**ER Diagram**

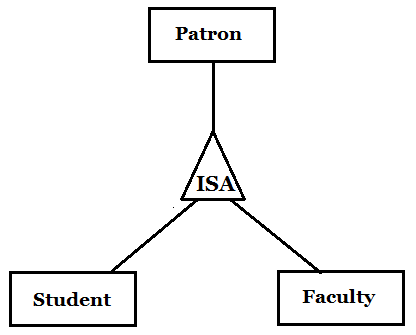
ER Model

The ER diagram shows the relationship between the different entity sets. This document describes the different relationships used, participation constraints, ISA relationships and aggregations used. A description of all the tables and the reason for their use is mentioned in the relational model.

**ISA relationships:**

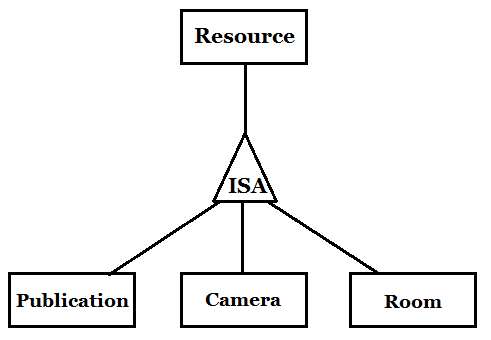
Below is a list of “ISA” relationships we have used:

1. Patron – Student and Faculty



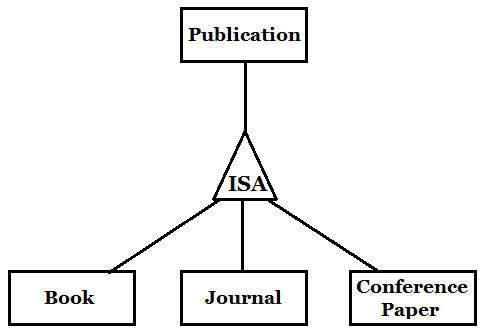
We used the ***ER approach*** to translate the above relationship into tables. Patron table contains the common information and the Student and Faculty table contain information specific to them. This ISA relationship is ***disjoint*** and ***complete***.

1. Resource – Publication, Camera, Room



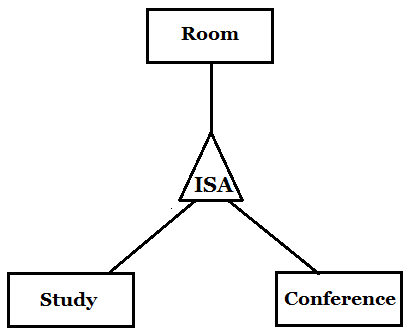
We used the ***ER approach*** to translate the above relationship into tables. Resource table contains a list of the different resources in the system and the individual tables contain information specific to that type of resource. This ISA relationship is ***disjoint*** and ***complete***.

1. Publication – Book, Journal, Conference Paper



We used the ***Object-oriented approach*** to translate the above relationship into tables. Only individual tables with their respective information is created. No common publications table is created. This ISA relationship is ***disjoint*** and ***complete***.

1. Room – Study, Conference



We only created one table for Room and used ***a field in the table*** to specify whether it is a Study or Conference room. This ISA relationship is ***disjoint*** and ***complete***.

**Participation constraints:**

We used three kinds of participation constraints – at most one, at least one and exactly one

***At most one –***

Course (entity) to Teaches (relationship)

A course is taught by at most one faculty

Faculty (entity) to Teaches (relationship)

A faculty teaches at most one course

Reservation (entity) to Estimates Return (relationship)

A reservation estimates return from at most one checkout reference

Reservation (entity) to CalculateLateFee (relationship)

A reservation calculates late fee from at most one late fee reference

Books (entity) to BlockBook (relationship)

A book can be blocked for at most one course by a faculty

***At least one –***

Student (entity) to Enrolled (relationship)

A student must be enrolled in at least one course

***Exactly one –***

Patron (entity) to Belongs to (relationship)

A patron can belong to exactly one department

Student (entity) to Doing (relationship)

A student can be a part of (doing) exactly one program

Course (entity) to Has (relationship)

A course is for exactly one department

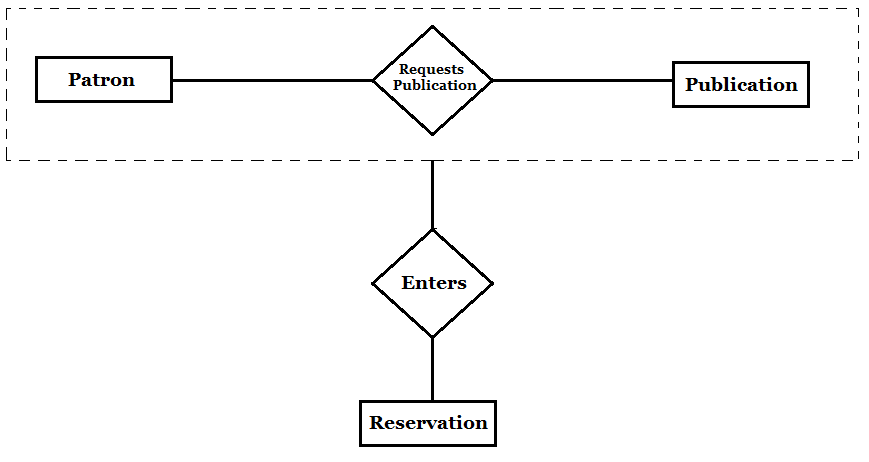
Resource (entity) to Present In (relationship)

A resource is present in exactly one library

**Aggregations:**

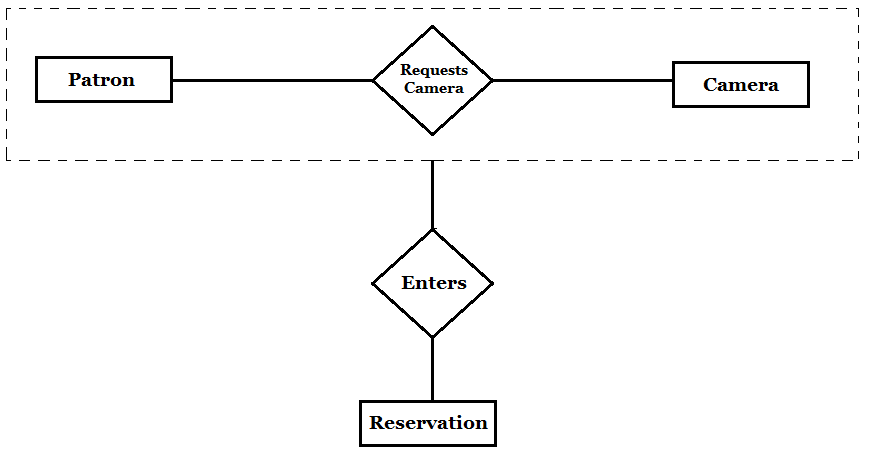
We used four cases of aggregation – all four have a similar logic. In each case, a patron requests a resource and enters a queue for the resource. Once the reservation is confirmed, that request enters the reservation table.

