Hostel Management System

BASED ON IIT INDORE

Prathamesh Naik | 160001037
Gaurav Naukudkar | 160001040

Table of Contents

Project description

The aim of this project is to create a hostel management system. Specifically, the system has been designed with IIT Indore in mind. Hostel authorities and employees will have access to an intuitive frontend with extensive options to manage tasks related to events, courier, gate records, visitors, hostels, complaints, hostel employees and students. They will be able to retrieve, insert or modify data in a much more ***efficient and safe*** manner compared to the manual method being used currently. Efficiency will be increased due to use of triggers, stored procedures and functions to minimize the info that needs to be entered by the end user. Safety will be guaranteed due to use of constraints and the inherent recovery system of DBIS software.

There is a big emphasis on automation. The central example of this is ***automatic allocation of rooms*** to new students. There is another feature which will be of great help at the end of academic year – ***automatic updating of student year info***. All students who passed will have their year value updated and they will be allocated a room in appropriate hostel id depending on their new year. Students in 4^{th} year will be marked as ex-students and their rooms will be freed.

The system also has a ***login for students***. Students can see their own uncollected couriers, submit new complaints as well as check the status of pending ones, see events taking place in the hostel they're residing in and search for other students to find out their rooms.

Please note that earlier ER analysis was submitted through moodle. However while implementing the project a few changes were made, so I've included the main parts again after modifying appropriately. Thus this version should be considered, not the previous one.

ER Analysis

Entity Sets

Serial no.	Entity set	Type	Attributes	Primary key
1	Student	Strong	roll_no, dob, gender, address, contact_no, year, branch, flat,room	roll_no
2	Visitor	Weak	name, visitor_id, contact_no, purpose, entry_time, exit_time	visitor_id*, roll_no
3	Courier	Weak	courier_id, description, received_date, collected_date	courier_id*, roll_no
4	Gate Record	Weak	purpose, entry_time, exit_time	exit_time*, roll_no
5	Complaint	Weak	complaint_id, description, registered_date, resolved_date	complaint_id*, coll_no
6	Hostel	Strong	name, hostel_id, capacity, no_students, no_rooms_available	hostel_id
7	Employee	Strong	name, employee_id, dob, gender, address, contact_no, date_of_joining, termination_date, designation, salary	employee_id
8	Event	Strong	event_id, description, start_time, expenditure	event_id

*** stands for discriminator.**

NOTES

- Entity set 'Student' has 4 subordinate weak entity sets – 'Visitor', 'Courier', 'Gate record', 'Complaint'.
- 'Courier' has an attribute 'collected_date'. It is null in case of uncollected courier.
- Gate Record has a row with null as exit_time for students outside the campus who haven't returned yet.
- For Visitors still inside the campus, exit_time is null.

Relationship sets

Serial no.	Relationship set	Attributes
1	Resides in	Roll no , Hostel id
2	Visited by	Roll no, Visitor id
3	Receives	Roll no , Courier id
4	Monitoring	Roll no, Exit time
5	Lodges	Roll no, Complaint id
6	Works at	Employee id, Hostel id
7	Conducted in	Event id, Hostel id

NOTES

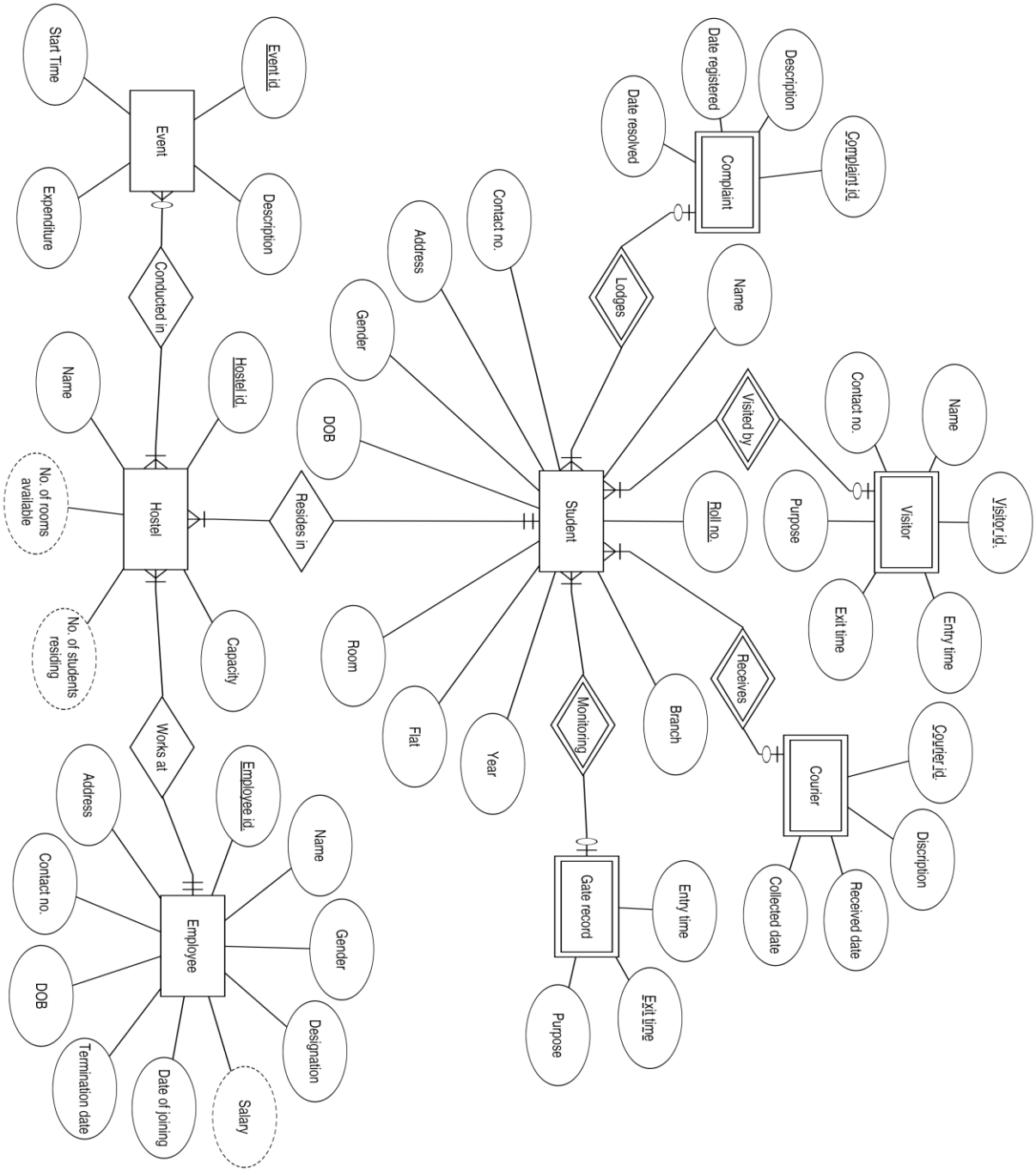
- Entity set 'Student' has 4 subordinate weak entity sets – 'Visitor', 'Courier', 'Administrative record', 'Complaint'.

1. 'Visited by' is the identifying relationship of 'Visitor'
2. 'Receives' is the identifying relationship of 'Courier'
3. 'Monitoring' is the identifying relationship of 'Gate record'
4. 'Lodges' is the identifying relationship of 'Complaint'

Mapping cardinalities and participation constraints

Serial no.	Entity set 1	Relationship set	Entity set 2	Mapping Cardinalities	Participation (in order)
1	Student	Resides in	Hostel	Many to One	Total, Total
2	Student	Visited by	Visitor	One to Many	Partial, Total
3	Student	Receives	Courier	One to Many	Partial, Total
4	Student	Monitoring	Gate record	One to Many	Partial, Total
5	Student	Lodges	Complaint	One to Many	Partial, Total
6	Employee	Works at	Hostel	Many to One	Total, Total
7	Event	Conducted in	Hostel	Many to One	Total, Partial

ER diagram



Please see the separate attached image "er.png" for easier viewing.

Transforming ER Diagram into the set of Tables.

Serial no.	Table	Columns	Primary key
1	Student	roll_no, dob, gender, address, contact_no, year, branch, hostel_id, flat,room	roll_no
2	Visitor	name, visitor_id, contact_no, roll_no, purpose, entry_time, exit_time	visitor_id*, roll_no
3	Courier	courier_id, roll_no, description, received_date, collected_date	courier_id*, roll_no
4	Gate Record	roll_no, purpose, entry_time, exit_time	exit_time*, roll_no
5	Complaint	complaint_id, roll_no, description, registered_date, resolved_date	complaint_id*, coll_no
6	Hostel	name, hostel_id, capacity, no_students, no_rooms_available	hostel_id
7	Employee	name, employee_id, dob, gender, address, contact_no, date_of_joining, termination_date, designation, hostel_id, salary	employee_id
8	Event	event_id, description, start_time, expenditure, hostel_id	event_id

NOTES

The table corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant. Thus 'Visited by', 'Receives', 'Monitoring' and 'Lodges' are not included here.

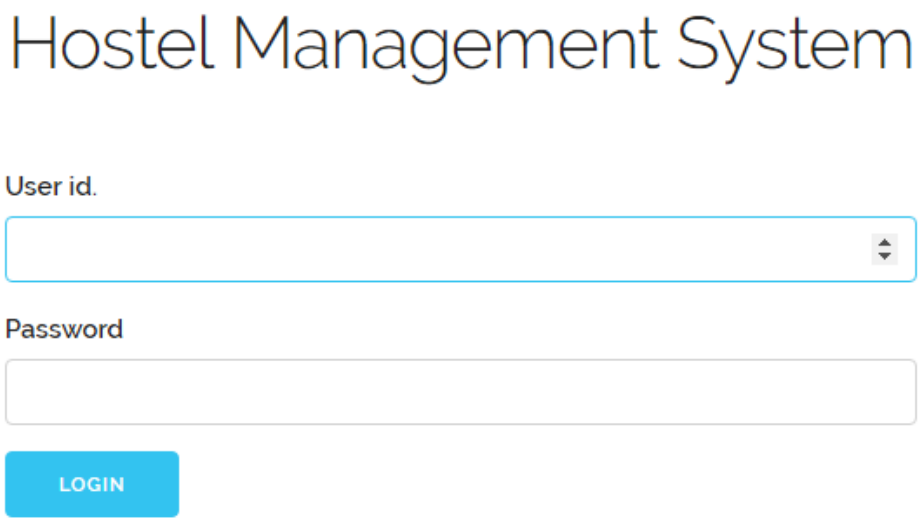
The relationship tables Resides in, Works at and Conducted in were not implemented and instead the relevant information was stored in Student, Employee and Events instead. This was done because it makes implementation of queries, triggers and stored procedure simpler *without adding any redundancy whatsoever.*

(Due to the fact that these 3 relationship sets are many to one and participation is total on the many side. Thus relationship details can be stored in the many side itself, instead of creating a separate table.)

