KIT - KALAIGNARKARUNANIDHI INSTITUTE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTION)
Coimbatore-641402.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ACADEMIC YEAR 2020-2021

B19CSP401- DATABASE MANAGEMENT SYSTEMS LABORATORY

Name:_____

Roll No.:_____ Reg No.:_____

Year/Semester.:

_____

Department:

_____

KIT - KALAIGNARKARUNANIDHI INSTITUTE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTION)
Coimbatore-641402.

BONAFIDE CERTIFICATE

Name of the Student: _____

Roll No._____Register No. _____

Branch: B.E- COMPUTER SCIENCE AND ENGINEERING

Subject Code & Name: **B19CSP401- DATABASE MANAGEMENT SYSTEMS LABORATORY**

Certified that this is a bonafide record work done by

Mr. /Ms. _____

of _____II CSE_____ during the year 2020-2021`.

_____ _____

Signature of the Faculty In-charge Signature of the HOD

Submitted for the Board of Practical Examination held on _____

_____ _____

INTERNAL EXAMINER EXTERNAL EXAMINER

Practical Record Book Index Page

Sl. No.	Date	Name of the Experiment	Page Number	Program (20 Marks)	Execution of the Program (25Marks)	Output & Inference (10 Marks)	Viva-Voce (20 Marks)	Total (75 Marks)	Signature of the Faculty Member

Model Exam Marks (25):_____Total (100):_____ **Signature of the Faculty Member**

Practical Record Book Index Page

Sl. No.	Date	Name of the Experiment	Page Number	Program (20 Marks)	Execution of the Program (25Marks)	Output & Inference (10 Marks)	Viva-Voce (20 Marks)	Total (75 Marks)	Signature of the Faculty Member

Model Exam Marks (25):_____Total (100):_____ **Signature of the Faculty Member**

Practical Record Book Index Page

Sl. No.	Date	Name of the Experiment	Page Number	Program (20 Marks)	Execution of the Program (25Marks)	Output & Inference (10 Marks)	Viva-Voce (20 Marks)	Total (75 Marks)	Signature of the Faculty Member

Model Exam Marks (25):_____Total (100):_____ **Signature of the Faculty Member**

VISION

To produce intellectual graduates to excel in the field of Computer Science Engineering and Technologies.

MISSION

- Providing excellent and intellectual inputs to the students through qualified faculty members.
- Imparting technical knowledge in latest technologies through the industry institute interaction and thereby making the graduates ready for the industrial environment.
- Enriching the student's knowledge for active participation in co-curricular and extracurricular activities.
- Promoting research based projects in contexts to social, legal and technical aspects.

PROGRAMME OUTCOMES (POs)

Engineering Graduates will be able to:

PO1 Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex Computer Science Engineering problems.

PO2 Problem Analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and computer engineering sciences.

PO3 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 in the field of Computer Science and Engineering.

PO4 Conduct investigations of complex problems: Using research-based knowledge and computer science oriented research methodologies including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5 Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex computer science engineering activities with an understanding of the limitations.

PO6 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.

PO7 Environment and Sustainability: Understand the impact of the professional Computer Science Engineering solutions in societal and environmental contexts, and demonstrate the knowledge, and

need for the sustainable development.

PO8 Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9 Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams and in multidisciplinary settings.

PO10Communication: 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.

PO11 Project management and finance: Demonstrate knowledge and understanding of the computer science engineering and management principles and apply these to one's own work, as a member and leader in a team and, to manage projects in multidisciplinary environments.

PO12 Lifelong 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 EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduates will be successful in their profession by taking part actively in the field of software and technology.

PEO2: Graduates will be proficient in analyzing and facing the challenges in Computer Science and Engineering.

PEO3: Graduates will engage in lifelong learning activities by adapting to the advanced software technologies for continuous professional development.

PROGRAM SPECIFIC OUTCOME (PSOs)

PSO1 Categorize the basic engineering knowledge to solve the problems in Computer Science and Engineering according to the environmental needs.

PSO2 Apply the modern tools to design and develop the software system ethically to the industrial needs.

COURSE OUTCOMES

At the end of this course, the student will be able to:

Course Outcomes	Knowledge Level
CO1: Utilize typical data definitions and manipulation commands.	K3
CO2: Develop applications to test Nested and Join Queries	K3
CO3: Build simple applications that use Views	K3
CO4: Construct PL/SQL blocks using Cursors	K3
CO5: Identify the use of Tables, Views, Triggers, Functions and Procedures	K3
CO6: Make use of Front-end Tool in Database applications	K3

CO/PO & PSO		PO1 (K3)	PO2 (K4)	PO3 (K5)	PO4 (K5)	PO5 (K6)	PO6 (K3) (A3)	PO7 (K2) (A3)	PO8 (K3) (A3)	PO9 (A3)	PO10 (A3)	PO11 (K3) (A3)	PO12 (A3)	PSO1 (K4 A3)	PSO2 (K3 A3)
CO1	K3	3	2	1	-	-	-	-	-	-	-	-	-	2	3
CO2	K3	3	2	1	-	-	-	-	-	-	-	-	-	2	3
CO3	K3	3	2	1	1	-	-	-	-	-	-	-	-	2	3
CO4	K3	3	2	-	1	-	-	-	-	-	-	-	-	2	3
CO5	K3	3	2	1	-	-	-	-	-	-	-	-	-	2	3
CO6	K3	3	2	-	1	-	-	-	-	-	-	-	-	2	3
Weighted average		3	2	1	1	-	-	-	-	-	-	-	-	2	3

SYLLABUS

LIST OF EXPERIMENTS:

1. Data Definition Commands, Data Manipulation Commands for inserting, deleting, updating and retrieving Tables and Transaction Control statements.
2. Database Querying – Simple queries, Nested queries, Sub queries and Joins.
3. Views, Sequences, Synonyms.
4. Database Programming: Implicit and Explicit Cursors.
5. Procedures and Functions.
6. Triggers.
7. Exception Handling.
8. Database Design using ER modeling, normalization and Implementation for any application.
9. Database Connectivity with Front End Tools
10. Case Study using real life database applications

CONTENT

Ex.No.	1	Creation of A Database and Writing SQL Queries to Retrieve Information from the Database.
Date		

AIM:

To create a database and to retrieve the information from the database using SQL queries.

ALGORITHM:

STEP-1: Login to the database using user id and password.

STEP-2: Create database table using SQL commands.

STEP-3: Enter data to the table using SQL commands.

STEP-4: Retrieve information from the table using SQL commands.

STEP-5: Close the database.

COMMAND/OUTPUT:

SQL> create table stud (sname varchar2(30), sid varchar2(10), sage number(2), sarea varchar2(20)); Table created.

SQL> desc stud;

Name	Null?	Type
SNAME		VARCHAR2(30)
SID		VARCHAR2(10)
SAGE		NUMBER(2)
SAREA		VARCHAR2(20)

SQL>alter table stud modify (sage number(10)); Table altered.

SQL> alter table stud add (sdept varchar2(20)); Table altered.

SQL> desc stud;

Name	Null?	Type
SNAME		VARCHAR2(30)
SID		VARCHAR2(10)
SAGE		NUMBER(10)
SAREA		VARCHAR2(20)

SDEPT VARCHAR2(20)

SQL> alter table stud drop (sdept varchar2(20));

Table altered.

SQL> desc studs;

Name Null? Type

SNAME VARCHAR2(30)

SID VARCHAR2(10)

SAGE NUMBER(10)

SAREA VARCHAR2(20)

SQL> truncate table studs;

Table truncated.

SQL> desc studs;

Name Null? Type

SNAME VARCHAR2(30)

SID VARCHAR2(10)

SAGE NUMBER(10)

SAREA VARCHAR2(20)

SDEPT VARCHAR2(20)

SQL> drop table studs; Table dropped.

INFERENCE:

REAL TIME APPLICATION:
Banks, IT and companies are using these commands to develop their databases.

VIVA QUESTIONS:

1. What is database?

2. What is table in database?

3. What are the advantages of databases?

4. What is SQL?

5. What is record?

RESULT

Thus the creation of database and the SQL queries to retrieve information from the database has been implemented and the output was verified.

Ex.No.	2	**Data Manipulation Commands for inserting, deleting, updating and retrieving Tables and Transaction Control statements.**
Date		

AIM:

To study the various categories of DML commands such as logical operations, aggregate functions, string functions, numeric functions, date functions, conversion functions, group functions and set operations.

ALGORITHM:

STEP-1: Login to the database using user id and password.

STEP-2: Create database table using SQL commands.

STEP-3: Enter data to the table using SQL commands.

STEP-4: Perform DML operations on database table using SQL commands.

STEP-5: Close the database.

COMMAND/OUTPUT:

CREATION OF TABLE

SQL>create table stud (sname varchar2(30), sid varchar2(10), sage number(10), sarea varchar2(20), sdept varchar2(20));

Table created.

INSERTION OF VALUES INTO THE TABLE

SQL> insert into stud values ('ashwin',101,19,'anna nagar','aeronautical'); 1 row created.

SQL> insert into stud values

('bhavesh',102,18,'nungambakkam','marine');

1 row created.

SQL> insert into stud values ('pruthvik',103,20,'anna nagar','aerospace');

1 row created.

SQL> insert into stud values

('charith',104,20,'kilpauk','mechanical'); 1 row created.

SQL> select * from stud;

SNAME	SID	SAGE	SAREA	SDEPT
ashwin	101	19	anna nagar	aeronautical
bhavesh	102	18	nungambakkam	marine
pruthvik	103	20	anna nagar	aerospace
charith	104	20	kilpauk	mechanical

RENAMING THE TABLE ‘STUD’

SQL> rename stud to studs;

Table renamed.

ARITHMETIC OPERATION

SQL> select sname, sid+100 "stid" from studs;

SNAME	stid
ashwin	201
bhavesh	202
pruthvik	203
charith	204

CONCATENATION OPERATOR

SQL> select sname || ' is a ' || sdept || ' engineer. ' AS "PROFESSION" from studs;

PROFESSION

ashwin is a aeronautical engineer.
bhavesh is a marine engineer.
pruthvik is a aerospace engineer.
charith is a mechanical engineer.

DISPLAY ONLY DISTINCT VALUES

SQL> select distinct sarea from studs; SAREA

anna nagar kilpauk

nungambakkam

USING THE WHERE CLAUSE

SQL> select sname,sage from studs where sage<=19;

SNAME	SAGE
ashwin	19
bhavesh	18

BETWEEN OPERATOR

SQL> select sname,sarea, sid from studs where sid between 102 and

104; SNAME SAREA SID

bhavesh nungambakkam 102

pruthvik anna nagar 103

charith kilpauk 104

IN PREDICATE

SQL> select sname,sarea , sid from studs where sid

in(102,104); SNAME SAREA SID

---

bhavesh nungambakkam 102

charith kilpauk 104

PATTERN MATCHING

SQL> select sname, sarea from studs where sarea like '%g%';

SNAME SAREA

---

ashwin anna nagar
bhavesh nungambakkam
pruthvik anna nagar

LOGICAL AND OPERATOR

SQL> select sname ,sid from studs where sid>102 and sarea='anna nagar';

SNAME SID

---

pruthvik 103

LOGICAL OR OPERATOR

SQL> select sname ,sid from studs where sid>102 or sarea='anna nagar';

SNAME SID

---

ashwin 101

pruthvik 103

charith 104

NOT IN PREDICATE

```
SQL> select sname, sid from studs where sid not  in(102,104);

SNAME                SID
-------------------------------- ----------
ashwin               101
pruthvik             103
```

UPDATING THE TABLE

```
SQL> alter table studs add ( spocket varchar2(20) );

Table altered.

SQL> update studs set spocket=750 where sid=101;
1 row updated.
SQL> update studs set spocket=500 where sid=102;
1 row updated.
SQL> update studs set spocket=250 where sid=103;
1 row updated.
SQL> update studs set spocket=100 where sid=104;
1 row updated.
SQL> select * from studs;

SNAME                SID        SAGE      SAREA        SDEPT        SPOCKET
-------------------------------- ---------- ---------- -------------------- ----------

ashwin               101        19        anna nagar           Aeronautical
750
bhavesh              102        18        nungambakkam         marine
500
pruthvik             103        20        anna nagar           aerospace
250
charith              104        20        kilpauk              mechanical
100
```

