

# T3A3 - Implement a System with Data and Application Layers.

---

R2) Produces a professional report that provides an analysis of privacy and security concerns relating to a system.

## **Privacy and security concerns:**

Security is protecting data where as privacy is protecting the users identity. People in cybersecurity often follow the CIA triad model which assist cyber security specialist's to think about the different parts of information security. The CIA model has three parts. Confidentiality, Integrity and Availability.

## **Confidentiality:**

Enforcing access levels through Authorization and Authentication. This may involve sorting information into certain access areas for different Authorisation levels.

In this application there were more then one layer of protection to protect the privacy and confidentiality of its users. These methods were:

- login: Username and password
- Bcrypt: Encrypting the passwords so there not visible in clear site and protecting against brute force hacks.
- Tokens: JWT tokens to authenticate the users and Authorise access.

The user in this application will need to sign up before logging into the system. The personal details will be saved on a postgresql database where the root user postgres and authorised data administrator Scott can access the database. They have personal passwords that are greater then 16 characters to prevent brute force attacks.

When the users create there password when signing up the password will be displayed as a hash generated by bcrypt technology which is the main technology to protect against brute force attacks. Bcrypt has built in algorithms that significantly slows the speed of which a brute force hack can be performed therefore making brute force attacks fairly obsolete. The bcrypt technology has a parameter named 'salt' which means that as technology grows in power the salt parameter can adjust to compensate against the new power. Since the passwords are hashed if there was a hack through brute force or sql injection which resulted in personal data accessed, personal passwords are still safer therefore access to other information systems by the hacker will be more difficult. Even Authorised personnel could not read clear passwords. The main reason to hash passwords is because people like to use the same password for everything. Example below:

```
t3a3=# Select id,first_name,password from public.user;
```

id	first_name	password
1	Angela	\$2b\$12\$n9P4D0vdiDI6dknz82ClH.bZ2uhybiwhwkpkFFQq3oJYnxnUbWDJ2
2	David	\$2b\$12\$LF5o2M8yG5qa79pHI6ltq.sPfYjTTSt59AsDZABb1tI.upqeZ2wem
3	Melanie	\$2b\$12\$pgvXtzA61JCVV0PJYSdG9OXfRU9ctJQGPMQinGqhKp0diAtQZVQeq
4	Steven	\$2b\$12\$/XDTGzmMvJJtYIeTBe/ag.h2V40awUKYm3M3aDr1NTX1ca16vJp7C
5	Jason	\$2b\$12\$c0bubLacN7uhop0l34wXwe/JkBpEOzdYlpTNR6lTeY91LSMrYutxe
6	Rebecca	\$2b\$12\$MVS/D/NprlZMvF1m7DnSPuAdsG2lpU470gqjOwdnIHKIO4fPsuHoa
7	Richard	\$2b\$12\$KPRBWLyu8339Z.70ivi5KeRk.wPGaB.pCugei5fQcJNqgKNLqqbzq
8	Robert	\$2b\$12\$z8jbq9xiFVuj/iz74syLluMVpXTBXGD4citU6tpLh3gZy0HdoQRLq
9	Patricia	\$2b\$12\$FYytQsZ6wzazQ.VdX/VTBuvE2v7bJrs.XaTfNtmBx2jAJZbe1cMrC
10	Janet	\$2b\$12\$m8w.uFy.GMKb40gy/4cX9u1QwC76tGL/QgGVqAdAMLwqI.5zDH8fK

(10 rows)

### Integrity:

Integrity is protecting data from deletion or modification from potential hackers. Furthermore if someone authorised makes a change it can be reversed. Integrity of a program is to protect it against hackers that may use SQL injection or other methods to access data. Data needs to be correct, authentic and reliable.

In this application the program can be reversed if an error has been made by someone Authorised by using version control. This program use's git where verions of the program are saved regularly after achieving milestones. Furthermore the program is push to a third party called github. These versions can be accessed online or locally.

