CAMPUS RECRUITMENT MANAGEMENT SYSTEM

PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE

OF

MASTER OF

COMPUTER APPLICATIONS

DEPARTMENT OF MATHEMATICAL AND COMPUTATIONAL

SCIENCES

NATIONAL INSTITUTE OF TECHNOLOGY

KARNATAKA

SURATHKAL, MANGALORE- 575025

AUGUST 2021

SUBMITTED BY :

AMAN SIROHI 204CA005

ATANU PRADHAN 204CA007

SRISHTI 204CA050

YOGITA SHARMA 204CA056

SUBMITTED TO :

MS.DEEPTHI L.

DECLARATION

We hereby declare that the project report entitled "CAMPUS RECRUITMENT MANAGEMENT SYSTEM" which is being submitted to the National Institute of Technology Karnataka, Surathkal in partial fulfilment of the requirements of **Master of Computer Applications** in the department of Mathematical and Computational Sciences, is a bonafide report of the work prepared by us.

Aman Sirohi 204ca005

Atanu Pradhan 204ca007

Srishti 204ca050

Yogita sharma 204ca056

MCA 2ND SEM

NITK,SURATHKAL

ACKNOWLEDGEMENT

We wish to express our profound and sincere gratitude to Ms. Deepthi L , Department of Mathematical and Computational Science NITK, Surathkal, who guided us into the intricacies of this project CAMPUS RECRUITMENT MANAGEMENT SYSTEM

We thank Prof. R. Madhusudhan , Head of the Dept. of Mathematical and Computational Science, NITK, Surathkal for extending their support during the course of this project. Each member is highly grateful to the team who evinced keen interest and invaluable support in the progress and successful completion of this project.

We are indebted to each team member for their constant encouragement, co-operation and help.Words of gratitude are not enough to describe the accommodation and fortitude which they have shown throughout this endeavor.

Ms. Deepthi L

CERTIFICATE

This is to certify that the project report entitled "CAMPUS RECRUITMENT MANAGEMENT SYSTEM" submitted by 'AMAN SIROHI, ATANU PRADHAN, SRISHTI, YOGITA SHARMA' as the record of the work carried out by them is accepted as the Report submission in partial fulfilment of the requirements for Master of Computer Applications in the department of Mathematical and Computational Sciences.

I wish them success in all future endeavours.

Submitted to:

Ms. Deepthi L.

INDEX

CHAPTER 1 :

PROBLEM STATEMENT:

The recruitment system allows the job seekers to apply for the job opportunity. The organization selects the applicants for the particular openings.The organisation selects the Applicant and finally the recruited applicants are informed. This system makes the task of the job seeker easier rather than waiting in queue for enrollment. This also reduces the time consumption for both for the job seeker and organization. If the entire process of 'Recruitment' is done in a manual manner then it would takes several days for the recruitment. Considering the fact that the number of applicants for recruitment is increasing every year, a proper system becomes essential to meet the demand. So the system uses several programming and database techniques to elucidate the work involved in this process,Provide a communication platform between the Applicant and the Organization.

Student provides portal for its details such as name ,birthdate,cgpa ,course,mobile number in order to apply for a particular job opening by the company. Company has to provide its details like name of company, person of contact details ,founder of company ,ceo,company location etc. to register itself in the portal after which the company can create as many job vacancy as it wants . In the job opening details company has to provide job profile ,eligible courses students that can apply along with necessary details like deadline till which students can apply . Every time a student registers itself for a particular job opening, an application is created showing the necessary details and showing if the student of that particular id is selected or not.

A particular student can apply only once for the particular job opening and a company can create as many job openings it wants to recruit students. Based on the requirement ,three sites are identified,(i)Student registers themselves in the portal,(ii)Company registers and creates job openings,(iii)An application is created by student who applies in job openings.

CHAPTER 2.

ACTORS:

