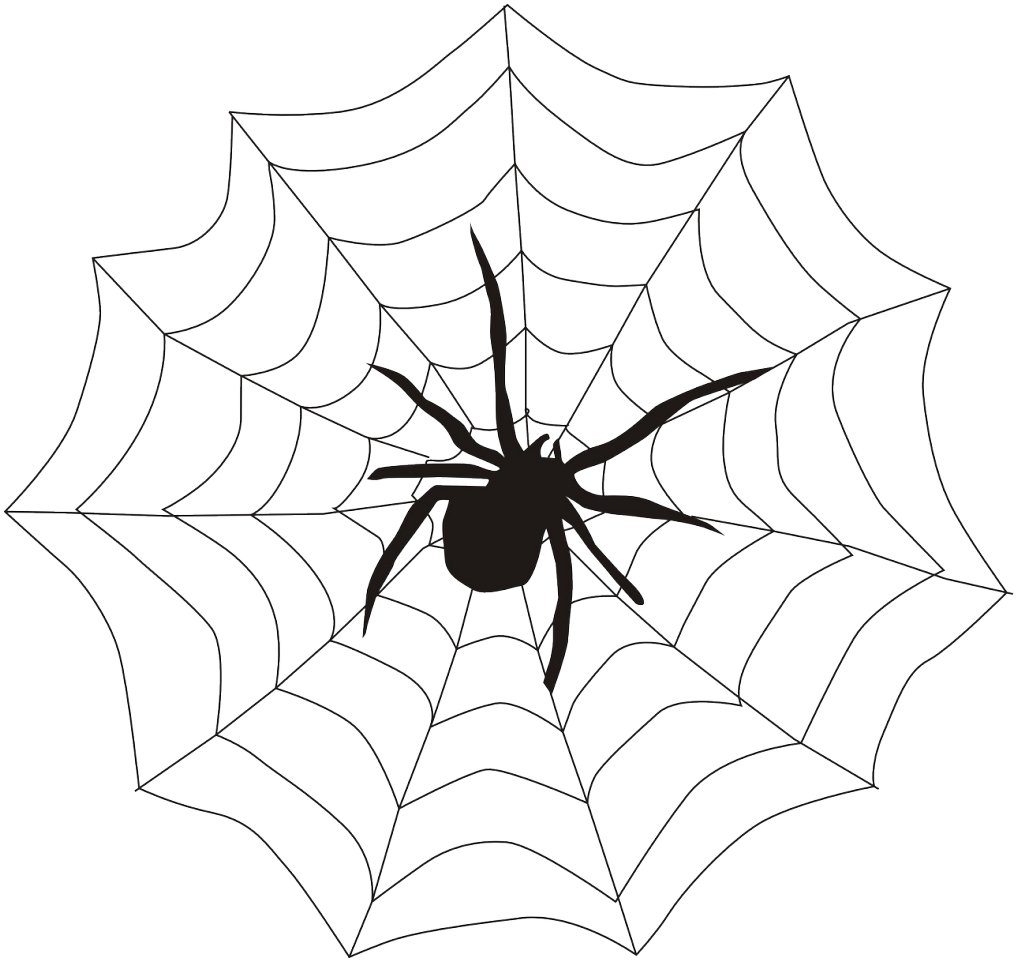
HackerFrogs Afterschool
Intro to SQL /w SQL Murder Mystery

Class:
Web App Hacking

Workshop Number:
AS-WEB-05

Document Version:
1.2

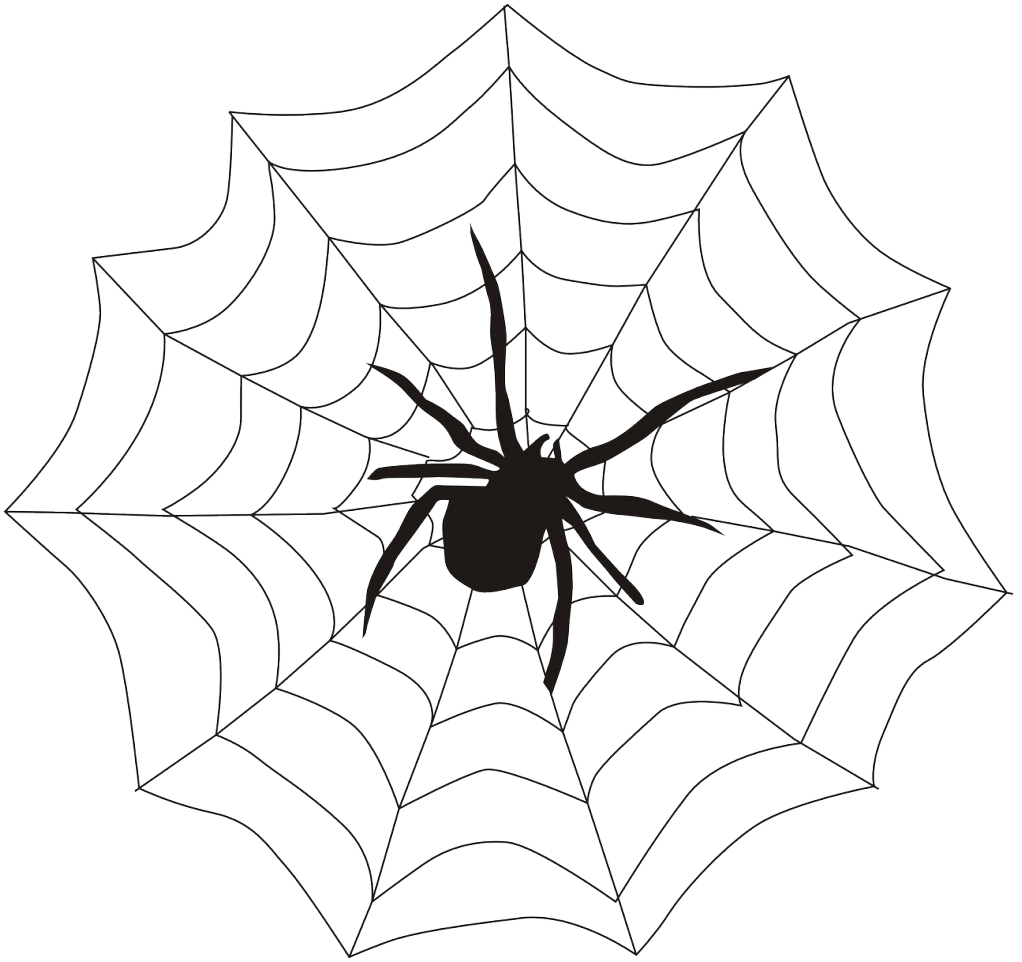
Special Requirements:
None



What We Learned In The Previous Workshop

This is the fifth workshop for intro to web app hacking.

Let's take a few moments to review the concepts we learned in the previous workshop.



Sourcecode Analysis

Sourcecode analysis is the process of analyzing the code of a piece of software with the goal of deeper understanding regarding its function.

```
<?

include "includes/secret.inc";

    if(array_key_exists("submit", $_POST)) {
        if($secret == $_POST['secret']) {
        print "Access granted. The password for
    } else {
        print "Wrong secret";
    }
    }
?>
```

Sourcecode Analysis

It is a common technique used in software security testing, but it requires the software sourcecode be readily available.

```
<?

include "includes/secret.inc";

    if(array_key_exists("submit", $_POST)) {
        if($secret == $_POST['secret']) {
        print "Access granted. The password for
    } else {
        print "Wrong secret";
    }
    }
?>
```

OS Command Injection

Operating System (OS) Command Injection is a web app vulnerability where arbitrary OS commands can be performed on the webserver through a web interface.

OS Command Injection

OS Command Injection can often lead to complete compromise of the webserver, and if so, the server can be used as a foothold to attack other machines on the network.