Flask has it owns built-in security protocols. The first security protocol is declaring the type of HTTP request in the app.routes. This will restrict the type of data that a user can input into the system. For example a GET request will only allow a user to see data and not send data as set in methods. Restricting data access minimises the vulnerabilities of a buffer overflow or sql injection. Flask also has it own functions that query data. Using raw sql is another vulnerability that can be used for SQL-injections. It is advised that queries be made with functions using the ORM model approach rather the raw sql in a string statement.

Another method of protecting data integrity is through tokens and cookies. In our program JWT tokens are given to users when logging in. The current setting to the token is 1 day. What that means is once the user authenticates who they are through username and password they will have authorisation to navigate through the website for one day until they need to log in again. Otherwise the user would need to log in every time a new page is open. In the application a user can create a personal playlist. Anyone can access there playlist and view the tracks but only the user who created the playlist can add tracks or delete tracks. So it was required to only Authorise the user who created the playlist to make changes to the playlist. This is where one method of using the JWT token can be used to implement Authorisation. The program matches the token with a user in the database given access to the playlist. Then changes can be made. This restricted unauthorised changes to personal playlists.

```
@artist.route("/<int:id>/Tracks",methods=["GET"])
def All_tracks(id):
```

An add on feature to flask is Marshmallow. Marshmallow is used for serialisation and deserialisation primarily but considering that marshmallow handles the input and output of data it has an extra feature. That is to validate

input from the user. Marshmallow can validate the type of data as particular to an email or date or simply a string or an integer. In this application we used all four data type validations and used them in combination with restricting the length of characters or range of integers. Marshmallow has its own invalid responses if the data is not correct. Validating is checking the accuracy and quality of source data. Also limiting the data will resist buffer overflows.

```
class UserSchema(ma.SQLAlchemyAutoSchema):
    class Meta:
        model = User
        # load_only = ["Password"] # wont output the password to the api.

        # Validation by specifying datatypes.
        email = ma.Email(required=True)
        first_name = ma.String(required=False, validate=Length(1,50))
        Surname = ma.String(required=False, validate=Length(1,50))
        Password = ma.String(required=True, validate=Length(1,200))
        Age = ma.Integer(required=False, validate=Range(0,150))
        Address = ma.String(required=False, validate=Length(1,100))
        City = ma.String(required=False, validate=Length(1,50))
        State = ma.String(required=False, validate=Length(1,50))
        Country = ma.String(required=False, validate=Length(1,50))
        Postcode = ma.Integer(required=False, validate=Range(1000,9999))
```

#### Availability:

Availability talks about the need to ensure data is accessible at all times. This may be related to the hardware availability. What can be done if there is a power failure, hardware failures and respond to upgrades. This program is only currently available on a local system. In deployment it would be recommended to deploy the application on the cloud that has multiple sources of hardware and backups.

R3) Produces a professional report which discusses professional, ethical and legal obligations relating to a system.

Users don't normally consider that they need to trust companies with their personal information. They assume that their data is safe. In fact the opposite is happening and companies are constantly monitoring user activities so they can sell their data or select advertising that the users might be interested in without the user even knowing. As an admin, where or when is it ethically ok to stop collecting someone's data. Where do they draw the line? So some ethical issues of the internet are privacy, confidentiality and anonymity. Not all methods of collecting data have the intentions to make profit. In some cases collecting data may be essential for solving major health issues. For example tracking people who have spread covid 19. Other reason may be an organisation wants to improve its product to the client. Is it ethically ok to collect the data from the user to find out which demographic they should focus on? A professional obligation would be to notify the users that their data will be collected if they choose to proceed. That means they need to consent to their data being collected. Some other ethical considerations may be whether a company should monitor emails and outgoing signals to protect intellectual property. There seems to be a grey area with ethics where what may seem ok for one company may not be ok for another company. An article from the Australian Privacy foundation

suggest that a user only give details relevant to the request and fill the gaps in with 'unnecessary' if they feel uncomfortable.

