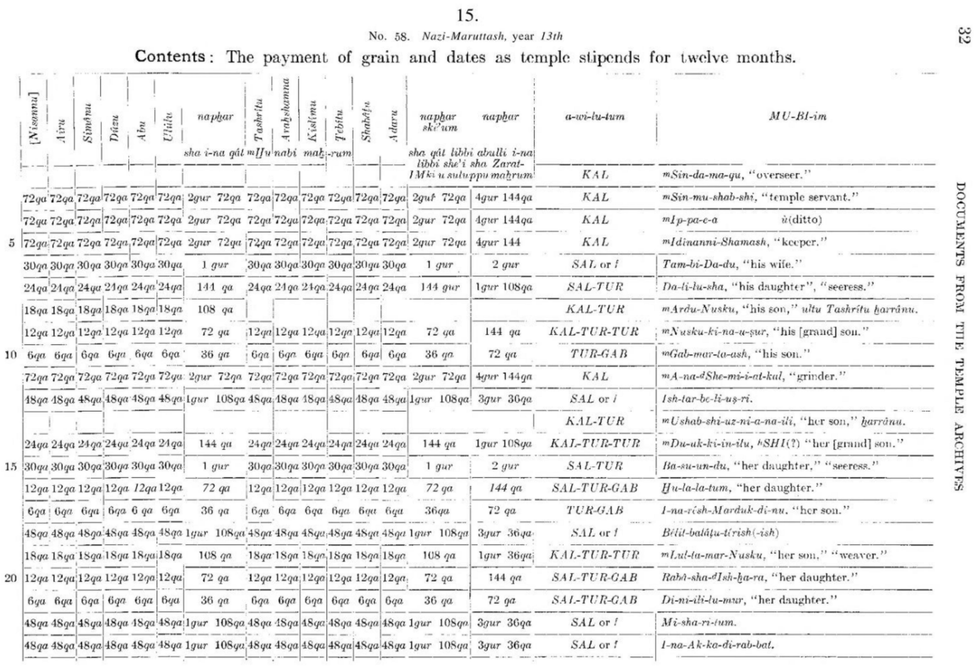
CS50's Introduction to Databases with SQL

Lecture 0

Introduction

- Databases (and SQL) are tools that can be used to interact with, store, and manage information. Although the tools we're using in this course are new, a database is an age-old idea.
- Look at this diagram from a few thousand years ago. It has rows and columns, and seems to contain stipends for workers at a temple. One could call this diagram a table, or even a spreadsheet.



15.

No. 58. *Nazi-Maruttash, year 13th*

Contents: The payment of grain and dates as temple stipends for twelve months.

