Part 4: SQL Databases and Pandas

- improving memory and performance efficiency

SQLite: PYTHON DOCS

- Databases contain many tables
- Relational databases are composed of linked tables, usually linked by a field
- Using SQL queries in Python and Pandas
- Using both together = storage can be more complex
- Long-term storage / large data = databases
- Only load data into Python and Pandas that is needed for immediate analysis
- SQLite - free, serverless, self-contained, requires no configuration, no installation required, also the most widely used database engine

---

1. Creating an SQLite Database

- Import sqlite3 (as sq3) and create a new SQLite Database with the name "movies.db".

```
import sqlite3 as sq3
import json
import pandas as pd
```

sq3.connect()

- You can connect to an existing database
- If the database you pass does not exist, it will create it
- This also creates a connection to the database

```
conn = sq3.connect("movies.db")
```

```
type(conn)
```

```
sqlite3.Connection
```

sq3.execute()

- used to perform a query or some other action in SQL

sq3.fetchall()

- returns all data as per the query

```
conn.execute("Select * FROM sqlite_master").fetchall()
```

```
[('table',
  'Movies',
  'Movies',
  2,
  'CREATE TABLE "Movies" (\n"id" INTEGER,\n  "title" TEXT,\n  "revenue" REAL,\n
"budget" REAL,\n  "belongs_to_collection_name" TEXT,\n  "release_date" TIMESTAMP\n)'),
 ('table',
  'Votes',
  'Votes',
  3,
  'CREATE TABLE "Votes" (\n"id" INTEGER,\n  "vote_count" INTEGER,\n  "vote_average"
REAL\n)'),
 ('table',
  'Genres',
  'Genres',
  4,
  'CREATE TABLE "Genres" (\n"genre_id" INTEGER,\n  "genre_name" TEXT,\n  "id"
INTEGER\n)'),
 ('table',
  'Production',
  'Production',
  5,
  'CREATE TABLE "Production" (\n"prod_id" INTEGER,\n  "prod_logo_path" TEXT,\n
"prod_name" TEXT,\n  "prod_origin_country" TEXT,\n  "id" INTEGER\n)')]
```

sq3.close()

- closes the connection to the database

---

2. Loading Data from DataFrames into an SQLite Database

- **Load** the json file "**some_movies.json**" and **split** the dataset into the following **four datasets** (save each dataset in a Pandas DataFrame).

```
with open('some_movies.json') as file:
    data = json.load(file)
```

```
data[0]
```

```
{'adult': False,
 'backdrop_path': '/orjiB3oUIsyz60hoEqkiGpy5CeO.jpg',
 'belongs_to_collection': {'id': 86311,
```

```
  'name': 'The Avengers Collection',
  'poster_path': '/yFSIUVTCvgYrpalUktulvk3Gi5Y.jpg',
  'backdrop_path': '/zuW6fOiusv4X9nnW3paHGfXcSll.jpg'},
 'budget': 356000000,
 'genres': [{'id': 12, 'name': 'Adventure'},
  {'id': 878, 'name': 'Science Fiction'},
  {'id': 28, 'name': 'Action'}],
 'homepage': 'https://www.marvel.com/movies/avengers-endgame',
 'id': 299534,
 'imdb_id': 'tt4154796',
 'original_language': 'en',
 'original_title': 'Avengers: Endgame',
 'overview': "After the devastating events of Avengers: Infinity War, the universe is
in ruins due to the efforts of the Mad Titan, Thanos. With the help of remaining
allies, the Avengers must assemble once more in order to undo Thanos' actions and
restore order to the universe once and for all, no matter what consequences may be in
store.",
 'popularity': 50.279,
 'poster_path': '/or06FN3Dka5tukK1e9sl16pB3iy.jpg',
 'production_companies': [{'id': 420,
   'logo_path': '/hUzeosd33nzE5MCNsZxCGEKTXaQ.png',
   'name': 'Marvel Studios',
   'origin_country': 'US'}],
 'production_countries': [{'iso_3166_1': 'US',
   'name': 'United States of America'}],
 'release_date': '2019-04-24',
 'revenue': 2797800564,
 'runtime': 181,
 'spoken_languages': [{'iso_639_1': 'en', 'name': 'English'},
  {'iso_639_1': 'ja', 'name': '□□□'},
  {'iso_639_1': 'xh', 'name': ''}],
 'status': 'Released',
 'tagline': 'Part of the journey is the end.',
 'title': 'Avengers: Endgame',
 'video': False,
 'vote_average': 8.3,
 'vote_count': 12607}
```


pd.json_normalize()

pandas.json_normalize

pandas.json_normalize(*data, record_path=None, meta=None, meta_prefix=None, record_prefix=None, errors='raise', sep='.', max_level=None*) [source]

Normalize semi-structured JSON data into a flat table.