Features and functions

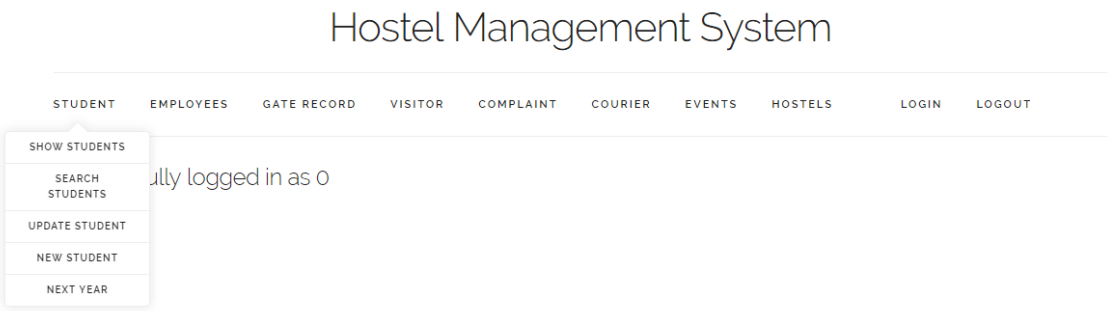
Note: Implementation details are in the next section

Login portal



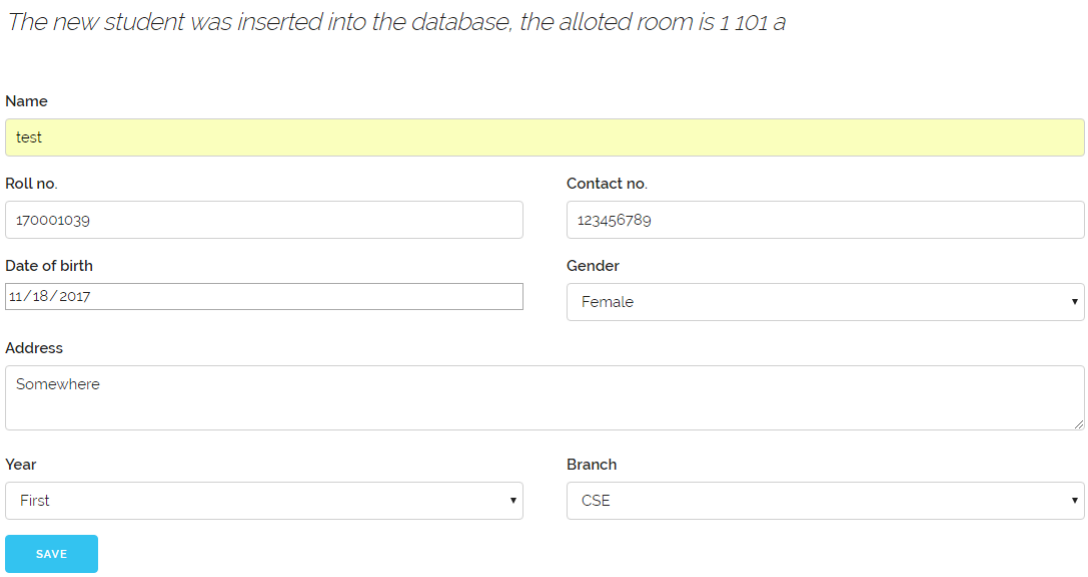
There are two types of users: administrator/hostel authorities and students. The functionality is different for each of them. So first, the user has to login. Currently the user id and password for admin is 0. The user id and password for students is their own roll no. This is only for simplicity and can be easily changed.

Features for hostel authorities



STUDENT

Add new student (*Automatic allocation of room*)



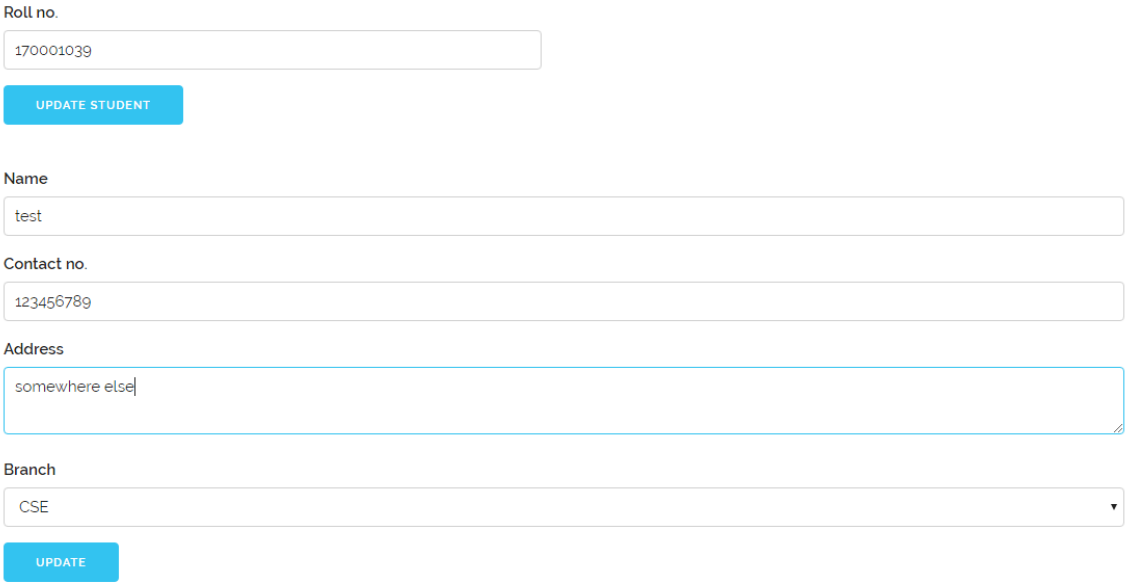
After a new student has been added to the db , the **room Allotment trigger** automatically allocates the next free hostel room from the appropriate hostel id(1 for first year, 2 for second and third year,3 for fourth year) to the student. It's assuming that every floor has 18 rooms.

For eg. First first year student will be allocated 1 101 a next 1 101 b so on till 1 118 e then 1 201 a will come.

Similarly, for fourth year it will be hostel id 3, for second and third it will be hostel id 2.

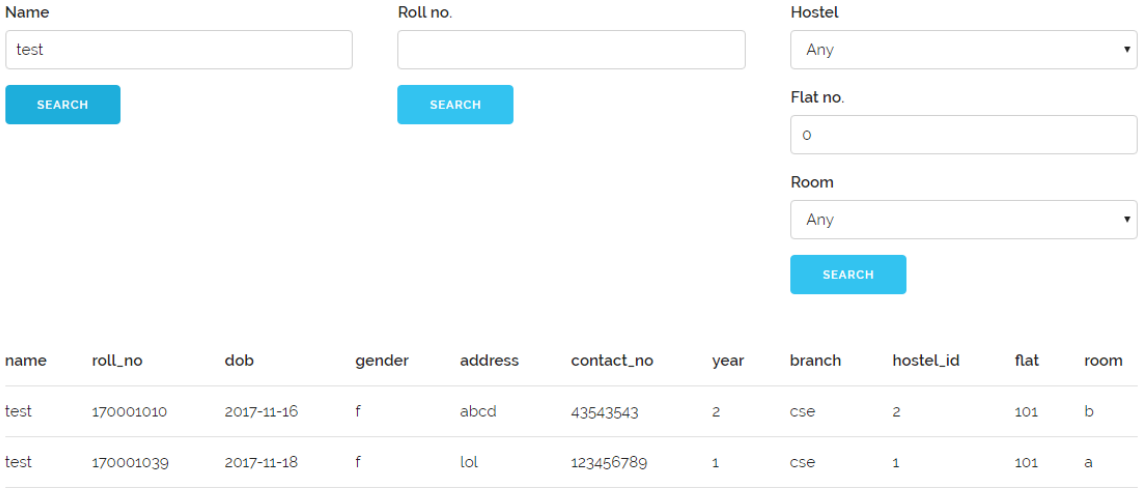
Please see the implementation section for detailed working of this feature.

Update student



First the user has to enter roll no. of student to be updated. Then after clicking update student, a dynamic form is generated containing current values of name, contact no., address and branch. They can be updated as necessary.

Search students



name	roll_no	dob	gender	address	contact_no	year	branch	hostel_id	flat	room
test	170001010	2017-11-16	f	abcd	43543543	2	cse	2	101	b
test	170001039	2017-11-18	f	lol	123456789	1	cse	1	101	a

Students can be searched by three methods:

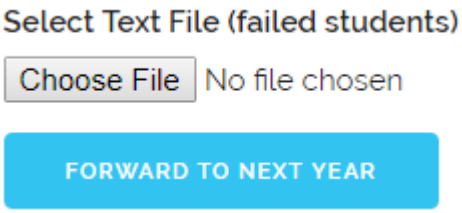
i. Name

ii. Roll no.

iii. Some combination of hostel id, flat no., room id. (There are 5 rooms A to E in one flat)

In the third case, if only hostel id is entered, server will return all students in that hostel. If hostel id. And flat is entered, server will return all students in that hostel in that flat (there will be 5) etc.

Next academic year (*Automatic changing of hostel rooms and year information at the end of academic year*)



At the end of academic year, authorities can submit a file containing list of failed students. Then using a ***Stored procedure and Before update trigger***, year info of all other students (who have passed) is updated and they are allocated new appropriate rooms. Failed students retain their year info and reallocated appropriate room in same hostel id.

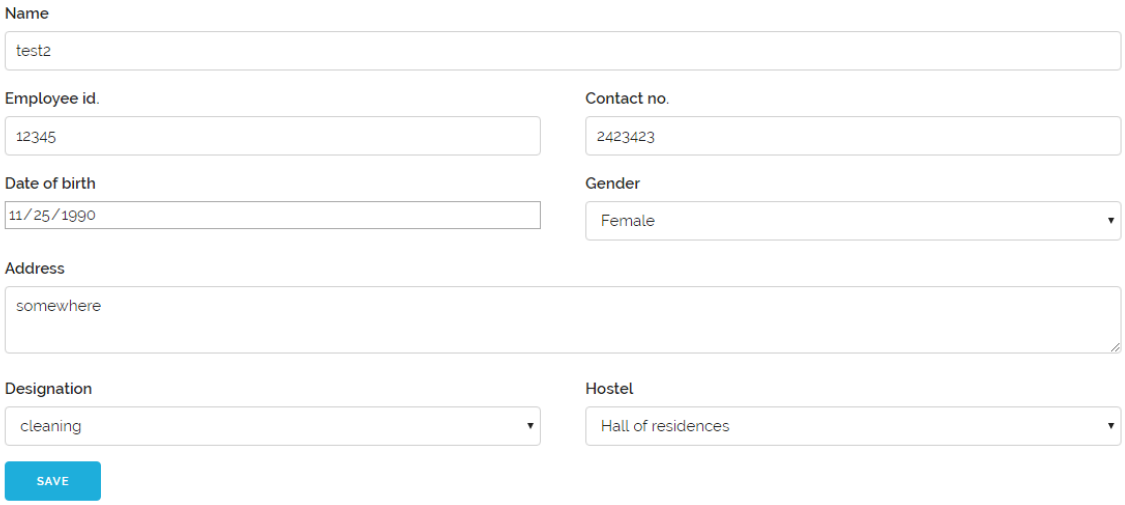
Eg. Student in third year is living in a certain room in hostel id 2. After this process they will be automatically allocated an appropriate room in hostel id 3 and their year will be made 4. Before that all fourth year students who have passed will be made ex-students and their rooms will be freed.

