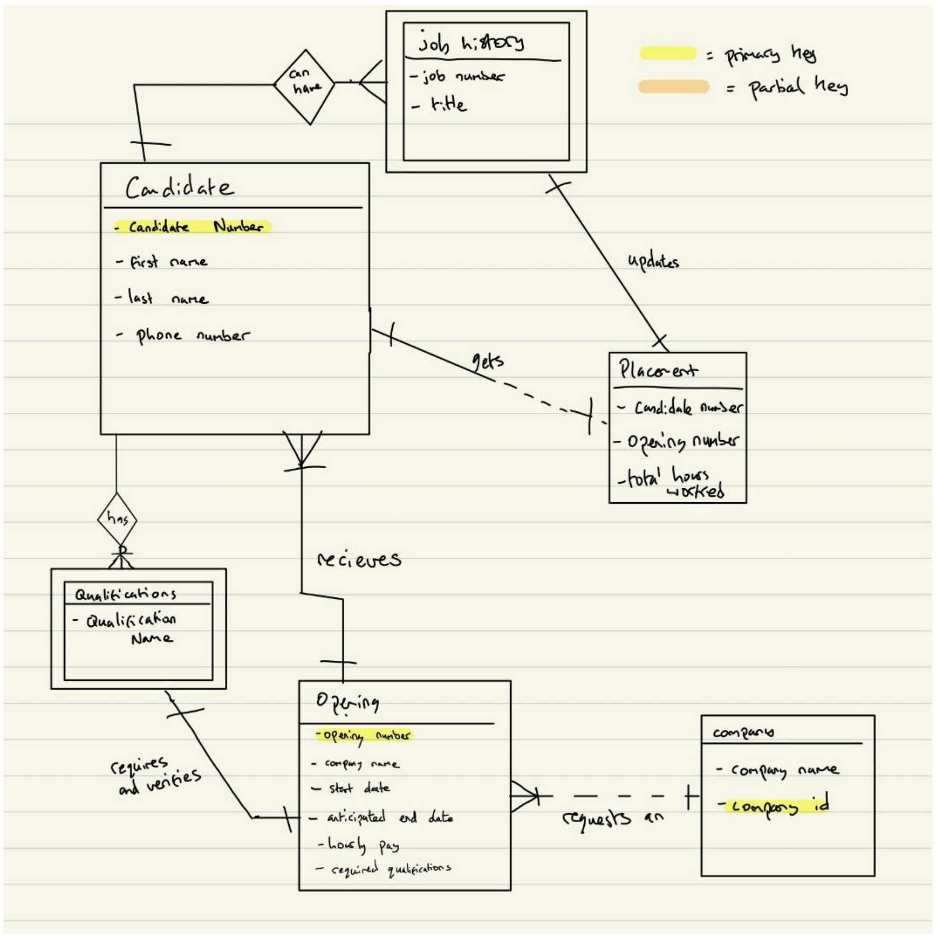
19C0A201 – DATABASES COURSEWORK
NAME: BAILEIGH CORDICE ID: B924898

TASK 1:

A)