AGGREGATE FUNCTIONS

```
SQL> select avg( spocket ) result from studs;
  RESULT
   400

SQL> select min(spocket) result from studs;
RESULT

100

SQL> select count(spocket) result from studs;
  RESULT

      4

SQL> select count(*) result from studs;
  RESULT

      4

SQL> select count(spocket) result from studs where sarea='anna
  nagar'; RESULT
    ----------
      2

SQL> select max(spocket) result from studs;
RESULT

750

SQL> select sum(spocket) result from studs;
  RESULT

    1600
    ----------
```

NUMERIC FUNCTIONS

```
SQL> select abs(-20) result from dual;
  RESULT

     20
    ----------

SQL> select power (2,10) result from dual;
  RESULT
----------

    1024
```

```
SQL> select round(15.359,2) result from dual;
  RESULT

   15.36
   ----------

SQL> select sqrt (36) result from dual;
  RESULT
   ----------

      6
```

STRING FUNCTIONS

```
SQL> select lower('ORACLE') result from dual;
RESULT

oracle
   -------

SQL> select upper('oracle') result from dual;
RESULT
   -------

ORACLE

SQL> select initcap('Oracle') result from dual;
RESULT
   -------
Oracle

SQL> select substr('oracle' ,2 ,5) result from dual;
RESULT

Oracle
   ------
SQL> select lpad('oracle',10,'#') result from dual;

RESULT
oracle

SQL> select rpad ('oracle',10,'^') result from dual;
RESULT

oracle^^^^
   -----------
```

CONVERSION FUNCTIONS

```
SQL> update studs set sage=to_number(substr(118,2,3));
4 rows updated.
```

SQL> select * from studs;

SNAME	SID	SAGE	SAREA	SDEPT	SPOCKET
ashwin	101	18	anna nagar	aeronautical	750
bhavesh	102	18	nungambakkam	marine	500
pruthvik	103	18	anna nagar	aerospace	250
charith	104	18	kilpauk	mechanical	100

SQL> select to_char(17145, '099,999') result from dual;
RESULT

017,145

SQL> select to_char(sysdate,'dd-mon-yyyy') result from dual;
RESULT

16-jul-2008

DATE FUNCTIONS

SQL> select sysdate from dual;
SYSDATE

16-JUL-08

SQL> select sysdate,add_months(sysdate,4) result from dual;
SYSDATE RESULT

16-JUL-08 16-NOV-08

SQL> select sysdate, last_day(sysdate) result from dual;
SYSDATE RESULT

16-JUL-08 31-JUL-08

SQL> select sysdate, next_day(sysdate,'sunday') result from dual;
SYSDATE RESULT

16-JUL-08 20-JUL-08

SQL> select months_between('09-aug-91','11-mar-90') result from dual; RESULT

16.935484

GROUP BY CLAUSE

SQL> select sarea, sum(spocket) result from studs group by sarea; SAREA RESULT

anna nagar	1000
nungambakkam	500
kilpauk	100

HAVING CLAUSE

SQL> select sarea, sum(spocket) result from studs group by sarea having spocket<600;

SAREA	RESULT
nungambakkam	500
kilpauk	100

DELETION

SQL> delete from studs where sid=101; 1 row deleted.
SQL> select * from studs;

SNAME	SID	SAGE	SAREA	SDEPT	SPOCKET
bhavesh	102	18	nungambakkam	marine	500
pruthvik	103	20	anna nagar	aerospace	250
charith	104	20	kilpauk	mechanical	100

CREATING TABLES FOR DOING SET OPERATIONS TO CREATE PRODUCT TABLE

SQL> create table product(prodname varchar2(30), prodno varchar2(10)); Table created.
SQL> insert into product values('table',10001); 1 row created.
SQL> insert into product values('chair',10010);
1 row created.
SQL> insert into product values('desk',10110);
1 row created.
SQL> insert into product

values('cot',11110); 1 row created.
SQL> insert into product
values('sofa',10010); 1 row created.
SQL> insert into product values('tvstand',11010);
1 row created.
SQL> select * from product;

PRODNAME	PRODNO
table	10001
chair	10010
desk	10110
cot	11110
sofa	10010
tvstand	11010

TO CREATE SALE TABLE

SQL> create table sale(prodname varchar2(30),orderno number(10),prodno
varchar2(10)); Table created.
SQL> insert into sale values('table',801,10001);
1 row created.
SQL> insert into sale values('chair',805,10010);
1 row created.
SQL> insert into sale values('desk',809,10110);
1 row created.
SQL> insert into sale
values('cot',813,11110); 1 row created.
SQL> insert into sale values('sofa',817,10010);
1 row created.
SQL> select * from sale;

PRODNAME	ORDERNO	PRODNO
table	801	10001
chair	805	10010
desk	809	10110
cot	813	11110
sofa	817	10010

SET OPERATIONS

SQL> select prodname from product where prodno=10010 union select prodname from sale

where prodno=10010;
PRODNAME
chair sofa
SQL> select prodname from product where prodno=11110 intersect select prodname from sale where prodno=11110;
PRODNAME
cot

RESULT

The DML commands were executed and the output was verified.

AIM

To study the nested queries using DML commands.

TO CREATE SSTUD1 TABLE

SQL> create table sstud1 (sname varchar2(20) , place varchar2(20));
Table created.
SQL> insert into sstud1 values (
'prajan','chennai'); 1 row created.
SQL> insert into sstud1 values ('anand','chennai');
1 row created.
SQL> insert into sstud1 values ('kumar','chennai');
1 row created.
SQL> insert into sstud1 values ('ravi','chennai');
1 row created.
SQL> select * from sstud1;

SNAME	PLACE
prajan	chennai
anand	chennai
kumar	chennai
ravi	Chennai

TO CREATE SSTUD2 TABLE

SQL> create table sstud2 (sname varchar2(20), dept varchar2(10), marks number(10));
Table created.
SQL> insert into sstud2 values ('prajan','cse',700);
1 row created.
SQL> insert into sstud2 values ('anand','it',650);

1 row created.
SQL> insert into sstud2 values ('vasu','cse',680);
1 row created.
SQL> insert into sstud2 values ('ravi','it',600);
1 row created.
SQL> select * from sstud2;

SNAME	DEPT	MARKS
prajan	cse	700
anand	it	650
vasu	cse	680
ravi	it	600

NESTED QUERIES

SQL> select sname from sstud1 where sstud1.sname in (select sstud2.sname from sstud2);
SNAME
-anand
prajan
ravi

SQL> select sname from sstud1 where sstud1.sname not in (select sstud2.sname from sstud2);
SNAME
-kumar
SQL> select sname from sstud2 where marks > some(select marks from sstud2 where dept='cse');
SNAME
-prajan
SQL> select sname from sstud2 where marks >= some (select marks from sstud2 where dept='cse');
SNAM
-prajan
vasu
SQL> select sname from sstud2 where marks > any (select marks from sstud2 where dept='cse');
SNAME

prajan
SQL> select sname from sstud2 where marks >= any (select marks from sstud2 where dept='cse');

SNAME
prajan
vasu

SQL> select sname from sstud2 where marks > all (select marks from sstud2 where dept='cse');
no rows selected

SQL> select sname from sstud2 where marks < all (select marks from sstud2 where dept='cse');

SNAME

-anand
ravi

SQL> select sname from sstud1 where exists (select sstud2.sname from sstud2
where sstud1.sname=sstud2.sname);
SNAME
prajan
anand ravi
SQL> select sname from sstud1 where not exists (select sstud2.sname from sstud2
where sstud1.sname=sstud2.sname);
SNAME

-kumar

INFERENCE:

REAL TIME APPLICATION:
Telephone service provider generates bill using these commands.

VIVA QUESTIONS:

1. What is DML?

2. What is aggregate function?

3. What is the use of conversion function?

4. List any two string functions.

5. What you mean by SET operations?

RESULT

The Nested Queries using DML commands were executed and the output was verified.

Ex.No.	3	Creation of Views, Synonyms, Sequence, Indexes, Save point
Date		

AIM:

To create views, synonyms, sequences, indexes and save points using DDL, DML and DCL statements.

ALGORITHM:

STEP-1: Login to the database using user id and password.

STEP-2: Create database table using SQL commands.

STEP-3: Enter data to the table using SQL commands.

STEP-4: Create views and synonyms in database table.

STEP-5: Create index and save point in the database table.

STEP-6: Close the database.

COMMAND/OUTPUT:

TYPES OF VIEWS

- Updatable views – Allow data manipulation
- Read only views – Do not allow data manipulation

TO CREATE THE TABLE 'FVIEWS'

SQL> create table fviews(name varchar2(20),no number(5), sal number(5),

dno number(5));

Table created.

SQL> insert into fviews

values('xxx',1,19000,11); 1 row

created. SQL> insert into fviews

values('aaa',2,19000,12); 1 row

created.

SQL> insert into fviews

values('yyy',3,40000,13);

1 row created.

SQL> select * from fviews;

NAME NO SAL DNO

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

xxx 1 19000 11

aaa 2 19000 12

yyy 3 40000 13

TO CREATE THE TABLE 'DVIEWS'

SQL> create table dviews(dno number(5), dname varchar2(20));

Table created.

SQL> insert into dviews

values(11,'x'); 1 row

created. SQL> insert into

dviews values(12,'y'); 1

row created. SQL> select

* from dviews;

DNO DNAME

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

11 x

12 y

CREATING THE VIEW ‘SVIEW’ ON ‘FVIEWS’ TABLE

SQL> create view sview as select name,no,sal,dno from fviews where

dno=11; View created.

SQL> select * from sview;

NAME NO SAL DNO

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

xxx 1 19000 11

Updates made on the view are reflected only on the table when the structure of the table and the view are not similar -- proof

SQL> insert into sview values

('zzz',4,20000,14); 1 row created.

SQL> select * from sview;

NAME NO SAL DNO

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

xxx 1 19000 11

SQL> select * from fviews;

NAME NO SAL DNO

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

xxx 1 19000 11

aaa 2 19000 12

yyy 3 40000 13

zzz 4 20000 14

Updates made on the view are reflected on both the view and the table when the structure of the table and the view are similar – proof

CREATING A VIEW 'IVIEW' FOR THE TABLE 'FVIEWS'

SQL> create view iview as select *

from fviews; View created.

SQL> select * from iview;

NAME NO SAL DNO

---

xxx 1 19000 11

aaa 2 19000 12

yyy 3 40000 13

zzz 4 20000 14

PERFORMING UPDATE OPERATION

SQL> insert into iview values

('bbb',5,30000,15); 1 row

created. SQL> select * from

iview;

NAME NO SAL DNO

---

xxx 1 19000 11

bbb 5 30000 15

SQL> select * from fviews;

NAME NO SAL DNO

---

xxx 1 19000 11

aaa 2 19000 12

yyy 3 40000 13

zzz 4 20000 14

bbb 5 30000 15

CREATE A NEW VIEW ‘SSVIEW’ AND DROP THE VIEW

SQL> create view ssview(cusname,id) as select name, no from fviews where dno=12;

View created.

SQL> select * from ssview;

CUSNAME ID

---

aaa 2

SQL> drop view ssview;

View dropped.

TO CREATE A VIEW ‘COMBO’ USING BOTH THE TABLES ‘FVIEWS’ AND ‘DVIEWS’

SQL> create view combo as select name,no,sal,dviews.dno,dname from fviews,dviews where fviews.dno=dviews.dno;

View created.

SQL> select * from combo;

NAME NO SAL DNO DNAME

---

xxx 1 19000 11 x

aaa 2 19000 12 y

TO PERFORM MANIPULATIONS ON THIS VIEW

SQL> insert into combo values('ccc',12,1000,13,'x'); insert into combo

values('ccc',12,1000,13,'x')

*ERROR at line 1:

ORA-01779: cannot modify a column which maps to a non key-preserved table

This shows that when a view is created from two different tables no manipulations can be performed using that view and the above error is displayed.

Synonyms

SQL> select * from class;

NAME ID

anu	1
brindha	2
chinthiya	3
divya	4
ezhil	5
fairoz	7
hema	9

7 rows selected.

