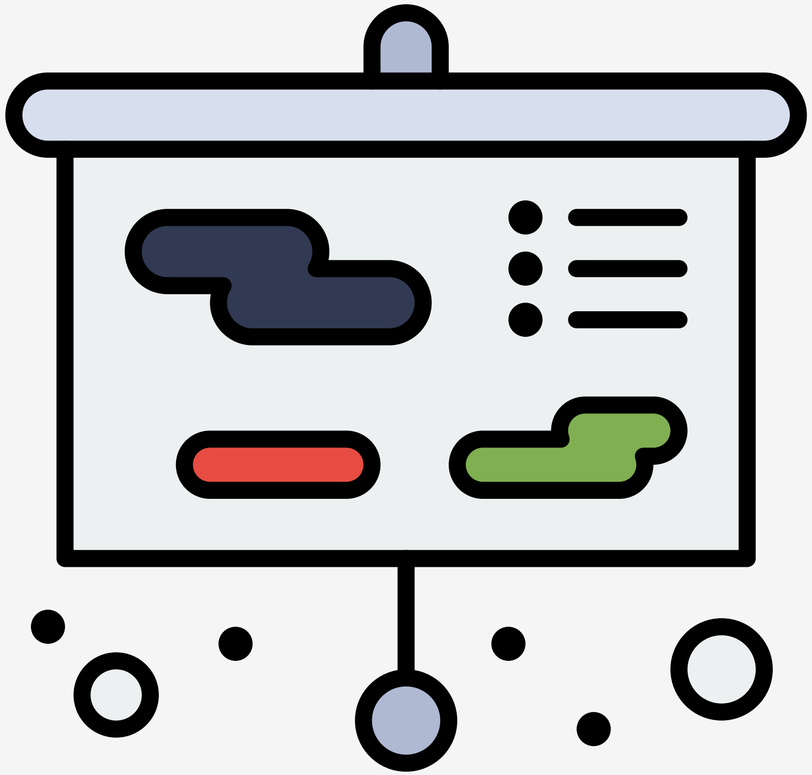
SERVER-SIDE
WEB DEVELOPMENT

Lecture 5

TODAY'S TOPICS

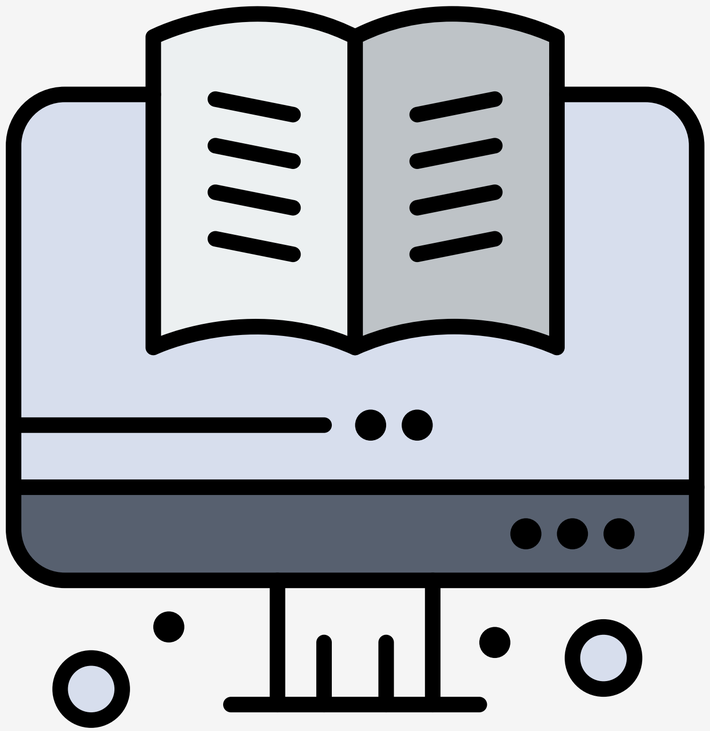


- INSERT statements
- UPDATE statements
- DELETE statements
- Joins
- Functions
- **Participation:** Movie Mayhem II
- **Exercise:** Seussology DB II

QUESTIONS

INSERT STATEMENTS

INSERT STATEMENTS



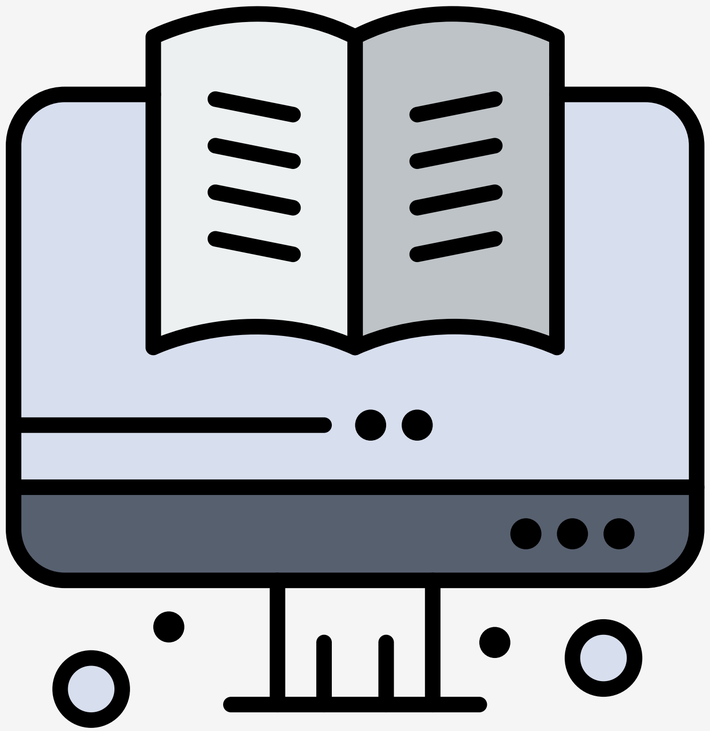
- The `INSERT` statement is used add new data to the database.
- A table name must be included, while a list o columns is optional
- If no columns are provide all columns will be assumed
- A list of values, in the same order as the columns is required
- If no value is given for a required column, an error will occur

INSERT

```sql
-- Insert a record into the `movies` table
INSERT
INTO movies (`movie_title`, `director`)
VALUES ('Pulp Fiction', 'Quentin Tarantino');
```