Please see the implementation section for detailed working of this feature.

EMPLOYEE

Add new employee (*Automatic assigning of salary and date of joining*)

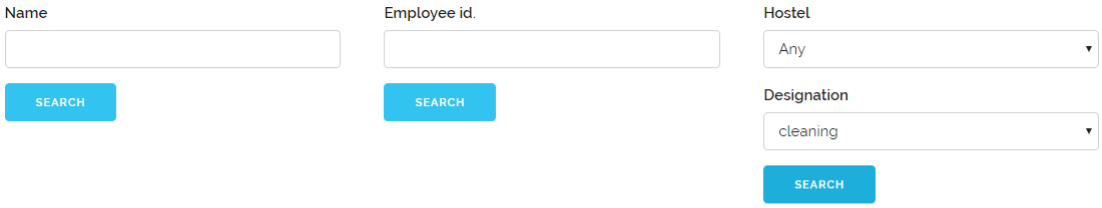
The new employee was inserted into the database, the joining date is 2017-11-18 and salary is 20000



Using a before insert trigger , appropriate salary is set for each new employee added.

Using a before insert trigger , current date is set as date of joining for each new employee added.

Search Employees



name	employee_id	dob	gender	address	contact_no	date_of_joining	termination_date	designation	hostel_id	salary
test2	12345	1990-11-25	f	somewhere	2423423	2017-11-18	None	cleaning	1	20000
23423	21321	2017-11-10	f	xvxz	45654	2017-11-11	None	cleaning	1	20000

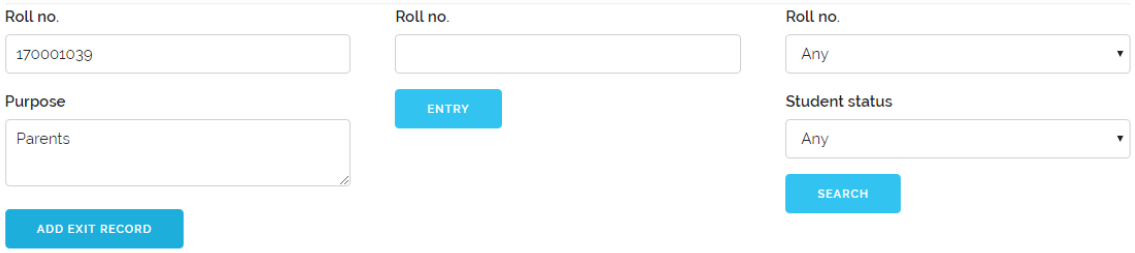
Based on one of the following:

i. Name ii. Employee id

iii. Hostel id they're working in and their designation

GATE RECORD

Add gate exit record (*Automatic exit time*)



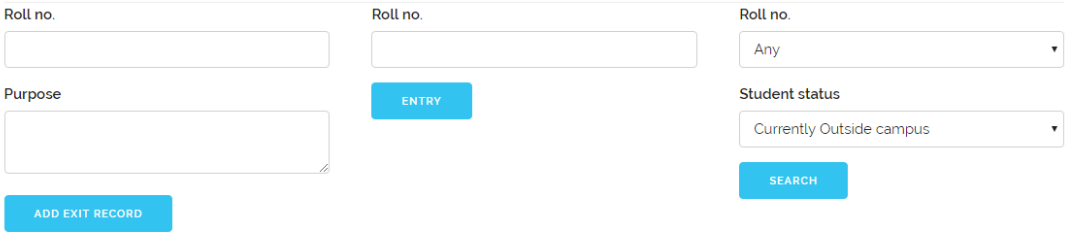
roll_no	purpose	entry_time	exit_time
170001039	Parents	None	2017-11-18 14:45:09

Automatically an exit time is assigned using before insert trigger.

Constraint : Student must exist and should not be ex student(year value shouldn't be 0)

Implemented using an before insert trigger. Outputs an error message if not valid.("Ex students not allowed").

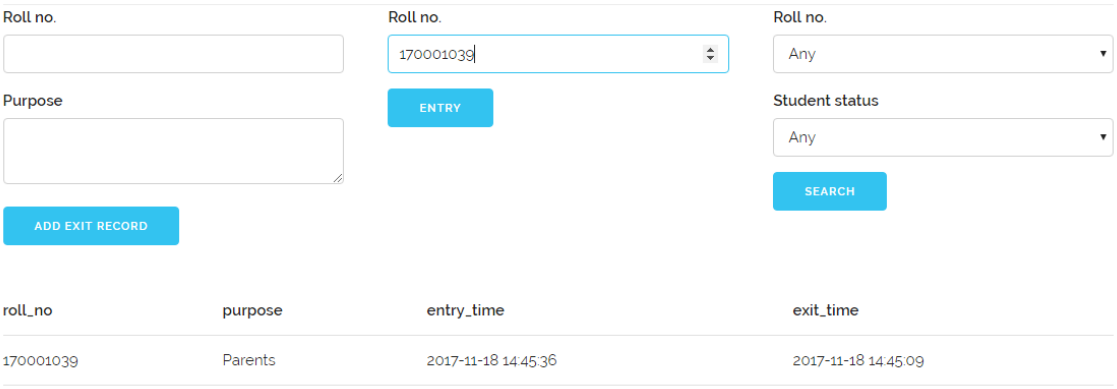
Search gate record, search currently outside students,see all gate records for a particular roll no.



roll_no	name	dob	gender	address	contact_no	year	branch	hostel_id	flat	room	purpose	entry_time	exit_time
170001039	test	2017-11-18	f	lol	123456789	1	cse	1	101	a	Parents	None	2017-11-18 14:46:38

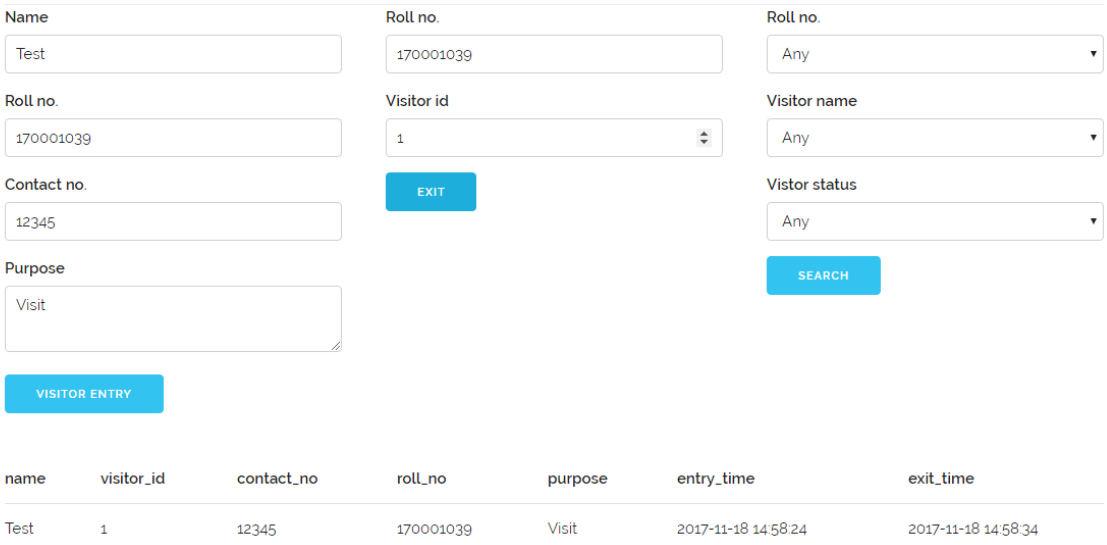
For example, in the above image all students currently outside the campus were searched.

Gate entry record using only roll no. (*Automatic entry time*)



When student comes back , only there roll no needs to be entered. Then the gate record that was made at the time they went outside is updated with current time as entry time using an stored procedure gate_record_in(in roll_no int).

VISITOR



Visitor entry (*Automatic entry time*)

Using a before insert trigger, automatically an entry time will be assigned.

Constraint : Student must exist and should not be ex student(year value shouldn't be 0)

Implemented using an before insert trigger. Outputs an error message if not valid.("Ex students not allowed").

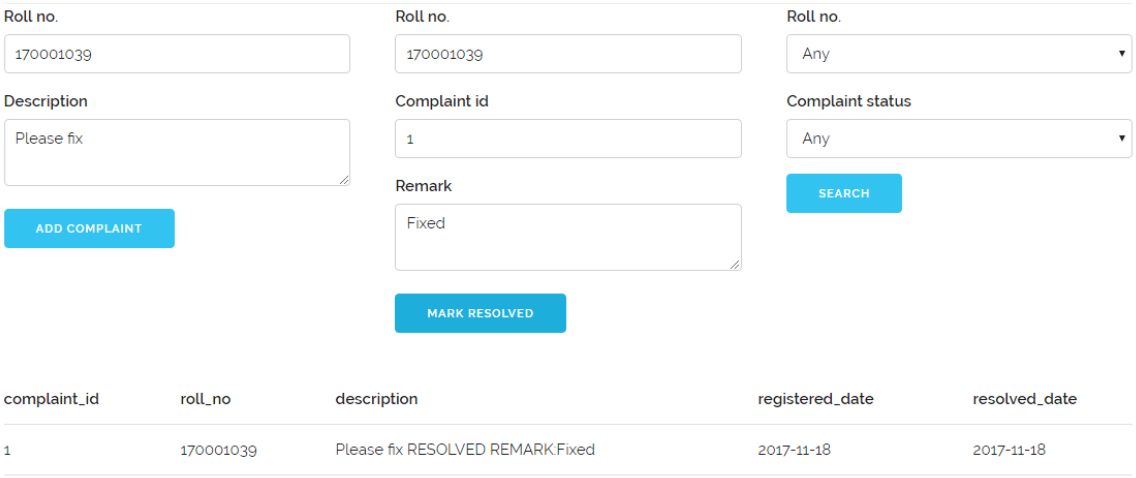
Visitor exit (*Automatic exit time*)

Exit time will be set for an visitor with given visitor id visiting a student with given roll no using a stored procedure visitor_out. Visitor id (which is discriminator for weak entity set visitor) can be found using search functionality.

Visitor record search

Can be searched by visitor name, visitors visiting a particular student , visitor status(inside or outside campus) or any combination of these.