1. Requests Publication – Enters – Reservation



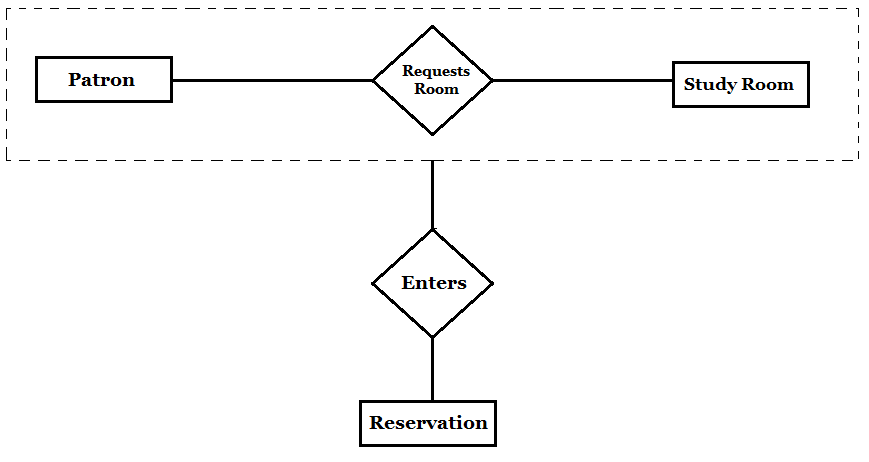
When a patron requests a publication, it enters a queue if that publication has already been checked-out by another patron. Once the reservation is confirmed, the request enters the reservation table.

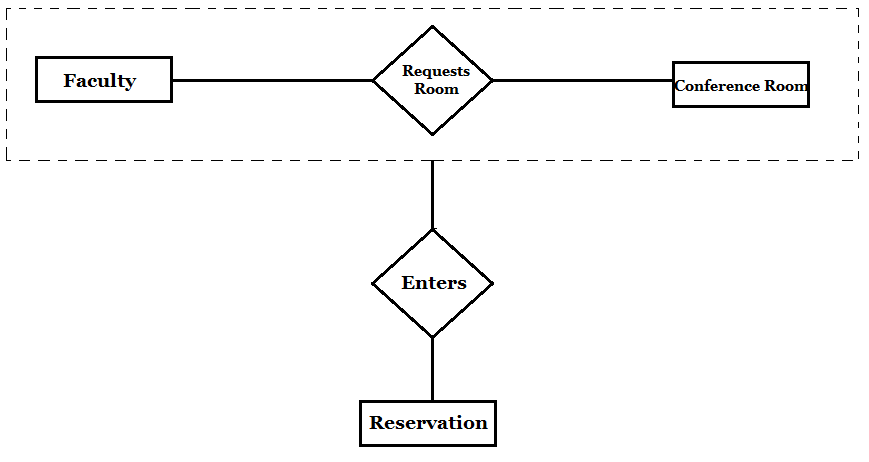
1. Requests Camera – Enters – Reservation



When a patron requests a camera, it enters a queue and the first person in the queue gets the camera at 9am on Friday. When he/she checks out the camera, an entry of the request is made in the reservation table.

1. Requests Room – Enters – Reservation

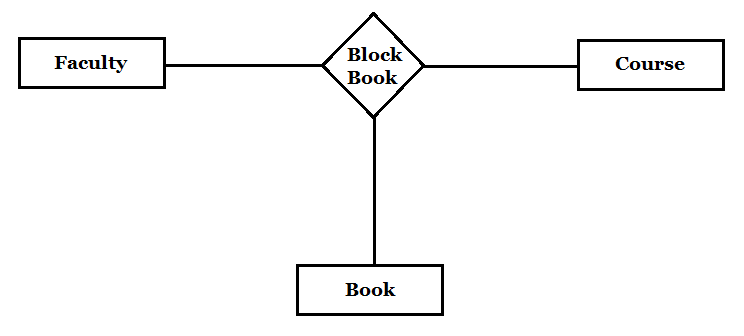




In both the above cases, when a patron requests a room, it enters a separate queue table until the room is checked-out at the time of the reservation. At this point, an entry is made into the reservation table.

**Binary vs Ternary Relationships:**

Most of our relationships are binary. We have used one ternary relationship in our ER diagram between Faculty, Course and Book.



The relationship above shows that a faculty blocks a book for a particular course. This needs to be, and hence has been expressed as a ternary relationship.

**Relational Model**

We tried to maintain 3NF but in some cases 3NF is not satisfied as we knew we would require lots of joins and decided against implementing 3NF.

**1.Patron**- It holds information that is common to both students and faculty including patron id, name of patron, nationality, department id, gender, password.

CONSTRAINTS-

Primary key- patron id,

Foreign key- department\_id (dept\_id) which references dept\_id from the department table

Not null – name , user\_id, password (passwd)

unique - user\_id

SQL> create table Patron(patron\_id varchar(20), name varchar(50) NOT NULL, user\_id varchar(20) UNIQUE NOT NULL, nationality varchar(20), dept\_id integer, gender varchar(10), passwd varchar(20)NOT NULL, privilege integer default 1, Primary Key(patron\_id), Foreign Key(dept\_id) references Department(dept\_id));

**2.Department**- It holds information about each department including department id, name of department.

CONSTRAINTS-

primary key- department\_id

not null - department\_name

SQL> create table Department (dept\_id integer, dept\_name varchar (30) NOT NULL, Primary Key(dept\_id));

**3. Student**- It hold information about each student including student number which is same as patron id, phone number, alternate phone number, date of birth.