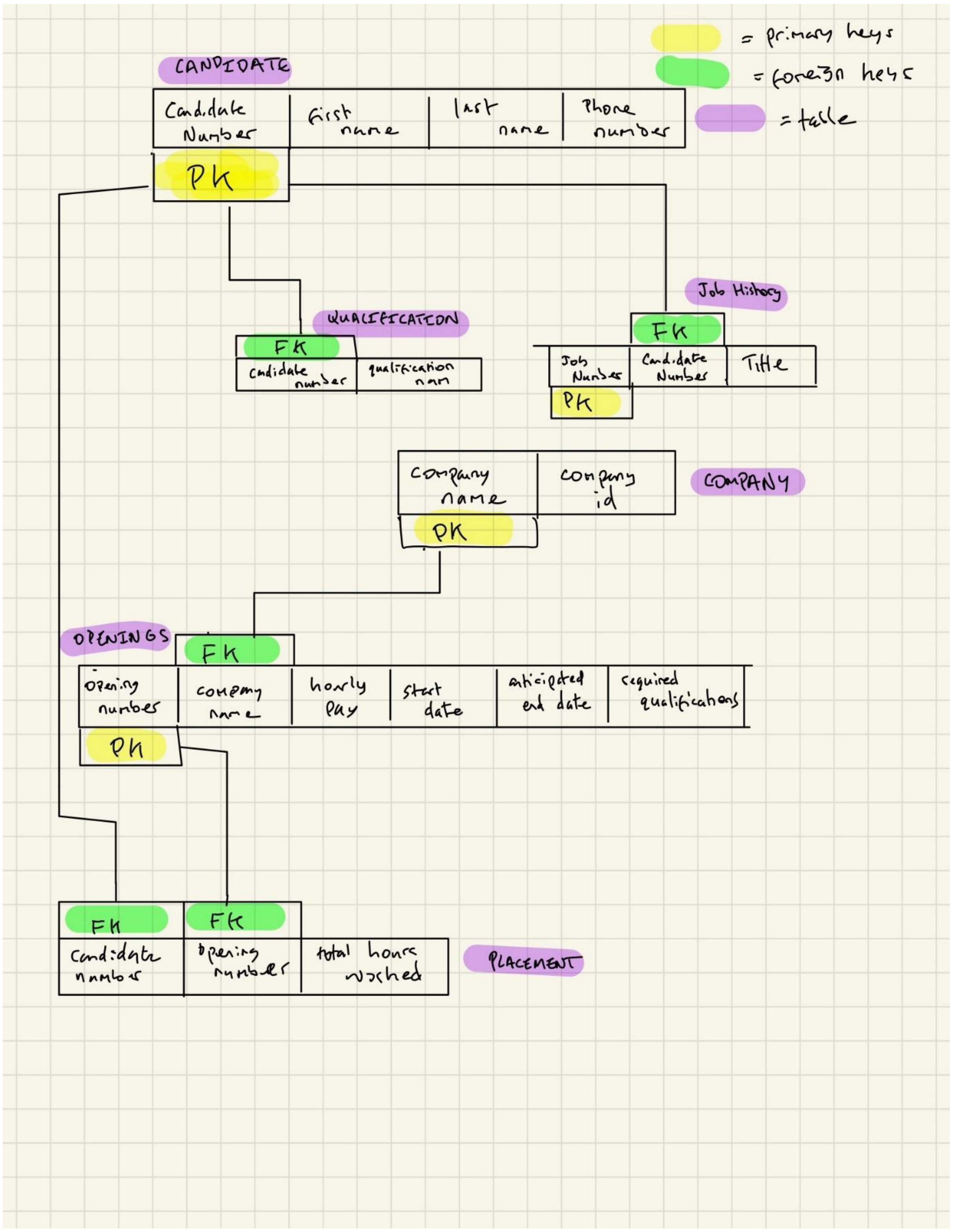


Assumptions:

- A candidate only applies for one job opening (therefore job number is unique to each candidate with that job.
- Each opening requires one qualification
- Assume company names are unique

19C0A201 – DATABASES COURSEWORK
NAME: BAILEIGH CORDICE   ID: B924898

B)



The words in each box asscociated with each table are the attributes.

The placement table is an intersection table of two foreign keys.

19C0A201 – DATABASES COURSEWORK
NAME: BAILEIGH CORDICE ID: B924898

TASK 2:

1NF:
All my tables are 1NF as atomic values and each row in the table can be uniquely identfied.

2NF:

All tables are 1NF and all attributes (non-key columns) depend on a primary key of a particlar table. There are no partial dependencies.

3NF:

My tables are in 3NF as there are no transitive dependencies.

To check if BCNF, it must satisfy two conditions:

- the table must be in 3NF
- For any dependency, A->B, A cannot be a non-prime attribute if B is prime

Candidate Table:

Is BCNF as: Candidate Number -> (Phone number, first name, last name)
Phone Number -> (Candidate Number, first name, last name)

Qualification Table:

Is BCNF as: Candidate Number -> (Qualification name)
Multiple entries could have the same qualification names so it wouldn't work the other way around.