COMPLAINT



Add complaint (*Automatic registered date*)

Using a before insert trigger, automatically a registered date will be assigned.

Constraint : Student must exist and should not be ex student(year value shouldn't be 0)

Implemented using an before insert trigger. Outputs an error message if not valid.("Ex students not allowed").

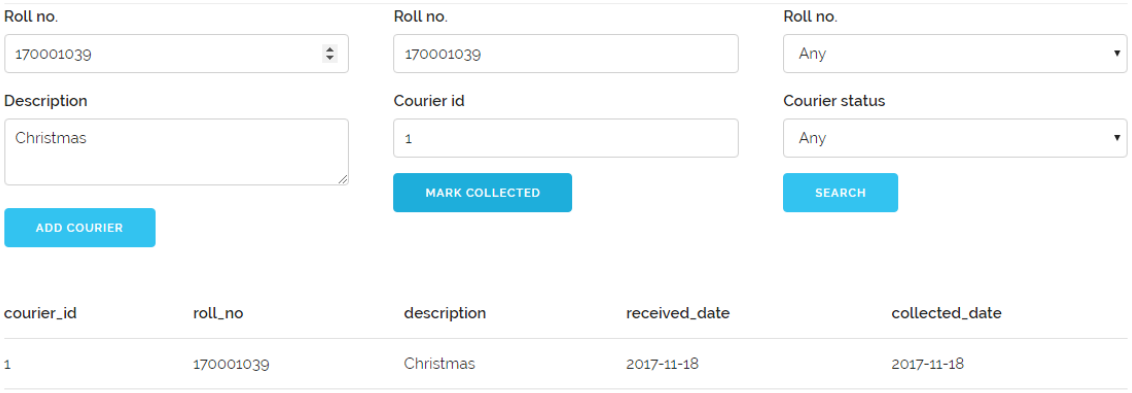
Mark resolved and add a remark (*Automatic resolved date*)

Using a stored procedure ,resolved date will be assigned and remark will be appended to the description.

Search complaints

Using roll no. , complaint status(resolved, unresolved , any) or combination.

COURIER



Add Courier (*Automatic received date*)

Using a before insert trigger, automatically an received date will be assigned.

Constraint : Student must exist and should not be ex student(year value shouldn't be 0)

Implemented using an before insert trigger. Outputs an error message if not valid.("Ex students not allowed").

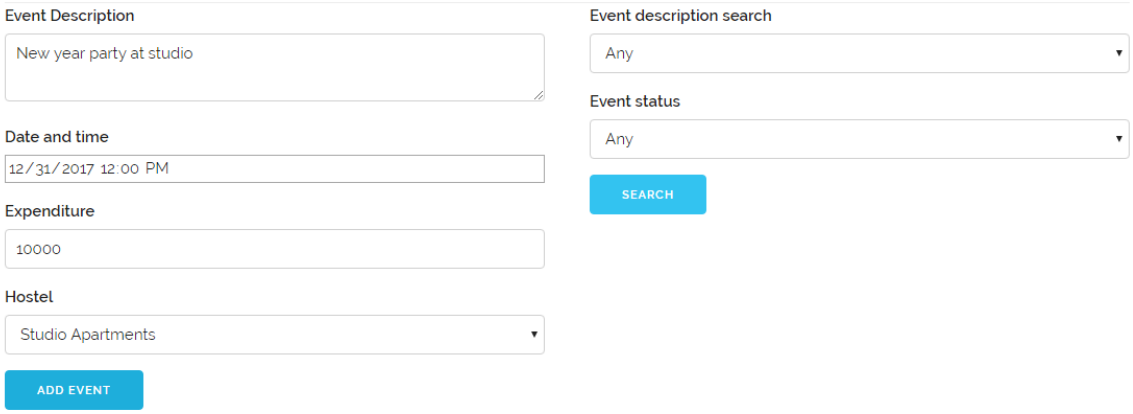
Mark collected (*Automatic collected date*)

Using a stored procedure ,collected date will be assigned .

Search couriers

Using roll no. , courier status(collected, uncollected , any) or combination.

EVENTS



event_id	description	start_time	expenditure	hostel_id
10	New year party at studio	2017-12-31 12:00:00	10000	2

Add upcoming event

Constraint : Start datetime must be greater than now.

Implemented using an before insert trigger. Outputs an error message if datetime is in past.

Search events

Using description or status (upcoming , earlier ,any) or combination

HOSTEL

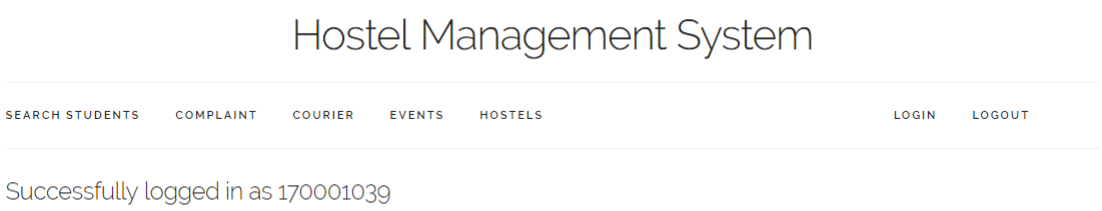
name	hostel_id	capacity	no_students	no_rooms_available
Hall of residences	1	450	1	449
Studio apartments	2	900	3	447
Silver Springs	3	450	3	447

Automatic updating of hostel info

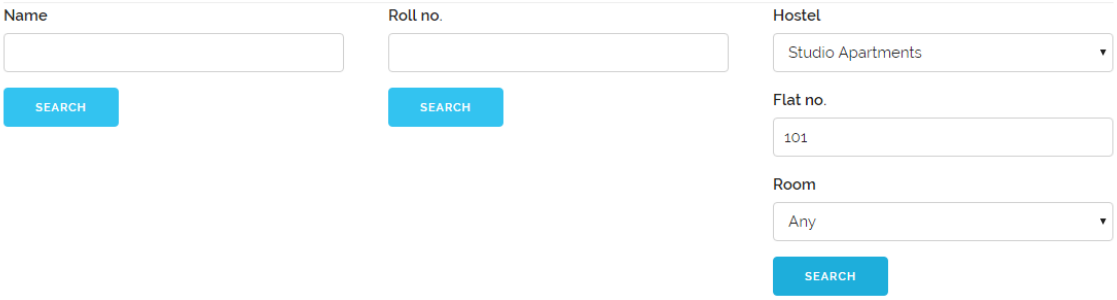
Using triggers, no of students residing and no of rooms available values for each hostel are always kept accurate every time student table is modified.

Features for students

Note: student login only works for current students, ex students can't login.



SEARCH STUDENTS

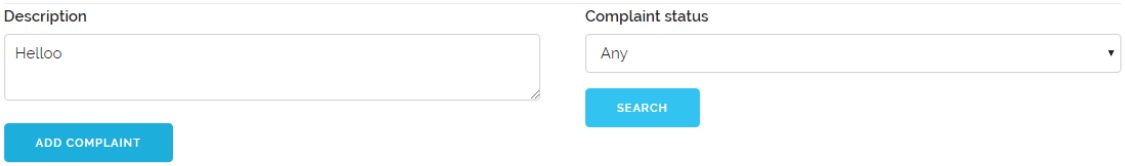


name	roll_no	year	branch	hostel_id	flat	room
welp	122	2	ee	2	101	c
vfdvdf	123	2	mems	2	101	a
test	170001010	2	cse	2	101	b

Same feature as for admins, except limited info is returned. For eg, contact no. and address are not shown.

COMPLAINT

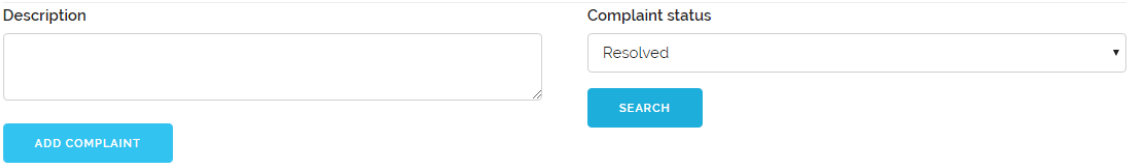
Add complaint (for their own roll no., *Automatic registered date*)



complaint_id	roll_no	description	registered_date	resolved_date
3	170001039	Helloo	2017-11-18	None

Using a before insert trigger, automatically an registered date will be assigned.

See own complaints



complaint_id	roll_no	description	registered_date	resolved_date
1	170001039	Please fix RESOLVED REMARK:Fixed	2017-11-18	2017-11-18

Can specify complaint status to search : resolved,unresolved,any.

SEE COURIERS (BELONGING TO THEM)

courier_id	roll_no	description	received_date	collected_date
2	170001039	Hello there	2017-11-18	None
1	170001039	Christmas	2017-11-18	2017-11-18

Will be ordered such that those that haven't been collected will be on top and have an collected date none.

SEE UPCOMING EVENTS

event_id	description	start_time	expenditure	hostel_id
4	sdfsd	2017-11-27 14:22:00	34543	1
5	sdfsd	2017-11-27 14:22:00	34543	1
6	sdfsd	2017-11-27 14:22:00	34543	1
7	sdfsd	2017-11-27 02:22:00	34543	1
8	sdcds	2017-11-24 15:33:00	22222	1
10	New year party at studio	2017-12-31 12:00:00	10000	2

Events taking place in their own hostel id will be shown first (in the order).

HOSTEL

See info regarding hostels. (Same as for authorities, please see the image given previously.)