The Australian government has a Privacy act 1988 which was introduced to promote and protect the privacy of individuals.

Policies summarised:

- APP 1: Organisations need to be transparent and expressed with a privacy policy
- APP 2: Users need to have the option not to identify themselves or using a pseudonym
- APP 3: Organisations need to outline when information is solicited
- APP 4: Outlines how organisations must deal with unsolicited information
- APP 5: Outlines circumstances in which an organisation needs to tell people about certain situations
- APP 6: Outlines circumstances in which an organisation may use or disclose information
- APP 7: Certain conditions need to be met before used for marketing
- APP 8: Outlines steps to protect information before disclosing overseas
- APP 9: Government related circumstances.
- APP 10: The organisation has to ensure the data they collect and disclose is accurate
- APP 11: An organisation has an obligation to protect personal information and to destroy or de-identify if necessary.
- APP 12: Individuals need to have the ability to gain access to their information
- APP 13: An organisation has an obligation to change or update personal information

So to implement the requirements of the privacy act into this application I will need the user upon registration to read and acknowledge a privacy policy that will have in the policy how I will collect and use their data. Disclose any information of their data to where it will be sent. Outline steps to protect their personal information and to destroy or de-identify the users data if there was a breach of any of the security protocols. That would be a quick overview to meet some or all the legal obligations. I would think to get professional legal advice before implementing the privacy policy. There are many different laws, some new and relevant to social media but it is definitely out of the scope of this app.

The privacy act is only for large business's that bring in \$3 million or more per year. In saying that most small business that earn less than that still opt into the privacy act to protect their business's as a matter of best practices. Benefits to the company could be improved consumer confidence and trust in their operations.

**Reference:**

OAIC. (2019). Australian Privacy Principles quick reference. [online] Available at: <https://www.oaic.gov.au/privacy/australian-privacy-principles/australian-privacy-principles-quick-reference/>.

**R4) Uses programming language features or frameworks to implement a data model.**

Please take note that a MVC data model was implemented where the model talks to postgresql database, the view renders templates covering the front end of the app and the controller combines and directs the data to where it needs to go. SQLAlchemy was used to communicate with the database. No raw sql was used in the commands to avoid sql injection. The add on feature Marshmallow was used to serialize/deserialize data and to validate data.

Example of marshmallow validating the user data. Also navigate to schemas:



```

from main import ma
from models.User_table import User
from marshmallow import fields
from marshmallow.validate import Length, Range

class UserSchema(ma.SQLAlchemyAutoSchema):
    class Meta:
        model = User

    # Validation by specifying datatypes.
    email = ma.Email(required=True)
    first_name = ma.String(required=False, validate=Length(1,50))
    Surname = ma.String(required=False, validate=Length(1,50))
    Password = ma.String(required=True, validate=Length(1,200))
    Age = ma.Integer(required=False, validate=Range(0,150))
    Address = ma.String(required=False, validate=Length(1,100))
    City = ma.String(required=False, validate=Length(1,50))
    State = ma.String(required=False, validate=Length(1,50))
    Country = ma.String(required=False, validate=Length(1,50))
    Postcode = ma.Integer(required=False, validate=Range(1000,9999))

user_schema = UserSchema()
users_schema = UserSchema(many=True)