Parameters:

data : *dict or list of dicts*

Unserialized JSON objects.

record_path : *str or list of str, default None*

Path in each object to list of records. If not passed, data will be assumed to be an array of records.

meta : *list of paths (str or list of str), default None*

Fields to use as metadata for each record in resulting table.

meta_prefix : *str, default None*

If True, prefix records with dotted (?) path, e.g. foo.bar.field if meta is ['foo', 'bar'].

record_prefix : *str, default None*

If True, prefix records with dotted (?) path, e.g. foo.bar.field if path to records is ['foo', 'bar'].

errors : *{'raise', 'ignore'}, default 'raise'*

Configures error handling.

- 'ignore' : will ignore KeyError if keys listed in meta are not always present.
- 'raise' : will raise KeyError if keys listed in meta are not always present.

sep : *str, default '.'*

Nested records will generate names separated by sep. e.g., for sep='.', {'foo': {'bar': 0}} -> foo.bar.

max_level : *int, default None*

Max number of levels(depth of dict) to normalize. if None, normalizes all levels.

```
df = pd.json_normalize(data, sep = "_")
```

```
df.head(2)
```

	adult	backdrop_path	budget	genres	homepage	
0	False	/orjiB3oUlsyz60hoEqkiGpy5CeO.jpg	356000000	[{'id': 12, 'name': 'Adventure'}, {'id': 878, ...	https://www.marvel.com/movies/avengers-endgame	2
1	False	/wcC7kCICL6x6zHUlUyNp9pWoqW1.jpg	237000000	[{'id': 28, 'name': 'Action'}, {'id': 12, 'nam...	http://www.avatarmovie.com/	

2 rows × 29 columns

Dataset #1

- (Movies) with columns ["id", "title", "revenue", "budget", "belongs_to_collection_name", "release_date"].
- Convert "release_date" to datetime and transform "budget" and "revenue" to Million USD before loading into the Database.

```
movies = df[["id", "title", "revenue", "budget", "belongs_to_collection_name", "release
movies.head(3)
```

	id	title	revenue	budget	belongs_to_collection_name	release_date
0	299534	Avengers: Endgame	2797800564	356000000	The Avengers Collection	2019-04-24
1	19995	Avatar	2787965087	237000000	Avatar Collection	2009-12-10
2	140607	Star Wars: The Force Awakens	2068223624	245000000	Star Wars Collection	2015-12-15

```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
Data columns (total 6 columns):
 #   Column                      Non-Null Count  Dtype 
---  ------                      --------------  ----- 
 0   id                          18 non-null     int64 
 1   title                       18 non-null     object
 2   revenue                     18 non-null     int64 
 3   budget                      18 non-null     int64 
 4   belongs_to_collection_name  15 non-null     object
 5   release_date                18 non-null     object
dtypes: int64(3), object(3)
memory usage: 992.0+ bytes
```

To avoid warnings about changing values in columns:

- This is only necessary if you do not make a copy when creating the sub dataframes

To avoid warnings about changing values in columns:

```
# pd.set_option('mode.chained_assignment', None)
```

```
movies.release_date = pd.to_datetime(df.release_date)
```

```
type(movies.release_date[0])
```

```
pandas._libs.tslibs.timestamps.Timestamp
```

```
movies.revenue = df.revenue/1000000
movies.budget = df.budget/1000000
```

```
movies.head(1)
```

	id	title	revenue	budget	belongs_to_collection_name	release_date

	id	title	revenue	budget	belongs_to_collection_name	release_date
0	299534	Avengers: Endgame	2797.800564	356.0	The Avengers Collection	2019-04-24

