A Laboratory Manual for

Database Management System

(2000454(022))

Semester– IV

Department of
Computer Science & Engineering

Government Polytechnic Jashpur

Prepared by
Manish Dongre
Lecturer, Computer Science
Government Polytechnic Jashpur

Certificate

This is to certify that Mr. / Ms. ………………………………….
Roll No………………………. of Fourth Semester of Diploma in
…………………………………………………………. of Institute
…………………………………………………………………......
(Code………….) has completed the term work satisfactorily
insubject **Database Management System (2000454(022))** for the
academic year 20…….to 20…...... as prescribed in the curriculum.

Place ……………….   Enrollment No……………………
Date: ....   Roll No. ………………………........

Subject Teacher   Head of the Department   Principal

INDEX

PREFACE

Welcome to the Database Management Systems (DBMS) Lab Manual, designed specifically for diploma students. This manual offers a hands-on approach to learning key DBMS concepts and skills, enhancing your understanding through practical exercises.

You will explore topics such as SQL, database design, normalization, and transaction management. Each lab session is structured to build progressively, helping you develop a solid foundation in database management.

This manual aims to prepare you for both academic success and real-world applications in the field of database management. We hope you find it informative and engaging.

Happy Learning!

COMPUTER SCIENCE AND ENGINEERING

DBMS LAB MANUAL

INSTITUTION VISION AND MISSION

Vision:

Create value based skilled professionals that will demonstrate their competence by integrating acquired knowledge and skills.

Mission:

- Impart value based quality education and skills.
- Adopt best pedagogy and practices for knowledge transfer and skill development among the students.
- Develop a conducive environment for education by providing good infrastructure, facility and amenities for improving the employability.
- Encourage the students to explore, build, and learn to enable them to apply their theoretical knowledge in real life situations
- Foster a culture of lifelong learning in the students to meet the changing needs of profession and society.

Core Values:

Professionalism, Honesty, Integrity, Commitment and Team Work.

COMPUTER SCIENCE AND ENGINEERING

DBMS LAB MANUAL

DEPARTMENT VISION AND MISSION

Vision:

The Diploma in Information Technology programme envisages to develop competent/ skilled technical manpower in the area of Information and communication technologies to meet the emerging challenging needs of the modern computing industrial/business society.

Mission:

The Mission of the Diploma in Information Technology Programme is to :

- Develop competent technical manpower of the Chhattisgarh region through quality education in Information technology.
- Provide demand driven quality education in the field of Information technology.
- Provide an atmosphere for students for continuous learning to investigate, apply and transfer knowledge.
- Develop communication skills, ethical values, environment awareness and analytical skills among students.

Programme Outcomes (POs) to be achieved through Practical’s of thisCourse

Following programme outcomes are expected to be achieved significantly out of the ten programme outcomes and Computer Engineering programme specific outcomes through the practical’s of the course on **Programming with Python.**

PO1	**Basic knowledge:** Apply knowledge of basic mathematics, sciences and basic engineering to solve the broad-based Computer engineering problem.
PO2	**Discipline knowledge:** Apply Computer engineering discipline-specific knowledge to solve core computer engineering related problems.
PO3	**Experiments and practice:** Plan to perform experiments and practices to use the results to solve broad-based Computer engineering problems.
PO4	**Engineering tools:** Apply relevant Computer technologies and tools with an understanding of the limitations.
PO5	**The engineer and society:** Assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to practice in field of Computer engineering.
PO6	**Environment and sustainability:** Apply Computer engineering solutions also for sustainable development practices in societal and environmental contexts and demonstrate the knowledge and need for sustainable development.
PO7	**Ethics:** Apply ethical principles for commitment to professional ethics, responsibilities and norms of the practice also in the field of Computer engineering.
PO8	**Individual and team work:** Function effectively as a leader and team member in diverse/ multidisciplinary teams.
PO9	**Communication:** Communicate effectively in oral and written form.
PO10	**Life-long learning:** Engage in independent and life-long learning activities in the context of technological changes in the Computer engineering field and allied industry.

Practical - Course Outcome matrix

Course Outcomes (COs)

a) **Describe basic concepts of database system.**
b) **Design a data model and schemas in RDBMS.**
c) **Use of Structured Query Language (SQL).**
d) **Use of Group functions in Structured Query Language (SQL).**
e) **Apply functional dependencies for designing a robust database.**

Sr. No.	Title of the Practical	CO a	CO b	CO c	CO d	CO e
1	TO STUDY DDL (Data-Definition Language) COMMANDS					
2	TO STUDY DML COMMANDS					
3	TO IMPLEMENT INTEGRITY CONSTRAINTS ON DATABASE					
4	TO USE ARITHMATIC, LOGICAL, COMPARISION, LOGICAL & DATA OPERATORS					
5	TO USE ARITHMATHIC CHARACTER, NUMERIC, DATA FUNCTION					
6	TO STUDY SET OPERATORS ON DATABASE OBJECTS					
7	TO STUDY CLAUSES AND AGGREGATE FUNCTIONS					
8	TO STUDY JOINS & SUB-QUERIES IN SQL					
9	TO STUDY VIEWS AND TRIGERS IN SQL					
10	STUDY EXPERIMENT (INTRODUCTION TO PL/SQL)					
11	WAP FOR DECLARING & USING VARIABLES CONSTANT USING LOOPS AND DATA STRUCTURE IN PL / SQL					
12	TO STUDY PROCEDURE AND FUNCTIONS IN PL / SQL					

Instructions for Students

Student shall read the points given below for understanding the theoretical concepts and practical applications.

1. Students shall listen carefully the lecture given by teacher about importance of subject, learning structure, course outcomes.
2. Students shall organize the work in the group of two or three members and make a record of all observations.
3. Students shall understand the purpose of experiment and its practical implementation.
4. Students shall write the answers of the questions during practical.
5. Student should feel free to discuss any difficulty faced during the conduct of practical.
6. Students shall develop maintenance skills as expected by the industries.
7. Student shall attempt to develop related hands on skills and gain confidence.
8. Students shall refer technical magazines; websites related to the scope of the subjects and update their knowledge and skills.
9. Students shall develop self-learning techniques.
10. Students should develop habit to submit the write-ups on the scheduled dates and time.

Index

List of Practical's and Progressive Assessment Sheet

9.	TO STUDY VIEWS AND TRIGERS IN SQL						
10	STUDY EXPERIMENT (INTRODUCTION TO PL/SQL)						
11	WAP FOR DECLARING & USING VARIABLES CONSTANT USING LOOPS AND DATA STRUCTURE IN PL / SQL						
12	TO STUDY PROCEDURE AND FUNCTIONS IN PL / SQL						

EXPERIMENT NO. 01

TO STUDY DDL (Data-Definition Language) COMMANDS

AIM: TO STUDY DDL (Data-Definition Language) COMMANDS

THEORY:

Data-Definition Language

We specify a database schema by a set of definitions expressed by a special language called a data-definition language.

The SQL DDL (Data Definition Language) allows specification of not only a set of relations, hut also the following information for each relation:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints.
- The set of indices for each relation.
- Security and authorization information.
- Physical storage structure on disk.

SOME EXAMPLES:

SQL>create table employees (employee_id number(6),first_name varchar2(20),last_name varchar2(25),email_id varchar2(25),phone_number number(20),hire_date date,job_id varchar2(10),salary number(8,2),commission_pct number(2,2),manager_id number(6),department_id number(4));

O/P:- Table Created.