```

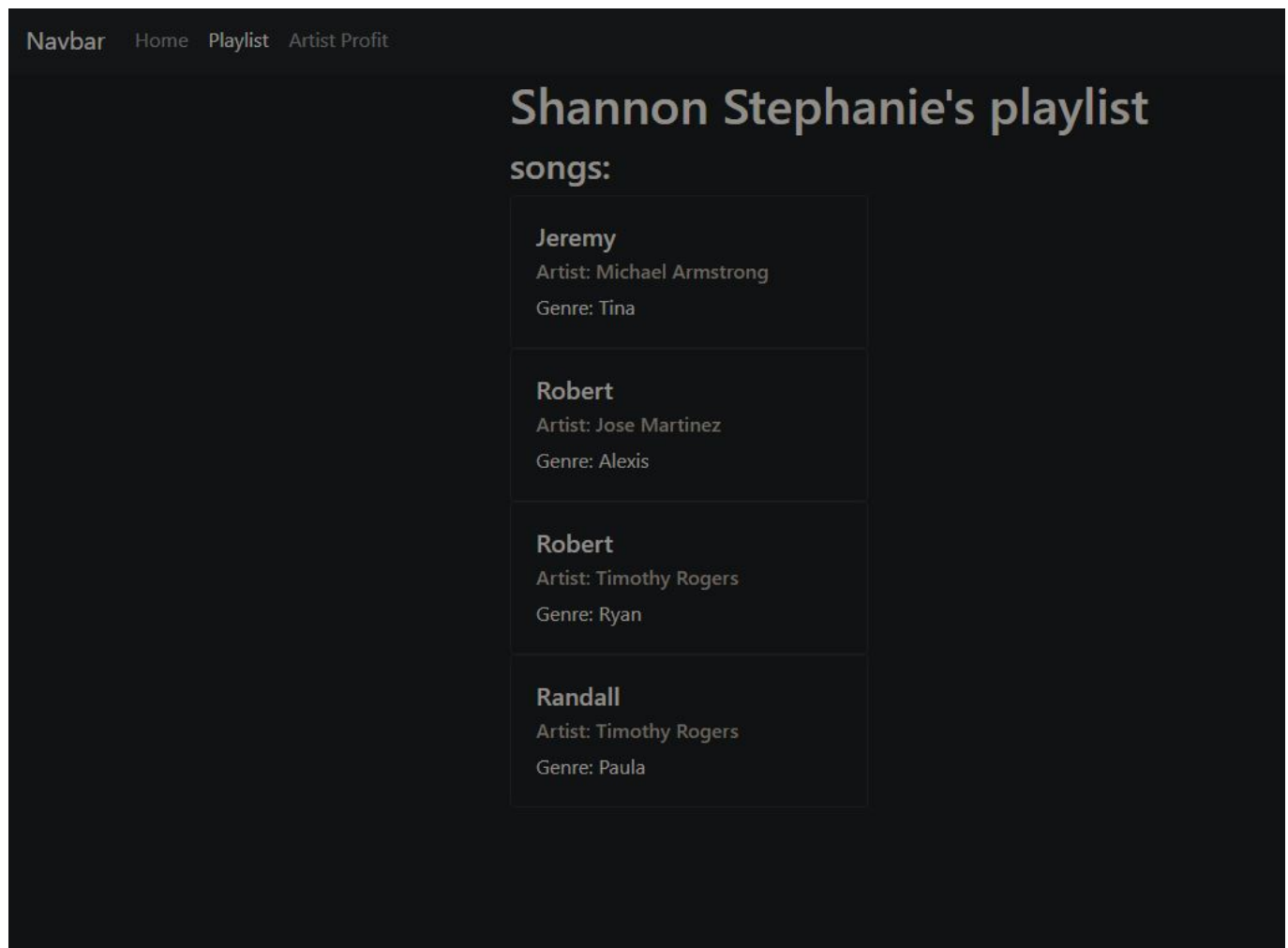
R5) Implement code using a Python framework to create, read, update and delete records, and export all data from the database from the database.

Check controllers/User\_controller for a full crud resource. Check controllers/Playlist\_controller to check that has a create, read and delete feature. Check controllers/Album\_controller for create, read and update.

R6) Uses programming language features or frameworks to display data

I have designed a template for the playlist display at controllers/Playlist\_controllers for the endpoint localhost:5000\playlist\id. Also check templates/layout.html and templates/playlist.html. I am using fake data that inputs first names for the data so try not to get confused.