Dataset #2

- (Votes) with columns ["id", "vote_count", "vote_average"].

```
votes = df[["id", "vote_count", "vote_average"]].copy()
```

```
votes.head(2)
```

	id	vote_count	vote_average
0	299534	12607	8.3
1	19995	21000	7.4

Dataset #3

- (Genres) with columns ["genre_id", "genre_name", "id"].

```
genres = pd.json_normalize(data = data, record_path = 'genres', meta = 'id', record_pre
```

```
genres.head(2)
```

	genre_id	genre_name	id
0	12	Adventure	299534
1	878	Science Fiction	299534

Dataset #4

- (Prod) with columns ["comp_id", "comp_logo_path", "comp_name", "comp_origin_country", "id"].

```
production = pd.json_normalize(data = data, record_path = "production_companies", meta
```

```
production.head(2)
```

	prod_id	prod_logo_path	prod_name	prod_origin_country	id
0	420	/hUzeosd33nzE5MCNsZxCGEKTXaQ.png	Marvel Studios	US	299534
1	444	/42UPdZl6B2cFXgNUASR8hSt9mpS.png	Dune Entertainment	US	19995

---

3. **Load** the individual dataframes **into the database**

- (each dataset should be a separate table in the database). **Name** the tables "Movies", "Votes", "Genres", "Prod".

```
conn = sq3.connect('movies.db')
conn
```

```
<sqlite3.Connection at 0x23f196c4e40>
```

pd.df.to_sql()

DataFrame.to_sql(*name, con, schema=None, if_exists='fail', index=True, index_label=None, chunksize=None, dtype=None, method=None*) [source]

Write records stored in a DataFrame to a SQL database.

Databases supported by SQLAlchemy [1] are supported. Tables can be newly created, appended to, or overwritten.

Parameters:

name : *str*

Name of SQL table.

con : *sqlalchemy.engine.(Engine or Connection) or sqlite3.Connection*

Using SQLAlchemy makes it possible to use any DB supported by that library. Legacy support is provided for sqlite3.Connection objects. The user is responsible for engine disposal and connection closure for the SQLAlchemy connectable See here.

schema : *str, optional*

Specify the schema (if database flavor supports this). If None, use default schema.

if_exists : *{'fail', 'replace', 'append'}, default 'fail'*

How to behave if the table already exists.

- fail: Raise a ValueError.
- replace: Drop the table before inserting new values.
- append: Insert new values to the existing table.

index : *bool, default True*

Write DataFrame index as a column. Uses *index_label* as the column name in the table.

index_label : *str or sequence, default None*

Column label for index column(s). If None is given (default) and *index* is True, then the index names are used. A sequence should be given if the DataFrame uses MultiIndex.

chunksize : *int, optional*

Specify the number of rows in each batch to be written at a time. By default, all rows will be written at once.

dtype : *dict or scalar, optional*

Specifying the datatype for columns. If a dictionary is used, the keys should be the column names and the values should be the SQLAlchemy types or strings for the sqlite3 legacy mode. If a scalar is provided, it will be applied to all columns.

method : *{None, 'multi', callable}, optional*

Controls the SQL insertion clause used:

- None : Uses standard SQL `INSERT` clause (one per row).
- 'multi': Pass multiple values in a single `INSERT` clause.
- callable with signature `(pd_table, conn, keys, data_iter)`.

Details and a sample callable implementation can be found in the section insert method.

Returns: **None or int**

Number of rows affected by to_sql. None is returned if the callable passed into `method` does not return an integer number of rows.

The number of returned rows affected is the sum of the `rowcount` attribute of `sqlite3.Cursor` or SQLAlchemy connectable which may not reflect the exact number of written rows as stipulated in the sqlite3 or SQLAlchemy.

```
# movies.to_sql("Movies", conn, index = False)
# votes.to_sql("Votes", conn, index = False)
# genres.to_sql("Genres", conn, index = False)
# production.to_sql("Production", conn, index = False)
```

```
conn.execute("Select * FROM sqlite_master").fetchall()
```

```
[('table',
  'Movies',
  'Movies',
  2,
  'CREATE TABLE "Movies" (\n"id" INTEGER,\n  "title" TEXT,\n  "revenue" REAL,\n
"budget" REAL,\n  "belongs_to_collection_name" TEXT,\n  "release_date" TIMESTAMP\n)'),
 ('table',
  'Votes',
  'Votes',
  3,
  'CREATE TABLE "Votes" (\n"id" INTEGER,\n  "vote_count" INTEGER,\n  "vote_average"
REAL\n)'),
 ('table',
  'Genres',
  'Genres',
  4,
  'CREATE TABLE "Genres" (\n"genre_id" INTEGER,\n  "genre_name" TEXT,\n  "id"
INTEGER\n)'),
 ('table',
  'Production',
  'Production',
```

```
  5,
  'CREATE TABLE "Production" (\n"prod_id" INTEGER,\n  "prod_logo_path" TEXT,\n
"prod_name" TEXT,\n  "prod_origin_country" TEXT,\n  "id" INTEGER\n)')]
```

```
conn.execute("SELECT name FROM sqlite_master WHERE type = 'table' ORDER BY name").fetch
```

```
[('Genres',), ('Movies',), ('Production',), ('Votes',)]
```