SQL> desc employees;

```
 Name                Null?                 Type
employee_id                              number(6)
first_name                                 varchar2(20)
last_name                                  varchar2(25)
email_id                                    varchar2(25)
```

```
phone_number                                   number(20)
hire_date                                      date
salary                                         number(8,2)
commission_pct                                 number(5)
manager_id                                     number(6)
department_id                                  number(4)
```

SQL> select * from employees

O/P: - no rows selected.
SQL> alter table employees(modify employee_id number(10));
O/P:- Table altered.
SQL> desc employees;

O/P:- Name   Null?   Type

```
4. employee_id            number(10)
first_name                 varchar2(20)

last_name                  varchar2(25)

email_id                  varchar2(25)
phone_number              number(20)
hire_date            date
job_id   varchar2(10)
salary            number(8,2)
commission_pct            number(5)
manager_id             number(6)
department_id             number(4)
```

SQL> alter table employees add (address varchar2(60));
O/P:- Table altered.
5.**SQL>** desc employees;

O/P:- Name   Null?   Type

```
employee_id                                         number(10)
first_name                                          varchar2(20)
```

last_name	varchar2(25)
email_id	varchar2(25)
phone_number	number(20)
hire_date	date
job_id	varchar2(10)
salary	number(8,2)
commission_pct	number(5)
manager_id	number(6)
department_id	number(4)
address	varchar2(60)

6.SQL> rename employees to emp;

O/P:- Table renamed.

<u>CONCLUSION:</u> THUS WE HAVE STUDIED DDL COMMANDS

EXPERIMENT NO. 02

TO STUDY DML COMMANDS

AIM: TO STUDY DML COMMANDS

THEORY:

Data Manipulation Language :- (DML):-

This language enables users to access or manipulate data as

Organized by the appropriate data model.

The types of access are;

Retrieval of information stored in the database.

Insertion of new information into the database.

Deletion of information from the database.

Modification of information stored in the database.

There are basically two types:-

Procedural DMLs require a user to specify what data are needed and how to get those data.

Declarative DMLs (non-procedural DMLs) require a user to specify what data are needed without specifying how to get those data.

The portion of a DML that involves information retrieval is called as query language.

SOME EXAMPLES:

1.

```
SQL> insert into student values
  2  ('&Rollno','&Name','&Address');
     Enter value for rollno: 1

     Enter value for name: akshay

    Enter value for address: asd
old   2: ('&Rollno','&Name','&Address')
new   2: ('1','a','asd')
1 row created.
```

```
        >
        2.SQL> select * from student;
O/P> ROLLNO     NAME        ADDRESS
        ----------   -------------------- --------------------
        1                Akshay       asd
        1 rows selected.
        SQL> update student set phone=2435677 where rollno=1;
        1 row updated.
        SQL> select * from student;
        ROLLNO                  NAME            ADDRESS        PHONE
        ------------      -------------------------       -------------------- --------------
        1                       Akshay                    asd             2435677
        1 rows selected.
        3. SQL> select * from student;
        ROLLNO                  NAME             ADDRESS        PHONE
        ----------                -------------------- -------------------- --------------
        1   Akshay                                  asd           2435677
        2   Amit                                    xyz           2376589
         SQL> delete from student where rollno=1;
        1 row deleted.
        SQL> select * from student;
         ROLLNO      NAME        ADDRESS     PHONE
        ---------- -------------------- -------------------- --------------
        1                  Amit            xyz            2376589
```

CONCLUSION: THUS WE HAVE STUDIED DML COMMANDS.

EXPERIMENT NO. 03

TO IMPLEMENT INTEGRITY CONSTRAINTS ON DATABASE

AIM: TO IMPLEMENT INTEGRITY CONSTRAINTS ON DATABASE

THEORY:

Integrity Constraints

1. Integrity constraints provide a way of ensuring that changes made to the database by authorized users do not result in a loss of data consistency.
2. An integrity constraint can be any arbitrary predicate applied to the database.

The different Entity integrity constraints are:

Primary key constraint
Unique key constraint
Referential integrity constraint etc.

SOME EXAMPLES:

1 SQL> create table employees (employee_id number(6),first_name varchar2(20) **not null**,last_name varchar2(25),email_id varchar2(25) **unique**,phone_number number(20),hire_date date,job_id varchar2(10),salary number(8,2) **not null**, commission_pct number(2,2),manager_id number(6),department_id number(4), **constraint emp_id_pk primary key (employee_id)**);

O/P:- Table created.

SQL> create table departments(department_id number(5),department_name varchar2(20) not null, manager_id number(10), location_id(10),**constraint dept_id_pk primary key (department_id)**);

O/P:- Table created.

SQL> alter table employee add **constraint dept_id_fk foreign key (department_id) references departments (department_id)**;

O/P:- Table altered

SQL> desc employees;

```
 O/P: - Name              Null?        Type
        ----------------------------------------
         employee_id      not null     number(6)
        first_name        not null     varchar2(20)
        last_name                      varchar2(25)
        email_id          unique       varchar2(25)
        phone_number                   number(20)
        hire_date                      date
        job_id                         varchar2(10)
```

```
    salary              not null     number(8,2)
    commission_pct                   number(5)
    manager_id                       number(6)
    department_id       not null     number(4)
```

SQL> desc departments;

```
O/P:- Name           Null?         Type
 -----------------------------------------------------
department_id        not null      number(5)
department_name      not null      varchar2(20)
manager_id                         number(10)
location_id                        number(10)
```

CONCLUSION: THUS WE HAVE STUDIED INTEGRITY CONSTRAINTS

EXPERIMENT NO. 04

TO USE ARITHMATIC, LOGICAL, COMPARISION, LOGICAL & DATA OPERATORS

AIM: TO USE ARITHMATIC, LOGICAL, COMPARISION, LOGICAL & DATA OPERATORS

THEORY:

Operators are evaluated in the order of precedence as shown below:

Arithmetic Operators.

Exponentiation.	^
Negation.	-
Multiplication and Division.	*, /
Integer Division.	\
Modulo Arithmetic.	Mod
Addition & Subtraction.	+, -

Comparison Operator.

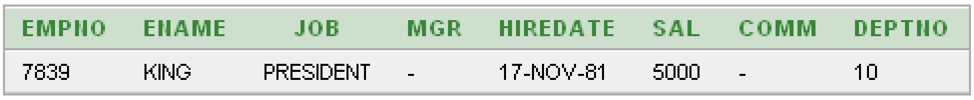
Equality.	=
Inequality.	<>
Less than.	<
Greater than.	>
Less than or equal to.	<=
Greater than or equal to.	>=

Logical operators.

Not.	Not
And.	And
Or.	Or
Xor.	Xor

SOME EXAMPLES:

1 SQL> select * from emp where empno=7839;
O/P: - 1 row selected.

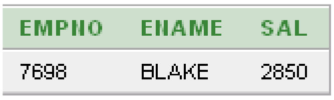


EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10

SQL> select * from emp where empno != 7839;
O/P: - ` 12 rows selected

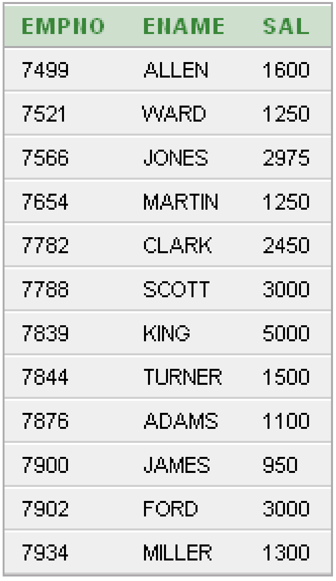
SQL> `select empno, ename, sal from emp where ename like 'BLAKE';

