as foreign key) that doesn't exist in table DEPARTMENT (no such department no value for department no as primary key)
  - Entity integrity:
    - if the primary key value is null in the new tuple

- DELETE may violate only referential integrity:
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database
    For example - deleting a tuple in DEPARTMENT (department no as primary key)
    that is referenced by tuples in table EMPLOYEE (with department no as foreign key)

        - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL
        - RESTRICT option: reject the deletion
        - CASCADE option: propagate the new primary key value into the foreign keys of the referencing tuples
        - SET NULL option: set the foreign keys of the referencing tuples to NULL
    - One of the above options must be specified during database design for each foreign key constraint

- UPDATE
    - may violate domain constraint and NOT NULL constraint on an attribute being modified
    - Any of the other constraints may also be violated, depending on the attribute being updated:
        - Updating the primary key (PK):
            - Similar to a DELETE followed by an INSERT
            - Need to specify similar options to DELETE
        - Updating a foreign key (FK):
            - May violate referential integrity
        - Updating an ordinary attribute (neither PK nor FK):
            - **Can only violate domain constraints**

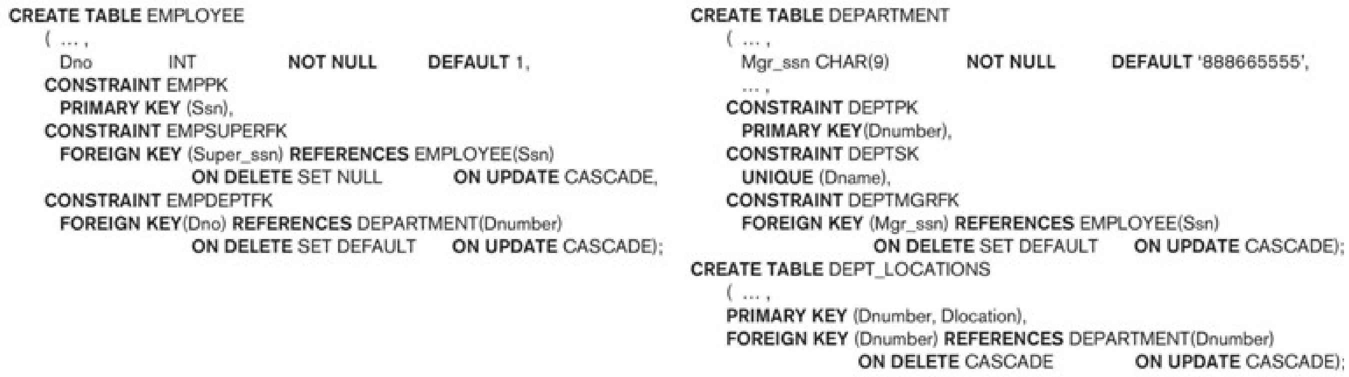
Giving Names to Constraints

- Using the Keyword CONSTRAINT
    - Name a constraint
    - Useful for later altering

Specifying Constraints on Tuples Using CHECK

- Additional Constraints on individual tuples within a relation are also possible using CHECK
- CHECK clauses at the end of a CREATE TABLE statement
    - Apply to each tuple individually
    - CHECK (Dept_create_date <= Mgr_start_date);

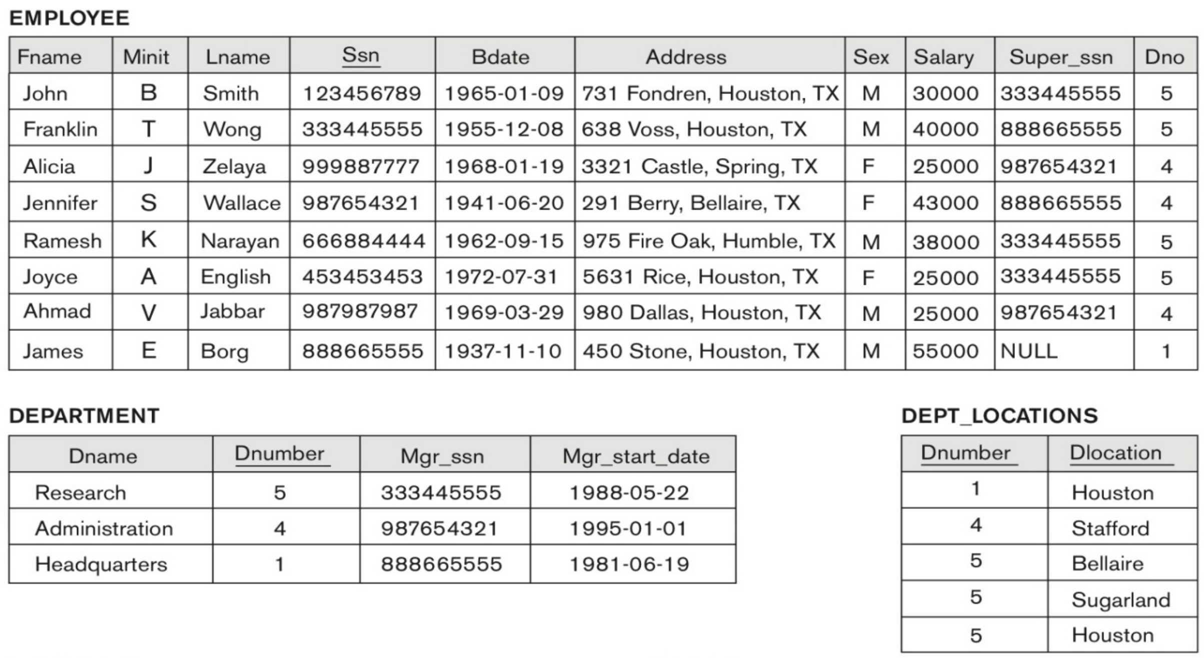
Default attribute values and referential integrity triggered action specification (Fig. 6.2)



```
CREATE TABLE EMPLOYEE
   ( ... ,
     Dno          INT          NOT NULL       DEFAULT 1,
   CONSTRAINT EMPPK
     PRIMARY KEY (Ssn),
   CONSTRAINT EMPSUPERFK
     FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
                     ON DELETE SET NULL          ON UPDATE CASCADE,
   CONSTRAINT EMPDEPTFK
     FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber)
                     ON DELETE SET DEFAULT       ON UPDATE CASCADE);
```

```
CREATE TABLE DEPARTMENT
   ( ... ,
     Mgr_ssn CHAR(9)              NOT NULL        DEFAULT '888665555',
     ... ,
   CONSTRAINT DEPTPK
     PRIMARY KEY(Dnumber),
   CONSTRAINT DEPTSK
     UNIQUE (Dname),
   CONSTRAINT DEPTMGRFK
     FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
                     ON DELETE SET DEFAULT       ON UPDATE CASCADE);
CREATE TABLE DEPT_LOCATIONS
   ( ... ,
   PRIMARY KEY (Dnumber, Dlocation),
   FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
                     ON DELETE CASCADE           ON UPDATE CASCADE);
```

One possible database state for the COMPANY relational database schema (Fig. 5.7)



EMPLOYEE

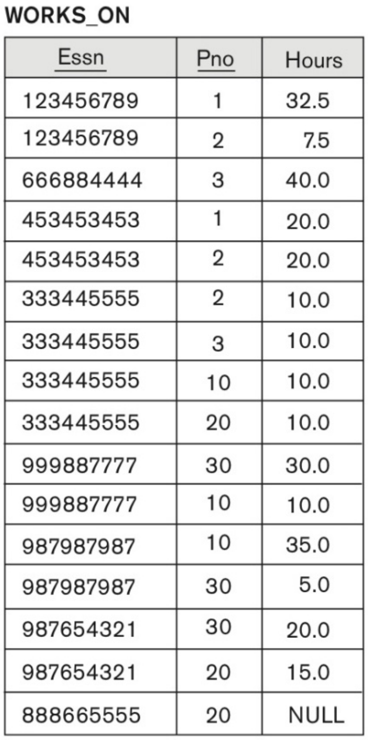
Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

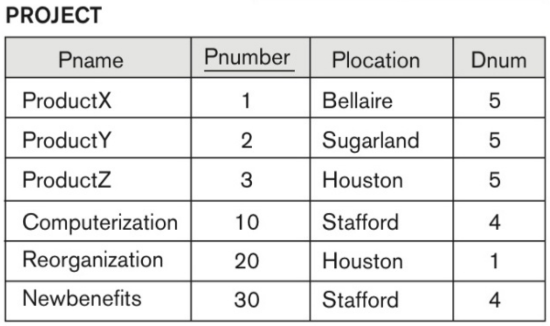
DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston



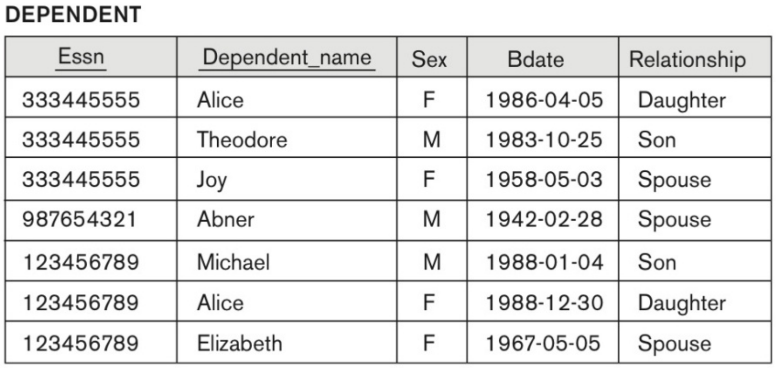
WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL



PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4



DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Basic form of the INSERT INTO Statement

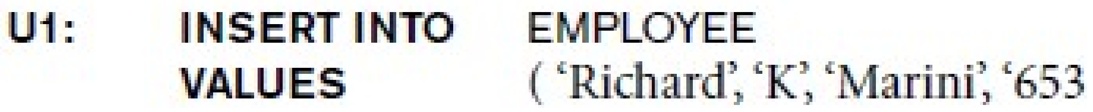
INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)
VALUES (*value1*, *value2*, *value3*, ...);

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT INTO syntax would be as follows:

INSERT INTO *table_name*
VALUES (*value1*, *value2*, *value3*, ...);

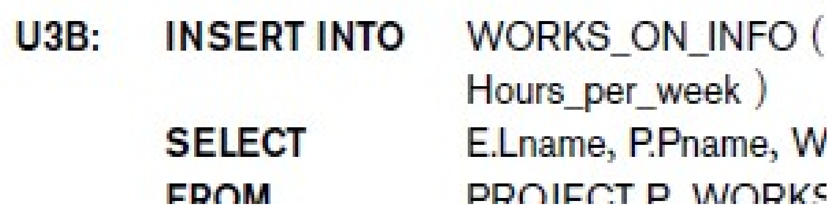
Example SQL INSERT INTO statements for populating the DEPARTMENT table of COMPANY schema from Figure 5.7 (Fig. 6.1)

- Specify the relation name and a list of values for the tuple. All values including nulls are supplied.



```
U1:     INSERT INTO     EMPLOYEE
        VALUES          ( 'Richard', 'K', 'Marini', '653
```

- The variation below inserts multiple tuples where a new table is loaded values from the result of a query.



```
U3B:    INSERT INTO    WORKS_ON_INFO (
                       Hours_per_week )
        SELECT         E.Lname, P.Pname, W
        FROM           PROJECT P, WORKS
```

BULK LOADING OF TABLES

- Another variation of INSERT is used for bulk-loading of several tuples into tables
- A new table TNEW can be created with the same attributes as T and using LIKE and DATA in the syntax, it can be loaded with entire data.
  EXAMPLE:

```
CREATE TABLE D5EMPS  LIKE  EMPLOYEE
                (SELECT   E.*
                 FROM     EMPLOYEE AS E
                 WHERE    E.Dno=5)
WITH DATA;
```

Note: Insertion operation can violate –

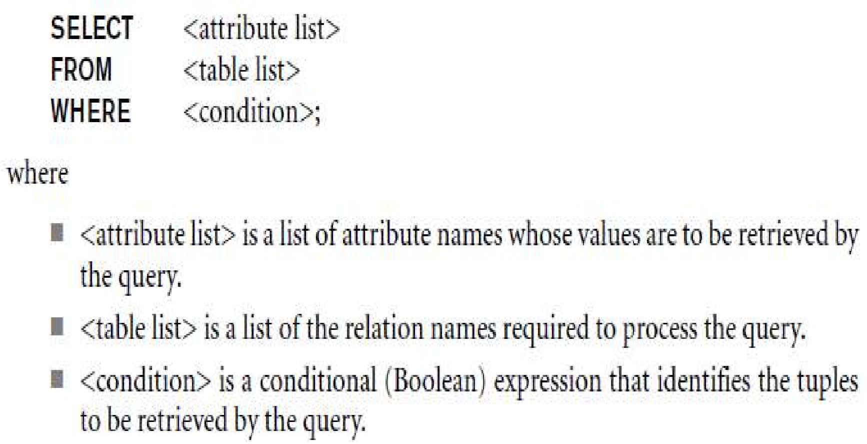
- Domain Constraint
- Key Constraint
- Integrity Constraint
- Referential Integrity Constraint

Basic Retrieval Queries in SQL

- SELECT statement
  - One basic statement for retrieving information from a database
- SQL allows a table to have two or more tuples that are identical in all their attribute values
  - Unlike relational model (relational model is strictly set-theory based)
  - Multiset or bag behavior
  - Tuple-id may be used as a key

The SELECT-FROM-WHERE Structure of Basic SQL Queries

- **Basic form of the SELECT statement:**



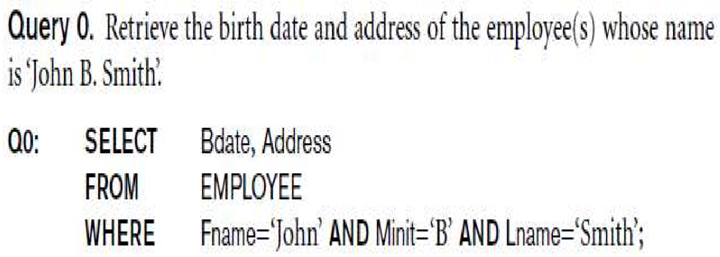
```
SELECT    <attribute list>
FROM      <table list>
WHERE     <condition>;
```

where

- <attribute list> is a list of attribute names whose values are to be retrieved by the query.
- <table list> is a list of the relation names required to process the query.
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

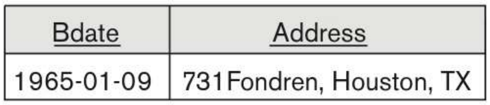
- Logical comparison operators
  - =, <, <=, >, >=, and <>
- Projection attributes
  - Attributes whose values are to be retrieved
- Selection condition
  - Boolean condition that must be true for any retrieved tuple. Selection conditions include join conditions when multiple relations are involved.