Implementation

Software Requirements

- Python 3.5.6
- Platform Independent MySQL connector for python
- The 'bottle' framework for python
- more_itertools library for python
- Skeleton css framework for website design
- Jquery and AJAX for loading data returned by db server on pages (eg. Search results) without refreshing and for generating dynamic forms
- Html, CSS , Javascript

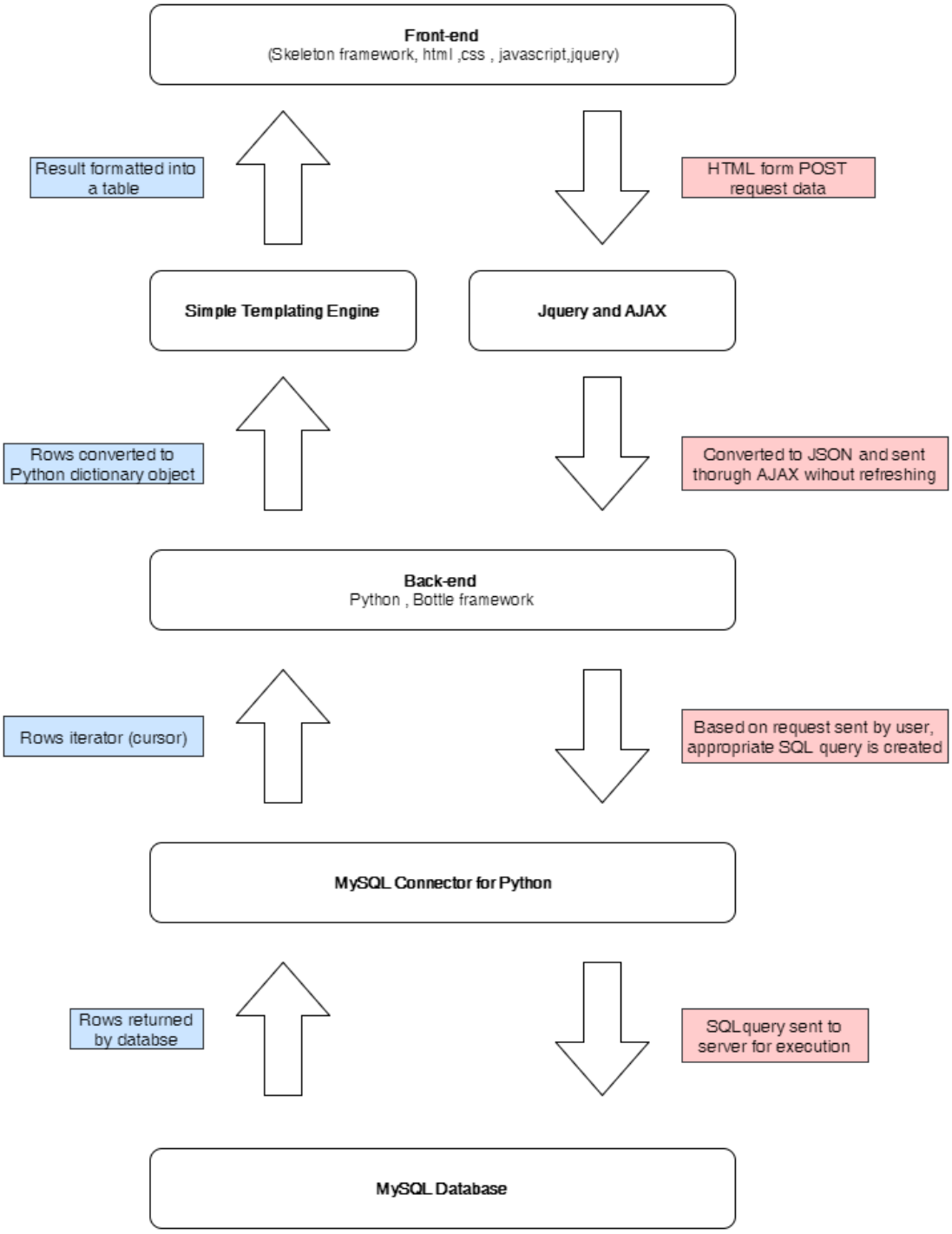
Python and bottle framework have been used to connect the frontend with the database.

Simple templating engine, a part of bottle framework , was used to generate dynamic content to show to the end user.

Frontend was created using 'Skeleton' framework , and html,css,javascript, jquery.

Jquery and AJAX were used to send forum data to server and return the response to user , all without refreshing the page.

Block Diagram



Creating schema structure

```
CREATE DATABASE  IF NOT EXISTS `hms`
USE `hms`;

--
-- Table structure for table `complaint`
--

DROP TABLE IF EXISTS `complaint`;

CREATE TABLE `complaint` (
  `complaint_id` int(11) NOT NULL,
  `roll_no` int(11) NOT NULL,
  `description` varchar(400) DEFAULT NULL,
  `registered_date` date NOT NULL,
  `resolved_date` date DEFAULT NULL,
  PRIMARY KEY (`complaint_id`,`roll_no`),
  KEY `fk_student_complaint` (`roll_no`),
  CONSTRAINT `fk_student_complaint` FOREIGN KEY (`roll_no`) REFERENCES
`student` (`roll_no`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


--
-- Table structure for table `courier`
--

DROP TABLE IF EXISTS `courier`;

CREATE TABLE `courier` (
  `courier_id` int(11) NOT NULL,
  `roll_no` int(11) NOT NULL,
  `description` varchar(400) DEFAULT NULL,
  `received_date` date NOT NULL,
  `collected_date` date DEFAULT NULL,
  PRIMARY KEY (`courier_id`,`roll_no`),
  KEY `fk_student_courier` (`roll_no`),
  CONSTRAINT `fk_student_courier` FOREIGN KEY (`roll_no`) REFERENCES
`student` (`roll_no`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;



--
-- Table structure for table `employee`
--
```

```
DROP TABLE IF EXISTS `employee`;

CREATE TABLE `employee` (
  `name` varchar(40) NOT NULL,
  `employee_id` int(11) NOT NULL,
  `dob` date DEFAULT NULL,
  `gender` enum('f','m') DEFAULT NULL,
  `address` varchar(200) DEFAULT NULL,
  `contact_no` int(11) DEFAULT NULL,
  `date_of_joining` date NOT NULL,
  `termination_date` date DEFAULT NULL,
  `designation` enum('cleaning','warden','mess','office','other')
DEFAULT NULL,
  `hostel_id` int(11) DEFAULT NULL,
  `salary` int(11) DEFAULT NULL,
  PRIMARY KEY (`employee_id`),
  KEY `fk_emp_hostel` (`hostel_id`),
  CONSTRAINT `fk_emp_hostel` FOREIGN KEY (`hostel_id`) REFERENCES
`hostel` (`hostel_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Table structure for table `event`
--

DROP TABLE IF EXISTS `event`;

CREATE TABLE `event` (
  `event_id` int(11) NOT NULL AUTO_INCREMENT,
  `description` varchar(400) DEFAULT NULL,
  `start_time` datetime NOT NULL,
  `expenditure` int(11) DEFAULT NULL,
  `hostel_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`event_id`),
  KEY `fk_hostel_event` (`hostel_id`),
  CONSTRAINT `fk_hostel_event` FOREIGN KEY (`hostel_id`) REFERENCES
`hostel` (`hostel_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `failed_student`
--

DROP TABLE IF EXISTS `failed_student`;

CREATE TABLE `failed_student` (
  `name` varchar(40) NOT NULL,
  `roll_no` int(11) NOT NULL,
```

```
  `dob` date DEFAULT NULL,
  `gender` enum('f','m') DEFAULT NULL,
  `address` varchar(200) DEFAULT NULL,
  `contact_no` int(11) DEFAULT NULL,
  `year` enum('0','1','2','3','4') NOT NULL,
  `branch` enum('cse','me','ee','mems','ce') NOT NULL,
  `hostel_id` int(11) NOT NULL,
  `flat` int(11) NOT NULL,
  `room` enum('a','b','c','d','e') NOT NULL,
  PRIMARY KEY (`roll_no`),
  KEY `fk_hostel_failed_student` (`hostel_id`),
  CONSTRAINT `fk_hostel_failed_student` FOREIGN KEY (`hostel_id`)
REFERENCES `hostel` (`hostel_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `gate_record`
--

DROP TABLE IF EXISTS `gate_record`;

CREATE TABLE `gate_record` (
  `roll_no` int(11) NOT NULL,
  `purpose` varchar(400) DEFAULT NULL,
  `entry_time` datetime DEFAULT NULL,
  `exit_time` datetime NOT NULL,
  PRIMARY KEY (`exit_time`,`roll_no`),
  KEY `fk_student_gate_record` (`roll_no`),
  CONSTRAINT `fk_student_gate_record` FOREIGN KEY (`roll_no`)
REFERENCES `student` (`roll_no`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `hostel`
--

DROP TABLE IF EXISTS `hostel`;

CREATE TABLE `hostel` (
  `name` varchar(40) NOT NULL,
  `hostel_id` int(11) NOT NULL,
  `capacity` int(11) DEFAULT NULL,
  `no_students` int(11) DEFAULT NULL,
  `no_rooms_available` int(11) DEFAULT NULL,
  PRIMARY KEY (`hostel_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;
```

```
--
-- Table structure for table `student`
--

DROP TABLE IF EXISTS `student`;

CREATE TABLE `student` (
  `name` varchar(40) NOT NULL,
  `roll_no` int(11) NOT NULL,
  `dob` date DEFAULT NULL,
  `gender` enum('f','m') DEFAULT NULL,
  `address` varchar(200) DEFAULT NULL,
  `contact_no` int(11) DEFAULT NULL,
  `year` enum('1','2','3','4','0') NOT NULL,
  `branch` enum('cse','me','ee','mems','ce') NOT NULL,
  `hostel_id` int(11) NOT NULL,
  `flat` int(11) DEFAULT NULL,
  `room` enum('a','b','c','d','e') DEFAULT NULL,
  PRIMARY KEY (`roll_no`),
  KEY `fk_hostel_student` (`hostel_id`),
  CONSTRAINT `fk_hostel_student` FOREIGN KEY (`hostel_id`) REFERENCES
`hostel` (`hostel_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `visitor`
--

DROP TABLE IF EXISTS `visitor`;

CREATE TABLE `visitor` (
  `name` varchar(40) NOT NULL,
  `visitor_id` int(11) NOT NULL,
  `contact_no` int(11) DEFAULT NULL,
  `roll_no` int(11) NOT NULL,
  `purpose` varchar(400) DEFAULT NULL,
  `entry_time` datetime NOT NULL,
  `exit_time` datetime DEFAULT NULL,
  PRIMARY KEY (`visitor_id`,`roll_no`),
  KEY `fk_student_visitor` (`roll_no`),
  CONSTRAINT `fk_student_visitor` FOREIGN KEY (`roll_no`) REFERENCES
`student` (`roll_no`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Final view structure for view `current_students`
```

```
--
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `current_students` AS
    SELECT
        `student`.`name` AS `name`,
        `student`.`roll_no` AS `roll_no`,
        `student`.`dob` AS `dob`,
        `student`.`gender` AS `gender`,
        `student`.`address` AS `address`,
        `student`.`contact_no` AS `contact_no`,
        `student`.`year` AS `year`,
        `student`.`branch` AS `branch`,
        `student`.`hostel_id` AS `hostel_id`,
        `student`.`flat` AS `flat`,
        `student`.`room` AS `room`
    FROM
        `student`
    WHERE
        (`student`.`year` <> '0')
```

Note: The view current_students simple means all students whose year attribute is not 0, which means they are not ex-students.

Triggers, stored procedures and functions

In total, 17 trigggers ,1 function and 6 stored procedures were used. The representative examples are shown below.

NO. OF AVAILABLE ROOMS FUNCTION

```
CREATE DEFINER = `root`@
`localhost`
FUNCTION `availableRooms` (h_id int) RETURNS int(11)
begin

declare no_of_available_rooms int;

set no_of_available_rooms = (select(select capacity from hostel where
hostel_id = h_id) - count(roll_no)

    from student

    where hostel_id = h_id and year != 0);

return no_of_available_rooms;

end
```

Returns the number of rooms available in specified hostel.