CONSTRAINTS-

primary key- student\_num

foreign key- student\_num which references patron\_id from patron, the deletion of student\_num

from patron will result in the corresponding student\_num being deleted from the student table.

not null- phone\_num

SQL> create table Student (student\_num varchar (20), phone\_num varchar(20) NOT NULL, alt\_phn\_num varchar(20), address varchar(50), dob date, Primary Key(student\_num), Foreign Key(student\_num) references Patron(patron\_id) on delete cascade);

**4.Faculty**- It holds information about each faculty including faculty number which is same as patron id, category id which is the type of faculty ( like Professor , Assistant Professor).

CONSTRAINTS –

primary key- faculty\_num

foreign key-

1. faculty\_num which references patron\_id from Patron table,

2. category\_id which references category\_id from FacultyCategory, the deletion of category\_id from FacultyCategory will result in the corresponding record being set to NULL.

SQL> create table Faculty(faculty\_num varchar(20), category\_id integer, Primary Key(faculty\_num), Foreign Key(faculty\_num) references Patron(patron\_id), Foreign Key(category\_id) references FacultyCategory(category\_id) on delete set NULL);

**5. FacultyCategory**- It contains information about the various categories of faculty.

E.g.- Professor, Assistant Professor

CONSTRAINTS

primary key- category\_id

not null- faculty type

SQL> create table FacultyCategory (category\_id integer, faculty\_type varchar (40) NOT NULL, Primary Key(category\_id));

**6. Course**- Contains information about what courses belong to what departments and the books that are reserved for a particular course.

CONSTRAINTS

primary key- course\_id, dept\_id

foreign keys- 1. dept\_id in the department table, which when deleted from the department table

will result in the corresponding record in the course table to be also deleted.,

2.book\_isbn which references isbn from the book table

SQL>

create table Course (course\_id integer, dept\_id integer, book\_isbn Varchar(20), Primary Key(course\_id, dept\_id), Foreign Key(dept\_id) references Department on delete cascade, Foreign Key(book\_isbn) references book(isbn));

**7. Teaches**- It contains information about which faculty teaches which course and the department the faculty belongs to.

CONSTRAINTS

foreign keys-

1. course\_id , dept\_id from the table course,

2. faculty\_num from the faculty table

SQL>create table Teaches (faculty\_num varchar(20), course\_id integer, dept\_id integer, Foreign Key(course\_id, dept\_id) references Course, Foreign Key(faculty\_num) references Faculty);

**8. Program**- Contains information about each program. Degree category has the possible values- undergraduate or graduate. Degree has possible values- MS, BS, Phd.

CONSTRAINTS

primary key- program\_id

not null- degree\_category, degree

SQL> create table Program (program\_id integer, degree\_category varchar (20) NOT NULL, degree varchar (30) NOT NULL, Primary Key(program\_id));

**9. Doing**- Contains information about what program each student is enrolled in ( program id) as well as the year the student is currently in.

CONSTRAINTS

foreign keys-

1.student\_num which references Student,

2.program\_id which references program

not null- prog\_year

SQL > create table Doing (student\_num varchar(20), program\_id integer, prog\_year varchar(30) NOT NULL, Foreign Key(student\_num) references Student, Foreign Key(program\_id) references Program);

**10. Enrolled**- Contains information about what course a student is enrolled in , as well as the department to which the particular course belongs to.

CONSTRAINTS

primary key- student\_num, course\_id, dept\_id

foreign keys-

1.student\_num references student table, deletion of student\_num from the student table will result in the deletion of the corresponding record in the Enrolled table.

2. course\_id, dept\_id which reference Course table , the deletion of course\_id or dept\_id from the Course table will result in the deletion of corresponding record in Enrolled table.

SQL> create table Enrolled (student\_num varchar(20), course\_id integer, dept\_id integer, Primary Key(student\_num, course\_id, dept\_id), Foreign Key(student\_num) references Student on delete cascade, Foreign Key(course\_id, dept\_id) references Course on delete cascade);

**11. Resources** – Contains information about the resources that can be borrowed from the library including books, cameras, rooms .

CONSTRAINTS

primary key- resc\_id

not null- resc\_type

SQL> create table Resources (resc\_id varchar (20), resc\_type varchar(20) NOT NULL, PRIMARY KEY(resc\_id));

**12. Book**- Contains information about books including the isbn , title of the book, name of the author , edition of the book, publisher, year the book was published in and whether or not an e\_copy of the book is available or not.

CONSTRAINTS

primary key- isbn

not null- title

SQL> create table Book(isbn varchar(20), title varchar(50) NOT NULL, author\_name varchar(40), edition integer, publisher varchar(40), year\_of\_pub integer, e\_copy integer default 0, Primary Key(isbn));

**13. Journal**- Contains information about journals including the issn , title of the journal, name of the author , year the journal was published in and whether or not an e\_copy of the journal is available or not.

CONSTRAINTS

primary key- issn

not null – title, author\_name , publ\_year

SQL > create table Journal (issn varchar (20), title varchar(100) NOT NULL, author\_name varchar(40) NOT NULL, publ\_year integer NOT NULL, e\_copy integer default 0, Primary Key(issn));

**14. Conference** ( conf)- Contains information about each conference including the conference number, name of the conference, name of the author, title of conference paper, year of publication and whether or not an e copy for the conference paper is present or not.

CONSTRAINTS

primary key- conf\_num

not null- name\_of\_conf, author\_name, publ\_year

SQL> create table Conf (conf\_num varchar(20), name\_of\_conf varchar(30) NOT NULL,author\_name varchar(40) NOT NULL, title varchar(200), publ\_year integer NOT NULL, e\_copy integer default 0, Primary Key(conf\_num));

**15. Library**

Contains information about each library including library number, name of the library

CONSTRAINTS

primary key- lib\_no

not null- lib\_name

SQL> create table Library (lib\_no integer, lib\_name varchar(30) NOT NULL, Primary Key(lib\_no));

**16. BlockBook**- Contains information about the books that have been blocked by faculty, including a unique blocking id, faculty number of the faculty that has blocked the book, the number of books that have been reserved, the department and course the book belongs to and the date till which the book has been blocked.

CONSTRAINTS

primary key- blocking\_id

foreign keys- 1.faculty\_num which references faculty table , deletion of the faculty\_num from

faculty table will result in the corresponding record to be also deleted from the blockbook table.