Basic Retrieval Queries

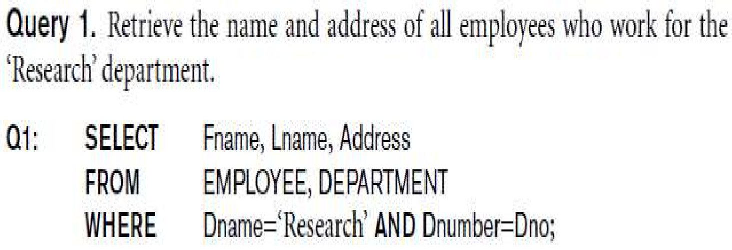


Query 0. Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'.

```
Q0:   SELECT   Bdate, Address
      FROM     EMPLOYEE
      WHERE    Fname='John' AND Minit='B' AND Lname='Smith';
```

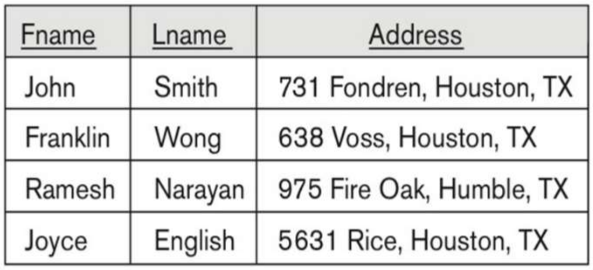


Bdate	Address
1965-01-09	731Fondren, Houston, TX

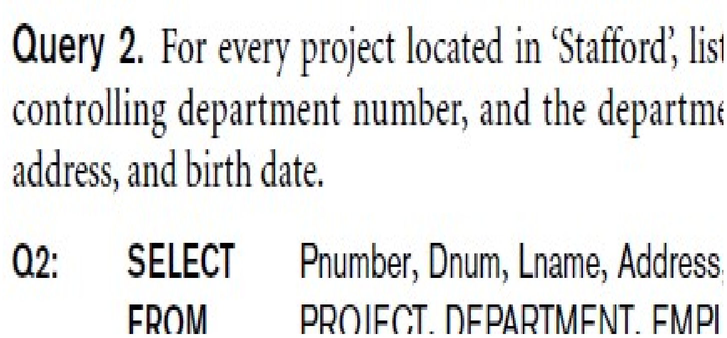


Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:   SELECT   Fname, Lname, Address
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    Dname='Research' AND Dnumber=Dno;
```

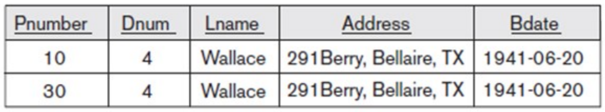


Fname	Lname	Address
John	Smith	731 Fondren, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 Fire Oak, Humble, TX
Joyce	English	5631 Rice, Houston, TX



Query 2. For every project located in 'Stafford', lis
controlling department number, and the departme
address, and birth date.

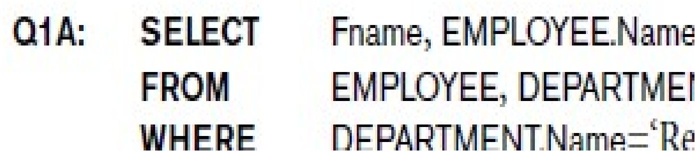
```
Q2:    SELECT    Pnumber, Dnum, Lname, Address,
       FROM      PROJECT, DEPARTMENT, EMPL
```



Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291Berry, Bellaire, TX	1941-06-20

Ambiguous Attribute Names

- Same name can be used for two (or more) attributes in different relations
  - As long as the attributes are in different relations
  - Must qualify the attribute name with the relation name to prevent ambiguity



```
Q1A:   SELECT    Fname, EMPLOYEE.Name
       FROM      EMPLOYEE, DEPARTME
       WHERE     DEPARTMENT.Name='Re
```

Aliasing, and Renaming

- Aliases or tuple variables
  - Declare alternative relation names E and S to refer to the EMPLOYEE relation twice in a query:

Query 8. For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
SELECT  E.Fname, E.Lname, S.Fname, S.Lname
FROM  EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.Super_ssn=S.Ssn;
```

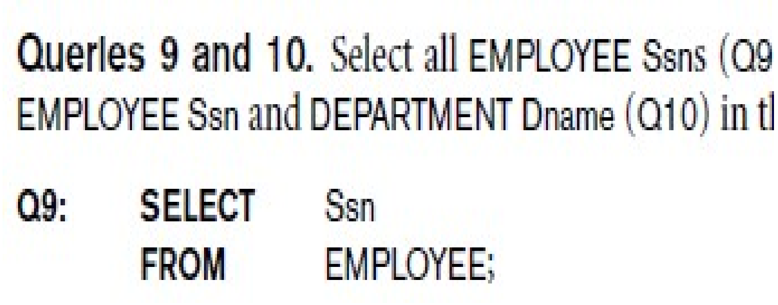
  - Recommended practice to abbreviate names and to prefix same or similar attribute from multiple tables.

- The attribute names can also be renamed
  - EMPLOYEE AS E(Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Sal, Sssn, Dno)
  - Note that the relation EMPLOYEE now has a variable name E which corresponds to a tuple variable

Note: The "AS" may be dropped in most SQL implementations

Unspecified WHERE Clause and Use of the Asterisk

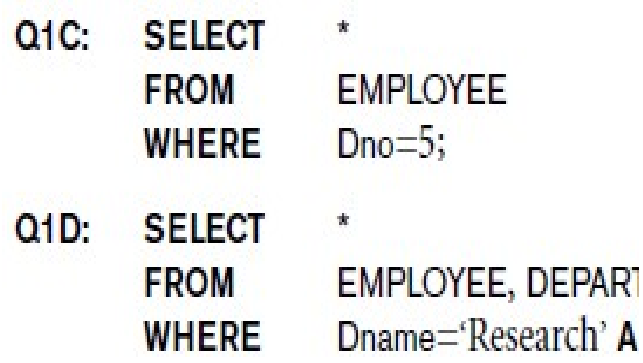
- Missing WHERE clause
  - Indicates no condition on tuple selection
- Effect is a CROSS PRODUCT
  - Result is all possible tuple combinations (or the Algebra operation of Cartesian Product result)



Queries 9 and 10. Select all EMPLOYEE Ssns (Q9
EMPLOYEE Ssn and DEPARTMENT Dname (Q10) in t

```
Q9:     SELECT     Ssn
        FROM       EMPLOYEE;
```

- Specify an asterisk (*)
  - Retrieve all the attribute values of the selected tuples
  - The * can be prefixed by the relation name; e.g., EMPLOYEE *

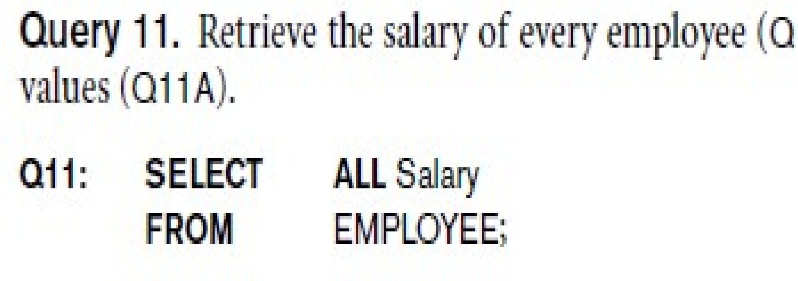


```
Q1C:    SELECT     *
        FROM       EMPLOYEE
        WHERE      Dno=5;

Q1D:    SELECT     *
        FROM       EMPLOYEE, DEPAR
        WHERE      Dname='Research' A
```

Tables as Sets in SQL

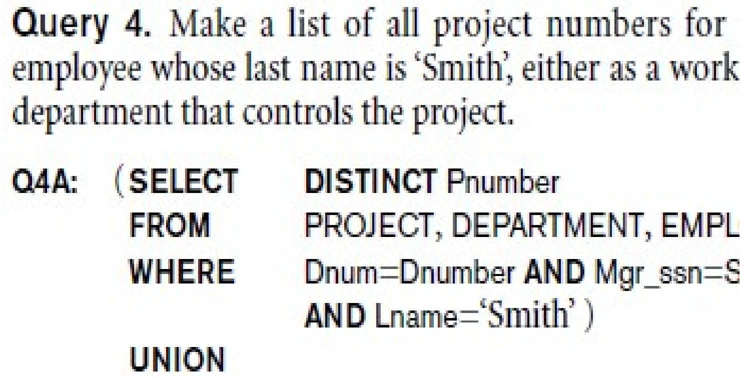
- SQL does not automatically eliminate duplicate tuples in query results
- For aggregate operations duplicates must be accounted for
- Use the keyword DISTINCT in the SELECT clause
  - Only distinct tuples should remain in the result



Query 11. Retrieve the salary of every employee (Q
values (Q11A).

```
Q11:   SELECT   ALL Salary
       FROM     EMPLOYEE;
```

- Set operations
  - UNION, EXCEPT (difference), INTERSECT
  - Corresponding multiset operations: UNION ALL, EXCEPT ALL, INTERSECT ALL)
  - Type compatibility is needed for these operations to be valid



Query 4. Make a list of all project numbers for
employee whose last name is 'Smith', either as a work
department that controls the project.

```
Q4A:  (SELECT   DISTINCT Pnumber
       FROM     PROJECT, DEPARTMENT, EMPL
       WHERE    Dnum=Dnumber AND Mgr_ssn=S
                AND Lname='Smith' )
       UNION
```

<u>Substring Pattern Matching and Arithmetic Operators</u>

- LIKE comparison operator
  - Used for string pattern matching
  - % replaces an arbitrary number of zero or more characters
  - underscore (_) replaces a single character
  - Examples: WHERE Address LIKE '%Houston,TX%';
  - WHERE Ssn LIKE '_ _ 1_ _ 8901';

- BETWEEN comparison operator

  E.g.: WHERE(Salary BETWEEN 30000 AND 40000) AND Dno = 5;

<u>Arithmetic Operations</u>

- Standard arithmetic operators:
  - Addition (+), subtraction (–), multiplication (*), and division (/) may be included as a part of SELECT

- Query 13. Show the resulting salaries if every employee working on the ‘ProductX’ project is given a 10 percent raise.

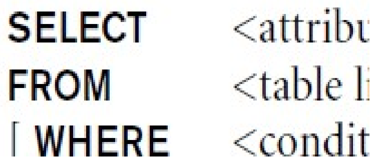
```
SELECT  E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal
FROM  EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
WHERE  E.Ssn=W.Essn AND W.Pno=P.Pnumber AND P.Pname=‘ProductX’;
```

Ordering of Query Results

- Use ORDER BY clause
  - Keyword DESC to see result in a descending order of values
  - Keyword ASC to specify ascending order explicitly
  - Typically placed at the end of the query

- ORDER BY D.Dname DESC, E.Lname ASC, E.Fname ASC

EXPANDED Block Structure of SQL Queries



```
SELECT     <attribu
FROM       <table li
[ WHERE    <condit
```

Comparisons Involving NULL and Three-Valued Logic

- **Meanings of NULL**
  - **Unknown value**
  - **Unavailable or withheld value**
  - **Not applicable attribute**
- **Each individual NULL value considered to be different from every other NULL value**
- **SQL uses a three-valued logic:**
  - **TRUE, FALSE, and UNKNOWN (like Maybe)**
- **NULL = NULL comparison is avoided**

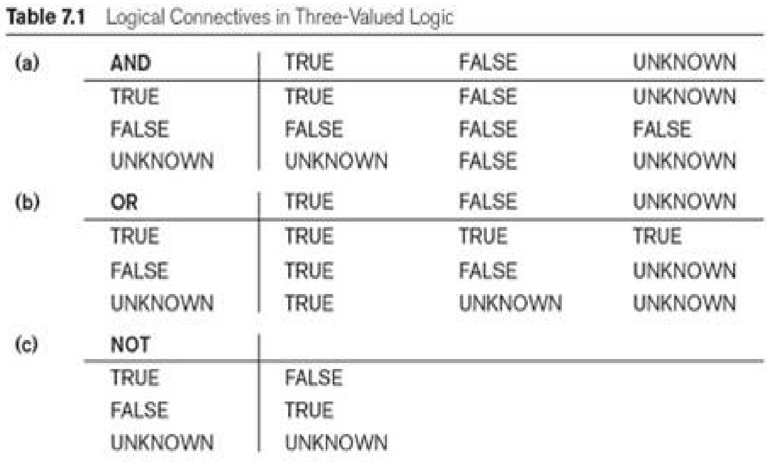
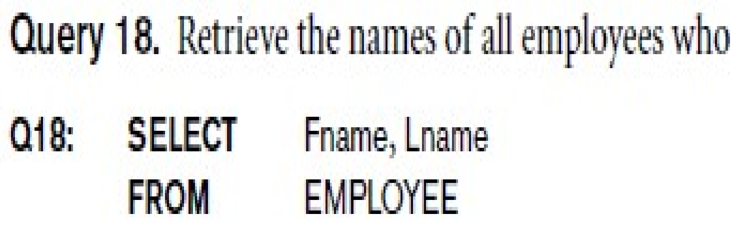


Table 7.1 Logical Connectives in Three-Valued Logic

(a)	**AND**	TRUE	FALSE	UNKNOWN
	TRUE	TRUE	FALSE	UNKNOWN
	FALSE	FALSE	FALSE	FALSE
	UNKNOWN	UNKNOWN	FALSE	UNKNOWN
(b)	**OR**	TRUE	FALSE	UNKNOWN
	TRUE	TRUE	TRUE	TRUE
	FALSE	TRUE	FALSE	UNKNOWN
	UNKNOWN	TRUE	UNKNOWN	UNKNOWN
(c)	**NOT**			
	TRUE	FALSE		
	FALSE	TRUE		
	UNKNOWN	UNKNOWN		

- SQL allows queries that check whether an attribute value is NULL
  - IS or IS NOT NULL



Query 18. Retrieve the names of all employees who

```
Q18:   SELECT   Fname, Lname
       FROM     EMPLOYEE
```

Problem Statement:

1a. Create the following table : **STUDENT** and display structure

Column Name	Data Type	Size	Constraints
RegNo	Varchar2	6	Not null
RollNo	Number	6	Not null
Name	Varchar2	10	Not null
Address	Varchar2	15	Not null
PhoneNo	Number	10	
YearOfAdm	Number	4	Not null
DeptCode	Varchar2	4	Not null
Year	Number	1	Not null
BirthDate	Date		Not null

1b. Insert the following data in the student table.

RegNo	RollNo	Name	Address	PhoneNo	YearOf Adm	DeptC ode	Year	BirthDate
012301	123001	Ashish	Jadavpur	24761892	2003	CSE	3	01-Jun-81
012315	123015	Kamal	Kasba	24424987	2003	CSE	3	19-Sep-81
012424	124024	Ipsita	Kaikhali	25739608	2004	CSE	2	15-Aug-82
012250	122050	Anita	Hooghly	36719695	2002	IT	4	22-Dec-80
012344	123044	Biplab	Howrah		2003	IT	3	03-Jan-82
012357	123057	Samik	Barasat	25426742	2003	IT	3	15-Jul-81
012419	124019	Srija	Garia	24755655	2004	EE	2	25-Oct-82
012427	124027	Saibal	Garia	24753306	2004	ECE	2	22-Mar-83
012236	122036	Santanu	DumDum		2002	ECE	4	11-Dec-80
012349	123049	Gita	Kasba	24428682	2003	MCA	3	14-Apr-81

1c. Display all records of students

1d. Display name, address and year of admission of each student

1e. List the name and year of students who are in Computer Science.

1f. List the names and departments of students belonging to 3rd year.

1g. Display names of students with ‘a’ as the second letter in their names.

1h. Display names of students in alphabetical order.

1i. Display names and addresses of students who took admission in the year 2004.

1j. List the names of students who do not have a phone number.