Evidence:



R7) Implements application layer which utilises a database to produce aggregated data relating to business matters

Check controllers/Artist\_controller.

Endpoints:

Start with localhost:5000/artist

Endpoints	Output
/Highest_profit	Finds the highest paid artist using the MAX function
/Lowest_profit	Finds the lowest paid artist using the MIN function
/Average_profit	Finds the average profit of all artists using the average function
/Sum_profit	Adds all the profit of all the artists using the sum function

R8) Implements input validation and integrity checks on data to address business risks

Marshmallow handles the serialization and deserialization of the data meaning it is the first feature to work with the input data from the user making it ideal to validate the data input. Developers have considered this already and developed a library of validation tools built into marshmallow which I have used to validate the input. For example I have defined that email = ma.Email(required = True). That means that email needs to be formatted like an email, it also means it does not except null values as given in the insomnia example and

the field is required so the user can not leave it blank when submitting that endpoint. Marshmallow even has its own invalid messages seen in the example below.

address: schemas/User\_schema

```
from main import ma
from models.User_table import User
from marshmallow import fields
from marshmallow.validate import Length, Range

class UserSchema(ma.SQLAlchemyAutoSchema):
    class Meta:
        model = User

    # Validation by specifying datatypes.
    email = ma.Email(required=True)
    first_name = ma.String(required=False, validate=Length(1,50))
    Surname = ma.String(required=False, validate=Length(1,50))
    Password = ma.String(required=True, validate=Length(1,200))
    Age = ma.Integer(required=False, validate=Range(0,150))
    Address = ma.String(required=False, validate=Length(1,100))
    City = ma.String(required=False, validate=Length(1,50))
    State = ma.String(required=False, validate=Length(1,50))
    Country = ma.String(required=False, validate=Length(1,50))
    Postcode = ma.Integer(required=False, validate=Range(1000,9999))

user_schema = UserSchema()
users_schema = UserSchema(many=True)
```

Insomnia example:

POST localhost:5000/user/create Send 400 BAD REQUEST 207 ms 5

JSON Auth Query Header 1 Docs Preview Header 4 Cookie

```

1 {
2   "Address": "37392 Pratt Prairie\nWest
   Marymouth, WI 41553",
3   "Age": 54,
4   "City": "Schwartztown",
5   "Country": "Sudan",
6   "password": "easy",
7   "Postcode": 2121,
8   "State": "Michigan",
9   "Surname": "Amy",
10  "email": "",
11  "first_name": "Michael",
12  "id": 4
13 }

```

```

1 {
2   "email": [
3     "Not a valid email address."
4   ]
5 }

```

R9) Analyses a problem scenario and creates database tables and fields

#### One to One example:

Look in models/User\_table.py and models/Alias\_table.py.

I created a one to one table between User\_tables and Alias\_table purely to meet the requirements of the assessment. I create the link in flask shell in the example below.

```

(venv) scott@LAPTOP-FHQGCOGQ:models$ flask shell
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
App: main [development]
Instance: /mnt/c/Users/scott/Desktop/CoderAcademy/ScottMalone_T3A3/instance
>>> from main import db
>>> from models.Alias_table import Alias
>>> alias = Alias(alias_name = "lightning", user_id = "1")
>>> db.session.add(alias)
>>> db.session.commit()
>>> alias = Alias(alias_name = "fire", user_id = "2")
>>> db.session.add(alias)
>>> db.session.commit()

```

```

>>> user_1.alias.alias_name
'lightning'
>>> user_1.first_name
'Sharon'
>>> alias_1.user_alias.first_name
'Sharon'
>>> alias_1.alias_name
'lightning'
>>> 