UPDATE STATEMENTS

UPDATE STATEMENTS



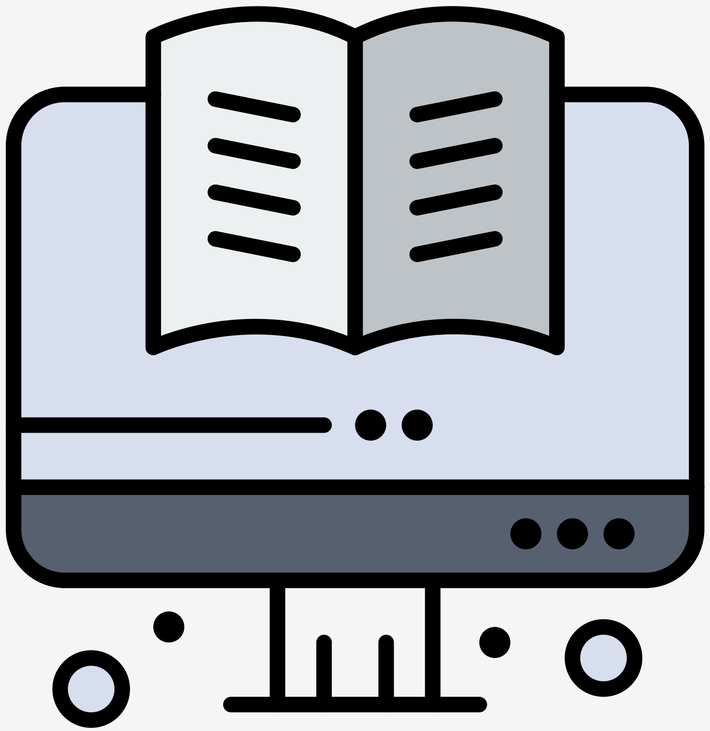
- The `UPDATE` is used to modify or update existing records in a table
- The `SET` clause is used to update value of a specific column
- The `WHERE` clause is used to specify which records should be updated
- If the `WHERE` clause is omitted, ***ALL*** records will be updated
- The `LIMIT` clause can be used to further limit the impact

UPDATE

```sql
-- Update the `director` for the movie with
-- the `movie_title` 'Transcendence'
UPDATE `movies`
SET `director` = 'Wally Pfister'
WHERE `movie_title` = 'Transcendence'
LIMIT 1;
```

DELETE STATEMENTS

DELETE STATEMENTS



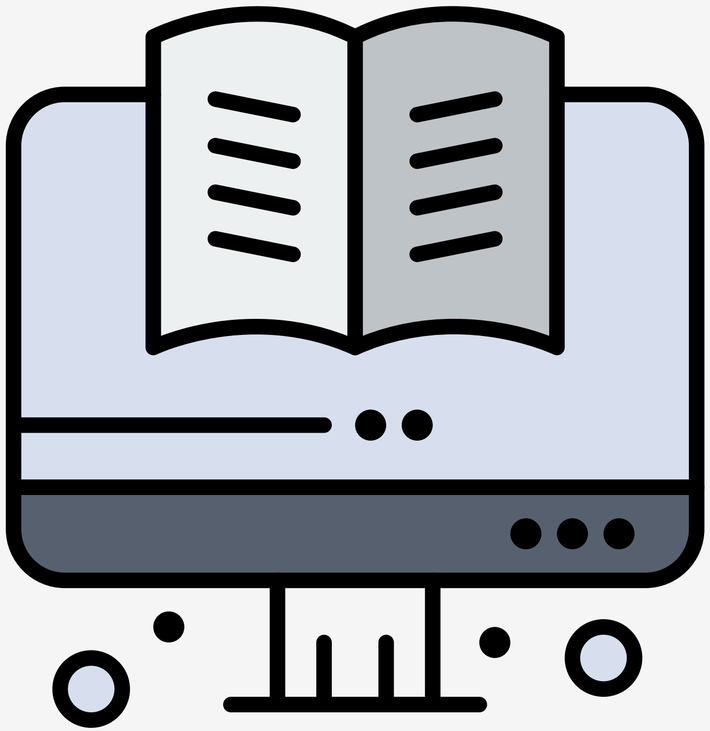
- The `DELETE` is used to remove one or more records from a table
- The `FROM` clause is used to set the table
- The `WHERE` clause is used to specify which records should be removed
- If the `WHERE` clause is omitted, ***ALL*** records will be removed
- The `LIMIT` clause can be used to further limit the impact

DELETE

```sql
-- Remove the movie with the movie_id of 24
DELETE
FROM movies
WHERE movie_id = 24
LIMIT 1;
```

