DBMS Project Report

PES University

Database Management Systems
UE18CS252

Submitted By

PES1201800366 **ADITEYA BARAL**

Automated Resume Generator

A resume or portfolio is an extremely important tool in today's world. It is used by employees and university graduates all over the world to market themselves and display their skills and achievements and serves as a documentation of one's professional life. Whether you are a professional with over 20 years of experience or a freshman from University, having an up to date and polished resume is of utmost importance to help you stand out and be a cut above the competition.

As students who are constantly on the lookout for internships, it is very crucial for us to have a professionally written and laid out resume to attract attention from employers. It helps other employers identify talent and find the right people for their company – people who are skilled and can bring a change in the company. However, many students find it difficult to create a good resume, since building a resume is also an art that must be perfected in today's world. Hence, to solve this issue and help students build professional resumes, we need an **Automated Resume Generator**.

As the name suggests, the application takes in all user details and creates a user profile from the unstructured data it is fed in. It then creates a well-structured resume out of these details by invoking a call to a web app – a Google script linked to a resume template to format the resume and personalise the template for the said user. It finally not only backs up a version of the resume on the cloud, but also emails the candidate a version of their resume. The application also possesses basic features such as modifying of inserted details and displaying a candidate's details.

Index

Introduction	2
Data Model	3
Functional Dependencies and Normalization	7
DDL	11
Triggers	14
SQL Queries	15
Conclusion	19

Introduction

The application was built using Python3 with Microsoft SQL as the backend server for the database to execute transactions. The application has a simple command line interface (CLI) and requires the user to type out their details one after the other. Python's ODBC API, pyodbc was used to perform the SQL transactions and its MIME libraries were used to send emails. The Google backend script was written and deployed as a web app connected to the template which was in turn created on Google Docs. The web app is deployed as a URL containing a link to the google script and a query string containing the details of the users.

Implementation

The application follows a top-down approach, where all the details entered by a user are broken down and stored as individual units where a single unit encapsulated information from other users. This allows for easy storage and retrieval of various fields of data related to a user and allows us to also visualise trends and make comparisons between potential candidates at various levels of data stored

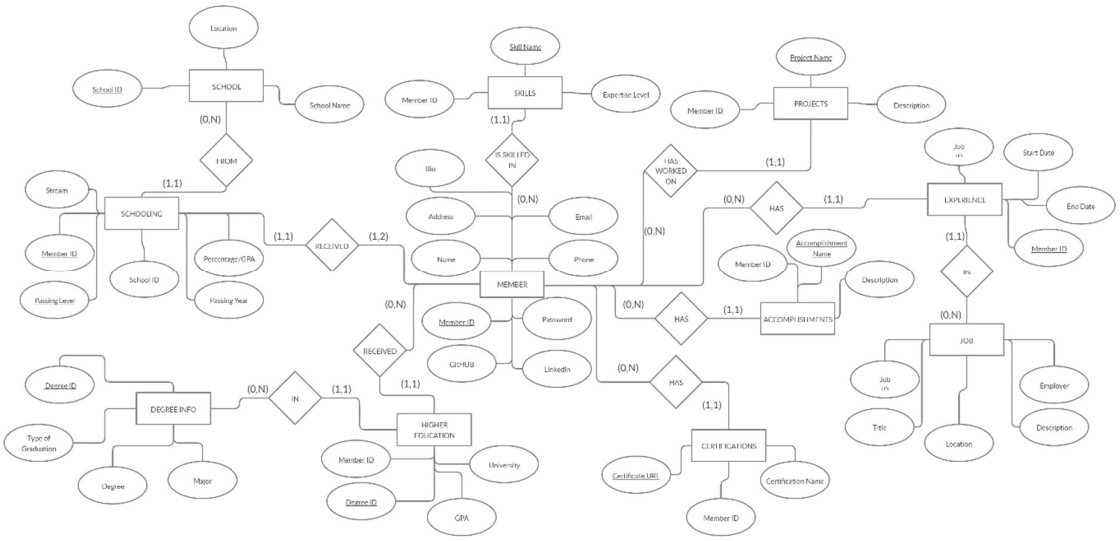
The different categories of information stored are their personal details, schooling, higher education, job experience and non-academic details. A user's personal details are stored as a single tuple. Each user is required to set a password for their account, after which they are assigned a public member ID and a key-value pair of this combination serve as authentication credentials while logging in the next time. The user's schooling which consists of their 10^{th} and 12^{th} grade qualifying examination marks are stored together in the schooling table and are made to refer to a common universal database which holds information about all the schools users have graduated from. Similarly, a user's higher education details are stored with a foreign key reference to a universal university table which holds information about all the universities that exist. If a user's school or university detail is not found in the universal relation, it is simply added to it. Users then must enter their non-academic details, like their experience, the skills they possess and their expertise, the projects they have worked on, their accomplishments and their certifications. These are all again stored as separate relations to ensure high integrity within a relation.

Once all these details have been entered, a user has the option to view these details, edit them or to generate their resume. If a user chooses to view their details, all the tables are joined on the member's unique ID and then their relevant details are displayed. If a user chooses to modify their details, an appropriate function call is made to the individual unit which stores the detail the user wants to modify. Their details are updated, and the

transaction is executed. If a user wishes to generate their resume, a call is made to the web app which activates and runs the Google Script to generate the resume. All the details of the user are passed within a URL to the script, which uses these details to format the resume template already in place and saves a copy of this on Google Drive. It then returns a shareable link to the same resume. An email is then sent to the user on his registered email address with his resume file as well as the shareable link.