Job History Table:

Is BCNF as: Job Number -> (Candidate Number, Title)
Candidate Number -> (Job Number, Title)

Company Table:

Is BCNF as: Company Name -> (Company ID)

Openings Table:

Is BNCF as: Opening Number -> (Company Name, Hourly pay, Start date, anticipated end date, req. qualifications)
Company Name -> (Opening Number, Hourly pay, Start date, anticipated end date, req. qualifications)

Placement Table:

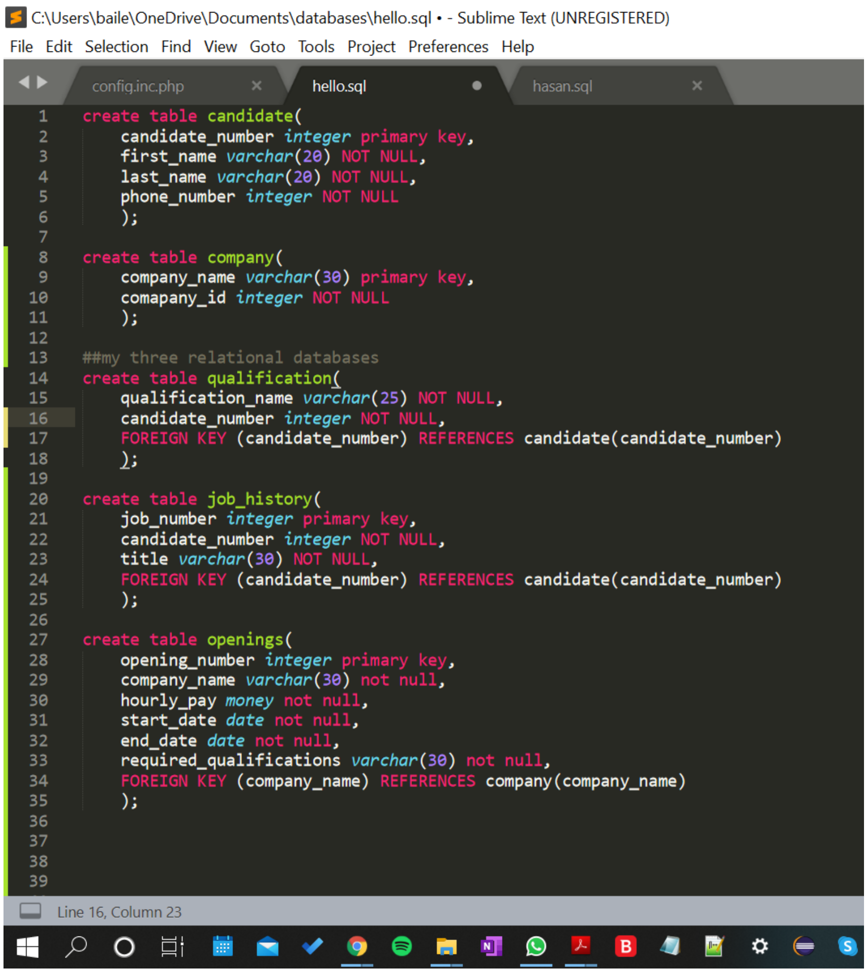
Is BCNF as : Candidate Number ->(Opening Number, total hours worked)
Opening Number -> (Candidate Number, total hours worked)

19C0A201 – DATABASES COURSEWORK
NAME: BAILEIGH CORDICE   ID: B924898

TASK 3:

A)

This code creates my relational databases:

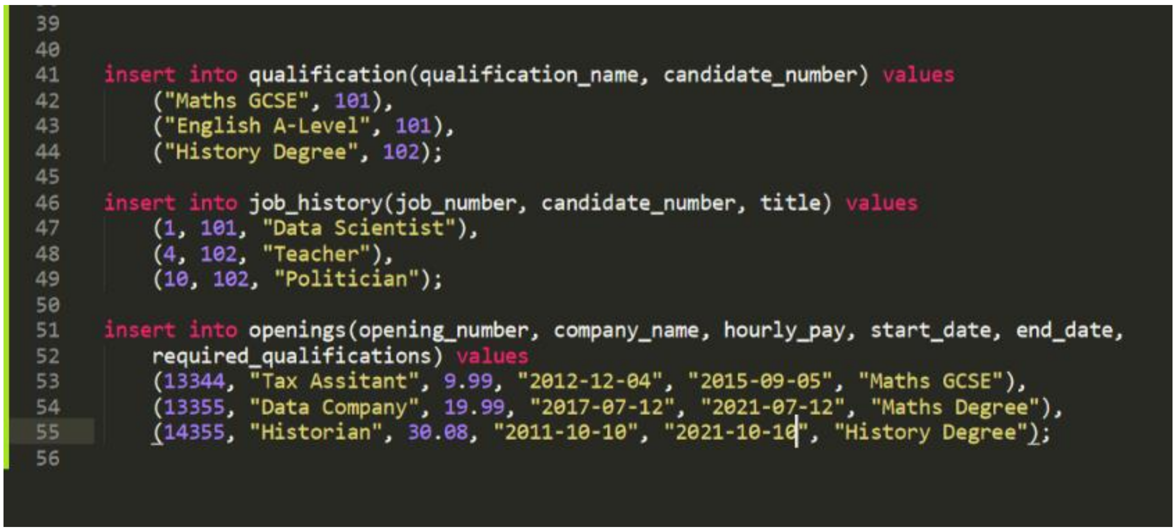

C:\Users\baile\OneDrive\Documents\databases\hello.sql • - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help
config.inc.php
hello.sql
hasan.sql

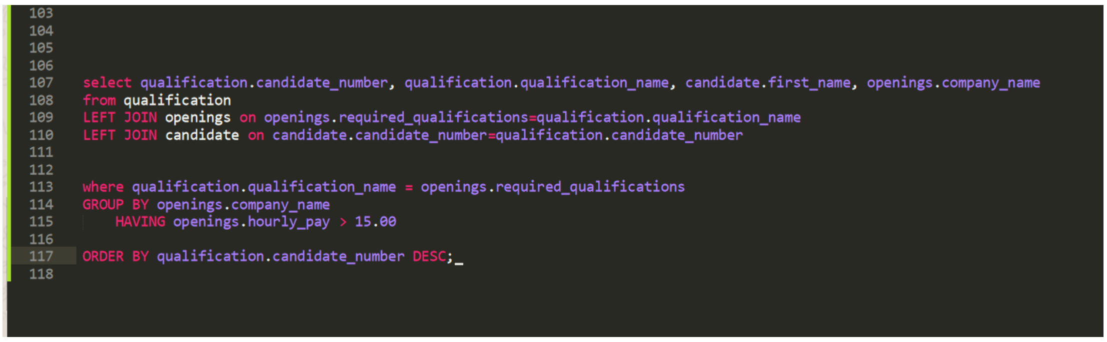
```sql
create table candidate(
    candidate_number integer primary key,
    first_name varchar(20) NOT NULL,
    last_name varchar(20) NOT NULL,
    phone_number integer NOT NULL
    );

create table company(
    company_name varchar(30) primary key,
    comapany_id integer NOT NULL
    );

##my three relational databases
create table qualification(
    qualification_name varchar(25) NOT NULL,
    candidate_number integer NOT NULL,
    FOREIGN KEY (candidate_number) REFERENCES candidate(candidate_number)
    );

create table job_history(
    job_number integer primary key,
    candidate_number integer NOT NULL,
    title varchar(30) NOT NULL,
    FOREIGN KEY (candidate_number) REFERENCES candidate(candidate_number)
    );

create table openings(
    opening_number integer primary key,
    company_name varchar(30) not null,
    hourly_pay money not null,
    start_date date not null,
    end_date date not null,
    required_qualifications varchar(30) not null,
    FOREIGN KEY (company_name) REFERENCES company(company_name)
    );
```