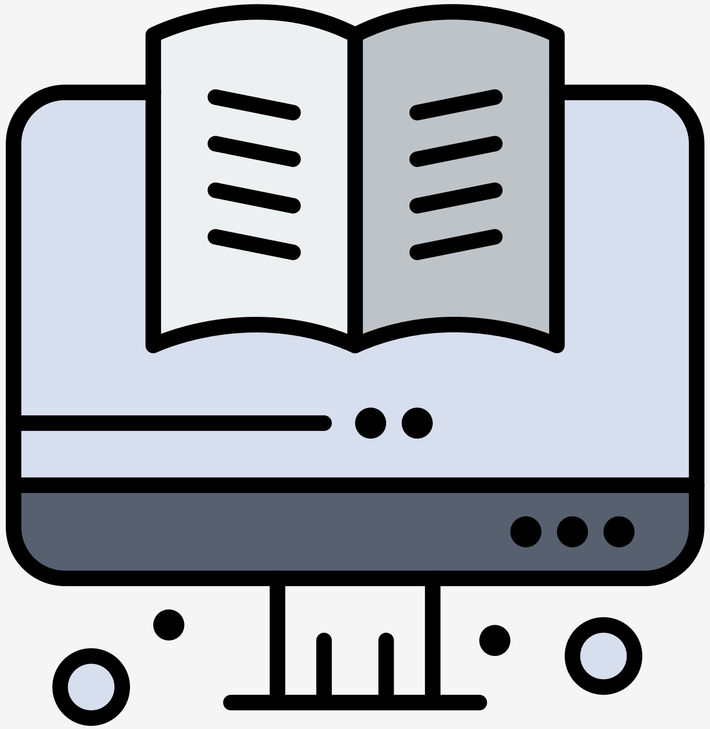
JOINS

JOINS

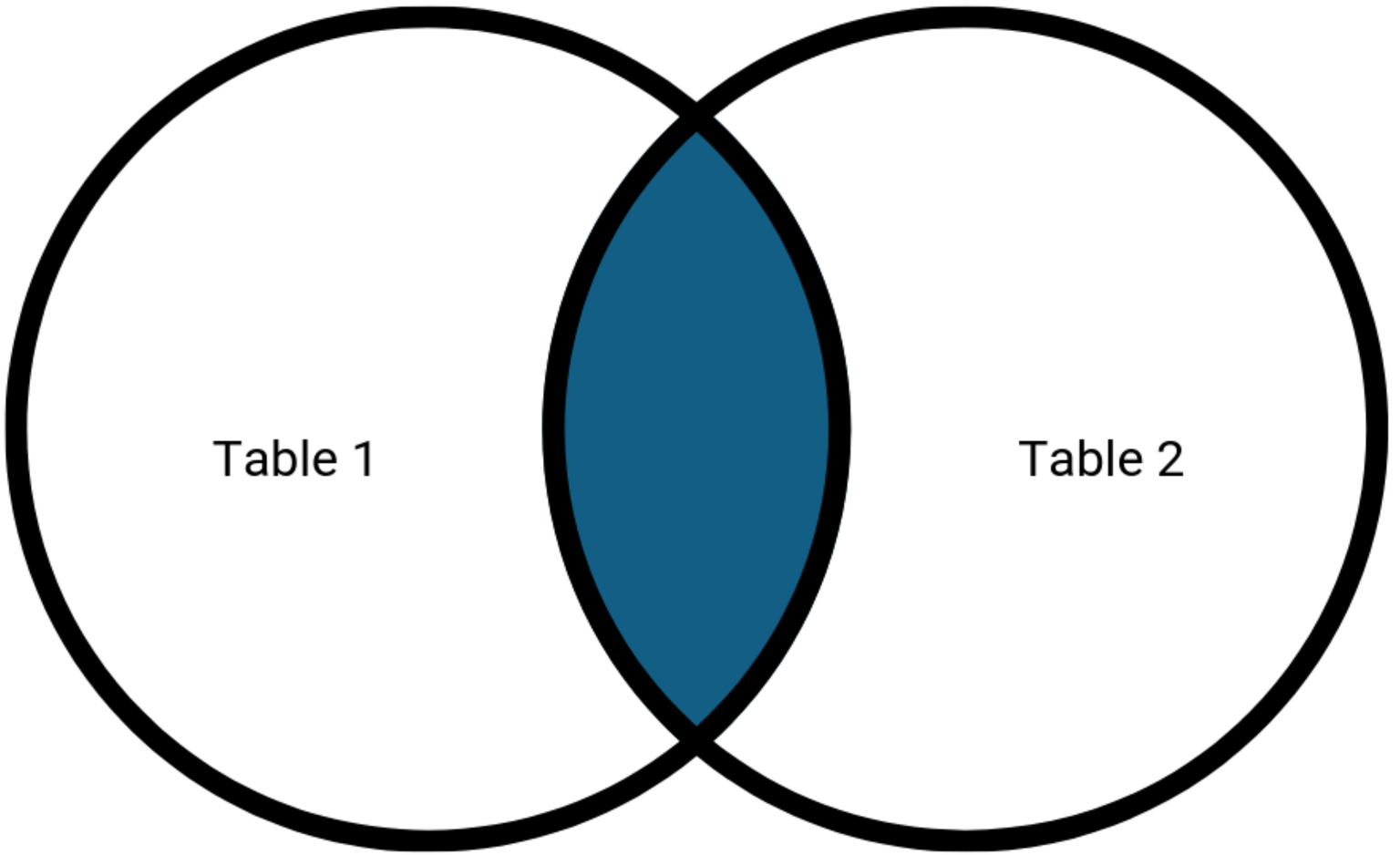


- The `JOIN` clause is used to return the combined result two or more tables based on a related column
- The `JOIN` clause cause no actual change to the tables
- There are three different types of JOIN clauses:
  - `INNER JOIN`
  - `LEFT JOIN`
  - `RIGHT JOIN`

INNER JOIN



- The `INNER JOIN` clause retrieves all of the records that has matching values in both tables
- When working with more than one table with the same column name, the table name must be prefixed to the column name
  - *`table_name.column_name`*



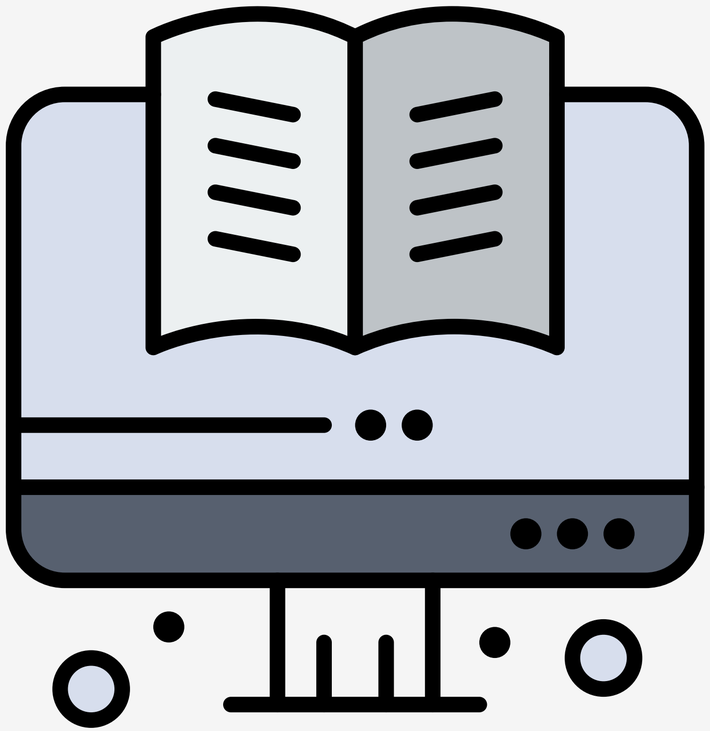
INNER JOIN

INNER JOIN

```sql
-- Retrieving movie title and genre title
SELECT m.movie_title, g.genre_title
FROM movies as m
INNER JOIN genres as g
ON m.genre_id = g.genre_id
```

movie_title	genre_title
Labyrinth	Fantasy
Highlander	Fantasy
Alien	Sci-Fi
Conan the Barbarian	Fantasy