O/P: - 1 row selected



EMPNO	ENAME	SAL
7698	BLAKE	2850

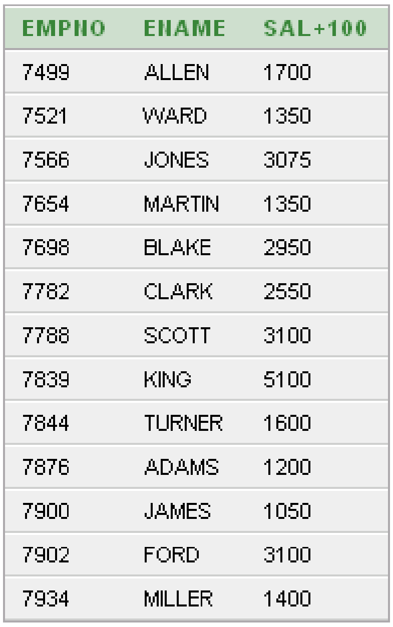
SQL > select empno, ename, sal from EMP where ename not like 'BLAKE';



EMPNO	ENAME	SAL
7499	ALLEN	1600
7521	WARD	1250
7566	JONES	2975
7654	MARTIN	1250
7782	CLARK	2450
7788	SCOTT	3000
7839	KING	5000
7844	TURNER	1500
7876	ADAMS	1100
7900	JAMES	950
7902	FORD	3000
7934	MILLER	1300

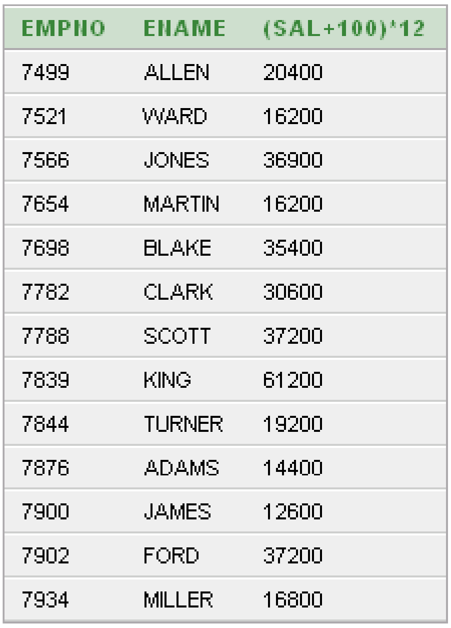
O/P:-
12 rows selected

SQL> select empno, ename, sal+100 from EMP;
O/P: - 13 rows selected



EMPNO	ENAME	SAL+100
7499	ALLEN	1700
7521	WARD	1350
7566	JONES	3075
7654	MARTIN	1350
7698	BLAKE	2950
7782	CLARK	2550
7788	SCOTT	3100
7839	KING	5100
7844	TURNER	1600
7876	ADAMS	1200
7900	JAMES	1050
7902	FORD	3100
7934	MILLER	1400

SQL> select empno, ename, (sal+100)*12 from EMP;

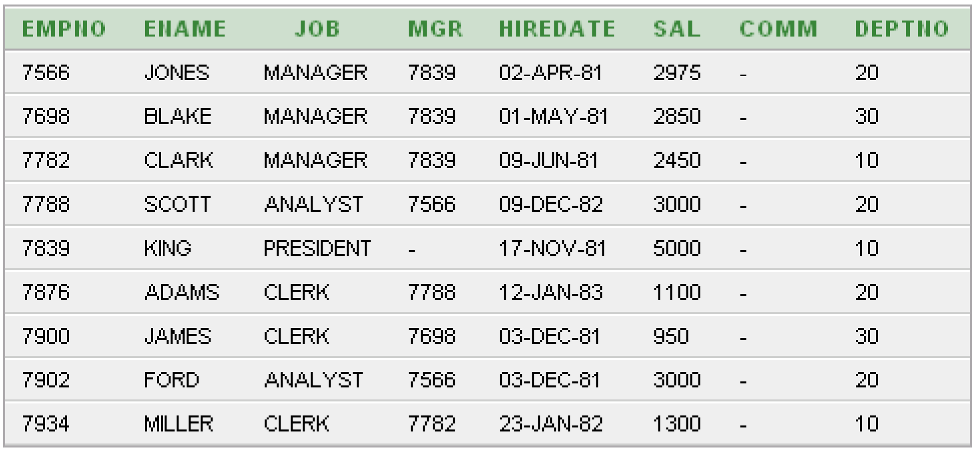


EMPNO	ENAME	(SAL+100)*12
7499	ALLEN	20400
7521	WARD	16200
7566	JONES	36900
7654	MARTIN	16200
7698	BLAKE	35400
7782	CLARK	30600
7788	SCOTT	37200
7839	KING	61200
7844	TURNER	19200
7876	ADAMS	14400
7900	JAMES	12600
7902	FORD	37200
7934	MILLER	16800

O/P:-
13 rows selected

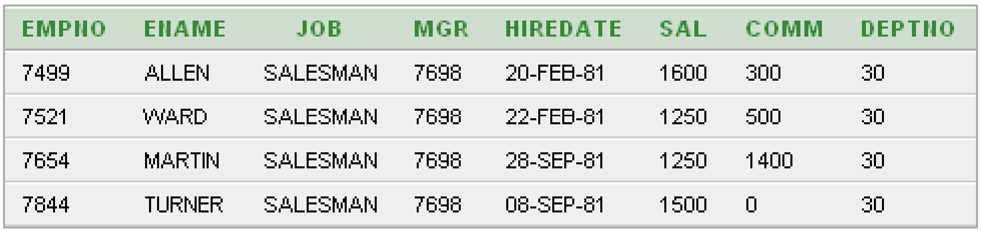
SQL> select empno, ename, (sal+100)*12 from EMP where job like 'MANAGER';
O/P: - 3 rows selected

SQL> select * from EMP where comm is null;
O/P: - 9 rows selected



EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7876	ADAMS	CLERK	7788	12-JAN-83	1100	-	20
7900	JAMES	CLERK	7698	03-DEC-81	950	-	30
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10

SQL> select * from EMP where comm is not null;

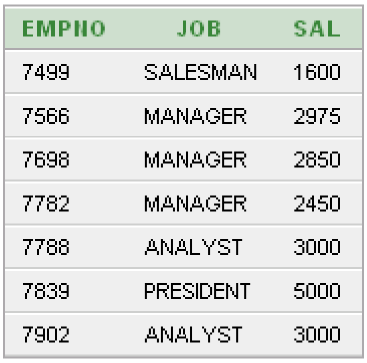


EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30

O/P:-
4rows selected

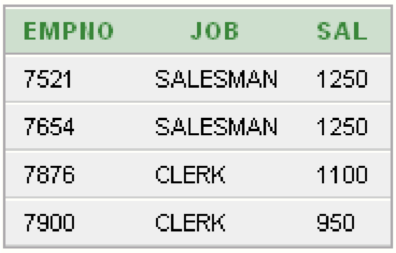
SQL> select empno, job, sal from EMP where sal >1500;

O/P:-
7 rows selected



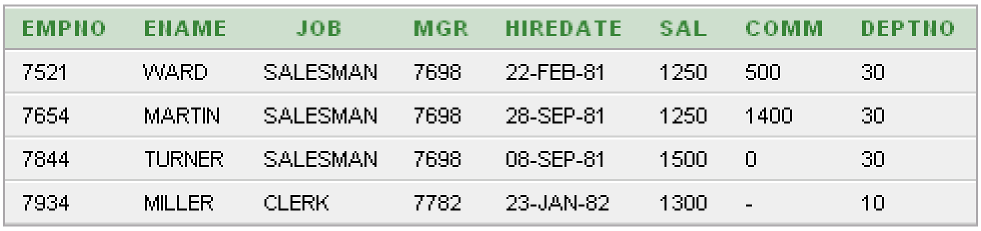
EMPNO	JOB	SAL
7499	SALESMAN	1600
7566	MANAGER	2975
7698	MANAGER	2850
7782	MANAGER	2450
7788	ANALYST	3000
7839	PRESIDENT	5000
7902	ANALYST	3000

