Database Techniques
Project Report

Ahmed Mohamud
Computer Science BEng.
TBIT2
2018

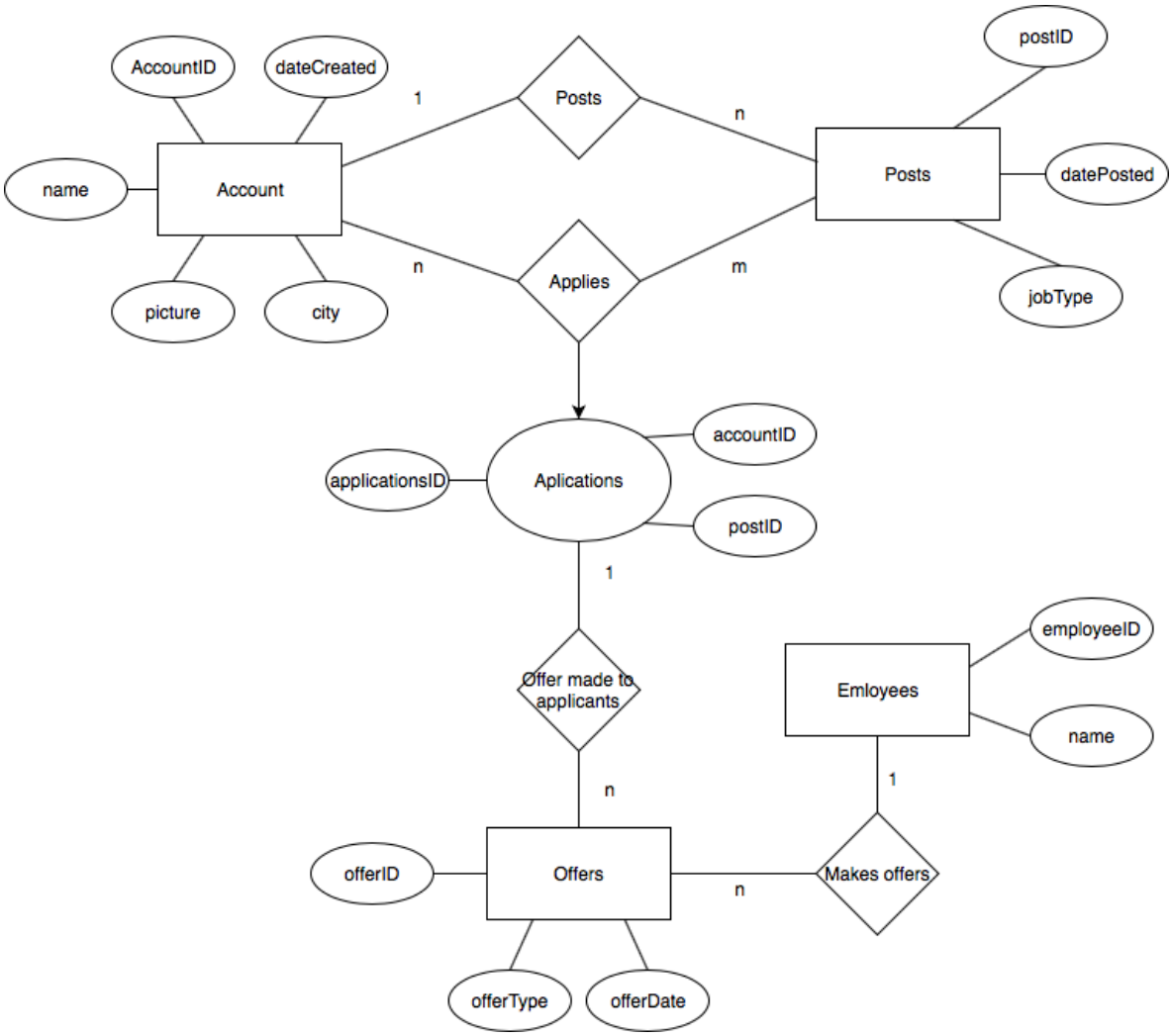
Table of Contents

1.DESCRIPTION OF DATABASE

the database I created is called the LOCAL JOB APP. It is part of a database for a app that is a platform for job postings and applications. It is essentially like LinkedIn the sole difference being the platform is moderated by an agency which links the best applicants to the jobs. It also facilitates payment through the platform and the agency's cash flow is mainly through the commissions gained from those transactions. The database is therefore, designed to help manage this through a correct data-strategy.