```

#### One to Many example:



Look in models/Artist\_table, models/Albums\_table, models/Tracks\_table

```
class Artist(db.Model):
    __tablename__ = "artist"

    Artist_id = db.Column(db.Integer, primary_key=True)
    Artist_name = db.Column(db.String(100))
    Country = db.Column(db.String(50))
    gross_worth = db.Column(db.Integer)
    albums = db.relationship(Album, backref='artist_album', cascade="all, delete")
    tracks = db.relationship(Tracks, backref='artist_track', cascade="all, delete")

3 class Tracks(db.Model):
4     __tablename__ = "tracks"
5
6     tracks_id = db.Column(db.Integer, primary_key=True)
7     tracks_name = db.Column(db.String(100))
8     date_released = db.Column(db.DateTime)
9     genre = db.Column(db.String(50))
10    Artist_id = db.Column(db.Integer, db.ForeignKey('artist.Artist_id'), nullable=False)
11    album_id = db.Column(db.Integer, db.ForeignKey('album.album_id'), nullable=False)
12
```

I have designed a database of artist, albums and tracks. In my design I have made a One artist to Many albums relationship, a One artist to Many tracks and a One album to Many tracks relationship.

Below is the controller endpoints for models/Artist\_controller. In the functions the queries call the artist with the matching id then display all albums or all track relating to that artist.

```
# One to many example of Artist to Albums. Artist is the parent and I was able
# to find the albums that the artists has create through the relationship
@artist.route("/<int:id>/Albums", methods=["GET"])
def All_albums(id):
    artist = Artist.query.filter_by(Artist_id=id).first()
    album_list = []
    for item in artist.albums:
        album_list.append(item.album_name)
    if not album_list:
        return f"<h1>Unfortunately {artist.Artist_name} has not deposited any albums yet.</h1>"
    artist_albums = {"Album names": album_list}
    return artist_albums

# One to many example of Artist to tracks. Artist is the parent and I was able
# to find the tracks that the artists has created through the relationship

@artist.route("/<int:id>/Tracks", methods=["GET"])
def All_tracks(id):
    artist = Artist.query.filter_by(Artist_id=id).first()
    tracks_list = []
    for item in artist.tracks:
        tracks_list.append(item.tracks_name)
    if not tracks_list:
        return f"<h1>Unfortunately {artist.Artist_name} has not deposited any tracks yet.</h1>"
    artist_tracks = {"Tracks names": tracks_list}
    return artist_tracks
```

Results of the controller endpoints displaying all albums related to artist with id=1 and all tracks to artist with id=1. Take note all Output I have just used fake names.

GET localhost:5000/artist/1/Albums Send

200 OK348 ms57 B

BodyAuthQueryHeaderDo

PreviewHeader 4

```
1 {
2   "Album names": [
3     "David",
4     "Christina"
5   ]
6 }
```

GET localhost:5000/artist/1/Tracks Send

200 OK212 ms53 B

BodyAuthQueryHeaderDo

PreviewHeader 4

```
1 {
2   "Tracks names": [
3     "Kiara",
4     "Dean"
5   ]
6 }
```

Many to Many relationship example: Bonus marks

address: models/playlist

I needed five tables to meet the requirement so I added a playlist table that is the bridge between Users to Tracks. Many Users can have Many tracks. Endpoints:

Begin with:

localhost:5000/playlist

Endpoints	Purpose
/id	Get the playlist of a user with correct id
/add_track	Add a track to the users playlist
/delete_track	Delete a track from the users playlist

```
t3a3=# select * from playlist;
 id | tracks_id
-----+-----
  9 |         8
  1 |         2
 10 |         8
  5 |         7
  6 |         5
  2 |         9
  8 |         6
  4 |         2
  4 |         7
  3 |         6
  4 |         4
  4 |         1
(12 rows)

t3a3=#
```

GET ▾ localhost:5000/playlist/4 Send 200 OK 217 ms 86 B

Body ▾ Auth ▾ Query Header Do

Preview ▾ Header 4 Co

```
1 {
2   "Rachels playlist": [
3     "Kiara",
4     "Megan",
5     "Alan",
6     "Alison"
7   ]
8 }
```