People who interact with the database

- Applicants
- Organisation

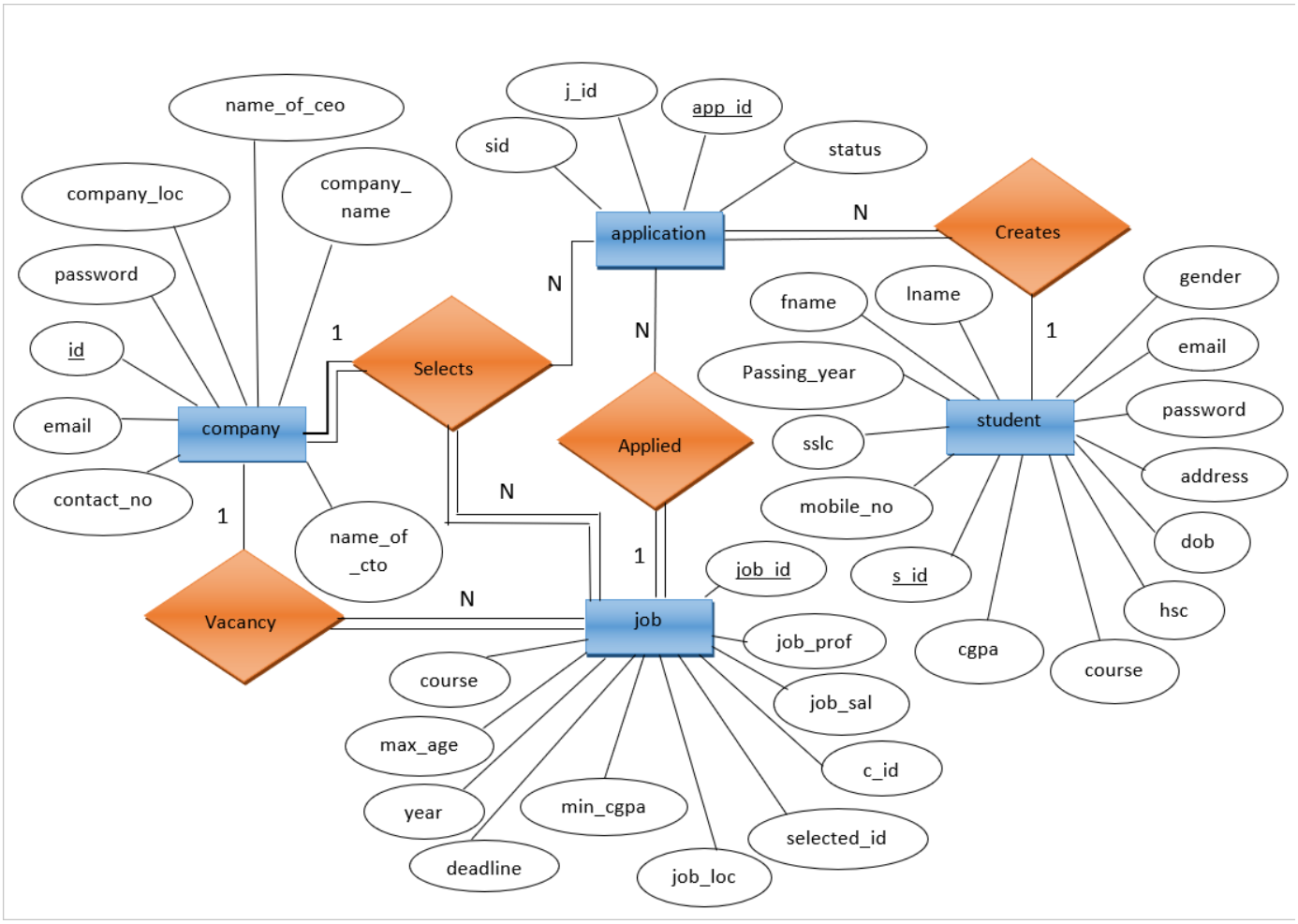
CHAPTER 3.

SOME SAMPLE QUERIES:

- A company can
  - register itself on the portal
  - create vacancy for students to recruit on.
  - view received applications of students.
  - view details of students who have applied.
  - select student for the job.
- A student can
  - register itself on the portal
  - view the vacancies open to apply in.
  - create application to apply in job vacancy.

CHAPTER 4.

EER MODEL (CONCEPTUAL MODEL)



CHAPTER 5.

GLOBAL CONCEPTUAL SCHEMA:

The transformation from the entity-relationship model to the relational model is very straightforward. A feasible set of relational schemas is as follows.

Student

sid	fname	lname	dob	gender	email	address	sslc	hsc	course	cgpa

password	mobile_no	passing_year

Company

id	company_name	email	contact_no	company_loc	password	name_of_ceo

hr	name_of_cto

Job

job_id	job_prof	job_sal	c_id	job_loc	deadline	year

min_cgpa	course	max_age

Application

app_id	j_id	s_id	status

CHAPTER 6.