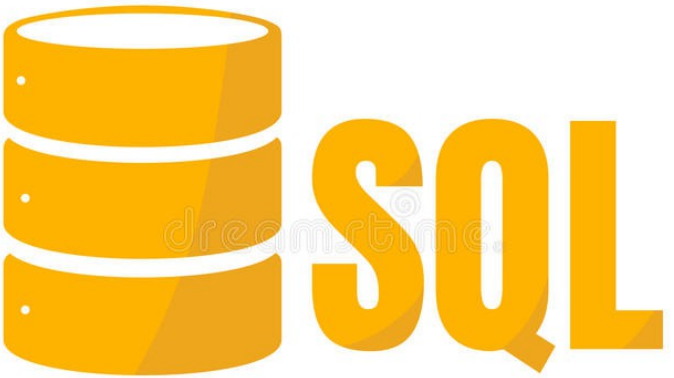
Now On To Our Topic!

Let's move on to the topic of this workshop:

The SQL database language

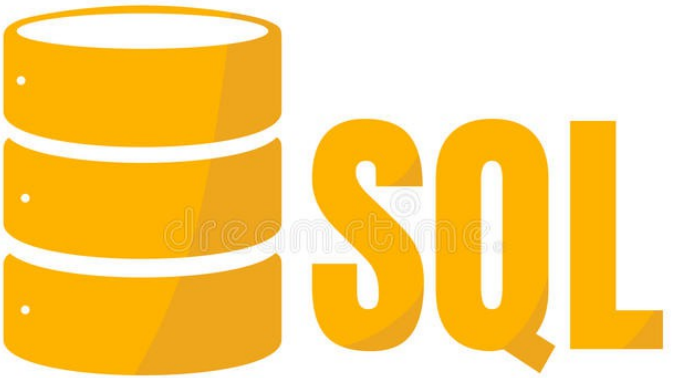
What is SQL?

Structured Query Language (SQL) is a programming language used for managing data held in relational databases.



What is SQL?

Simply put, SQL allows users to access and manage data contained in relational databases, which are common components in most web applications.



Why Learn About SQL?

As far as web app hacking is concerned, insecure implementation of SQL in web apps can lead to a very serious vulnerability called SQL Injection.

Why Learn About SQL?

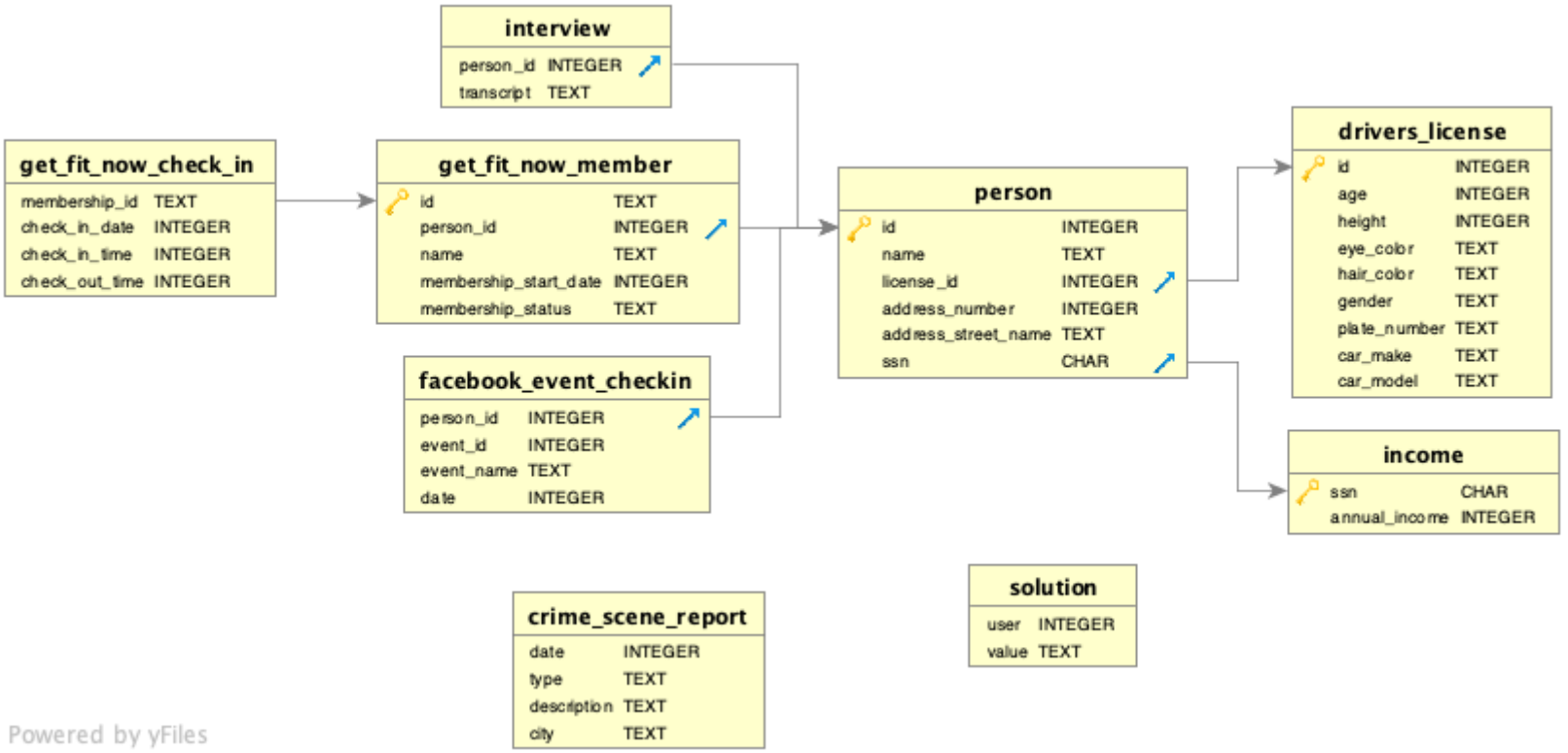
We will learn about SQL Injection in a later workshop, but we should first learn how to use SQL in its intended manner before we learn how to abuse it.

Let's Learn SQL Through a Game

In a web browser, navigate to the following URL:

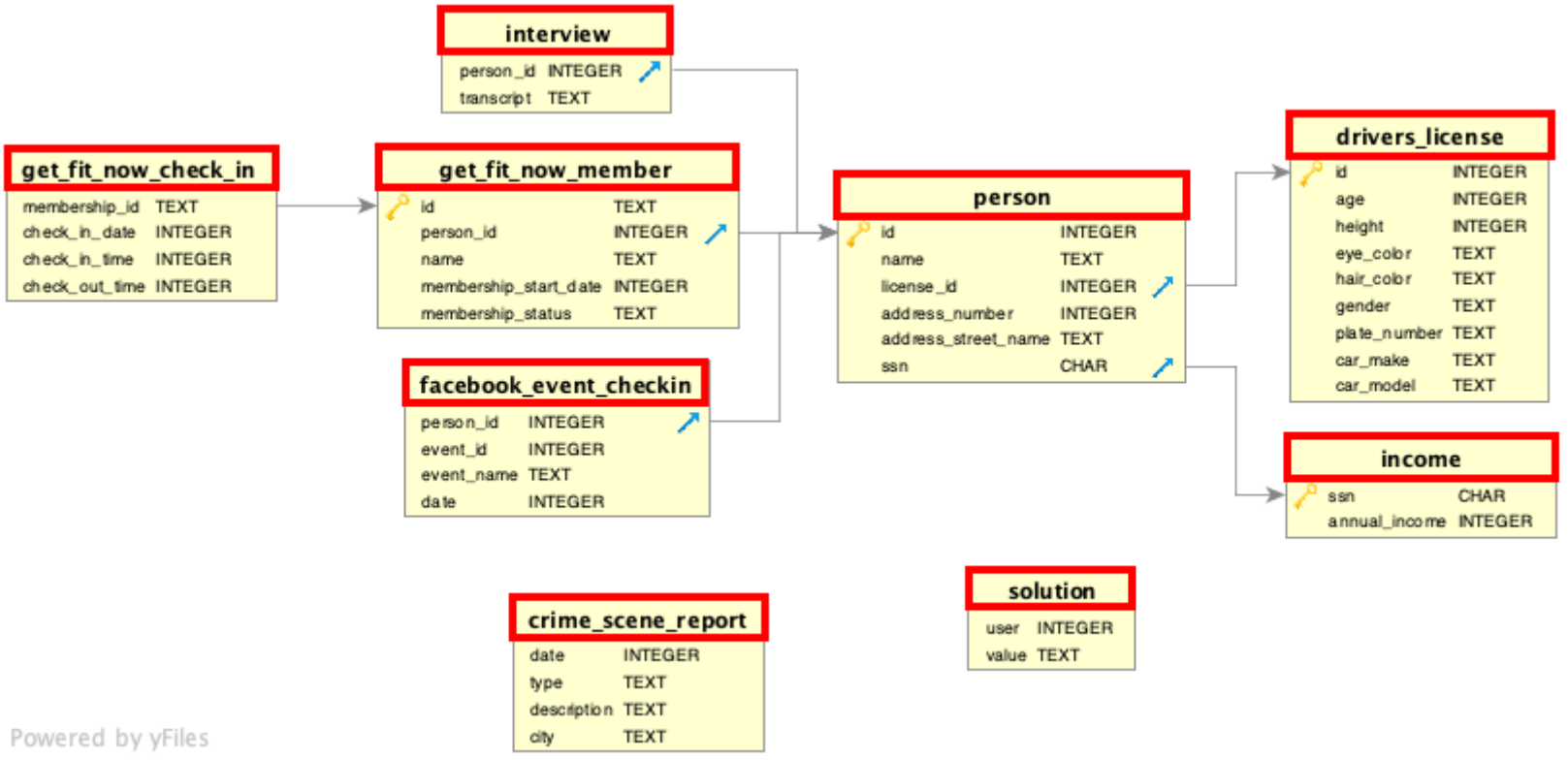
https://mystery.knightlab.com/walkthrough.html