OUTER JOIN



- The `LEFT JOIN` clause and RIGHT JOIN clause are referred to as Outer Joins
- The `LEFT JOIN` will return all the records from the left table (table 1), and the matched records from the right table (table 2).
- The `RIGHT JOIN` will return all the records from the right table (table 2), and the matched records from the left table (table 1)

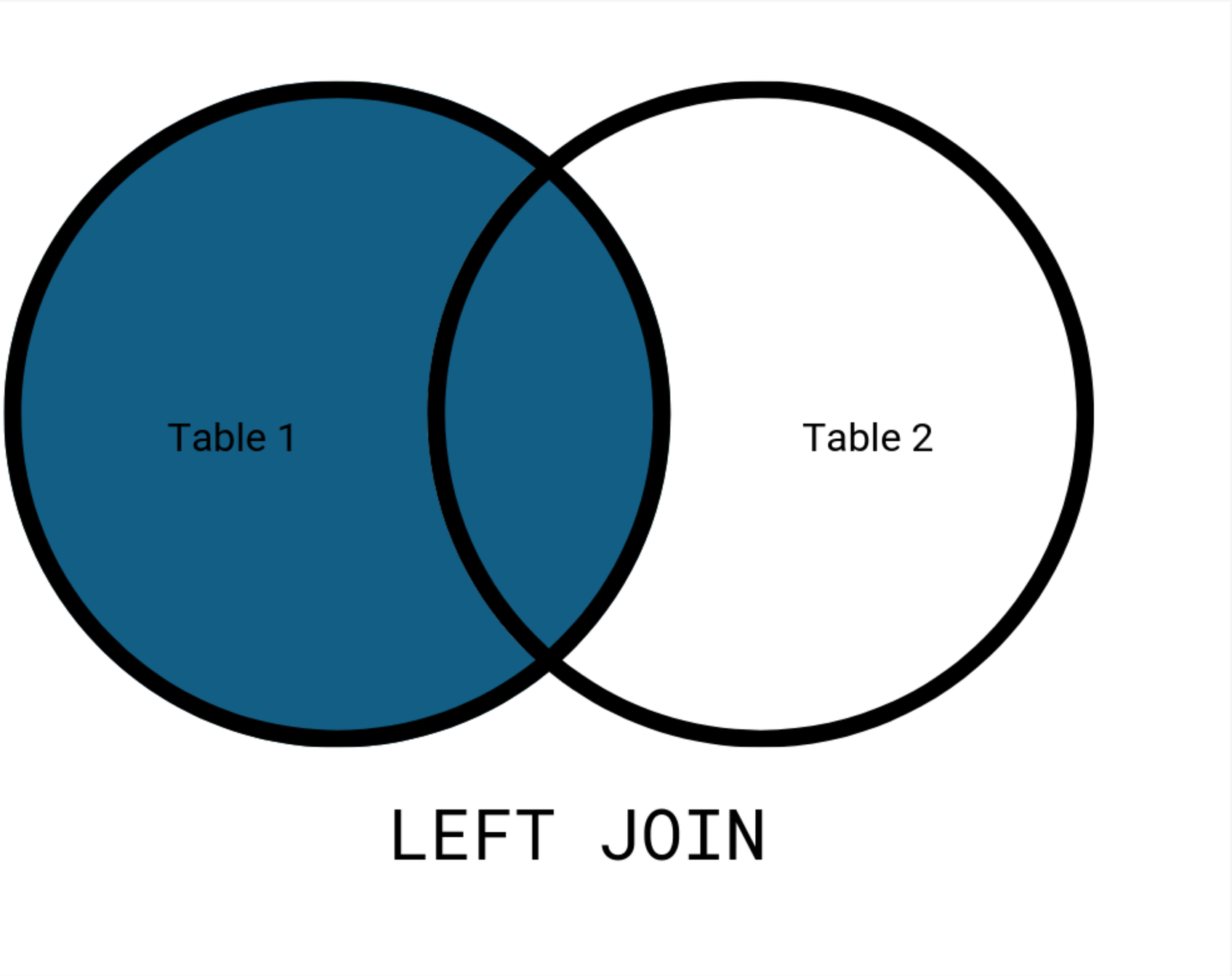
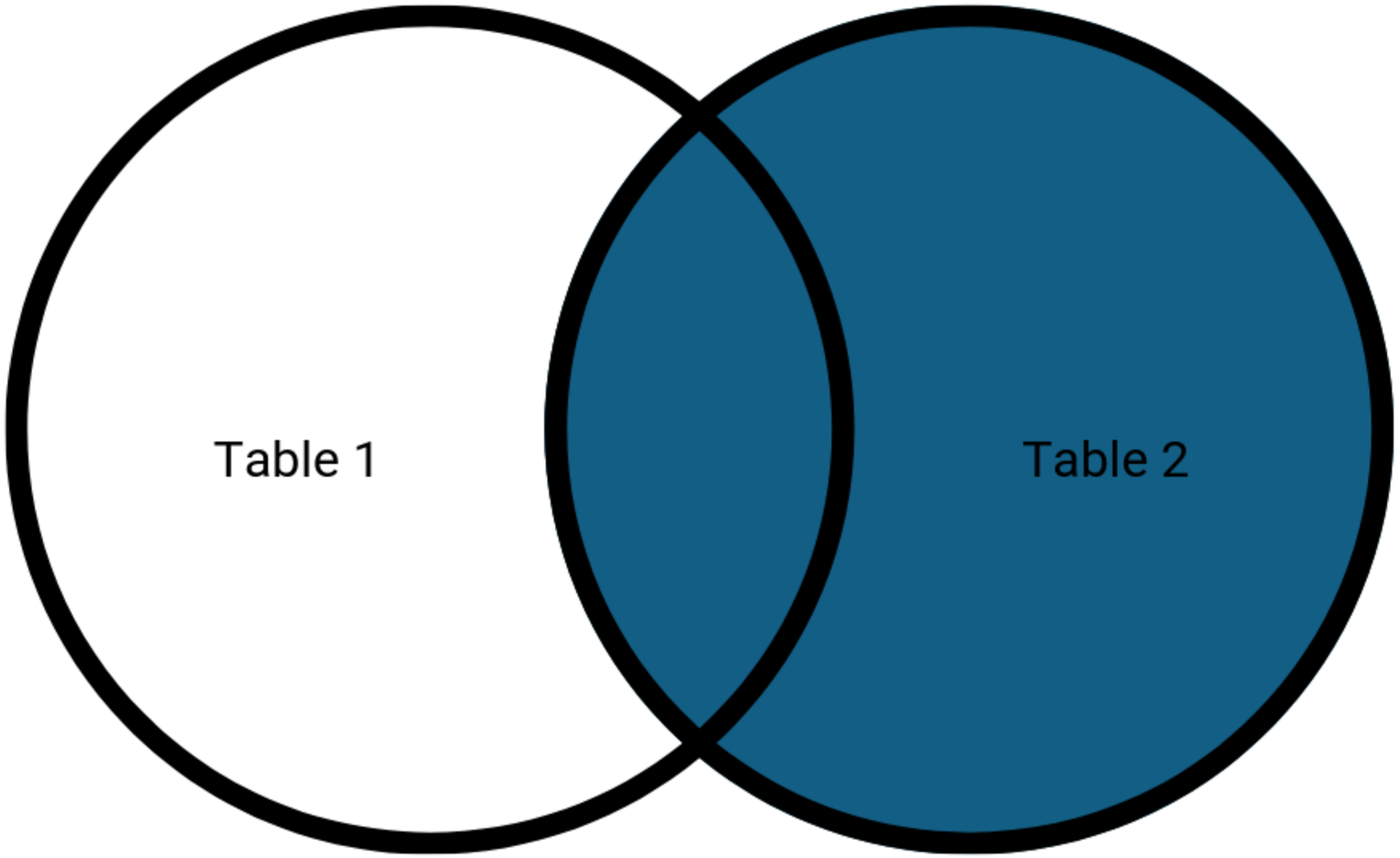