SQL> select empno, job, sal from EMP where sal <=1250;
O/P:-
4 rows selected



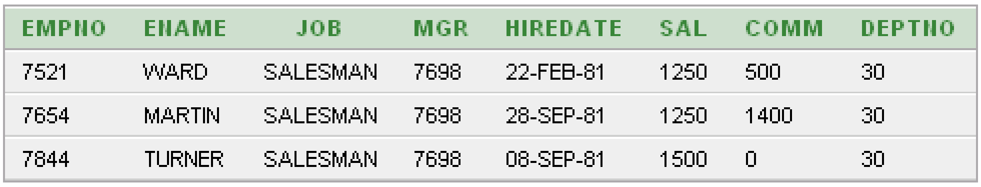
EMPNO	JOB	SAL
7521	SALESMAN	1250
7654	SALESMAN	1250
7876	CLERK	1100
7900	CLERK	950

SQL> select * from EMP where sal between 1250 and 1500;
O/P:-
4 rows selected



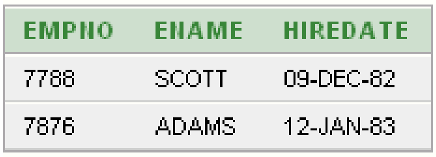
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10

SQL> select * from EMP where sal in (1250, 1500);
O/P:-
3 rows selected



EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30

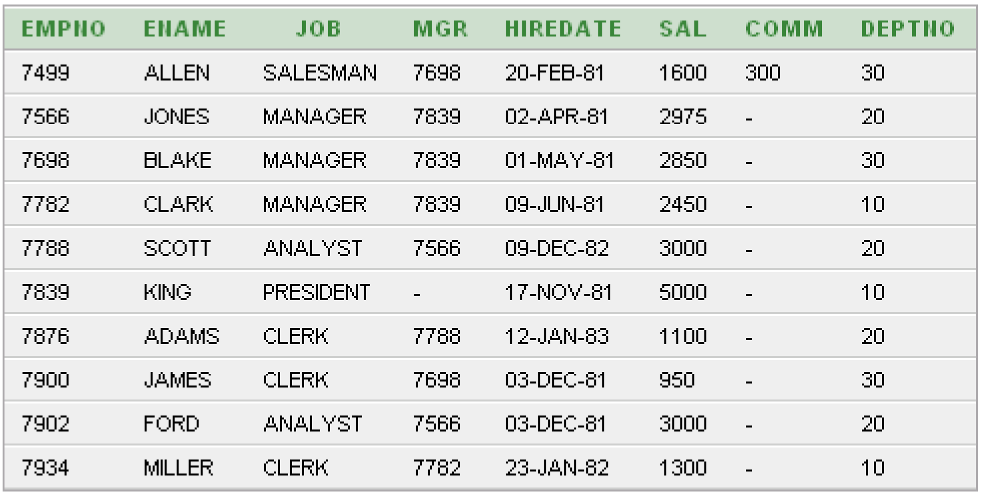
SQL> select empno, ename, hiredate from EMP where hiredate > '20-mar-82';
O/P:-
2 rows selected



EMPNO	ENAME	HIREDATE
7788	SCOTT	09-DEC-82
7876	ADAMS	12-JAN-83

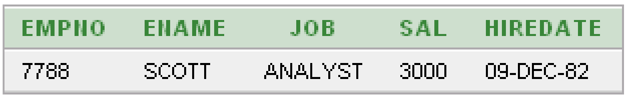
SQL> select * from emp where sal not in(1250,1500);

O/P:- 10 rows selected



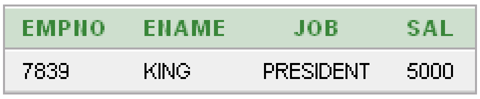
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7876	ADAMS	CLERK	7788	12-JAN-83	1100	-	20
7900	JAMES	CLERK	7698	03-DEC-81	950	-	30
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10

SQL> select empno, ename, job, sal, hiredate from EMP where ename like 'S%';
O/P:-
1 row selected



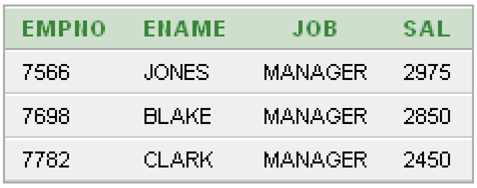
EMPNO	ENAME	JOB	SAL	HIREDATE
7788	SCOTT	ANALYST	3000	09-DEC-82

SQL> select empno, ename, job, sal from EMP where ename like 'K%' and job like 'PRESIDENT'; O/P:-
1 row selected



EMPNO	ENAME	JOB	SAL
7839	KING	PRESIDENT	5000

SQL> select empno, ename, job, sal from EMP where ename like 'B%' or job like 'MANAGER';
O/P:-
3 rows selected



EMPNO	ENAME	JOB	SAL
7566	JONES	MANAGER	2975
7698	BLAKE	MANAGER	2850
7782	CLARK	MANAGER	2450

Note: Execute above all commands & attach print out

CONCLUSION:

THUS WE HAVE STUDIED ARITHMATIC, LOGICAL, COMPARISON, DATA OPERATORS

EXPERIMENT NO. 05

TO USE ARITHMATHIC CHARACTER, NUMERIC, DATA FUNCTION

AIM :

TO USE ARITHMATHIC CHARACTER, NUMERIC, DATA FUNCTION.

THEORY:

Character function accepts input and return either character or number

SOME EXAMPLES:

1.

```
SQL> select initcap (“hello‟) from dual;
O/P:- INITCAP
------------------
Hello

SQL> select lower (“HELLO‟) from dual;
O/P:- LOWER
-------------------
Hello

SQL> select upper (“hello‟) from dual;
O/P: - UPPER
-------------------
HELLO

SQL> select ltrim (“DATABASE MANAGEMENT‟, “DATABASE‟) from dual; O/P:-
LTRIM
 --------------------
MANAGEMENT

SQL> select rtrim(“DATABASE MANAGEMENT‟, “MANAGEMENT‟) from dual;
O/P: - RTRIM
---------------------
DATABASE

SQL> select lpad (“HELLO‟, 8, ‟*‟) from dual;
O/P:- LPAD
 ***HELLO

SQL> select rpad (“HELLO‟, 8,  ‟*‟) from dual;
O/P: - RPAD
-------------------------
HELLO***
SQL> select substr (“TRIANGLE‟, 4, 5) from dual;
O/P: - SUBSTR
 --------------------------
```

```
ANGLE

SQL> select length (“HELLO‟) from dual;
O/P:- LENGTH
----------------------------

 SQL> select translate (“CLICK‟,‟C‟,‟F‟) from dual;
 O/P: - TRANSLATE
-----------------------------
FLICK

SQL> select replace (“CLICK AND CLOCK‟,  ‟CL‟,  ‟T‟) from dual;
O/P:- REPLACE
-----------------------------
TICK AND TOCK

SQL> select (“HELLO‟||‟WORLD‟) as “CONCATE‟ from dual;
 O/P:- CONCATE
----------------------------
HELLOWORLD

SQL> select sin(40),cos(40),tan(40) from dual;
O/P:- SIN(40)        COS(40)            TAN(40)
 ------------------- -------------------- --------------------
0.74511316        -0.66693806        -1.1172149

SQL> select abs(-90),exp(4) from dual;
O/P:-   ABS      EXP
---------------  -------------------
90                  54.59815

SQL> select ceil(88.88),floor(88.88) from dual;
O/P:-   CEIL      FLOOR
    --------------     -------------------------
          89           88

SQL> select round(88.8888,2),trunc(88.8888,2) from dual;
O/P:- ROUND          TRUNC
----------------------    --------------------
88.89                        88.88

SQL> select sqrt(25),power(4,3) from dual;
O/P:- SQRT               POWER
---------------------- -----------------------
```

```
    5                         64
```

```
SQL> select sysdate from dual;
O/P:- SYSDATE
 ----------------------
07-APR-09
```

```
SQL> select add_months (sysdate, 4) from dual;
O/P:- ADD_MONTHS
 07-AUG-09
```

```
SQL> select months_between(‟01-MAR-08‟,‟01-AUG-89‟) from dual;
O/P:- MONTHS_BETWEEN
       -----------------------------
                223
```

Note:Execute all the commands & attach printouts.