[Nisannu]	Airu	Simânu	Dûzu	Abu	Ulûlu	naphar	Tashrîtu	Arahshamna	Kislimu	Tebitu	Shabâṭu	Adaru	naphar ske'um	naphar	a-wi-lu-tum	MU-BI-im
						sha i-na qât	mIlu	nabi	maḫ-	-rum			sha qât libbi libbi she'i IMki u	abulli i-na sha Zarat- suluppu maḫrum		
															KAL	mSin-da-ma-qu, "overseer."
72qa	72qa	72qa	72qa	72qa	72qa	2gur 72qa	72qa	72qa	72qa	72qa	72qa	72qa	2gur 72qa	4gur 144qa	KAL	mSin-mu-shab-shi, "temple servant."
72qa	72qa	72qa	72qa	72qa	72qa	2gur 72qa	72qa	72qa	72qa	72qa	72qa	72qa	2gur 72qa	4gur 144qa	KAL	mIp-pa-e-a ù(ditto)
72qa	72qa	72qa	72qa	72qa	72qa	2gur 72qa	72qa	72qa	72qa	72qa	72qa	72qa	2gur 72qa	4gur 144	KAL	mIdinanni-Shamash, "keeper."
30qa	30qa	30qa	30qa	30qa	30qa	1 gur	30qa	30qa	30qa	30qa	30qa	30qa	1 gur	2 gur	SAL or f	Tam-bi-Da-du, "his wife."
24qa	24qa	24qa	24qa	24qa	24qa	144 qa	24qa	24qa	24qa	24qa	24qa	24qa	144 gur	1gur 108qa	SAL-TUR	Da-li-lu-sha, "his daughter", "seeress."
18qa	18qa	18qa	18qa	18qa	18qa	108 qa									KAL-TUR	mArdu-Nusku, "his son," ultu Tashrîtu ḫarrânu.
12qa	12qa	12qa	12qa	12qa	12qa	72 qa	12qa	12qa	12qa	12qa	12qa	12qa	72 qa	144 qa	KAL-TUR-TUR	mNusku-ki-na-u-ṣur, "his [grand] son."
6qa	6qa	6qa	6qa	6qa	6qa	36 qa	6qa	6qa	6qa	6qa	6qa	6qa	36 qa	72 qa	TUR-GAB	mGab-mar-ta-ash, "his son."
72qa	72qa	72qa	72qa	72qa	72qa	2gur 72qa	72qa	72qa	72qa	72qa	72qa	72qa	2gur 72qa	4gur 144qa	KAL	mA-na-dShe-mi-i-at-kal, "grinder."
48qa	48qa	48qa	48qa	48qa	48qa	1gur 108qa	48qa	48qa	48qa	48qa	48qa	48qa	1gur 108qa	3gur 36qa	SAL or f	Ish-tar-be-li-uṣ-ri.
															KAL-TUR	mUshab-shi-uz-ni-a-na-ili, "her son," ḫarrânu.
24qa	24qa	24qa	24qa	24qa	24qa	144 qa	24qa	24qa	24qa	24qa	24qa	24qa	144 qa	1gur 108qa	KAL-TUR-TUR	mDu-uk-ki-in-ilu, hSHI(?) "her [grand] son."
30qa	30qa	30qa	30qa	30qa	30qa	1 gur	30qa	30qa	30qa	30qa	30qa	30qa	1 gur	2 gur	SAL-TUR	Ba-su-un-du, "her daughter," "seeress."
12qa	12qa	12qa	12qa	12qa	12qa	72 qa	12qa	12qa	12qa	12qa	12qa	12qa	72 qa	144 qa	SAL-TUR-GAB	Ḫu-la-la-tum, "her daughter."
6qa	6qa	6qa	6qa	6 qa	6qa	36 qa	6qa	6qa	6qa	6qa	6qa	6qa	36qa	72 qa	TUR-GAB	I-na-rêsh-Marduk-di-nu, "her son."
48qa	48qa	48qa	48qa	48qa	48qa	1gur 108qa	48qa	48qa	48qa	48qa	48qa	48qa	1gur 108qa	3gur 36qa	SAL or f	Bêlit-balâṭu-tîrish(-ish)
18qa	18qa	18qa	18qa	18qa	18qa	108 qa	18qa	18qa	18qa	18qa	18qa	18qa	108 qa	1gur 36qa	KAL-TUR-TUR	mLul-ta-mar-Nusku, "her son," "weaver."
12qa	12qa	12qa	12qa	12qa	12qa	72 qa	12qa	12qa	12qa	12qa	12qa	12qa	72 qa	144 qa	SAL-TUR-GAB	Rabâ-sha-dIsh-ḫa-ra, "her daughter."
6qa	6qa	6qa	6qa	6qa	6qa	36 qa	6qa	6qa	6qa	6qa	6qa	6qa	36 qa	72 qa	SAL-TUR-GAB	Di-ni-ili-lu-mur, "her daughter."
48qa	48qa	48qa	48qa	48qa	48qa	1gur 108qa	48qa	48qa	48qa	48qa	48qa	48qa	1gur 108qa	3gur 36qa	SAL or f	Mi-sha-ri-tum.
48qa	48qa	48qa	48qa	48qa	48qa	1gur 108qa	48qa	48qa	48qa	48qa	48qa	48qa	1gur 108qa	3gur 36qa	SAL or f	I-na-Ak-ka-di-rab-bat.

32 DOCUMENTS FROM THE TEMPLE ARCHIVES

- Based on what we see in the diagram above, we can conclude that:
  - A table stores some set of information (here, worker stipends).
  - Every row in a table stores one item in that set (here, one worker).
  - Every column has some attribute of that item (here, the stipend for a particular month).

- Let us now consider a modern context. Say you are a librarian tasked with organizing information about the book titles and authors in this diagram.

- One way of organizing the information would be to have each book title followed by its author, as below.

title	author
Song of Solomon	Toni Morrison
Goodnight Moon	Margaret Wise Brown
The Overstory	Richard Powers
The Great Gatsby	F. Scott Fitzgerald
Frankenstein	Mary Shelley
Le Petit Prince	Antoine de Saint-Exupéry
Prince of Tides	Pat Conroy

    - Notice that each book is now a row in this table.
    - Every row has two columns — each a different attribute of the book (book title and author).
- In today's information age, we can store our tables using software like Google Sheets instead of paper or stone tablets. However, in this course we will talk about databases and not spreadsheets.

- Three reasons to move beyond spreadsheets to databases are
    - **Scale**: Databases can store not just items numbering to tens of thousands but even millions and billions.
    - **Update Capacity**: Databases are able to handle multiple updates of data in a second.
    - **Speed**: Databases allow faster look-up of information. This is because databases provide us with access to different algorithms to retrieve information. In contrast, spreadsheets that merely allow the use of Ctrl+F or Cmd+F to go through hits one at a time.

What is a Database?

