

College Allotment System

Prepared For

Prof. Amit Dua

(Instructor In-Charge)

CS F212– Database System



Prepared By -

Adarsh Goel – 2020B3A70821P

Arjoo Kumari – 2020B3A70770P

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

January – May 2023

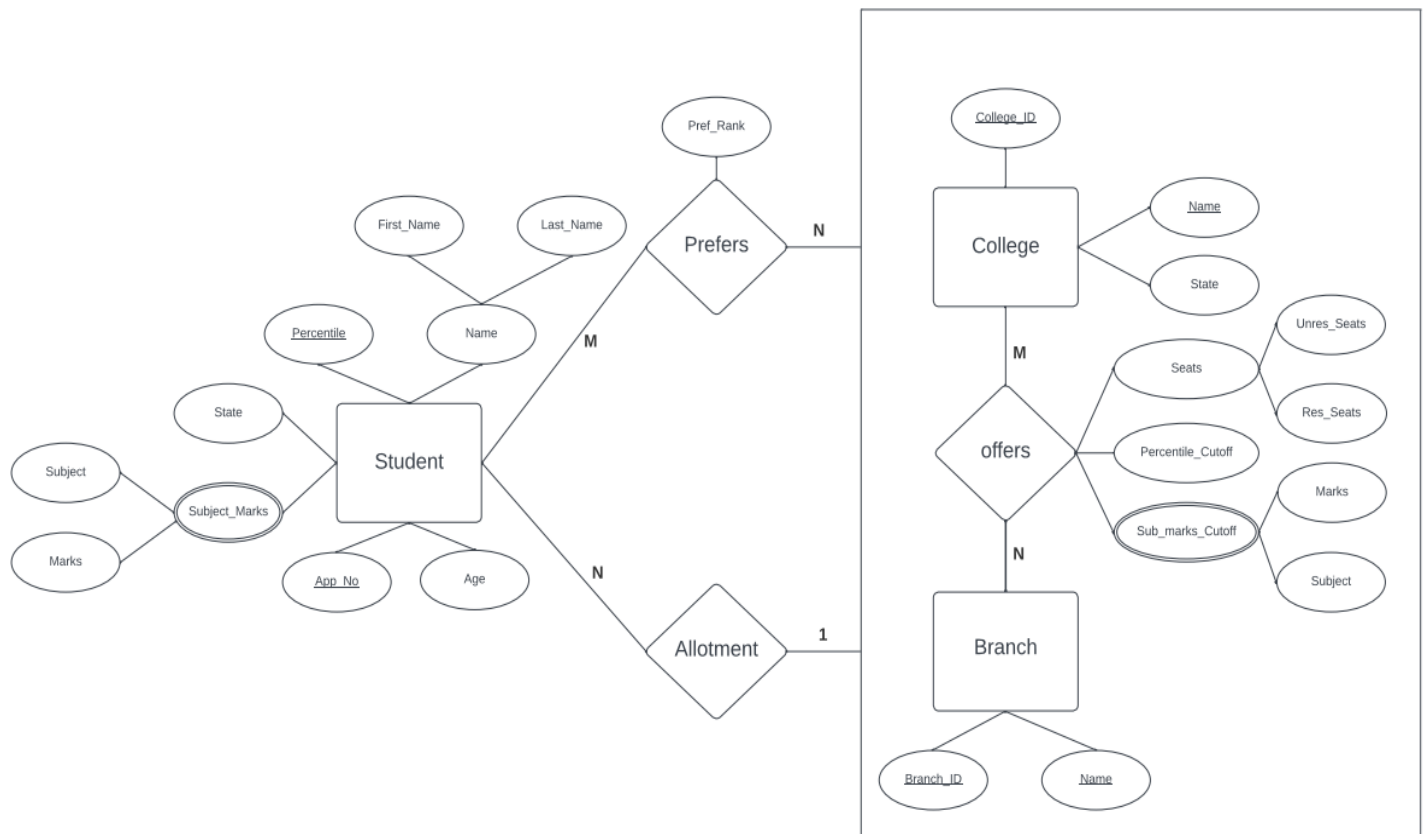
Video Drive Link -

https://drive.google.com/drive/folders/1OVWHxULPZ5ZelT6osBOssR90j4bAwQxs?usp=share_link

Documentation

I. ER Diagram: Following is the ER diagram for the College Allotment Database System.

Note: Aggregation is used in ER diagram. This is explained later.



1. Explanation of Entities: This part of the documentation will discuss each entity in detail.

a. Student: This entity stores the details of each student.

Attribute include:

- i. **name:** Stores the name of the student. This is a composite attribute composed of simple attribute namely first_name and last_name.
- ii. **app_no:** Stores the application number of the student.
- iii. **age:** Stores the age of the student. (**Constraints:** Age should be greater than 17).

- iv. **percentile:** Stores the percentile of the student in a national level examination for college allocation. **Remember:** Percentile by its nature of calculation cannot be same for two individuals.
- v. **state:** Stores the state to which student belongs.
- vi. **subject_marks:** This is a complex attribute. This stores the percentage marks of the all the subject undertaken by the student in class 12th. It consists of simple attribute subject and marks. This is a multivalued attribute.

Super Keys: app_no

Note: More constraints are mentioned in relational schema.

- b. **College:** This entity stores the details of the college.

Attribute include:

- i. **college_id:** Stores the unique id of the college.
- ii. **name:** Stores the unique name of the college (Name of the college is unique).
- iii. **state:** Stores the state in which college is located.

Super Keys: college_id, name.

- c. **branch:** This entity stores the details of the branch that college can offer.

Attribute include:

- i. **branch_id:** Stores the unique id of the branch.
- ii. **name:** Stores the unique name of the branch (Name of the college is unique).