Create synonym:
SQL> create synonym c1 for class;

Synonym created.
SQL> insert into c1 values('kalai',20);

1 row created.
SQL> select * from class;

NAME	ID
anu	1
brindha	2
chinthiya	3
divya	4
ezhil	5
fairoz	7
hema	9
kalai	20

8 rows selected.

SQL> select * from c1;

NAME	ID
anu	1
brindha	2
chinthiya	3
divya	4
ezhil	5
fairoz	7
hema	9
kalai	20

8 rows selected.

SQL> insert into class values('Manu',21);

1 row created.

SQL> select * from c1;

NAME	ID
anu	1
brindha	2
chinthiya	3
divya	4
ezhil	5
fairoz	7
hema	9
kalai	20
Manu	21

9 rows selected.

Drop Synonym:

SQL> drop synonym c1;

Synonym dropped.

SQL> select * from c1;

```
select * from c1
       *
ERROR at line 1:
ORA-00942: table or view does not exist
```

Sequences

Oracle provides the capability to generate sequences of unique numbers, and they are called **sequences.**

Just like tables, views, indexes, and synonyms, a sequence is a type of database object.

Sequences are used to generate unique, sequential integer values that are used as primary key values in database tables.

The sequence of numbers can be generated in either ascending or descending order.

Creation of table:

SQL> create table class(name varchar(10),id number(10)); Table created.

Insert values into table:

SQL> insert into class values('&name',&id); Enter value for name: anu
Enter value for id: 1
old 1: insert into class values('&name',&id) new 1:
insert into class values('anu',1)

1 row created. SQL> /
Enter value for name: brindha
Enter value for id: 02
old 1: insert into class values('&name',&id) new 1:
insert into class values('brindha',02)

1 row created. SQL> /
Enter value for name: chinthiya
Enter value for id: 03
old 1: insert into class values('&name',&id) new 1:
insert into class values('chinthiya',03)

1 row created.

SQL> select * from class;

NAME	ID
anu	1
brindha	2
chinthiya	3

Create Sequence:

SQL> create sequence s_1 2 start
with 4
3 increment by 1
4 maxvalue 100
5 cycle; Sequence

created.

SQL> insert into class values('divya',s_1.nextval);

1 row created.

SQL> select * from class;

Name	ID
anu	1
brindha	2
chinthiya	3
divya	4

Alter Sequence:

SQL> alter sequence s_1 2
increment by 2;
Sequence altered.

SQL> insert into class values('fairoz',s_1.nextval); 1 row created.
SQL> select * from class;

NAME ID

NAME	ID
anu	1
brindha	2
chinthiya	3
divya	4
ezhil	5
fairoz	7

Drop Sequence:

SQL> drop sequence s_1; Sequence

dropped.

Indexes

An index can be created in a table to find data more quickly and efficiently.
The users cannot see the indexes; they are just used to speed up searches/queries.

Updating a table with indexes takes more time than updating a table without; because the indexes also need an update. So we should only create indexes on columns (and tables) that will be frequently searched against.

Syntax:

Create Index:

CREATE INDEX index_name ON table_name (column_name)

SQL> create table splr(sname varchar(10),sid number(10),scity varchar(10)); Table created.
SQL> insert into splr values('hcl',01,'chennai'); 1 row created.
SQL> insert into splr values('dell',04,'madurai'); 1 row created.
SQL> insert into splr values('HP',02,'kovai'); 1 row created.
SQL> insert into splr values('Lenovo',03,'trichy'); 1 row created.
SQL> select * from splr; SNAME

	SID	SCITY
hcl	1	chennai
dell	4	madurai
HP	2	kovai
Lenovo	3	trichy

SQL> create index sp1 on splr(sid); Index created.
SQL> create index sp2 on splr(sid,scity);

Index created.

Drop Index:

SQL> drop index sp1; Index dropped.
SQL> drop index sp2; Index dropped.

DCL statements

DESCRIPTION

The DCL language is used for controlling the access to the table and hence securing the database. DCL is used to provide certain privileges to a particular user. Privileges are rights to be allocated. The privilege commands are namely,

Grant Revoke

Commit
Savepoint
Rollback

GRANT COMMAND: It is used to create users and grant access to the database. It requires database administrator (DBA) privilege, except that a user can change their password. A user can grant access to their database objects to other users.

REVOKE COMMAND: Using this command , the DBA can revoke the granted database privileges from the user.

COMMIT : It is used to permanently save any transaction into database.

SAVEPOINT: It is used to temporarily save a transaction so that you can rollback to that point whenever necessary

ROLLBACK: It restores the database to last commited state. It is also use with savepoint command to jump to a savepoint in a transaction

SYNTAX

GRANT COMMAND

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

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

REVOKE COMMAND

Revoke <database_priv> from <user [, user] >;

Revoke <object_priv> on <object> from < user | public >;

<database_priv> -- Specifies the system level priveleges 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 priveleges.

[With Grant Option] – Allows the recipient user to give further grants on the objects.

The priveleges can be granted to different users by specifying their names or to all users by using the “Public” option.

COMMIT:

Commit;

SAVEPOINT:

Savepoint savapoint_name;

ROLLBACK:

Rollback to savepoint_name;

EXAMPLES

Consider the following tables namely “DEPARTMENTS” and
“EMPLOYEES” Their schemas are as follows ,
Departments (dept _no , dept_ name , dept_location);
Employees (emp_id , emp_name , emp_salary);
SQL> Grant all on employees to abcde;
Grant succeeded.
SQL> Grant select , update , insert on departments to abcde with grant option;
Grant succeeded.
SQL> Revoke all on employees from abcde;
Revoke succeeded.
SQL> Revoke select , update , insert on departments from abcde;
Revoke succeeded.

COMMIT, ROLLBACK and SAVEPOINT:

SQL> select * from class;

NAME	ID
anu	1
brindha	2
chinthiya	3
divya	4
ezhil	5
fairoz	7

SQL> insert into class values('gayathri',9);
1 row created.
SQL> commit; Commit
complete.
SQL> update class set name='hema' where id='9';
1 row updated.
SQL> savepoint A;
Savepoint created.

```
SQL> insert into class values('indu',11);
1 row created.
SQL> savepoint B;

Savepoint created.

SQL> insert into class values('janani',13);
1 row created.
SQL> select * from class;

NAME              ID
------------------
anu               1
brindha           2
chinthiya         3
divya             4
ezhil             5
fairoz            7
hema              9
indu              11
janani            13

9 rows selected.

SQL> rollback to B;
Rollback complete.
SQL> select * from class;

NAME              ID
------------------
anu               1
brindha           2
chinthiya         3
divya             4
ezhil             5
fairoz            7
hema              9
indu              11

8 rows selected.

 SQL> rollback to A;
Rollback complete.
SQL> select * from class;
NAME    ID
 anu              1
 brindha          2
 chinthiya        3
 divya            4
```

ezhil	5
fairoz	7
hema	9

INFERENCE:

REAL TIME APPLICATION:

Access restrictions in the web server can be implemented using these commands.

VIVA QUESTIONS:

1. What is DCL?

2. What is view?

3. Define synonyms.

4. Define save point.

5. What is the use of ROLLBACK?

RESULT:

Thus the Views, Synonyms, and Sequences, indexes and save points has been executed using DDL, DML and DCL statements

Ex.No.	4	Implementation of Cursors
Date		

AIM:

To implement cursors in a data base table using PL/SQL.

ALGORITHM:

STEP-1: Login to the database using user id and password.
STEP-2: Create database table using SQL commands.
STEP-3: Enter data to the table using SQL commands.
STEP-4: Create cursor using PL/SQL.
STEP-5: Close the procedure.
STEP-6: Close the database.

Implicit Cursor

Create table

```
create table customers(id number(2),name varchar2(20),age number(20),Address varchar2(20),Salary number(10));
```

Table created.

Insert values into table

```
insert into customers values(1,'Ramesh',23,'Allahabad',20000);
1 row inserted
insert into customers values(2,'suresh',22,'kanpur',22000);
1 row inserted
insert into customers values(3,'Mahesh',24,'Ghaziabad',24000);
1 row inserted
insert into customers values(4,'Chandam',25,'Noida',26000);
1 row inserted
insert into customers values(5,'Alex',21,'Paris',28000);
1 row inserted
insert into customers values(6,'Sunita',20,'Delhi',30000);
1 row inserted

select * from customers;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
3	Mahesh	24	Ghaziabad	24000
4	Chandam	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Create Procedure

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  customers
  SET salary = salary + 5000;
  IF sql%notfound THEN
    dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers updated ');
  END IF;
END;
/
```

Output:

```
5 customers updated
PL/SQL procedure successfully completed.
```

select * from customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	25000
3	Mahesh	24	Ghaziabad	29000
4	Chandam	25	Noida	31000
5	Alex	21	Paris	33000
6	Sunita	20	Delhi	35000

Explicit Cursors

```
DECLARE
  c_id customers.id%type;
  c_name customers.name%type;
  c_addr customers.address%type;
  CURSOR c_customers is
    SELECT id, name, address FROM customers;
BEGIN
  OPEN c_customers;
  LOOP
    FETCH c_customers into c_id, c_name, c_addr;
    EXIT WHEN c_customers%notfound;
```

```
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
/
```

Output:

```
1 Ramesh Allahabad
3 Mahesh Ghaziabad
4 Chandam Noida
5 Alex Paris
6 Sunita Delhi

Statement processed.
```

INFERENCE:

VIVA QUESTIONS:

1. What is cursor?

2. List the types of cursor.

3. What you mean by fetch?

4. What is Active Data Set?

5. What is the use of Deallocate?

RESULT:

Thus the cursor implementation was executed successfully.

Ex.No.	5	Creation of Procedures
Date		

AIM:

To write PL/SQL programs that executes the concept of procedures.

ALGORITHM:

STEP-1: Create or replace procedure.
STEP-2: Declare variables for procedure.
STEP-3: Declare constants for procedure.
STEP-4: Create PL/SQL sub program body.
STEP-5: Create required exception.
STEP-6: End the procedure

Procedures

Table Creation :

```
create table ititems(itemid number(3),actualprice number(5),ordid number(4), prodid number(4));
```

Table created.

```
insert into ititems values(101,2000,500,201);
1 row inserted
insert into ititems values(102,3000,1600,202);
1 row inserted
insert into ititems values(103,4000,600,202);
1 row inserted
select * from ititems;
```

ITEMID	ACTUALPRICE	ORDID	PRODID
101	2000	500	201
102	3000	1600	202
103	4000	600	202

PROGRAM FOR GENERAL PROCEDURE – SELECTED RECORD'S PRICE IS INCREMENTED BY 500 , EXECUTING THE PROCEDURE CREATED AND DISPLAYING THE UPDATED TABLE

```
create procedure itsum1(identity number, total number) is price number;
null_price exception;
begin
selectactualprice into price from ititems where itemid=identity;
if price is null then
raisenull_price;
else
```

```
updateititems set actualprice=actualprice+total where itemid=identity;
end if;
exception
whennull_price then
dbms_output.put_line('price is null');
end;
/
execitsum(101, 500);
PL/SQL procedure successfully completed.
```

SQL> select * from ititems;

ITEMID	ACTUALPRICE	ORDID	PRODID
101	2500	500	201
102	3000	1600	202
103	4000	600	202

1)PROCEDURE FOR 'IN' PARAMETER – CREATION, EXECUTION

```
SQL> set serveroutput on;
SQL> create procedure yyy (a IN number) is price number;
begin
selectactualprice into price from ititems where itemid=a;
dbms_output.put_line('Actual price is ' || price);
if price is null then
dbms_output.put_line('price is null');
end if;
end;
 /
 Procedure created.
SQL> exec yyy(103);
Actual price is 4000
PL/SQL procedure successfully completed.
```

2)PROCEDURE FOR 'OUT' PARAMETER – CREATION, EXECUTION

```
SQL> set serveroutput on;
SQL> create procedure zzz (a in number, b out number) is identity number;
begin
selectordid into identity from ititems where itemid=a;
if identity<1000 then
b:=100;
end if;
end;
 /
```

Procedure created

```
SQL> declare
a number;
```

```
b number;
begin
zzz(101,b);
dbms_output.put_line('The value of b is '|| b);
end;
/
The value of b is 100
PL/SQL procedure successfully completed.
```