Data Model

ER Model



Description of Schema

MEMBER

Stores personal details (name, phone, email etc) of every member along with a password. Every member is assigned a unique ID which acts as a primary key. All the fields store a VARCHAR with suitable size limits.

SCHOOL

Stores all the school IDs, school names and locations, identified uniquely by school ID. All the fields store a VARCHAR with suitable size limits.

SCHOOLING

Stores the schooling received by every user. It stores the member ID, passing level (10^{th} or 12^{th}) and school ID along with stream, percentage and passing year. The primary key is a composite key consisting of the member ID and passing level. (A member can only have two passing levels for schooling - one for 10th and one for 12th). All the fields store a VARCHAR except passing level, passing year and score which store an INT, DATE and FLOAT, respectively.

DEGREE INFO
Stores details of the various degrees that are possible. Each degree is identified with a degree ID. All the fields store a VARCHAR with suitable size limits.

HIGHER EDUCATION
Stores the higher education received by every user. It stores the degree ID, member ID, GPA, and college name. The primary key is a composite key consisting of the member ID and degree ID. (A member can have multiple degrees). All fields store a VARCHAR except GPA which holds a FLOAT.

SKILLS
Stores the skillsets of all users. It stores the member ID, skill name and expertise level (1-10). A combination of the member ID and skill name acts as a primary key. (A member can have multiple skills and multiple members can have the same skill). All fields store a VARCHAR except expertise level which stores a FLOAT.

PROJECTS
Stores all the projects users have worked on. The member ID and the project name act as a primary key. (A member can work on multiple projects and multiple members can have the same project name). All the fields store a VARCHAR with suitable size limits.

JOB
It stores the Job ID, Title, Location, Description and Employer. The Job ID acts as a primary key and every job is assigned a unique ID. All the fields store a VARCHAR with suitable size limits.

EXPERIENCE
It stores the member ID, Job ID, start date and the end date. The primary key is a combination of the member ID and the job ID since multiple members can have the same job. All the fields store a VARCHAR except start and end date which hold a DATE.

ACCOMPLISHMENTS
Stores the member ID, accomplishment name and description of the accomplishment. The primary key is a composite key consisting of member ID and accomplishment name. All the fields store a VARCHAR with suitable size limits.

CERTIFICATIONS
Stored the member ID, certification URL and the certificate name. The certification URL is unique for every individual and acts as the primary key. All the fields store a VARCHAR with suitable size limits.

Choice of Keys

Candidate key is a set of attributes that uniquely identify tuples in a table. It is a super key with no repeated attributes. The primary key should be selected from the candidate keys. Every table must have at least a single candidate key. A table can have multiple candidate keys but only a single primary key.

Properties:

- It must contain unique values.
- Candidate key may have multiple attributes.
- Must not contain null values.
- It should contain minimum fields to ensure uniqueness.
- Uniquely identify each record in a table.

The following are the primary keys for each entity in the database.

MEMBER
MEMBER_ID

Reason: Each user has a unique member ID which is generated every time a new user is created. Hence it is unique and can be used to get all information about the user.

SCHOOL
SCHOOL_ID

Reason: This table generates a school ID for every school with the school's name and location. Since there can only be one school in a location with the same name, it is unique.

SCHOOLING
(MEMBER_ID, PASSING_LEVEL)

Reason: A combination of member ID and passing level are unique.
Since this table stores schooling details of the client, member ID is bound to be repeated i.e. a single member ID will have two entries, one for 10^{th} and the other for 12^{th}.
For instance, it is of the form (MEM#000001, 10), (MEM#000001, 12)

DEGREE INFO
DEGREE_ID

Reason: This table generates a degree ID for every combination of graduation, degree and major which makes it unique.

HIGHER EDUCATION
(MEMBER_ID, DEGREE_ID)

Reason: A combination of member ID and degree ID is unique and can be used to identify any record in this table. A member can have multiple degrees and hence only a composite key can identify a tuple uniquely.

SKILLS
MEMBER_ID, SKILL_NAME

Reason: A combination of member ID and skills is unique since this table stores skill details of all clients. Member ID and skills cannot be independent on their own since one user can possess multiple skills.

PROJECTS
MEMBER_ID, PROJECT_NAME

Reason: A combination of member ID and project name is unique since this table stores project details of all clients. Member ID and project names cannot be independent on their own since one user can work on multiple projects and multiple users can work on the same project.

JOB
JOB_ID

Reason: This table generates a job ID based on job title, job location, job description and employer which makes it unique.

EXPERIENCE
MEMBER_ID, JOB_ID

Reason: This table stores the experience details of all users. A combination of member ID along with job ID is unique and can uniquely identify a row in the table.

ACCOMPLISHMENTS
MEMBER_ID, ACCOMPLISHMENT_NAME

Reason: This table stores the accomplishment details of all users. A combination of member ID along with accomplishment is unique as multiple clients can have the same accomplishment, but one client can have one accomplishment only once. Thus, a combination of the two can be used to identify any record in the table.

CERTIFICATIONS
CERTIFICATION_URL

Reason: Even if multiple clients take the same course, for each of them a unique certification URL is generated by the issuing authority. Hence it can be used to identify any record.

