- **List** - An ordered collection of items
- **Dictionary** - A collection of key-value pairs
- **Set** - An unordered collection of unique items
- **Tuple** - An immutable ordered collection of items
- **Index** - The position of an item in an ordered collection
- **Key** - The lookup value in a dictionary
- **Value** - The data associated with a key in a dictionary
- **Method** - A function associated with an object
- **Append** - Add an item to the end of a list

- **Extend** - Add multiple items to the end of a list
- **For loop** - A control flow statement to iterate over a sequence
- **List comprehension** - A compact way to process and generate lists
- **Unpack** - Assigning multiple variables from a collection in one statement
- **Tuple** - An immutable ordered collection
- **Absolute path** - The full path to a file or directory
- **Append** - Add an item to the end of a list
- **Conditional statement** - Code that runs if a condition is true
- **Normalize** - Standardize data into a consistent format

- **JSON module** - Module to work with JSON data in Python
- **Load** - Read JSON data into a Python object
- **Dump** - Serialize a Python object to JSON
- **String** - Sequence of characters in Python
- **Serialize** - Convert a Python object to a format like JSON
- **Deserialize** - Convert JSON to a Python object
- **Read** - Get contents of a file as a string
- **Write** - Save data to a file

- **Module** - A Python file that can be imported
- **Dunder main** - Special variable main that indicates if module run directly
- **Entry point** - Main function that runs when executing a script
- **Import** - Load functions and classes from another module
- **Absolute path** - Full path to a file or directory from the root folder
- **Traverse** - Recursively visit nodes in a tree data structure
- **Directory** - A folder that can contain files and other directories
- **File path** - The location of a file in the file system
- **Join** - Concatenate directory and file path into full path
- **Process** - Perform operations on data

- **SQLite** - Embedded, serverless SQL database
- **Table** - Collection of related data in columns and rows
- **Query** - SQL statement to insert, retrieve, update, or delete data
- **Commit** - Make changes permanent in a database transaction
- **Cursor** - Object to execute queries and fetch results. It is like a file handle that we can use to perform operations on the data stored in the database. Calling cursor() is very similar conceptually to calling open() when dealing with text files.
- **Primary key** - Unique identifier for a row in a table
- **Generate** - Produce sample data programmatically

- **Persist** - Save data so it exists after program execution
- **Query** - Retrieve data from a database
- **Select** - Specify columns to return from a query
- **Where** - Filter rows returned by a query
- **Like** - Find values matching a pattern with wildcards
- **Wildcard** - Special characters % and _ to match patterns
- **Order by** - Sort results by a column
- **Distinct** - Remove duplicate rows from results
- **Column** - Field in a database table
- **Row** - Record in a database table
- **Process** - Perform operations on data

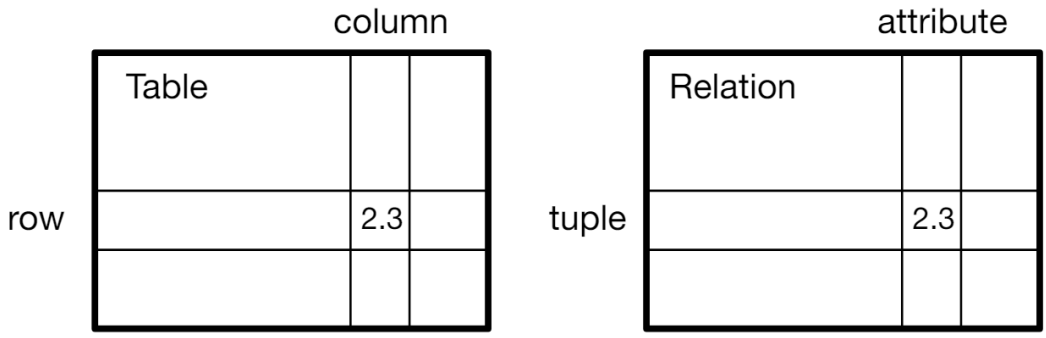
- **Web scraping**: The process of extracting data from websites. This is done when websites don't have APIs available.
- **HTML parsing**: Analyzing and extracting data from HTML code using tools like BeautifulSoup or Python's HTML parser.
- **APIs**: Application Programming Interfaces that allow programs to communicate with each other. Provides structured data like JSON.
- **Unstructured data**: Data that doesn't have a predefined format like HTML webpages. Difficult to analyze.
- **JSON**: JavaScript Object Notation - a common structured data format that is easy for programs to parse.
- **Normalization**: Converting unstructured or messy data into a structured, consistent format.
- **Persisting data**: Saving extracted data to files or databases for later use.
- **XML**: Extensible Markup Language - an older structured data format like JSON.
- **CSV**: Comma Separated Values - a simple text-based format for tabular data.
- **Data engineering**: Manipulating, cleaning, and transforming raw data into a more usable format.