2.course\_id, dept\_id references course, deletion of course\_id, dept\_id from the

course table will result in the corresponding record to be also deleted from the blockbook table.

3. isbn references isbn from book table, deletion of isbn from the

book table will result in the corresponding record to be also deleted from the blockbook table.

not null- reserved\_till

SQL> create table BlockBook (blocking\_id integer, faculty\_num varchar(20), course\_id integer, dept\_id integer, isbn varchar(20), num\_resv integer default 1, reserved\_till date NOT NULL, Primary Key(blocking\_id),Foreign Key(faculty\_num) references Faculty on delete cascade, Foreign Key(course\_id, dept\_id) references Course on delete cascade, Foreign Key(isbn) references Book(isbn) on delete cascade);

**17. BookList** – contains information about each book ( not just each isbn) , including whether the book is available to the patrons or not.

primary key- book\_id

foreign key- 1. isbn references isbn from the book table, deletion of the isbn from the

book table will result in the corresponding record to be also deleted from the booklist table.

2. blocking\_id references blocking id from the blockbook table. deletion of the blocking\_id from the blockbook table will result in the corresponding record to be also deleted from the booklist table.

3. lib\_no references lib\_no from the library table

4.book\_id references resc\_id from the resources table.

SQL> create table BookList(book\_id varchar(20), isbn varchar(20), is\_available integer default 1, blocking\_id integer, lib\_no integer, isblocked Number(1) default 0, Primary Key(book\_id), Foreign Key(isbn) references Book(isbn) on delete cascade, Foreign Key(blocking\_id) references BlockBook(blocking\_id), Foreign Key(lib\_no) references Library(lib\_no), Foreign Key(book\_id) references Resources(resc\_id));

**18. JournalList** – contains information about each journal ( not just each issn) , including whether the journal is available to the patrons or not

primary key- journal\_id ,

foreign keys- 1. issn references issn from the journal table , deletion of the issn from the

journal table will result in the corresponding record to be also deleted from the journallist table.

2. lib\_no references lib\_no from the library table

3. journal-id references resc\_id from the resources table

SQL> create table JournalList(journal\_id varchar(20), issn varchar(20), is\_available integer default 1, lib\_no integer, Primary key(journal\_id), Foreign Key(issn) references Journal (issn) on delete cascade, Foreign Key(lib\_no) references Library(lib\_no), Foreign Key(journal\_id) references Resources(resc\_id));

**19. CPList** - contains information about each conference paper , including whether the conference paper is available to the patrons or not

primary key- paper\_id

foreign keys-

1.conf\_num references conf\_num from the Conf table, deletion of the conf\_num from the conf table will result in the corresponding record to be also deleted from the CPList table.

2. lib\_no references lib\_no from library table

3. paper\_id references resc\_id from the resources table.

SQL> create table CPList(paper\_id varchar(20), conf\_num varchar(20), is\_available integer default 1, lib\_no integer, Primary key(paper\_id), Foreign Key(conf\_num) references Conf(conf\_num) on delete cascade, Foreign Key(lib\_no) references Library(lib\_no), Foreign Key(paper\_id) references Resources(resc\_id));

**20. Reservation**– Contains information about reservations made, by whom they have been made , and what resources have been reserved. isactive takes 2 possible values- 0 indicates that the reservation is no longer active, 1 indicates that it is active.

CONSTRAINTS

primary key- res\_id

foreign keys- 1.res\_id references resc\_id from the resources table

2. patron\_id references patron\_id from the patron table, deletion of the patron\_id from the patron table will result in the corresponding record to be also deleted from the Reservations table.

SQL> create table Reservation (res\_id integer, patron\_id varchar(20), resc\_type varchar(30), resc\_id varchar(20), checkout\_date date, checkin\_date date, due\_date date, isactive integer default 1, Primary Key(res\_id), Foreign Key(patron\_id) references Patron(patron\_id) on delete cascade, Foreign Key(resc\_id) references Resources(resc\_id));

**21. Checkout**- Contains information about the maximum possible amount of time that a patron can borrow a resource for.

CONSTRAINTS

primary key- resc\_type, patron\_type, is\_blocked

not null- duration\_hrs

SQL> create table Checkout(resc\_type varchar(20), patron\_type varchar(1),is\_blocked integer, duration\_hrs integer NOT NULL, Primary Key(resc\_type, patron\_type, is\_blocked));

**22. CalcLateFees**- This table shows the late fee for each reservation ( if any). The res\_id indicates for what resource the late fee has to be payed, late\_fee indicates the amount in dollars to be payed, status takes 2 possible values ( 0 indicates patron has not payed the fee and 1 indicates he has).

CONSTRAINTS

primary key- res\_id

foreign key- res\_id references res\_id from the reservations table

not null- late\_fee

SQL> create table CalcLateFee(res\_id integer, late\_fee integer NOT NULL , status integer default 0, Primary Key(res\_id), Foreign Key(res\_id) references Reservation(res\_id));

**23. Camera** – Contains information about each camera including the make, model, camera\_id, lens, memory available.

CONSTRAINTS

primary key- camera\_id

foreign keys- lib\_no references lib\_no from the library table

camera\_id references resc\_id from the resources table.

not null- mek, model, lens, memory\_avail

SQL> create table Camera(camera\_id varchar(20), make varchar(20) NOT NULL, model varchar(20) NOT NULL, Lens varchar(50) NOT NULL, Memory\_avail varchar(10) NOT NULL, lib\_no integer, Primary Key(camera\_id), Foreign Key(lib\_no) references Library(lib\_no), Foreign Key(camera\_id) references Resources(resc\_id));

**24. Room**- Contains information about each room including the room number, type of room, the floor number , capacity of room.

CONSTRAINTS

primary key- room\_no

foreign keys-

1. lib\_no references lib\_no from the library table.

2. room\_no references resc\_id from the resources table.

not null- capacity, room\_type

SQL > create table Room(room\_no varchar(20), room\_type varchar(20) NOT NULL,floorno integer, capacity integer NOT NULL, lib\_no integer, Foreign Key(lib\_no) references Library(lib\_no) on delete cascade, Primary Key(room\_no), Foreign Key(room\_no) references Resources(resc\_id));

**25. Resv\_camera\_q** – Contains information about the patrons that are currently in the queue for a particular camera , as well as their position in the queue.