Functional Dependencies and Normalization

Functional Dependencies

MEMBER
MEMBER_ID -> PASSWORD, NAME, EMAIL, PHONE, ADDRESS, BIO, GITHUB, LINKEDIN

SCHOOL
SCHOOL_ID -> LOCATION, NAME

SCHOOLING
MEMBER_ID, PASSING_LEVEL -> SCHOOL_ID, STREAM, PASSING_YEAR, SCORE

DEGREE_INFO
DEGREE_ID -> TYPE_OF_GRADUATION, DEGREE, MAJOR

HIGHER_EDUCATION
MEMBER_ID, DEGREE_ID -> GPA, UNIVERSITY

SKILLS
MEMBER_ID, SKILL_NAME -> EXPERIENCE_LEVEL

PROJECTS
MEMBER_ID, PROJECT_NAME -> DESCRIPTION

JOB
JOB_ID -> TITLE, LOCATION, DESCRIPTION, EMPLOYER

EXPERIENCE
MEMBER_ID, JOB_ID -> START_DATE, END_DATE

ACCOMPLISHMENTS
MEMBER_ID, ACCOMPLISHMENT_NAME -> DESCRIPTION

CERTIFICATIONS
CERTIFICATION_URL -> MEMBER_ID, CERTIFICATION_NAME

Normalization

Normalization is the process of minimizing redundancy from a relation or set of relations. Redundancy in relations may cause insertion, deletion, and update anomalies. It helps minimize that. Normal forms are used to eliminate or reduce redundancy in database tables.

First Normal Form - 1NF

A relation is in first normal form if **every attribute in that relation is singled valued attribute**. Since all the attributes used in the relations are single valued, it thus satisfies the first normal form.

Second Normal Form – 2NF

A relation is in 2NF if it has **no partial dependency and it is in 1NF.** Since most tables have a single key attribute and the ones with a composite key display no inherent dependency, all the relations satisfy the second normal form.

We have,

SCHOOL
SCHOOL_ID -> LOCATION, NAME.

If NAME had been a part of the primary key that is, SCHOOL_ID, NAME -> LOCATION, then it would lead to a violation of the second normal form since NAME->LOCATION would be an underlying inherent dependency.

Third Normal Form – 3NF

A relation is in third normal form if there is **no transitive dependency for non-prime attributes as well as it is in 2NF**. Since all attributes in every relation are independent of each other, transitive relationships are not possible and hence all relations satisfy the third normal form.

The initial model included a DURATION attribute for the relation EXPERIENCE that is,

EXPERIENCE
MEMBER_ID, JOB_ID -> START_DATE, END_DATE, DURATION.

However, this attribute was removed since we have a transitive dependency for non-prime attributes - MEMBER_ID, JOB_ID -> START_DATE, END_DATE and START_DATE, END_DATE -> DURATION this is, it was of the form A -> B, B -> C within the same relation R. Hence, this was removed to prevent any violation of the normal form and to prevent data redundancy.

Testing for Lossless Join

A decomposition is lossless if it is feasible to reconstruct relation R from decomposed tables using natural joins. No information is lost when the relation is decomposed. The natural join would result in the same original relation.

Performing Natural Join

We can get all information of the user by performing a natural join on all tables with the member ID. The screenshots below show that all information related to a user can be obtained and no spurious tuples are generated nor is any information missing. Hence, the decomposition is lossless i.e. no loss of information.

```
SELECT
M.MEMBER_ID, M.NAME, M.ADDRESS, M.PHONE, M.EMAIL,
S.NAME AS SCHOOLNAME, S.LOCATION,
SCH.PASSING_LEVEL, SCH.PASSING_YEAR, SCH.STREAM, SCH.SCORE,
D.DEGREE, D.MAJOR,
HED.UNIVERSITY, HED.GPA,
J.EMPLOYER, J.LOCATION, J.TITLE, J.DESCRIPTION,
E.START_DATE, E.END_DATE,
P.PROJECT_NAME, P.DESCRIPTION,
SK.SKILL_NAME, SK.EXPERIENCE_LEVEL,
A.ACCOMPLISHMENT_NAME, A.DESCRIPTION
FROM MEMBER M
INNER JOIN SCHOOLING SCH ON M.MEMBER_ID = SCH.MEMBER_ID INNER JOIN SCHOOL S ON
SCH.SCHOOL_ID = S.SCHOOL_ID
INNER JOIN HIGHER_EDUCATION HED ON M.MEMBER_ID = HED.MEMBER_ID INNER JOIN
DEGREE_INFO D ON HED.DEGREE_ID = D.DEGREE_ID
INNER JOIN EXPERIENCE E ON M.MEMBER_ID = E.MEMBER_ID INNER JOIN JOB J ON E.JOB_ID
= J.JOB_ID
INNER JOIN PROJECTS P ON M.MEMBER_ID = P.MEMBER_ID
INNER JOIN SKILLS SK ON M.MEMBER_ID = SK.MEMBER_ID
INNER JOIN ACCOMPLISHMENTS A ON M.MEMBER_ID = A.MEMBER_ID;
```

	MEMBER_ID	NAME	ADDRESS	PHONE	EMAIL	SCHOOLNAME	LOCATION	PASSING_LEVEL	PASSING_YEAR	STREAM
1	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science
2	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science
3	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science
4	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science
5	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science
6	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science
7	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science
8	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science
9	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science
10	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science
11	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science
12	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science
13	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science
14	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science
15	MEM#000001	Aditeya Baral	Bangalore	9686041205	aditeya.baral@...	BISHOP COTTO...	RESIDENCY RO...	10	2016	Science