AUTOMATIC ALLOCATION OF HOSTEL ROOMS

```
CREATE DEFINER = `root`@
`localhost`
trigger roomAllotment

before insert on student

for each row

begin

declare n int; /*Number of flats available*/

declare r int; /*flat number*/
declare h int;

declare f varchar(1); /*room number*/
set h = 0;
```

```
if new.year like '1'
then set h = 1;
end
if;
if new.year like '2'
then set h = 2;
end
if;
if new.year like '3'
then set h = 2;
end
if;
if new.year like '4'
then set h = 3;
end
if;


set new.hostel_id = h;

set n = availableRooms(h);



if n > 0 then

set r = (select flat from student where hostel_id = h and year != '0'
    order by flat desc limit 1);


set f = (select room from student where flat = r and hostel_id = h and
year != '0'
    order by room desc limit 1);


if isnull(r) then
set r = 100;
set f = 'E';
end
if;
if mod(r, 100) = 18 then

if f = 'A'
then

set new.flat = r;

set new.room = 'B';
```

```
elseif f = 'B'
then

set new.flat = r;

set new.room = 'C';

elseif f = 'C'
then

set new.flat = r;

set new.room = 'D';

elseif f = 'D'
then

set new.flat = r;

set new.room = 'E';

else

    set new.flat = ((r / 100) + 1) * 100 + 1;

set new.room = 'A';

end
if;

else

if f = 'A'
then

set new.flat = r;

set new.room = 'B';

elseif f = 'B'
then

set new.flat = r;

set new.room = 'C';

elseif f = 'C'
then
```

```
set new.flat = r;

set new.room = 'D';

elseif f = 'D'
then

set new.flat = r;

set new.room = 'E';

else

    set new.flat = r + 1;

set new.room = 'A';

end
if;

end
if;

else

    set new.flat = NULL;

set new.room = NULL;

end
if;
end
```

There is a similar before update trigger.

After a new student has been added to the db , the room Allotment trigger automatically allocates the next free hostel room from the appropriate hostel id(1 for first year, 2 for second and third year,3 for fourth year) to the student. It's assuming that every floor has 18 rooms.

For eg. First first year student will be allocated 1 101 a next 1 101 b so on till 1 118 e then 1 201 a will come.

Similarly, for fourth year it will be hostel id 3, for second and third it will be hostel id 2.

FORWARD TO NEXT YEAR

(OR AUTOMATIC CHANGING OF HOSTEL ROOMS AND YEAR INFORMATION EVERY ACADEMIC YEAR)

At the end of academic year, authorities can submit a file containing list of failed students. Then using a ***Stored procedure and Before update trigger***, year info of all other students is updated and they are allocated new appropriate rooms. Failed students retain there year info and reallocated appropriate room in same hostel id.

Eg. Student in third year is living in a certain room in hostel id 2. After this process they will be automatically allocated an appropriate room in hostel id 3. Before that all fourth year students who have passed will be made ex students and there rooms will be freed.

Transactions control *(start transaction, commit)* in MySQL is used since this process must be atomic. Thus it is ensured that partial updating of student years will never happen**.**

Working :

First a stored procedure fail(in roll_no int) is called on all students in failed list. Essentially it deletes them from student table and temporarily stores them in a temp table.

Next a stored procedure forward() is called which updates the year info of students in the following order:

First fourth year students are made ex students(by setting there year info 0). This frees there rooms. Next third year students year is changed from 3 to 4. This causes the Before update trigger uroomAllotment to fire and they shifted from hostel id 2 to hostel id 3. In this process they are allocated appropriate rooms in continuous order. This also happens for second and first year students.

Then failed students are inserted back to original table with same year. Then the before insert trigger roomAllotment allocates them next free rooms in appropriate hostel.

Stored procedure fail(in roll_no1 int) :

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `fail`(in roll_no1 int)
begin

DECLARE exit handler for sqlexception
  BEGIN
    -- ERROR
  ROLLBACK;
END;

DECLARE exit handler for sqlwarning
 BEGIN
```

```
    -- WARNING
 ROLLBACK;
END;

START TRANSACTION;
insert into failed_student select * from student where
roll_no=roll_no1;
DELETE FROM student
WHERE
    roll_no = roll_no1;
COMMIT;

end
```

Stored procedure forward ():

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `forward`()
begin
DECLARE exit handler for sqlexception
  BEGIN
    -- ERROR
  ROLLBACK;
END;

DECLARE exit handler for sqlwarning
 BEGIN
    -- WARNING
 ROLLBACK;
END;

START TRANSACTION;

UPDATE student
SET
    flat = NULL,
    room = NULL
WHERE
    year != '0' AND year != '4';

UPDATE student
SET
    year = '0'
WHERE
    year = '4';

UPDATE student
SET
    year = '4'
WHERE
```

```
    year = '3';

UPDATE student
SET
    year = '3'
WHERE
    year = '2';

UPDATE student
SET
    year = '2'
WHERE
    year = '1';

insert into student
select * from failed_student;

DELETE FROM failed_student;


COMMIT;

end
```

Transactions control *(start transaction, commit)* in MySQL is used since this process must be atomic. **Thus it is ensured that partial updating of student years will never happen.**

AUTOMATIC UPDATING OF HOSTEL INFO

```
CREATE
    DEFINER = `root`@`localhost` trigger iupdateHostel AFTER INSERT
            ON student FOR EACH row BEGIN UPDATE
                hostel
            SET
                no_rooms_available = availableRooms (hostel_id);
 UPDATE
        hostel
    SET
        no_students = capacity - no_rooms_available;
END
```

Similarly, there's an after insert trigger just like this one.

AUTOMATIC ASSIGNING OF SALARY TO EMPLOYEE

```
CREATE DEFINER = `root`@
`localhost`
trigger employeeSalary

before insert on employee

for each row

begin


if new.designation = "Cleaning"
then

set new.salary = 20000;

elseif new.designation = "Warden"
then

set new.salary = 80000;

elseif new.designation = "Mess"
then

set new.salary = 15000;

elseif new.designation = "Office"
```

```
then

set new.salary = 60000;

else

    set new.salary = 17000;

end
if;

end
```

AUTOMATIC RECORDING OF DATETIME WHILE ADDING NEW ENTRY

There are 5 triggers based on this principle : complaint_in, , courier_in, , employee_in, , gate_record_out, visitor_id_set.

One example is shown below. It shows how complaint_id and registered_date are automatically decided.

```
CREATE DEFINER=`root`@`localhost` trigger complaint_in
before insert on complaint

for each row

begin

declare r int;
    declare i int;
    set new.registered_date=curdate();
    set r=new.roll_no;
    set i=(select complaint_id from complaint where roll_no=r order by
complaint_id desc limit 1);
    if(isnull(i)) then set i=0;
    end if;
    set new.complaint_id=i+1;
end
```

AUTOMATIC RECORDING OF DATETIME WHILE MARKING COLLECTED/RESOLVED/OUT/IN

There are 4 stored procedures for this purpose : complaint_res, gate_record_in , visitor_out and courier_col.

An example is shown below. The complaint_res adds current date as resolved date to the given (complaint id and roll no) and also appends a remark to its description.

```
CREATE DEFINER = `root`@
`localhost`
PROCEDURE `complaint_res` (roll_no1 int, complaint_id1 int, add_des
varchar(191))
begin
update complaint
set resolved_date = curdate(), description = concat(description, "
RESOLVED REMARK:", add_des)
where roll_no = roll_no1 and isnull(resolved_date) and complaint_id =
complaint_id1;

end
```

CHECKING STUDENT IS NOT EX-STUDENT (4 TRIGGERS)

```
CREATE DEFINER = `root`@
`localhost`
trigger chk_st_comp before insert on complaint
for each row
begin
if (select year from student where roll_no = new.roll_no) = '0'
then
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'Ex students not allowed';
end
if;
end
```

Checking that student exists is done using foreign key constraint . However, for checking that student year is not 0 (that is student is not ex student), triggers are used.

There are 4 such triggers : chk_st_comp ,chk_st_visit, chk_st_cour, chk_st_gate

One example is shown above.

CHECKING THAT NEW EVENT DATE TIME IS IN FUTURE

```
CREATE
    DEFINER = `root`@`localhost` trigger chk_date BEFORE INSERT
            ON event FOR EACH row BEGIN IF new.start_time < now ()
        THEN SIGNAL SQLSTATE '45000'
    SET
        MESSAGE_TEXT = 'Date time must be in future';
END IF;
END
```

This disallows the possibility of erroneously entering a date time in past as starting time of an upcoming event.

CHECKING THAT STUDENT IS ATLEAST 17 YEARS OLD

```
drop trigger if exists chk_age_student;
delimiter //

CREATE DEFINER=`root`@`localhost` trigger chk_age_student before insert
on student
      for each row
       begin
      if  TIMESTAMPDIFF(YEAR, new.dob, curdate())<17  then
       SIGNAL SQLSTATE '45000'
       SET MESSAGE_TEXT = 'Student must be at least 17 year old.';
       end if;
    end//
    delimiter ;
```

A summary of constraints used

- Check constraint implemented using trigger enforces that starting date time of an upcoming event must be in future.
- Check constraint implemented using trigger enforces that age of new student must be at least 17.
- Check constraints (4 total) implemented using triggers enforce that student roll no. given as input in various options for gate_record, visitor ,courier, complaints etc belongs to a current student , not an ex-student.
- Foreign key constraint enforces that the roll no. given as input in various options for gate_record, visitor ,courier, complaints etc is valid.
- Foreign key constraint enforces that the hostel id. given as input in various options for employee, event etc is valid.

Connecting Python with MySQL: An example

The following code initializes the MySQL connector for Python:

```
#python -m http.server 8000 --bind 127.0.0.1
from bottle import route, run, debug, template, request, static_file,
error ,get,post,response,redirect

# only needed when you run Bottle on mod_wsgi
from bottle import default_app
import datetime
import mysql.connector
import collections
from mysql.connector import errorcode
from  more_itertools import unique_everseen


config = {
  'user': 'root',
  'password': ‘your_password’,
  'host': '127.0.0.1',
  'database': 'hms',
  'raise_on_warnings': True,
}
cnx = mysql.connector.connect(**config)
```

After this , queries can be excuted using Python.

For example , consider the simple ‘Hostel’ page, which shows details of hostels:

Hostel Management System

STUDENT EMPLOYEES GATE RECORD VISITOR COMPLAINT COURIER EVENTS HOSTELS LOGIN LOGOUT

name	hostel_id	capacity	no_students	no_rooms_available
Hall of residences	1	450	1	449
Studio apartments	2	900	3	447
Silver Springs	3	450	3	447

For this page, the function in python is as follows:

```
@get('/show_hostel')
def show_hostel():


    c = cnx.cursor()

    try:
        c.execute("SELECT * FROM hostel ")
    except mysql.connector.Error as err:
        return ("Failed fetching from table hostel: {}".format(err))
    result = c.fetchall()
    cnx.commit()
    c.close()

    c = cnx.cursor()

    try:
        c.execute("SELECT column_name from information_schema.columns
where table_name='hostel' and table_schema='hms'")
    except mysql.connector.Error as err:
        return ("Failed fetching column names of table hms.hostel,
please make sure that it exists: {}".format(err))
    column_names = c.fetchall()
    cnx.commit()
    c.close()


    output = template('make_table',
rows=result,columns=column_names,lolcat=request.get_cookie("user"))
    return output
```

The try,except blocks are there to catch any errors and in that case send a message to user describing the error.