3)PROCEDURE FOR 'INOUT' PARAMETER – CREATION, EXECUTION

```
 SQL> create procedure itit( ainout number) is
begin
a:=a+1;
end;
/
```

Procedure created.

```
SQL> declare
a number:=7;
begin
itit(a);
dbms_output.put_line('The updated value is '||a);
end;
/
The updated value is 8
PL/SQL procedure successfully completed.
```

4)Functions:

CREATE THE TABLE 'ITTRAIN' TO BE USED FOR FUNCTIONS

```
SQL>create table ittrain( tno number(10), tfare number(10));
Table created.
SQL>insert into ittrain values (1001,550);
1 row created.
SQL>insert into ittrain values (1002,600);
 1 row created.

SQL>select * from ittrain;
```

TNO	TFARE
1001	550
1002	600

TO CREATE THE TABLE 'ITEMPLS'

```
SQL> create table itempls (enamevarchar2(10), eid number(5), salary number(10));
Table created.
```

```
SQL> insert into itemplsvalues('xxx',11,10000);
1 row created.
SQL> insert into itemplsvalues('yyy',12,10500);
1 row created.
SQL> insert into itemplsvalues('zzz',13,15500);
 1 row created.
SQL> select * from itempls;
```

ENAME	EID	SALARY
xxx	11	10000
yyy	12	10500
zzz	13	15500

PROGRAM FOR FUNCTION AND IT'S EXECUTION

```
create function aaa (trainnumber number) return number is
trainfunctionittrain.tfare % type;
begin
selecttfare into trainfunction from ittrain where tno=trainnumber;
return(trainfunction);
end;
/
Function created.
SQL> set serveroutput on;
SQL> declare
total number;
begin
total:=aaa (1001);
dbms_output.put_line ('Train fare is Rs. '||total);
end;
 /
```

Output:

```
Train fare is Rs.550
PL/SQL procedure successfully completed
```

INFERENCE:

REAL TIME APPLICATION:
Procedure uses for bill preparation in all service provision sector.

VIVA QUESTIONS:

1. What is procedure?

2. List few keywords in procedure.

3. What is exception?

4. What is replace?

5. What is argument?

RESULT

The PL/SQL programs were executed and their respective outputs were verified.

Ex.No.	6	Creation of database triggers and functions
Date		

AIM:

To study and implement the concepts of triggers and functions.

ALGORITHM:

STEP-1: Create table.
STEP-2: Create or replace trigger.
STEP-3: Raise the error.
STEP-4: End the trigger.
STEP-5: Create statement to start trigger.
STEP-6: Create function.
STEP-7: Declare variables and constants.
STEP-8: Write function statements.
STEP-9: End function.

DEFINITION

A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database. The parts of a trigger are,

Trigger statement: Specifies the DML statements and fires the trigger body. It also specifies the table to which the trigger is associated.

Trigger body or trigger action: It is a PL/SQL block that is executed when the triggering statement is used.

Trigger restriction: Restrictions on the trigger can be achieved

The different uses of triggers are as follows,

- To generate data automatically
- To enforce complex integrity constraints
- To customize complex securing authorizations
- To maintain the replicate table
- To audit data modifications

TYPES OF TRIGGERS

The various types of triggers are as follows,

- Before: It fires the trigger before executing the trigger statement.

- After: It fires the trigger after executing the trigger statement.
- For each row: It specifies that the trigger fires once per row.
- For each statement: This is the default trigger that is invoked. It specifies that the trigger fires once per statement.

VARIABLES USED IN TRIGGERS

- :new
- :old

These two variables retain the new and old values of the column updated in the database. The values in these variables can be used in the database triggers for data manipulation

TRIGGERS - SYNTAX

create or replace trigger triggername [before/after] {DML statements}

on [tablename] [for each row/statement]

begin

---

exception

end;

USER DEFINED ERROR MESSAGE

The package “raise_application_error” is used to issue the user defined error messages

Syntax: raise_application_error(error number,‘error message‘);

The error number can lie between -20000 and -20999.

The error message should be a character string.

TO CREATE A SIMPLE TRIGGER THAT DOES NOT ALLOW INSERT UPDATE AND DELETE OPERATIONS ON THE TABLE

SQL> create trigger ittrigg before insert or update or delete on itempls for each row

2 begin
3 raise_application_error(-20010,'You cannot do manipulation');
4 end;
5
6 /
Trigger created.

```
SQL> insert into itemplsvalues('aaa',14,34000); insert into itempls values('aaa',14,34000)
*

ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
SQL> delete from itempls where ename='xxx';
delete from itempls where ename='xxx'

*

ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
SQL> update itempls set eid=15 where ename='yyy';
updateitempls set eid=15 where ename='yyy'
*
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
TO DROP THE CREATED TRIGGER
SQL> drop trigger ittrigg;
Trigger dropped.
```

TO CREATE A TRIGGER THAT RAISES AN USER DEFINED ERROR MESSAGE AND DOES NOT ALLOW UPDATION AND INSERTION

```
SQL> create trigger ittriggs before insert or update of salary on itempls for each row
2 declare
3       triggsalitempls.salary%type;
4       begin
5       select salary into triggsal from itempls where eid=12;
6       if(:new.salary>triggsal or :new.salary<triggsal) then
7       raise_application_error(-20100,'Salary has not been changed');
8       end if;
9       end;
10 /

Trigger created.

SQL> insert into itempls values ('bbb',16,45000); insert into itempls values ('bbb',16,45000)

*
ERROR at line 1:

ORA-04098: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation
```

SQL> update itempls set eid=18 where ename='zzz';updateitempls set eid=18 where ename='zzz'

*

ERROR at line 1:

ORA-04298: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation
FUNCTIONS – SYNTAX
create or replace function <function name> (argument in datatype,……) return datatype {is,as} variable declaration;
constant declaration; begin
PL/SQL subprogram body; exception
exception PL/SQL block; end;
CREATE THE TABLE 'ITTRAIN' TO BE USED FOR FUNCTIONS

SQL>create table ittrain(tno number(10), tfare number(10));
Table created.
SQL>insert into ittrain values (1001, 550); 1 row created.
SQL>insert into ittrain values (1002, 600); 1 row created.
SQL>select * from ittrain;
TNO TFARE

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

1001 550
1002 600
TO CREATE THE TABLE 'ITEMPLS'
SQL> create table itempls (enamevarchar2(10), eid number(5), salary number(10)); Table created.

SQL> insert into itemplsvalues('xxx',11,10000); 1 row created.
SQL> insert into itemplsvalues('yyy',12,10500); 1 row created.
SQL> insert into itemplsvalues('zzz',13,15500); 1 row created.
SQL> select * from itempls;

ENAME EID SALARY

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

xxx 11 10000

yyy 12 10500

zzz13 15500

PROGRAM FOR FUNCTION AND IT'S EXECUTION

SQL> create function aaa (trainnumber number) return number is

```
2       trainfunctionittrain.tfare % type;
3       begin
4       select tfare into trainfunction from ittrain where tno=trainnumber;
5       return(trainfunction);
6       end;
7       /
```

Function created.

```
SQL> set serveroutput on;
SQL> declare
2       total number;
3       begin
4       total:=aaa (1001);
5       dbms_output.put_line('Train fare is Rs. '||total);
6       end;
7       /
```

Train fare is Rs.550

PL/SQL procedure successfully completed.

FACTORIAL OF A NUMBER USING FUNCTION — PROGRAM AND EXECUTION

```
SQL> create function itfact (a number) return number is

2       fact number:=1;
3       b number;
4       begin
5       b:=a;
6       while b>0
7       loop
8       fact:=fact*b;
9       b:=b-1;
10      end loop;
11      return(fact);
12      end;
13      /
```

Function created.

```
SQL> set serveroutput on;
SQL> declare
2       a number:=7;
3       f number(10);
4       begin
5       f:=itfact(a);
6       dbms_output.put_line(‘The factorial of the given number is’||f);
```

```
7       end;
8       /
```

The factorial of the given number is 5040

PL/SQL procedure successfully completed.

INFERENCE:

VIVA QUESTIONS:

1. What is trigger?

2. List few uses of triggers.

3. What are the variables used in triggers?

4. What is function?

5. What is the use of function?

RESULT

The triggers and functions were created, executed and their respective outputs were verified.

Ex.No.	7	Exception Handling
Date		

AIM:

To write a PL/SQL program to handle exceptions.

ALGORITHM:

STEP-1: Create or replace procedure.
STEP-2: Declare variables for procedure.
STEP-3: Declare constants for procedure.
STEP-4: Create required exception.
STEP-5: Write the error handling statements.
STEP-6: End the procedure.

PROCEDURE/OUTPUT:

PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exception Handling. Using Exception Handling we can test the code and avoid it from exiting abruptly.
When an exception occurs amessages which explains its cause is recieved.
PL/SQL Exception message consists of three parts.
1) Type of Exception
2) An Error Code
3) A message
General Syntax for coding the exception section

```
DECLARE
  Declaration section
BEGIN
  Exception section
EXCEPTION
WHEN ex_name1 THEN
   -Error handling statements
WHEN ex_name2 THEN
   -Error handling statements
WHEN Others THEN
  -Error handling statements
END;
```

Program with user defined exception:

```
SQL> DECLARE
  2  N INTEGER:=&N;
  3  A EXCEPTION;
  4  B EXCEPTION;
  5  BEGIN
  6  IF MOD(N,2)=0 THEN
  7  RAISE A;
  8  ELSE
  9  RAISE B;
 10  END IF;
 11  EXCEPTION
 12  WHEN A THEN
 13  DBMS_OUTPUT.PUT_LINE('THE INPUT IS EVEN.....')
 14  WHEN B THEN
 15  DBMS_OUTPUT.PUT_LINE('THE INPUT IS ODD.....');
 16  END;
 17  /
Enter value for n: 20
old   2: N INTEGER:=&N;
new   2: N INTEGER:=20;
THE INPUT IS EVEN.....

PL/SQL procedure successfully completed.

SQL> /
Enter value for n: 21
old   2: N INTEGER:=&N;
```

```
new   2: N INTEGER:=21;
THE INPUT IS ODD.....

PL/SQL procedure successfully completed.
```

Program with system defined exception:
Divide by zero exception:

```
SQL> DECLARE
  2   L_NUM1 NUMBER;
  3   L_NUM2 NUMBER;
  4
  5  BEGIN
  6   L_NUM1 := 10;
  7   L_NUM2 := 0;
  8   DBMS_OUTPUT.PUT_LINE('RESULT:'||L_NUM1/L_NUM2);
  9
 10  EXCEPTION
 11  WHEN ZERO_DIVIDE THEN
 12   DBMS_OUTPUT.PUT_LINE(SQLCODE);
 13   DBMS_OUTPUT.PUT_LINE(SQLERRM);
 14
 15  END;
 16  /
-1476
ORA-01476: divisor is equal to zero

PL/SQL procedure successfully completed.
```

Handling the Exceptions on 'no data found'

```
SQL> create table employee1 (  id     number, employee_type_id      number,  external_id
varchar2(30),  first_name       varchar2(30),  middle_name      varchar2(30),  last_name
varchar2(30),  name    varchar2(100),   birth_date       date  ,  gender_id      number );
Table created.

SQL> create table gender (
  2   id                      number,
  3   code                      varchar2(30),
  4   description                 varchar2(80),
  5   active_date                 date         default SYSDATE  not null,
  6   inactive_date                date );
Table created.
SQL> insert into gender ( id, code, description ) values ( 1, 'F', 'Female' );
1 row created.
SQL> insert into gender ( id, code, description ) values ( 2, 'M', 'Male' );
1 row created.
SQL> insert into gender ( id, code, description ) values ( 3, 'U', 'Unknown' );
1 row created.
SQL> set serveroutput on size 1000000;
SQL> declare
  2
  3     d_birth_date                    employee1.birth_date%TYPE;
  4     n_gender_id                     employee1.gender_id%TYPE;
  5     n_selected                     number := -1;
  6     n_id                          employee1.id%TYPE;
```

```
 7    v_first_name                  employee1.first_name%TYPE;
 8    v_last_name                   employee1.last_name%TYPE;
 9    v_middle_name                    employee1.middle_name%TYPE;
10    v_name                       employee1.name%TYPE;
11
12    begin
13      v_first_name  := 'JOHN';
14     v_middle_name := 'J.';
15      v_last_name   := 'DOUGH';
16      v_name        := rtrim(v_last_name||', '||v_first_name||' '||v_middle_name);
17      d_birth_date  := to_date('19800101', 'YYYYMMDD');
18
19      begin
20        select id into n_gender_id from gender where code = 'M';
21      exception
22        when OTHERS then
23          raise_application_error(-20001, SQLERRM||' on select gender');
24      end;
25
26      begin
27        select id
28       into   n_id
29        from   employee1
30        where  name       = v_name
31        and    birth_date = d_birth_date
32        and    gender_id  = n_gender_id;
33
34        n_selected := sql%rowcount;
35      exception
36        when NO_DATA_FOUND then
37          n_selected := sql%rowcount;
38          DBMS_OUTPUT.PUT_LINE('Caught raised exception NO_DATA_FOUND');
39        when OTHERS then
40          raise_application_error(-20002, SQLERRM||' on select employee');
41      end;
42
43      DBMS_OUTPUT.PUT_LINE(to_char(n_selected)||' row(s) selected.');
44    end;
45  /
Caught raised exception NO_DATA_FOUND
0 row(s) selected.
PL/SQL procedure successfully completed.
```

INFERENCE:

VIVA QUESTIONS:

1. What is procedure?

2. List few keywords in procedure.

3. What is exception?

4. What is replace?

5. What is argument?

RESULT

Thus the PL/SQL program that handles exception has been implemented and output was verified.