<u>Solution to lab Assignment 1:</u>

1a.
```
create table student4161 (RegNo varchar2(6) NOT NULL, RollNo number(6) NOT NULL, Name
varchar2(10) NOT NULL, Address varchar2(15) NOT NULL, PhoneNo number(10), YearOfAdm
number(4) NOT NULL, DeptCode varchar2(4) NOT NULL, Year number(1) NOT NULL, BirthDate Date
NOT NULL);
```

1b.
```
insert into student4161 values(012301,123001,'Ashish','Jadavpur',24761892,2003,'CSE',3,'01-Jun-81');

insert into student4161
values(012315,123015,'Kamal','Kasba',24424987,2003,'CSE',3,'19-Sep-81');

insert into student4161
values(012424,124024,'Ipsita','Kaikhali',25739608,2004,'CSE',2,'15-Aug-82');

insert into student4161
values(012250,122050,'Anita','Hooghly',36719695,2002,'IT',4,'22-Dec-80');

insert into student4161
```

```
values(012344,123044,'Biplab','Howrah',",2003,'IT',3,'03-Jan-82');

insert into student4161
values(012357,123057,'Samik','Barasat',25426742,2003,'IT',3,'15-Jul-81');

insert into student4161
values(012419,124019,'Srija','Garia',24755655,2004,'EE',2,'25-Oct-82');

insert into student4161
values(012427,124027,'Saibal','Garia',24753306,2004,'ECE',2,'22-Mar-83');

insert into student4161
values(012236,122036,'Santanu','DumDum',",2002,'ECE',4,'11-Dec-80');

insert into student4161
values(012349,123049,'Gita','Kasba',24428682,2003,'MCA',3,'14-Apr-81');
```

1c. select * from student4161;

```
REGNO  ROLLNO  NAME     ADDRESS    PHONENO   YEAROFADM  DEPT  YEAR   BIRTHDATE
------ ---------- ---------- -------------- ---------- ---------- ---- --------------------------------------------- ---------
12301   123001   Ashish   Jadavpur   24761892   2003       CSE    3    01-JUN-81

12315   123015   Kamal    Kasba      24424987   2003       CSE    3    19-SEP-81

12424   124024   Ipsita   Kaikhali   25739608   2004       CSE    2    15-AUG-82

12250   122050   Anita    Hooghly    36719695   2002       IT     4    22-DEC-80

12357   123057   Samik    Barasat    25426742   2003       IT     3    15-JUL-81

12419   124019   Srija    Garia      24755655   2004       EE     2    25-OCT-82

12427   124027   Saibal   Garia      24753306   2004       ECE    2    22-MAR-83

12349   123049   Gita     Kasba      24428682   2003       MCA    3    14-APR-81

12344   123044   Biplab   Howrah     23345678   2003       IT     3    03-JAN-82

12236   122036   Santanu  DumDum                2002       ECE    4    11-DEC-80
```

```
    1d.  select Name,Address,YearOfAdm from student4161;

NAME       ADDRESS         YEAROFADM
----------  ---------------  ----------
Ashish      Jadavpur          2003
Kamal       Kasba             2003
Ipsita      Kaikhali          2004
Anita       Hooghly           2002
Samik       Barasat           2003
Srija       Garia             2004
Saibal      Garia             2004
Gita        Kasba             2003
Biplab      Howrah            2003
Santanu     DumDum            2002


10 rows selected.
```

```
    1e.  select Name,Year from student4161 where DeptCode='CSE';

NAME          YEAR
----------    ----------
Ashish          3
Kamal           3
Ipsita          2
```

```
    1f.  select Name,DeptCode from student4161  where Year=3;

NAME       DEPT
----------  --------
Ashish      CSE
Kamal       CSE
Samik       IT
Gita        MCA
Biplab      IT
```

```
    1g. select Name from student4161  where Name like '_a%';

NAME
----------
Kamal
Samik
Saibal
Santanu
```

1h. select Name from student4161 Order by Name asc;

```
NAME
----------
Anita
Ashish
Biplab
Gita
Ipsita
Kamal
Saibal
Samik
Santanu
Srija


10 rows selected.
```

1i. select Name,Address from student4161 where YearOfAdm=2004;

```
NAME       ADDRESS
---------- ---------------
Ipsita     Kaikhali
Srija      Garia
Saibal     Garia
```

1j. select Name from student4161 where PhoneNo is NULL;

```
NAME
----------
Biplab
Santanu
```

Lab Assignment 2

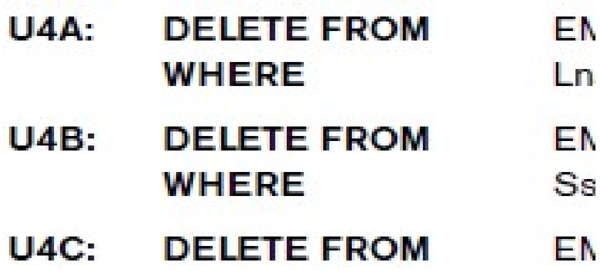
Topic : Use of DML - select rows, delete rows and update table operations

Readings:

NOTE: Readings for Previous Assignments are also required for this lab assignment

The DELETE Command

- Removes tuples from a relation
    - Includes a WHERE-clause to select the tuples to be deleted
    - Referential integrity should be enforced
    - Tuples are deleted from only *one table* at a time (unless CASCADE is specified on a referential integrity constraint)
    - A missing WHERE-clause specifies that *all tuples* in the relation are to be deleted; the table then becomes an empty table
    - The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause



```
U4A:    DELETE FROM     EN
        WHERE           Ln
U4B:    DELETE FROM     EN
        WHERE           Ss
U4C:    DELETE FROM     EN
```

The UPDATE Command

- Used to modify attribute values of one or more selected tuples
- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples *in the same relation*
- Referential integrity specified as part of DDL specification is enforced

Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively

```
U5:     UPDATE          PROJECT
        SET     PLOCATION = 'Bellaire',         DNUM = 5
        WHERE           PNUMBER=10
```

Example: Give all employees in the 'Research' department a 10% raise in salary.

```
U6:    UPDATE     EMPLOYEE
       SET        SALARY = SALARY *1.1
       WHERE      DNO  IN (SELECT     DNUMBER
                    FROM      DEPARTMENT
                    WHERE     DNAME='Research')
```

- In this request, the modified SALARY value depends on the original SALARY value in each tuple
  - The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification
- The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification

The DROP Command

- DROP command
  - Used to drop named schema elements, such as tables, domains, or constraint
- Drop behavior options:
  - CASCADE and RESTRICT
- Example:
  - DROP SCHEMA COMPANY CASCADE;
  - This removes the schema and all its elements including tables,views, constraints, etc.

The ALTER table command

- Alter table actions include:
  - Adding or dropping a column (attribute)
  - Changing a column definition
  - Adding or dropping table constraints
- Example:
  - ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);

Adding and Dropping Constraints

- Change constraints specified on a table
  - Add or drop a named constraint

```
ALTER TABLE COMPANY.EMPLC
```

Dropping Columns, Default Values

- To drop a column
  - Choose either CASCADE or RESTRICT
  - CASCADE would drop the column from views etc. RESTRICT is possible if no views refer to it.

```
ALTER TABLE COMPANY.EMPLOYEE DROP COLUMN    Address CASCADE;
```

- Default values can be dropped and altered :

```
ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn DROP DEFAULT;

ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn SET DEFAULT
‘333445555’;
```

Problem Statement:

Note : Tables created previously in lab assignments may be used if required

2a. Delete the name of a student whose roll no, year and department code is given.
2b. Display the number of students in each department.
2c. Change the address of a student whose roll no and name is given.
2d. Add the college phone number (25739607) to each of these students.
2e. Change the size of column Name to 15 characters.
2f. Add a column MarksObtained (number) to the student table.
2g. Insert values against marks column.
2h. Drop column MarksObtained from table student.
2i. Add constraint primary key to the column RegNo of table student.
2j. Add check constraints to the column year of student table. (year should be entered within 1,2,3,4).

Solution to Lab Assignment 2:

2a. delete from student4161 where RollNo=122050 AND Year=4 AND DeptCode='IT';

2b. select DeptCode, count(*) from student4161 group by DeptCode;

```
DEPT   COUNT(*)
----   ----------
CSE        3
ECE        2
EE         1
IT         2
MCA        1
```

2c. Update student4161 set Address='Birati' where RollNo=124027;
2d. Alter table student4161 add ColPhone number(10);

Table altered.

Update student4161 ColPhone=25739607;

9 rows updated.

2e. Alter table student4161 modify Name varchar2(15);
2f. Alter table student4161 add Marks number(10);

Table altered.

2g. update student4161 set Marks=99 where DeptCode='CSE';

update student4161 set Marks=80 where DeptCode='IT';

update student4161 set Marks=85 where DeptCode='ECE';

update student4161 set Marks=70 where DeptCode='EE';

update student4161 set Marks=75 where DeptCode='MCA';

Select Name,Marks from student4161;

```
NAME            MARKS
---------------  ----------
Ashish          99
Kamal            99
Ipsita           99
Biplab           80
Samik             80
Srija            70
Saibal           85
Santanu          85
Gita             75
```

9 rows selected.

2h. Alter table student4161 drop(Marks);
2i. Alter table student4161 add primary key (RegNo);
2j. Alter table student4161 add check(Year>=1 and Year<=4);

Table altered.

Lab Assignment 3

Topic : Use of DDL - Alter Table Statement, Check Constraints, Foregin Key constraints in SQL

Readings:

NOTE: Readings for Previous Assignments are sufficient for this lab assignment

Problem Statement:

Note : Tables created previously in lab assignments may be used if required

3a. Create table **DEPARTMENT**

Column Name	Data Type	Size	Constraints
DeptCode	Varchar2	4	Not null, Primary key
DeptName	Varchar2	15	Not null
HOD	Varchar2	4	Not null

FACULTY

Column Name	Data Type	Size	Constraints
FacultyCode	Varchar2	4	Not null, Primary key, Starts with 'F'
FacultyName	Varchar2	15	Not null
DateOfJoin	Date		Not null
DeptCode	Varchar2	4	Must be either CSE,IT, CA, CHEM, MTHS, PHYS, HUM, BBA

3b. Insert appropriate values in the above table.
3c. Add constraint : DeptCode of Faculty is foreign key and references DeptCode in Department
3d. Find the names of faculties of CSE Department.
3e. Find the number of faculties in the Computer application department
3f. Show the names of the heads of departments with department name.
3g. Find the number of faculties who joined in August.
3h. Add an extra attribute to the faculty table - Salary Number(8,2)
3i. Insert values into the corresponding field Salary Number(8,2).
3j. Find the name and salary of the faculty who earn more than 8000.
3k. Find the name, department of the faculties who earn between 8000 and 12000.

Solution to lab Assignment 3:

```
3a. create table faculty4161 (FacultyCode varchar2(4) PRIMARY KEY, FacultyName varchar2(15) NOT
    NULL, DateOfJoin Date NOT NULL, DeptCode varchar2(4));

    create table DEPARTMENT41(DeptCode varchar2(4) PRIMARY KEY, DEptName varchar2(15)
    NOT NULL, HOD varchar2(4));

    Alter table DEPARTMENT41 add FOREIGN KEY(HOD) references faculty4161(FacultyCode);

    Alter table faculty41 add check(FacultyCode like 'F%');

    Alter table faculty41 add CHECK (DeptCode IN ('CSE', 'IT',
    'CA','CHEM','MTHS','PHY','BBA','HUM')

3b. insert into faculty41 values('F01','S.Chakraborty','22-Dec-01','IT');

    insert into faculty41 values('F02','M.Mohanto','10-May-02','CSE');

    insert into faculty41 values('F03','S.M.Roy','15-Aug-01','CSE');

    insert into faculty41 values('F04','K.K.Patil','20-Aug-02','CA');

    insert into faculty41 values('F05','S.C.Kareem','10-Jun-01','CHEM');

    insert into faculty41 values('F06','P.Roy','10-Feb-02','HUM');

    insert into faculty41 values('F07','M.Singh','11-Jul-02','BBA');

    insert into faculty41 values('F08','P.Mukherjee','10-Sep-02','MTHS');

    insert into faculty41 values('F09','K.Mondal','22-Oct-01','PHYS');

    insert into faculty41 values('F10','B.Das','10-May-01','IT');

    insert into faculty41 values('F11','M.Dasgupta','11-Feb-02','CSE');

    insert into DEPARTMENT41 values('CSE','Computer Sci','F03');

    insert into DEPARTMENT41 values('IT','InfoTech','F01');

    insert into DEPARTMENT41 values('CA','Comp Appli.','F04');

    insert into DEPARTMENT41 values('CHEM','Chemistry','F05');

    insert into DEPARTMENT41 values('MTHS','MAthematics','F08');

    insert into DEPARTMENT41 values('PHYS','Physics','F09');
    insert into DEPARTMENT41 values('HUM','Humanities','F06');

    insert into DEPARTMENT41 values('BBA','Busi. Admns','F07');
```

3c. Alter table Faculty41 add FOREIGN KEY(DeptCode) references DEPARTMENT41(DeptCode);

3d. Select FacultyName from faculty41 where DeptCode='CSE';

```
FACULTYNAME
---------------
M.Mohanto
S.M.Roy
M.Dasgupta
```

3e. Select FacultyName from faculty41 where DeptCode='CA';

```
FACULTYNAME
---------------
K.K.Patil
```

3f. Select FacultyName,DeptName from faculty41,DEPARTMENT41 where DEPARTMENT41.HOD=faculty41.FacultyCode;

```
FACULTYNAME     DEPTNAME
---------------  ---------------
S.M.Roy          Computer Sci
S.Chakraborty    InfoTech
K.K.Patil        Comp Appli.
S.C.Kareem       Chemistry
P.Mukherjee      MAthematics
K.Mondal         Physics
P.Roy            Humanities
M.Singh          Busi. Admns
```

3g. Select count(FacultyCode) from faculty41where DateOfJoin like '%AUG%';

```
COUNT(FACULTYCODE)
------------------
        2
```

3h. Alter table Faculty41 add salary number(8,2);

3i.

```
Update faculty41 set salary=15000 where FacultyCode='F01';

Update faculty41 set salary=7000 where FacultyCode='F02';

Update faculty41 set salary=25000 where FacultyCode='F03';

Update faculty41 set salary=10000 where FacultyCode='F04';

Update faculty41 set salary=10000.50 where FacultyCode='F05';

Update faculty41 set salary=12500 where FacultyCode='F06';

Update faculty41 set salary=15050 where FacultyCode='F07';

Update faculty41 set salary=11200.75 where FacultyCode='F08';

Update faculty41 set salary=12000 where FacultyCode='F09';

Update faculty41 set salary=11000 where FacultyCode='F10';

Update faculty41 set salary=5000 where FacultyCode='F11';
```

3j.

```
select FacultyName,salary from faculty41 where salary>=8000;
```

```
FACULTYNAME        SALARY
---------------    ----------
S.Chakraborty      15000
S.M.Roy            25000
K.K.Patil          10000
S.C.Kareem         10000.5
P.Roy              12500
M.Singh            15050
P.Mukherjee        11200.75
K.Mondal           12000
B.Das              11000
```

3k.

```
select FacultyName,DeptCode from faculty41 where salary between 8000 and 12000;
```

```
FACULTYNAME     DEPT
---------------  ----
K.K.Patil       CA
S.C.Kareem      CHEM
P.Mukherjee     MTHS
K.Mondal        PHYS
B.Das           IT
```

Lab Assignment 4

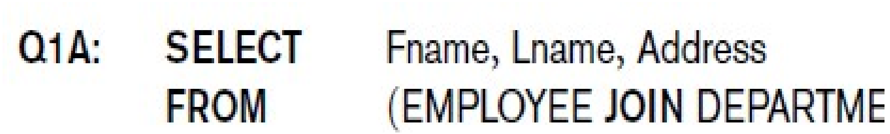
Topic : Join Operations Cartesian Product, Natural Join, Outer Join

Readings:

NOTE: Readings for Previous Assignments are also required for this lab assignment

Specifying Joined Tables in the FROM Clause of SQL

- Joined table
  - Permits users to specify a table resulting from a join operation in the FROM clause of a query
- The FROM clause in Q1A
  - Contains a single joined table. JOIN may also be called INNER JOIN



```
Q1A:   SELECT   Fname, Lname, Address
       FROM     (EMPLOYEE JOIN DEPARTME
```

Different Types of JOINed Tables in SQL

- Specify different types of join
  - NATURAL JOIN
  - Various types of OUTER JOIN (LEFT, RIGHT, FULL)
- NATURAL JOIN on two relations R and S
  - No join condition specified
  - Is equivalent to an implicit EQUIJOIN condition for each pair of attributes with same name from R and S

NATURAL JOIN

- Rename attributes of one relation so it can be joined with another using NATURAL JOIN:

```
Q1B:   SELECT   Fname, Lname, Address
        FROM    (EMPLOYEE NATURAL JOIN
                (DEPARTMENT AS DEPT (Dname, Dno, Mssn,
                 Msdate)))
```

```
            WHERE       Dname='Research';
```

The above works with EMPLOYEE.Dno = DEPT.Dno as an implicit join condition

INNER and OUTER Joins

- INNER JOIN (versus OUTER JOIN)
    - Default type of join in a joined table
    - Tuple is included in the result only if a matching tuple exists in the other relation
- LEFT OUTER JOIN
    - Every tuple in left table must appear in result
    - If no matching tuple
        - Padded with NULL values for attributes of right table

```
SELECT E.Lname AS Employee_Name
            S.Lname AS Supervisor_Name
FROM Employee AS E LEFT OUTER JOIN EMPLOYEE AS S
                       ON E.Super_ssn = S.Ssn)
```

ALTERNATE SYNTAX:

```
SELECT E.Lname , S.Lname
FROM  EMPLOYEE E, EMPLOYEE S
WHERE E.Super_ssn + = S.Ssn
```

- RIGHT OUTER JOIN
    - Every tuple in right table must appear in result
    - If no matching tuple
        - Padded with NULL values for attributes of left table