NORMALIZATION:

Some queries might have to travel with more than one table based on foreign key by some kind of joining. If we have 100's of table then joining kind of operation will take a lot of time. So, it is undesirable. Alternative solution is keeping a universal or central table having all attributes together. It eases our information retrieval but there can be lots duplication or redundant values. **Redundancy** is repeated values in a same table, which leads to wastage of space, example, for 100 employees working for a same department, we have to repeat same department information for all 100 employees in a table. We have huge storage capacity at lower cost nowadays. So, what is the trouble here? anomalies.

Anomalies is a kind of inconsistent information and overhead. Database won't show any error and will simply accept the values. It results to imprecise output, example, from 100 employees in "sales" department, I changed department name to "production" by mistake for 10 employees. Now if I query for list of employees in sales department, it will show only 90 employees details, 10 are left out. Database won't show any error for changing 10 employees detail but it simply accepts. Such kind of inconsistency is called anomaly. It can happen while insert, delete and updating information.

1. **Insert anomaly:** while inserting employee information we might enter wrong department details, which will lead to anomaly.
2. **Delete anomaly:** there are 100 employees in a department "sales". If we delete all 100 employees in that department then we would lose department details too. There is no other way to extract department details of "sales".
3. **Update anomaly:** while updating a table we might enter some other details by mistake, which will lead to anomaly.

These are the overheads, when you combine all tables into a centralized one. So, how can we eliminate them? Solution is dividing tables as smallest as possible. But, how small a table should be? To the level of anomalies wont occurs, ideally table having 2 attributes. But, achieving this will increase querying time. Therefore, how to determine the number of attributes to keep in table which won't lead to large querying time and anomalies?

Idea is dividing large table into small tables having less number of attributes in such a way that your design reduces anomalies and subsequently degree of redundancy that are present in the table. This systematic procedure is called **normalization**.

Normalization is carried out having functional dependency and candidate keys in mind. It is a step-wise process that divides relations into several pieces until we eliminate redundancy and anomalies.

There are many steps to achieve this normalization.

1. 1st normal form
2. 2nd normal form
3. 3rd normal form
4. Boyce-codd normal form

A	B	C
1	c	d
2	a	b
3	e	f

Each follows their own set of rules. Let's discuss with the help of tourism agency databse management system's global conceptual schema.

What is key in general? Given a value of an attribute, we must be able to uniquely identify all other attributes given in a table. That is, a complete row also called tuple or record.

A → BC

→This symbols denotes "determines" or "returns"
(group of attributes) → (group of attributes)

LHS is called key and RHS is called values.

Example, if I say 3 in the attribute A, then it must return BC values.→is called functional dependency. The term "functional" implies group of attributes that determine another group of attributes. We can also say attribute level dependency instead of functional dependency. But, mathematically speaking this kind of representation (LHS → RHS) is called **"functionally dependent".**Therefore, applying mathematics (set theory) on table design helps us dealing attributes. There are 3 different kinds of FD's.

1. **Trivial Functional Dependency (FD):** What is got on RHS is already LHS

Example: A determines itself.

A→A

A→ AB

AB→A

In general form to represent this $X \supseteq Y$. however, there are no useful operation using this kind of dependency.

2. **Non-trivial FD:** given a value of an attribute get a unique value.

Example:

A → B

A → BC

AB → CD

In general, the rule is $X \cap Y = \varphi$

3. **Semi non-trivial FD:** it is not deriving completely new information. It partially brings a record, that is, not the value of entire set of attributes.

Example:

AB → BC B is common on both sides

ACD → EFCD CD is common both sides

It is generally represented as $X \cap Y \neq \varphi$

FD's are quite useful in identifying keys, identifying equivalences of FS's and finding minimal FD set. There are rules applied on FD such as inference rules (reflexive, transitivity, augmentation, union, etc), closure.

Closure – how many attributes you are able to determine by one attribute. It is the base of entire normalization process. Using closure property we can determine candidate keys.

Candidate keys are key or set of keys that identify all other attributes in a table. A table can have many candidate keys, but at any moment, only one candidate key can be as a primary key of a table. If there are n attributes then $2^n - 1$candidate keys are possible except null.

Sometimes, candidate key with non-key attribute uniquely determine a table. Such key is called **superkey.**

Minimal super key or candidate key which has less no of attributes is called **primary key,** which uniquely identifies a tuple.

In FD's, LHS must be a key for every table. In BCNF, we have 0% redundancy in table. To achieve this, we go through series of normalization from 1st NF to BCNF. It is not mandatory to go from 1st NF to BCNF. But, it is a convention to follow this sequence.

Every normal form should be lossless, and FD preserved

1st Normal Form:A relation is said to be in first normal form then it should satisfy the following

- ✓ No multi-valued attribute
- ✓ No composite attribute
- ✓ identify primary key

Here, the relationship is converted to either relation or foreign key or merging relations.
Foreign key: giving primary key of one table as a reference to another table.