Ex.No.	8	Database Design Using E-R Model and Normalization
Date		

AIM:

. To design a database table using E-R Model.

ALGORITHM:

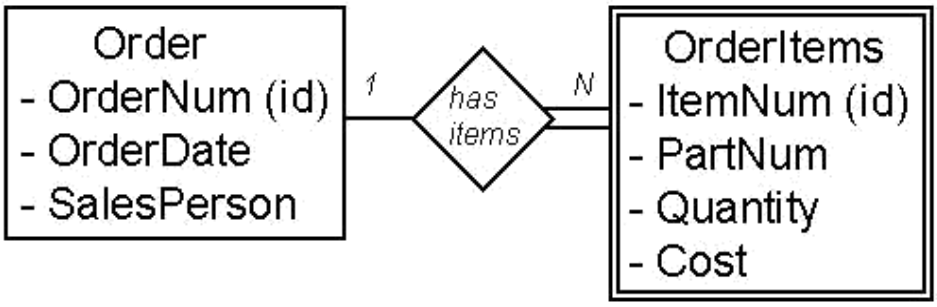
STEP-1: Draw the E-R notation oftable1.

STEP-2: Draw the E-R notation oftable2.

STEP-3: Represent the relationship between the entities.

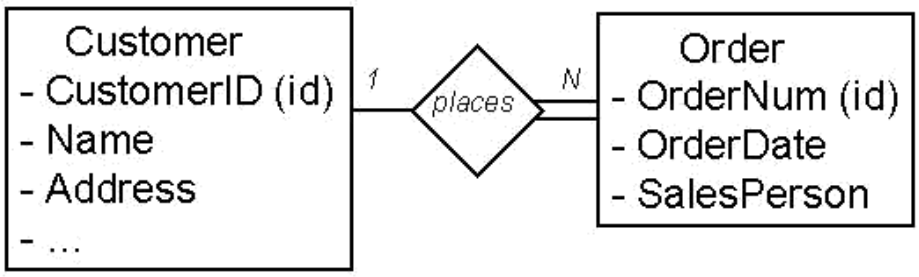
STEP-4: End the procedure.

DIAGRAM:



- **ORDER** (OrderNum (key), OrderDate, SalesPerson)
  ORDERITEMS (OrderNum (key)(fk) , ItemNum (key), PartNum, Quantity, Cost)
  - In the above example, in the ORDERITEMS Relation: OrderNum is the *Foreign Key* and OrderNum plus ItemNum is the *Composite Key*.

Chen Notation



In the ORDER Relation: OrderNum is the *Key*.

Representing Relationships

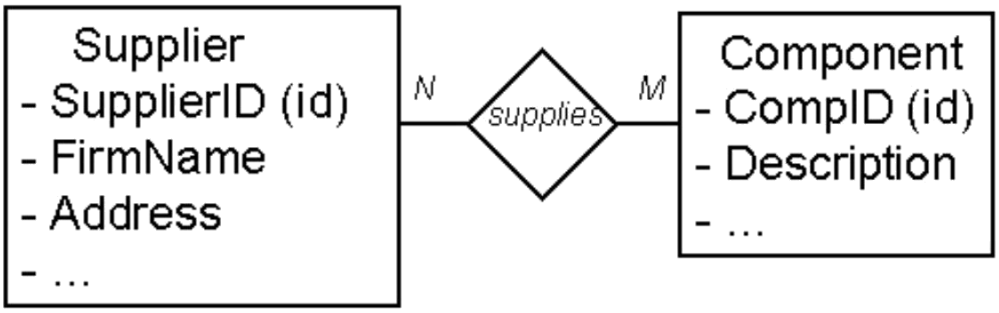
1:1 Relationships. The key of one relation is stored in the second relation. Look at example queries to determine which key is queried most often.

- **1:N** Relationships.
  Parent - Relation on the "1" side.
  Child - Relation on the "Many" side.
- Represent each Entity as a relation.
  Copy the key of the parent into the child relation.
- **CUSTOMER (CustomerID (key), Name, Address, ...)**
  ORDER (OrderNum (key), OrderDate, SalesPerson, CustomerID (fk))

M:N Relationships. Many to Many relationships can not be directly implemented i relations.

- Solution: Introduce a third *Intersection relation* and copy keys from original two relations.

Chen Notation



- SUPPLIER (SupplierID (key), FirmName, Address, ...)
  COMPONENT (CompID (key), Description, ...)
  SUPPLIER_COMPONENT (SupplierID (key), CompID (key))
- Note that this can also be shown in the ER diagram. Also, look for potential added attributes in the intersection relation.

INFERENCE:

VIVA QUESTIONS:

1. What is E-Rmodel?

2. What iskey?

3. What 1:1relation?

4. What is 1:Nrelation?

5. What is M:Nrelation?

RESULT:

Thus the design of a database table using E-R Model was implemented successfully.

Ex.No.	09	Database Design And Implementation for Payroll Processing
Date		

AIM:

To design and implement a database for payroll processing.

ALGORITHM:

STEP-1: Create the pay roll processing database.

STEP-2: Establish ODBC connections.

STEP-3: In the administrator tools open data source ODBC.

STEP-4: Write appropriate program.

STEP-5: End the procedure.

STEPS:

1. Create a database for payroll processing which request the using SQL

2. Establish ODBC connection

3. In the administrator tools open data source ODBC

4. Click add button and select oracle in ORA home 90, click finish

5. A window will appear given the data source home as oracle and select TNS source name as lion and give the used id as SWTT

6. ADODC CONTROL FOR SALARY FORM:-

7. The above procedure must be follow except the table , A select the table as salary

8. Write appropriate Program in form each from created in VB from each from created in VB form project.

SQL>create table emp(eno number primary key,enamr varchar(20),age number,addr varchar(20),DOB date,phno number(10));
Table created.

SQL>create table salary(eno number,edesig varchar(10),basic number,da number,hra number,pf number,mc number,met number,foreign key(eno) references emp);
Table created.
TRIGGER to calculate DA,HRA,PF,MC
SQL> create or replace trigger employ
2 after insert on salary
3 declare
4 cursor cur is select eno,basic from salary;
5 begin
6 for cur1 in cur loop

8 hra=basic*0.1,da=basic*0.07,pf=basic*0.05,mc=basic*0.03 where hra=0; 9 end loop;
10 end;
11 / Trigger created.

PROGRAM FOR FORM 1

```
Private Sub emp_Click() Form
2.Show End
Sub Private
Sub exit_Click()
Unload Me
End Sub Private
Sub salary_Click()
Form3.Show
End Sub
```

PROGRAM FOR FORM 2

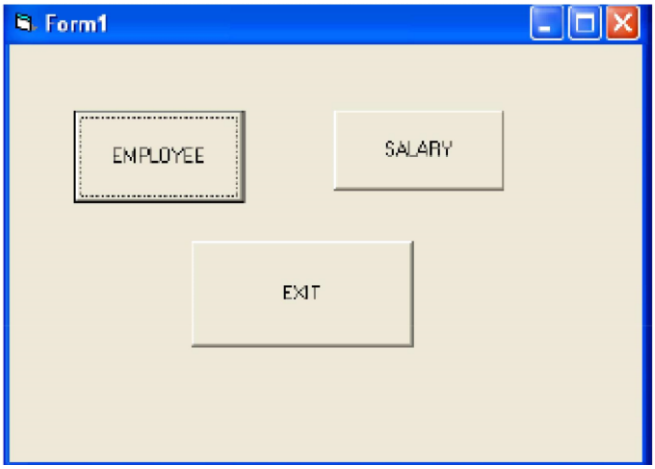
```
Private Sub add_Click()
Adodc1.Recordset.AddNew MsgBox "Record added"

End Sub Private
Sub clear_Click()
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
End Sub Private Sub delte_Click()
Adodc1.Recordset.Delete MsgBox "Record Deleted"
If Adodc1.Recordset.EOF = True
Then Adodc1.Recordset.MovePrevious
End If
End
Sub Private Sub exit_Click()
Unload Me
End Sub
Private Sub main_Click()
Form1.Show
End Sub
Private Sub modify_Click()
Adodc1.Recordset.Update
End Sub
```

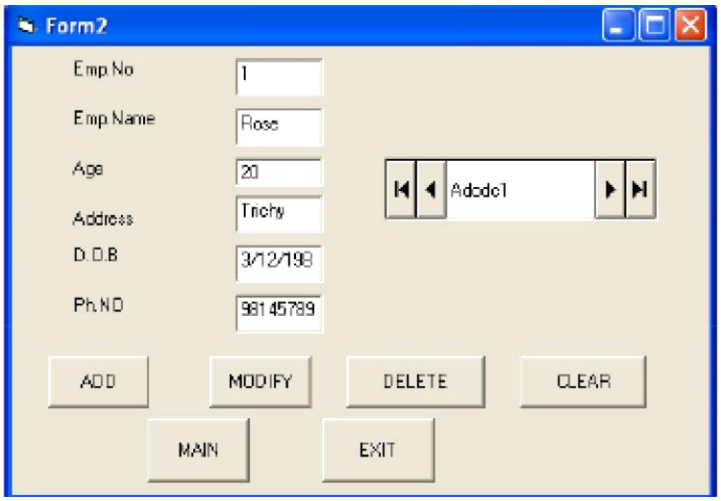
PROGRAM FOR FORM 3

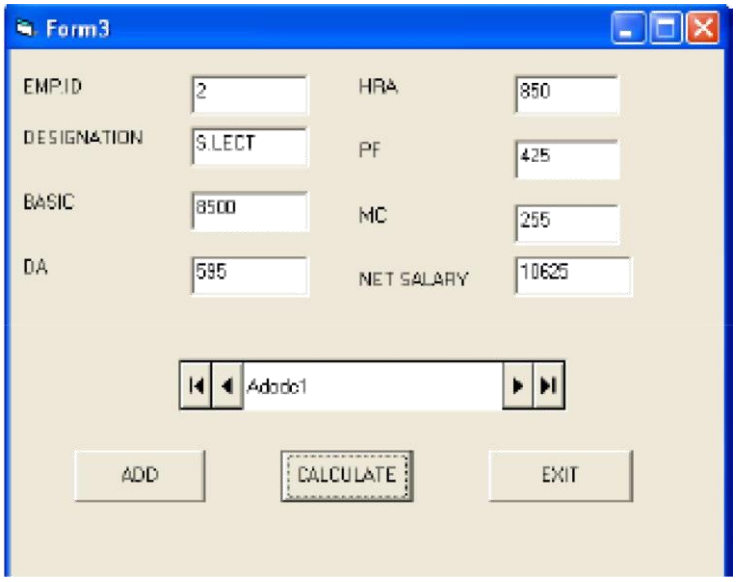
```
Private Sub add_Click()
Adodc1.Recordset.AddNew MsgBox "Record added"
End Sub
Private Sub
clear_Click()
Text1.Text = ""
```

```
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
End Sub
Private Sub delte_Click()
Adodc1.Recordset.Delete MsgBox "Record Deleted"
If Adodc1.Recordset.EOF = True
Then Adodc1.Recordset.MovePrevious
End If
End Sub
Private Sub exit_Click()
Unload Me
End Sub
Private Sub main_Click()
Form1.Show
End Sub
Private Sub
modify_Click()
Adodc1.Recordset.Update
End Sub
```

Output:







RESULT:

Thus payroll system was designed and implemented successfully.