Super Keys: branch_id, name.

- 2. **Explanation of the relationships:** This part of the documentation will discuss each relationship in detail.

- a. **offers:** This relationship shows which branches are offered by college.

Descriptive attribute include:

- i. **Seats:** This is a composite attribute storing total seats offered by a college for a particular branch. The simple attribute includes res_seats which stores number of seats reserved for domicile students and unres_seats which stores number of seats available for general category.
- ii. **percentile_cutoff:** This attribute stores the percentile required to get a particular branch in a college.
- iii. **sub_marks_cutoff:** This is a complex attribute. This attribute stores the cut-off for subjects (in percentage) for a particular branch in a college. This is a multivalued and composite attribute. The simple attribute includes subject and marks.

Cardinality constraints: This is M: N relationship as a college can offer multiple branches and a branch can be offered by multiple colleges.

Participation Constraint: A college may not offer any branch so there is partial participation from college side. Similarly, a branch may not be offered by any college. So, there is partial participation from branch as well.

- b. **prefers:** This relationship stores which colleges and branches is preferred by a student. The relationship utilizes aggregation so that student can choose only those branches that are offered by college in which they are seeking admission.

Descriptive attribute include:

- i. **pref_rank:** This attribute stores the priority of the preference entered by the student.

Cardinality constraints: This is M: N relationship as a student can prefer multiple branches & college combination and same branch & college combination can be preferred by multiple students.

Participation Constraint: A student may not prefer any branch (in which case no allotment will occur) so there is partial participation from student side. Similarly, a branch and college combination may not be preferred by any student. So, there is partial participation from other side as well.

- c. **allotment:** This relationship stores the college and branch allotted to a student. The relationship utilizes aggregation so that student can get only those branches that are offered by college in which they are seeking admission.

Cardinality constraints: This is N: 1 relationship as a student can get only one branch & college combination but same branch & college combination can be allotted to multiple students.

Participation Constraint: A student may not get any branch so there is partial participation from student side. Similarly, a branch and college combination may not be allotted to any student. So, there is partial participation from other side as well.

3. Conversion of ER to Relational Schema: Following steps were followed to convert ER diagram to relational schema.

- a. All the entity set is converted into relations. Ignore all complex attributes. Include all the other attributes except composite attribute into the relation. Include the simple attribute of the composite attribute into relation. Results after this conversion:
 - i. **student** (app_no, first_name, last_name, state, age, percentile).
 - ii. **college** (college_id, name, state).
 - iii. **branch** (branch_id, name).
- b. Since there is no (binary, total participation and non – aggregated) relationship with cardinality N: 1, create separate relations for all the relationship in ER diagram. Include primary key of all participating entities into the relation as its primary key. Ignore all complex attributes. Include all the other descriptive attributes except composite attribute into the relation. Include the simple attribute of the descriptive composite attribute into relation. Results after this conversion:
 - i. **offers** (college_id, branch_id, res_seats, unres_seats, percentile_cutoff, res_alloted, unres_alloted).
 - ii. **prefers** (app_no, college_id, branch_id, pref_rank).
 - iii. **allotment** (app_no, college_id, branch_id).

Note: It is not necessary that union of primary keys of all participating entities act as **primary key** of relation always but a subset of these keys would always. Eg: allotment

c. Create relations for complex attributes and multivalued attributes. Results after this conversion:

i. sub_marks_cutoff attribute of offer relationship is changed into relation:

sub_cutoff (college_id, branch_id, subject, marks)

ii. subject_marks attribute of student is converted into relation:

sub_marks (app_no, subject, marks)

4. Functional Dependencies and Normalization: This section talks about the conversion of relational schema into 3 NF. We will discuss each relation in detail.

Note: Additional constraints in relational schema is written in next part

a. **student** (app_no, first_name, last_name, state, age, percentile)

Candidate keys: app_no, percentile

Remember: Percentile by its nature of calculation cannot be same for two individuals.

Primary Key: app_no

Functional dependencies include:

i. app_no → app_no, first_name, last_name, state, age, percentile

ii. percentile → app_no, first_name, last_name, state, age, percentile

1NF: Meets the definition of a relation

2NF: No partial Key dependencies

3NF: No transitive dependency.

Relation is already in 3NF.

b. college (college_id, name, state)

Candidate keys: college_id, name

Primary Key: college_id

Functional dependencies include:

- i. college_id \rightarrow college_id, name, state
- ii. name \rightarrow college_id, name, state

1NF: Meets the definition of a relation

2NF: No partial Key dependencies

3NF: No transitive dependency.

Relation is already in 3NF.

c. branch (branch_id, name).

Candidate keys: branch_id, name

Primary Key: branch_id

Functional dependencies include:

- i. branch_id \rightarrow branch_id, name
- ii. name \rightarrow branch_id, name

1NF: Meets the definition of a relation

2NF: No partial Key dependencies

3NF: No transitive dependency.

Relation is already in 3NF.

d. offers (college_id, branch_id, res_seats, unres_seats, percentile_cutoff, res_alloted, unres_alloted)

Candidate keys: (branch_id, college_id)

Primary Key: (branch_id, college_id)

Functional dependencies include:

- i. branch_id, college_id \rightarrow college_id, branch_id, res_seats, unres_seats, percentile_cutoff, res_alloted, unres_alloted

1NF: Meets the definition of a relation

2NF: No partial Key dependencies

3NF: No transitive dependency.

Relation is already in 3NF.

e. prefers (app_no, college_id, branch_id, pref_rank).

Candidate keys: (app_no, branch_id, college_id), (app_no, pref_rank)

Note: It is not necessary to make candidate key with minimum attributes as primary key.