Now this function executes the query, gets the result and then send it to sends it to templating engine for formatting.

Column names for table are taken from the information schema instead of manually typing them.

For the hostel page, the template code('make_table.tpl') is:

```
%x=(lolcat!='0')*1
%y='header{}'.format(x)
% include(y)

<table id="customers" class="u-full-width">
  <tr>
    %for name in columns:
    %for subname in name:
      <th>{{subname}}</th>
      %end
    %end
</tr>
  %for row in rows:
  <tr>
  %for col in row:
    <td>{{col}}</td>
  %end
  </tr>
  %end

</table>
<!-- End Document
  ------------------------------------------------ -->
</body>
</html>
```

The header file contains the website design code. Since functionality is different for authorities and students, the header file is different as well. The first three lines determine appropriate file for each user based on their cookie contents.

After that the table is appended and shown to user.

Like this page, other pages are created. However, they are rather big and contain complex queries. The total python code size, even after excluding core website design, is more than

4000 lines, hence it is not possible to show them all here and only a representative simple page was chosen.

Experimentation below:

Experimentation (Testing)

Please note that while extensive testing was done, due to the constraint of space only representative test cases are shown below.

EXCEPTION HANDLING

All queries in python code are executed in a try except block. This ensures that the system itself will never fail or behave in unexpected ways. In case the query fails, the system alerts the user with appropriate reason.

(eg. “Event datetime must be in future” , “Ex-students not allowed”,”Roll no. doesn’t exist. “ etc)

TESTING THE AUTOMATIC ALLOCATION OF ROOMS

The entire students table was wiped and following 20 students were added:

```
INSERT INTO `hms`.`student` (`name`, `roll_no`, `dob`, `gender`,
`address`, `contact_no`, `year`, `branch`) VALUES
('a', '1', '1990/1/1', 'f', 'a', '1', '1', 'cse'),
('b', '2', '1990/1/1', 'f', 'a', '1', '1', 'cse'),
('c', '3', '1990/1/1', 'f', 'a', '1', '1', 'cse'),
('d', '4', '1990/1/1', 'f', 'a', '1', '1', 'cse'),
('e', '5', '1990/1/1', 'f', 'a', '1', '1', 'cse'),
('a', '11', '1990/1/1', 'f', 'a', '1', '2', 'cse'),
('b', '12', '1990/1/1', 'f', 'a', '1', '2', 'cse'),
('c', '13', '1990/1/1', 'f', 'a', '1', '2', 'cse'),
('d', '14', '1990/1/1', 'f', 'a', '1', '2', 'cse'),
('e', '15', '1990/1/1', 'f', 'a', '1', '2', 'cse'),
('a', '21', '1990/1/1', 'f', 'a', '1', '3', 'cse'),
('b', '22', '1990/1/1', 'f', 'a', '1', '3', 'cse'),
('c', '23', '1990/1/1', 'f', 'a', '1', '3', 'cse'),
('d', '24', '1990/1/1', 'f', 'a', '1', '3', 'cse'),
('e', '25', '1990/1/1', 'f', 'a', '1', '3', 'cse'),
('a', '31', '1990/1/1', 'f', 'a', '1', '4', 'cse'),
('b', '32', '1990/1/1', 'f', 'a', '1', '4', 'cse'),
('c', '33', '1990/1/1', 'f', 'a', '1', '4', 'cse'),
('d', '34', '1990/1/1', 'f', 'a', '1', '4', 'cse'),
('e', '35', '1990/1/1', 'f', 'a', '1', '4', 'cse')
;
```

After this the student table was:

name	roll_no	dob	gender	address	contact_no	year	branch	hostel_id	flat	room
a	1	1/1/1990	f	a	1	1	cse	1	101	a
b	2	1/1/1990	f	a	1	1	cse	1	101	b
c	3	1/1/1990	f	a	1	1	cse	1	101	c
d	4	1/1/1990	f	a	1	1	cse	1	101	d
e	5	1/1/1990	f	a	1	1	cse	1	101	e
a	11	1/1/1990	f	a	1	2	cse	2	101	a
b	12	1/1/1990	f	a	1	2	cse	2	101	b
c	13	1/1/1990	f	a	1	2	cse	2	101	c
d	14	1/1/1990	f	a	1	2	cse	2	101	d
e	15	1/1/1990	f	a	1	2	cse	2	101	e
a	21	1/1/1990	f	a	1	3	cse	2	102	a
b	22	1/1/1990	f	a	1	3	cse	2	102	b
c	23	1/1/1990	f	a	1	3	cse	2	102	c
d	24	1/1/1990	f	a	1	3	cse	2	102	d
e	25	1/1/1990	f	a	1	3	cse	2	102	e
a	31	1/1/1990	f	a	1	4	cse	3	101	a
b	32	1/1/1990	f	a	1	4	cse	3	101	b
c	33	1/1/1990	f	a	1	4	cse	3	101	c
d	34	1/1/1990	f	a	1	4	cse	3	101	d
e	35	1/1/1990	f	a	1	4	cse	3	101	e

As expected, first years were allocated hostel id 1, second and third years hostel id 2 and fourth years hostel id 3. The room allocation was conflict free and contiguous ensuring no wastage of space.

TESTING FORWARD TO NEXT YEAR
(AUTOMATIC UPDATING OF YEAR INFO AND HOSTEL ID AT THE END OF ACADEMIC YEAR)

The following file was used as input to the frontend : ('failed.txt')

```
1
11
21
32
41
```

After that forward to next year button was pressed .(See earlier features section for details).

After that following statement was run:

```
SELECT * FROM current_students;
```

Current_students is a view that excludes ex-students, i.e. students with year 0.(See creating schema structure in implementation section for code to create the view)

The result was:

name	roll_no	dob	gender	address	contact_no	year	branch	hostel_id	flat	room
a	1	1/1/1990	f	a	1	1	cse	1	101	a
b	2	1/1/1990	f	a	1	2	cse	2	101	e
c	3	1/1/1990	f	a	1	2	cse	2	102	a
d	4	1/1/1990	f	a	1	2	cse	2	102	b
e	5	1/1/1990	f	a	1	2	cse	2	102	c
a	11	1/1/1990	f	a	1	2	cse	2	102	d
b	12	1/1/1990	f	a	1	3	cse	2	101	a
c	13	1/1/1990	f	a	1	3	cse	2	101	b
d	14	1/1/1990	f	a	1	3	cse	2	101	c
e	15	1/1/1990	f	a	1	3	cse	2	101	d
a	21	1/1/1990	f	a	1	3	cse	2	102	e
b	22	1/1/1990	f	a	1	4	cse	3	101	a
c	23	1/1/1990	f	a	1	4	cse	3	101	b
d	24	1/1/1990	f	a	1	4	cse	3	101	c
e	25	1/1/1990	f	a	1	4	cse	3	101	d
b	32	1/1/1990	f	a	1	4	cse	3	101	e

The result is as expected.

Students 1 ,11,21 and 31 failed so they remained in same year and hostel. All others had the info modified.

All fourth year students other than 31(i.e. 32,33,34,35) were made ex-students by changing their year value to 0 hence they were removed from the current_students view.

To test again , the forward to next year function was used again , except this time failed students input file was empty so all students passed:

name	roll_no	dob	gender	address	contact_no	year	branch	hostel_id	flat	room
a	1	1/1/1990	f	a	1	2	cse	2	102	a
b	2	1/1/1990	f	a	1	3	cse	2	101	a
c	3	1/1/1990	f	a	1	3	cse	2	101	b
d	4	1/1/1990	f	a	1	3	cse	2	101	c
e	5	1/1/1990	f	a	1	3	cse	2	101	d
a	11	1/1/1990	f	a	1	3	cse	2	101	e
b	12	1/1/1990	f	a	1	4	cse	3	101	a
c	13	1/1/1990	f	a	1	4	cse	3	101	b
d	14	1/1/1990	f	a	1	4	cse	3	101	c
e	15	1/1/1990	f	a	1	4	cse	3	101	d
a	21	1/1/1990	f	a	1	4	cse	3	101	e

Again , as expected.

Next the following file was used as input: ('failed.txt')

```
1
11
```

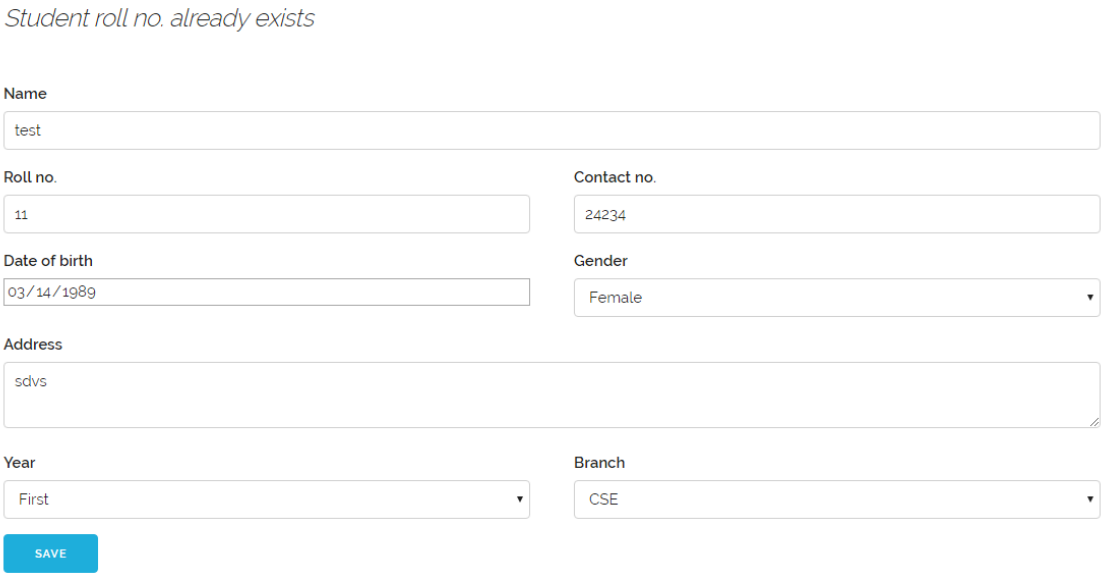
The result was as expected:

name	roll_no	dob	gender	address	contact_no	year	branch	hostel_id	flat	room
a	1	1/1/1990	f	a	1	2	cse	2	101	a
b	2	1/1/1990	f	a	1	4	cse	3	101	a
c	3	1/1/1990	f	a	1	4	cse	3	101	b
d	4	1/1/1990	f	a	1	4	cse	3	101	c
e	5	1/1/1990	f	a	1	4	cse	3	101	d
a	11	1/1/1990	f	a	1	3	cse	2	101	b