INFERENCE:

VIVA QUESTIONS:

1. What is FrontEnd?

2. What is BackEnd?

3. What is datasource?

4. What is the need of back endconnectivity?

5. What isform?

Ex.No.	10	Case Study using real life database applications
Date		

AIM:

To study about Database system for Library Management system.

Project Overview Statement

As You Know that a Library is collection of books in any institute .Librarian resposibilty is to manage all the records of books issued and also returned on Manualy.

Case study

All the Transaction(books issues & books returned) are manualy

recorded(registars.) Students search books by racks it so time consuming

And there is noarrangement.

Also threat of losingrecorde.

Project Aim and Objective

The project aim and objective are:

To eliminate the paper –work in library -to record every transaction in computerized system

so that problem such as record file missing won't happen again

Backgroud of Project

Library Management system is an application refer to other library system and is suitable to use by small and medium size libray .

It is use by librarian and libray admin to manage the library using a computerized system.

The system was designed to help librain record every book transcation so that the problem such as file missing will not happened again**.**

Design view

The library has the following tables in its database;

1. Books (book_id, ISBN,bookName, BOOKAUTHOR andbookedition)
2. student (student_id, studentname, student_email,student_address)

3. Staff (staff_id, staff_name, staff_address, staff_gender,staff_phone)

4. department (department_id,branch_name)

5. Issue (issue_id, issue_date, expiry_date, book_name,book_id)

6. Return (return_id, expiry_date, isuue_date,book_id)

NORAMALIZATION OF TABLE

Why Normalization:

Database normalization is the process of removing redundant data from your tables in order to improve storage efficiency, data integrity, and scalability.
Normalization generally involves splitting existing tables into multiple ones, which must be re-joinedorlinkedeachtimeaqueryisissued.

Given table is convertedto its 1NF as follows.

• STEP NUMBER1:

elimination of duplicative columns from table 1.

• Step number2:

create separate table for each group of related data and identify each row with unique column (primary key).

2nd normalform

A table is in first normal form and each non-key field is functionally dependent upon primary key.
Now we'll take the table above and design new tables that will eliminate the repeted date in non key _field

To decide what fields belong together in a table, think about which field determines the values in other fields.

Create a table for those fields and enter the sample data.

- Think about what the primary key for each table would be and about the relationship between the tables.
- Mark the primary key for each table and make sure that you do not have repeated data in non-key fields.

- Third normal form (3NF) requires that there are no functional dependencies of non-key attributes on something other than a candidate key.
- A table is in 3NF if all of the non primary-key attributes are mutually independent
- There should not be transitive depeinces.

Normalization of Tables in Database

ISSue_id	Book_id	Student_id
1122	110,120,320	bitE183

In the **ISSUE Table**there is repatingbook_id . A student has issued 3 books.

After first Normalization

ISSUE_ID	book_id	Student_id
1122	**110**	**bitE183**
1122	**320**	**bitE183**
1122	**120**	**bitE183**

Second normalized Form:

In the following Student relation all attributes are dependent on the primary key StudID

Student_id	Name	Depid	Issue_date	Expairy_datePhone		
BITf13E183	**Azhar**	**20**	**17-6-15**	**1-7-15**		**3127400**

We can create two other relations from Student Table one is Department fields are fully dependent on the primary keys DEp_id

DEp_id	Dep_name
11	CS & ITDepartment
22	**EducationDepartment**
33	**EconomicsDepartment**
44	**Laaw Department**

Student_id	Name	Issue_date	Expairy_date	Phone
BITf13E183	Azhar	15-6-15	1-7-15	312-7400558

Before third normal form

Staff_id	Name	Gender	Designation	Address	City	state	cell
1101	Shaid	M	Librarian	House no 12 street 6	Sargodha	punjab	300-1234567

1345	Riaz	m	Data entery	Statilete town	Sargodha	Punjab	0346-1234567
2264	Arshad	m	Naibqaisd	Raza garden	Sargodha	Punjab	0333-1234567

After 3rd Normalization

Staff table

Staff_id **Name** **Gender**

Staff conatact

Staff_id	Address	City	State	Telephone	cell

STUDENT Table before Third normalized Form :

Std_id Name Gender Address City State Phone

Dep_idAfter thirdnormal

Student_id	Dep_id	Department
IT-113	**C-26**	**Cs & IT**
Lm-456	**L-11**	**Law**
Eng-98	**E-41**	**ENGLISH**

Studentcontact table:

Student_id	Address	City	State	Phone
IT-111	Statlitetwon	Sargodha	Punjab	312-1234567
Cs-786	Sahiwal	sargoda	punjab	300-1234567

Student table:

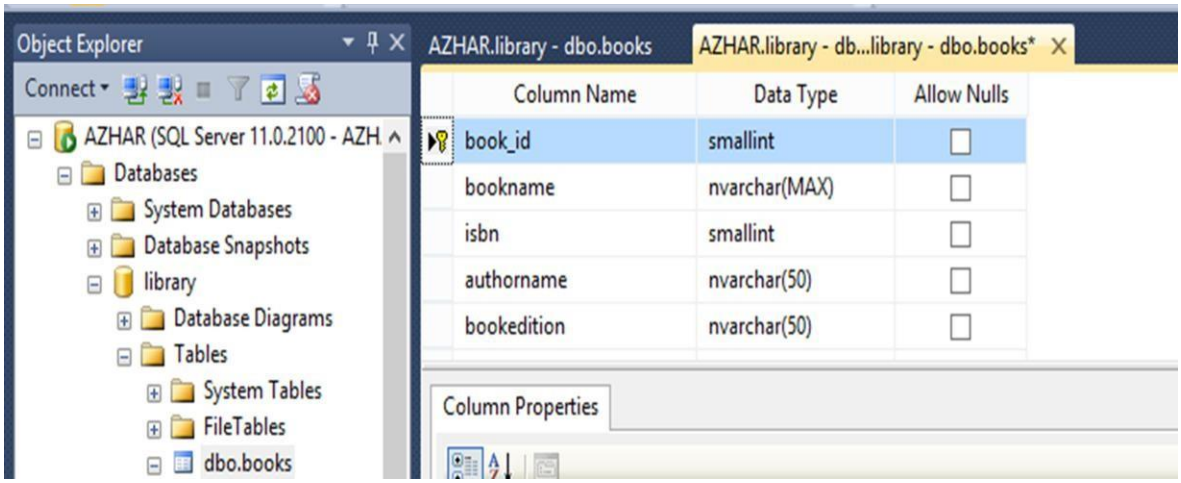
Student_id	Name	Gender	studentDepartment