Primary Key: (app_no, branch_id, college_id)

Functional dependencies include:

- i. app_no, branch_id, college_id \rightarrow app_no, branch_id, college_id, pref_rank
- ii. app_no, pref_rank \rightarrow app_no, branch_id, college_id, pref_rank

1NF: Meets the definition of a relation

2NF: No partial Key dependencies

3NF: No transitive dependency.

Relation is already in 3NF.

f. allotment (app_no, college_id, branch_id)

Candidate keys: app_no

Primary Key: app_no

Functional dependencies include:

- i. app_no \rightarrow app_no, branch_id, college_id

1NF: Meets the definition of a relation

2NF: No partial Key dependencies

3NF: No transitive dependency.

Relation is already in 3NF.

g. sub_cutoff (college_id, branch_id, subject, marks)

Candidate keys: (college_id, branch_id, subject)

Primary Key: (college_id, branch_id, subject)

Functional dependencies include:

- i. college_id, branch_id, subject \rightarrow college_id, branch_id, subject, marks

1NF: Meets the definition of a relation

2NF: No partial Key dependencies

3NF: No transitive dependency.

Relation is already in 3NF.

h. sub_marks (app_no, subject, marks)

Candidate keys: (app_no, subject)

Primary Key: (app_no, subject)

Functional dependencies include:

- i. app_no, subject \rightarrow app_no, subject, marks

1NF: Meets the definition of a relation

2NF: No partial Key dependencies

3NF: No transitive dependency.

Relation is already in 3NF.

II. Relational Schema:

- 1. Constraints and description of relational schema:** This part discuss the final relational schema and its various constraints.

a. student (app_no, first_name, last_name, state, age, percentile)

Candidate keys: app_no, percentile

Primary Key: app_no

Other constraints: age \geq 18

b. college (college_id, name, state)

Candidate keys: college_id, name

Primary Key: college_id

c. branch (branch_id, name).

Candidate keys: branch_id, name

Primary Key: branch_id

d. offers (college_id, branch_id, res_seats, unres_seats, percentile_cutoff, res_alloted, unres_alloted)

Candidate keys: (branch_id, college_id)

Primary Key: (branch_id, college_id)

Other constraint: percentile_cutoff ≥ 0 and ≤ 100

e. prefers (app_no, college_id, branch_id, pref_rank).

Candidate keys: (app_no, branch_id, college_id), (app_no, pref_rank)

Primary Key: (app_no, branch_id, college_id)

Foreign key include:

- i. app_no references student(app_no)
- ii. (college_id, branch_id) references offers (college_id, branch_id)

f. allotment (app_no, college_id, branch_id)

Candidate keys: app_no

Primary Key: app_no

Foreign key include:

- i. app_no references student(app_no)
- ii. (college_id, branch_id) references offers (college_id, branch_id)

g. sub_cutoff (college_id, branch_id, subject, marks)

Candidate keys: (college_id, branch_id, subject)

Primary Key: (college_id, branch_id, subject)

Foreign key include:

i. (college_id, branch_id) references offers (college_id, branch_id)

Other constraints: marks ≥ 0 and ≤ 100

h. sub_marks (app_no, subject, marks)

Candidate keys: (app_no, subject)

Primary Key: (app_no, subject)

Foreign key include:

i. (app_no) references student (app_no)

Other constraints: marks ≥ 0 and ≤ 100

III. Queries: The following section contains the sample queries, their implementation and outputs.

1. Create all the necessary tables such as student table, college table, allotment table etc.

Query: All the queries creating tables are present in Appendix A.

✓	26	23:27:55	DROP DATABASE IF EXISTS allotment	8 row(s) affected	0.062 sec
✓	27	23:27:55	CREATE DATABASE 'allotment' /*!40100 DEFAULT CHARACTER SET utf8mb4...	1 row(s) affected	0.0012 sec
✓	28	23:27:55	USE allotment	0 row(s) affected	0.00014 sec
✓	29	23:27:55	CREATE TABLE student (first_name VARCHAR(20) NOT NULL, last_name...	0 row(s) affected	0.0060 sec
✓	30	23:27:55	CREATE TABLE sub_marks (app_no INT UNSIGNED, subject VARCHAR(20)...	0 row(s) affected	0.0030 sec
✓	31	23:27:55	CREATE TABLE college (college_id INT UNSIGNED, name VARCHAR(50)...	0 row(s) affected	0.0050 sec
✓	32	23:27:55	CREATE TABLE branch (branch_id VARCHAR(5), name VARCHAR(50) NO...	0 row(s) affected	0.014 sec
✓	33	23:27:55	CREATE TABLE offers (college_id INT UNSIGNED, branch_id VARCHAR(5)...	0 row(s) affected	0.0043 sec
✓	34	23:27:55	CREATE TABLE sub_cutoff (branch_id VARCHAR(5), college_id INT UNSI...	0 row(s) affected	0.0030 sec
✓	35	23:27:55	CREATE TABLE prefers (app_no INT UNSIGNED, college_id INT UNSIGNE...	0 row(s) affected	0.0038 sec
✓	36	23:27:55	CREATE TABLE allotment (app_no INT UNSIGNED, college_id INT UNSIG...	0 row(s) affected	0.0035 sec
✓	37	08:46:02	DROP DATABASE IF EXISTS allotment	8 row(s) affected	0.052 sec
✓	38	08:46:02	CREATE DATABASE 'allotment' /*!40100 DEFAULT CHARACTER SET utf8mb4...	1 row(s) affected	0.0010 sec
✓	39	08:46:02	USE allotment	0 row(s) affected	0.000091 sec
✓	40	08:46:02	CREATE TABLE student (first_name VARCHAR(20) NOT NULL, last_name...	0 row(s) affected	0.0050 sec
✓	41	08:46:02	CREATE TABLE sub_marks (app_no INT UNSIGNED, subject VARCHAR(20)...	0 row(s) affected	0.0026 sec
✓	42	08:46:02	CREATE TABLE college (college_id INT UNSIGNED, name VARCHAR(50)...	0 row(s) affected	0.0022 sec
✓	43	08:46:02	CREATE TABLE branch (branch_id VARCHAR(5), name VARCHAR(50) NO...	0 row(s) affected	0.0022 sec
✓	44	08:46:02	CREATE TABLE offers (college_id INT UNSIGNED, branch_id VARCHAR(5)...	0 row(s) affected	0.0040 sec
✓	45	08:46:02	CREATE TABLE sub_cutoff (branch_id VARCHAR(5), college_id INT UNSI...	0 row(s) affected	0.0034 sec
✓	46	08:46:02	CREATE TABLE prefers (app_no INT UNSIGNED, college_id INT UNSIGNE...	0 row(s) affected	0.0039 sec
✓	47	08:46:02	CREATE TABLE allotment (app_no INT UNSIGNED, college_id INT UNSIG...	0 row(s) affected	0.0037 sec