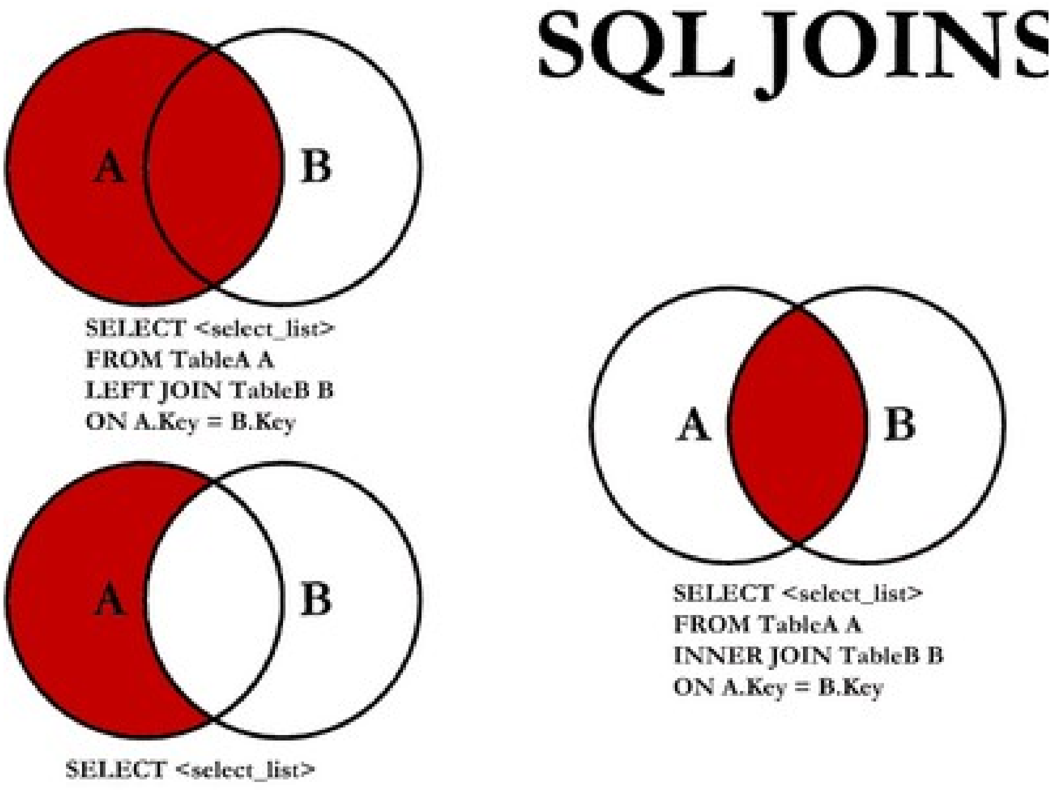
Multiway JOIN in the FROM clause

- FULL OUTER JOIN – combines result if LEFT and RIGHT OUTER JOIN
- Can nest JOIN specifications for a multiway join:

```
Q2A:    SELECT Pnumber, Dnum, Lname, Address, Bdate
                FROM     ((PROJECT JOIN DEPARTMENT ON
                       Dnum=Dnumber)  JOIN EMPLOYEE ON
```

```
                    Mgr_ssn=Ssn)
              WHERE   Plocation='Stafford';
```

Summary of SQL Joins



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
```

Problem Statement:

Note : Tables created previously in lab assignments may be used if required

4a. Create table **SUBJECT** and insert appropriate values.

Column Name	**Data Type**	**Size**	**Constraints**
SubjectCode	Varchar2	4	Not null, Primary key
SubjectName	Varchar2	15	Not null
Faculty	Varchar2	4	Foreign key references FacultyCode of table **FACULTY**

4b. Find the number of faculties in each department with their department name.

4c. Increment the salary of each faculty by Rs 500.

4d. Find the names of students and faculties whose name start with ‘S’.
4e. Find the students who stay in Kaikhali
4f. Find the names of faculties who take classes in the IT department.
4g. Find the names of all faculties whose HOD is given.

Solution to Lab Assignment 4:

4a. create table subject41(SubjectCode varchar2(4) PRIMARY KEY, SubjectName varchar2(15) NOT NULL, Faculty varchar2(4), FOREIGN KEY(Faculty) references faculty41(FacultyCode));

insert into subject41 values('I21','Control System','F01');

insert into subject41 values('C01','DBMS','F11');

insert into subject41 values('H23','Public Speaking','F06');

insert into subject41 values('B22','Economics','F07');

insert into subject41 values('M25','Basic Algebra','F08');

insert into subject41 values('P29','Aerodynamics','F09');

insert into subject41 values('CH11','Organic Chem','F05');

insert into subject41 values('CA31','PPL','F04');

4b. SELECT DeptName,Count(DeptName) from Faculty41 natural join DEPARTMENT41 group by DeptName;

4c. update faculty41 set salary=salary+500;

4d. select student4161.NAME,faculty41.FacultyName from student4161,faculty41 where student4161.NAME like 'S%' AND faculty41.FacultyName like 'S%';

```
NAME       FACULTYNAME
---------- ---------------
Samik      S.Chakraborty
Srija      S.Chakraborty
Saibal     S.Chakraborty
Santanu    S.Chakraborty
Samik      S.M.Roy
```

```
Srija      S.M.Roy
Saibal     S.M.Roy
Santanu    S.M.Roy
Samik      S.C.Kareem
Srija      S.C.Kareem
Saibal     S.C.Kareem
Santanu   S.C.Kareem

        12 rows selected.
```

```
    4e. select NAME from student4161 where ADDRESS='Kaikhali';

NAME
----------
Ipsita
```

```
    4f.  select FacultyName from faculty41 where DeptCode='IT';

FACULTYNAME
---------------
S.Chakraborty
B.Das
```

```
    4g. select FacultyName,FacultyCode from faculty41, DEPARTMENT41 where FacultyCode=HOD;

FACULTYNAME     FACULTYCODE
---------------    ----
S.M.Roy            F03
S.Chakraborty      F01
K.K.Patil          F04
S.C.Kareem         F05
P.Mukherjee        F08
K.Mondal           F09
P.Roy              F06
M.Singh            F07

        8 rows selected.
```

B. P. Poddar Institute of Management and Technology
Department of Computer Science and Engineering
Academic Year : 2017-18 [Even Semester]

Lab Assignment 5

Topic : Queries using aggregate functions (count,sum,avg,max,min) and group by, having

Readings:

NOTE: Readings for Previous Assignments are also required for this lab assignment

Aggregate Functions in SQL

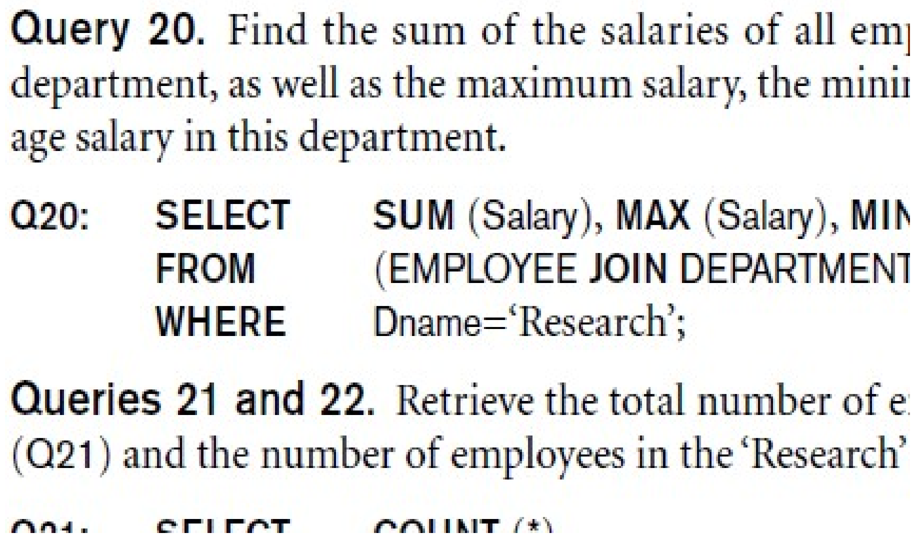
- Used to summarize information from multiple tuples into a single-tuple summary
- Built-in aggregate functions
    - COUNT, SUM, MAX, MIN, and AVG
- Grouping
    - Create subgroups of tuples before summarizing
- To select entire groups, HAVING clause is used
- Aggregate functions can be used in the SELECT clause or in a HAVING clause
- Following query returns a single row of computed values from EMPLOYEE table:

```
Q19:        SELECT   SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
               FROM  EMPLOYEE;
```

- The result can be presented with new names:

```
Q19A:       SELECT   SUM (Salary) AS Total_Sal, MAX (Salary) AS Highest_Sal,
                     MIN (Salary) AS Lowest_Sal, AVG (Salary) AS Average_Sal
               FROM  EMPLOYEE;
```

- NULL values are discarded when aggregate functions are applied to a particular column



Query 20. Find the sum of the salaries of all em
department, as well as the maximum salary, the mini
age salary in this department.

```
Q20:    SELECT    SUM (Salary), MAX (Salary), MIN
        FROM      (EMPLOYEE JOIN DEPARTMENT
        WHERE     Dname='Research';
```

Queries 21 and 22. Retrieve the total number of e
(Q21) and the number of employees in the 'Research'

```
Q21:    SELECT    COUNT (*)
```

Aggregate Functions on Booleans

- SOME and ALL may be applied as functions on Boolean Values.
- SOME returns true if at least one element in the collection is TRUE (similar to OR)
- ALL returns true if all of the elements in the collection are TRUE (similar to AND)

Grouping: The GROUP BY Clause

- Partition relation into subsets of tuples
  - Based on grouping attribute(s)
  - Apply function to each such group independently
- GROUP BY clause
  - Specifies grouping attributes
- COUNT (*) counts the number of rows in the group
- The grouping attribute must appear in the SELECT clause:

```
Q24:          SELECT        Dno, COUNT (*), AVG (Salary)
              FROM          EMPLOYEE
              GROUP BY      Dno;
```

- If the grouping attribute has NULL as a possible value, then a separate group is created for the null value (e.g., null Dno in the above query)
- GROUP BY may be applied to the result of a JOIN:

```
Q25:            SELECT      Pnumber, Pname, COUNT (*)
                FROM        PROJECT, WORKS_ON
                WHERE       Pnumber=Pno
                GROUP BY    Pnumber, Pname;
```

Grouping: The GROUP BY and HAVING Clauses

- HAVING clause
    - Provides a condition to select or reject an entire group:
- Query 26. For each project *on which more than two employees work,* retrieve the project number, the project name, and the number of employees who work on the project.
- Q26:

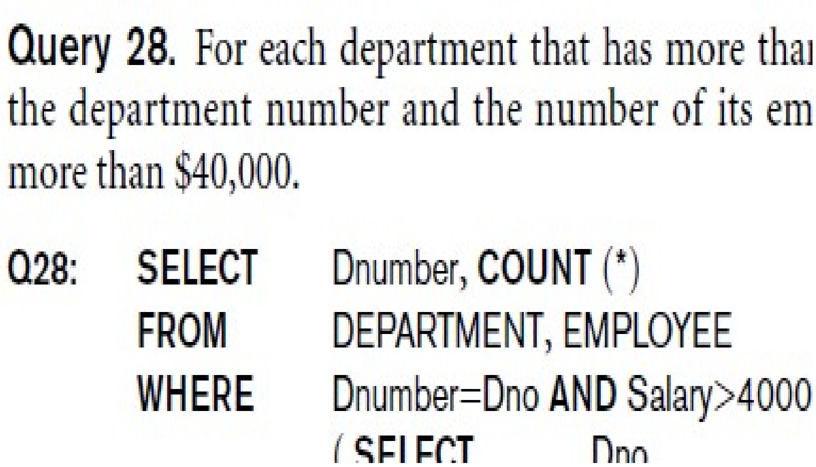
```
Q26:            SELECT      Pnumber, Pname, COUNT (*)
                FROM        PROJECT, WORKS_ON
                WHERE       Pnumber=Pno
                GROUP BY    Pnumber, Pname
                HAVING      COUNT (*) > 2;
```

Combining the WHERE and the HAVING Clause

- Consider the query: we want to count the *total* number of employees whose salaries exceed $40,000 in each department, but only for departments where more than five employees work.
- INCORRECT QUERY:

```
SELECT          Dno, COUNT (*)
FROM            EMPLOYEE
WHERE           Salary>40000
GROUP BY        Dno
HAVING          COUNT (*) > 5;
```

- Correct Specification of the Query:
  Note: the WHERE clause applies tuple by tuple whereas HAVING applies to entire group of tuples



Query 28. For each department that has more tha
the department number and the number of its em
more than $40,000.

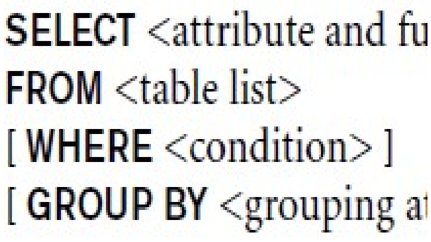
```
Q28:  SELECT   Dnumber, COUNT (*)
      FROM     DEPARTMENT, EMPLOYEE
      WHERE    Dnumber=Dno AND Salary>4000
               ( SELECT        Dno
```

Use of WITH

- The WITH clause allows a user to define a table that will only be used in a particular query (not available in all SQL implementations)
- Used for convenience to create a temporary "View" and use that immediately in a query
- Allows a more straightforward way of looking a step-by-step query
- See an alternate approach to doing Q28:

```
Q28':        WITH  BIGDEPTS (Dno) AS
                   (SELECT      Dno
                   FROM         EMPLOYEE
                   GROUP BY     Dno
                   HAVING       COUNT (*) > 5)
             SELECT       Dno, COUNT (*)
             FROM         EMPLOYEE
             WHERE        Salary>40000 AND Dno IN BIGDEPTS
             GROUP BY Dno;
```

EXPANDED Block Structure of SQL Queries



```
SELECT <attribute and fu
FROM <table list>
[ WHERE <condition> ]
[ GROUP BY <grouping at
```

Problem Statement:

Note : Tables created previously in lab assignments may be used if required

5a. Add extra attribute to the Subject table - department varchar2 (4), year varchar2 (1)
5b. Insert values into the fields - department, year.
5c. Find the maximum salary among the faculties.
5d. Find the names of faculties who earn more than the average of all faculties.
5e. List the names of faculties of CSE department who earn more than the average salary of the department.
5f. Find the maximum and minimum salaries among faculties.
5g. Find the second maximum salary among all faculties.
5h. Find the names of faculties who are not the HOD's of any department.
5i. Find the names of subjects for students of CSE 3rd year.

Solution to Lab Assignment 5

5a. Alter table subject41 add department varchar2(4);

Alter table subject41 add year varchar2(1);

5b. update subject41 set department='CSE',year='1' where faculty='F11';

update subject41 set department='IT',year='2' where faculty='F01';

update subject41 set department='CA',year='2' where faculty='F04';

update subject41 set department='CHEM',year='1' where faculty='F05';

update subject41 set department='HUM',year='3' where faculty='F06';

update subject41 set department='BBA',year='3' where faculty='F07';

update subject41 set department='MTHS',year='1' where faculty='F08';

update subject41 set department='PHYS',year='2' where faculty='F09';

select * from subject41;

```
SUBJ   SUBJECTNAME      FACU  DEPA  Y
----   ---------------  ----  ----  -
I21    Control System   F01   IT    2
H23    Public Speaking  F06   HUM   3
B22    Economics        F07   BBA   3
P29    Aerodynamics     F09   PHYS  2
CA31   PPL              F04   CA    2
CH11   Organic Chem     F05   CHEM  1
C01    DBMS             F11   CSE   1
M25    Basic Algebra    F08   MTHS  1
```

```
8 rows selected.
```

5c. select FacultyName,salary from faculty41 where salary>all(select avg(salary) from faculty41);

```
FACULTYNAME        SALARY
---------------    ----------
S.Chakraborty      15500
S.M.Roy            25500
P.Roy              13000
M.Singh            15550
```

5d. select FacultyName,salary from faculty41 where DeptCode='CSE' and salary>all(select avg(salary) from faculty41 where DeptCode='CSE');

```
FACULTYNAME        SALARY
---------------    ----------
S.M.Roy             25500
```

5e. select max(salary),min(salary) from faculty41;

```
MAX(SALARY)    MIN(SALARY)
-----------    -----------
     25500            5500
```

5f. select max(salary) from faculty41 where salary not in (select max(salary) from faculty41);

```
MAX(SALARY)
-----------
     15550
```

5g. select FacultyName from faculty41 where FacultyCode not in (select HOD from DEPARTMENT41);

```
FACULTYNAME
---------------
M.Mohanto
B.Das
M.Dasgupta
```

5h. select SubjectName from subject41 where Department='CSE' and year='1';