- **Web scraping**: Extracting data from websites, usually when an API is not available.
- **HTML parsing**: Analyzing and extracting data from HTML code.
- **BeautifulSoup**: A Python library for parsing HTML and extracting data.

- **CSS selectors**: Patterns used to select HTML elements for scraping.
- **Scrapy**: A Python web scraping framework.
- **XPath**: A query language for selecting elements in XML documents.
- **SQLite**: A lightweight relational database that stores data in .db files.
- **Scraping challenges**: Issues like changing webpages, scaling, and efficiency.
- **Downloading HTML**: Saving webpages locally to improve scraping speed and stability.
- **Coding defensively**: Writing scraping code that is robust to changes in webpage structure.
- **Testing locally**: Using downloaded HTML files to test scrapers offline.

- **MySQL** - An open-source relational database management system. It is used to store and manage data.
- **Database** - An organized collection of data stored and accessed electronically.
- **Table** - A set of data elements organized in rows and columns in a database.
- **Column** - A set of data values of a particular type in a table.
- **Query** - A request for data from a database. Used to retrieve, insert, update, or delete data.
- **SSL** (Secure Sockets Layer) - A standard security technology used to encrypt connections between a client and server over the internet. Provides secure communication.
- **Root user** - The most privileged user account in MySQL. Has full access to make changes to the database.
- **Text editor** - A software used to edit text files like code, markup, etc. Examples: VS Code, Sublime Text, Atom.
- **VS Code** - Visual Studio Code. A free source-code editor made by Microsoft with support for debugging, syntax highlighting, intelligent code completion, snippets, etc.
- **Extension** - Add-on programs that extend the capabilities of VS Code editor like adding support for other programming languages, tools, etc.
- **Command Palette** - Used to quickly execute commands in VS Code editor via keyboard shortcut rather than going through menus.

In technical descriptions of relational databases the concepts of table, row, and column are more formally referred to as relation, tuple, and attribute, respectively. We will use the less formal terms in this chapter.



When we create a database table we must tell the database in advance the names of each of the columns in the table and the type of data which we are planning to store in each column.

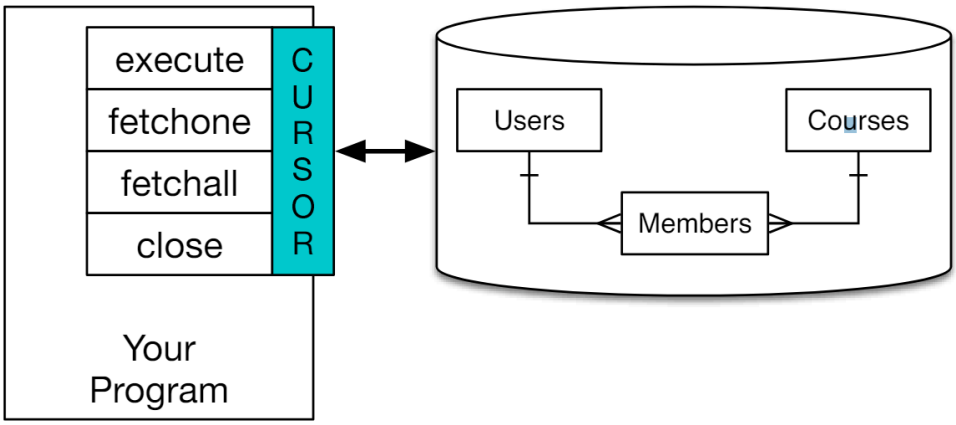


Figure 15.2: A Database Cursor

While **database normalization** seems very complex on the surface and contains a lot of mathematical justifications, for now we can reduce it all into one simple rule that we will follow. We should never put the same string data in a column more than once. If we need the data more than once, we create a numeric key for the data and reference the actual data using this key. Especially if the multiple entries refer to the same object.

BASIC COMMADS – (INSERT, SELECT, UPDATE, and DELETE):

1. CREATE TABLE Track (title TEXT, plays INTEGER)
2. INSERT INTO Track (title, plays) VALUES ('My Way', 15)
3. SELECT * FROM Track WHERE title = 'My Way':
    a. The SELECT statement lets you specify which columns you would like to retrieve.
    b. The WHERE clause is used to select which rows you would like to see.
    c. It also allows an optional ORDER BY clause to control the sorting of the returned rows.
    d. Using * indicates that you want the database to return all the columns for each row that matches the WHERE clause.
    e. LIMIT clause can be use to limit the output records
4. UPDATE Track SET plays = 16 WHERE title = 'My Way':
    a. The UPDATE statement specifies a table and then a list of fields and values to change after the SET keyword.
    b. An optional WHERE clause to select the rows that are to be updated.
    c. If a WHERE clause is not specified, it performs the UPDATE on all the rows in the table.
5. DELETE FROM Track WHERE title = 'My Way'