Output: The output is the creation of tables.

2. Insert a new student record.

Query: `INSERT INTO student VALUES ('Jash','Ranipa',1635,'Gujarat',20,94);`

Output:

	first_name	last_name	app_no	state	age	percentile
►	Kshitij	Tandon	1000	Uttar Pradesh	21	98.70
	Adarsh	Goel	1135	Uttar Pradesh	21	99.71
	Chinmay	Anand	1200	Jharkhand	21	99.00
	Arjoo	Kumari	1234	Rajasthan	20	99.50
	Praneet	Karna	1300	Bagmati	21	91.10
	Vedant	Bansal	1400	Punjab	20	76.00
	Ashwin	Kothari	1435	Karnataka	19	70.00
	Rudresh	Patel	1500	Gujarat	19	85.00
	Jaysheel	Shah	1535	Gujarat	23	90.00
	Devashish	Siwatch	1600	Haryana	21	92.00
	NULL	NULL	NULL	NULL	NULL	NULL



	first_name	last_name	app_no	state	age	percentile
►	Kshitij	Tandon	1000	Uttar Pradesh	21	98.70
	Adarsh	Goel	1135	Uttar Pradesh	21	99.71
	Chinmay	Anand	1200	Jharkhand	21	99.00
	Arjoo	Kumari	1234	Rajasthan	20	99.50
	Praneet	Karna	1300	Bagmati	21	91.10
	Vedant	Bansal	1400	Punjab	20	76.00
	Ashwin	Kothari	1435	Karnataka	19	70.00
	Rudresh	Patel	1500	Gujarat	19	85.00
	Jaysheel	Shah	1535	Gujarat	23	90.00
	Devashish	Siwatch	1600	Haryana	21	92.00
	Jash	Ranipa	1635	Gujarat	20	94.00
	NULL	NULL	NULL	NULL	NULL	NULL

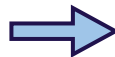
3. Delete a student record.

Query: `CALL delete_student(1635);`

The procedure is present in Appendix B.

Output:

	first_name	last_name	app_no	state	age	percentile
►	Kshitij	Tandon	1000	Uttar Pradesh	21	98.70
	Adarsh	Goel	1135	Uttar Pradesh	21	99.71
	Chinmay	Anand	1200	Jharkhand	21	99.00
	Arjoo	Kumari	1234	Rajasthan	20	99.50
	Praneet	Karna	1300	Bagmati	21	91.10
	Vedant	Bansal	1400	Punjab	20	76.00
	Ashwin	Kothari	1435	Karnataka	19	70.00
	Rudresh	Patel	1500	Gujarat	19	85.00
	Jaysheel	Shah	1535	Gujarat	23	90.00
	Devashish	Siwatch	1600	Haryana	21	92.00
	Jash	Ranipa	1635	Gujarat	20	94.00
	Poojan	Gandhi	1700	Gujarat	20	91.00
	NULL	NULL	NULL	NULL	NULL	NULL



	first_name	last_name	app_no	state	age	percentile
►	Kshitij	Tandon	1000	Uttar Pradesh	21	98.70
	Adarsh	Goel	1135	Uttar Pradesh	21	99.71
	Chinmay	Anand	1200	Jharkhand	21	99.00
	Arjoo	Kumari	1234	Rajasthan	20	99.50
	Praneet	Karna	1300	Bagmati	21	91.10
	Vedant	Bansal	1400	Punjab	20	76.00
	Ashwin	Kothari	1435	Karnataka	19	70.00
	Rudresh	Patel	1500	Gujarat	19	85.00
	Jaysheel	Shah	1535	Gujarat	23	90.00
	Devashish	Siwatch	1600	Haryana	21	92.00
	Poojan	Gandhi	1700	Gujarat	20	91.00
	NULL	NULL	NULL	NULL	NULL	NULL

4. Insert a new college record.

Query: `INSERT INTO college VALUES (170,'Hyderabad University', 'Telangana');`

Output:

college_id	name	state
▶ 111	Indian Institute of Technology, Madras	Tamil Nadu
112	Indian Institute of Technology, Bombay	Maharastra
113	Indian Institute of Technology, Delhi	Delhi
120	Birla Institute of Technology, Pilani	Rajasthan
121	Birla Institute of Technology, Goa	Goa
122	Birla Institute of Technology, Hyderabad	Telangana
130	Indian Institute of Technology, Gandhinagar	Gujarat
141	Lucknow University	Uttar Pradesh
151	Punjab University	Punjab
184	Thapar Institute of Technology	Punjab
NULL	NULL	NULL




college_id	name	state
▶ 111	Indian Institute of Technology, Madras	Tamil Nadu
112	Indian Institute of Technology, Bombay	Maharastra
113	Indian Institute of Technology, Delhi	Delhi
120	Birla Institute of Technology, Pilani	Rajasthan
121	Birla Institute of Technology, Goa	Goa
122	Birla Institute of Technology, Hyderabad	Telangana
130	Indian Institute of Technology, Gandhinagar	Gujarat
141	Lucknow University	Uttar Pradesh
151	Punjab University	Punjab
170	Hyderabad University	Telangana
184	Thapar Institute of Technology	Punjab
NULL	NULL	NULL

5. Allocate a seat to a student based on their preference and eligibility criteria.

Query: `CALL allot(1135);` # The procedure is present in Appendix B.