R10) Develops complex queries which select, filter, group and order data.

all query's can be found in /controllers/Artist\_controller clearly labelled 1/3, 2/3 or 3/3.

query	type
artist_name = Artist.query.filter_by(gross_worth=max_pay).first()	selection filter
ordering = Artist.query.order_by(Artist.gross_worth.desc()).all()	ordering descending order
filtering = Artist.query.filter((Artist.gross_worth < 40000000)).all()	filtering by gross_worth magnitude

R11) Develops complex queries which join tables together.

Below is a picture of raw sql in the postgresql terminal. I joined three tables together. I match the playlist table with the name of the user and the name of the track they displayed.

```
t3a3=#
SELECT first_name, public.user.id, tracks.tracks_id, tracks_name
FROM public.user
INNER JOIN public.playlist ON
public.user.id = public.playlist.id
INNER JOIN public.tracks ON
public.tracks.tracks_id = public.playlist.tracks_id;
 first_name | id | tracks_id | tracks_name
-----+-----+-----+-----
Anita       |  9 |         8 | Belinda
Sharon      |  1 |         2 | Megan
Nicholas    | 10 |         8 | Belinda
John        |  5 |         7 | Alison
Daniel      |  6 |         5 | Leslie
Tim         |  2 |         9 | William
Brooke      |  8 |         6 | Katelyn
Rachel      |  4 |         2 | Megan
Rachel      |  4 |         7 | Alison
Wanda       |  3 |         6 | Katelyn
Rachel      |  4 |         4 | Alan
Rachel      |  4 |         1 | Kiara
(12 rows)
```

Below is another raw sql query where I joined another three tables together. Currently I have joined up the artist table with the albums table and the tracks table. Both the albums and the tracks belong to the artist. Please take into consideration that I just seeded all the names with first\_name.faker().

```
t3a3=# SELECT artist_name, album_name, tracks_name
FROM public.artist
INNER JOIN public.album ON
public.artist.artist_id = public.album.artist_id
INNER JOIN public.tracks ON
public.artist.artist_id = public.tracks.artist_id;
 artist_name | album_name | tracks_name
-----+-----+-----
Jose Martinez | Crystal   | Jennifer
Jose Martinez | Crystal   | Robert
April Ruiz   | Richard   | Jason
April Ruiz   | Bradley   | Jason
Timothy Rogers | Michael   | Dylan
Timothy Rogers | Michael   | Randall
Timothy Rogers | Michael   | Robert
Timothy Rogers | Michael   | Amy
Michael Armstrong | Todd     | Neil
Michael Armstrong | Todd     | Jeremy
Evan Johnson  | James     | Jason
April Ruiz    | Erin      | Jason
(12 rows)
```



Below is an example of joining artists and tracks using SQLAlchemy. The formatting is not as nice but it hopefully fulfills any requirements. Completed in the flask shell.

```
>>> from main import db
>>> from models.Artist_table import Artist
>>> from models.Albums_table import Album
>>> from models.Tracks_table import Tracks
>>> results = db.session.query(Artist,Tracks).join(Tracks).all()
>>> for artist,tracks in results:
...     print(artist.artist_name,",",tracks.tracks_name)
...
Michael Armstrong , Jeremy
April Ruiz , Jason
Michael Armstrong , Neil
Timothy Rogers , Amy
Jose Martinez , Robert
Timothy Rogers , Robert
Evan Johnson , Jason
Timothy Rogers , Randall
Jose Martinez , Jennifer
Timothy Rogers , Dylan
>>> █
```

R12) Implements a script to export all data from the database.

Script: check dump.sql in directory SCOTTMALONE\_T3A3 to find the evidence;

pg\_dump --no-owner t3a3 > dump.sql ( Save everything on a database to a sql file for transporting )  
psql {new database} < dump.sql ( Load into new database )