Line 16, Column 23

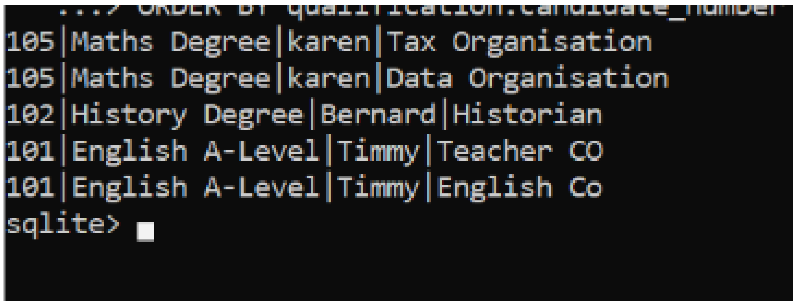
These are my insert statements:


```sql
insert into qualification(qualification_name, candidate_number) values
    ("Maths GCSE", 101),
    ("English A-Level", 101),
    ("History Degree", 102);

insert into job_history(job_number, candidate_number, title) values
    (1, 101, "Data Scientist"),
    (4, 102, "Teacher"),
    (10, 102, "Politician");

insert into openings(opening_number, company_name, hourly_pay, start_date, end_date,
    required_qualifications) values
    (13344, "Tax Assitant", 9.99, "2012-12-04", "2015-09-05", "Maths GCSE"),
    (13355, "Data Company", 19.99, "2017-07-12", "2021-07-12", "Maths Degree"),
    (14355, "Historian", 30.08, "2011-10-10", "2021-10-10", "History Degree");
```

B)



```
select qualification.candidate_number, qualification.qualification_name, candidate.first_name, openings.company_name
from qualification
LEFT JOIN openings on openings.required_qualifications=qualification.qualification_name
LEFT JOIN candidate on candidate.candidate_number=qualification.candidate_number


where qualification.qualification_name = openings.required_qualifications
GROUP BY openings.company_name
    HAVING openings.hourly_pay > 15.00

ORDER BY qualification.candidate_number DESC;_
```



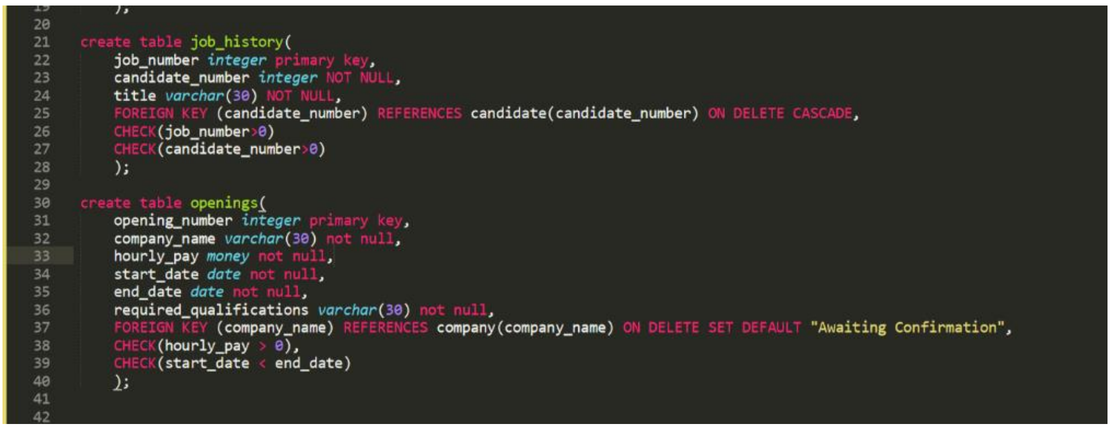
```
105|Maths Degree|karen|Tax Organisation
105|Maths Degree|karen|Data Organisation
102|History Degree|Bernard|Historian
101|English A-Level|Timmy|Teacher CO
101|English A-Level|Timmy|English Co
sqlite>
```

This code displays a table containing the jobs that candidates are eligible for, based on their qualifications, given that the hourly pay is greater than £15.00. The table is then ordered by descending candidate numbers.