SCORE	DEGREE	MAJOR	UNIVERSITY	GPA	EMPLOYER	LOCATION	TITLE	DESCRIPTION	START_DATE
95	BTECH	CSE	PES UNIVERSITY	8.8	Rubble	Bangalore	Co-Founder	Founded a gar...	2016-12-08
95	BTECH	CSE	PES UNIVERSITY	8.8	Rubble	Bangalore	Co-Founder	Founded a gar...	2016-12-08
95	BTECH	CSE	PES UNIVERSITY	8.8	Rubble	Bangalore	Co-Founder	Founded a gar...	2016-12-08
95	BTECH	CSE	PES UNIVERSITY	8.8	Rubble	Bangalore	Co-Founder	Founded a gar...	2016-12-08
95	BTECH	CSE	PES UNIVERSITY	8.8	Center for Clou...	Bangalore	Research Intern	Decreasing Tail...	2020-05-20
95	BTECH	CSE	PES UNIVERSITY	8.8	Center for Clou...	Bangalore	Research Intern	Decreasing Tail...	2020-05-20
95	BTECH	CSE	PES UNIVERSITY	8.8	Center for Clou...	Bangalore	Research Intern	Decreasing Tail...	2020-05-20
95	BTECH	CSE	PES UNIVERSITY	8.8	Center for Clou...	Bangalore	Research Intern	Decreasing Tail...	2020-05-20
95	BTECH	CSE	PES UNIVERSITY	8.8	Rubble	Bangalore	Co-Founder	Founded a gar...	2016-12-08
95	BTECH	CSE	PES UNIVERSITY	8.8	Rubble	Bangalore	Co-Founder	Founded a gar...	2016-12-08
95	BTECH	CSE	PES UNIVERSITY	8.8	Rubble	Bangalore	Co-Founder	Founded a gar...	2016-12-08
95	BTECH	CSE	PES UNIVERSITY	8.8	Rubble	Bangalore	Co-Founder	Founded a gar...	2016-12-08
95	BTECH	CSE	PES UNIVERSITY	8.8	Center for Clou...	Bangalore	Research Intern	Decreasing Tail...	2020-05-20
95	BTECH	CSE	PES UNIVERSITY	8.8	Center for Clou...	Bangalore	Research Intern	Decreasing Tail...	2020-05-20
95	BTECH	CSE	PES UNIVERSITY	8.8	Center for Clou...	Bangalore	Research Intern	Decreasing Tail...	2020-05-20

TITLE	DESCRIPTION	START_DATE	END_DATE	PROJECT_NAME	DESCRIPTION	SKILL_NAME	EXPERIENCE_LE...	ACCOMPLISHM...	DESCRIPTION
Co-Founder	Founded a gar...	2016-12-08	*NULL*	AutoCorrect	Automated Gra...	C	8	Survived COVI...	Survival 100
Co-Founder	Founded a gar...	2016-12-08	*NULL*	AutoCorrect	Automated Gra...	DBMS	10	Survived COVI...	Survival 100
Co-Founder	Founded a gar...	2016-12-08	*NULL*	AutoCorrect	Automated Gra...	HTML/CSS	8	Survived COVI...	Survival 100
Co-Founder	Founded a gar...	2016-12-08	*NULL*	AutoCorrect	Automated Gra...	Python	10	Survived COVI...	Survival 100
Research Intern	Decreasing Tail...	2020-05-20	*NULL*	AutoCorrect	Automated Gra...	C	8	Survived COVI...	Survival 100
Research Intern	Decreasing Tail...	2020-05-20	*NULL*	AutoCorrect	Automated Gra...	DBMS	10	Survived COVI...	Survival 100
Research Intern	Decreasing Tail...	2020-05-20	*NULL*	AutoCorrect	Automated Gra...	HTML/CSS	8	Survived COVI...	Survival 100
Research Intern	Decreasing Tail...	2020-05-20	*NULL*	AutoCorrect	Automated Gra...	Python	10	Survived COVI...	Survival 100
Co-Founder	Founded a gar...	2016-12-08	*NULL*	Tail Bench	Decreasing tail ...	C	8	Survived COVI...	Survival 100
Co-Founder	Founded a gar...	2016-12-08	*NULL*	Tail Bench	Decreasing tail ...	DBMS	10	Survived COVI...	Survival 100
Co-Founder	Founded a gar...	2016-12-08	*NULL*	Tail Bench	Decreasing tail ...	HTML/CSS	8	Survived COVI...	Survival 100
Co-Founder	Founded a gar...	2016-12-08	*NULL*	Tail Bench	Decreasing tail ...	Python	10	Survived COVI...	Survival 100
Research Intern	Decreasing Tail...	2020-05-20	*NULL*	Tail Bench	Decreasing tail ...	C	8	Survived COVI...	Survival 100
Research Intern	Decreasing Tail...	2020-05-20	*NULL*	Tail Bench	Decreasing tail ...	DBMS	10	Survived COVI...	Survival 100
Research Intern	Decreasing Tail...	2020-05-20	*NULL*	Tail Bench	Decreasing tail ...	HTML/CSS	8	Survived COVI...	Survival 100

Decomposition of SCHOOL and SCHOOLING Information

We can decompose all the schooling information associated with a user into two relations – SCHOOL and SCHOOLING and test for a lossless join. The set of attributes in each relation consists of {SCHOOL ID, LOCATION, NAME} and {MEMBER ID, SCHOOL ID, PASSING LEVEL, STREAM, PASSING YEAR, SCORE} respectively. The union of these attribute sets gives us all the schooling information about a user. The intersection of these attributes is the SCHOOL ID, and hence as shown the decomposition is lossless. The FD sets are {SCHOOL ID -> LOCATION, NAME} and {MEMBER ID, PASSING LEVEL -> SCHOOL ID, STREAM, PASSING YEAR, SCORE}.