CONSTRAINTS

primary key- patron\_id, camera\_id, intended\_Friday

foreign key –

1. patron\_id references patron-id from the patron table, deletion of the patron\_id from the

patron table will result in the corresponding record to be also deleted from the resv\_camera\_q table.

2. camera\_id references camera\_id from the camera table , deletion of the camera\_id from the

camera table will result in the corresponding record to be also deleted from the resv\_camera\_q table.

SQL> create table resv\_camera\_q(camera\_id varchar(20), patron\_id varchar(20), intended\_friday date, que\_pos integer, Primary Key(patron\_id, camera\_id, intended\_friday), Foreign Key(patron\_id) references Patron(patron\_id) on delete cascade, Foreign Key(camera\_id) references Camera(camera\_id) on delete cascade);

**26. Resv\_publ\_q** –Contains information about the patrons that are currently in the queue for a particular publication ( books, journals and conference papers) , as well as their position in the queue.

CONSTRAINTS-

primary key- publ\_no, patron\_id

foreign keys-

1. patron\_id references patron\_id from the patron table, deletion of the patron\_id from the

patron table will result in the corresponding record to be also deleted from the resv\_publ\_q table.

2. publ\_no references resc\_id from the resources table.

SQL >create table resv\_publ\_q(publ\_no varchar(20), patron\_id varchar(20), req\_date date, Primary Key(publ\_no, patron\_id), Foreign Key(patron\_id) references Patron(patron\_id) on delete cascade, Foreign Key(publ\_no) references Resources(resc\_id));

**27. Resv\_room**- Contains information about the reservations made on rooms including the room number, intended check out time as start\_time , intended check in time as end\_time. is\_active takes 3 possible values- 0 indicates the reservation is no longer active but has been used, 1 indicates the room is currently occupied, 2 indicates the reservation is not active and has been cancelled as the patron did not turn up.

CONSTRAINTS

primary key- room\_no,start\_time,end\_time,is\_active

foreign keys-

1 .patron\_id references patron\_id from the patron table, deletion of the patron\_id from the

patron table will result in the corresponding record to be also deleted from the resv\_room table.

2. room\_no references room\_no from the room table, deletion of the room\_no from the

room table will result in the corresponding record to be also deleted from the resv\_room table.

not null- start\_time, end\_time

not null- start\_time, end\_time

SQL > create table resv\_room(room\_no varchar(20), patron\_id varchar(20), start\_time date NOT NULL, end\_time date NOT NULL, is\_active Number(1) default 1, Primary Key(room\_no,start\_time,end\_time,is\_active), Foreign Key(patron\_id) references Patron(patron\_id) on delete cascade, Foreign Key(room\_no) references Room(room\_no) on delete cascade);

**28 . The reservation\_seq** is for entering a reservation number into the reservation table

Create sequence reservation\_seq

start with 1 increment by 1;

**29. The blocking\_seq** is for entering blocking\_id into the blockbook table.

CREATE SEQUENCE blocking\_seq

START WITH 1

INCREMENT BY 1

NOCACHE

NOCYCLE;

**30. Notifications** – Contains information about the notifications that are sent to patrons.

primary key- patron\_id, message

foreign key- patron\_id references patron\_id from the patron table, deletion of the patron\_id from the patron table will result in the corresponding record to be also deleted from the notifications table

SQL>

Create table notifications

(Patron\_id Varchar(20),

Message Varchar2(2000),

Foreign Key(patron\_id) references Patron(patron\_id) on delete cascade,

primary key (Patron\_id,Message));

**31. PROCEDURE : Account\_deactivateprocedure**

This procedure checks if a patron has status=1 in the calclate fee table ( i.e the patron has not paid the late fee) and if the current date is 90 days past the due\_date for the particular resource by that particular patron. If yes, it sets the privilege to 0 – which means the account has been deactivated.

SQL>

Create or replace

procedure Account\_deactivate (p\_id IN VARCHAR2,

p\_out OUT VARCHAR2)

is

counter NUMBER;

BEGIN

select count(\*)

into counter

from reservation r

where sysdate - r.due\_date > 90

and r.patron\_id = p\_id

and not exists (select \*

from calclatefee cl

where cl.res\_id = r.res\_id

and cl.status = 1

);

if counter > 0 then

Update patron p

set privilege = 0

where p.patron\_id = p\_id;

p\_out := 'deactivate';

else

p\_out := 'nochange';

end if;

END;

/

**32. PROCEDURE : Roomy**

This Procedure checks if a patron who has reserved a room has checked out or not. If a patron has not checked out an hour after the start time, the reservation is cancelled by setting the is\_active column to 0.

SQL>

CREATE OR REPLACE PROCEDURE roomy

AS

BEGIN

update resv\_room

set is\_active=0

where is\_active=1 AND (24\*(sysdate-start\_time)) >=1;

EXCEPTION

WHEN OTHERS THEN

NULL;

END;

/

**33. PROCEDURE: roomy2**

This procedure sets the isactive flag to 0 in the reservation table ( indicating that reservation is no longer active) when the current time is past the check in time.

CREATE OR REPLACE PROCEDURE roomy2

AS

BEGIN

update reservation set isactive=0 , checkin\_date =due\_date

where isactive=1 AND  resc\_type='Room' AND (24\*(sysdate-due\_date)) >0;

END;

/

**34. PROCEDURE: notify\_camera**

This procedure checks if the camera(s) that have been requested ( i.e present in the resv\_camera\_q ) have been returned or not ( i.e if the isactive is 1 in reservation then the camera has not been returned).

If they have not been returned then the request in the resv\_camera\_q for that camera is cancelled and a message is sent to the concerned patron.

If they have been returned then a message is sent to the 1st patron in the resv\_camera\_q for a particular camera, that it is available for pickup.