Output:

app_no	college_id	branch_id
▶ NULL	NULL	NULL



app_no	college_id	branch_id
▶ 1135	113	cse
NULL	NULL	NULL

Note: Before the execution of other queries, we called the procedure `allot_all` present in `Proj_Extra.sql` file which allotted seats to all students according to their rank based on percentile.

6. Retrieve a list of all students who have been allotted seats in a particular college and course.

Query: `CALL allotted_students(113,'cse');`

The procedure is present in Appendix B.

Output:

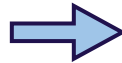
Application...	Student Name
▶ 1135	Adarsh Goel
1300	Praneet Karna

7. Update a student's information.

Query: `UPDATE student SET state = 'Haryana' WHERE (app_no = 1234);`

Output:

	first_name	last_name	app_no	state	age	percentile
▶	Kshitij	Tandon	1000	Uttar Pradesh	21	98.70
	Adarsh	Goel	1135	Uttar Pradesh	21	99.71
	Chinmay	Anand	1200	Jharkhand	21	99.00
	Arjoo	Kumari	1234	Rajasthan	20	99.50
	Praneet	Karna	1300	Bagmati	21	91.10
	Vedant	Bansal	1400	Punjab	20	76.00
	Ashwin	Kothari	1435	Karnataka	19	70.00
	Rudresh	Patel	1500	Gujarat	19	85.00
	Jaysheel	Shah	1535	Gujarat	23	90.00
	Devashish	Siwatch	1600	Haryana	21	92.00
	Poojan	Gandhi	1700	Gujarat	20	91.00
	NULL	NULL	NULL	NULL	NULL	NULL



	first_name	last_name	app_no	state	age	percentile
▶	Kshitij	Tandon	1000	Uttar Pradesh	21	98.70
	Adarsh	Goel	1135	Uttar Pradesh	21	99.71
	Chinmay	Anand	1200	Jharkhand	21	99.00
	Arjoo	Kumari	1234	Haryana	20	99.50
	Praneet	Karna	1300	Bagmati	21	91.10
	Vedant	Bansal	1400	Punjab	20	76.00
	Ashwin	Kothari	1435	Karnataka	19	70.00
	Rudresh	Patel	1500	Gujarat	19	85.00
	Jaysheel	Shah	1535	Gujarat	23	90.00
	Devashish	Siwatch	1600	Haryana	21	92.00
	Poojan	Gandhi	1700	Gujarat	20	91.00
	NULL	NULL	NULL	NULL	NULL	NULL

8. Update a student's preference.

Query: `UPDATE prefers SET branch_id = 'ece'`
`WHERE (pref_rank = 3 AND app_no = 1234);`

Output:

	app_no	college_id	branch_id	pref_rank
	1135	113	cse	1
	1135	120	ece	2
	1135	184	cse	3
	1200	113	cse	1
	1200	184	cse	2
	1200	112	phy	3
	1234	113	cse	1
	1234	112	eee	2
	1234	120	me	3
	1300	151	ece	1
	1300	120	ece	2
	1300	113	cse	3
	1400	184	cse	1
	1400	113	cse	2
	1400	151	math	3
	1435	151	ece	1



	app_no	college_id	branch_id	pref_rank
▶	1000	122	ece	1
	1000	141	cse	2
	1135	113	cse	1
	1135	120	ece	2
	1135	184	cse	3
	1200	113	cse	1
	1200	184	cse	2
	1200	112	phy	3
	1234	113	cse	1
	1234	112	eee	2
	1234	120	ece	3
	1300	151	ece	1
	1300	120	ece	2
	1300	113	cse	3
	1400	184	cse	1

9. Update a college's eligibility criteria.

In Query A we change percentile cutoff of eligibility criteria and in Query B we change subject cutoffs of eligibility criteria.

Query A: `UPDATE offers SET percentile_cutoff = 93.00`
`WHERE (college_id = 113);`

Output A:

	college_id	branch_id	res_seats	unres_seats	percentile_cut...	res_alloted	unres_alloted
▶	111	math	1	1	65.00	1	1
	111	mne	1	1	60.00	1	1
	112	eee	1	2	90.00	0	1
	112	phy	1	1	70.00	0	1
	113	cse	1	2	93.00	1	2
	113	eee	1	1	93.00	1	1
	120	ece	1	1	92.00	0	0
	120	me	1	1	92.00	0	0
	121	bio	1	1	50.00	0	1
	121	cse	1	1	90.00	1	0
	122	ece	1	1	99.00	0	0
	122	phy	1	1	70.00	0	0
	130	math	1	1	75.00	0	2
	141	ce	1	1	45.00	0	0
	141	cse	1	1	50.00	0	1



	college_id	branch_id	res_seats	unres_seats	percentile_cut...	res_alloted	unres_alloted
▶	111	math	1	1	65.00	1	1
	111	mne	1	1	60.00	1	1
	112	eee	1	2	90.00	0	1
	112	phy	1	1	70.00	0	1
	113	cse	1	2	90.00	1	2
	113	eee	1	1	90.00	1	1
	120	ece	1	1	92.00	0	0
	120	me	1	1	92.00	0	0
	121	bio	1	1	50.00	0	1
	121	cse	1	1	90.00	1	0
	122	ece	1	1	99.00	0	0
	122	phy	1	1	70.00	0	0
	130	math	1	1	75.00	0	2
	141	ce	1	1	45.00	0	0
	141	cse	1	1	50.00	0	1