---

4. Loading Data from SQLite Databases into DataFrames

- **Load** the full tables "Movies", "Votes", "Genres", "Prod" from "movies.db" into Pandas (four DataFrames). **Set** "id" as Index.

```
conn.execute('SELECT * FROM sqlite_master').fetchall()
```

```
[('table',
  'Movies',
  'Movies',
  2,
  'CREATE TABLE "Movies" (\n"id" INTEGER,\n  "title" TEXT,\n  "revenue" REAL,\n
"budget" REAL,\n  "belongs_to_collection_name" TEXT,\n  "release_date" TIMESTAMP\n)'),
 ('table',
  'Votes',
  'Votes',
  3,
  'CREATE TABLE "Votes" (\n"id" INTEGER,\n  "vote_count" INTEGER,\n  "vote_average"
REAL\n)'),
 ('table',
  'Genres',
  'Genres',
  4,
  'CREATE TABLE "Genres" (\n"genre_id" INTEGER,\n  "genre_name" TEXT,\n  "id"
INTEGER\n)'),
 ('table',
  'Production',
  'Production',
  5,
  'CREATE TABLE "Production" (\n"prod_id" INTEGER,\n  "prod_logo_path" TEXT,\n
"prod_name" TEXT,\n  "prod_origin_country" TEXT,\n  "id" INTEGER\n)')]
```

pd.read_sql()

pandas.read_sql(*sql, con, index_col=None, coerce_float=True, params=None, parse_dates=None, columns=None, chunksize=None*) [source]

Read SQL query or database table into a DataFrame.

This function is a convenience wrapper around `read_sql_table` and `read_sql_query` (for backward compatibility). It will delegate to the specific function depending on the provided input. A SQL query will be routed to `read_sql_query`, while a database table name will be routed to `read_sql_table`. Note that the delegated function might have more specific notes about their functionality not listed here.

Parameters:

sql : *str or SQLAlchemy Selectable (select or text object)*

SQL query to be executed or a table name.

con : *SQLAlchemy connectable, str, or sqlite3 connection*

Using SQLAlchemy makes it possible to use any DB supported by that library. If a DBAPI2 object, only sqlite3 is supported. The user is responsible for engine disposal and connection closure for the SQLAlchemy connectable; str connections are closed automatically. See here.

index_col : *str or list of str, optional, default: None*

Column(s) to set as index(MultiIndex).

coerce_float : *bool, default True*

Attempts to convert values of non-string, non-numeric objects (like decimal.Decimal) to floating point, useful for SQL result sets.

params : *list, tuple or dict, optional, default: None*

List of parameters to pass to execute method. The syntax used to pass parameters is database driver dependent. Check your database driver documentation for which of the five syntax styles, described in PEP 249's paramstyle, is supported. Eg. for psycopg2, uses %(name)s so use params={'name' : 'value'}.

parse_dates : *list or dict, default: None*

- List of column names to parse as dates.
- Dict of `{column_name: format string}` where format string is strftime compatible in case of parsing string times, or is one of (D, s, ns, ms, us) in case of parsing integer timestamps.
- Dict of `{column_name: arg dict}`, where the arg dict corresponds to the keyword arguments of `pandas.to_datetime()` Especially useful with databases without native Datetime support, such as SQLite.

columns : *list, default: None*

List of column names to select from SQL table (only used when reading a table).

chunksize : *int, default None*

If specified, return an iterator where *chunksize* is the number of rows to include in each chunk.

Returns: **DataFrame or Iterator[DataFrame]**

```
pd.read_sql("SELECT * FROM Movies", conn).head(2)
```

	id	title	revenue	budget	belongs_to_collection_name	release_date
0	299534	Avengers: Endgame	2797.800564	356.0	The Avengers Collection	2019-04-24 00:00:00
1	19995	Avatar	2787.965087	237.0	Avatar Collection	2009-12-10 00:00:00

```
pd.read_sql("SELECT * FROM Movies", conn).info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 18 entries, 0 to 17
Data columns (total 6 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   id                          18 non-null     int64
 1   title                       18 non-null     object
 2   revenue                     18 non-null     float64
 3   budget                      18 non-null     float64
 4   belongs_to_collection_name  15 non-null     object
 5   release_date                18 non-null     object
dtypes: float64(2), int64(1), object(3)
memory usage: 992.0+ bytes
```

parse_dates = 'release_dates'