Table 1
Table 2
LEFT JOIN



RIGHT JOIN

LEFT JOIN

```sql
-- List all books, even those without quotes
SELECT book_title, quote
FROM books
LEFT JOIN quotes
ON books.book_id = quotes.book_id
```

book_title	quote
And to Think That I Saw It on Mulberry...	Stop telling such outlandish tales. Stop...
And to Think That I Saw It on Mulberry...	For I had a story that no one could beat...
Bartholomew and the Oobleck	*NULL*
The 500 Hats of Bartholomew Cubbins	Dig a hole five furlongs deep, Down...

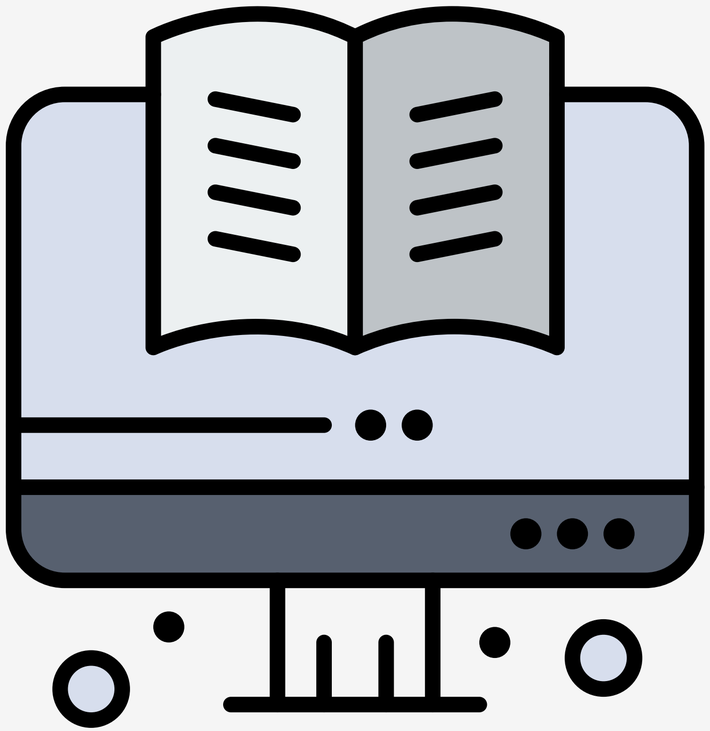
RIGHT JOIN

```sql
-- List only the books with quotes
SELECT book_title, quote
FROM books
RIGHT JOIN quotes
ON books.book_id = quotes.book_id
```

book_title	quote
And to Think That I Saw It on Mulberry...	Stop telling such outlandish tales. Stop...
And to Think That I Saw It on Mulberry...	For I had a story that no one could beat...
The 500 Hats of Bartholomew Cubbins	Dig a hole five furlongs deep, Down...
If I Ran the Zoo	I think I could find some beasts of a...

AGGREGATE FUNCTIONS

AGGREGATE FUNCTIONS



- Aggregate Functions are use to perform calculations on multiples rows of a single column
  - `COUNT()`
  - `SUM()`
  - `AVG()`
  - `MIN() / MAX()`
- When using Aggregate Functions without `GROUP BY`, only one column can be returned

COUNT

```sql
-- Count all the rows with a movie id
SELECT COUNT(`movie_id`) FROM `movies`;

-- Count all the rows in the table
SELECT COUNT(*) FROM `movies`;
```

COUNT(`movie_id`)
28

COUNT(*)
28

SUM

```sql
-- Total of the year column for every movie
SELECT SUM(`year`) FROM `movies`;

-- Only works with number datatype
```

SUM(`year`)
55931

AVG

```sql
-- Getting the average year for all movies
SELECT AVG(`year`) FROM `movies`;

-- Only works with number datatype
```