RELATION	SCHOOL ID	LOCATION	NAME	MEMBER ID	PASSING LEVEL	STREAM	PASSING YEAR	SCORE
SCHOOL	a1	a2	a3	b14	b15	b16	b17	b18
SCHOOLING	a1	a2	a3	a4	a5	a6	a7	a8

Decomposition of DEGREE and HIGHER EDUCATION Information

We can similarly decompose all the higher education information associated with a user into two relations – DEGREE INFO and HIGHER EDUCATION and test for a lossless join. The set of attributes in each relation consists of {DEGREE ID, GRAD TYPE, DEGREE, MAJOR} and {MEMBER ID, DEGREE ID, GPA, UNIVERSITY} respectively. The union of these attribute sets gives us all the higher education pursued about a user. The intersection of these attributes is the DEGREE ID, and hence as shown the decomposition is lossless. The FD sets are {DEGREE ID -> GRAD TYPE, DEGREE, MAJOR} and {MEMBER ID, DEGREE ID -> GPA, UNIVERSITY}.

RELATION	DEGREE ID	GRAD TYPE	DEGREE	MAJOR	MEMBER ID	GPA	UNIVERSITY
DEGREE INFO	a1	a2	a3	a4	b15	b16	b17
HIGHER EDUCATION	a1	a2	a3	a4	a5	a6	a7

Decomposition of JOB and EXPERIENCE Information

We can finally decompose all the job ana experience information associated with a user into two relations – JOB and EXPERIENCE and test for a lossless join. The set of attributes in each relation consists of {JOB ID, TITLE, LOCATION, DESCRIPTION, EMPLOYER} and {MEMBER ID, JOB ID, START DATE, END DATE} respectively. The union of these attribute sets gives us all the higher education pursued about a user. The intersection of these attributes is the JOB ID, and hence as shown the decomposition is lossless. The FD sets are {JOB ID -> TITLE, LOCATION, DESCRIPTION, EMPLOYER } and {MEMBER ID, JOB ID -> START DATE, END DATE}.

RELATION	JOB ID	TITLE	LOCATION	DESCRIPTION	EMPLOYER	MEMBER ID	START DATE	END DATE
JOB	a1	a2	a3	a4	a5	b16	b17	b18
EXPERIENCE	a1	a2	a3	a4	a5	a6	a7	a8