SQL Database Structure



The largest unit in a SQL system is the **database**,

SQL Database Structure



The largest unit in a SQL system is the **database**, which contain one or more **tables**.

SQL Database Structure

id	name	license_id	address_number	address_street_name	ssn
10000	Christoper Peteuil	993845	624	Bankhall Ave	747714076
10007	Kourtney Calderwood	861794	2791	Gustavus Blvd	477972044
10010	Muoi Cary	385336	741	Northwestern Dr	828638512

Tables themselves are made up of two types of elements:

SQL Database Structure

id	name	license_id	address_number	address_street_name	ssn
10000	Christoper Peteuil	993845	624	Bankhall Ave	747714076
10007	Kourtney Calderwood	861794	2791	Gustavus Blvd	477972044
10010	Muoi Cary	385336	741	Northwestern Dr	828638512

Columns, whose names identify the data that should be entered into them...

SQL Database Structure

id	name	license_id	address_number	address_street_name	ssn
10000	Christoper Peteuil	993845	624	Bankhall Ave	747714076
10007	Kourtney Calderwood	861794	2791	Gustavus Blvd	477972044
10010	Muoi Cary	385336	741	Northwestern Dr	828638512

And **rows**, which contain data entries that correspond to the columns they fall under.

The SELECT Command

```
1 SELECT count(*)
2 FROM person;
```

The **SELECT** command is the most common SQL command, which returns SQL entries depending on what additional parameters are provided.

The FROM Command

```
1 SELECT count(*)
2 FROM person;
```

The **FROM** command specifies which table to retrieve data from.

The Count Function

```
1 SELECT count(*)
2 FROM person;
```

count(*)
10011

The SQL **count** function will return the number of entries in the specified column.

The LIMIT Filter

```
1 SELECT * FROM person LIMIT 2;
```

id	name	license_id	address_number	address_street_name	ssn
10000	Christoper Peteuil	993845	624	Bankhall Ave	747714076
10007	Kourtney Calderwood	861794	2791	Gustavus Blvd	477972044

The **LIMIT** filter restricts the number of entries returned to the number specified.

The * Wildcard Character

```
1 SELECT * FROM person LIMIT 2;
```

id	name	license_id	address_number	address_street_name	ssn
10000	Christoper Peteuil	993845	624	Bankhall Ave	747714076
10007	Kourtney Calderwood	861794	2791	Gustavus Blvd	477972044

The * wildcard character can be used in the place of column names in a SQL query, and represents all columns in a specified table.

The DISTINCT Clause

```
1 SELECT DISTINCT type FROM crime_scene_report;
```

The **DISTINCT** clause will restrict the number of results returned by the query so that there will be no duplicate values.

The WHERE Filter

```
1 SELECT * FROM person WHERE name = 'Kinsey Erickson'
```

id	name	license_id	address_number	address_street_name	ssn
89906	Kinsey Erickson	510019	309	Northwestern Dr	635287661

The **WHERE** filter is the most common SQL filter, and it restricts the results that are returned according to arguments given to it.

The WHERE Filter

```
1 SELECT * FROM person WHERE name = 'Kinsey Erickson'
```

id	name	license_id	address_number	address_street_name	ssn
89906	Kinsey Erickson	510019	309	Northwestern Dr	635287661

Here, the results are restricted to rows where the **name** column has the text **Kinsey Erickson**.

The = Operator

```
1 SELECT * FROM person WHERE name = 'kinsey Erickson'
```

No data returned

The = operator is used with the WHERE filter to return exact matches. If the contents of a row differ by even a single character, the match will not return. (The 'k' in the above example is not capitalized, so a valid match is not found)

The AND Clause

```
1 SELECT * FROM person WHERE name = 'John Dile'
2 AND address_street_name = 'Icehouse Ave'
```

id	name	license_id	address_number	address_street_name	ssn
13915	John Dile	909334	3783	Icehouse Ave	957131634

The **AND** clause is used with the **WHERE** filter to further restrict the results returned, only returning rows that match both parameters on either side of the **AND** clause.

The Like Operator and % Wildcard

```
1 SELECT * FROM person WHERE name like '%John%'
2 AND address_street_name like '%Icehouse%'
```

id	name	license_id	address_number	address_street_name	ssn
13915	John Dile	909334	3783	Icehouse Ave	957131634

The **like** operator is used with the % wildcard character to return partial matches. The % symbol matches any characters on its respective side of the string.

The Upper and Lower Functions

```
1 SELECT upper(name), lower(name) FROM person
2 WHERE name = 'Kinsey Erickson'
```

upper(name)	lower(name)
KINSEY ERICKSON	kinsey erickson

The SQL **upper** and **lower** functions are used to render output in all uppercase or lowercase letters, respectively.

The Upper and Lower Functions

```
1 SELECT name FROM person
2 WHERE lower(name) = 'kinsey erickson'
```

name
Kinsey Erickson

It can also be used in the WHERE portion of a query to match results where the capitalization of a search term is unknown.

The Sum Function

```
1 SELECT sum(age)
2 FROM drivers_license;
```

sum(age)
532265

The **sum** function is used to return the sum of all entries in a specified column.

The Avg Function

```
1 SELECT avg(age)
2 FROM drivers_license;
```

avg(age)
53.18926751274108

The **avg** function is used to return the average value of all entries in a specified column.

The MAX and MIN Functions

```
1 SELECT max(age)
2 FROM drivers_license;
```

max(age)
89

The **max** and **min** functions are used to return either the maximum or minimum values of a specified column, respectively.

The ORDER BY Keyword

```
1 SELECT * FROM drivers_license ORDER BY id ASC LIMIT 2
```

id	age	height	eye_color	hair_color	gender	plate_number	car_make	car_model
100280	72	57	brown	red	male	P24L4U	Acura	MDX
100460	63	72	brown	brown	female	XF02T6	Cadillac	SRX

The ORDER BY keyword is used to return rows in ascending order (A-Z or small to large numbers), depending on the name of the column provided.

The ASC and DESC Keywords

```
1 SELECT * FROM drivers_license ORDER BY id ASC LIMIT 2
```

id	age	height	eye_color	hair_color	gender	plate_number	car_make	car_model
100280	72	57	brown	red	male	P24L4U	Acura	MDX
100460	63	72	brown	brown	female	XF02T6	Cadillac	SRX

The ASC and DESC keywords are used with the ORDER BY keyword to return rows in ascending or descending order, respectively.

Part 2: Obtaining the Crime Report

We know that the **type** of crime is **murder**, the **date** of the crime was Jan 15 2018 (**20180115**), and the **city** was **SQL City**.

Using this information, obtain info from the **crime_scene_report** table.