Student:

sid	name	dob	gender	email	address	sslc	hsc	course	cgpa	passing_year

password	branch	mobile_no

Company:

id	company_name	email	contact_no	company_loc	password	name_of_ceo

name_of_cto

Job:

job_id	job_prof	job_sal	c_id	job_loc	deadline	year

min_cgpa	course	max_age

Application:

app_id	j_id	s_id	status

Outcome of 1st normalization:

- ✓ Primary key has been identified in each table using closure property (minimal super key)
- ✓ Composite attributes has been resolved
- ✓ Multi-valued attributes has been resolved

2nd Normal Form:

(i) Repeating column values are taken out and maintained in a separate table. So that change can be done only once in the new table rather than all entries in the first table. Rule is foreign key must be on the N side else again multi-value in a column will occur.

(ii) Identify **prime attribute** (part of candidate key that determines anything else), it is also called **partial dependency**, and eliminate it. Because, 2nd NF is based on **Full Functional dependency** (key should determine all other attributes in a table)

Use foreign key on many side

Student:

sid	name	dob	gender	email	address	ssc	hsc	course	cgpa	passing_year

password	mobile_no

Company:

id	company_name	email	contact_no	company_loc	password	name_of_ceo

name_of_cto

Job:

job_id	job_prof	job_sal	c_id	job_loc	deadline	year

min_cgpa	course	max_age

Application:

app_id	j_id	s_id	status

3rd Normal Form:

- ✓ Only columns with direct dependency of the primary key shall be in the entity.
- ✓ No transitive dependencies: non-prime attributes transitively depending on the key. .
- ✓ Example: A→B→C == A → C.

A → B

B is non-key attribute here

B→C suddenly becomes key attribute here. because of this, we will get repeated values in a column. Therefore, it should be eliminated.

- ✓ 3rd NF should hold the condition that: if X → Y then

Either X is a super key

Or Y is a prime attribute

Following this condition will never allow transitive dependency.

Student:

sid	name	dob	gender	email	address	ssc	hsc	course	cgpa	passing_year

password	mobile_no

Company:

cid	c_name	c_email	contact_number	c_loc	c_pswd	ceo	cto

founded_on	founder

Job:

job_id	job_prof	job_salary	selected_id	job_loc	deadline	year

min_cgpa	course	max_age	c_id

Application:

app_id	j_id	s_id	status

Every candidate key in a table determines all other attributes and no non-key attributes determine any attributes in the tables.

CHAPTER 7.

GLOBAL SCHEMA:

students	
Attribute name	**Attribute size (type in bytes)**
sid	Int(11)
fname	varchar(20)
lname	varchar(20)
dob	date
gender	varchar(7)
email	varchar(25)
address	varchar(50)
sslc	int(5)
hsc	int(5)
course	varchar(20)
cgpa	int(5)
passing_year	int(5)
Mobile_no	varchar(15)
password	varchar(30)

company	
Attribute name	**Attribute size (type in bytes)**
id	Int(11)
company_name	varchar(250)
email	varchar(150)
contact_no	varchar(15)
company_loc	varchar(255)
password	varchar(255)
name_of_ceo	varchar(255)
name_of_cto	varchar(255)

application	
Attribute name	**Attribute size (type in bytes)**
app_id	Int(11)
a_job_id	int(11)
a_s_id	int(11)
status	varchar(10)

jobs	
Attribute name	**Attribute size (type in bytes)**
job_id	Int(11)
job_prof	varchar(255)
job_salary	int(11)
c_id	int(11)
job_loc	varchar(255)
deadline	date
year	date
min_cgpa	int(3)
max_age	int(3)

CHAPTER 8.

DATABASE QUERIES:

Database creation Queries :

```
CREATE TABLE students
(
 s_id INT NOT NULL AUTO_INCREMENT,
fname VARCHAR(20) NOT NULL,
lname VARCHAR(20) NOT NULL,
dob DATE NOT NULL,
gender VARCHAR(7) NOT NULL,
email VARCHAR(25) NOT NULL,
password VARCHAR(30) NOT NULL,
mobile_no VARCHAR(15) NOT NULL,
address VARCHAR(50) NOT NULL,
sslc INTEGER(5) NOT NULL,
hsc INTEGER(5) NOT NULL,
cgpa INTEGER(5) NOT NULL,
course VARCHAR(20) NOT NULL,
passing_year INTEGER(5) NOT NULL,
reg_date DATETIME DEFAULT CURRENT_TIMESTAMP
 )
```

```
CREATE TABLE company
(
Company_Name VARCHAR(250) NOT NULL,
Email VARCHAR (150) NOT NULL,
Password VARCHAR (255) NOT NULL,
Contact_No VARCHAR(15) NOT NULL,
Name_Of_Ceo VARCHAR(255),
Name_Of_Cto VARCHAR(255),
Company_loc VARCHAR(255) NOT NULL,
id INT(11) AUTO_INCREMENT,
```

```
Created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);


CREATE TABLE jobs
(
job_id INT AUTO_INCREMENT,
job_prof VARCHAR(255) NOT NULL,
job_sal INT NOT NULL,
 job_loc VARCHAR(255) NOT NULL,
deadline DATE NOT NULL,
year DATE NOT NULL,
min_cgpa INT(3) NOT NULL,
course VARCHAR(255) NOT NULL,
max_age INT(3) NOT NULL,
c_id INT(11)
);

CREATE TABLE application
(
app_id INT AUTO_INCREMENT,
sid INT(11),
j_id INT(11),
status VARCHAR(10) DEFAULT 'NULL',
);
```

ADDING primary keys

```
ALTER TABLE student ADD PRIMARY KEY (sid);

ALTER TABLE company ADD PRIMARY KEY (id);

ALTER TABLE jobs ADD PRIMARY KEY (job_id);

ALTER TABLE application ADD PRIMARY KEY (app_id);
```

Adding foreign key constraints

ALTER TABLE students ADD (FOREIGN KEY (sid) REFERENCES application(s_id));

ALTER TABLE company ADD (FOREIGN KEY (id) REFERENCES jobs(c_id));

ALTER TABLE jobs ADD (FOREIGN KEY (job_id) REFERENCES application(j_id));

Queries that can be executed:

1. **Check list of students with less than 9 CGPA.**

SELECT * FROM students WHERE cgpa < 9;

2. **Obtain details of student with highest hsc marks.**

SELECT * FROM students WHERE s_id IN (SELECT a.s_id FROM students a,students b WHERE a.hsc>b.hsc);

3. **Obtain details of students who applied in job with id 7**.

SELECT * FROM students INNER JOIN application
ON students.s_id=application.sid
WHERE application.j_id=7;

4. **Obtain id of students selected by company id 3;**

SELECT application.s_id FROM application INNER JOIN jobs
ON application.j_id=jobs.job_id
WHERE jobs.c_id=3 AND application.status='Accepted';

5. Obtain number of applications created by student with first name “AMAN”.

```
SELECT COUNT(*) FROM application INNER JOIN students
ON students.s_id=application.sid
WHERE students.fname=”AMAN”;
```

6. Obtain the students id first name last name application id of student who applied in company with id=1 and has maximum marks in 10th std.

```
SELECT students.s_id,students.fname,students.lname,application.app_id FROM students
INNER JOIN application ON students.s_id =application.sid
INNER JOIN jobs ON jobs.job_id=application.j_id
INNER JOIN company ON company.id=jobs.c_id
WHERE students.sslc IN (SELECT MAX(students.sslc) FROM students WHERE
company.id=1);
```

7. Obtain the details of jobs not created by company id 2

```
SELECT * FROM jobs WHERE jobs.c_id NOT IN(SELECT company.id FROM company
WHERE company.id=2);
```

OR

```
SELECT * FROM jobs WHERE jobs.c_id!=2;
```

8. Obtain the list of all the students who are selected by companies

```
SELECT * FROM students INNER JOIN application ON application.app_id=students.s_id
WHERE application.status="Accepted";
```

OR

```
SELECT  *  FROM students WHERE students.s_id IN (SELECT sid FROM application
WHERE status="Accepted");
```

9. List of companies located in place whose name starts with letter B.

SELECT * FROM company WHERE company_loc LIKE 'M%';

OR

SELECT * FROM company WHERE company_loc REGEXP '^M';

10. Give counts of applications received for each job in descending order

SELECT application.j_id,COUNT(*) FROM application GROUP BY j_id ORDER BY j_id DESC;

REFERENCES:

1. **E-R model-GeeksForGeeks**
   https://www.geeksforgeeks.org/introduction-of-er-model/
2. **Form Validation**
   https://www.w3schools.com/php/php_forms.asp
3. **SQL queries**
   https://www.w3resource.com/sql-exercises/
4. **Normalisation**
   https://www.javatpoint.com/dbms-normalization