SQL>

CREATE OR REPLACE PROCEDURE notify\_camera

AS

cursor pats1 is

SELECT camera\_id,patron\_id,intended\_friday

FROM resv\_camera\_q

WHERE que\_pos=1 AND intended\_friday =(SELECT sysdate from dual)

AND camera\_id IN(SELECT resc\_id

FROM Reservation

WHERE resc\_type= 'Camera' AND isactive=1);

cursor pats2 is

SELECT camera\_id, patron\_id FROM resv\_camera\_q

WHERE que\_pos=1 AND to\_date(intended\_friday) = to\_date(sysdate) -- '13-NOV-15'

AND camera\_id IN(SELECT resc\_id

FROM Reservation

WHERE resc\_type= 'Camera' AND isactive=0);

pats\_id1 resv\_camera\_q.patron\_id%type;

pats\_id2 resv\_camera\_q.patron\_id%type;

cam\_id resv\_camera\_q.camera\_id%type;

cam\_id2 resv\_camera\_q.camera\_id%type;

int\_fri resv\_camera\_q.intended\_friday%type;

BEGIN

OPEN pats1;

LOOP

FETCH pats1 into cam\_id,pats\_id1,int\_fri;

EXIT WHEN pats1%notfound;

dbms\_output.put\_line('Camera ' || cam\_id || 'has not been returned. Reservation cancelled'); /\* message to be sent to pats\_id1 \*/

Insert into notifications

values(pats\_id1, 'Camera ' || cam\_id || 'has not been returned. Reservation cancelled');

DELETE FROM resv\_camera\_q

WHERE camera\_id = cam\_id

AND intended\_friday = int\_fri; /\* deleting records in queue for camera(s) that have not been returned by fri 8am \*/

commit;

END LOOP;

CLOSE pats1;

OPEN pats2;

LOOP

FETCH pats2 into cam\_id2,pats\_id2;

EXIT WHEN pats2%notfound;

dbms\_output.put\_line('Camera' || cam\_id2 || 'available for checkout. please collect it before 10 am');

Insert into notifications

values(pats\_id2, 'Camera' || cam\_id2 || 'available for checkout. please collect it before 10 am');

commit;

END LOOP;

CLOSE pats2;

EXCEPTION

WHEN OTHERS THEN

NULL;

END;

/

**35. PROCEDURE- notify\_camera2**

This procedure runs at Friday 10AM. It checks if a record with que\_pos=1 exists in the resv\_camera\_q ( A record only exists at this time if a patron has not checked out the requested camera).

So the request for the camera is cancelled for that patron and a message is sent to the concerned patron and the record is deleted from the resv\_camera\_q. The queue positions for a request made for that camera are decremented by 1.

SQL>

CREATE OR REPLACE PROCEDURE notify\_camera2

AS

cursor pats3 is

SELECT patron\_id,camera\_id,intended\_friday FROM resv\_camera\_q

WHERE que\_pos=1 AND to\_date(intended\_friday)= to\_date( sysdate); --'13-NOV-15';

pats\_id3 resv\_camera\_q.patron\_id%type;

cam\_id3 resv\_camera\_q.camera\_id%type;

int\_fri3 resv\_camera\_q.intended\_friday%type;

BEGIN

OPEN pats3;

LOOP

FETCH pats3 into pats\_id3,cam\_id3,int\_fri3;

EXIT WHEN pats3%notfound;

dbms\_output.put\_line('You have not checked out the reserved camera '|| cam\_id3 || ' Reservation cancelled');

Insert into notifications

values(pats\_id3,'You have not checked out the reserved camera '|| cam\_id3 || ' Reservation cancelled');

UPDATE resv\_camera\_q

SET que\_pos = que\_pos-1

WHERE camera\_id = cam\_id3

AND intended\_friday=int\_fri3;

DELETE FROM resv\_camera\_q

WHERE camera\_id=cam\_id3 AND que\_pos=0 AND intended\_friday=int\_fri3;

commit;

END LOOP;

CLOSE pats3;

--EXCEPTION

--WHEN OTHERS THEN

--NULL;

END;

/

**36. PROCEDURE: notify\_camera3**

This procedure runs immediately after notify\_camera2. It sends notifications to the patrons whose position in the queue has been updated to 1 as a result of notify\_camera2.

SQL>

CREATE OR REPLACE PROCEDURE notify\_camera3 /\* Runs immediately after procedure notify\_camera2 \*/

AS

cursor pats4 is

SELECT patron\_id,camera\_id,intended\_friday FROM resv\_camera\_q

WHERE que\_pos=1 AND to\_date(intended\_friday)= to\_date( sysdate );

pats\_id4 resv\_camera\_q.patron\_id%type;

cam\_id4 resv\_camera\_q.camera\_id%type;

int\_fri4 resv\_camera\_q.intended\_friday%type;

BEGIN

OPEN pats4;

LOOP

FETCH pats4 into pats\_id4,cam\_id4,int\_fri4;

EXIT WHEN pats4%notfound;

dbms\_output.put\_line('Camera '|| cam\_id4 || ' is available for pickup');

where que\_pos=1, camera\_id= cam\_id4, and intended\_friday = int\_fri4

Insert into notifications

values (pats\_id4,'Camera '|| cam\_id4 || ' is available for pickup');

commit;

END LOOP;

CLOSE pats4;

--EXCEPTION

--WHEN OTHERS THEN

--NULL;

END;

/

**37. PROCEDURE: notify\_books**

This procedure sends notifications (reminders to concerned patrons when they have checked out a book and their reservation is due in 1 or 3 days.

It also sends reminders when a patron has not returned a book and the current date is either 30 ,60 or 90 days past the due date of that particular book.

CREATE OR REPLACE PROCEDURE notify\_books

AS

cursor d\_date is

SELECT due\_date,patron\_id FROM Reservation

WHERE resc\_type='Book';

du\_date Reservation.due\_date%type;

p\_id Reservation.patron\_id%type;

today date;

today1 number;

today2 number;

BEGIN

SELECT sysdate into today from dual;

OPEN d\_date;

LOOP

FETCH d\_date into du\_date,p\_id;

EXIT WHEN d\_date%notfound;

today1 := ceil(du\_date-today) ;

today2 := ceil(today- du\_date) ;

dbms\_output.put\_line( today1 );

IF (today1 = 3 )

THEN dbms\_output.put\_line(' The due date for your book is in 3 days');

Insert into notifications

values(p\_id, ' The due date for your book is in 3 days');

ELSIF ( today1 = 1 )

THEN dbms\_output.put\_line(' The due date for your book is in 1 day');

Insert into notifications

values(p\_id, ' The due date for your book is in 1 day');

ELSIF ( today2 = 30 )

THEN dbms\_output.put\_line(' The borrowed book was due 30 days ago.Please return the borrowed item and pay the late fee');