Part 2: Obtaining the Crime Report

We can use the following query to retrieve the crime report:

```
SELECT * FROM crime_scene_report
WHERE type = 'murder' AND city = 'SQL
City' AND date = 20180115;
```

Part 3: Identifying the Witnesses

Now that we know one of the witnesses lives at the **last address** on **Northwestern Dr**, and another, whose **name** is **Annabel**, lives on **Franklin Ave**.

Send one or two queries to the SQL database **person** table to obtain the full names of the witnesses. Remember partial matches (`name LIKE 'Example%'`)

Part 3: Identifying the Witnesses

We can use the following query to identify the first witness:

```
SELECT * FROM person WHERE
address_street_name = 'Northwestern
Dr' ORDER BY address_number DESC
LIMIT 1;
```

Part 3: Identifying the Witnesses

We can use the following query to identify the second witness:

```
SELECT * FROM person WHERE name LIKE
'Annabel%' AND address_street_name =
'Franklin Ave';
```

Part 4: Retrieving the Interview Transcipts

Now that we have the witness names, we can make a query with data from both the **person** and **interview** tables.

But before we can do that, we must learn about a few more SQL keywords...

JOIN Queries

```sql
SELECT person.name, income.annual_income
FROM income
JOIN person
  ON income.ssn = person.ssn
WHERE annual_income > 450000
```

JOIN queries are queries that include information from more than one table.

JOIN Queries

```sql
SELECT person.name, income.annual_income
FROM income
JOIN person
  ON income.ssn = person.ssn
WHERE annual_income > 450000
```

Here we are selecting data from both the **person** table and the **income** table. To do so, the **JOIN** clause must be used.

JOIN Queries

When returned, the query incorporates search criteria from both the **income** and **person** tables.

name	annual_income
Claudio Carlan	473100
Felice Prudden	486600

JOIN Queries

```
1 SELECT person.name, income.annual_income
2 FROM income
3 JOIN person
4   ON income.ssn = person.ssn
5 WHERE annual_income > 450000
```

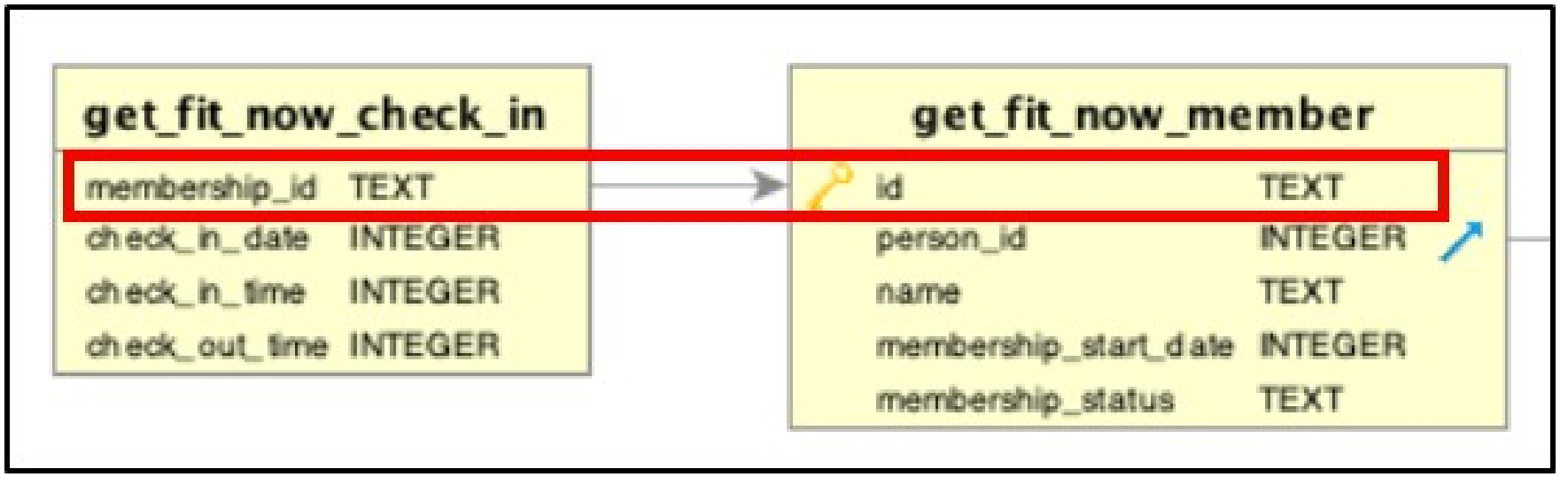
One import thing to notice is the ON clause (line 4), which provides the system with the columns which connect the tables involved in the query.

JOIN Queries

```sql
SELECT person.name, income.annual_income
FROM income
JOIN person
  ON income.ssn = person.ssn
WHERE annual_income > 450000
```

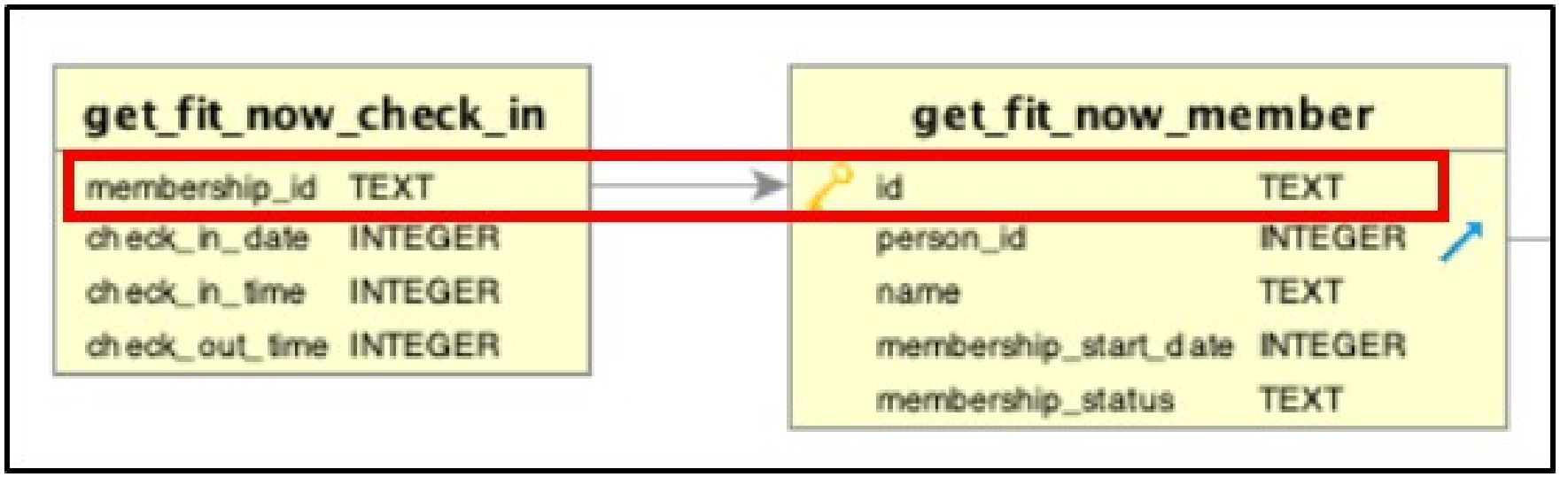
These connecting columns are called the **primary key** and **foreign key**. The columns are connected through the relational nature of SQL systems.