Query B: `UPDATE sub_cutoff SET subject = 'Physics'`

`WHERE (branch_id = 'cse')`

`AND (college_id = 113)`

`AND (subject = 'Mathematics');`

Output B:

	branch_id	college_id	subject	marks
▶	cse	113	Computer Science	95.00
	cse	113	Mathematics	75.00
	cse	141	English Literature	50.00
	cse	141	Physics	80.00
	cse	184	Mathematics	85.00
	cse	184	Physics	75.00
	cse	190	Mathematics	78.00
	ece	122	Chemistry	50.00
	ece	122	Physics	94.00
	eco	184	Economics	90.00
	eee	112	Mathematics	90.00
	math	130	Mathematics	75.00
	math	151	Chemistry	75.00
	math	151	Mathematics	70.00
	me	120	Chemistry	75.00



	branch_id	college_id	subject	marks
	cse	113	Computer Science	95.00
	cse	113	Physics	75.00
	cse	141	English Literature	50.00
	cse	141	Physics	80.00
	cse	184	Mathematics	85.00
	cse	184	Physics	75.00
	cse	190	Mathematics	78.00
	ece	122	Chemistry	50.00
	ece	122	Physics	94.00
	eco	184	Economics	90.00
	eee	112	Mathematics	90.00
	math	130	Mathematics	75.00
	math	151	Chemistry	75.00
	math	151	Mathematics	70.00
	me	120	Chemistry	75.00

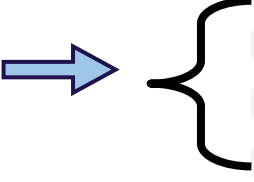
10. Retrieve a count of the number of applications received for a particular college and course.

Query: `CALL application_count(113,'cse');`

The procedure is present in Appendix B.

Output:

No. of Applications	
▶ 6	



app_no	college_id	branch_id	pref_rank
▶ 1700	111	math	1
1234	112	eee	2
1600	112	eee	1
1200	112	phy	3
1135	113	cse	1
1200	113	cse	1
1234	113	cse	1
1300	113	cse	3
1400	113	cse	2
1600	113	cse	2
1135	120	ece	2
1234	120	ece	3
1300	120	ece	2
1535	120	me	1
1435	121	bio	2

11. Retrieve a list of all colleges that have filled all their seats.

Query: `CALL filled_seats();`

The procedure is present in Appendix B.

Output:

College with Filled Seats	
▶ 113: Indian Institute of Technology, Delhi	
111: Indian Institute of Technology, Madras	

12. Retrieve a count of the number of seats available for a particular course in all colleges.

Query: `CALL seats_available('cse');`

The procedure is present in Appendix B.

Output:

	Total Available Domicile Reserved Seats	Total Available Unreserved Seats	Total Available Seats
▶	2	2	4

APPENDIX A

CREATE TABLE student

```
(  
    first_name VARCHAR(20) NOT NULL,  
    last_name VARCHAR(20),  
    app_no INT UNSIGNED,  
    state VARCHAR(30),  
    age INT UNSIGNED CHECK(age >= 18) NOT NULL,  
    percentile DECIMAL(5 , 2 ),  
    PRIMARY KEY (app_no),  
    UNIQUE(percentile)  
);
```

CREATE TABLE sub_marks

```
(  
    app_no INT UNSIGNED,  
    subject VARCHAR(20),  
    marks DECIMAL(5 , 2 ) NOT NULL CHECK(marks >= 0 AND marks <=100),  
    PRIMARY KEY (app_no , subject),  
    FOREIGN KEY (app_no)  
        REFERENCES student (app_no) ON DELETE CASCADE );
```

CREATE TABLE college

```
(  
    college_id INT UNSIGNED,  
    name VARCHAR(50) NOT NULL UNIQUE,
```

```
state VARCHAR(30) NOT NULL,  
  
PRIMARY KEY (college_id));
```

CREATE TABLE branch

(
branch_id VARCHAR(5),
name VARCHAR(50) NOT NULL UNIQUE,
PRIMARY KEY (branch_id));

CREATE TABLE offers

(
college_id INT UNSIGNED,
branch_id VARCHAR(5),
res_seats INT UNSIGNED NOT NULL,
unres_seats INT UNSIGNED NOT NULL,
percentile_cutoff DECIMAL(5 , 2) NOT NULL CHECK(percentile_cutoff >=0
AND percentile_cutoff <=100),
res_alloted INT UNSIGNED NOT NULL DEFAULT 0,
unres_alloted INT UNSIGNED NOT NULL DEFAULT 0,
PRIMARY KEY (college_id , branch_id),
FOREIGN KEY (college_id)
REFERENCES college (college_id) ON DELETE CASCADE,
FOREIGN KEY (branch_id)
REFERENCES branch (branch_id) ON DELETE CASCADE);

CREATE TABLE sub_cutoff

(
branch_id VARCHAR(5),
college_id INT UNSIGNED,
subject VARCHAR(20),

marks DECIMAL(5 , 2) NOT NULL CHECK (marks >=0 AND marks <=100),

PRIMARY KEY (branch_id , college_id , subject),

FOREIGN KEY (college_id, branch_id)

REFERENCES offers(college_id,branch_id) ON DELETE CASCADE);