The core of the database which is what I created is simply composed of 4 tables and one derivative table. These tables are the Accounts , Posts, Offers and Employees. The fifth table present is the result of the many to many connection between the Account and Posts tables. The attributes and the cardinality of the tables will be described further down in the report.

2. ER MODEL



This ER diagram describes the job-application platform's database. It is composed of 4 tables/entities **Account, Posts, Offers and Employees**. Applications is table that would be generated from the many to many relationship between Account and Posts.
Each entity has at least 3 attributes except Employees and there are no total participations (all entities are involved in a partial participation). The most common type of relationship is the one to many cardinality. Below is described the rationality behind each cardinality.

One to many:
Each account can make many posts but each post is made by one account
Each application can receive many offers but each offer is directed at one application.
Each employee can make many offers but each offer is made by one employee

Many to many:
Each account can apply to many job posts and each job post can be applied to by many accounts

3. IMPLEMENTATION DIAGRAM

*I have purposefully left in the 'applicationID' primary key in the 'Applications' table so that can demonstrate how I am fixing my mistake (redundancy and not 3NF) through SQL commands.

4.SQL STATEMENTS

In this segment I will show my proficiency in the SQL language by demonstrating how I use the commands I learned. It will be demonstrated through applying DDL and DML commands on a self-designed database. This is the same database that has been introduced prior in the report.

Note:
I have been filling and using the database before creating the report and therefore actions aren't necessarily in the order they should be!

SELECT

The first statement to demonstrate is the SELECT statement. This is usually the first statement in command and therefore appropriate to show first.

```
select * from offers;
```

100% 1:168

Result Grid | Filter Rows: Search | Edit: | Export/Import:

idOffers	offerDate	offerType	Employee_idEmployee	Applications_applicatio...
NULL	NULL	NULL	NULL	NULL

I used select to show me the content of an unfilled table to see what is inside this table. As expected it returned nothing but null values for all the column of that table. We will later see how this table will change.

Another demonstration of SELECT is demonstrated below to show a filled table. The whole 'Account' table.

```
select * from Account ;
```

100% 24:29

Result Grid | Filter Rows: Search | Edit: | Export/Import:

idAccount	dateCreated	name	city	picture
2	01/01/18	Alex	kristianstad	NULL
3	02/01/18	Aries	Bankok	NULL
4	02/01/18	Muhis	Huagzhou	NULL
5	02/01/18	Majid	Dubai	NULL
6	03/01/18	Major	Nairobi	NULL
7	04/01/18	Hermione	Addis Ababa	NULL
8	04/01/18	Kristian	Dear es Salaam	NULL
9	04/01/18	Kolaris	Lagos	NULL
10	04/01/18	ahmed	Djibouti	NULL
NULL	NULL	NULL	NULL	NULL

INSERT

This command is used to put data into the columns of the tables. As demonstrated in the inserted screenshot prior, the 'Account' table only has 10 rows. U used a combination of SELECT and INSERT to show how I insert data into the table.

```
INSERT INTO Account (dateCreated, name,  city)
VALUES ('05/01/18' , 'Jojo' ,  'Tehran');

select * from Account ;
```

100% 42:30

Result Grid | Filter Rows: Search | Edit: | Export/Import:

idAccount	dateCreated	name	city	picture
3	02/01/18	Aries	Bankok	NULL
4	02/01/18	Muhis	Huagzhou	NULL
5	02/01/18	Majid	Dubai	NULL
6	03/01/18	Major	Nairobi	NULL
7	04/01/18	Hermione	Addis Ababa	NULL
8	04/01/18	Kristian	Dear es Salaam	NULL
9	04/01/18	Kolaris	Lagos	NULL
10	04/01/18	ahmed	Djibouti	NULL
11	05/01/18	Jojo	Tehran	NULL
NULL	NULL	NULL	NULL	NULL

The commands shown insert the date, name and city into the columns 'dateCreated', 'name' and 'city'.