Relational Databases



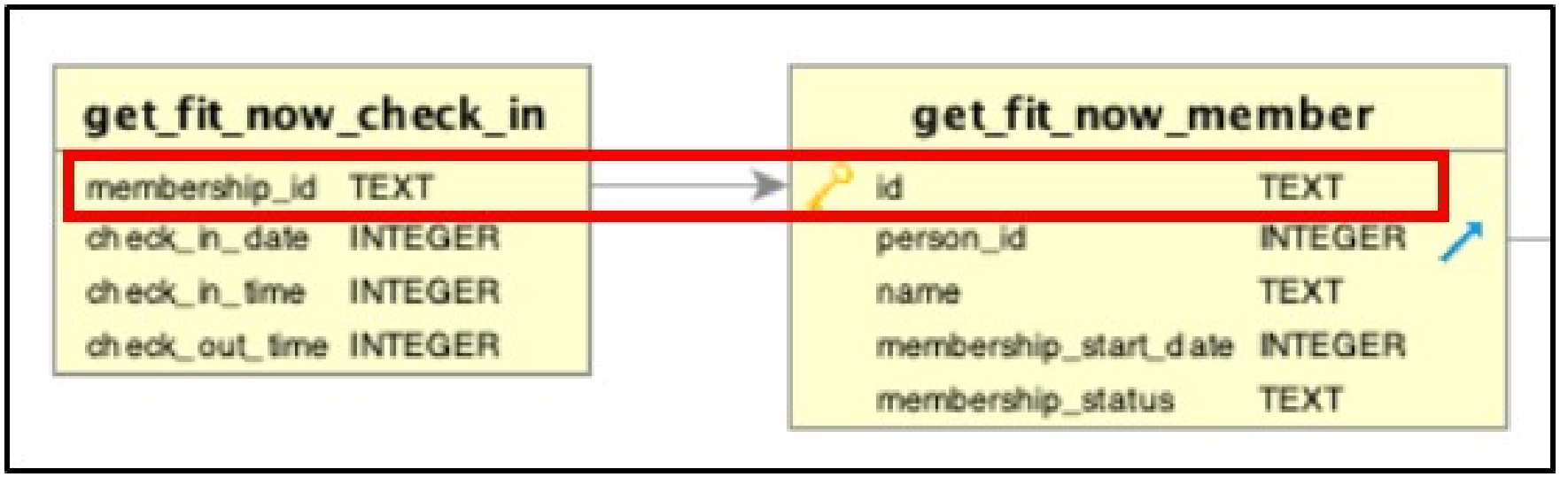
Relational databases are databases where data in one table in the database can be connected to data in another table in the database.

Relational Databases



In this database, the **get_fit_now_check_in** table and the **get_fit_now_member** table are related through the **membership_id** and **id** columns, respectively.

Relational Databases



In this relationship, the **id** column is the **primary key**, and the **membership_id** column is the **foreign key**.

The OR Clause

```
1 SELECT * FROM person WHERE name = 'Kinsey Erickson'
2 OR name = 'John Dile'
```

id	name	license_id	address_number	address_street_name	ssn
13915	John Dile	909334	3783	Icehouse Ave	957131634
89906	Kinsey Erickson	510019	309	Northwestern Dr	635287661

The **OR** clause is another clause we can use with the **WHERE** filter, and it returns entries if a match is found for either of the two conditions on either side of it.

Part 4: Retrieving the Interview Transcipts

Now we have enough information about SQL JOIN queries, so we can make a query with data from both the **person** and **interview** tables.

Part 4: Retrieving the Interview Transcipts

We need to **SELECT** the **person.name** and **interview.transcript** columns, **JOIN** ing the two tables **ON** the **interview.person_id** and **person.id** columns.

Remember, the **name** is either **Annabel Miller OR Morty Schapiro**

Part 4: Retrieving the Interview Transcipts

The following query will retrieve both interview entries at the same time:

```
SELECT
person.name,interview.transcript
FROM person JOIN interview ON
interview.person_id = person.id
WHERE name = 'Annabel Miller'
OR name = 'Morty Schapiro';
```

Part 5: Identifying the Murderer

We obtained the following information about the murderer from the witnesses:

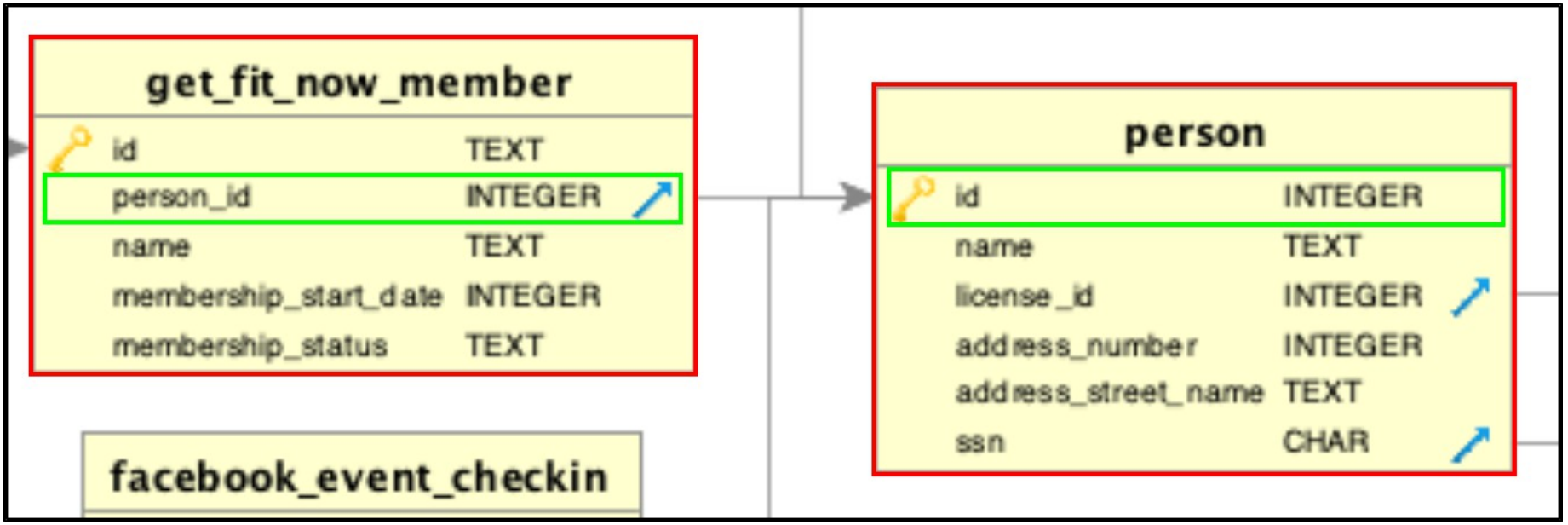
- The suspect is **male**
- He had a **Get Fit Now Gym** bag (**gold** member)
- The gym membership number starts with **48Z**
- The suspect's license plate number contains the string **H42W**
- The suspect was at the gym on Jan 9 (**20180109**)

Part 5: Identifying the Murderer

To make all the JOIN queries, we need to know what tables / columns are involved:

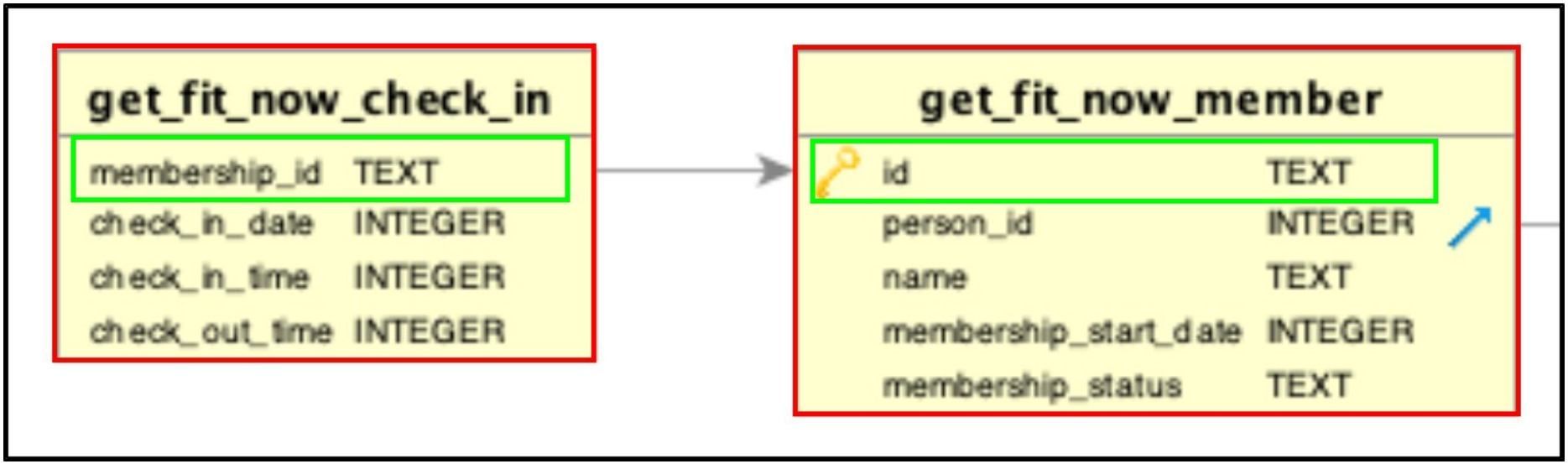
- The suspect is **male** (person.name)
- Get Fit Now Gym **gold** member (get_fit_now_member.membership_status)
- Gym membership number starts with **48Z** (get_fit_now_member.id)
- License plate number contains **H42W** (drivers_license.plate_number)
- Was at the gym on Jan 9, **20180109** (get_fit_now_check_in.check_in_date)