- so our imported dates will be datetime objects

```
df = pd.read_sql('SELECT * FROM Movies', conn, index_col = 'id', parse_dates = 'release
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18 entries, 299534 to 260513
Data columns (total 5 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   title                       18 non-null     object
 1   revenue                     18 non-null     float64
 2   budget                      18 non-null     float64
 3   belongs_to_collection_name  15 non-null     object
 4   release_date                18 non-null     datetime64[ns]
dtypes: datetime64[ns](1), float64(2), object(2)
memory usage: 864.0+ bytes
```

```
genres_df = pd.read_sql("SELECT * FROM Genres", conn, index_col = 'id')
genres_df.head(2)
```

id	genre_id	genre_name
299534	12	Adventure
299534	878	Science Fiction

5.Some Simple SQL Queries¶

- **Perform** the following simple **SQL Queries** and **store** the results in DataFrames:

Load the full "Movies" Table.

- it is common to use new lines for query specifics

```
pd.read_sql("SELECT * \
            FROM Movies", conn).head(3)
```

	id	title	revenue	budget	belongs_to_collection_name	release_date
0	299534	Avengers: Endgame	2797.800564	356.0	The Avengers Collection	2019-04-24 00:00:00
1	19995	Avatar	2787.965087	237.0	Avatar Collection	2009-12-10 00:00:00
2	140607	Star Wars: The Force Awakens	2068.223624	245.0	Star Wars Collection	2015-12-15 00:00:00

Load the columns "id", "revenue" and "release_date" from "Movies".

```
pd.read_sql("SELECT id, revenue, release_date FROM Movies", conn).head(3)
```

	id	revenue	release_date
0	299534	2797.800564	2019-04-24 00:00:00
1	19995	2787.965087	2009-12-10 00:00:00
2	140607	2068.223624	2015-12-15 00:00:00

SUM

Get the Total Revenue (sum) over all movies from "Movies".

```
pd.read_sql("SELECT SUM(revenue) FROM Movies", conn)
```

	SUM(revenue)
0	29748.575327

conn.execute()

- when using this for queries, you do not have to specify the connection
- but you must specify `.fetchall()`
- will return a tuple in this case, `[0][0]` returns just the number

```
conn.execute("SELECT SUM(revenue) FROM Movies").fetchall()[0][0]
```

```
29748.575327000002
```

COUNT

Count the number of Movies in "Movies".

```
pd.read_sql('SELECT COUNT(title) FROM Movies', conn)
```

	COUNT(title)
0	18

```
pd.read_sql('SELECT COUNT(*) FROM Movies', conn)
```

	COUNT(*)
0	18

Count the number of Movies that do belong to a collection.

```
pd.read_sql('SELECT COUNT(belongs_to_collection_name) FROM Movies', conn)
```

	COUNT(belongs_to_collection_name)
0	15

AVG

Get the average budget from "Movies".

```
pd.read_sql("SELECT AVG(budget) FROM Movies", conn)
```

	AVG(budget)
0	209.055556

CONDITIONAL FILTERING Queries

- Only load films that meet certain conditions

6. More Advanced SQL Queries

- **Perform** the following advanced **SQL Queries** and **store** the results in DataFrames:

WHERE

Load all columns for the movie with movie id 597.

```
pd.read_sql("SELECT * FROM Movies WHERE id = 597", conn)
```

	id	title	revenue	budget	belongs_to_collection_name	release_date
0	597	Titanic	1845.034188	200.0	None	1997-11-18 00:00:00

Load all columns for all movies with a revenue greater than 2000 (MUSD).

```
pd.read_sql("SELECT * FROM Movies WHERE revenue > 2000", conn)
```

	id	title	revenue	budget	belongs_to_collection_name	release_date
0	299534	Avengers: Endgame	2797.800564	356.0	The Avengers Collection	2019-04-24 00:00:00
1	19995	Avatar	2787.965087	237.0	Avatar Collection	2009-12-10 00:00:00
2	140607	Star Wars: The Force Awakens	2068.223624	245.0	Star Wars Collection	2015-12-15 00:00:00
3	299536	Avengers: Infinity War	2046.239637	300.0	The Avengers Collection	2018-04-25 00:00:00