DDL

Attached below are the DDL commands to create all the relations in the database along with check and referential integrity constraints.

```
CREATE TABLE MEMBER
(
    MEMBER_ID VARCHAR(10) NOT NULL PRIMARY KEY,
    PASSWORD VARCHAR(40) NOT NULL,
    NAME VARCHAR(30) NOT NULL,
    EMAIL VARCHAR(40) NOT NULL UNIQUE,
    PHONE VARCHAR(10) NOT NULL UNIQUE,
    ADDRESS VARCHAR (200),
    BIO VARCHAR(100),
    GITHUB VARCHAR(50),
```

```
    LINKEDIN VARCHAR(50)
);



CREATE TABLE SCHOOL
(
    SCHOOL_ID VARCHAR(10) NOT NULL PRIMARY KEY,
    LOCATION VARCHAR(100),
    NAME VARCHAR(80) NOT NULL
);



CREATE TABLE SCHOOLING
(
    MEMBER_ID VARCHAR(10) NOT NULL FOREIGN KEY REFERENCES MEMBER(MEMBER_ID) ON
    DELETE CASCADE ON UPDATE CASCADE,
    SCHOOL_ID VARCHAR(10) NOT NULL FOREIGN KEY REFERENCES SCHOOL(SCHOOL_ID) ON
    UPDATE CASCADE,
    PASSING_LEVEL INT NOT NULL CHECK (PASSING_LEVEL=10 OR PASSING_LEVEL=12),
    STREAM VARCHAR(10) NOT NULL CHECK (STREAM IN
    ('SCIENCE','COMMERCE','ARTS','HUMANITIES')),
    PASSING_YEAR INT NOT NULL CHECK (LEN(CONVERT(VARCHAR(10),PASSING_YEAR))=4),
    SCORE FLOAT NOT NULL CHECK(SCORE>=0 AND SCORE<=100),
    PRIMARY KEY(MEMBER_ID,PASSING_LEVEL)
);



CREATE TABLE DEGREE_INFO
(
    DEGREE_ID VARCHAR(10) NOT NULL PRIMARY KEY,
    TYPE_OF_GRADUATION VARCHAR(30) NOT NULL,
    DEGREE VARCHAR(30) NOT NULL,
    MAJOR VARCHAR(30) NOT NULL
);



CREATE TABLE HIGHER_EDUCATION
(
    MEMBER_ID VARCHAR(10) NOT NULL FOREIGN KEY REFERENCES MEMBER(MEMBER_ID) ON
    DELETE CASCADE ON UPDATE CASCADE,
    DEGREE_ID VARCHAR(10) NOT NULL FOREIGN KEY REFERENCES DEGREE_INFO(DEGREE_ID)
    ON UPDATE CASCADE,
    GPA FLOAT NOT NULL CHECK (GPA>=0 AND GPA<=10),
    UNIVERSITY VARCHAR(40) NOT NULL,
    PRIMARY KEY(MEMBER_ID, DEGREE_ID)
);
```

```
CREATE TABLE SKILLS
(
    MEMBER_ID VARCHAR(10) NOT NULL FOREIGN KEY REFERENCES MEMBER(MEMBER_ID) ON
    DELETE CASCADE ON UPDATE CASCADE,
    SKILL_NAME VARCHAR(30) NOT NULL,
    EXPERIENCE_LEVEL INT CHECK (EXPERIENCE_LEVEL>0 AND EXPERIENCE_LEVEL<=10),
    PRIMARY KEY(MEMBER_ID, SKILL_NAME)
);


CREATE TABLE PROJECTS
(
    MEMBER_ID VARCHAR(10) NOT NULL FOREIGN KEY REFERENCES MEMBER(MEMBER_ID) ON
    DELETE CASCADE ON UPDATE CASCADE,
    PROJECT_NAME VARCHAR(30) NOT NULL,
    DESCRIPTION VARCHAR(100),
    PRIMARY KEY(MEMBER_ID, PROJECT_NAME)
);


CREATE TABLE JOB
(
    JOB_ID VARCHAR(10) NOT NULL PRIMARY KEY,
    TITLE VARCHAR(50) NOT NULL,
    LOCATION VARCHAR(50) NOT NULL,
    DESCRIPTION VARCHAR(100),
    EMPLOYER VARCHAR(80) NOT NULL
);


CREATE TABLE EXPERIENCE
(
    MEMBER_ID VARCHAR(10) NOT NULL FOREIGN KEY REFERENCES MEMBER(MEMBER_ID) ON
    DELETE CASCADE ON UPDATE CASCADE,
    JOB_ID VARCHAR(10) NOT NULL FOREIGN KEY REFERENCES JOB(JOB_ID) ON UPDATE
    CASCADE,
    START_DATE DATE NOT NULL,
    END_DATE DATE,
    PRIMARY KEY(MEMBER_ID, JOB_ID)
);


CREATE TABLE ACCOMPLISHMENTS
(
    MEMBER_ID VARCHAR(10) NOT NULL FOREIGN KEY REFERENCES MEMBER(MEMBER_ID) ON
    DELETE CASCADE ON UPDATE CASCADE,
    ACCOMPLISHMENT_NAME VARCHAR(30) NOT NULL,
    DESCRIPTION VARCHAR(100),
    PRIMARY KEY(MEMBER_ID, ACCOMPLISHMENT_NAME)
);
```

```
CREATE TABLE CERTIFICATIONS
(
    MEMBER_ID VARCHAR(10) NOT NULL FOREIGN KEY REFERENCES MEMBER(MEMBER_ID) ON
    DELETE CASCADE ON UPDATE CASCADE,
    CERTIFICATION_URL VARCHAR(30) NOT NULL,
    CERTIFICATE_NAME VARCHAR(100) NOT NULL,
    PRIMARY KEY(MEMBER_ID, CERTIFICATION_URL)
);
```

Triggers

A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

There are two triggers in the database model, one on **EXPERIENCE** and the other on **SCHOOLING**. The former trigger is used to check and set a default value to a field and the latter is used to perform an arithmetic operation on an attribute's value.

EXPERIENCE
CHECK_END_AND_START_DATES

In the given database, we have a table called EXPERIENCE in which the experience details of all users are stored. Whenever a record is inserted into the table, we need to verify whether the end date of each job is after the start date automatically, and if not, we need to assign it to the default value of NULL. The reason we assign it to NULL and not the present date is because if the present date is recorded then we will have to update the database daily to update the end date since the user is still working in that role. It would also lead to a false notion while retrieval of data that a user will leave his role at the end of the present working day. On the other hand, a NULL value would not only signify that the user is still working in his present role, but the duration of work can also be easily calculated by swapping the NULL value with the current date, and hence the value needs to be updated only when the user quits his current job.

To do so, we created a trigger on the relation EXPERIENCE, that checks whether the end date of each job falls after the start date and updates it as required in the respective column after a record consisting of job title, location, description, employer, start date and end date is inserted in the table.

```
GO
CREATE TRIGGER CHECK_END_AND_START_DATES
ON EXPERIENCE
AFTER INSERT
AS SET NOCOUNT ON
BEGIN
    UPDATE EXPERIENCE SET END_DATE = NULL WHERE (END_DATE <= START_DATE)
END;
```

SCHOOLING
CGPA_TO_PERCENTAGE

In the given database, we have a table called SCHOOLING in which the schooling details of all users are stored. Whenever a record is added to this relation, we need to ensure that all the scores entered by users for their 10^{th} and 12^{th} grade examinations are on the same scale. Since the widely used metric is percentage, the database model uses this scale to store a user's score in the examinations. This leads to uniformity in data collected and helps in analysis of stored information. However, it might be possible that a user has not converted their score to percentage while registering themselves, and hence the score reflected in the database is their CGPA which is out of 10. To prevent this mishap, we need a system that automatically converts scores in CGPA to percentage whenever a user registers or a new entry is made in the relation. We make an assumption here that the scores from 0 to 10 are scores in terms of CGPA and not percentage since attaining a score so low (less than 10%) would imply that a user has not passed his 10^{th} or 12^{th} examinations.