CONCLUSION:

THUS WE HAVE STUDIED NUMERIC, ARITHMATIC DATA FUNCTION

EXPERIMENT NO. 06

TO STUDY SET OPERATORS ON DATABASE OBJECTS

AIM: TO STUDY SET OPERATORS ON DATABASE OBJECTS

THEORY:

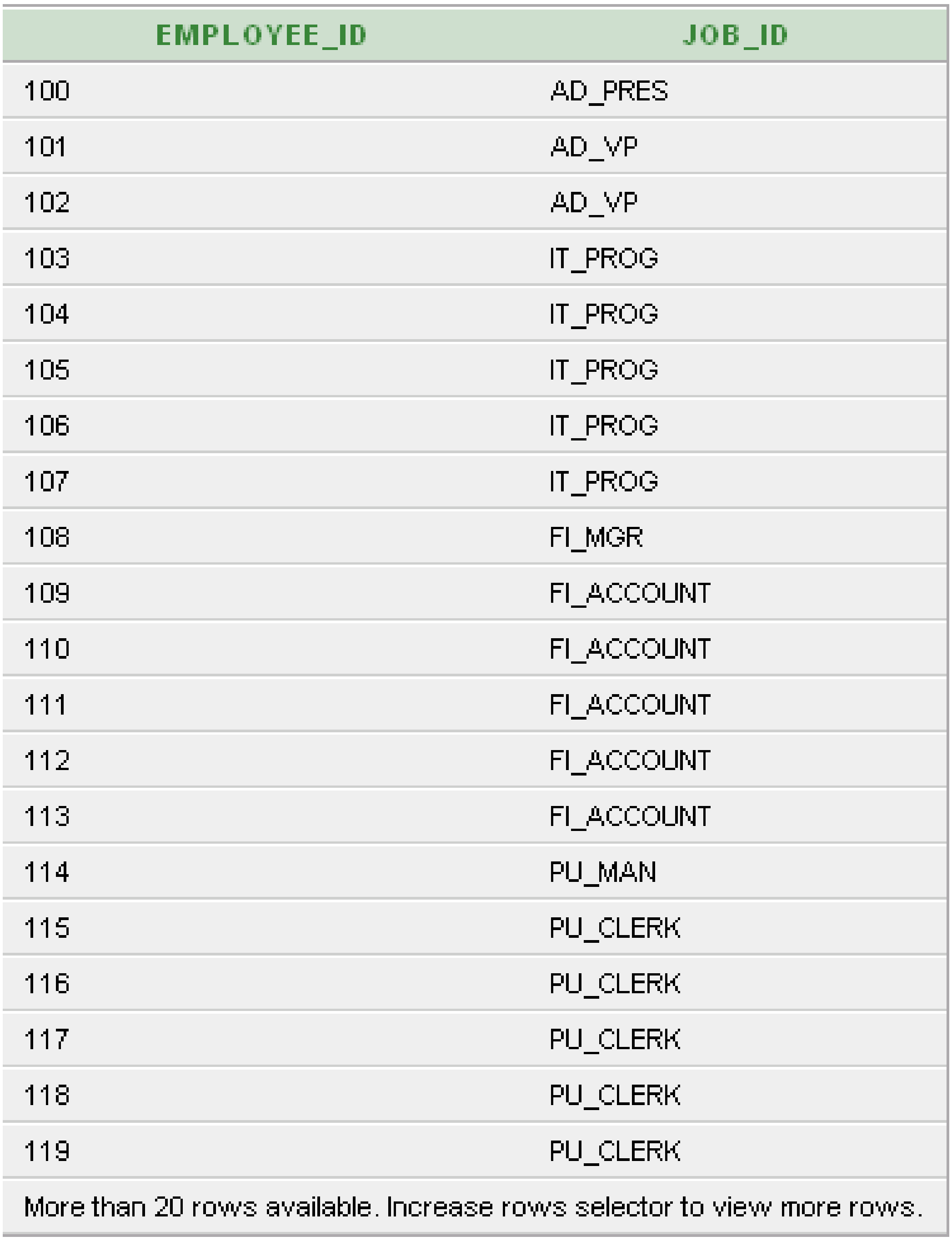
Set Operations

1. SQL has the set operations union, intersect and except.
2. Find all customers having an account.
   select distinct *cname*
   from *depositor*
3. union: Find all cnstomers having a loan, an account, or both. branch.
   (select cname
   from *depositor)*
   union
   (select cname
   from *borrower)*
4. intersect: Find customers having a loan and an account.
   (select distinct *cname*
   from *depositor)*
   intersect
   (select distinct *cname*
   from *borrower)*
5. except: Find customers having an account, but not a loan.
   (select distinct *cname*
   from *depositor)*
   except
   (select *cname*
   from *borrower)*

SOME EXAMPLES:

```
SQL>  select employee_id,job_id
        from employees
        minus
        select employee_id,job_id
        from job_history;
```