AVG(`year`)
1997.5357

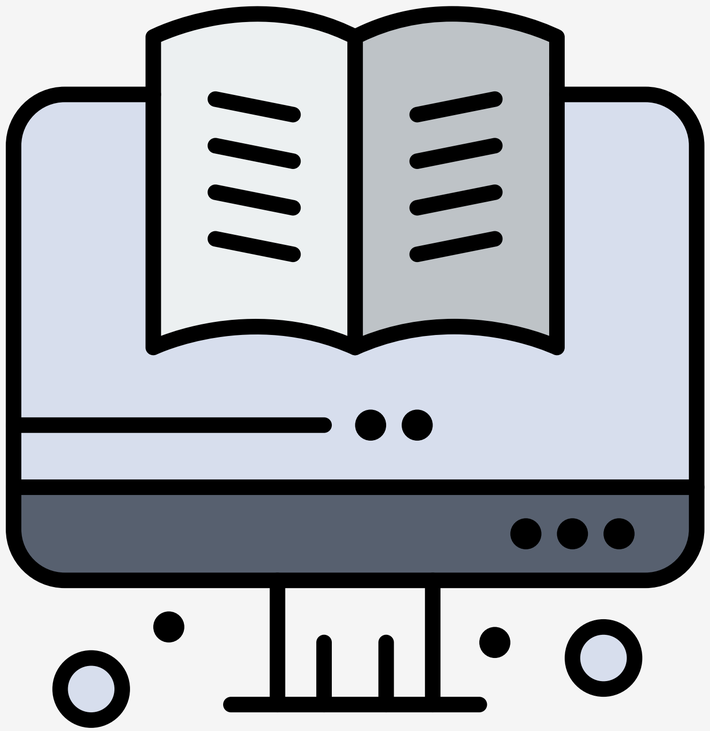
MIN / MAX

```sql
-- The oldest movie in the movies table
SELECT MIN(`year`) as `Oldest` FROM `movies`;

-- The newest movie in the movies table
SELECT MAX(`year`) as `Newest` FROM `movies`;

-- Can be used with strings
```

MIN(`year`)
1977

MAX(`year`)
2016

AGGREGATE FUNCTIONS



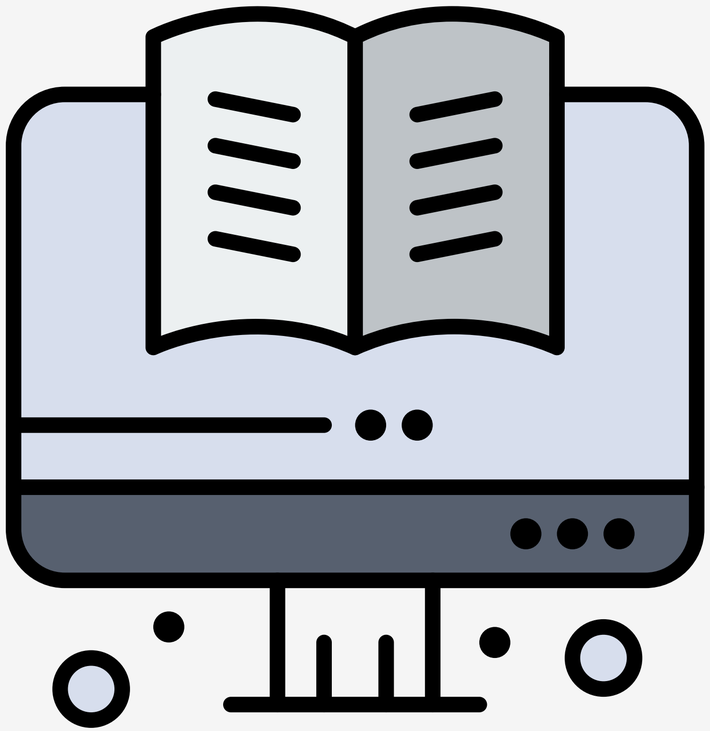
- The `WHERE` clause can be used with aggregate functions
- The `WHERE` clause will filter the rows before the aggregate functions does it calculations

WHERE

```sql
-- The newest movie with the genre 'Sci-Fi'
SELECT MAX(`year`) as `Newest`
FROM `movies`
WHERE `genre_id` =
  (SELECT `genre_id`
  FROM `genres`
  WHERE `genre_title` = 'Sci-Fi' LIMIT 1);
```

Newest
2014

GROUP BY



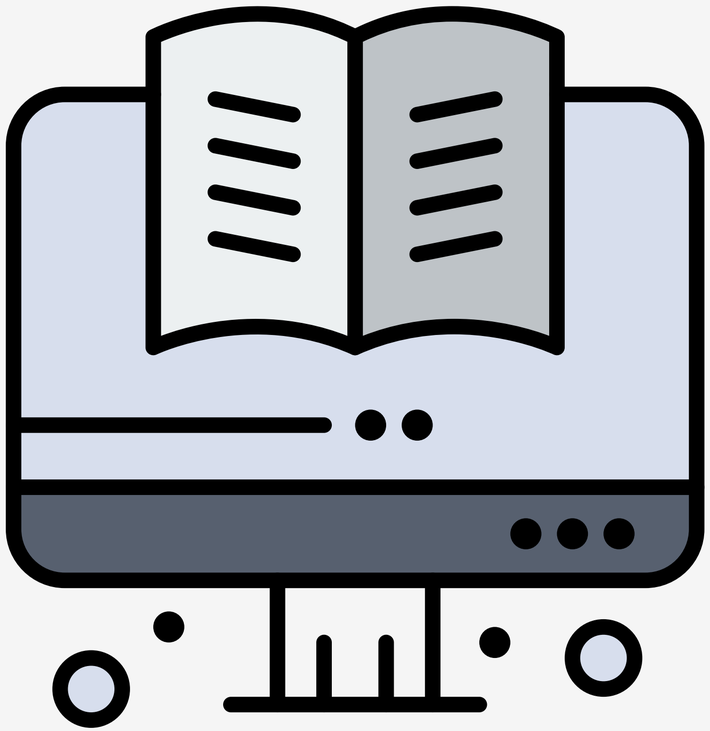
- The `GROUP BY` clause can be used with aggregate functions to group the result by one or more columns
- A separate aggregate row will be returned for each group