< and >

Load all columns for all movies with a revenue greater than 1500 (MUSD) and a budget below 200 (MUSD).

```
pd.read_sql("SELECT * FROM Movies WHERE revenue > 1500 AND budget < 200", conn)
```

	id	title	revenue	budget	belongs_to_collection_name	release_date
0	135397	Jurassic World	1671.713208	150.0	Jurassic Park Collection	2015-06-06 00:00:00
1	168259	Furious 7	1515.047671	190.0	The Fast and the Furious Collection	2015-04-01 00:00:00

MIN

Get the minimum budget from those movies with a revenue greater than 1250 (MUSD).

```
pd.read_sql("SELECT MIN(budget) FROM Movies WHERE revenue > 1250", conn)
```

	MIN(budget)
0	125.0

DISTINCT

Get all unique collection Names from "Movies".

```
pd.read_sql("SELECT DISTINCT belongs_to_collection_name FROM Movies", conn)
```

	belongs_to_collection_name
0	The Avengers Collection
1	Avatar Collection
2	Star Wars Collection
3	None
4	Jurassic Park Collection
5	The Fast and the Furious Collection

	belongs_to_collection_name
6	Black Panther Collection
7	Harry Potter Collection
8	Frozen Collection
9	The Incredibles Collection

ORDER BY and DESC

Load all movies (all columns) and sort by budget from high to low.

```
pd.read_sql("SELECT * FROM Movies ORDER BY budget DESC", conn)
```

	id	title	revenue	budget	belongs_to_collection_name	release_date
0	299534	Avengers: Endgame	2797.800564	356.0	The Avengers Collection	2019-04-24 00:00:00
1	299536	Avengers: Infinity War	2046.239637	300.0	The Avengers Collection	2018-04-25 00:00:00
2	420818	The Lion King	1656.943394	260.0	None	2019-07-12 00:00:00
3	99861	Avengers: Age of Ultron	1405.403694	250.0	The Avengers Collection	2015-04-22 00:00:00
4	140607	Star Wars: The Force Awakens	2068.223624	245.0	Star Wars Collection	2015-12-15 00:00:00
5	19995	Avatar	2787.965087	237.0	Avatar Collection	2009-12-10 00:00:00
6	24428	The Avengers	1519.557910	220.0	The Avengers Collection	2012-04-25 00:00:00
7	597	Titanic	1845.034188	200.0	None	1997-11-18 00:00:00
8	284054	Black Panther	1346.739107	200.0	Black Panther Collection	2018-02-13 00:00:00
9	181808	Star Wars: The Last Jedi	1332.539889	200.0	Star Wars Collection	2017-12-13 00:00:00
10	260513	Incredibles 2	1241.891456	200.0	The Incredibles Collection	2018-06-14 00:00:00
11	168259	Furious 7	1515.047671	190.0	The Fast and the Furious Collection	2015-04-01 00:00:00
12	351286	Jurassic World: Fallen Kingdom	1303.459585	170.0	Jurassic Park Collection	2018-06-06 00:00:00
13	321612	Beauty and the Beast	1263.521126	160.0	None	2017-03-16 00:00:00
14	135397	Jurassic World	1671.713208	150.0	Jurassic Park Collection	2015-06-06 00:00:00
15	330457	Frozen II	1330.764959	150.0	Frozen Collection	2019-11-20 00:00:00
16	109445	Frozen	1274.219009	150.0	Frozen Collection	2013-11-27 00:00:00
17	12445	Harry Potter and the Deathly Hallows: Part 2	1341.511219	125.0	Harry Potter Collection	2011-07-07 00:00:00

IS NULL

Load all movies (all columns) that DO NOT belong to a collection.

```
pd.read_sql("SELECT * FROM Movies WHERE belongs_to_collection_name IS NULL", conn)
```

	id	title	revenue	budget	belongs_to_collection_name	release_date
0	597	Titanic	1845.034188	200.0	None	1997-11-18 00:00:00
1	420818	The Lion King	1656.943394	260.0	None	2019-07-12 00:00:00
2	321612	Beauty and the Beast	1263.521126	160.0	None	2017-03-16 00:00:00