In fact, the INSERT INTO is how I added all the data into the table as demonstrated below.

```
INSERT INTO Account (dateCreated, name,  city)
VALUES ('02/01/18' , 'Majid' ,  'Dubai');

INSERT INTO Account (dateCreated, name,  city)
VALUES ('03/01/18' , 'Major' ,  'Nairobi');

INSERT INTO Account (dateCreated, name,  city)
VALUES ('04/01/18' , 'Hermione' ,  'Addis Ababa');

INSERT INTO Account (dateCreated, name,  city)
VALUES ('04/01/18' , 'Kristian' ,  'Dear es Salaam');

INSERT INTO Account (dateCreated, name,  city)
VALUES ('04/01/18' , 'Kolaris' ,  'Lagos');

INSERT INTO Account (dateCreated, name,  city)
VALUES ('04/01/18' , 'ahmed' ,  'Djibouti');
```

100% 47:10

Result Grid | Filter Rows: Search | Edit: | Export/Import:

idAccount	dateCreated	name	city	picture
5	02/01/18	Majid	Dubai	NULL
6	03/01/18	Major	Nairobi	NULL
7	04/01/18	Hermione	Addis Ababa	NULL
8	04/01/18	Kristian	Dear es Salaam	NULL
9	04/01/18	Kolaris	Lagos	NULL
10	04/01/18	ahmed	Djibouti	NULL

DELETE

However, we don't necessarily need 11 rows of data. Therefore, I will just get rid of the newly created row in the 'Account' table through the DELETE command.

```
34  DELETE FROM Account WHERE idAccount = 11;
35
36  select * from Account ;
37
38
39
40
41
42
```

100% 42:34

Result Grid | Filter Rows: Search | Edit: | Export/Import:

idAccount	dateCreated	name	city	picture
1	01/01/18	ahmed	Kristianstad	NULL
2	01/01/18	Alex	kristianstad	NULL
3	02/01/18	Aries	Bankok	NULL
4	02/01/18	Muhis	Huagzhou	NULL
5	02/01/18	Majid	Dubai	NULL
6	03/01/18	Major	Nairobi	NULL
7	04/01/18	Hermione	Addis Ababa	NULL
8	04/01/18	Kristian	Dear es Salaam	NULL
9	04/01/18	Kolaris	Lagos	NULL
10	04/01/18	ahmed	Djibouti	NULL
NULL	NULL	NULL	NULL	NULL

This statement eradicates the row where 'idAccount' = 11. That was the column most recently created (as demonstrated in the INSERT section).

UPDATE

I had also noticed a spelling mistake in one of the names. I will therefore use UPDATE to correct that mistake.

I will use update to change an account owners name from Kristian (as shown in all previous examples) to Christian.

```
40  UPDATE Account
41  SET name = 'Christian'
42  WHERE idAccount = 8;
43
44
45  select * from Account ;
46
47
```

100% 24:45

Result Grid | Filter Rows: Search | Edit: | Export/Import:

idAccount	dateCreated	name	city	picture
2	01/01/18	Alex	kristianstad	NULL
3	02/01/18	Aries	Bankok	NULL
4	02/01/18	Muhis	Huagzhou	NULL
5	02/01/18	Majid	Dubai	NULL
6	03/01/18	Major	Nairobi	NULL
7	04/01/18	Hermione	Addis Ababa	NULL
8	04/01/18	Christian	Dear es Salaam	NULL
9	04/01/18	Kolaris	Lagos	NULL
10	04/01/18	ahmed	Djibouti	NULL
NULL	NULL	NULL	NULL	NULL

ALTER

Before using the Join commands, I have to fix something on a table. I have a table that does not meet the 3NF design requirements and I will use alter to change that.

This table is the resultant table from the many-to-many connection of ‘Accounts’ and ‘Posts’. I added an extra column named ‘ApplicationID’.

I will get rid of this column because,

1. Its redundant (the two foreign keys make up a composite key)
2. Doesn’t conform to 3NF

```
150  select * from Applications;
151
152
153
154
155
```

100% 28:150

Result Grid | Filter Rows: Search | Edit: | Export/Import:

Account_idAccou...	Posts_idPosts	applicationID
2	2	1
8	14	10
8	6	11
2	5	2
2	6	3

*here you can see the 3 columns in the ‘Applications’ table.

```
151  ALTER TABLE Applications DROP COLUMN ApplicationID;
152
153  select * from Applications;
154
155
156
```

100% 28:153

Result Grid | Filter Rows: Search | Export:

Account_idAccou...	Posts_idPosts
2	2
8	14
8	6
2	5
2	6
2	8

Now the Applications table only has two columns and they are the default composite key that distinguish each row.

DROP

MySQL wasn't allowing me to 'drop' a the applicationID Column since it was beign used as a foreign key in the Offers table.