- A database is a way of organizing data such that you can perform four operations on it
  - create
  - read
  - update
  - delete
- A database management system (DBMS) is a way to interact with a database using a graphical interface or textual language.
- Examples of DBMS: MySQL, Oracle, PostgreSQL, SQLite, Microsoft Access, MongoDB etc.
- The choice of a DBMS would rest on factors like
  - **Cost**: proprietary vs. free software,
  - **Amount of support**: free and open source software like MySQL, PostgreSQL and SQLite come with the downside of having to set up the database yourself,
  - **Weight**: more fully-featured systems like MySQL or PostgreSQL are heavier and require more computation to run than systems like SQLite.
- In this course, we will start with SQLite and then move on to MySQL and PostgreSQL.

SQL

- SQL stands for Structured Query Language. It is a language used to interact with databases, via which you can create, read, update, and delete data in a database. Some important notes about SQL
  - it is structured, as we'll see in this course,
  - it has some keywords that can be used to interact with the database, and
  - it is a query language — it can be used to ask questions of data inside a database.
- In this lesson, we will learn how to write some simple SQL queries.

Questions

> Are there subsets of SQL?
>
> SQL is a standard both of the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). Most DBMS support some subset of the SQL language. So for SQLite, for example, we're using a subset of SQL that is supported by SQLite. If we wanted to port our code to a different system like MySQL, it is likely we would have to change some of the syntax.

Getting Started with SQLite

- It is worth noting that SQLite is not merely something we use for this class, but a database used in plenty of other applications including phones, desktop applications and websites.

- Now, consider a database of books that have been longlisted for the International Booker Prize. Each year, there are 13 books on the longlist and our database contains 5 years' worth of such longlists.
- Before we begin interacting with this database:
    - Log in to Visual Studio Code for CS50. This is where we will write code and edit files.
    - The SQLite environment is already set up in your Codespace! Open it up on the terminal.

Terminal Tips

Here are some useful tips for writing SQL code on the terminal.

- To clear the terminal screen, hit Ctrl + L.
- To get the previously executed instruction(s) on the terminal, press the Up Arrow key.
- If your SQL query is too long and wrapping around the terminal, you can hit enter and continue writing the query on the next line.
- To exit a database or the SQLite environment, use `.quit`.

SELECT

- What data is actually in our database? To answer this, we will use our first SQL keyword, `SELECT`, which allows us to select some (or all) rows from a table inside the database.
- In the SQLite environment, run

```
SELECT *
FROM "longlist";
```

  This selects all the rows from the table called `longlist`.

- The output we get contains all the columns of all the rows in this table, which is a lot of data. We can simplify it by selecting a particular column, say the title, from the table. Let's try

```
SELECT "title"
FROM "longlist";
```

- Now, we see a list of the titles in this table. But what if we want to see titles and authors in our search results? For this, we run

```
SELECT "title", "author"
FROM longlist;
```

Questions

> Is it necessary to use the double quotes (") around table and column names?

It is good practice to use double quotes around table and column names, which are called SQL identifiers. SQL also has strings and we use single quotes around strings to differentiate them from identifiers.

> Where is the data in this database coming from?

- This database contains data from various sources.
- Longlists of books (years 2018—2023) come from the Booker Prize website.
- Ratings and other information about these books come from Goodreads.

> How do we know what tables and columns are in a database?

The database schema contains the structure of the database, including table and column names. Later in this course, we will learn how to get the database schema and understand it.

> Is SQLite 3 case-sensitive? Why are some parts of the query in capital letters and some in small letters?

SQLite is case-insensitive. However, we do follow some style conventions. Observe this query:

```
SELECT *
FROM "longlist";
```

SQL keywords are written in capital letters. This is especially useful in improving the readability of longer queries. Table and column names are in lowercase.

LIMIT

- If a database had millions of rows, it might not make sense to select all of its rows. Instead, we might want to merely take a peek at the data it contains. We use the SQL keyword `LIMIT` to specify the number of rows in the query output.
- ```
  SELECT "title"
  FROM "longlist"
  LIMIT 10;
  ```

  This query gives us the first 10 titles in the database. The titles are ordered the same way in the output of this query as they are in the database.

WHERE

- The keyword `WHERE` is used to select rows based on a condition; it will output the rows for which the specified condition is true.
  ```

- ```
  SELECT "title", "author"
  FROM "longlist"
  WHERE "year" = 2023;
  ```

  This gives us the titles and authors for the books longlisted in 2023. Note that `2023` is not in quotes because it is an integer, not a string or identifier.

- The operators that can be used to specify conditions in SQL are `=` ("equal to"), `!=` ("not equal to") and `<>` (also "not equal to").

- To select the books that are not hardcovers, we can run the query

  ```
  SELECT "title", "format"
  FROM "longlist"
  WHERE "format" != 'hardcover';
  ```

  Note that `hardcover` is in single quotes because it is an SQL string and not an identifier.

- `!=` can be replaced with the operator `<>` to get the same results. The modified query would be