IS NOT NULL

Load all movies (all columns) that belong to a collection.

```
pd.read_sql("SELECT * FROM Movies WHERE belongs_to_collection_name IS NOT NULL", conn)
```

	id	title	revenue	budget	belongs_to_collection_name	release_date
0	299534	Avengers: Endgame	2797.800564	356.0	The Avengers Collection	2019-04-24 00:00:00
1	19995	Avatar	2787.965087	237.0	Avatar Collection	2009-12-10 00:00:00
2	140607	Star Wars: The Force Awakens	2068.223624	245.0	Star Wars Collection	2015-12-15 00:00:00
3	299536	Avengers: Infinity War	2046.239637	300.0	The Avengers Collection	2018-04-25 00:00:00
4	135397	Jurassic World	1671.713208	150.0	Jurassic Park Collection	2015-06-06 00:00:00
5	24428	The Avengers	1519.557910	220.0	The Avengers Collection	2012-04-25 00:00:00
6	168259	Furious 7	1515.047671	190.0	The Fast and the Furious Collection	2015-04-01 00:00:00
7	99861	Avengers: Age of Ultron	1405.403694	250.0	The Avengers Collection	2015-04-22 00:00:00
8	284054	Black Panther	1346.739107	200.0	Black Panther Collection	2018-02-13 00:00:00
9	12445	Harry Potter and the Deathly Hallows: Part 2	1341.511219	125.0	Harry Potter Collection	2011-07-07 00:00:00
10	181808	Star Wars: The Last Jedi	1332.539889	200.0	Star Wars Collection	2017-12-13 00:00:00
11	330457	Frozen II	1330.764959	150.0	Frozen Collection	2019-11-20 00:00:00
12	351286	Jurassic World: Fallen Kingdom	1303.459585	170.0	Jurassic Park Collection	2018-06-06 00:00:00
13	109445	Frozen	1274.219009	150.0	Frozen Collection	2013-11-27 00:00:00
14	260513	Incredibles 2	1241.891456	200.0	The Incredibles Collection	2018-06-14 00:00:00

LIKE and %

Load all movies (all columns) where "Avengers..." is in the title.

```
pd.read_sql("SELECT * FROM Movies WHERE title LIKE 'Avengers%'", conn)
```

	id	title	revenue	budget	belongs_to_collection_name	release_date
0	299534	Avengers: Endgame	2797.800564	356.0	The Avengers Collection	2019-04-24 00:00:00
1	299536	Avengers: Infinity War	2046.239637	300.0	The Avengers Collection	2018-04-25 00:00:00
2	99861	Avengers: Age of Ultron	1405.403694	250.0	The Avengers Collection	2015-04-22 00:00:00

---

7. Join Queries

- **Perform** the following **SQL Join Queries** and **store** the results in DataFrames:

JOIN and ON and the syntax

Join "Movies" and "Votes" (all columns).

```
pd.read_sql("SELECT * \
             FROM Movies \
             JOIN Votes \
             ON Movies.id = Votes.id", conn).head(3)
```

	id	title	revenue	budget	belongs_to_collection_name	release_date	id	vote_count	vote_aver
0	299534	Avengers: Endgame	2797.800564	356.0	The Avengers Collection	2019-04-24 00:00:00	299534	12607	
1	19995	Avatar	2787.965087	237.0	Avatar Collection	2009-12-10 00:00:00	19995	21000	
2	140607	Star Wars: The Force Awakens	2068.223624	245.0	Star Wars Collection	2015-12-15 00:00:00	140607	14205	

Specifying columns when joining

Join "Movies" and "Votes" (only the columns "id", "title", "vote_average").

```
pd.read_sql("SELECT Movies.id, Movies.title, Votes.vote_average \
             FROM Movies \
             JOIN VOTES \
             ON Movies.id = Votes.id", conn, index_col = 'id').head(3)
```

	title	vote_average
id		

id	title	vote_average
299534	Avengers: Endgame	8.3
19995	Avatar	7.4
140607	Star Wars: The Force Awakens	7.4

Join "Movies" and "Votes" (only the columns "id", "title", "vote_average") and return only those movies with vote_average > 8.

```
pd.read_sql("SELECT Movies.id, Movies.title, Votes.vote_average \
            FROM Movies \
            JOIN Votes ON Movies.id = Votes.id \
            WHERE Votes.vote_average > 8", conn, index_col = 'id')
```

id	title	vote_average
299534	Avengers: Endgame	8.3
299536	Avengers: Infinity War	8.3
12445	Harry Potter and the Deathly Hallows: Part 2	8.1

Join "Movies" and "Votes" (only the columns "id", "title", "vote_average") and return only those movies with vote_average > 8 and in ascending budget order.

```
pd.read_sql('SELECT Movies.id, Movies.title, Votes.vote_average \
            FROM Movies \
            JOIN Votes ON Movies.id = Votes.id \
            WHERE Votes.vote_average > 8 \
            ORDER BY Movies.budget ASC', conn, index_col='id')
```

id	title	vote_average
12445	Harry Potter and the Deathly Hallows: Part 2	8.1
299536	Avengers: Infinity War	8.3
299534	Avengers: Endgame	8.3