CREATE TABLE prefers

(

app_no INT UNSIGNED,

college_id INT UNSIGNED,

branch_id VARCHAR(5),

pref_rank INT UNSIGNED,

PRIMARY KEY (app_no , college_id , branch_id),

UNIQUE (app_no, pref_rank),

FOREIGN KEY (app_no)

REFERENCES student (app_no) ON DELETE CASCADE,

FOREIGN KEY (college_id, branch_id)

REFERENCES offers (college_id, branch_id) ON DELETE CASCADE);

CREATE TABLE allotment

(

app_no INT UNSIGNED,

college_id INT UNSIGNED NOT NULL,

branch_id VARCHAR(5) NOT NULL,

PRIMARY KEY (app_no),

FOREIGN KEY (app_no)

REFERENCES student (app_no) ON DELETE CASCADE,

FOREIGN KEY (college_id, branch_id)

REFERENCES offers (college_id, branch_id) ON DELETE CASCADE);

APPENDIX B

PROCEDURE allot

This is a lengthy procedure. Refer to the sql file for better alignment.

delimiter \$\$

create procedure allot(IN application int unsigned)

begin

 declare top int unsigned; # declaring a top variable to choose preference one by one of particular applicant

 set top = 1;

 loop1: loop # loop for choosing preference one by one

 # checks if a student has already been allotted seat. If the student is already allocated then return.

 if not isnull((select app_no from allotment where app_no=application)) then

 leave loop1;

 end if;

 # checks if all the preferences have been exhausted or not. If all the preferences are exhausted then return.

 if isnull((select pref_rank from prefers where (pref_rank = top and app_no=application))) then

 leave loop1;

 end if;

 # checks if the student falls under domicile reservation or not,

```

# if not reserved enters the if statement

if isnull((select A.state from

                (select state from student where app_no = application) A

            join

                (select state from college

                    where college_id = (select college_id from
prefers where (pref_rank = top and app_no=application))) B

                where A.state=B.state)) then

# checks if unreserved seats for the given preference are completely
filled or not

# if not fully filled enters the if statement

if not isnull((SELECT unres_seats

                from (select unres_seats

                    from offers

                        where (branch_id=(select branch_id from prefers where (app_no =
application and pref_rank = top))

                            and college_id=(select college_id from
prefers where (app_no = application and pref_rank = top)))) A

                join

                    (select unres_alloted

                        from offers

                            where (branch_id=(select branch_id from prefers where (app_no =
application and pref_rank = top))

                                and college_id=(select college_id from
prefers where (app_no = application and pref_rank = top)))) B

```



```

where A.unres_seats>B.unres_alloted)) then

# checks if the student satisfies the percentile cutoff for the branch

if not isnull((select A.percentile from
                                (select percentile from student where
app_no=application) A
                                join
                                (select percentile_cutoff from offers
                                where (branch_id=(select branch_id from
prefers where (app_no = application and pref_rank = top))
                                and college_id=(select
college_id from prefers where (app_no = application and pref_rank = top)))) B
                                where
A.percentile>=B.percentile_cutoff)) then

# checks if the student clears the eligibility of subject cutoffs

# if clears the eligibility then allots the seats, updates the no of allotted seats
and leaves the loop

if not isnull((select alpha.A from
                                (select count(*) as A from
sub_cutoff where (
                                branch_id=(select branch_id from prefers where (app_no = application
and pref_rank = top))
                                and college_id=(select college_id from prefers where (app_no =
application and pref_rank = top)))) alpha

```

```

join
(select count(*) as B from

(select * from sub_cutoff where (

branch_id=(select branch_id from prefers where (app_no = application
and pref_rank = top))

and college_id=(select college_id from prefers where (app_no =
application and pref_rank = top)))) A

join

(select * from sub_marks where (app_no = application)) B

where (A.subject = B.subject and A.marks<=B.marks)) beta

where alpha.A = beta.B))

then

insert into

allotment values (application,

(select college_id from prefers where (app_no =
application and pref_rank = top)),

(select branch_id from prefers where (app_no =
application and pref_rank = top)));

update offers set

unres_alloted = unres_alloted+1

```

```

                                where
(branch_id=(select branch_id from prefers where (app_no = application and pref_rank
= top))

                                and
college_id=(select college_id from prefers where (app_no = application and pref_rank
= top)));

                                leave loop1;

                                else

                                set top = top+1;

                                end if;

                                else

                                set top = top+1;

                                end if;

                                else

                                set top = top+1;

                                end if;

                                else

# checks if reserved seats for the given preference are completely filled or not

# if not fully filled enters the if statement

if not isnull((SELECT res_seats

                                from (select res_seats

                                from offers

                                where (branch_id=(select branch_id from prefers where (app_no =
application and pref_rank = top))

```

and college_id=(select college_id from
prefers where (app_no = application and pref_rank = top)))) A

join

(select res_alloted

from offers

where (branch_id=(select branch_id from prefers where (app_no =
application and pref_rank = top)))

and college_id=(select college_id from
prefers where (app_no = application and pref_rank = top)))) B

where A.res_seats>B.res_alloted)) then

checks if the student clears the percentile cutoff

enters the if statement if it clears

if not isnull((select A.percentile from

(select percentile from student where
app_no=application) A

join

(select percentile_cutoff from offers

where (branch_id=(select branch_id from
prefers where (app_no = application and pref_rank = top)))

and college_id=(select
college_id from prefers where (app_no = application and pref_rank = top)))) B

where

A.percentile>=B.percentile_cutoff)) then

checks if the student clears the eligibility of subject
cutoffs

if clears the eligibility then allots the seats, updates the no of allotted seats
and leaves the loop

if not isnull((select alpha.A from
(select count(*) as A from
sub_cutoff where (

branch_id=(select branch_id from prefers where (app_no = application
and pref_rank = top)))