To do this, we create a trigger on the relation SCHOOLING, that checks whether the score entered for the qualifying examination is above 10%, if not, it transforms that to out of 100 by multiplying it by 9.5. It performs this update in the required column after a record consisting of stream, passing year and score is inserted into the relation.

```
GO
CREATE TRIGGER CGPA_TO_PERCENTAGE
ON SCHOOLING
AFTER INSERT
AS SET NOCOUNT ON
BEGIN
    UPDATE SCHOOLING SET SCORE = SCORE*9.5 WHERE (SCORE<=10)
END;
```

SQL Queries

Query 1

An employer wants to recruit for an internship. They ask for project details of all clients who satisfy the following criteria:
1) GPA > 8.5
2) Year of passing 12^{th} grade is 2018
We can help the employer by giving him the necessary details.

```
SELECT
M.MEMBER_ID, P.PROJECT_NAME, P.DESCRIPTION FROM MEMBER M
INNER JOIN PROJECTS P ON M.MEMBER_ID = P.MEMBER_ID
INNER JOIN HIGHER_EDUCATION H ON H.MEMBER_ID = M.MEMBER_ID
WHERE H.GPA > 8.5 AND M.MEMBER_ID IN
(SELECT MEMBER_ID FROM SCHOOLING WHERE PASSING_LEVEL = 12 AND PASSING_YEAR = 2018)
ORDER BY H.GPA DESC;
```

	MEMBER_ID	PROJECT_NAME	DESCRIPTION
1	MEM#000026	AutoGrader	Automatic grading of C programs.
2	MEM#000001	AutoCorrect	Automated Grading of Papers.
3	MEM#000001	Tail Bench	Decreasing tail latency.

Query 2

My neighbours are looking for schools to send their son as he turns 3 next year. To plan out their visits, they need a distribution of the schools across the city. We need to provide them with a count of the number of schools in each area so they can decide how to plan their visits.

```
SELECT
S.LOCATION, COUNT(S.LOCATION) AS TOTAL_SCHOOLS
FROM
SCHOOL S
GROUP BY
S.LOCATION
ORDER BY
COUNT(S.LOCATION) DESC;
```

	LOCATION	TOTAL_SCHOOLS
1	RESIDENCY ROAD	5
2	RICHMOND ROAD	3
3	KALYAN NAGAR	3
4	INDIRANAGAR	2
5	OLD AIRPORT ROAD	2
6	RAJAJINAGAR	1

Query 3

It is internship season and students are busy applying for various roles at different companies. PES University wants to email their students no objection certificates (NOC) so they can have a smooth hiring process. We can provide them with the necessary details.

```
SELECT M.EMAIL FROM MEMBER M WHERE M.MEMBER_ID IN
(SELECT MEMBER_ID FROM HIGHER_EDUCATION WHERE UNIVERSITY = 'PES UNIVERSITY');
```

	EMAIL
1	aditeya.baral@gmail.com
2	kanikashukla55@gmail.com
3	shanaya66@gmail.com
4	bikram37@gmail.com
5	akshatparekh37@gmail.com
6	abaksy@gmail.com

Query 4

A user seems to have forgotten his unique ID. However, he needs access to his job details since today is the last day to apply for an internship. He does not have access to his computer. Using his email address and password, we can retrieve his details for him.

```
SELECT
J.EMPLOYER, J.LOCATION, J.TITLE, J.DESCRIPTION,
JOB_YEARS = (CASE
WHEN END_DATE IS NULL THEN DATEDIFF(YEAR, START_DATE, GETDATE())
WHEN END_DATE IS NOT NULL THEN DATEDIFF(YEAR, START_DATE, END_DATE)END)
FROM
MEMBER M
INNER JOIN EXPERIENCE E ON M.MEMBER_ID = E.MEMBER_ID
INNER JOIN JOB J ON E.JOB_ID = J.JOB_ID
WHERE M.EMAIL = 'aditeya.baral@gmail.com' AND M.PASSWORD = 'hello123';
```

	EMPLOYER	LOCATION	TITLE	DESCRIPTION	JOB_YEARS
1	Rubble	Bangalore	Co-Founder	Founded a garbage schduling company	4
2	Center for Cloud Computing and Big Data	Bangalore	Research Intern	Decreasing Tail Latency	0

Query 5

A recruiter from a well-known company is looking to interview top clients for a high paying role. However, he has been instructed to only interview those clients that possess at least 10 years of job experience. We can retrieve such potential employees for them.

```
DECLARE @TEMP_TABLE TABLE(
    ID VARCHAR(10),
    TOTAL_EXP INT
);
```

```
WITH MEMBER_EXPERIENCE(MEMBER_ID, NEW_DUR) AS
(SELECT MEMBER.MEMBER_ID, NEW_DUR = (CASE
WHEN END_DATE IS NULL THEN DATEDIFF(YEAR, START_DATE, GETDATE())
WHEN END_DATE IS NOT NULL THEN DATEDIFF(YEAR, START_DATE, END_DATE)END)
FROM EXPERIENCE E JOIN MEMBER ON E.MEMBER_ID = MEMBER.MEMBER_ID)
INSERT INTO @TEMP_TABLE select MEMBER_ID, SUM(NEW_DUR) AS TOTAL_EXPERIENCE
FROM MEMBER_EXPERIENCE GROUP BY MEMBER_ID HAVING SUM(NEW_DUR) > 10
SELECT  M.NAME, M.EMAIL, M.PHONE, M.GITHUB, M.LINKEDIN, TOTAL_EXP
FROM @TEMP_TABLE INNER JOIN MEMBER M ON M.MEMBER_ID = ID;
```