Insert into notifications

values(p\_id, ' The borrowed book was due 30 days ago.Please return the borrowed item and pay the late fee');

ELSIF ( today2 = 60 )

THEN dbms\_output.put\_line(' The borrowed book was due 60 days ago.Please return the borrowed item and pay the late fee');

Insert into notifications

values(p\_id, ' The borrowed book was due 60 days ago.Please return the borrowed item and pay the late fee');

ELSIF ( today2 = 90 )

THEN dbms\_output.put\_line(' The borrowed book was due 90 days ago.Please return the borrowed item and pay the late fee to avoid suspension of library privileges.');

Insert into notifications

values(p\_id, ' The borrowed book was due 90 days ago.Please return the borrowed item and pay the late fee to avoid suspension of library privileges.');

END IF;

commit;

END LOOP;

CLOSE d\_date;

EXCEPTION

WHEN OTHERS THEN

NULL;

END;

/

**38. PROCEDURE: checkreserve**

This procedure sets the isblocked flag in the booklist table to 0 ( indicating it is no longer blocked by the faculty) when the current date is past the reservation date ( reserved\_till) .

CREATE OR REPLACE PROCEDURE checkreserve

AS

BEGIN

update booklist

set isblocked=0

where book\_id in ( select bo.book\_id from booklist bo, blockbook bl where bo.isblocked=1 and bo.blocking\_id=bl.blocking\_id and to\_date(bl.reserved\_till) <= to\_date(sysdate));

END;

/

**39**. **TRIGGER: CalculateLateFee**

This trigger is used to calculate the late fee for a resource. After a patron has checked in a resource , the trigger checks if the checked in date is after the due date for the resource. If yes, the late fee is calculated based on the type of resource borrowed and whether or not a book has been blocked .

The late fee rates are fetched from the LateFeeReference table.

SQL>

CREATE OR REPLACE TRIGGER CalculateLateFee

AFTER UPDATE

OF Checkin\_date ON Reservation

FOR EACH ROW

WHEN (NEW.Resc\_type IN ('Book','Journal','Conf','Camera') AND NEW.Checkin\_date > NEW.Due\_date)

DECLARE

v\_unit varchar2(30);

n\_apu number;

n\_diff number;

n\_amount number;

BEGIN

-- fetch unit and amount per unit from reference table

SELECT Unit, Amt\_per\_unit INTO v\_unit, n\_apu FROM LateFeeReference WHERE Resc\_type = :NEW.Resc\_type;

IF v\_unit = 'Day' THEN

n\_diff := :NEW.Checkin\_date - :NEW.Due\_date;

--dbms\_out.put\_line(:n\_diff);

END IF;

IF v\_unit = 'Hour' THEN

n\_diff := (:NEW.Checkin\_date - :NEW.Due\_date)\*24 + 1;

--dbms\_out.put\_line(:n\_diff);

END IF;

n\_amount := n\_diff \* n\_apu;

INSERT INTO CalcLateFee VALUES (:NEW.Res\_id, n\_amount, 0);

END;

/

**40. TRIGGER: CalculateDueDate**

This trigger is used to calculate the due date of all resources except rooms. The due date is calculated based on the type of patron( faculty or student) , type of resource , and for a book whether or not it has been blocked by a faculty.

CREATE OR REPLACE TRIGGER CalculateDueDate

BEFORE INSERT

ON Reservation

FOR EACH ROW

WHEN (NEW.Resc\_type IN ('Book','Journal','Conf','Camera'))

DECLARE

duration number;

cam\_date varchar2(20);

p\_type varchar2(3);

BEGIN

SELECT SUBSTR(:NEW.Patron\_id,1,1) INTO p\_type FROM Dual;

--dbms\_output.put\_line(p\_type);

IF :NEW.Resc\_type = 'Book' THEN

SELECT Duration\_Hrs INTO duration FROM CheckOut WHERE Resc\_type = :NEW.Resc\_type AND Patron\_type = p\_type AND Is\_blocked = (SELECT IsBlocked FROM BookList WHERE Book\_id = :NEW.Resc\_id);

--dbms\_output.put\_line(duration);

:NEW.Due\_date := :NEW.Checkout\_date + (duration/24);

ELSIF :NEW.Resc\_type = 'Camera' THEN

SELECT TO\_CHAR(:NEW.Checkout\_date+6,'YYYY/MM/DD')||' 18:00:00' INTO cam\_date FROM DUAL;

--dbms\_output.put\_line(cam\_date);

:NEW.Due\_date := TO\_DATE(cam\_date, 'YYYY/MM/DD HH24:MI:SS');

ELSE

SELECT Duration\_Hrs INTO duration FROM CheckOut WHERE Resc\_type = :NEW.Resc\_type AND Patron\_type = p\_type;

--dbms\_output.put\_line(duration);

:NEW.Due\_date := :NEW.Checkout\_date + (duration/24);

END IF;

END;

/

**Constraints**

1. **Some combinations should clearly not be allowed e.g. an undergraduate in a Ph.D degree program.**

We have stored a list of all possible valid combinations in a table called Program. For any new student, only these combinations will be shown in the drop down while entering the details.

1. **Restricting course reserved materials to only appropriate students**.

We are handling this by showing only the list of books that the student is allowed to request when he goes to the publication module. The data is fetched by joining a few table:

Enrolled, Course, BlockBook, Student

1. **Book may be reserved for a course by a faculty member for a period up till 4 months.** We have created a separated module which is accessible only by faculty in which the faculty can block books for the course he/she teaches. Course table has a list of books assigned to each course, so only those books are shown to the faculty.
2. **Reserved books can be checked out for maximum of 4hrs and by only students of the class for which the book is reserved.**

Firstly the student is able to view only books which are restricted by its enrollment, as mentioned above. The checkout duration is actually calculated by looking up a table called Checkout, which has a list of check out durations in various scenarios. The calculation of due time/date is done by a **trigger** which is triggered each time an entry is made in the reservation table.

1. **Calculation of all due dates for all resources.**

The checkout duration is actually calculated by looking up a table called Checkout, which has a list of check out durations in various scenarios. The calculation of due time/date is done by a **trigger** which is triggered each time an entry is made in the reservation table.

1. **Waitlist for the resources.**

We have created separate tables for each type of resources which maintains the waitlist and clears the waitlist as and when needed.