Particular (and strict) order of SQL statements:

- SELECT
- FROM
- WHERE
- GROUP BY
- HAVING
- ORDER BY

Scrapy:

1. pip install scrapy
2. scrapy startproject <project name>
3. scrapy genspider <spider name> <link>
4. scrapy shell <url>
5. To run the crawler, go into the scrapy project and the further into the directory holding the scraper: scrapy crawl <spider name>

Scrapy Shell:

1. scrapy shell <url> … Open the website
2. response
3. response.xpath('//table')
4. response.xpath('//table')[0] …. So on and so forth until you find the right one
5. len(response.xpath('//table')[0] .xpath('tr')) … Checking the length of first table row
6. table = response.xpath('//table')[3] … Selecting the exact table you want to with
7. child = table.xpath('//tr')[10] … 10th row of out selected table
8. child.xpath('td//text()')[0].extract() … Extract the first item or data piece of our selected row
9. for row in table.xpath('//tr'):
   try:
   print(row.xpath('td//text()')[1].extract())
   except IndexError:
   pass
10. CRTL + D to exit shell
11. To run the crawler go into the scrapy project and the further into the directory holding the scraper: scrapy crawl <spider name>

Shifting Databases:

Instructions

Exporting to CSV

There are several ways to export data. You can use the power of SQL to create a new script and export the fields and data you need into a CSV file.

Find the *export_csv.sql* file in the *sql_script* directory and execute it against the database.

What happens if you run it more than once?

Try changing the script to create a CSV file with only the names and ratings instead of including the region.

Exporting as SQL

Different databases have their own ways of exporting data. In MySQL databases you can use the `mysqldump` utility to export data. Open up a terminal and run the following command:

```
$ mysqldump -u root -p ratings > vscode-mysql/export.sql
```

Remember that the coursera database doesn't have a password for the `root` user. In a real-life scenario, you would have to type a password and perhaps use a specific user instead of using `root`.

Inspect the *export.sql* file and check its contents. You can use this same file to load data elsewhere. This operation is useful if you are migrating to and from databases or to create a backup.

Note that the SQL file also has instructions on how to re-create the table and database itself, not only the actual data.

SQL Commands

1. `mysql -u root -p`: For logging into the terminal as user 'root'.
2. `ALTER USER 'root'@'localhost' IDENTIFIED BY 'dbpassword'`: For changing the password of username root to 'dbpassword'.
3. `CREATE DATABASE mydemo`: Making a database called mydemo.
4. `USE mydemo`: Selecting the database mydemo for usage.
5. `CREATE TABLE users ( id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(50), email VARCHAR(50) );`

   a. This command creates a table named user with three columns. Name and email can take up to 50 characters per entry. Id generates a unique identifier using a count increment method.
6. `INSERT INTO users (name, email) VALUES ('Jane', 'example.org')`

   a. This command inserts two example values into the users table.
7. `SELECT * FROM users`: Displays everything in the users table.
8. `mysqldump -u root -p mydemo > backup_mydemo.sql`: To create a backup of the mydemo database.
9. `head -n 10 backup_mydemo.sql`: Display the first 10 lines of the backup_mydemo.sql
10. `show databases;`: Shows all created databases.
11. `SHOW FULL TABLES`: Shows all tables with the currently selected database.
12. `CREATE TABLE users2 AS SELECT * FROM mydemo.users;`: Populates the users2 table using the users table located in the mydemo database.
13. `DROP TABLE users2;`: Deletes the users2 table.
14. `DROP DATABASE mydemo2;`: Deletes the database mydemo2.
15. `du -sh *`: Shows the human-readable size of each file.
16. `SOURCE <path to schema.sql>`: Creates database structure laid out in the schema.sql file.
17. `DESCRIBE <table name>`: Displays the contents of the schema for a table. (Note: `DESCRIBE` is commonly used instead of `DESC`)
18. `SELECT COUNT(*) FROM <table name>`: Shows the number of rows in the table.

19.
```
SELECT f.title,
        l.name FROM film f
        JOIN language l ON f.langauge = l.language_id LIMIT 5
```
This query retrieves the title and name columns from the film (f) and language (l) tables, joining them based on the language_id and limiting the results to the first 5 rows.

20.
```
SELECT TABLE directors_notes (
        note_id INT AUTO_INCREMENT PRIMARY KEY,
        film_id INT,
        director_note VARCHAR (255)
        edit_date DATE
        );
```
This code creates a table named "directors_notes" to store details about film director's notes, including a unique ID, linked film (likely by ID), the note itself, and the date it was last edited.

21. `SELECT * FROM actor INTO OUTFILE '<path to txt or csv>'` This command saves data from the table actors into a file located on the disk

22. `Cat actors.txt | grep 'PENN' > penn_actors.txt` This command looks for specific actor names and save them to another file.