	NAME	EMAIL	PHONE	GITHUB	LINKEDIN	TOTAL_EXP
1	Jeet Anne	anne42@gmail....	998419646	jeetanne	jeetanne178	28
2	Anushka Date	date27@gmail....	977472283	*NULL*	dateee	23
3	Kanika Shukla	kanikashukla55...	798618536	kanikaaaa	*NULL*	37
4	Jayanti Sidhu	jayantisidhu71...	789786673	*NULL*	*NULL*	25
5	Bikram Sami	bikram37@gm...	998345797	bikramsami258	*NULL*	63
6	Karun Mangal	karunmangal97...	787179486	karunnn	*NULL*	46
7	Indrani Khare	indranikhare51...	879264244	*NULL*	indranikhare	17
8	Shridevi Menon	sm47@gmail.c...	899643446	shridevii	shrideviiiii	21
9	Akshat Parekh	akshatparekh3...	878611581	*NULL*	akshatparekh472	12
10	Daljeet Balakris...	daljeetbalakris...	997869268	daljeetbalakris...	*NULL*	46
11	Geetha Kapadia	gk71@gmail.co...	777677186	geethaaaaa	*NULL*	19
12	Uma Saran	saran99@gmail...	979915452	*NULL*	*NULL*	13
13	Anaisha Samra	as05@gmail.com	978255235	*NULL*	samraaaa	33

Query 6

A recruiter wants to find all the Bachelor programmes being offered and the academic profiles of the students pursuing them. We can retrieve these details.

```
SELECT D.DEGREE, D.MAJOR, H.MEMBER_ID, H.GPA, H.UNIVERSITY FROM DEGREE_INFO D LEFT
OUTER JOIN HIGHER_EDUCATION H ON H.DEGREE_ID = D.DEGREE_ID
WHERE D.TYPE_OF_GRADUATION = 'BACHELOR''S';
```

DEGREE	MAJOR	MEMBER_ID	GPA	UNIVERSITY
BSC	PHYSICS	MEM#000025	5	IIT Bombay
BSC	CHEMISTRY	*NULL*	*NULL*	*NULL*
BSC	STATISTICS	MEM#000003	7	NITK
BSC	STATISTICS	MEM#000004	7	Amrita College ...
BSC	POLITICAL SCIENCE	MEM#000015	5	Amrita College ...
BSC	POLITICAL SCIENCE	MEM#000025	7	BITS Pilani
BSC	COMPUTER SCIEN...	*NULL*	*NULL*	*NULL*

Query 7

A final year student would like to know about all the SWE job roles Google and Microsoft are offering and about their current employees in that role. We can retrieve these details from our database.

```
SELECT J.JOB_ID, J.LOCATION, J.EMPLOYER, E.MEMBER_ID
FROM JOB J LEFT OUTER JOIN EXPERIENCE E ON J.JOB_ID = E.JOB_ID
WHERE J.EMPLOYER IN ('Google','Microsoft') AND J.TITLE = 'SWE';
```

	JOB_ID	LOCATION	EMPLOYER	MEMBER_ID
1	JOB#000005	Bangalore	Google	*NULL*
2	JOB#000014	Mumbai	Google	MEM#000023
3	JOB#000023	New Delhi	Google	*NULL*
4	JOB#000032	Kolkata	Google	*NULL*
5	JOB#000041	Lucknow	Google	MEM#000014
6	JOB#000050	Chennai	Google	*NULL*

Conclusion

The aim of the project was to develop an application that can store a user's professional details such that the details can be aggregated to produce a well formatted profile for any candidate. Automated Resume Generator can use these details to build top notch resumes for any candidate that registers themself using the application.

It thus decreases the effort needed to apply for internships and jobs and helps students by presenting their skills and achievements in an attractive manner to match with lucrative offers on the job market. It reduces the time and effort put in to build a resume that stands out and can build polished resumes in a matter of mere minutes.

The application can also be used by recruiters and other organizations to filter their searches and look for candidates and other details that match their need based on their requirements and matching criteria. It can be used to retrieve potential employees as well as other general information about certifications, accomplishments such as published papers, skillsets and even information about schools and universities.

Hence Automated Resume Generator is a very practical utility that finds large scale application as both an information aggregator and resume builder across various fields of interest. It finds consumers in both university and graduate level students looking for job offers as well as employers and recruiters from large companies looking to expand and enhance their work force.

Automated Resume Generator does have a few limitations, and these can be removed or worked upon in future updates.

1. **Lack of Resume Templates**
   There is only one supported template for a resume. The application can be further customized if users had the option to pick a template for their resume. This idea can be further improved upon by allowing them to customize the template itself then preparing the resume for a user. The generated resume would then be just how the user would want it and would also add a personal touch to the generated resumes.

2. **Lack of Verification**
   Some of the details entered by the user must be verified before they can be used in a resume. These include work links such as GitHub and LinkedIn, certifications URL's, phone numbers, email addresses and experience. This would make the application more robust for large scale use by recruiters and organizations.

3. **Outdated Information**
   The two universal relations, SCHOOL and DEGREE_INFO store details that are too shallow to provide any valuable information apart from the location and can easily become outdated. This database can be connected in real time to an institution database and its records can be updated periodically to store more information about an institution such as the exact location, reviews, popularity, programmes offered and so on. This can similarly be done for the JOB table to store updated job postings to make the application dynamic.

4. **Additional Resume Sections**
   More sections containing information can be added, such as activities taken part in during their schooling or higher education days or test scores and known languages.