1. **Faculty always have priority on the wait queue of publications.**

We maintain all patrons in the same waitlist queue, but before clearing any student’s waitlist, we check if there is a faculty in the same queue. If there is a faculty we send the notification to the faculty.

1. **Late Fee**

We have two tables, lateFeeReference and CalcLateFee. lateFeeReference stores the late fee calculation rules for each of the resources. CalcLateFee stores the actual calculated late fee against each reservation. Late fee gets calculated only when a patron returns the resources after comparing the check-in date and the duedate. All these late fee calculations are done using a **trigger**. As soon as there is an update in the reservation table with a checkin date the **trigger** runs and makes an entry in the CalcLateFee table in case there is a late fee applicable.

1. **Due date reminders**

We have created a **procedure** to generate all notifications related to upcoming duedates.

1. **Late fee reminders**

We have created a **procedure** to generate all late fee reminders.

1. **Account deactivation**

We have created a **procedure** to check all the patrons who have not cleared their dues for more than 90 days and this proc itself deactivated their account.

1. **A conference room can be booked only by faculty while a study room can be booked by both student and faculty.**

We do not allow such a request at the interface itself.

1. **If patron is not authorized for room type, an error message is presented.**

We do not allow such a request at the interface itself.

1. **If a room is not “checked out” by 1 hour after reserved start time, the reservation is automatically cancelled.**

We have created a **procedure** for the same.

1. **Camera reservation confirmation**

We have created a **procedure** which runs on every Friday and checks if the requested camera is returned and the first person in the queue gets a notification.

If the camera is not returned, then the reservations for the camera are cancelled by the procedure and the patron at que pos 1 is informed.

1. **If a patron fails to checkout the camera by 10am**.

We have created a **procedure** for the same. If the camera is not checked out by 10am, then the second person in the queue get the notification to collect the camera and the first person gets a notification that his/her reservation is canned.

**SQL Queries and snapshots**

--1. This query returns the patron names (desc order) along with the no. of times they have failed to return a resource on time between 5th oct and 5th nov

select count(r.res\_id), p.name

from reservation r, patron p

where r.due\_date >= '05-OCT-2015'

and r.due\_date <= '05-NOV-2015'

and r.resc\_type <> 'Room'

and r.patron\_id = p.patron\_id

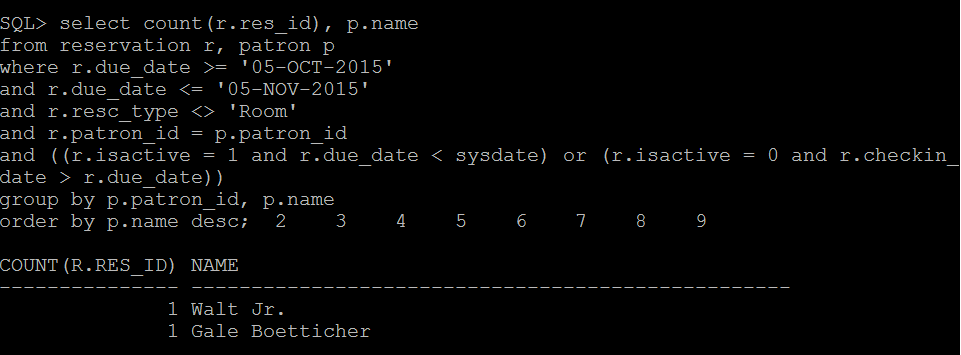
and ((r.isactive = 1 and r.due\_date < sysdate) or (r.isactive = 0 and r.checkin\_date > r.due\_date))

group by p.patron\_id, p.name

order by p.name desc;

--We are taking all those reservations in consideration for which either the resource is not returned till date or for which the resource was returned after due date.

Output:



--2. This gives the count of all rooms which were rendered null and void because the patron did not checkout.

select count(\*)

from resv\_room r

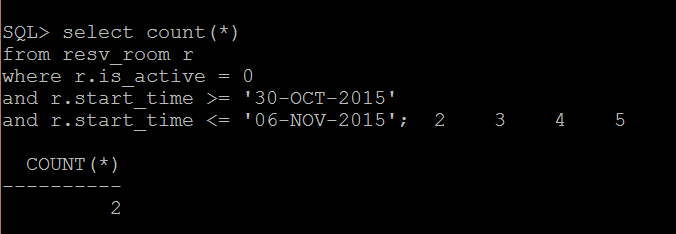
where r.is\_active = 0

and r.start\_time >= '30-OCT-2015'

and r.start\_time <= '06-NOV-2015';

-- Below is the description of how our table "resv\_room" stores the data

-- is\_active = 0 : null void ; is\_active = 1: active reservation waiting for checkout ; is\_active = 2: checked out



--3. This gives the sum of the late fee that was actually collected by the library between 5th oct and 5th Nov

select sum(late\_fee)

from CalcLateFee cl, reservation rsv

where cl.status = 1

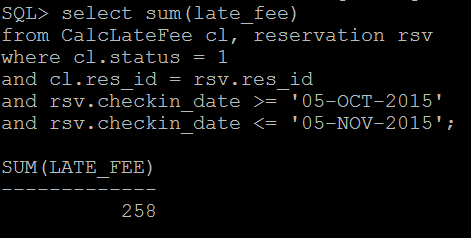
and cl.res\_id = rsv.res\_id

and rsv.checkin\_date >= '05-OCT-2015'

and rsv.checkin\_date <= '05-NOV-2015';

-- Entries get inserted in our "calclatefee" table only if the resource has been returned after the due date. The status column checks if the late fee was paid or unpaid.

Output:



--4. This gives the capacity of the room which is maximum in demand based on the reservations made between 5th oct and 5th nov

select temp.rm

from (select r.capacity rm, count(\*) ct

from resv\_room rsv, room r

where rsv.room\_no = r.room\_no

and rsv.start\_time >= '05-OCT-2015'

and rsv.start\_time <= '05-NOV-2015'

group by r.capacity ) temp

where temp.ct = (select max(temp1.ct) from (select r.capacity rm, count(\*) ct

from resv\_room rsv, room r

where rsv.room\_no = r.room\_no

and rsv.start\_time >= '05-OCT-2015'

and rsv.start\_time <= '05-NOV-2015'

group by r.capacity) temp1) ;

-- room table has a list of all rooms in the library, resv\_table has a list of all requests made to reserve any room.