O/P:-

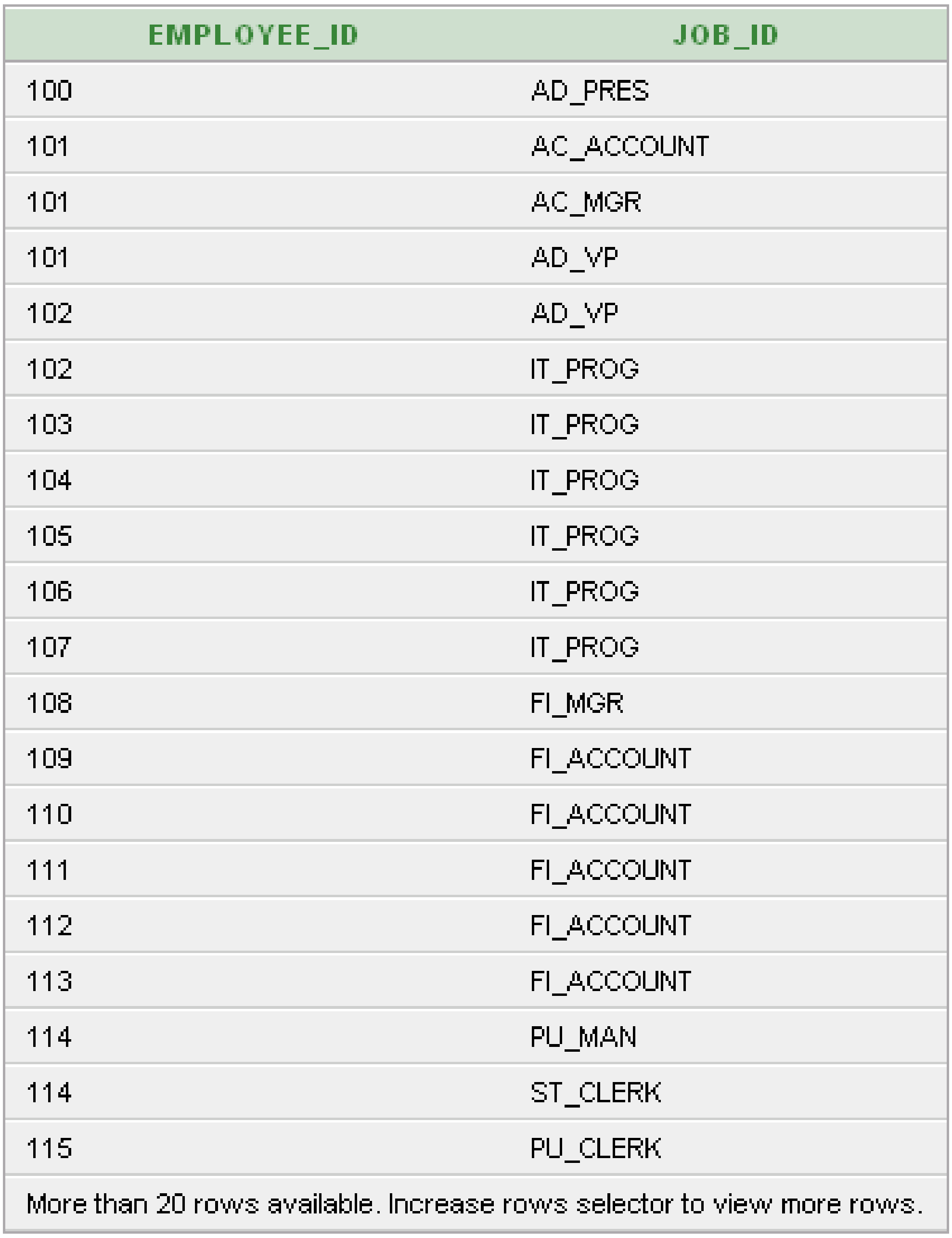


EMPLOYEE_ID	JOB_ID
100	AD_PRES
101	AD_VP
102	AD_VP
103	IT_PROG
104	IT_PROG
105	IT_PROG
106	IT_PROG
107	IT_PROG
108	FI_MGR
109	FI_ACCOUNT
110	FI_ACCOUNT
111	FI_ACCOUNT
112	FI_ACCOUNT
113	FI_ACCOUNT
114	PU_MAN
115	PU_CLERK
116	PU_CLERK
117	PU_CLERK
118	PU_CLERK
119	PU_CLERK

More than 20 rows available. Increase rows selector to view more rows.

```
SQL>  select employee_id,job_id
        from employees
        union
        select employee_id,job_id
        from job_history;
```

O/P:-

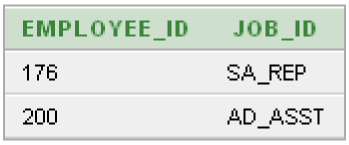


EMPLOYEE_ID	JOB_ID
100	AD_PRES
101	AC_ACCOUNT
101	AC_MGR
101	AD_VP
102	AD_VP
102	IT_PROG
103	IT_PROG
104	IT_PROG
105	IT_PROG
106	IT_PROG
107	IT_PROG
108	FI_MGR
109	FI_ACCOUNT
110	FI_ACCOUNT
111	FI_ACCOUNT
112	FI_ACCOUNT
113	FI_ACCOUNT
114	PU_MAN
114	ST_CLERK
115	PU_CLERK

More than 20 rows available. Increase rows selector to view more rows.

```
SQL>  select employee_id,job_id
        from employees
        intersect
        select employee_id,job_id
        from job_history
```

O/P:-



EMPLOYEE_ID	JOB_ID
176	SA_REP
200	AD_ASST

2 rows selected.

Note: Execute all commands & attach printouts

CONCLUSION:

THUS WE HAVE STUDIED AGGREGATE FUNCTIONS AND CLAUSES

EXPERIMENT NO. 07

TO STUDY CLAUSES AND AGGREGATE FUNCTIONS

AIM: TO STUDY CLAUSES AND AGGREGATE FUNCTIONS

THEORY:

CLAUSES:

1. Basic structure of an SQL expression consists of select, from and where clauses.

- select clause lists attributes to be copied - corresponds to relational algebra project.
- from clause corresponds to Cartesian product - lists relations to be used.
- where clause corresponds to selection predicate in relational algebra.

Aggregate Functions

1. In SQL we can compute functions on groups of tuples using the group by clause. Attributes given are used to form groups with the same values. SQL can then compute

- average value avg
- minimum value mm
- maximum value — max
- total sum of values sum
- number in group — count

These are called aggregate functions. They return a single value.

2. Some examples:

(a) Find the average account balance at each branch.

select *bname,* avg *(balance)*
from *account*
group by *bname*

(b) Find the number of depositors at each' branch.

select *bname,* count (distinct *cname)*
from *account, depositor*

where *account.accoun# = deposor.account*
group by *bname*
We use distinct so that a person having more than one account will not he counted more than once.

(c) Find branches and their average balances where the average balance is more than .1200. select *bname*, avg *(balance)*

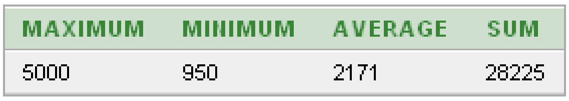
from *account*
group by *bname*
having avg *(balance)* > 1200

Predicates in the having clause are applied after the formation of groups.

SOME EXAMPLES :

```
SQL> select max (sal) "MAXIMUM”, in(sal) "MINIMUM”, round(avg(sal),0)
        "AVERAGE”, sum (sal) "SUM"
        From EMP;
```

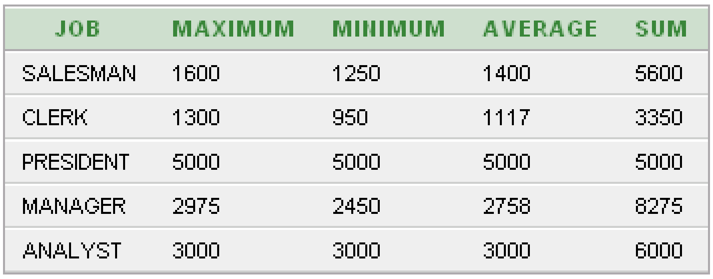
O/P:-



MAXIMUM	MINIMUM	AVERAGE	SUM
5000	950	2171	28225

```
SQL>  select job,max(sal) "MAXIMUM",min(sal) "MINIMUM",round(avg(sal),0)
        "AVERAGE",sum(sal) "SUM"
        from emp
        group by job;
```

O/P:-

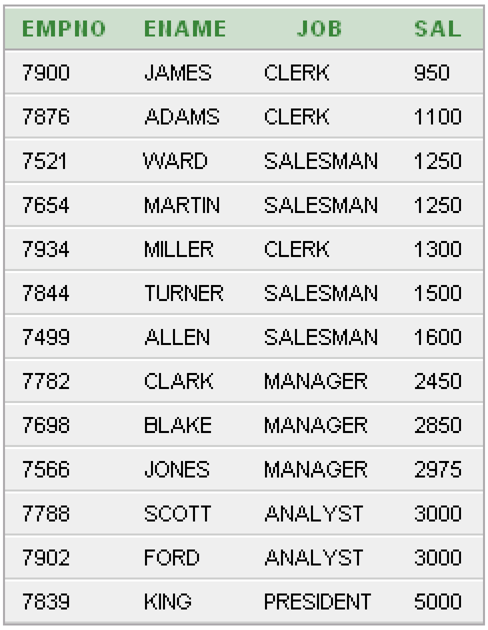


JOB	MAXIMUM	MINIMUM	AVERAGE	SUM
SALESMAN	1600	1250	1400	5600
CLERK	1300	950	1117	3350
PRESIDENT	5000	5000	5000	5000
MANAGER	2975	2450	2758	8275
ANALYST	3000	3000	3000	6000

5 rows selected.

```
SQL>  select empno,ename,job,sal
         from emp
         order by sal;
```