```
SUBJECTNAME
---------------
DBMS
```

Lab Assignment 6

Topic : Creation and Dropping of Views

Readings:

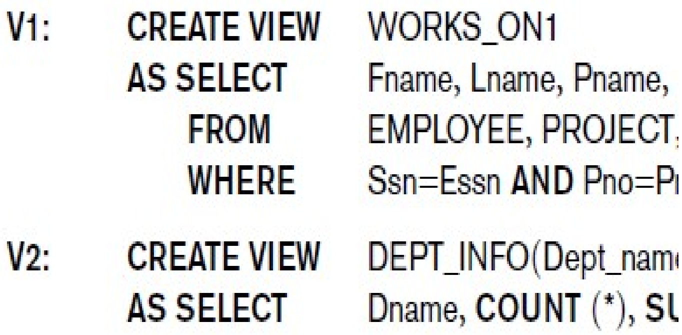
NOTE: Readings for Previous Assignments are also required for this lab assignment

Views (Virtual Tables) in SQL

- Concept of a view in SQL
  - Single table derived from other tables called the defining tables
  - Considered to be a virtual table that is not necessarily populated

Specification of Views in SQL

- CREATE VIEW command
  - Give table name, list of attribute names, and a query to specify the contents of the view
  - In V1, attributes retain the names from base tables. In V2, attributes are assigned names



```
V1:    CREATE VIEW    WORKS_ON1
       AS SELECT      Fname, Lname, Pname,
           FROM       EMPLOYEE, PROJECT,
           WHERE      Ssn=Essn AND Pno=Pı

V2:    CREATE VIEW    DEPT_INFO(Dept_nam
       AS SELECT      Dname, COUNT (*), SU
```

- Once a View is defined, SQL queries can use the View relation in the FROM clause
- View is always up-to-date
  - Responsibility of the DBMS and not the user
- DROP VIEW command
  - Dispose of a view

View Update

- Update on a view defined on a single table without any aggregate functions
    - Can be mapped to an update on underlying base table- possible if the primary key is preserved in the view
- Update not permitted on aggregate views. E.g.,

```
UV2:  UPDATE            DEPT_INFO
      SET               Total_sal=100000
      WHERE             Dname='Research';
```

cannot be processed because Total_sal is a computed value in the view definition

View Update and Inline Views

- View involving joins
    - Often not possible for DBMS to determine which of the updates is intended
- Clause WITH CHECK OPTION
    - Must be added at the end of the view definition if a view is to be updated to make sure that tuples being updated stay in the view
- In-line view
    - Defined in the FROM clause of an SQL query (e.g., we saw its used in the WITH example)

Views as authorization mechanism

- SQL query authorization statements (GRANT and REVOKE) are described in detail later
- Views can be used to hide certain attributes or tuples from unauthorized users
- E.g., For a user who is only allowed to see employee information for those who work for department 5, he may only access the view DEPT5EMP:

```
CREATE VIEW       DEPT5EMP   AS
SELECT            *
FROM        EMPLOYEE
WHERE             Dno = 5;
```

Problem Statement:

Note : Tables created previously in lab assignments may be used if required

6a. Name the departments having highest number of faculties and display the names of faculties

6b. Create a view on the STUDENT table named V_STD selecting all the columns. Run the following queries on the view.

v. Display all data from the view.
vi. Insert a new row into the view with the following data –

012363	123011	Bishakh	Salt Lake	23371987	2005	IT	2	01-May-82

vii. Display data from student table to verify that the row has been inserted into the Table.
viii. Update the address of Bishakh to “SectorV” & verify the change in the table.

6c. Create a view on student table snamed V_STD_2 selecting the columns – RegNo, Name, Year, Deptcode.

v. Display data from the view.
vi. Try to insert data into table through view.
vii. Update the Deptcode of ‘Kamal’ to ‘IT’ through view.
viii. Delete records of students of 4th year through view.

6d. Create a view named V_FACULTY consisting of columns FacultyName, DeptCode from FACULTY table and HOD from Department table.

iv. Display data from V_FACULTY
v. Try to insert a new row into this view V_FACULTY.
vi. Try to update the DeptCode of a CSE faculty to IT.

Solution to Lab Assignment 6:

6a. create view countfaculty41_view as(select DeptCode, count(FacultyCode) as S from faculty41 group by DeptCode);

select * from countfaculty41_view;

```
DEPT      S
----     ----------
BBA       1
CA        1
CHEM      1
CSE       3
HUM       1
IT        2
MTHS      1
PHYS      1


8 rows selected.
```

```
select DeptCode from countfaculty41_view where S in (select max(S) from countfaculty41_view);
```

```
DEPT
----
CSE
```

```
select FacultyName from faculty41 where DeptCode in (select DeptCode from countfaculty41_view where
S in (select max(S) from countfaculty41_view));
```

```
FACULTYNAME
---------------
M.Mohanto
S.M.Roy
M.Dasgupta
```

6b. i.

```
CREATE VIEW V_STD AS SELECT * FROM STUDENT;
SELECT * FROM V_STD;
```

ii.

```
INSERT INTO V_STD VALUES('012363',123011,'Bishakh','Salt
Lake',23371987,2005,'IT',2,'01-May-82');
```

iii.

```
SELECT * FROM STUDENT;
```

iv.

```
UPDATE STUDENT SET ADDRESS='SECTOR V' WHERE ROLLNO='123011';

SELECT * FROM STUDENT;
```

6c. i.

```
CREATE VIEW V_STD_2 AS SELECT REGNO,NAME,YEAR,DEPTCODE
FROM STUDENT;

SELECT * FROM V_STD_2;
```

ii.

```
INSERT INTO V_STD_2 VALUES('12345','SASWATA',3,'CSE');
```

iii.

```
UPDATE V_STD_2 SET DEPTCODE='IT' WHERE REGNO='012315';

SELECT * FROM V_STD_2;
```

iv.

```
DELETE FROM V_STD_2 WHERE YEAR='4';

SELECT * FROM V_STD_2;
```

6d. i.

```
CREATE VIEW V_FACULTY AS SELECT FACULTY.FACULTYNAME,
FACULTY.DEPTCODE,DEPARTMENT.HOD FROM DEPARTMENT INNER JOIN
FACULTY ON FACULTY.DEPTCODE=DEPARTMENT.DEPTCODE;
```

```
SELECT * FROM V_FACULTY;
```

ii.

```
INSERT INTO V_FACULTY VALUES('PRIYA DAS','MTHS','fmt1');
```

```
Error : Cannot modify more than one base table through a join view
```

iii.

```
UPDATE V_FACULTY SET DEPTCODE='IT' WHERE FACULTYNAME='Saswata Das';

SELECT * FROM V_FACULTY;
```

Lab Assignment 7

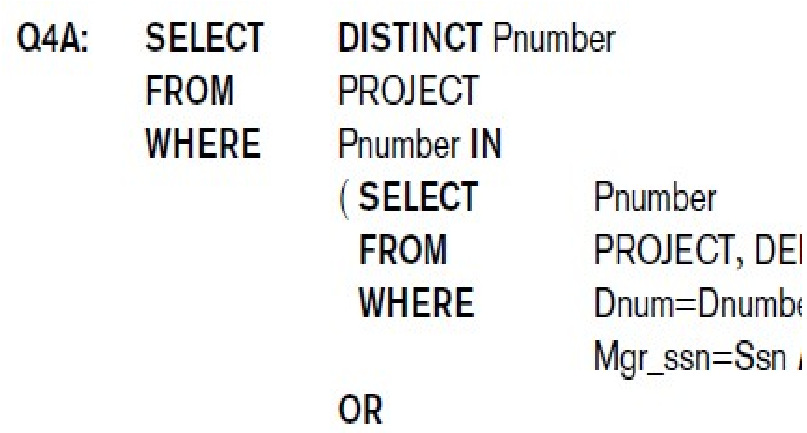
Topic : Nested Queries using any, all in, exist, not exists, unique, intersect constraints.

Readings:

NOTE: Readings for Previous Assignments are also required for this lab assignment

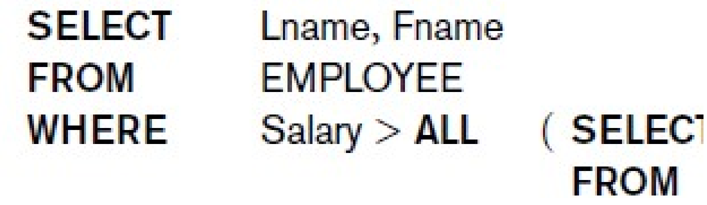
Nested Queries, Tuples, and Set/Multiset Comparisons

- Nested queries
  - Complete select-from-where blocks within WHERE clause of another query
  - Outer query and nested subqueries
- Comparison operator IN
  - Compares value *v* with a set (or multiset) of values *V*
  - Evaluates to TRUE if *v* is one of the elements in *V*



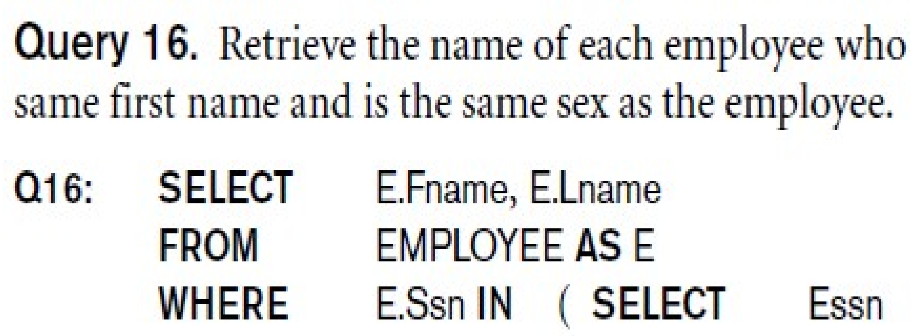
```
Q4A:   SELECT    DISTINCT Pnumber
       FROM      PROJECT
       WHERE     Pnumber IN
                 ( SELECT      Pnumber
                   FROM        PROJECT, DE
                   WHERE       Dnum=Dnumb
                               Mgr_ssn=Ssn
                 OR
```

- Use other comparison operators to compare a single value *v*
  - = ANY (or = SOME) operator
    - Returns TRUE if the value *v* is equal to some value in the set *V* and is hence equivalent to IN
  - Other operators that can be combined with ANY (or SOME): >, >=, <, <=, and <>
  - ALL: value must exceed all values from nested query



```
SELECT      Lname, Fname
FROM        EMPLOYEE
WHERE       Salary > ALL    ( SELEC
                              FROM
```

- Avoid potential errors and ambiguities
  - Create tuple variables (aliases) for all tables referenced in SQL query



Query 16. Retrieve the name of each employee who same first name and is the same sex as the employee.

```
Q16:    SELECT    E.Fname, E.Lname
        FROM      EMPLOYEE AS E
        WHERE     E.Ssn IN   ( SELECT     Essn
```

Correlated Nested Queries

- Queries that are nested using the = or IN comparison operator can be collapsed into one single block: E.g., Q16 can be written as:

```
Q16A:  SELECT            E.Fname, E.Lname
                         FROM        EMPLOYEE AS E, DEPENDENT AS D
                         WHERE       E.Ssn=D.Essn AND E.Sex=D.Sex
                                             AND
                                     E.Fname=D.Dependent_name;
```

- Correlated nested query
  - Evaluated once for each tuple in the outer query

The EXISTS and UNIQUE Functions in SQL for correlating queries

- EXISTS function
  - Check whether the result of a correlated nested query is empty or not. They are Boolean functions that return a TRUE or FALSE result.
- EXISTS and NOT EXISTS
  - Typically used in conjunction with a correlated nested query
- SQL function UNIQUE(Q)
  - Returns TRUE if there are no duplicate tuples in the result of query Q
  -

USE OF NOT EXISTS

```
Q7:
SELECT Fname, Lname
FROM Employee
WHERE EXISTS (SELECT *
                   FROM DEPENDENT
                  WHERE Ssn= Essn)
           AND EXISTS (SELECT   *
                          FROM Department
                          WHERE Ssn= Mgr_Ssn)
```

USE OF NOT EXISTS

- To achieve the “for all” (universal quantifier) effect, we use double negation this way in SQL:
- Query: List first and last name of employees who work on ALL projects controlled by Dno=5.

```
SELECT Fname, Lname
FROM Employee
WHERE NOT EXISTS ( (SELECT  Pnumber
                    FROM PROJECT
                    WHERE Dno=5)
            EXCEPT (SELECT   Pno
                    FROM WORKS_ON
                    WHERE Ssn= ESsn)
```

The above is equivalent to double negation: List names of those employees for whom there does NOT exist a project managed by department no. 5 that they do NOT work on.

- Q3B:

```
Q3B:  SELECT        Lname, Fname
      FROM          EMPLOYEE
      WHERE         NOT EXISTS ( SELECT * FROM WORKS_ON B
                    WHERE        ( B.Pno IN  (  SELECT Pnumber
                                 FROM PROJECT
                                 WHERE Dnum=5  AND
                                 NOT EXISTS (SELECT * FROM  WORKS_ON C
                                 WHERE  C.Essn=Ssn
                                 AND    C.Pno=B.Pno )));
```

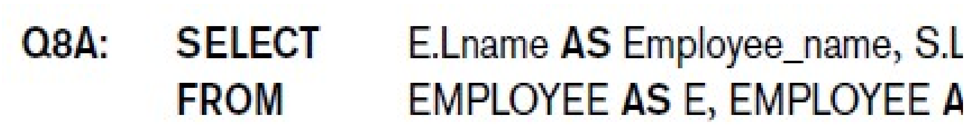
The above is a direct rendering of: List names of those employees for whom there does NOT exist a project managed by department no. 5 that they do NOT work on.

Explicit Sets and Renaming of Attributes in SQL

- Can use explicit set of values in WHERE clause

```
Q17:        SELECT      DISTINCT Essn
            FROM        WORKS_ON
            WHERE       Pno IN (1, 2, 3);
```

- Use qualifier AS followed by desired new name
  - Rename any attribute that appears in the result of a query



```
Q8A:    SELECT    E.Lname AS Employee_name, S.L
        FROM      EMPLOYEE AS E, EMPLOYEE A
```

Problem Statement:

Note : Tables created previously in lab assignments may be used if required

Considering -

Branch Schema <branch-name, branch-city, assets>
Customer Schema <customer-name, customer-street, customer-city>
Loan Schema <loan-number, branch-name, amount>
Borrower Schema <customer-name, loan-number>
Account Scheme <account-number, branch-name, balance>
Depositor Scheme <customer-name, account-number>

BRANCH TABLE

Branch Name	Branch City	Assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	800000

CUSTOMER TABLE

Customer Name	Customer Street	Customer City
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto

Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

BORROWER TABLE

Customer Name	Loan Number
Adams	l-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

ACCOUNT TABLE

Account Number	Branch Name	Balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

7a. To find all customers having a loan, an account or both at the bank, without duplicates.
7b. To find all customers having a loan, an account or both at the bank, with duplicates.
7c. To find all customers having both a loan and an account at the bank, without duplicates.
7d. To find all customers having a loan, an account or both at the bank, with duplicates.
7e. To find all customers who have an account but no loan at the bank, without duplicates.
7f. To find all customers who have an account but no loan at the bank, with duplicates.
7g. Find the number of depositors for each branch where average account balance is more than Rs 1200.
7h. Find all customers who have both an account and a loan at the Perryridge branch.
7i. Find the names of all branches that have assets greater than that of each branch located in Brooklyn.
7j. Find all customers who have an account at all the branches located in Brooklyn.

7k. Find all customers who have at most one account at the Perryridge branch.
7l. Find all customers who have at least two accounts at the Perryridge branch.
7m. Find the all customers who have an account but no loan at the bank.
7n. Find the all customers who have either an account or a loan (but not both) at the bank.

Solution to Lab Assignment 7:

```
7a. (SELECT customer_name FROM Depositor) UNION (SELECT customer_name FROM Borrower);

7b. (SELECT customer_name FROM Depositor) UNION ALL (SELECT customer_name FROM Borrower);

7c. (SELECT customer_name FROM Depositor) INTERSECT (SELECT customer_name FROM Borrower);

7d. (SELECT customer_name FROM Depositor) INTERSECT ALL (SELECT customer_name FROM
    Borrower);

7e. (SELECT DISTINCT customer_name FROM Depositor) EXCEPT (SELECT customer_name FROM
    Borrower);