The only option for me was to completely DROP the whole table if I was to succeed in removing the column like I wanted to.

```
148  DROP table Offers;
149  select * from Offers;
150
```

100% 22:149

Action Output

	Time	Action	Response
85	11:43:15	select sum(Account.idAccount) from Account LIMIT 0, 50000	1 row(s) re
86	19:28:19	select * from offers LIMIT 0, 50000	0 row(s) re

*output nothing since the table no longer exists

105	21:12:29	DROP table Offers	0 row(s) affected

*this shows the DROP command succeeded

108	21:16:58	select * from Offers LIMIT 0, 50000	Error Code: 1146. Table 'local_job_app.offers' doesn't exist

*this shows that it no longer exists

CREATE

Since I deleted my Offers table through the use of DROP I had to recreate it using the CREATE command.

```
148  DROP table Offers;
149
150
151
152  CREATE TABLE Offers (
153      idOffers INT,
154      offerDate VARCHAR(45),
155      OfferType ENUM ("Trial ", "Temporary " ,"Permanent"),
156      idEmployee INT,
157      Account_idAccount INT,
158      idPosts INT,
159      PRIMARY KEY (idOffers),
160      FOREIGN KEY (idEmployee) REFERENCES Employee(idEmployee),
161      FOREIGN KEY (Account_idAccount , idPosts) REFERENCES Applications(Account_idAccount , Posts_idPosts)
162  );
163
164  CREATE INDEX indexApplications
165  ON Applications (Account_idAccount, Posts_idPosts);
166
167  select * from Offers;
168
```

100% 22:167

Result Grid Filter Rows: Search Edit: Export/Import:

idOffers	offerDate	OfferType	idEmployee	Account_idAccou...	idPosts
NULL	NULL	NULL	NULL	NULL	NULL

JOIN

1. In the **first join command** I would like to join two table. These are employee and the Offers table.

I am doing this to know which employee gave offers and which type of offers they gave.

To do this I would use the **inner join** with the syntax

$$select * from\ T1\ , T2\ \ where\ T1.C1\ \ = T2.C2;$$

Results:

```
select name , OfferType from Employee, Offers where Offers.idEmployee = Employee.idEmployee;
```

100% 1:268

Result Grid | Filter Rows: Search | Export:

name	OfferType
Kyyro	Permanent
Kyyro	Trial
Abdi	Trial
Abdi	Temporary
Abdi	Trial
Abdi	Trial
Abdi	Temporary
Abdi	Permanent
Colonel	Temporary
Colonel	Temporary

2. In the **second Join command** I am joining another 2 tables and these are the Account and Posts tables.

I am doing this to find who posted jobs, in which city and what type of jobs did they post.

```
select name, city, jobType from Account , Posts where Account.idAccount = Posts.Account_idAccount;
```

100% 99:271

Result Grid | Filter Rows: Search | Export:

name	city	jobType
Muhis	Huagzhou	Construction
Muhis	Huagzhou	Construction
Muhis	Huagzhou	Construction
Muhis	Huagzhou	Construction
ahmed	kristianstad	Marketing
Muhis	Huagzhou	Construction
Majid	Dubai	Programming
Major	Nairobi	database Adminstration
Muhis	Huagzhou	Construction
Alex	kristianstad	Child Minding
Majid	Dubai	Programming
Herm…	Addis Aba…	Consultancy
Herm…	Addis Aba…	Consultancy
Alex	kristianstad	Cleaner

3. In the **third join command** I will be **joining three tables**.

In order to discover if there is any “favouritism” of the employees towards certain accounts or applicants we would like to create a table showing the applicant that received the offer, the offer type and who gave the offer.

```
select Account.name as applicant , offerType, Employee.name as employee
from Account, Offers , Employee
where Offers.Account_idAccount = Account.idAccount and Offers.idEmployee = Employee.idEmployee;
```

100% 96:277

Result Grid Filter Rows: Search Export:

applicant	offerType	employee
Aries	Permanent	Kyyro
Alex	Trial	Kyyro
Alex	Trial	Abdi
Alex	Temporary	Abdi
Alex	Trial	Abdi
Christian	Trial	Abdi
Aries	Temporary	Abdi
Aries	Permanent	Abdi
Christian	Temporary	Colonel
Christian	Temporary	Colonel

*this showed us that Abdi made half of his offers to the same person Alex and this warrants us to do a background check if they have any personal links as such favouritism would be against company policy.

5.CONCLUSION

I would like to state that I thoroughly enjoyed working on this project and that it was a good learning experience. I was great that we had the option of choosing out own database to design and that was what made it exciting for me. Perhaps the timing of the project could’ve been better although I also understand the constraints of the university. So, in conclusion this type of project is great and I would like to see more of it.