Create a connection

```
import sqlite3
conn = sqlite3.connect('rpg_db.sqlite3')
```

Select statement

```
curs = conn.cursor()
query = 'SELECT * FROM armory_item;'
curs.execute(query)
results = curs.fetchall()
```

rpg_example.py

```
import sqlite3
def connect_to_db(db_name='rpg_db.sqlite3'):
    return sqlite3.connect(db_name)
def execute_query(cursor, query):
    cursor.execute(query)
    return cursor.fetchall()
GET_CHARACTERS = """
  SELECT *
  FROM charactercreator_character;
"""
if __name__ == '__main__':
    conn = connect_to_db()
    curs = conn.cursor()
    results = execute_query(curs, GET_CHARACTERS)
    print(results)
```

Query log:

```
SELECT * FROM charactercreator_character;
SELECT COUNT(*) FROM charactercreator_character;
SELECT COUNT(DISTINCT name) FROM charactercreator_character;
```

Selecting columns:

```
SELECT character_id, name FROM charactercreator_character;
```

Limiting rows:

```
SELECT character_id, name FROM charactercreator_character LIMIT 10;
```

Filtering rows with condition:

```
SELECT character_id, name
FROM charactercreator_character
WHERE character_id > 50;
```

```
SELECT character_id, name
FROM charactercreator_character
WHERE character_id > 50
AND character_id < 55;
```

Equivalent:

```
SELECT character_id, name
FROM charactercreator_character
WHERE character_id BETWEEN 51 AND 54;
General theme - often more than one way to do it!
```

Let's figure out what the duplicate character names are

```
SELECT name, COUNT(*)
FROM charactercreator_character
GROUP BY name;
```

Our first business query!

Let's figure out what the duplicate character names are

```
SELECT name, COUNT(*)
FROM charactercreator_character
GROUP BY 1
ORDER BY 2 DESC
LIMIT 5;
```

SELECT - how we choose which columns to get
WHERE - how we set conditions on the rows to be returned
LIMIT - when we only want a certain number of rows
ORDER - when we want to sort the output
JOIN - when we need data from multiple tables combined

A first join:

```
SELECT * FROM charactercreator_character
INNER JOIN charactercreator_fighter
ON character_id = character_ptr_id
WHERE character_id = 1;
```

Non-inner joins introduce missing values!

```
SELECT character_id, name, rage FROM charactercreator_character
LEFT JOIN charactercreator_fighter
ON character_id = character_ptr_id;
```

Explicit inner join:

```
SELECT character_id, name, rage FROM charactercreator_character
INNER JOIN charactercreator_fighter
ON character_id = character_ptr_id
WHERE character_id BETWEEN 40 and 50;
```

Equivalent implicit join:

```
SELECT character_id, name, rage
FROM charactercreator_character, charactercreator_fighter
WHERE character_id = character_ptr_id
AND character_id BETWEEN 40 and 50;
```

Queries result in tables that can be queried! (Silly example but can be useful)

```
SELECT * FROM
(SELECT * FROM charactercreator_character);
```

Sometimes you need to join >2 tables...

This is where I particularly like implicit joins

```
SELECT cc.character_id, cc.name, ai.item_id, ai.name
FROM charactercreator_character AS cc,
armory_item AS ai,
charactercreator_character_inventory AS cci
WHERE cc.character_id = cci.character_id
AND ai.item_id = cci.item_id;
```

Use a subquery to make a temp table to query from

```
SELECT character_id, COUNT(DISTINCT item_id) FROM
(SELECT cc.character_id, cc.name AS character_name, ai.item_id, ai.name AS
item_name
FROM charactercreator_character AS cc,
armory_item AS ai,
charactercreator_character_inventory AS cci
WHERE cc.character_id = cci.character_id
AND ai.item_id = cci.item_id)
GROUP BY 1 ORDER BY 2 DESC;
```