7f. (SELECT DISTINCT customer_name FROM Depositor) EXCEPT ALL (SELECT customer_name
    FROM Borrower);

7g. SELECT branch_name, COUNT(DISTINCT customer_name)
    FROM Depositor D, Account A
    WHERE D.account_number = A.account_number
    GROUP BY branch_name
    HAVING AVG(balance) > 1200;

7h. SELECT DISTINCT B.customer_name FROM Borrower B, Loan L WHERE B.loan_number
    L.loan_number AND branch_name = ‘Perryridge’ AND (branch_name, customer_name) IN (SELECT
    branch_name, customer_name FROM Depositor D, Account A WHERE D.account_number =
    A.account_number);

    or

    SELECT customer_name FROM Borrower B WHERE EXISTS (SELECT * FROM Depositor D WHERE
    D.customer_name = B.customer_name);

7i. SELECT branch_name FROM Account GROUP BY branch_name HAVING AVG(balance) >= ALL
    (SELECT AVG(balance) FROM Account GROUP BY branch_name);

7j. SELECT DISTINCT S.customer_name FROM Depositor AS D WHERE NOT EXISTS ((SELECT
    branch_name FROM Branch WHERE branch_city = ‘Brroklyn) EXCEPT (SELECT R.branch_name
```

FROM Depositor AS T, Account AS R WHERE T.account_number = R.account_number AND D.customer_name = t.customer_name));

7k. SELECT T.customer_name FROM Depositor AS T WHERE UNIQUE (SELECT R.customer_name FROM Depositor AS R, Account AS A WHERE T.customer_name = R.customer_name AND R.account_number = A.account_number AND A.branch_name = 'Perryridge');

7l. SELECT DISTINCT T.customer_name FROM Depositor AS T WHERE NOT UNIQUE (SELECT R.customer_name FROM Depositor AS R, Account AS A WHERE T.customer_name = R.customer_name AND R.account_number = A.account_number AND A.branch_name = 'Perryridge');

7m. SELECT d-CN FROM (Depositor LEFT OUTER JOIN Borrower ON Depositor.customer_name = Borrower.customer_name) AS db1(d-CN, account_number, b-CN, loan_number) WHERE b-CN is null;

7n. SELECT customer_name FROM (Depositor NATURAL FULL OUTER JOIN Borrower) WHERE account_number IS NULL OR loan_number IS NULL;

B. P. Poddar Institute of Management and Technology
Department of Computer Science and Engineering
Academic Year : 2017-18 [Even Semester]

Lab Assignment 8

Topic: DDL DCL TCL Commands for DBA

Readings:

DDL Commands

- CREATE USER: The DBA creates user by executing CREATE USER statement. The user is someone who connects to the database if enough privilege is granted.

```
CREATE USER < username>     --   (name of user to be created )
                    IDENTIFIED BY  <password>  --   (specifies that the user must
                    login with this password)
```

Eg: create user James identified by bob; (The user does not have privilege at this time, it has to be granted.These privileges determine what user can do at database level.)

- CHANGE PASSWORD: The DBA creates an account and initializes a password for every user.You can change password by using ALTER USER statement.

```
Alter USER <some user name>       IDENTIFIED BY<New password>
```

```
Eg:       ALTER USER James       IDENTIFIED BY sam
```

DCL Commands

- PRIVILEGES: A privilege is a right to execute an SQL statement or to access another user's object. In Oracle, there are two types of privileges :
  - System Privileges : are those through which the user can manage the performance of database actions. It is normally granted by DBA to users. Eg: Create Session,Create Table,Create user etc..
  - Object Privileges : allow access to objects or privileges on object, i.e. tables, table columns. tables,views etc..It includes alter,delete,insert,select update etc. (After creating the user, DBA grant specific system privileges to user)

- GRANT COMMAND

```
Grant < database_priv [database_priv…..] > to <user_name> identified by <password> [,<password…..];
```

Eg: Grant create session, create table, create view to James;

Grant <object_priv> | All on <object> to <user | public> [With Grant Option];

Eg: GRANT select, insert ON emp TO James; GRANT select ,update (e_name,e_address) ON emp TO James;

- REVOKE COMMAND

Revoke <database_priv> from <user [, user] >;
Revoke <object_priv> on <object> from < user | public >;

Eg: REVOKE create session,create table from James;
REVOKE select ,insert ON emp FROM James

<database_priv> -- Specifies the system level privileges to be granted to the users or roles. This includes create / alter / delete any object of the system. <object_priv> -- Specifies the actions such as alter / delete / insert / references / execute / select / update for tables.

<all> -- Indicates all the privileges.

[With Grant Option] – Allows the recipient user to give further grants on the objects. The privileges can be granted to different users by specifying their names or to all users by using the "Public" option.

- ROLE: A role is a named group of related privileges that can be granted to user. In other words, role is a predefined collection of privileges that are grouped together, thus privileges are easier to assign user.

Eg: Create role custom;
Grant create table, create view TO custom;
Grant select, insert ON emp TO custom;

Grant custom to James, Steve;

TCL COMMANDS:

- SAVEPOINT: SAVEPOINT <SAVE POINT NAME>;
- ROLLBACK: ROLL BACK <SAVE POINT NAME>;
- COMMIT: Commit;

Problem Statement:

Note : Tables created previously in lab assignments may be used if required

Consider the following tables namely “DEPARTMENTS” & “EMPLOYEES”

Their schemas are as follows -
Departments (dept _no , dept_ name , dept_location);
Employees (emp_id , emp_name , emp_salary);

8a. Develop a query to grant all privileges of employees table into departments table
8b. Develop a query to grant some privileges of employees table into departments table
8c. Develop a query to revoke all privileges of employees table from departments table
8d. Develop a query to revoke some privileges of employees table from departments table
8e. Write a query to implement the save point
8f. Write a query to implement the rollback
8g. Write a query to implement the commit

Solution to Lab Assignment 8:

8a.

```
Grant all on employees to departments;
Grant succeeded.
```

8b.

```
Grant select, update , insert on departments to departments with grant option; Grant succeeded.
```

8c.

```
Revoke all on employees from departments;
Revoke succeeded.
```

8d.

```
Revoke select, update , insert on departments from departments;
Revoke succeeded.
```

8e.

```
SAVEPOINT S1;
Savepoint created.

select * from emp;
```

```
EMPNO    ENAME      JOB     DEPTNO        SAL
---------- ------------------- ------------- ---------- ----------
1          Mathi      AP      1             10000
2          Arjun      ASP     2             15000
3          Gugan      ASP     1             15000
4          Karthik    Prof    2             30000
```

```
INSERT INTO EMP VALUES(5,'Akalya','AP',1,10000);
1 row created.

select * from emp;
```

```
EMPNO    ENAME      JOB     DEPTNO        SAL
---------- ------------------- ------------- ---------- ----------
1          Mathi      AP      1             10000
2          Arjun      ASP     2             15000
3          Gugan      ASP     1             15000
4          Karthik    Prof    2             30000
5          Akalya     AP      1             10000
```

8f.

```
rollback s1;
select * from emp;
```

```
EMPNO ENAME          JOB DEPTNO  SAL
---------- -------------------- --- ---------------------
1        Mathi            AP      1        10000
2        Arjun            ASP     2        15000
3        Gugan            ASP     1        15000
4        Karthik          Prof    2        30000
```

8g.

```
COMMIT;
Commit complete.
```

B. P. Poddar Institute of Management and Technology
Department of Computer Science and Engineering
Academic Year : 2017-18 [Even Semester]

Lab Assignment 9

Topic : PL/Sql Basic

Readings:

Basic Structure of PL/SQL

PL/SQL stands for Procedural Language/SQL. PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL. The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other. Typically, each block performs a logical action in he program. A block has the following structure:

```
DECLARE

  /* Declarative section: variables, types, and local subprograms. */

  BEGIN

  /* Executable section: procedural and SQL statements go here. */

  /* This is the only section of the block that is required. */

  EXCEPTION

  /* Exception handling section: error handling statements go here. */

  END;
```

Only the executable section is required. The other sections are optional. The only SQL statements allowed in a PL/SQL program are SELECT, INSERT, UPDATE, DELETE and several other data manipulation statements plus some transaction control. However, the SELECT statement has a special form in which a single tuple is placed in variables; more on this later. Data definition statements like CREATE, DROP, or ALTER are not allowed. The executable section also contains constructs such as assignments, branches, loops, procedure calls, and triggers, which are all described below (except triggers). PL/SQL is not case sensitive. C style comments (/* ... */) may be used.

To execute a PL/SQL program, we must follow the program text itself by

- A line with a single dot ("."), and then
- A line with run;

Variables and Types

Information is transmitted between a PL/SQL program and the database through variables. Every variable has a specific type associated with it. That type can be

- One of the types used by SQL for database columns
- A generic type used in PL/SQL such as NUMBER
- Declared to be the same as the type of some database column

The most commonly used generic type is NUMBER. Variables of type NUMBER can hold either an integer or a real number. The most commonly used character string type is VARCHAR(n), where n is the maximum length of the string in bytes. This length is required, and there is no default. For example, we might declare:

```
DECLARE

   price  NUMBER;

   myBeer VARCHAR(20);
```

Note that PL/SQL allows BOOLEAN variables, even though Oracle does not support BOOLEAN as a type for database columns.

Types in PL/SQL can be tricky. In many cases, a PL/SQL variable will be used to manipulate data stored in a existing relation. In this case, it is essential that the variable have the same type as the relation column. If there is any type mismatch, variable assignments and comparisons may not work the way you expect. To be safe, instead of hard coding the type of a variable, you should use the %TYPE operator. For example:

```
DECLARE

   myBeer Beers.name%TYPE;
```

gives PL/SQL variable myBeer whatever type was declared for the name column in relation Beers.

A variable may also have a type that is a record with several fields. The simplest way to declare such a variable is to use %ROWTYPE on a relation name. The result is a record type in which the fields have the same names and types as the attributes of the relation. For instance:

```
DECLARE

   beerTuple Beers%ROWTYPE;
```

makes variable beerTuple be a record with fields name and manufacture, assuming that the relation has the schema Beers(name, manufacture).

The initial value of any variable, regardless of its type, is NULL. We can assign values to variables, using the ":=" operator. The assignment can occur either immediately after the type of the variable is declared, or anywhere in the executable portion of the program. An example:

```
DECLARE

   a NUMBER := 3;

BEGIN

   a := a + 1;

END;
```

```
.
run;
```

This program has no effect when run, because there are no changes to the database.

Simple Programs in PL/SQL

The simplest form of program has some declarations followed by an executable section consisting of one or more of the SQL statements with which we are familiar. The major nuance is that the form of the SELECT statement is different from its SQL form. After the SELECT clause, we must have an INTO clause listing variables, one for each attribute in the SELECT clause, into which the components of the retrieved tuple must be placed.

Notice we said "tuple" rather than "tuples", since the SELECT statement in PL/SQL only works if the result of the query contains a single tuple. The situation is essentially the same as that of the "single-row select" in connection with embedded SQL. If the query returns more than one tuple, you need to use a *cursor*, as described in the next section. Here is an example:

```
CREATE TABLE T1(
   e INTEGER,
   f INTEGER
);

DELETE FROM T1;
INSERT INTO T1 VALUES(1, 3);
INSERT INTO T1 VALUES(2, 4);

/* Above is plain SQL; below is the PL/SQL program. */

DECLARE
   a NUMBER;
   b NUMBER;
BEGIN
```

```
   SELECT e,f INTO a,b FROM T1 WHERE e>1;

   INSERT INTO T1 VALUES(b,a);

END;

.

run;
```

Fortuitously, there is only one tuple of T1 that has first component greater than 1, namely (2,4). The INSERT statement thus inserts (4,2) into T1.

Control Flow in PL/SQL

PL/SQL allows you to branch and create loops in a fairly familiar way.

An IF statement looks like:

```
IF <condition> THEN <statement_list> ELSE <statement_list> END IF;
```

The ELSE part is optional. If you want a multiway branch, use:

```
IF <condition_1> THEN ...

ELSIF <condition_2> THEN ...

... ...

ELSIF <condition_n> THEN ...

ELSE ...

END IF;
```

The following is an example, slightly modified from the previous one, where now we only do the insertion if the second component is 1. If not, we first add 10 to each component and then insert:

```
DECLARE

   a NUMBER;

   b NUMBER;

BEGIN

   SELECT e,f INTO a,b FROM T1 WHERE e>1;

   IF b=1 THEN

      INSERT INTO T1 VALUES(b,a);
   ELSE

      INSERT INTO T1 VALUES(b+10,a+10);

   END IF;

END;

.

run;
```

Loops are created with the following:

```
LOOP

   <loop_body> /* A list of statements. */

END LOOP;
```

At least one of the statements in <loop_body> should be an EXIT statement of the form
EXIT WHEN <condition>;
The loop breaks if <condition> is true. For example, here is a way to insert each of the pairs (1, 1) through (100, 100) into T1 of the above two examples:

```
DECLARE

   i NUMBER := 1;

BEGIN

   LOOP

      INSERT INTO T1 VALUES(i,i);

      i := i+1;

      EXIT WHEN i>100;

   END LOOP;

END;

.

run;
```

Some other useful loop-forming statements are:

- EXIT by itself is an unconditional loop break. Use it inside a conditional if you like.
- A WHILE loop can be formed with

  ```
  WHILE <condition> LOOP

     <loop_body>

  END LOOP;
  ```
- A simple FOR loop can be formed with:

  ```
  FOR <var> IN <start>..<finish> LOOP

     <loop_body>

  END LOOP;
  ```

  Here, <var> can be any variable; it is local to the for-loop and need not be declared.
  Also, <start> and <finish> are constants.

Cursors

A cursor is a variable that runs through the tuples of some relation. This relation can be a stored table, or it can be the answer to some query. By fetching into the cursor each tuple of the relation, we can write a program to read and process the value of each such tuple. If the relation is stored, we can also update or delete the tuple at the current cursor position.

The example below illustrates a cursor loop. It uses our example relation T1(e,f) whose tuples are pairs of integers. The program will delete every tuple whose first component is less than the second, and insert the reverse tuple into T1.

```
1) DECLARE

      /* Output variables to hold the result of the query: */

2)    a T1.e%TYPE;

3)    b T1.f%TYPE;

      /* Cursor declaration: */

4)    CURSOR T1Cursor IS

5)       SELECT e, f

6)       FROM T1

7)       WHERE e < f

8)       FOR UPDATE;

9) BEGIN
```

```
10)    OPEN T1Cursor;

11)    LOOP

         /* Retrieve each row of the result of the above query

            into PL/SQL variables: */

12)       FETCH T1Cursor INTO a, b;

         /* If there are no more rows to fetch, exit the loop: */

13)       EXIT WHEN T1Cursor%NOTFOUND;

         /* Delete the current tuple: */

14)       DELETE FROM T1 WHERE CURRENT OF T1Cursor;

         /* Insert the reverse tuple: */

15)       INSERT INTO T1 VALUES(b, a);

16)    END LOOP;

       /* Free cursor used by the query. */

17)    CLOSE T1Cursor;

18) END;

19) .