and college_id=(select college_id from prefers where (app_no =
application and pref_rank = top)))) alpha

join

(select count(*) as B from

(select * from sub_cutoff where (

branch_id=(select branch_id from prefers where (app_no = application
and pref_rank = top)))

and college_id=(select college_id from prefers where (app_no =
application and pref_rank = top)))) A

join

(select * from sub_marks where (app_no = application)) B

where (A.subject = B.subject and A.marks<=B.marks)) beta

where alpha.A = beta.B))

then

```

insert into allotment values
(application,
(select college_id from prefers where (app_no = application
and pref_rank = top)),
(select branch_id from prefers where (app_no = application and
pref_rank = top)));
update offers set unres_alloted =
unres_alloted+1
where (branch_id=(select
branch_id from prefers where (app_no = application and pref_rank = top))
and
college_id=(select college_id from prefers where (app_no = application and pref_rank
= top)));
leave loop1;
else
set top = top+1;
end if;
else
set top = top+1;
end if;

# if the reserved seats are filled checks if the unreserved seats are available
# if yes enter the if statement
elseif not isnull((SELECT unres_seats

```

```

from (select unres_seats

from offers

where (branch_id=(select branch_id from prefers where (app_no =
application and pref_rank = top))

and college_id=(select college_id from
prefers where (app_no = application and pref_rank = top)))) A

join

(select unres_alloted

from offers

where (branch_id=(select branch_id from prefers where (app_no =
application and pref_rank = top))

and college_id=(select college_id from
prefers where (app_no = application and pref_rank = top)))) B

where A.unres_seats>B.unres_alloted)) then

# checks if the student clears the percentile cutoff

# enters the if statement if it clears

if not isnull((select A.percentile from

(select percentile from student where
app_no=application) A

join

(select percentile_cutoff from offers

where (branch_id=(select branch_id from
prefers where (app_no = application and pref_rank = top))

and college_id=(select
college_id from prefers where (app_no = application and pref_rank = top)))) B

```

where

A.percentile>=B.percentile_cutoff)) then

checks if the student clears the eligibility of subject
cutoffs

if clears the eligibility then allots the seats, updates the no of allotted seats
and leaves the loop

if not isnull((select alpha.A from
(select count(*) as A from
sub_cutoff where (

branch_id=(select branch_id from prefers where (app_no = application
and pref_rank = top))

and college_id=(select college_id from prefers where (app_no =
application and pref_rank = top)))) alpha

join

(select count(*) as B from

(select * from sub_cutoff where (

branch_id=(select branch_id from prefers where (app_no = application
and pref_rank = top))

and college_id=(select college_id from prefers where (app_no =
application and pref_rank = top)))) A

join


```

(select * from sub_marks where (app_no = application)) B

where (A.subject = B.subject and A.marks<=B.marks)) beta

where alpha.A = beta.B))

then

insert into allotment values

(application,

(select college_id from prefers where (app_no = application

and pref_rank = top)),

(select branch_id from prefers where (app_no = application and

pref_rank = top)));

update offers set unres_alloted =

unres_alloted+1

where (branch_id=(select

branch_id from prefers where (app_no = application and pref_rank = top))

and

college_id=(select college_id from prefers where (app_no = application and pref_rank

= top)));

leave loop1;

else

set top = top+1;

end if;

else

set top = top+1;

```

```

                end if;

            else

                set top = top+1;

            end if;

        end if;

    end loop loop1;

end $$

delimiter ;

```

Procedure allotted_students

```

delimiter $$

create procedure allotted_students(IN college_no int unsigned, branch_no varchar(5))

begin

select A.app_no as 'Application No.', concat(A.first_name, ' ', A.last_name) as
'Student Name'

        from student A                # Selecting name and application number from join
of student table and subquery

        join

        (select * from allotment where (college_id = college_no and
branch_id=branch_no)) B

# Selecting the rows from allotment table for specific college and branch

        where A.app_no = B.app_no; # Joining the tables when app_no is the same.

end $$

delimiter ;

```

PROCEDURE delete_student

delimiter \$\$

create procedure delete_student(IN student_id int unsigned)

begin

delete from student where (app_no = student_id);

end \$\$

delimiter ;

PROCEDURE application_count

creating procedure

delimiter \$\$

create procedure application_count(IN college_no int unsigned, branch_no
varchar(5))

begin

select count(*) as 'No. of Applications'

from prefers where (college_id=college_no and branch_id=branch_no);

end \$\$

delimiter ;

PROCEDURE filled_seats

create procedure

delimiter \$\$

create procedure filled_seats()

begin

```

select concat(beta.college_id, ': ', beta.name) as 'College with Filled Seats' from

    (select * from

        (select college_id, sum(unres_seats) as total_unres_seats,
sum(res_seats) as total_res_seats, sum(unres_alloted) as total_unres_alloted,
sum(res_alloted) as total_res_alloted

            from offers group by college_id) A    # calculating total seats
alloted and total seats

        where (total_unres_seats=total_unres_alloted and
total_res_seats=total_res_alloted)) alpha    # selecting those colleges with all filled
seats

    join

        college beta

        where alpha.college_id=beta.college_id;

end $$

delimiter ;

```

PROCEDURE seats_available

```

delimiter $$

create procedure seats_available(IN branch_no varchar(5))

begin

# Sum over the seats that are allocated and initially given. Then these values are
subtracted

select sum(res_seats) - sum(res_alloted) as 'Total Available Domicile Reserved Seats',
sum(unres_seats) - sum(unres_alloted) as 'Total Available Unreserved Seats',
sum(res_seats) + sum(unres_seats) - sum(res_alloted) - sum(unres_alloted) as 'Total
Available Seats'

from offers

```

```
where branch_id = branch_no;
```

```
end $$
```

```
delimiter ;
```