The first column shows their candidate number, the second shows their qualifications, the third shows their first name, and the fourth shows the job title.

TASK 4:

A)

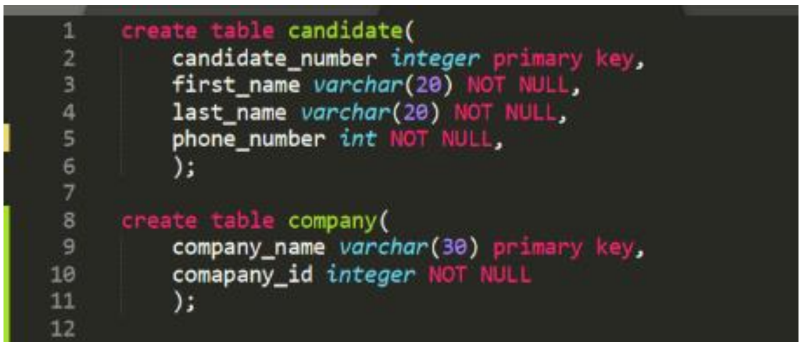


```
    );

create table job_history(
    job_number integer primary key,
    candidate_number integer NOT NULL,
    title varchar(30) NOT NULL,
    FOREIGN KEY (candidate_number) REFERENCES candidate(candidate_number) ON DELETE CASCADE,
    CHECK(job_number>0)
    CHECK(candidate_number>0)
    );

create table openings(
    opening_number integer primary key,
    company_name varchar(30) not null,
    hourly_pay money not null,
    start_date date not null,
    end_date date not null,
    required_qualifications varchar(30) not null,
    FOREIGN KEY (company_name) REFERENCES company(company_name) ON DELETE SET DEFAULT "Awaiting Confirmation",
    CHECK(hourly_pay > 0),
    CHECK(start_date < end_date)
    );
```

For my job_history table, if I delete candidate_number from the candidate table, (because it no longer has a candidate_number) the entire row containing candidate(candidate_number) will be deleted from the job_history table.

For my openings table, if a row containing company(company_name) is deleted from the company table (e.g if a company is rebranding such as changing their name) their details aren't deleted in the openings table – the row containing that company_name is changed to "Awaiting Confirmation". This means that candidates are still able to apply to work at the company, as well as compare their pay and start dates against other openings they are also suitable for.
However, a potential disadvantage of this could be that it may be hard to track which opening belongs to what company if they all say "Awaiting Confirmation".

Here are my **candidate** and **company tables** for reference:



```
create table candidate(
    candidate_number integer primary key,
    first_name varchar(20) NOT NULL,
    last_name varchar(20) NOT NULL,
    phone_number int NOT NULL,
    );

create table company(
    company_name varchar(30) primary key,
    comapany_id integer NOT NULL
    );
```

B)

If I wanted to see the name of each candidate with an English A-Level and include their other candidate information, I could denormalise my tables in 3NF from:

> Candidate Number -> (Phone number, first name, last name) **[Candidate Table]**
> Candidate Number -> (Qualification name) **[Qualification Table]**

To:

> Candidate Number -> first name, last name, phone number, qualification name

This could make is easier for people (who have access to the database) to contact candidates by phone number if they have suitable qualifications.

If I wanted to find the names of people with a previous job history of working at “Data Company” I could denormalise from:

> Candidate Number -> (Phone number, first name, last name) **[Candidate Table]**
> Candidate Number -> (Job Number, Title) **[Job History Table]**

To:

> Candidate Number -> first name, last name, phone number, Job Number, Title

This would allow a faster way to verify where candidates have previously worked (because I can see their name and surname) and I potentially recommend applicable openings for them.

Both of these examples eliminate the need for joins, but will increase the amount of redundant data. However the queries are critical so de-normalising is acceptable.