20) run;
```

Here are explanations for the various lines of this program:

- Line (1) introduces the declaration section.
- Lines (2) and (3) declare variables a and b to have types equal to the types of attributes e and f of the relation T1. Although we know these types are INTEGER, we wisely make sure that whatever types they may have are copied to the PL/SQL variables (compare with the previous example, where we were less careful and declared the corresponding variables to be of type NUMBER).
- Lines (4) through (8) define the cursor T1Cursor. It ranges over a relation defined by the SELECT-FROM-WHERE query. That query selects those tuples of T1 whose first component is less than the second component. Line (8) declares the cursor FOR UPDATE since we will modify T1 using this cursor later on Line (14). In general, FOR UPDATE is unnecessary if the cursor will not be used for modification.
- Line (9) begins the executable section of the program.
- Line (10) opens the cursor, an essential step.
- Lines (11) through (16) are a PL/SQL loop. Notice that such a loop is bracketed by LOOP and END LOOP. Within the loop we find:

    - On Line (12), a fetch through the cursor into the local variables. In general, the FETCH statement must provide variables for each component of the tuple retrieved. Since the query of Lines (5) through (7) produces pairs, we have correctly provided two variables, and we know they are of the correct type.
    - On Line (13), a test for the loop-breaking condition. Its meaning should be clear: %NOTFOUND after the name of a cursor is true exactly when a fetch through that cursor has failed to find any more tuples.
    - On Line (14), a SQL DELETE statement that deletes the current tuple using the special WHERE condition CURRENT OF T1Cursor.
    - On Line (15), a SQL INSERT statement that inserts the reverse tuple into T1.
- Line (17) closes the cursor.
- Line (18) ends the PL/SQL program.
- Lines (19) and (20) cause the program to execute.

Procedures

PL/SQL procedures behave very much like procedures in other programming language. Here is an example of a PL/SQL procedure addtuple1 that, given an integer i, inserts the tuple (i, 'xxx')into the following example relation:

```
CREATE TABLE T2 (

   a INTEGER,

   b CHAR(10)

);



CREATE PROCEDURE addtuple1(i IN NUMBER) AS

BEGIN

   INSERT INTO T2 VALUES(i, 'xxx');

END addtuple1;

.

run;
```

A procedure is introduced by the keywords CREATE PROCEDURE followed by the procedure name and its parameters. An option is to follow CREATE by OR REPLACE. The advantage of doing so is that should you have already made the definition, you will not get an error. On the other hand, should the previous definition be a different procedure of the same name, you will not be warned, and the old procedure will be lost.

There can be any number of parameters, each followed by a *mode* and a type. The possible modes are IN (read-only), OUT (write-only), and INOUT (read and write). **Note:** Unlike the type specifier in a PL/SQL variable declaration, the type specifier in a parameter declaration must be unconstrained. For example, CHAR(10) and VARCHAR(20) are illegal; CHAR or VARCHAR should be used instead. The actual length of a parameter depends on the corresponding argument that is passed in when the procedure is invoked.

Following the arguments is the keyword AS (IS is a synonym). Then comes the body, which is essentially a PL/SQL block. We have repeated the name of the procedure after the END, but this is optional. However, the DECLARE section should *not* start with the keyword DECLARE. Rather, following AS we have:

```
... AS

<local_var_declarations>

BEGIN

   <procedure_body>

END;

.

run;
```

The run at the end runs the statement that creates the procedure; it does not execute the procedure. To execute the procedure, use another PL/SQL statement, in which the procedure is invoked as an executable statement. For example:

```
BEGIN addtuple1(99); END;

.

run;
```

The following procedure also inserts a tuple into T2, but it takes both components as arguments:

```
CREATE PROCEDURE addtuple2(

   x T2.a%TYPE,

   y T2.b%TYPE)

AS

BEGIN

   INSERT INTO T2(a, b)

   VALUES(x, y);

END addtuple2;

.
```

```
run;
```

Now, to add a tuple (10, 'abc') to T2:

```
BEGIN
   addtuple2(10, 'abc');
END;
.
run;
```

The following illustrates the use of an OUT parameter:

```
CREATE TABLE T3 (
   a INTEGER,
   b INTEGER
);


CREATE PROCEDURE addtuple3(a NUMBER, b OUT NUMBER)
AS
BEGIN
   b := 4;
   INSERT INTO T3 VALUES(a, b);
END;
.
run;


DECLARE
   v NUMBER;
BEGIN
```

```
   addtuple3(10, v);

END;

.

run;
```

Note that assigning values to parameters declared as OUT or INOUT causes the corresponding input arguments to be written. Because of this, the input argument for an OUT or INOUT parameter should be something with an "lvalue", such as a variable like v in the example above. A constant or a literal argument should not be passed in for an OUT/INOUT parameter.

We can also write functions instead of procedures. In a function declaration, we follow the parameter list by RETURN and the type of the return value:
CREATE FUNCTION <func_name>(<param_list>) RETURN <return_type> AS ...
In the body of the function definition, "RETURN <expression>;" exits from the function and returns the value of <expression>.

To find out what procedures and functions you have created, use the following SQL query:

```
select object_type, object_name

from user_objects

where object_type = 'PROCEDURE'

   or object_type = 'FUNCTION';
```

To drop a stored procedure/function:

```
drop procedure <procedure_name>;

drop function <function_name>;
```

Discovering Errors

PL/SQL does not always tell you about compilation errors. Instead, it gives you a cryptic message such as "procedure created with compilation errors". If you don't see what is wrong immediately, try issuing the command show errors procedure <procedure_name>;
Alternatively, you can type, SHO ERR (short for SHOW ERRORS) to see the most recent compilation error.
Note that the location of the error given as part of the error message is not always accurate!

Printing Variables

Sometimes we might want to print the value of a PL/SQL local variable. A ``quick-and-dirty" way is to store it as the sole tuple of some relation and after the PL/SQL statement print the relation with a SELECT statement. A more couth way is to define a bind variable, which is the only kind that may be printed with a print command. Bind variables are the kind that must be prefixed with a colon in PL/SQL statements, such as :new .

The steps are as follows:

1. We declare a bind variable as follows:
   VARIABLE <name> <type>
   where the type can be only one of three things: NUMBER, CHAR, or CHAR(*n*).
2. We may then assign to the variable in a following PL/SQL statement, but we must prefix it with a colon.
3. Finally, we can execute a statement
   PRINT :<name>;
   outside the PL/SQL statement

Here is a trivial example, which prints the value 1.

```
VARIABLE x NUMBER

BEGIN

    :x := 1;

END;
.

run;

PRINT :x;
```

Problem Statement:

9a. Write a PL/SQL code, EX_INVNO.SQL, block for inverting a number using all forms of loops.
9b. Write a PL/SQL code, EX_SUMNO.SQL that prints the sum of ‘n’ natural numbers.
9c. Write a PL/SQL program to print all the prime numbers between 100 and 400
9d. Write a PL/SQL program to print 10 terms of fibonacci series.
9e. Write a PL/SQL program to calculate HCF of two numbers.
9f. Write a PL/SQL code, EX_AREA.SQL, of block to calculate the area of the circle for the values of radius varying from 3 to 7. Store the radius and the corresponding values of calculated area in the table AREA_VALUES.

Solution to Lab Assignment 9:

9a.

```
declare
n number(20):=123;
s number(13):=0;
d number(3):=1;
r number(3):=10;
begin
dbms_output.put_line('the number is :' || n);
while n>0 loop
d:=mod(n,10);
s:=(s*r)+d;
n:=n/r;
end loop;
dbms_output.put_line('inverted values' || s);
end;
/
```

OUTPUT:-

```
the number is:123
inverted value is:321
```

9b.

```
prompt enter number:
```

```
accept number n

declare

isum number(2):=0;

i number;

n number:=&n;

begin

for i in 1..n loop

isum:=isum+i;

end loop;

dbms_output.put_line('sum is ' || isum);

end;

/

OUTPUT:-

enter the number:7

sum is 28
```

9c.

```
declare

    x number:=100;

    flag number:=0;

    no number;

    r number;

 begin
```

```
  while x<400 loop
 flag:=0;
                no:=x-1;
        while no>1 loop
          r:=mod(x,no);
          if r=0 then
                  flag:=1;
            exit;
          end if;
        no:=no-1;
        end loop;
        if flag=0 then
          dbms_output.put_line(x);
        end if;
        x:=x+1;
  end loop;
 end;
/
```

9d.

```
declare
  f1 number(3);
  f2 number(3);

  f3 number(3);

  num number(3);

begin

  f1:=0;

  f2:=1;

  f3:=0;

  num:=1;

  while num<=10

  loop

 dbms_output.put_line(f3);

 f1 :=f2;

 f2:=f3;

 f3:=f1+f2;

 num:=num+1;

  end loop;

end;
```

9e.

```
DECLARE

   -- declare variable num1, num2 and t
   -- and these three variables datatype are integer
   num1 INTEGER;
   num2 INTEGER;
   t    INTEGER;
BEGIN
   num1 := 8;

   num2 := 48;
```

```
    WHILE MOD(num2, num1) != 0 LOOP
      t := MOD(num2, num1);

      num2 := num1;

      num1 := t;
    END LOOP;

    dbms_output.Put_line('GCD of '
                         ||num1
                         ||' and '
                         ||num2
                         ||' is '
                         ||num1);
  END;
```

9f.

```
set serveroutput on

declare

area number(5);

rad number(3);

pi number(4):=3.14;

begin

for rad in 3..7 loop

area:=pi*rad*rad;

dbms_output.put_line('area is' || area);

insert into area_values values(area,rad);

end loop;

end;

/
```

OUTPUT:-

```
area is :27
area is :48
area is :75
area is :108
area is :147
```

```
select * from area_values;
area   rad
____ ____
27     3
48     4
75     5
108    6
147    7
```

Lab Assignment 10

Topic : Procedures and cursors using PL/SQL

Readings:

NOTE: Readings for Previous Assignment are sufficient for this lab assignment

Problem Statement:

10a. Create a PL/SQL program using cursors, to retrieve first tuple from the department relation. (use table dept(dno, dname, loc))

10b. Create a PL/SQL program using cursors, to retrieve each tuple from the department relation. (use table dept(dno, dname, loc))

10c. Create a PL/SQL program using cursors, to display the number, name, salary of the three highest paid employees. (use table emp(empno, ename,sal))

10d. Create a PL/SQL program using cursors, to delete the employees whose salary is more than 3000.

10e. Create a PL/SQL program using cursors, to update the salary of each employee by the avg salary if their salary is less than avg salary.

10f. Create a PL/SQL program using cursors, to insert into a table, NEWEMP, the record of ALL MANAGERS. Also DISPLAY on the screen the NO, NAME, JOIN_DATE. Handle any user defined exceptions.(use table emp(emp_no, emp_name, join_date, desig))

Solution to Lab Assignment 10:

10 a.

```
declare

        vdno    dept.deptno%type;

        vdname          dept.dname%type;

        vloc    dept.loc%type;

        cursor  c1 is select * from dept;

        or // cursor c1 is select * from dept where rowno = 1;

begin

        open c1;

        fetch c1
        into vdno,vdname,vloc;
        dbms_output.put_line('vdno = ' ||vdno|| ' vdname = '||vdname||' vloc = '||vloc);
        close c1;
    end;
    /
```

10 b.

```
    declare
        vdept  dept%rowtype;
        cursor c1 is select * from dept;
    begin
        for vdept in c1 loop
            dbms_output.put_line('vdno = ' ||vdept.deptno|| ' vdname = '||vdept.dname||' vloc = '||vdept.loc);
    end loop;

    end;
    /
```

10 c.

```
    declare
        no emp.empno%type;
        name emp.ename%type;
        salary emp.sal%type;
        cursor c1 is select empno, ename, sal from emp order by sal desc;

    begin
```

```
        open c1;

        loop

                fetch c1 into no,name,salary;

                exit when c1 %notfound;

                exit when c1 %rowcount >3;

                dbms_output.put_line(no||name||salary);

        end loop;

        close c1;

end;

/
```

10 d.

```
declare

        vrec emp%rowtype;

cursor c1 is select * from emp where sal>3000 for update;

begin

        open c1;

        loop

                fetch c1 into vrec;

                exit when c1 %notfound;

                delete from emp where current of c1;

                dbms_output.put_line('Record deleted');

        end loop;

        close c1;

end;

/
```

10 e.

```
declare
        vrec emp%rowtype;
        avgsal number(10,2);
cursor c1 is select * from emp for update;


begin
        select avg(sal) into avgsal from emp;
        for vrec in c1 loop
                if vrec.sal < avgsal then
                        vrec.sal := avgsal;
                        update emp set sal = vrec.sal where current of c1;
                        dbms_output.put_line('Record updated');
                end if;
        end loop;
end;
/
```

10 f.

```
set serveroutput on
declare
        ctr     number(2) := 2;
        dno     number(4);
        dname   varchar2(30);
        ddate   date;
```

```
    cursor cur_mgr is
        select emp_no, emp_name, join_date
        from emp
        where upper(desig) = ‘MGR’;

    no_manager_found      exception;

begin
    open cur_mgr;
    loop
        fetch cur_mgr
        into dno, dname, ddate;

        exit when cur_mgr%notfound;
        ctr := ctr + 1;

        dbms_output.put_line(ctr || ‘Record inserted into NEWEMP’);
        dbms_output.put_line(dno || ‘ ‘ || dname || ‘ ‘ ddate);

        insert into new emp
        values (dno, dname, ddate);

    end loop;

    if cur_mgr%rowcount = 0
    then
```

```
                close cur_mgr;
                raise no_manager_found;
        end if;

        dbms_output.put_line(‘TOTAL number of records’ || ctr);
        close cur_mgr;

        exception
                when no_manager_found then
                        dbms_output.put_line(‘NO RECORS FOUND’);
end;
/
```

B. P. Poddar Institute of Management and Technology
Department of Computer Science and Engineering
Academic Year : 2017-18 [Even Semester]

Lab Assignment 11

Topic : Creation and usage of trigger

Readings:

Basic Trigger Syntax

Below is the syntax for creating a trigger in Oracle (which differs slightly from standard SQL syntax):

```
CREATE [OR REPLACE] TRIGGER <trigger_name>

   {BEFORE|AFTER} {INSERT|DELETE|UPDATE} ON <table_name>

   [REFERENCING [NEW AS <new_row_name>] [OLD AS <old_row_name>]]

   [FOR EACH ROW [WHEN (<trigger_condition>)]]

   <trigger_body>
```

Some important points to note:

- You can create only BEFORE and AFTER triggers for tables. (INSTEAD OF triggers are only available for views; typically they are used to implement view updates.)
- You may specify up to three triggering events using the keyword OR. Furthermore, UPDATE can be optionally followed by the keyword OF and a list of attribute(s) in <table_name>. If present, the OF clause defines the event to be only an update of the attribute(s) listed after OF. Here are some examples:

  ```
  ... INSERT ON R ...

  ... INSERT OR DELETE OR UPDATE ON R ...

  ... UPDATE OF A, B OR INSERT ON R ...
  ```

- If FOR EACH ROW option is specified, the trigger is row-level; otherwise, the trigger is statement-level.
- Only for row-level triggers:
  - The special variables NEW and OLD are available to refer to new and old tuples respectively. **Note:** In the trigger body, NEW and OLD must be preceded by a colon (":"), but in the WHENclause, they do not have a preceding colon! See example below.
  - The REFERENCING clause can be used to assign aliases to the variables NEW and OLD.
  - A trigger restriction can be specified in the WHEN clause, enclosed by parentheses. The trigger restriction is a SQL condition that must be satisfied in order for Oracle to fire the trigger. This condition cannot contain subqueries. Without the WHEN clause, the trigger is fired for each row.

- <trigger_body> is a PL/SQL block, rather than sequence of SQL statements. Oracle has placed certain restrictions on what you can do in <trigger_body>, in order to avoid situations where one trigger performs an action that triggers a second trigger, which then triggers a third, and so on, which could potentially create an infinite loop. The restrictions on <trigger_body>include:
    - You cannot modify the same relation whose modification is the event triggering the trigger.
    - You cannot modify a relation connected to the triggering relation by another constraint such as a foreign-key constraint.

Trigger Example

We illustrate Oracle's syntax for creating a trigger through an example based on the following two tables:

```
CREATE TABLE T4 (a INTEGER, b CHAR(10));

CREATE TABLE T5 (c CHAR(10), d INTEGER);
```

We create a trigger that may insert a tuple into T5 when a tuple is inserted into T4. Specifically, the trigger checks whether the new tuple has a first component 10 or less, and if so inserts the reverse tuple into T5:

```
CREATE TRIGGER trig1
   AFTER INSERT ON T4
   REFERENCING NEW AS newRow
   FOR EACH ROW
   WHEN (newRow.a <= 10)
   BEGIN
      INSERT INTO T5 VALUES(:newRow.b, :newRow.a);
   END trig1;
.
run;
```

Notice that we end the CREATE TRIGGER statement with a dot and run, as for all PL/SQL statements in general. Running the CREATE TRIGGER statement only creates the trigger; it does not execute the trigger. Only a triggering event, such as an insertion into T4 in this example, causes the trigger to execute.

Displaying Trigger Definition Errors

As for PL/SQL procedures, if you get a message

```
Warning: Trigger created with compilation errors.
```

you can see the error messages by typing

```
show errors trigger <trigger_name>;
```

Alternatively, you can type, SHO ERR (short for SHOW ERRORS) to see the most recent compilation error. Note that the reported line numbers where the errors occur are not accurate.

Viewing Defined Triggers

To view a list of all defined triggers, use:

```
select trigger_name from user_triggers;
```

For more details on a particular trigger:

```
select trigger_type, triggering_event, table_name, referencing_names, trigger_body
from user_triggers
where trigger_name = '<trigger_name>';
```

Dropping Triggers

To drop a trigger:

```
drop trigger <trigger_name>;
```

Disabling Triggers

To disable or enable a trigger:

```
alter trigger <trigger_name> {disable|enable};
```

Aborting Triggers with Error

Triggers can often be used to enforce contraints. The WHEN clause or body of the trigger can check for the violation of certain conditions and signal an error accordingly using the Oracle built-in function RAISE_APPLICATION_ERROR. The action that activated the trigger (insert, update, or delete) would be aborted. For example, the following trigger enforces the constraint Person.age >= 0:

```
create table Person (age int);

CREATE TRIGGER PersonCheckAge
AFTER INSERT OR UPDATE OF age ON Person
FOR EACH ROW
BEGIN
   IF (:new.age < 0) THEN
      RAISE_APPLICATION_ERROR(-20000, 'no negative age allowed');
   END IF;
END;
```

```
.
RUN;
```

If we attempted to execute the insertion:

```
insert into Person values (-3);
```

we would get the error message:

```
ERROR at line 1:
ORA-20000: no negative age allowed
ORA-06512: at "MYNAME.PERSONCHECKAGE", line 3
ORA-04088: error during execution of trigger 'MYNAME.PERSONCHECKAGE'
```

and nothing would be inserted. In general, the effects of both the trigger and the triggering statement are rolled back.

Mutating Table Errors

Sometimes you may find that Oracle reports a "mutating table error" when your trigger executes. This happens when the trigger is querying or modifying a "mutating table", which is either the table whose modification activated the trigger, or a table that might need to be updated because of a foreign key constraint with a CASCADE policy. To avoid mutating table errors:

- A row-level trigger must not query or modify a mutating table. (Of course, NEW and OLD still can be accessed by the trigger.)
- A statement-level trigger must not query or modify a mutating table if the trigger is fired as the result of a CASCADE delete.

Problem Statement:

Note : Tables created previously in lab assignments may be used if required

Considering -

Empa Schema<id number, name, dname, age, income, expence, savings>

Emp Schema<institute name, employee id, salary>

Sal <institute name, total employee, total salary>

11a. For every insert or delete or update in Empa table create trigger to display the message TABLE IS INSERTED or TABLE IS DELETED or TABLE IS UPDATED

11b. Define trigger to force all department names to uppercase.

11c. Create a Trigger to check the age valid or not using message after every insert or delete or update in Trig table

11d. Create a Trigger to check the age valid and Raise appropriate error code and error message.

11e. A trigger restricting updates that allows changes to Empa records only on Mondays through Fridays, and only during the hours of 8:00am to 5:00pm.

11f. Create a Trigger for Emp table it will update another table Sal while inserting values.

Solution to Lab Assignment 11:

11a.

```
create table empa(id number(3),name varchar2(10),income number(4),expence number(3),savings
number(3));

Table created.

insert into empa values(2,'kumar',2500,150,650); 1 row created.

insert into empa values(3,'venky',5000,900,950); 1 row created.

insert into empa values(4,'anish',9999,999,999); 1 row created.

select * from empa;

     ID        NAME          INCOME          EXPENCE         SAVINGS
   -------     --------      ------------    --------------  ------------
     2         kumar          2500           150                650
     3         venky         5000             900                950
     4         anish        9999             999                 999

CREATE OR REPLACE TRIGGER VIJAY AFTER UPDATE OR INSERT OR
DELETE ON EMP FOR EACH ROW BEGIN
IF UPDATING THEN DBMS_OUTPUT.PUT_LINE('TABLE IS UPDATED');
ELSIF INSERTING THEN DBMS_OUTPUT.PUT_LINE('TABLE IS INSERTED');
ELSIF DELETING THEN DBMS_OUTPUT.PUT_LINE('TABLE IS DELETED');
END IF;
END;
/
Trigger created.


update emp set income =900 where empname='kumar'; TABLE IS UPDATED

1 row updated.
```

```
insert into emp values ( 4,'Chandru',700,250,80); TABLE IS INSERTED

1 row created.

DELETE FROM EMP WHERE EMPID = 4; TABLE IS DELETED

1 row deleted.

select * from emp;
```

EMPID	EMPNAME	INCOME	EXPENSE	SAVINGS
---------	---------	---------	---------	---------
2	vivek	830	150	100
3	kumar	5000	550	50
9	vasanth	987	6554	644

11b.

```
create trigger t2 before
insert
on Empa
for each row
declare
s1 varchar2(20);
begin
s1:=:new.name;
:new.name:=UPPER(s1);
end;

insert into Empa values(12,'sayan','CSE',21,10000,900,1800);

select * from Empa;
```

11c.

```
create trigger t3 before
insert
on Empa
for each row
declare
agenew number;
begin
agenew:=:new.age;
if ( agenew > 15 ) then
dbms_output.put_line('Valid');
```

```
else
dbms_output.put_line('Invalid');
end if;
end;


insert into Empa values(90,'saurav','CSE',30,10000,900,700);
```

11d.

```
create table data(name char(10),age number(3));

Table created.

desc data;

  Name                                     Null?  Type
----------------------------------------- -------- ------------------------

 NAME                                               CHAR(10)

 AGE                                               NUMBER(3)


        CREATE TRIGGER DATACHECK

        AFTER INSERT OR UPDATE OF AGE ON DATA FOR
        EACH ROW

        BEGIN IF(:NEW.AGE<0) THEN

        RAISE_APPLICATION_ERROR(-20000,'NO NEGATIVE AGE ALLOWED'); END IF;

        END;

        /

        Trigger created.


 INSERT INTO DATA VALUES('ABC',10);

 1 ROW CREATED.
```

```
INSERT INTO DATA VALUES ('DEF',-15)

*  ERROR at line 1:

ORA-20000: No negative age allowed

ORA-06512: at "4039.DATACHECK", line 3

ORA-04088: error during execution of trigger '4039.DATACHECK'


NAME            AGE

----------    ----------

abc              10
```

11e.

```
CREATE TABLE SRM_EMP2(INAME VARCHAR2(10),

                              IID NUMBER(5), SALARY NUMBER(10));

Table created.

CREATE TABLE SRM_SAL2(INAME VARCHAR2(10),

                              TOTALEMP NUMBER(5),
                              TOTALSAL NUMBER(10));

Table created.

CREATE OR REPLACE TRIGGER EMPTRIGR22 AFTER INSERT ON SRM_EMP2 FOR EACH
ROW

 DECLARE

 A VARCHAR2(10); BEGIN
 A:=:NEW.INAME;

 UPDATE SRM_SAL2 SET
 TOTALSAL=TOTALSAL+:NEW.SALARY,TOTALEMP=TOTALEMP+1 WHERE INAME=A;

END;
```

```
/

Trigger created.

INSERT INTO SRM_SAL2 VALUES('VEC',0,0);

1 row created.

INSERT INTO SRM_SAL2 VALUES('SRM',0,0);

1 row created.

INSERT INTO SRM_EMP2 VALUES('VEC',100,1000);

1 row created.

SELECT * FROM SRM_SAL2;

INAME     TOTALEMP   TOTALSAL

----------   ----------   ----------
VEC              1         1000
SRM              0            0


INSERT INTO SRM_EMP2 VALUES('SRM',200,3000);

1 row created.


SELECT * FROM SRM_SAL2;


INAME      TOTALEMP   TOTALSAL

----------    ----------   ----------
VEC             1         1000
SRM             1         3000


INSERT INTO SRM_EMP2 VALUES('VEC',100,5000);

1 row created.


SELECT * FROM SRM_SAL2;


INAME       TOTALEMP     TOTALSAL
```

```
----------          ----------         ----------
VEC                     2                6000
SRM                     1                 3000


INSERT INTO SRM_EMP2 VALUES('VEC',100,2000);

1 row created.


SELECT * FROM SRM_SAL2;


INAME        TOTALEMP   TOTALSAL

----------         ----------          ----------
VEC                     3                  8000
SRM                     1                   3000


INSERT INTO SRM_EMP2 VALUES('SRM',200,8000);

1 row created.

SELECT * FROM SRM_SAL2;


INAMETOTAL       EMP         TOTALSAL

----------               ----------          ----------
VEC                          3              8000
SRM                          2             11000
```

11f.

```
CREATE OR REPLACE TRIGGER only_during_business_hours
BEFORE INSERT OR UPDATE OR DELETE ON employee
BEGIN
IF  TO_NUMBER(TO_CHAR(SYSDATE,'hh24')) < 8
OR TO_NUMBER(TO_CHAR(SYSDATE,'hh24')) >= 5
OR TO_CHAR(SYSDATE,'dy') in ('sun','sat') THEN
RAISE_APPLICATION_ERROR (-20000, 'Employee changes only allowed during business hours.');
END IF;
END;
/
```

Questionnaire for Lab-report

Assignment 1 : Identify the advantage of creating a table in DBMS over storing the data using structure and file in C/C++.

Assignment 2 : What are the restriction for modifying a column ?

Assignment 3 : Can we add foreign key after creating and adding data in both the tables. State the conditions.

Assignment 4 : Write the syntax for different type of join operation in oracle.

Assignment 5 : Explain the role of having keyword in using aggregate functions.

Assignment 6 : What is view ? Can we update view? State the conditions.

Assignment 7 : Differentiate correlated sub-query and nested sub-query, with example.

Assignment 8 : Explain Grant, Revoke, Commit, Rollback and Savepoint.

Assignment 9 : Differentiate PL/SQL and SQL?

Assignment 10: What is the difference between FUNCTION, PROCEDURE AND PACKAGE in PL/SQL? What is cursor and why it is required?

Assignment 11: Explain the difference in execution of triggers and stored procedures.

A Quick Reference to SQL Syntax

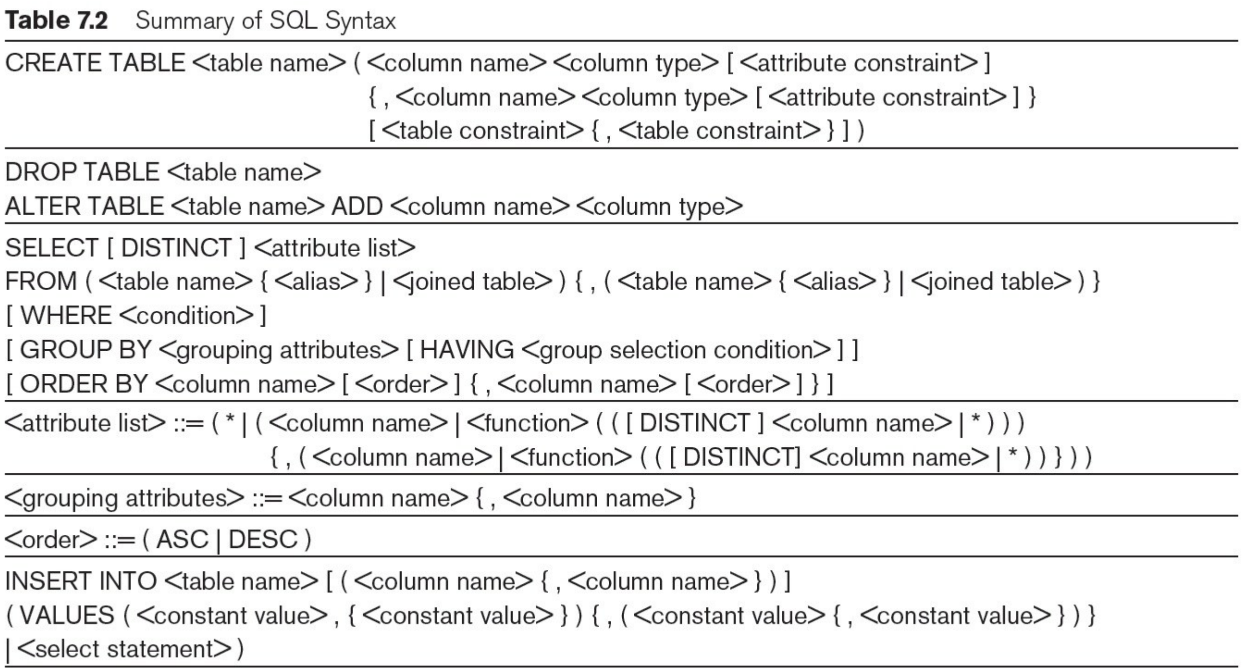


Table 7.2 Summary of SQL Syntax

```
CREATE TABLE <table name> ( <column name> <column type> [ <attribute constraint> ]
                            { , <column name> <column type> [ <attribute constraint> ] }
                            [ <table constraint> { , <table constraint> } ] )
```

```
DROP TABLE <table name>
ALTER TABLE <table name> ADD <column name> <column type>
```

```
SELECT [ DISTINCT ] <attribute list>
FROM ( <table name> { <alias> } | <joined table> ) { , ( <table name> { <alias> } | <joined table> ) }
[ WHERE <condition> ]
[ GROUP BY <grouping attributes> [ HAVING <group selection condition> ] ]
[ ORDER BY <column name> [ <order> ] { , <column name> [ <order> ] } ]
```

```
<attribute list> ::= ( * | ( <column name> | <function> ( ( [ DISTINCT ] <column name> | * ) ) )
                     { , ( <column name> | <function> ( ( [ DISTINCT] <column name> | * ) ) } ) )
```

```
<grouping attributes> ::= <column name> { , <column name> }
```

```
<order> ::= ( ASC | DESC )
```

```
INSERT INTO <table name> [ ( <column name> { , <column name> } ) ]
( VALUES ( <constant value> , { <constant value> } ) { , ( <constant value> { , <constant value> } ) }
| <select statement> )
```

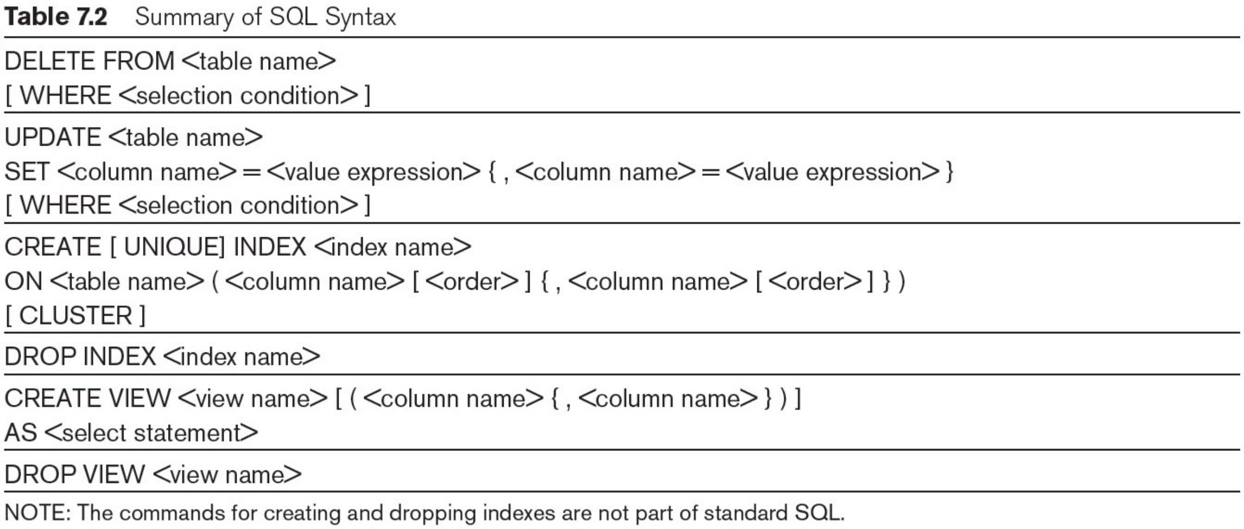


Table 7.2 Summary of SQL Syntax

```
DELETE FROM <table name>
[ WHERE <selection condition> ]
```

```
UPDATE <table name>
SET <column name> = <value expression> { , <column name> = <value expression> }
[ WHERE <selection condition> ]
```

```
CREATE [ UNIQUE] INDEX <index name>
ON <table name> ( <column name> [ <order> ] { , <column name> [ <order> ] } )
[ CLUSTER ]
```

```
DROP INDEX <index name>
```

```
CREATE VIEW <view name> [ ( <column name> { , <column name> } ) ]
AS <select statement>
```

```
DROP VIEW <view name>
```

NOTE: The commands for creating and dropping indexes are not part of standard SQL.