O/P:-

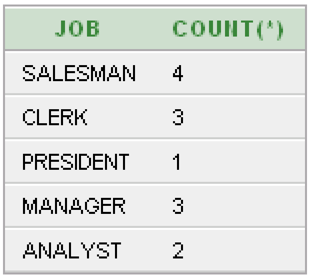


EMPNO	ENAME	JOB	SAL
7900	JAMES	CLERK	950
7876	ADAMS	CLERK	1100
7521	WARD	SALESMAN	1250
7654	MARTIN	SALESMAN	1250
7934	MILLER	CLERK	1300
7844	TURNER	SALESMAN	1500
7499	ALLEN	SALESMAN	1600
7782	CLARK	MANAGER	2450
7698	BLAKE	MANAGER	2850
7566	JONES	MANAGER	2975
7788	SCOTT	ANALYST	3000
7902	FORD	ANALYST	3000
7839	KING	PRESIDENT	5000

13 rows selected

```
SQL>  select job,count(*)
         from emp
         group by job;
```

O/P:-

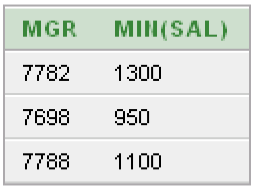


JOB	COUNT(*)
SALESMAN	4
CLERK	3
PRESIDENT	1
MANAGER	3
ANALYST	2

5 rows selected.

```
SQL>  select mgr,min(sal)
         from emp
         group by mgr
         having min(sal)<2000;
```

O/P:-



MGR	MIN(SAL)
7782	1300
7698	950
7788	1100

3 rows selected.

Note: Execute all commands & attach printouts

<u>CONCLUSION:</u>

THUS WE HAVE STUDIED CLAUSES AND AGGREGATE FUNCTIONS.

EXPERIMENT NO. 08

TO STUDY JOINS & SUB-QUERIES IN SQL

AIM: TO STUDY JOINS & SUB-QUERIES IN SQL

THEORY:

1. Two given relations: *loan* and *borrower.*
2. inner join:

   loan inner join *borrower* on *loan. loan# borrower. loan#*

   Notice that the loan# will appear twice in the inner joined relation.

bname	loan#	amount
Downtown	L-170	3000
Redwood	L-230	4000
Perryridge	L-260	1700

cname	loan#
Jones	L-170
Smith	L-230
Hayes	L-155

Figure 4.1: The *loan* and *borrower* relations.

Bname	loan#	amount	cname	loan#
Downtown	L-170	3000	Jones	L-170
Redwood	L-230	4000	Smith	L-230

Figure 4.2: Result of *loan* inner join *borrower.*

3. left outer join:

   loan left outer join *borrower* on *loan.loan# = borrower.loan*

4. natural inner join:

   loan natural inner join *borrower*

Join types and conditions :

1. Each variant of the join operations in SQL-92 consists of a *join lype* and a *join condz2on.*

2. Join types: inner join, left outer join, right outer join, full outer join. The keyword inner and outer are optional since the rest of the join type enables us to deduce whether the join is an inner join or an outer join.

SQL-92 also provides two other join types:

(a) Cross join: an inner join without a join condition.

(b) Union join: a full outer join on the fa1se” condition, i.e., where the inner join is empty.

1. Join conditions: natural, on predicate, using *(A_1,A_2....A_n)* The use of join condition is mandatory for outer joins, hut is optional for inner joins (if it is omitted, a Cartesian product results).

2. Ex. A natural full outer join:

loan natural full outer join *borrower* using (loan#)

bname	loan#	amount	cname	loan#
Downtown	L-170	3000	Jones	L-170
Redwood	L-230	4000	Smith	L-230
Perryridge	L-260	1700	*null*	null

Figure 4.3: Result of *loan* left outer join *borrower.*

bname	loan#	amount	cname
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith

Figure 4.4: Result of *loan* natural inner join *borrower.*

bname	loan#	amount	cname
Downtown	L-170	3000	Jones
Redwood	L-230	4000	Smith
Perryridge	L-260	1700	null
null	L-155	null	Hayes

Figure 4.5: Result of *loan* natural full outer join *borrower* using (loan#).

3. Ex. Find all customers who have either an account or a loan (but not both) at the bank.

select cname from (natural full outer join *borrower)* where *account#* is *null* or *loan#* is *null*

Note: Explain all types of joins.& Execute all commands & attach printouts

CONCLUSION:

THUS WE HAVE STUDIED JOINS AND SUB-QUEIRES

EXPERIMENT NO. 09

TO STUDY VIEWS AND TRIGERS IN SQL

AIM: TO STUDY VIEWS AND TRIGERS IN SQL

THEORY:

Views :

1. We have assumed up to now that the relations we are given are the actual relations stored in the database.
2. For security and convenience reasons, we may wish to create a personalized collection of relations for a user.
3. We use the term view to refer to any relation, not part of the conceptual model, that is made visible to the user as a “virtual relation”.
4. As relations may be modified by deletions, insertions and updates, it is generally not possible to store views. (Why?) Views must then be recomputed for each query referring to them.

View Definition:

1. A view is defined using the create view command: **create view** v as ⟨query expression> where ⟨query expression⟩ is any legal query expression. The view created is given the name v.
2. To create a view all_customer of all branches and their customers: create view all_customer as

$$\pi_{bname,cname}(deposit) \cap \pi_{bname,cname}(borrow)$$

3. Having defined a view, we can now use it to refer to the virtual relation it creates. View names can appear anywhere a relation name can.
4. We can now find all customers of the SFIJ branch by writing

πcname(σbname□’SFU” (all_customer))

TRIGGERS:

1. Another feature not present in the SQL standard is the **trigger.** Several existing systems have their own non-standard trigger features.

2. A trigger is a statement that is automatically executed by the system as a side effect of a modification to the database.

3. We need to

   - Specify the conditions under which the trigger is executed.
   - Specify the actions to be taken by the trigger.

4. For example, suppose that an overdraft is intended to result in the account balance being set to zero, and a loan being created for the overdraft amount. The trigger actions for tuple **t** with a negative balance are then

   - I nsert a new tuple *s* in the *borrow* relation with

   s[bname] = t[bname]
   s[loan#] = t[account#]
   s[amount] = —t[balance]
   s[cname] = I [ename]

   - We need to negate balance to get amount, as balance is negative.
   - Set t [balance] to 0.

   Note that this is not a good example. What would happen if the customer already had a loan?

5. SQL-92 does not include triggers. To write this trigger in terms of the original System R. trigger:

   define trigger *overdraft*
   on update of *account T*
   (if new *T.balance* < 0
   then (insert into *loan* **values**
   (T.bname, T.account#, — **new** *T.balance)*

```
                    insert into borrower
                    (select cname, account#
                    from depositor
                    where T. account# = depositor. account#)
                    update account S
                    set S.balance = 0
                    where S.account# = T.account# ))
```

SOME EXAMPLES:

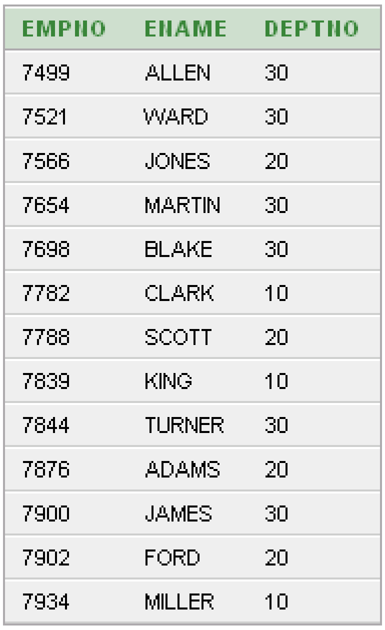
```
SQL>  create or replace view emp_vu as
        select empno,ename,deptno
        from emp;
```