Part 5: Identifying the Murderer



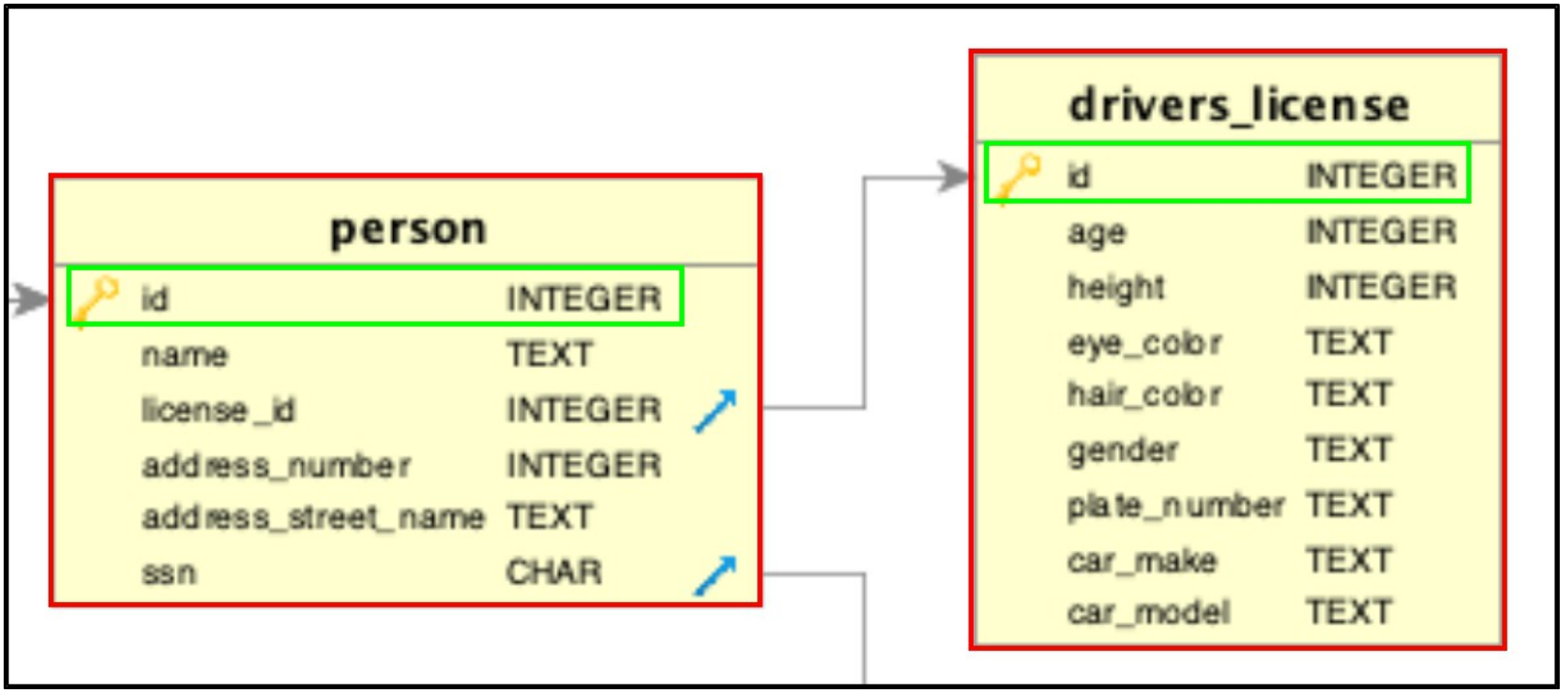
on get_fit_now_member.person_id = person.id

Part 5: Identifying the Murderer



on get_fit_now_member.id =
get_fit_now_check_in.membership_id

Part 5: Identifying the Murderer



on drivers_license.id = person.license_id

Part 5: Identifying the Murderer

The following query will retrieve the name of the murderer:

Part 5: Identifying the Murderer

```
SELECT person.name FROM person
JOIN get_fit_now_member
ON person.id = get_fit_now_member.person_id
JOIN get_fit_now_check_in
ON get_fit_now_check_in.membership_id =
get_fit_now_member.id
JOIN drivers_license
ON drivers_license.id = person.license_id
WHERE LOWER(get_fit_now_member.membership_status)
= 'gold'
AND get_fit_now_check_in.check_in_date = 20180109
AND drivers_license.plate_number LIKE '%H42W%'
AND get_fit_now_member.id LIKE '48Z%';
```

Part 5: Identifying the Murderer

And to confirm the identity of the murderer, we should insert the following into the **Check your solution box**:

```
INSERT INTO solution VALUES (1, 'Jeremy Bowers'); SELECT value FROM solution;
```

Part 6: Identifying the Mastermind

There is one more person to identify before we finish this exercise, the person who hired the murderer. First we have to obtain the murderer's interview data:

```
SELECT person.name,interview.transcript
FROM person JOIN interview
ON interview.person_id = person.id
WHERE name = 'Jeremy Bowers';
```

Part 6: Identifying the Mastermind

We are told the following information:

Gender: female
Income: high
Height: 65-67 inches
Hair color: red
Car make / model: Tesla Model S
Event attended: SQL Symphony Concert (Dec 2017)

Part 6: Identifying the Mastermind

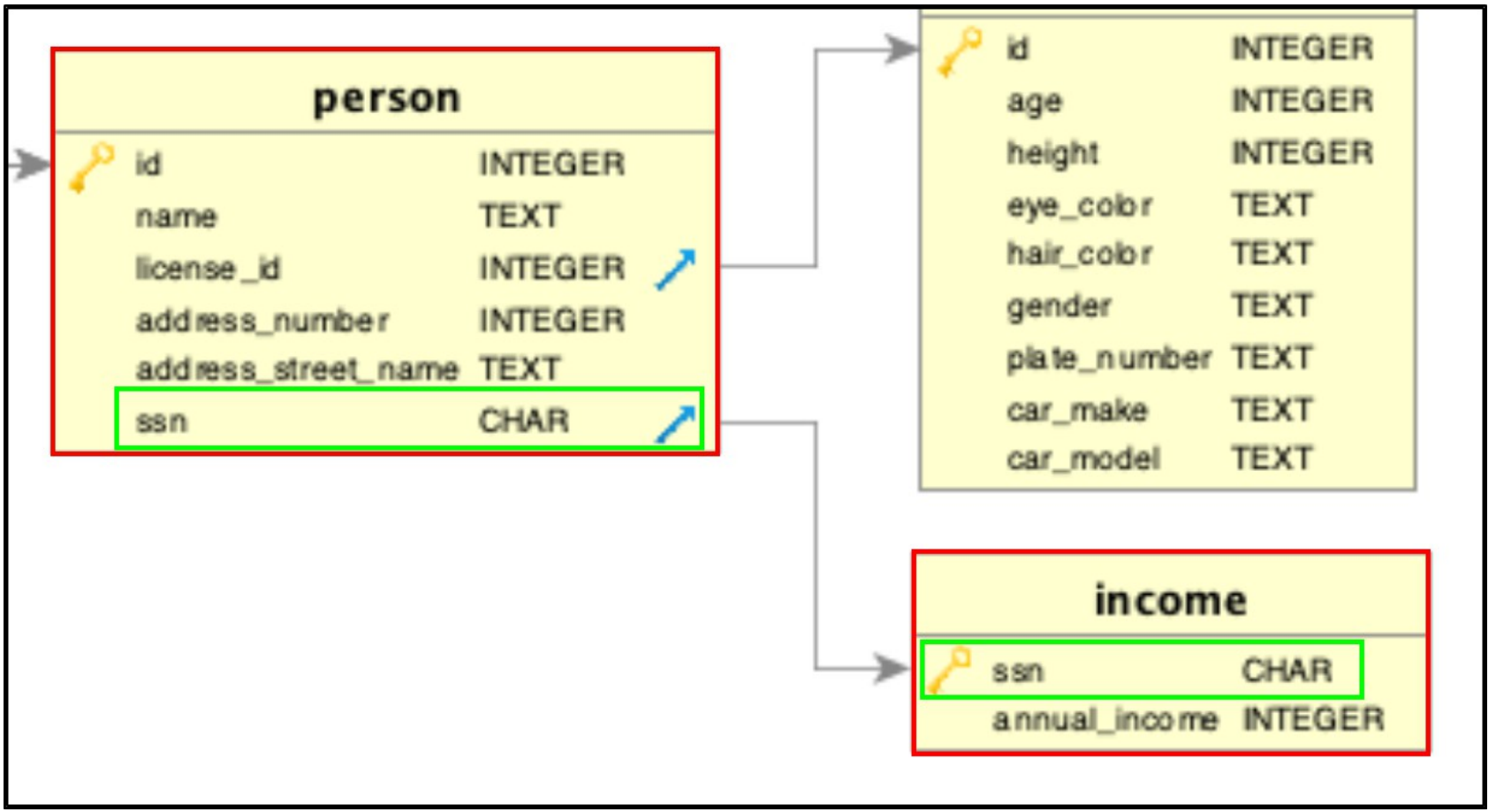
The following tables / columns are involved in the query:

person.name
facebook_event_checkin.event_name
drivers_license.height
drivers_license.gender
drivers_license.car_make
drivers_license.car_model
income.annual_income

Part 6: Identifying the Mastermind

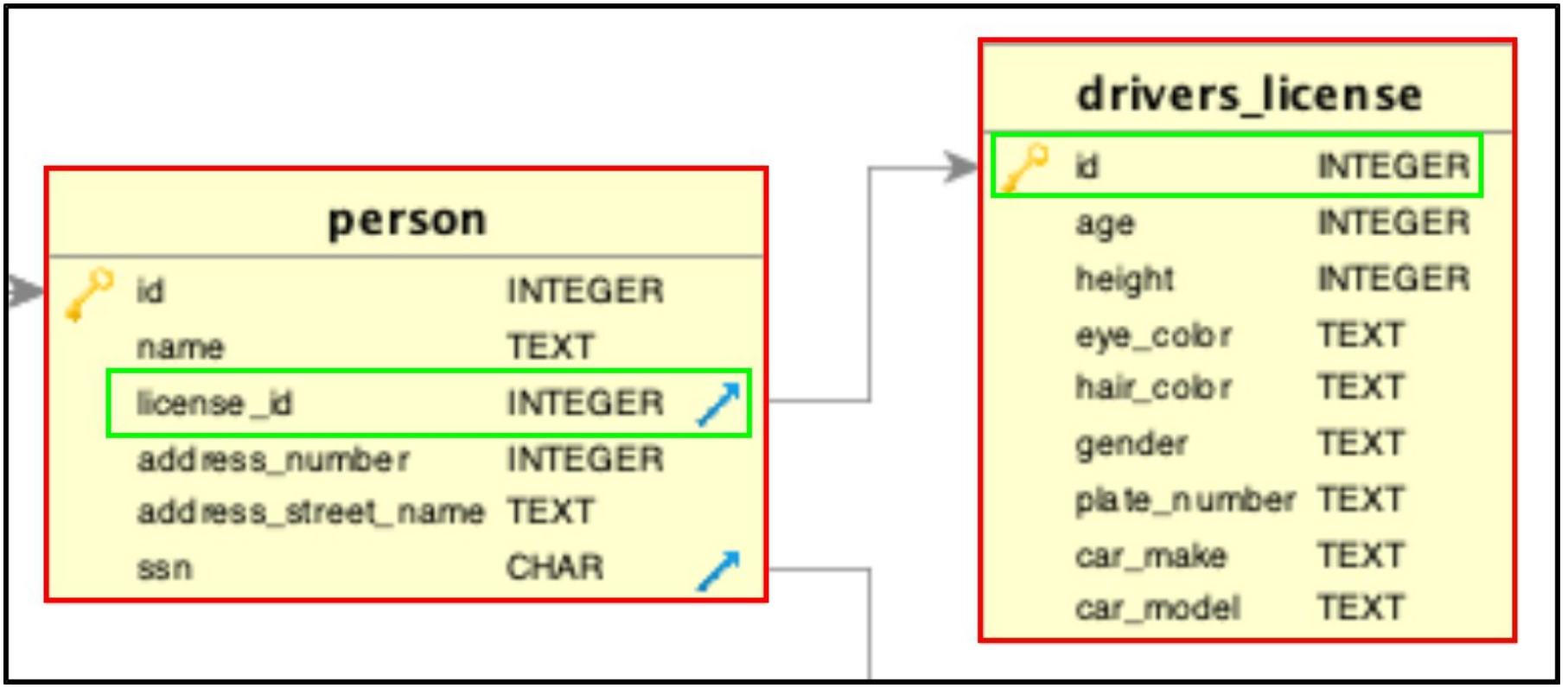
Looking at the ERD, we will need to join the following tables:

Part 6: Identifying the Mastermind



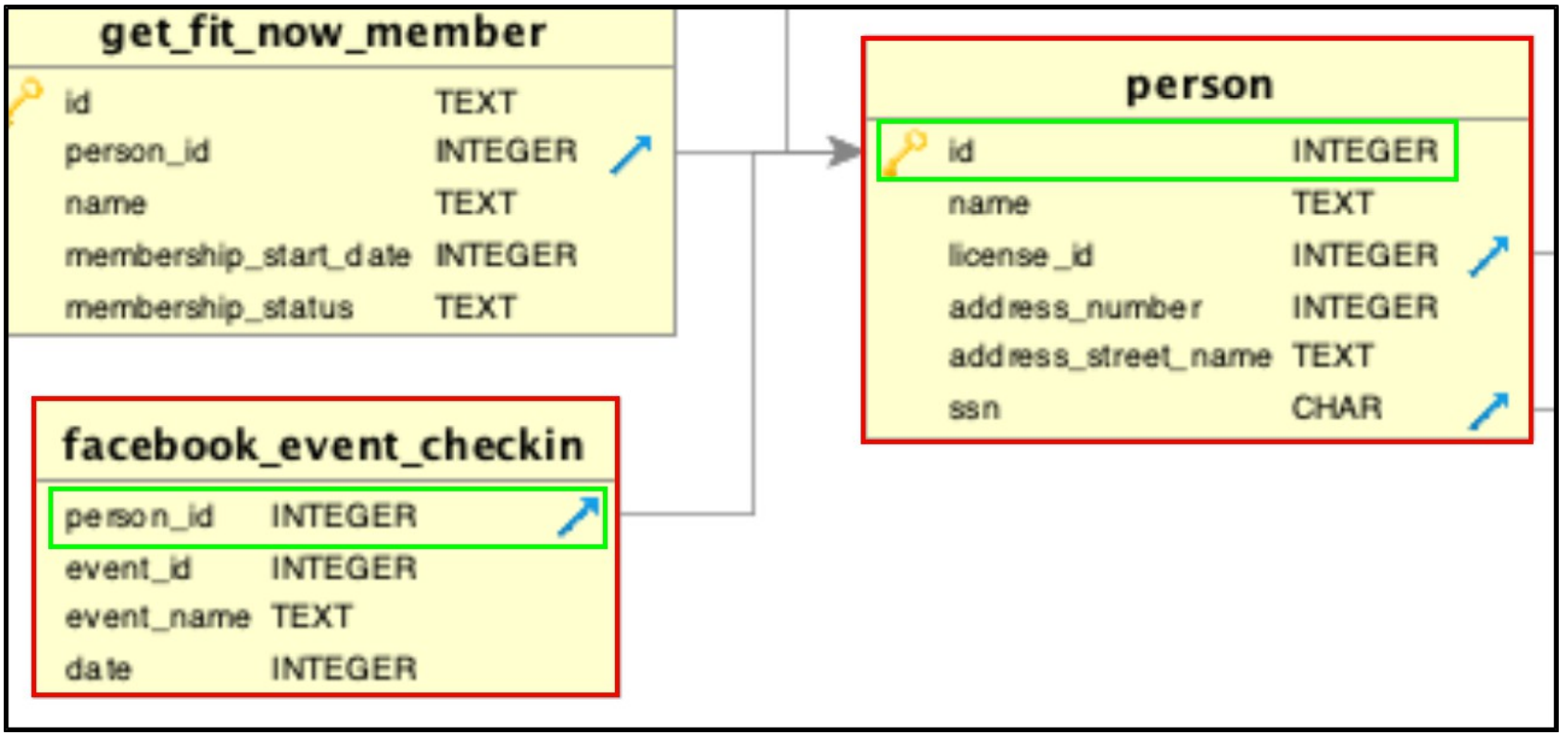
on person.ssn = income.ssn

Part 6: Identifying the Mastermind



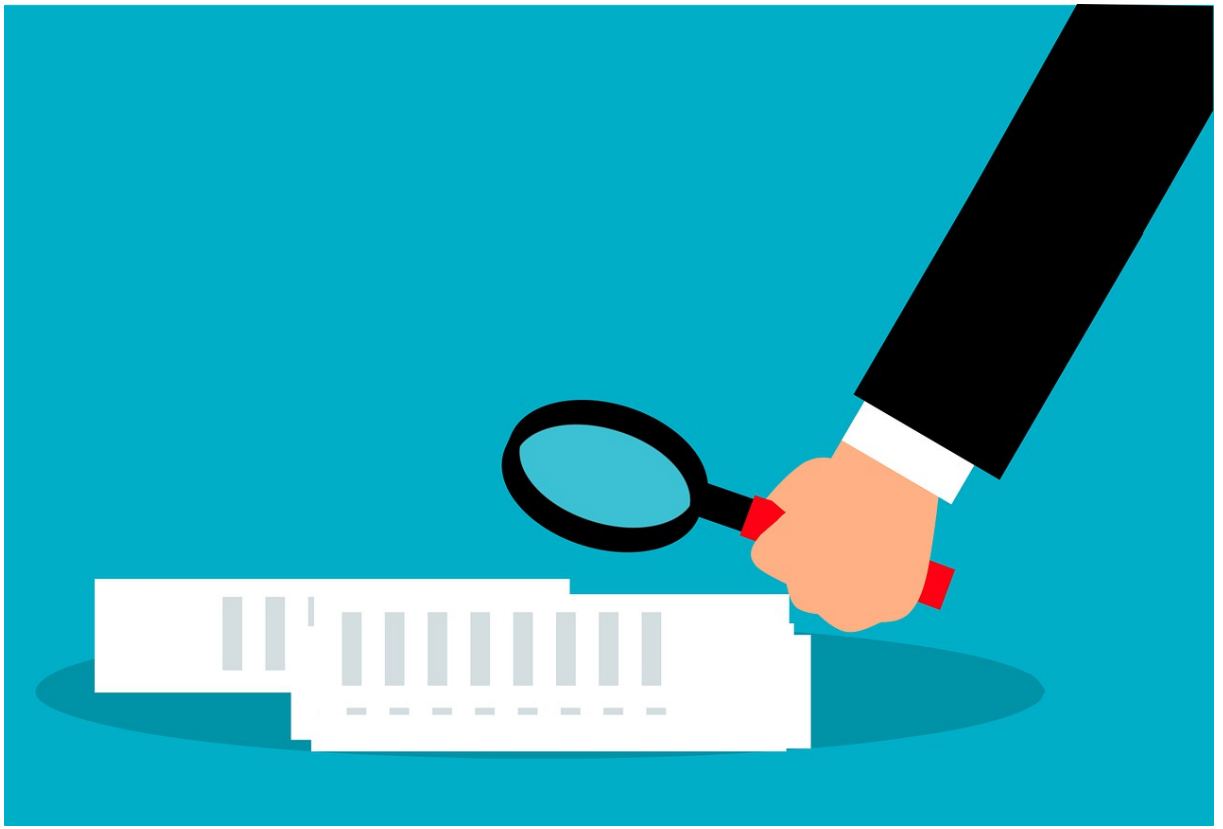
on person.license_id = drivers_license.id

Part 6: Identifying the Mastermind



on person.id = facebook_event_checkin.person_id

Summary

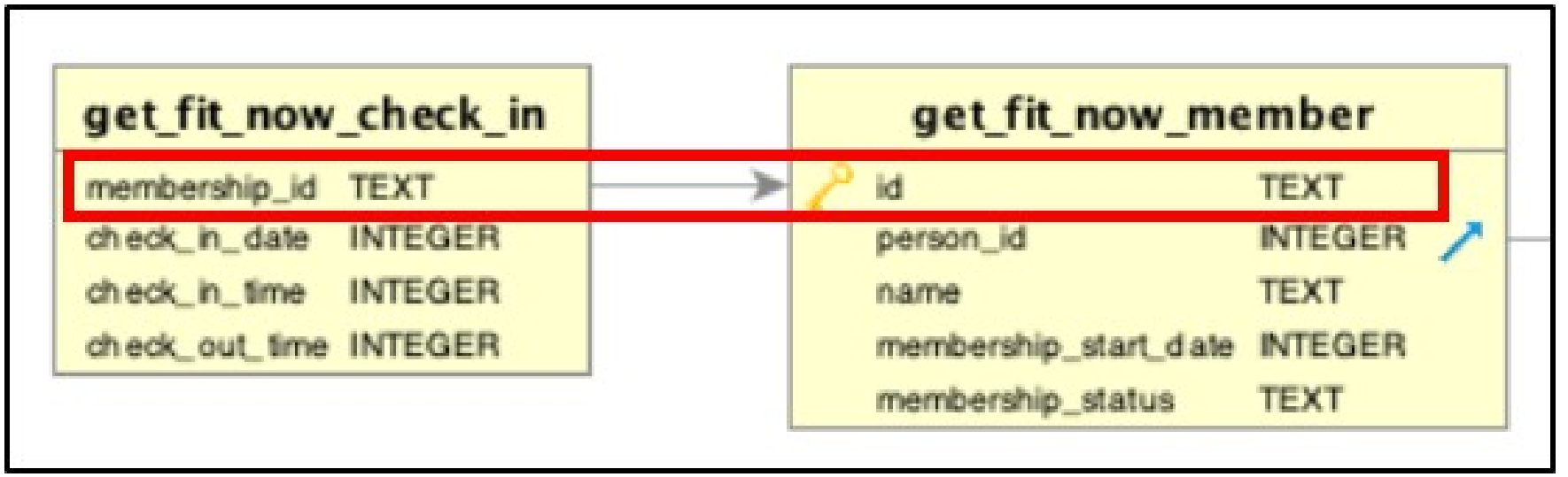


Let's review the SQL concepts we learned in this workshop:

Structured Query Language (SQL)

Structured Query Language (SQL) is a programming language used for managing data held in relational databases.

Relational Databases



Relational databases are databases where data in one table in the database can be connected to data in another table in the database.

The SELECT Command

```
1 SELECT count(*)
2 FROM person;
```

The **SELECT** command is common to all SQL queries looking to return specific info from the database.

The WHERE Filter

```
1 SELECT * FROM person WHERE name = 'Kinsey Erickson'
```

id	name	license_id	address_number	address_street_name	ssn
89906	Kinsey Erickson	510019	309	Northwestern Dr	635287661

And the **WHERE** filter allows very specific information to be returned by a query.

JOIN Queries

```sql
SELECT person.name, income.annual_income
FROM income
JOIN person
  ON income.ssn = person.ssn
WHERE annual_income > 450000
```

In order for SQL queries to return information from more than one table, the JOIN keyword must be used to connect all of the tables involved.

What's Next?

In the next HackerFrogs Afterschool web exploitation workshop, we'll learn about SQL injection vulnerabilities with the TryHackMe platform.

Extra Credit

Looking for more study material on this workshop's topics?

See this video's description for links to supplemental documents and exercises!

Until Next Time, HackerFrogs!