---

8. Further Advanced Queries

- **Perform** the following advanced **SQL Queries** and **store** the results in DataFrames:

LEFT JOIN

Get the Total Revenue (sum) for each Production Company.

```
prod_df = pd.read_sql("SELECT Production.id, Production.prod_name, \
                       Movies.revenue, Movies. title\
                       FROM Production \
                       LEFT JOIN Movies \
                       ON Production.id = Movies.id", conn)

prod_df.head(3)
```

	id	prod_name	revenue	title
0	299534	Marvel Studios	2797.800564	Avengers: Endgame
1	19995	Dune Entertainment	2787.965087	Avatar
2	19995	Lightstorm Entertainment	2787.965087	Avatar

INNER JOIN

- Also the default setting for JOIN

```
prod_df = pd.read_sql("SELECT Production.id, Production.prod_name, \
                       Movies.revenue, Movies. title\
                       FROM Production \
                       INNER JOIN Movies \
                       ON Production.id = Movies.id", conn)

prod_df.head(3)
```

	id	prod_name	revenue	title
0	299534	Marvel Studios	2797.800564	Avengers: Endgame
1	19995	Dune Entertainment	2787.965087	Avatar
2	19995	Lightstorm Entertainment	2787.965087	Avatar

```
prod_df.groupby('prod_name').revenue.sum().sort_values(ascending = False)
```

```
prod_name
Walt Disney Pictures                    9446.618940
Marvel Studios                          9115.740912
20th Century Fox                        4632.999275
Lightstorm Entertainment                4632.999275
Universal Pictures                      4490.220464
Lucasfilm                               3400.763513
Paramount                               3364.592098
Fuji Television Network                 3186.760879
Dentsu                                  3186.760879
Legendary Entertainment                 2975.172793
Amblin Entertainment                    2975.172793
Ingenious Media                         2787.965087
Dune Entertainment                      2787.965087
Walt Disney Animation Studios           2604.983968
```

```
Bad Robot                                        2068.223624
Truenorth Productions                            2068.223624
The Kennedy/Marshall Company                     1671.713208
Fairview Entertainment                           1656.943394
Colorado Office of Film, Television & Media      1515.047671
Abu Dhabi Film Commission                        1515.047671
China Film Co.                                   1515.047671
Media Rights Capital                             1515.047671
One Race                                         1515.047671
Original Film                                    1515.047671
Québec Production Services Tax Credit            1515.047671
Warner Bros. Pictures                            1341.511219
Heyday Films                                     1341.511219
Ram Bergman Productions                          1332.539889
Perfect World Pictures                           1303.459585
Mandeville Films                                 1263.521126
Pixar                                            1241.891456
Name: revenue, dtype: float64
```

Get all Production Companies for the movie "Titanic".

```
pd.read_sql("SELECT Production.prod_name \
            FROM Production \
            LEFT JOIN Movies \
            ON Production.id == Movies.id \
            WHERE title = 'Titanic'", conn)
```

	prod_name
0	20th Century Fox
1	Lightstorm Entertainment
2	Paramount

Get the Total Revenue (sum) for each Genre.

```
genres_df = pd.read_sql("SELECT Genres.id, Genres.genre_name, \
                        Movies.title, Movies.revenue \
                        FROM Genres \
                        JOIN Movies \
                        ON Genres.id = Movies.id", conn)
```

```
genres_df.head(3)
```

	id	genre_name	title	revenue
0	299534	Adventure	Avengers: Endgame	2797.800564
1	299534	Science Fiction	Avengers: Endgame	2797.800564
2	299534	Action	Avengers: Endgame	2797.800564

```
genres_df.groupby('genre_name').revenue.sum().sort_values(ascending = False)
```

```
genre_name
Adventure          25124.972342
Action             21036.581432
Science Fiction    18279.642305
Fantasy             8807.960163
Family              6767.339944
Animation           3846.875424
Thriller            3186.760879
Romance             3108.555314
Drama               1845.034188
Name: revenue, dtype: float64
```

Get all Genres for the movie "Frozen II".

```
pd.read_sql("SELECT Genres.genre_name \
            FROM Genres \
            JOIN Movies \
            ON Genres.id = Movies.id \
            WHERE title = 'Frozen II'", conn)
```

	genre_name
0	Adventure
1	Animation
2	Family