O/P:-

view created

```
SQL>  select * from emp_vu
```

O/P:-



EMPNO	ENAME	DEPTNO
7499	ALLEN	30
7521	WARD	30
7566	JONES	20
7654	MARTIN	30
7698	BLAKE	30
7782	CLARK	10
7788	SCOTT	20
7839	KING	10
7844	TURNER	30
7876	ADAMS	20
7900	JAMES	30
7902	FORD	20
7934	MILLER	10

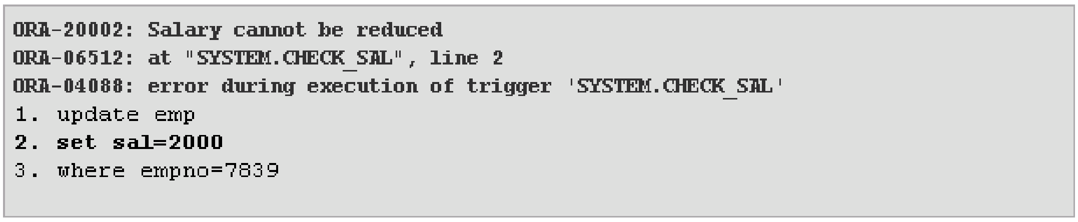
13 rows selected

```
SQL>  create or replace trigger check_sal
        before update of sal on emp
        for each row
        when (new.sal<old.sal)
        begin
                raise_application_error(-20002,'Salary cannot be reduced');
        end check_sal;
```

O/P: - trigger created

```
SQL>  update emp
        set sal=2000
        where empno=7839;
```

O/P:-



```
ORA-20002: Salary cannot be reduced
ORA-06512: at "SYSTEM.CHECK_SAL", line 2
ORA-04088: error during execution of trigger 'SYSTEM.CHECK_SAL'
1. update emp
2. set sal=2000
3. where empno=7839
```

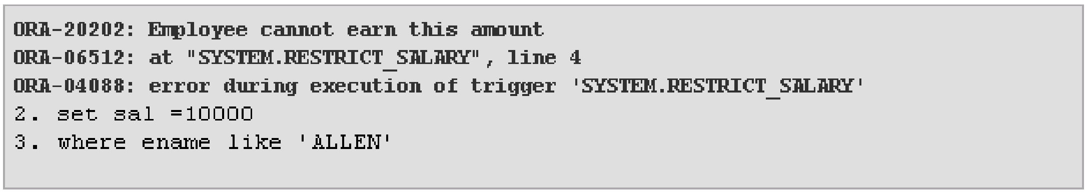
```
SQL>  create or replace trigger restrict_salary
        before insert or update of sal on emp
        for each row
        begin
                if not (:new.job in ('PRESIDENT'))
                and :new.sal >5000
                        then raise_application_error (-20202,'Employee cannot earn this
                        amount');
                end if;
        end;
```

O/P:-

        trigger created

```
SQL>  update emp
        set sal =10000
        where ename like 'ALLEN';
```

O/P:-



```
ORA-20202: Employee cannot earn this amount
ORA-06512: at "SYSTEM.RESTRICT_SALARY", line 4
ORA-04088: error during execution of trigger 'SYSTEM.RESTRICT_SALARY'
2. set sal =10000
3. where ename like 'ALLEN'
```

CONCLUSION:

THUS WE HAVE STUDIED VIEWS AND TRIGGERS IN SQL

EXPERIMENT NO. 10

STUDY EXPERIMENT (INTRODUCTION TO PL/SQL)

STUDY EXPERIMENT: **INTRODUCTION TO PL / SQL**

Write a short note on PL/SQL

EXPERIMENT NO. 11

WAP FOR DECLARING & USING VARIABLES CONSTANT USING LOOPS AND DATA STRUCTURE IN PL / SQL

AIM:

WAP FOR DECLARING & USING VARIABLES CONSTANT USING LOOPS AND DATA STRUCTURE IN PL / SQL

THEORY:

LOOPS IN PL / SQL

WHILE LOOP

DO WHILE LOOP

FOR LOOP (MOST PREFERED LOOP)

SOME EXAMPLES:

```
SQL>  create or replace procedure raise_salary

        (p_id in emp.empno%type)

        is

        begin

                update emp

                set sal=sal*10

                where empno=p_id;

        end raise_salary;


O/P:-  Procedure created


SQL>  EXECUTE raise_salary(7839);


PL/SQL procedure completed successfully.


SQL>  create or replace procedure add_job
```

```
    (job_id varchar2,title varchar2)

    is

    begin

        insert into jobs(job_id,job_title)

        values(job_id,title);

    end;
```

O/P:- Procedure created

```
EXECUTE add_job(100,'Engineer')
```

O/P:- PL/SQL procedure completed successfully.

```
SQL>  create or replace function get_sal

    (p_id in emp.empno%type)
    return number
    is v_sal emp.sal%type:=0;
    begin
        select sal
        into n_sal
        from emp
        where empno=p_id;
        return v_sal;
    end get_sal;
```

O/P:- Function created

```
SQL>  variable g_sal number;

SQL>  execute :g_sal:= get_sal(7839);
```

O/P:- PLSQL procedure completed successfully

```
SQL>  print g_sal;
```

O/P:-

```
G_SAL
```

```
-----------------------
50000
```

Note:Execute program & attach printouts

CONCLUSION:

THUS WE HAVE STUDIED DECLARING AND USING VARIABLES CONST USING LOOPS AND DATA STRUCTURE IN SQL

EXPERIMENT NO. 12

TO STUDY PROCEDURE AND FUNCTIONS IN PL / SQL

AIM: TO STUDY PROCEDURE AND FUNCTIONS IN PL / SQL

THEORY:

PROCEDURE: is a sub-program that performs a specific action

SOME EXAMPLES FOR PROCEDURE ARE:

--- can be called in SQL * Plus, PL * SQL Forms, Reports, Menus, Graphics, Java, .Net, VB…………

Global Procedure: all programs make them stored procedure, so 'exe' is created (executable)

```
SQL>  begin
        for i in 1..10 loop
                dbms_output.put_line(i);
        end loop;
        end;
```

O/P:-

```
1
2
3
4
5
6
7
8
9
10
```

```
PL/SQL procedure completed successfully

SQL>  declare
        v_empno emp.empno%type := 7839;
        v_sal emp.sal%type;
        v_bonus_per number(7,2);
        v_bonus number(7,2);
        begin
                select sal
                into v_sal
                from emp
                where empno=v_empno;
                if v_sal<1000 then
                        v_bonus_per:=.10;
                elsif v_sal between 1000 and 2000 then
                        v_bonus_per:=.15;
                        elsif v_sal>3000 then
                        v_bonus_per:=.20;
                else
                        v_bonus_per:=0;
                end if;
                v_bonus:=v_sal*v_bonus_per;

                dbms_output.put_line('The bonus for the employee with
                 emp_id'||v_empno||'and salary'||v_sal||'is'||v_bonus);
        end

O/P:-
The bonus for the employee with emp_id 7839 and salary 5000 is 1000

PL/SQL procedure completed successfully

SQL>  declare
        i number :=1;
```

```
j number :=2;
begin
        while(i<10)
        loop
                if (i%j=0) then
                        dbms_output.put_line(i||' '||'not a prime number');
                end if;
                j=j+1;
                if(j=i) then
                dbms_output.put_line(i||' '|| 'prime number');
                end if;
        end loop;
        dbms_output.put_line(i||' '||'prime number');
end;
```

O/P:-

```
1     prime number
2     prime number
3     prime number
4     not a prime number
5     prime number
6     not a prime number
7     prime number
8     not a prime number
9     not a prime number
10    not a prime number

PL/SQL procedure completed successfully.
```

Note: Execute program & attach printouts

CONCLUSION:

THUS WE HAVE STUDIED PROCEDURE & FUNCTION IN PL / SQL