Normalization End

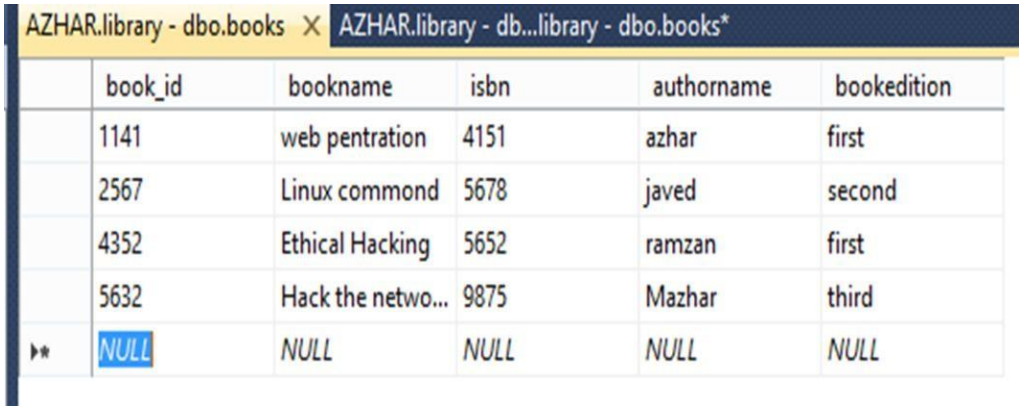
ARCHITECTURE OF TABLES IN SQL SEERVER 2012 AND RECORD FIRST TABLE IS

BOOK

Design view

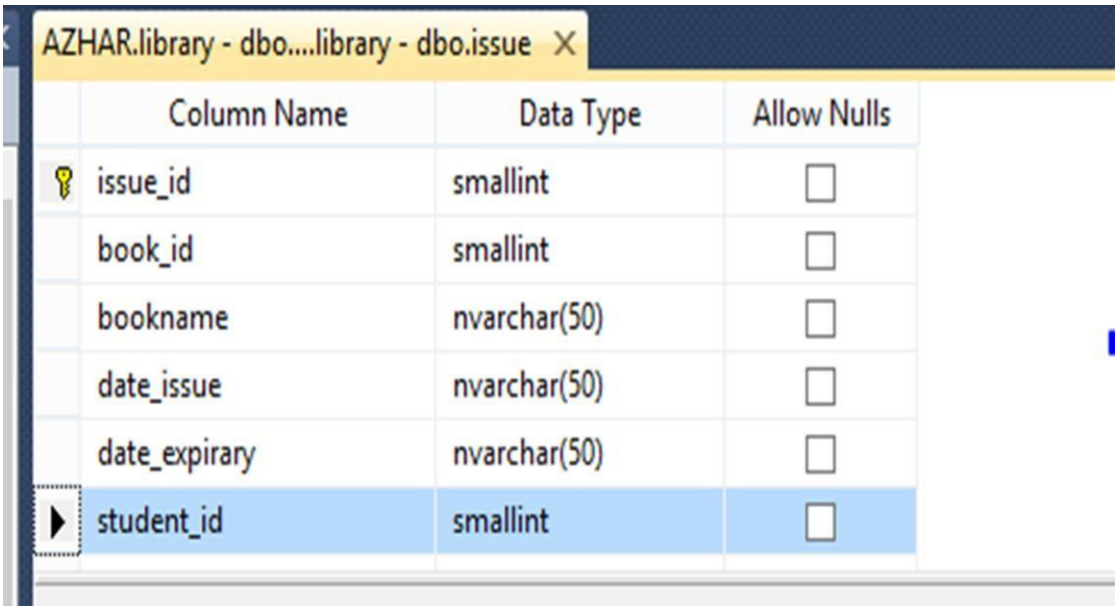


Records



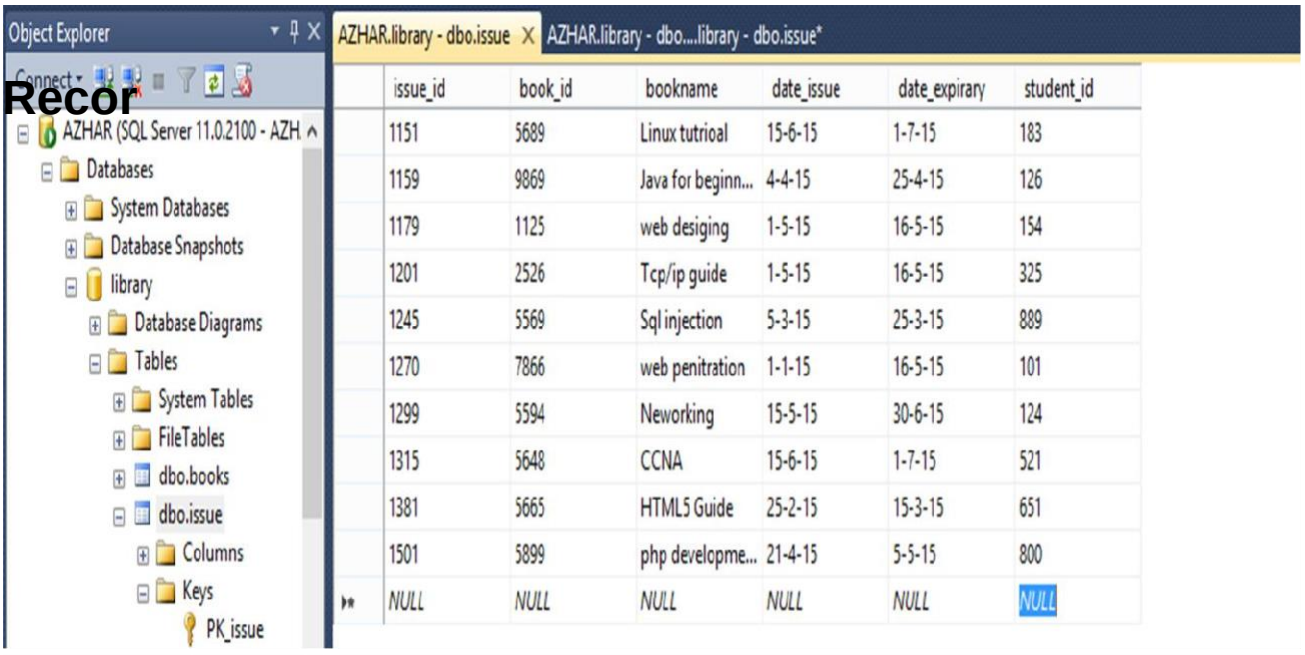
nd
2 tableIssues

Design view



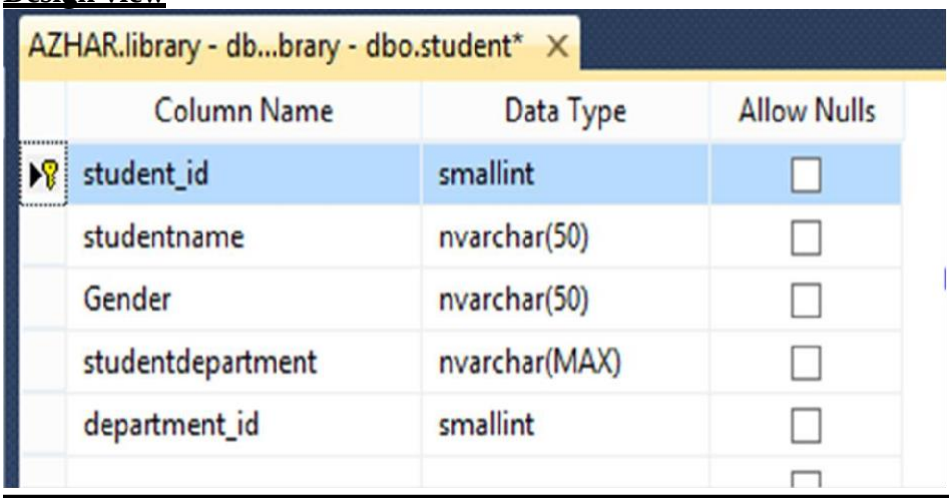
Student _id is forigen key in book table

Recor



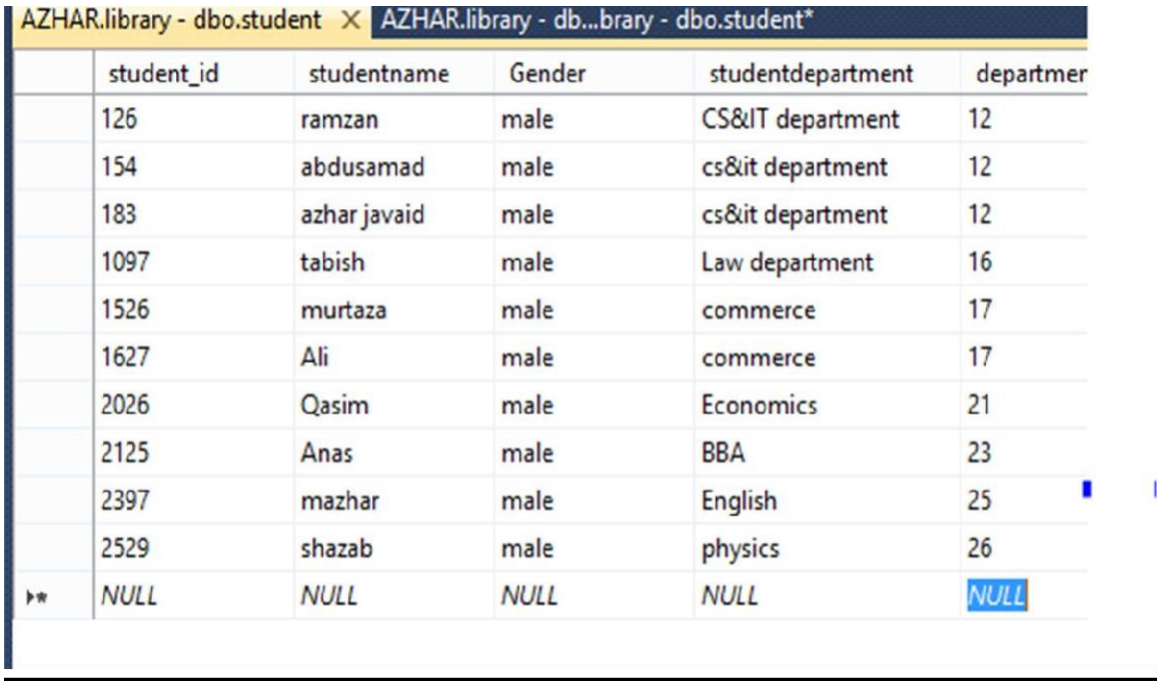
3rd tablestudent

Design view



Dep_id is forign key in student table

Record view



student_id	studentname	Gender	studentdepartment	departmer
126	ramzan	male	CS&IT department	12
154	abdusamad	male	cs&it department	12
183	azhar javaid	male	cs&it department	12
1097	tabish	male	Law department	16
1526	murtaza	male	commerce	17
1627	Ali	male	commerce	17
2026	Qasim	male	Economics	21
2125	Anas	male	BBA	23
2397	mazhar	male	English	25
2529	shazab	male	physics	26
NULL	*NULL*	*NULL*	*NULL*	*NULL*

Student

contactDesign

AZHAR.library - db...bo.student contact*

Column Name	Data Type	Allow Nulls
student_id	smallint	☐
address	nvarchar(50)	☐
city	nvarchar(50)	☐
state	nvarchar(50)	☐
phone	smallint	☐

view

Record

AZHAR.library - dbo.student contact | AZHAR.library - db...bo.student contact

student_id	address	city	state	phone
126	chack 81	sargodha	punjab	3008136255
154	raza town	sargodha	punjab	3075408811
183	behria twon	sargodha	punjab	3127400558
196	block 12	sargodha	punjab	3001234567
224	block 4	sargodha	punjab	31212345678
231	block6	sargodha	punjab	3331234567
236	block 11	sargodha	punjab	3451234567
NULL	*NULL*	*NULL*	*NULL*	*NULL*

Departmenttable - Designview

AZHAR.library - db...y - dbo.department

Column Name	Data Type	Allow Nulls
Dep_id	smallint	☐
Departmentname	nvarchar(50)	☐
student_id	smallint	☐
		☐

Here student_id is forigen key

Recordview

AZHAR.library - dbo.department | AZHAR.library - db...y - dbo.department

Dep_id	Departmentna...	student_id
11	IT Department	183
13	physics	315
14	chemistry	501
17	commerce	751
18	business	901
19	economics	1171
21	food science	1401
22	english	1655
23	math	1831
24	islamic	2101
25	urdu	2405
26	education	2627
NULL	*NULL*	*NULL*

Returntable

AZHAR.library - dbo.Table_1*

Column Name	Data Type	Allow Nulls
return_id	smallint	☐
book_id	smallint	☐
issue_date	date	☐
expairy_date	date	☐
return_date	date	☐
student_id	smallint	☐
staff_id	nvarchar(50)	☐
issue_id	smallint	☐
		☐

Here book_id,issue_id, staff_id ,student_id are forigen keys

Record view

AZHAR.library - dbo.return | AZHAR.library - dbo...ibrary - dbo.return

return_id	book_id	issue_date	expairy_date	return_date	student_id	staff_id	issue_id
5645	456	2015-01-03	2015-03-10	2015-09-03	115	sf12	2156
5699	1122	2015-01-05	2015-05-01	2015-07-04	526	sf15	556
5756	2654	2015-01-06	2015-01-15	2015-01-15	556	sf13	556
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

AZHAR.library - dbo.....library - dbo.staff*

Column Name	Data Type	Allow Nulls
staff_id	nvarchar(50)	☐
name	nchar(10)	☐
gender	nchar(10)	☐
Deg_id	nvarchar(50)	☐
		☐

Staff tableDesign

view

Deg_id is forigen key

Record view

AZHAR.library - dbo.staff | SQLQuery4.sql - A...hammad Azhar (52)) | AZHAR.library - dbo.....library - dbo.sta

staff_id	name	gender	Deg_id
sf12	arshad	male	dg56
sf15	latif	male	dg60
sf16	irfan	male	dg65
sf17	faisal	male	dg66
sf18	roqia	femal	dg69
sf19	saira	female	dg70
sf20	muazamel	male	dg71
sf21	furqan	male	dg72
NULL	NULL	NULL	NULL

Staff deginations table

Design view

AZHAR.library - dbo.designation | AZHAR.library - db...y - dbo.designation

Column Name	Data Type	Allow Nulls
Dg_id	nvarchar(50)	☐
designation	nchar(10)	☐
staff_id	nvarchar(50)	☐
		☐

Staff_id isforigen key Recordview

AZHAR.library - dbo.designation | AZHAR.library - db...y - dbo.designation

Dg_id	desiqnation	staff_id
dg3	naibqasid	sf15
dg4	sweeper	sf10
dg5	bookkeeper	sf11
dg11	librairan	sf13
dg12	clerk	sf16
NULL	*NULL*	*NULL*

Staffcontacttable Designview

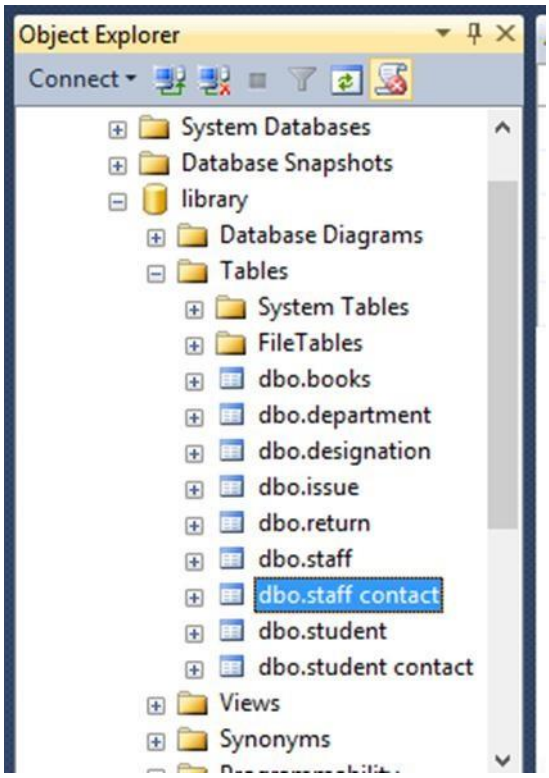
AZHAR.library - dbo...- dbo.staff contact*

Column Name	Data Type	Allow Nulls
staff_id	nvarchar(50)	☐
address	nvarchar(50)	☐
city	nchar(10)	☐
state	nchar(10)	☐
phone	numeric(18, 0)	☑
		☐