GROUP BY

```sql
-- Total number of movies for each year
SELECT COUNT(`movie_id`) as `total`, `year`
FROM `movies`
GROUP BY `year`;
```

total	year
2	1986
1	1979
3	1982
1	2012

GROUP BY



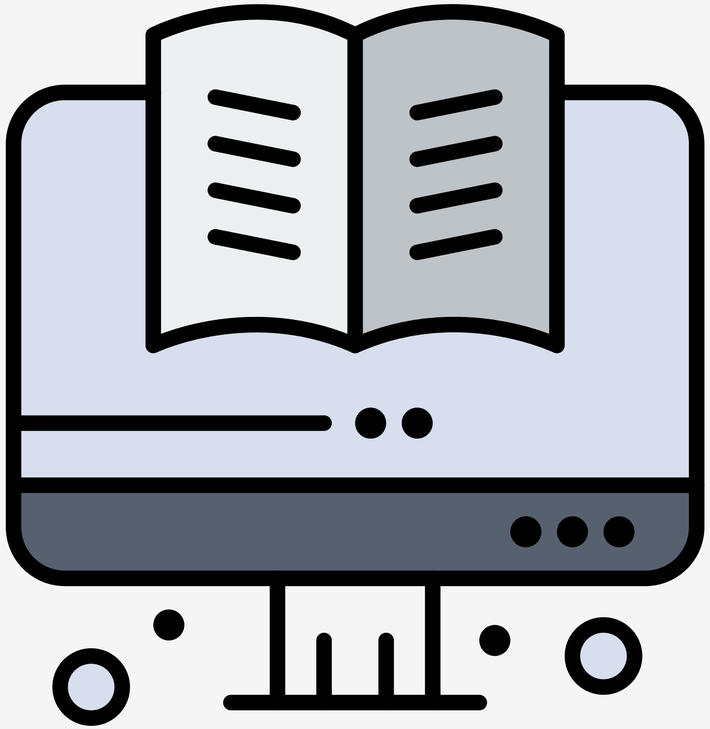
- The `GROUP BY` clause can be used with `INNER JOIN` to group rows across multiple tables

GROUP BY

```sql
-- Total number of movies for each genre title
SELECT COUNT(movie_id) as total, genre_title
FROM movies
INNER JOIN genres
ON movies.genre_id = genres.genre_id
GROUP BY genre_title;
```

total	genre_title
10	Fantasy
16	Sci-Fi
2	Drama

HAVING



- The `WHERE` clause cannot be used after grouping has occurred
- The `HAVING` clause is used to filter results AFTER grouping has occurred and can be used with aggregate functions

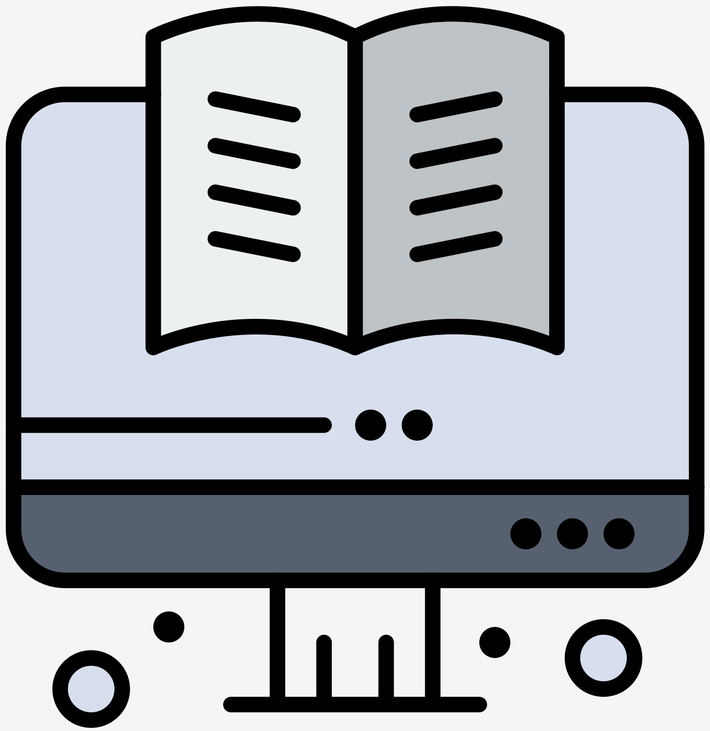
HAVING

```sql
-- Total number of movies by year
-- for years having more than 1 movie
SELECT COUNT(*) as total, year
FROM movies
GROUP BY year
HAVING total > 1
```