TESTING ABNORMAL CASES OF NEW STUDENT INSERTION

There are two possible abnormal cases that may happen:

- A already existing roll no is added again. In this cases an error message will be shown: **"Student roll no. already exists"**

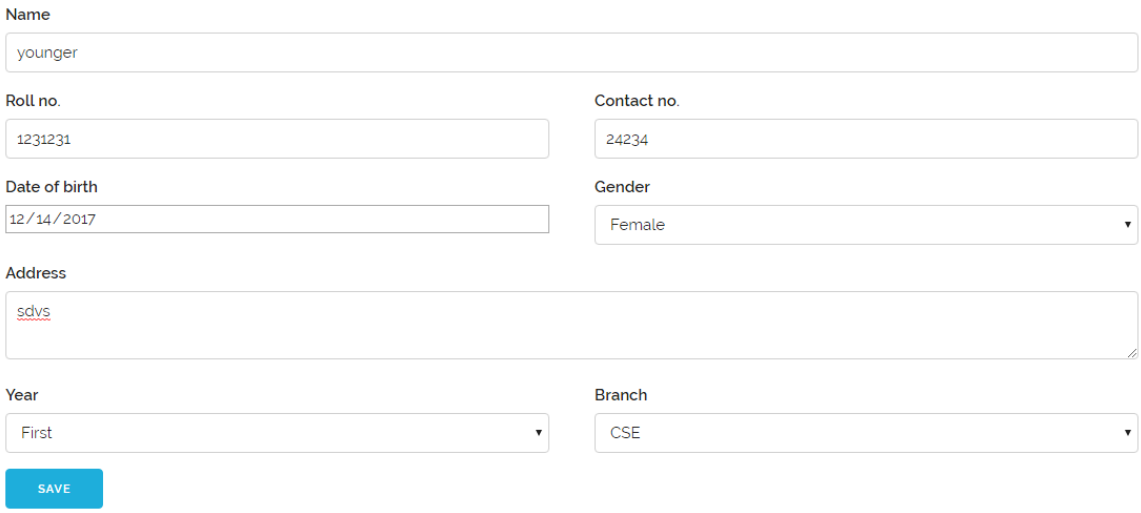


- The code for above is: (Note that 1062 is the error code for duplicate entry for primary key)

```
    c = cnx.cursor()

    query="""INSERT INTO `hms`.`student` (`name`, `roll_no`, `dob`,
`gender`, `address`, `contact_no`, `year`, `branch`)
            VALUES (""" + "%s,"*7 +"%s);"



    try:
        c.execute(query,slist)
    except mysql.connector.Error as err:
        if err.errno==1062:
            return ("Student roll no. already exists")
        return ("Failed adding student to database: {}".format(err))

    cnx.commit()
    c.close()
```

- DOB is such that student is less than 17 years old. In this cases an error message will be shown: **"Student must be at least 17 years old"**

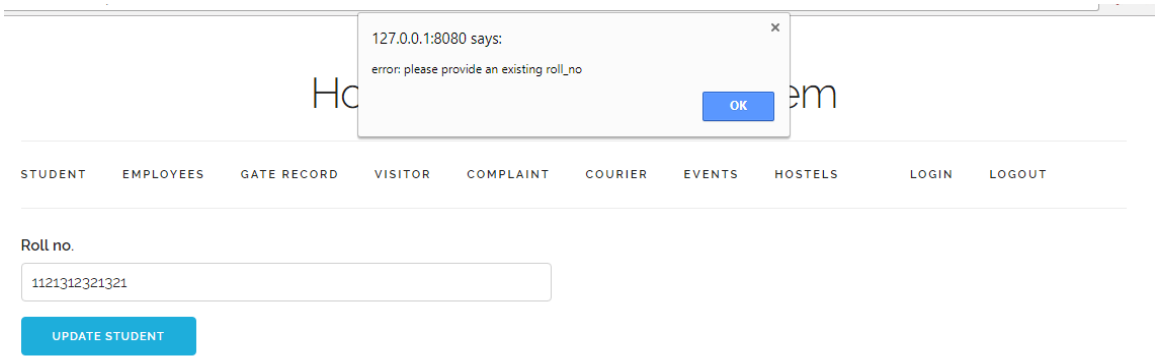
  Failed adding student to database: 1644 (45000): Student must be atleast 17 year old.



- Since other choices such as year and branch , gender are chosen from a list of valid choices, they can't be abnormal.

TESTING ABNORMAL CASES OF UPDATE STUDENT

In this case , if a wrong (non-existent) roll no. is entered, an error message will be shown: "**please provide an existing roll_no**"



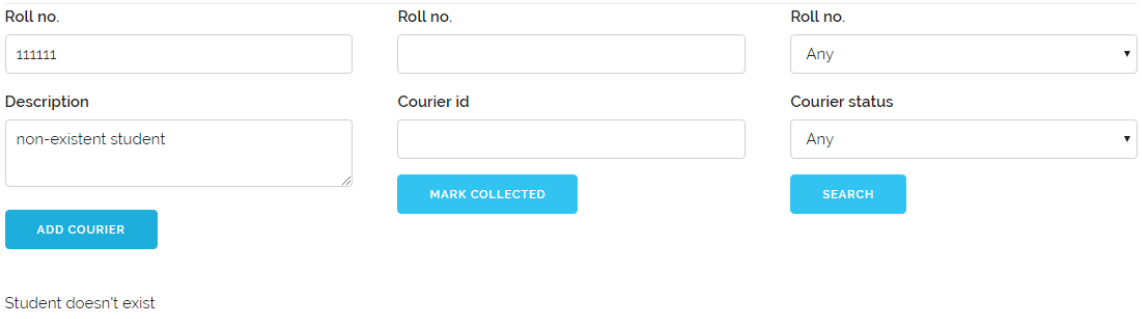
TESTING OTHER FEATURES

Other functionalities such as gate_record, events, visitor , courier and complaints are only accessed through front end, thus they were tested through frontend. So it's not possible to represent test cases as a dataset or table.

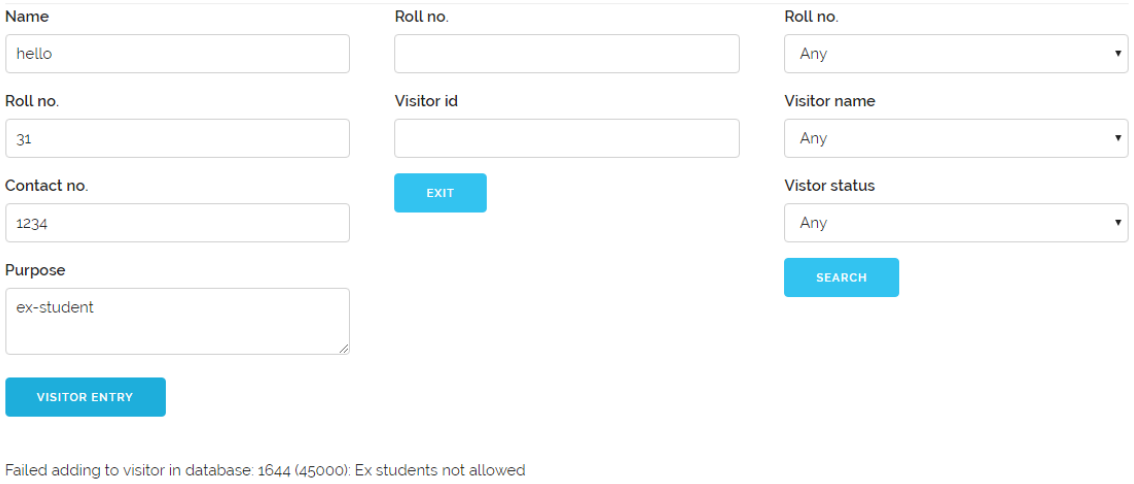
A representative MP4 video "video.mp4" showing testing done on gate_records has been attached, please watch it. (Due to 2mb file limit, not much could be shown.)

Here is a summary of what happens in abnormal cases:

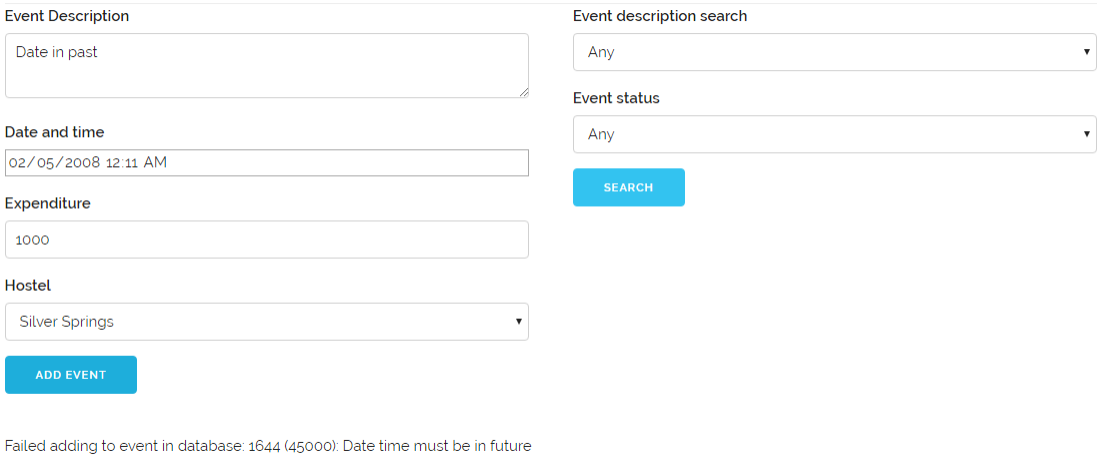
- If non-existant roll no is given as input to either gate_record, visitor, courier or complaint , an error message will be shown: **"Student doesn't exist"**
- For example, see below the error when a courier for non-existent roll no is added



- If roll no. of an ex-student is given as input to either gate_record, visitor, courier or complaint , an error message will be shown: **"Ex-students not allowed"**
- For example, see below the error when a visitor ex-student roll no is added:



- If a date time in past is given when adding an upcoming event, an error message will be shown: "**Date time must be in future**"



HANDLING UNEXPECTED ERRORS

Other than this , if there is any other unexpected error returned by MySQL databse, Python forwards it to the end user making debugging easier.

The code for this functionality is as follows:

```
c = cnx.cursor()

try:
    c.execute(query) #any query
except mysql.connector.Error as err:
    return ("Unexpected Error : {}".format(err)

result = c.fetchall()
cnx.commit()

c.close()
```

The end.

Thank you for reading!

- Project implemented and Project report prepared by Prathamesh Naik 160001037