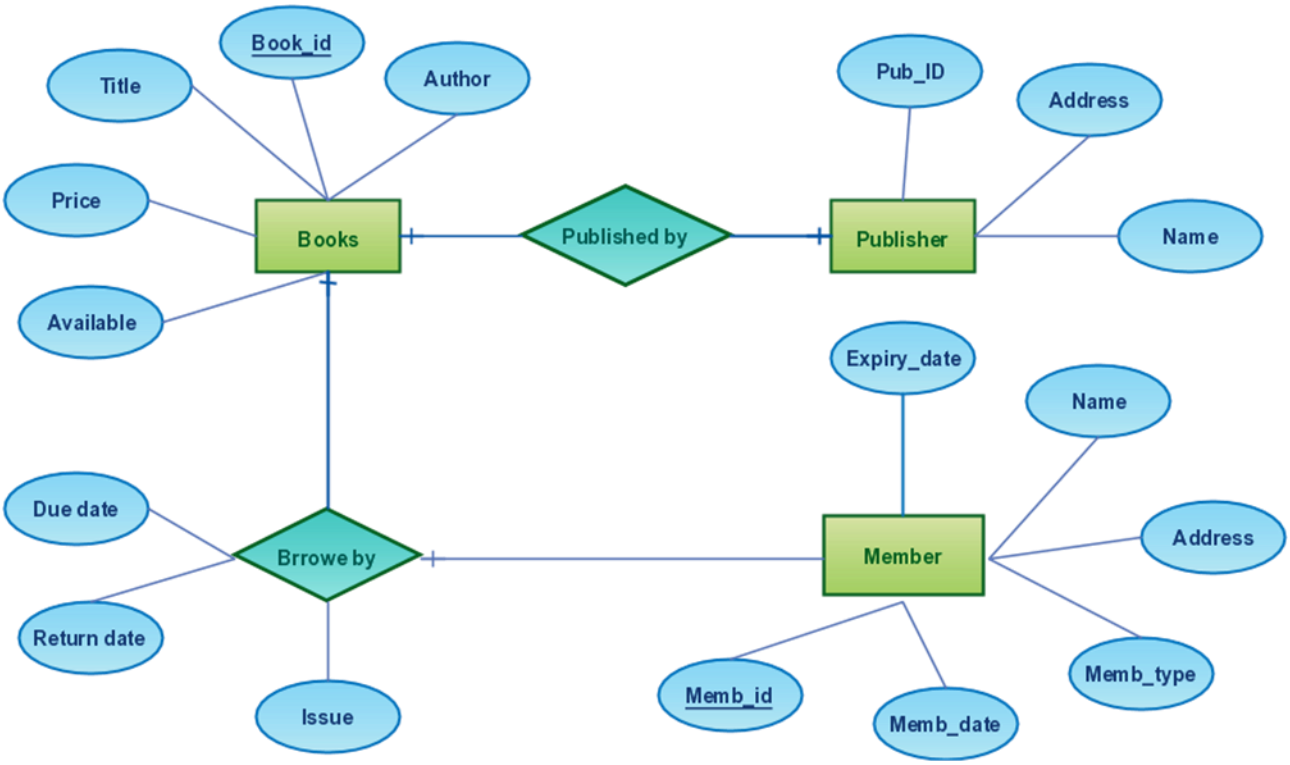
Record view

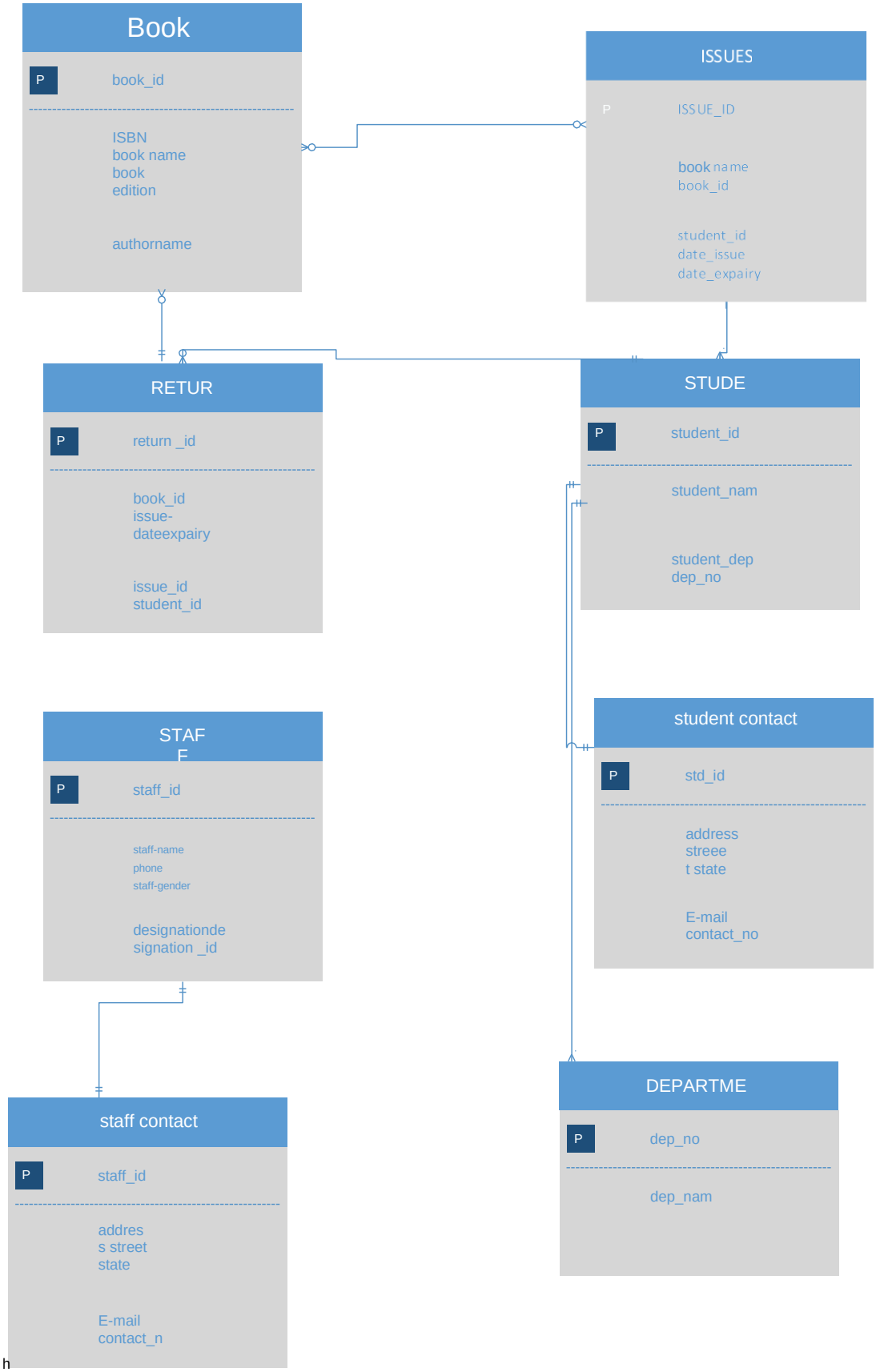
AZHAR.library - dbo.staff contact | AZHAR.library - dbo...- dbo.staff contact*

staff_id	address	city	state	phone
sf12	block 4	sargodha	punjab	3001234567
sf15	main city	sargodha	punjab	3121234567
sf17	green twon	sargodha	punjab	3331234567
sf26	raza twon	sargodha	punjab	3461234567
NULL	*NULL*	*NULL*	*NULL*	*NULL*



E-R Diagram for Library Management System




Book
P
book_id
ISBN
book name
book
edition
authorname
ISSUES
P
ISSUE_ID
book name
book_id
student_id
date_issue
date_expairy
RETUR
P
return _id
book_id
issue-
dateexpairy
issue_id
student_id
STUDE
P
student_id
student_nam
student_dep
dep_no
STAF
F
P
staff_id
staff-name
phone
staff-gender
designationde
signation _id
student contact
P
std_id
address
streee
t state
E-mail
contact_no
staff contact
P
staff_id
addres
s street
state
E-mail
contact_n
DEPARTME
P
dep_no
dep_nam

h

Relational Model of ERD Staff

contact

Staff_id	Address	City	State	Phone

Staff

Staff_id	Name	Desgination	Gender

Designation

Designation_id **Designation**

Studentcontact

Student_id	Address	City		State	phone

Student

Sttudent_id	Name	Gender	Student_dep

Department

Dep_id **Department name**

Book

Book_id	Isbn	Book_name	Edition	Author	Rack_no
				name	

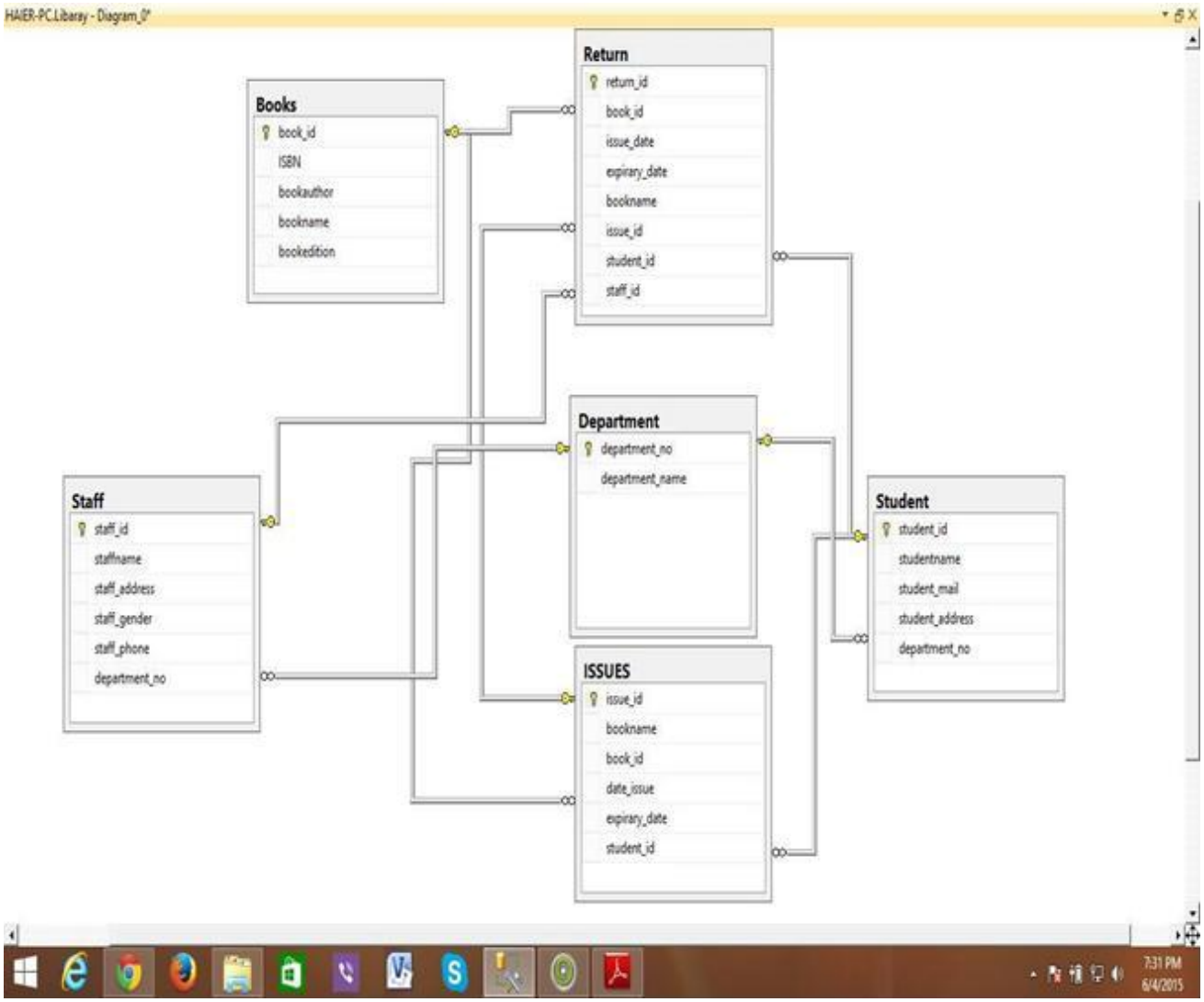
ISSUEs

Issue_id	Book_name	Book_id	Stud_id	Issue_date	Expairy_date

RETURN

Return_id	Book_id	Issues_date	Retuen_date

Entity Relationship Model in SQL SERVER 2012



INFERENCE:

VIVA QUESTIONS:

1. What are the application ofVB.NET?

2. What isform?

3. List few elements offorms.

4. What is componentfield?

5. What iscommand-line-compiler?

RESULT

Thus the case study has been done on library management system.