total	year
2	1986
3	1982
2	1981

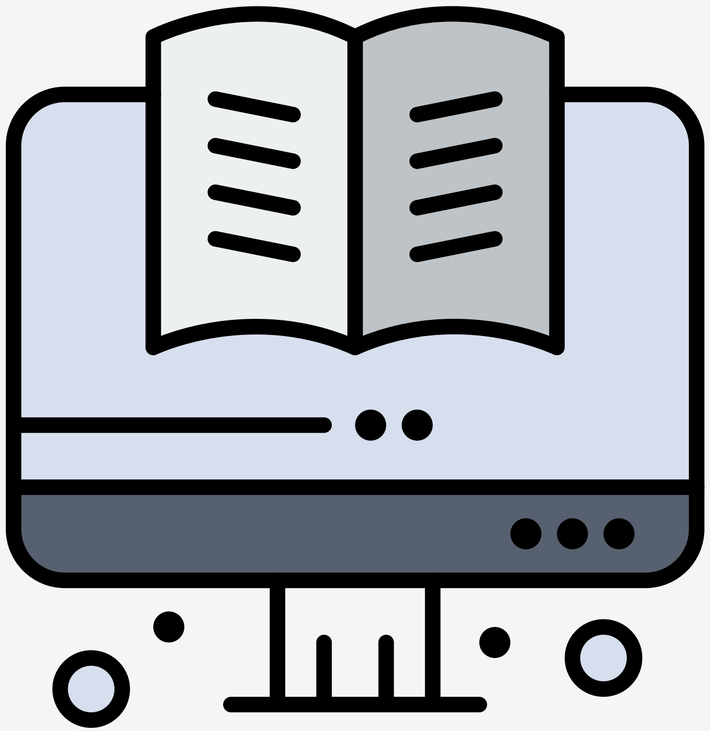
OTHER FUNCTIONS

STRING FUNCTIONS



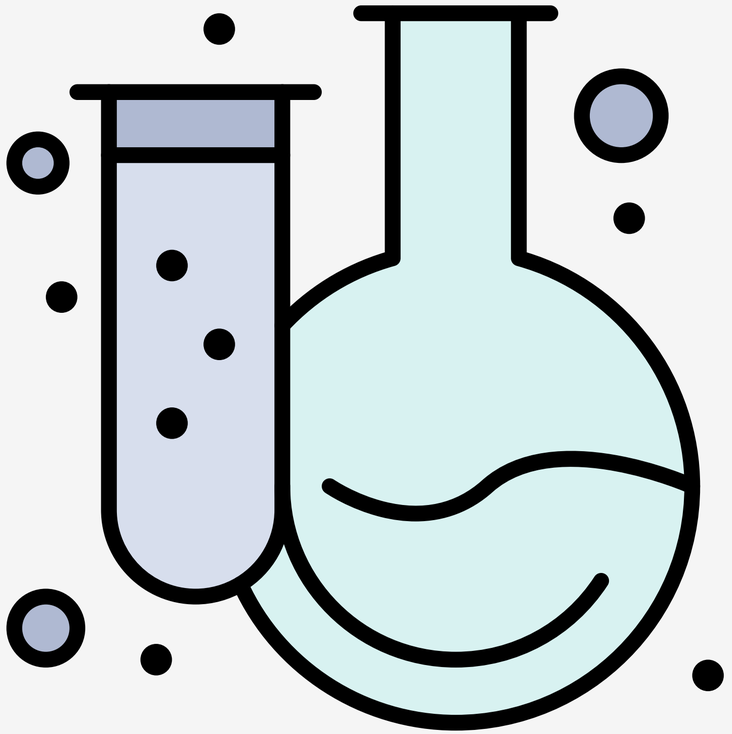
- String functions are used to alter the returned string data *ONLY*
- String functions include:
  - `UPPER() / LOWER()`
  - `LENGTH()`
  - `CONCAT()`
  - `TRIM()`
  - `FORMAT()`

NUMERIC FUNCTIONS



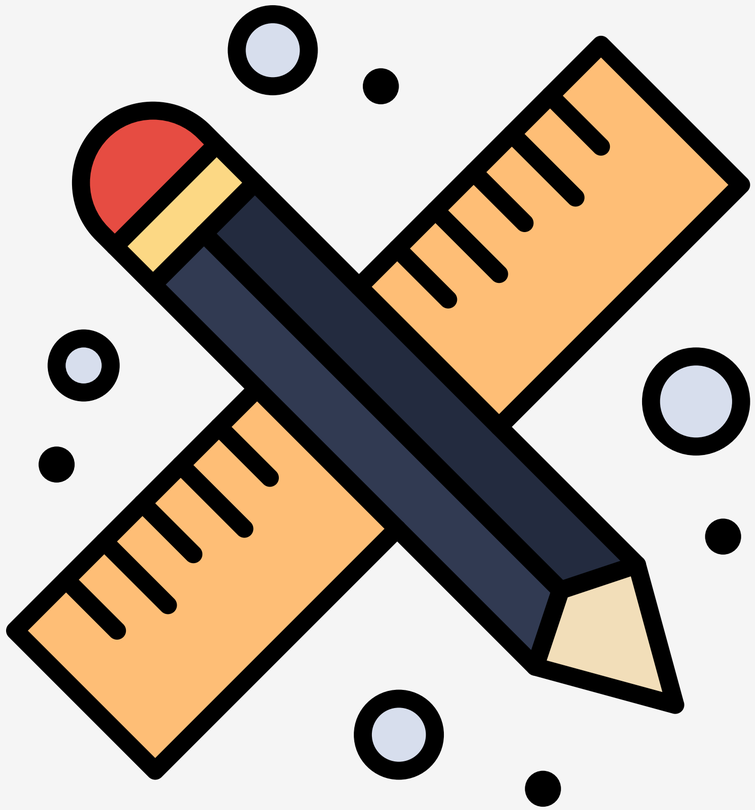
- Numeric functions are used to alter the returned number data OR return the result of a calculation
- Numeric functions include:
  - RAND()
  - ROUND()
  - DEGREES()
  - POWER()

MOVIE MAYHEM II



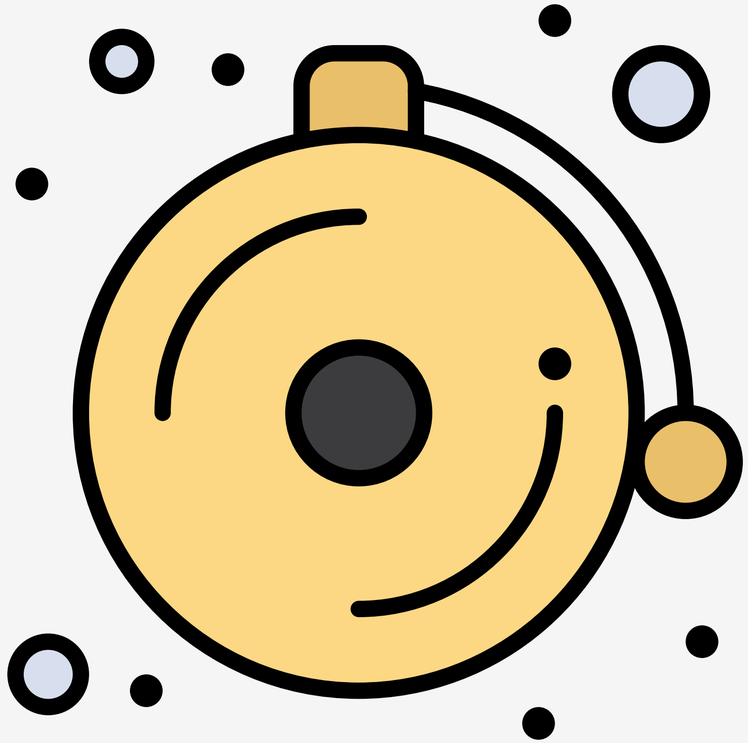
- *GITHUB CLASSROOM ASSIGNMENT*
- Create *ONE* query for each task
- Use phpMyAdmin to run queries
- Save queries to `queries.sql`
- *DUE:* **Wed. Jun. 24 @ 11:59 PM**

SEUSSOLOGY DB II



- *GITHUB CLASSROOM ASSIGNMENT*
- Import the Seussology DB
- Create *ONE* query for each task
- Use phpMyAdmin to run queries
- Save queries to `queries.sql`
- *DUE:* **Wed. Jul. 8 @ 11:59 PM**

NEXT TIME...



- PHP Data Objects
- **Participation:** Hybrid #3
- **Midterm:** Seussology