  ```
  SELECT "title", "format"
  FROM "longlist"
  WHERE "format" <> 'hardcover';
  ```

- Yet another way to get the same results is to use the SQL keyword `NOT`. The modified query would be

  ```
  SELECT "title", "format"
  FROM "longlist"
  WHERE NOT "format" = 'hardcover';
  ```

- To combine conditions, we can use the SQL keywords `AND` and `OR`. We can also use parentheses to indicate how to combine the conditions in a compound conditional statement.
- To select the titles and authors of the books longlisted in 2022 or 2023

  ```
  SELECT "title", "author"
  FROM "longlist"
  WHERE "year" = 2022 OR "year" = 2023;
  ```

- To select the books longlisted in 2022 or 2023 that were **not** hardcovers

  ```
  SELECT "title", "format"
  FROM "longlist"
  WHERE ("year" = 2022 OR "year" = 2023) AND "format" != 'hardcover';
  ```

  Here, the parantheses indicate that the `OR` clause should be evaluated before the `AND` clause.

NULL
  ```

- It is possible that tables may have missing data. `NULL` is a type used to indicate that certain data does not have a value, or does not exist in the table.
- For example, the books in our database have a translator along with an author. However, only some of the books have been translated to English. For other books, the translator value will be `NULL`.
- Conditions used with `NULL` are `IS NULL` and `IS NOT NULL`.
- To select the books for which translators don't exist, we can run

```
SELECT "title", "translator"
FROM "longlist"
 WHERE "translator" IS NULL;
```

- Let's try the reverse: selecting the books for which translators do exist.

```
SELECT "title", "translator"
FROM "longlist"
WHERE "translator" IS NOT NULL;
```

LIKE

- This keyword is used to select data that roughly matches the specified string. For example, `LIKE` could be used to select books that have a certain word or phrase in their title.
- `LIKE` is combined with the operators `%` (matches any characters around a given string) and `_` (matches a single character).
- To select the books with the word "love" in their titles, we can run

```
SELECT "title"
FROM "longlist"
WHERE "title" LIKE '%love%';
```

  `%` matches 0 or more characters, so this query would match book titles that have 0 or more characters before and after "love" — that is, titles that contain "love".

- To select the books whose title begin with "The", we can run

```
SELECT "title"
FROM "longlist"
WHERE "title" LIKE 'The%';
```

- The above query may also return books whose titles begin with "Their" or "They". To select only the books whose titles begin with the **word** "The", we can add a space.

```
SELECT "title"
FROM "longlist"
WHERE "title" LIKE 'The %';
```

- Given that there is a book in the table whose name is either “Pyre” or “Pire”, we can select it by running

```
SELECT "title"
FROM "longlist"
WHERE "title" LIKE 'P_re';
```

This query could also return book titles like “Pore” or “Pure” if they existed in our database, because `_` matches any single character.

Questions

> Can we use multiple `%` or `_` symbols in a query?

- Yes, we can! Example 1: If we wanted to select books whose titles begin with “The” and have “love” somewhere in the middle, we could run

```
SELECT "title"
FROM "longlist"
WHERE "title" LIKE 'The%love%';
```

- Note: No book from our current database matches this pattern, so this query returns nothing.
- Example 2: If we knew there was a book in the table whose title begins with “T” and has four letters in it, we can try to find it by running

```
SELECT "title"
FROM "longlist"
WHERE "title" LIKE 'T____';
```

> Is the comparison of strings case-sensitive in SQL?

In SQLite, comparison of strings with `LIKE` is by default case-*in*sensitive, whereas comparison of strings with `=` is case-sensitive. (Note that, in other DBMS’s, the configuration of your database can change this!)

Ranges

- We can also use the operators `<`, `>`, `<=` and `>=` in our conditions to match a range of values. For example, to select all the books longlisted between the years 2019 and 2022 (inclusive), we can run

```
SELECT "title", "author"
FROM "longlist"
WHERE "year" >= 2019 AND "year" <= 2022;
```

- Another way to get the same results is using the keywords `BETWEEN` and `AND` to specify inclusive ranges. We can run

```
SELECT "title", "author"
FROM "longlist"
WHERE "year" BETWEEN 2019 AND 2022;
```

- To select the books that have a rating of 4.0 or higher, we can run

```
SELECT "title", "rating"
FROM "longlist"
WHERE "rating" > 4.0;
```

- To further limit the selected books by number of votes, and have only those books with at least 10,000 votes, we can run

```
SELECT "title", "rating", "votes"
FROM "longlist"
WHERE "rating" > 4.0 AND "votes" > 10000;
```

- To select the books that have less than 300 pages, we can run

```
SELECT "title", "pages"
FROM "longlist"
WHERE "pages" < 300;
```

Questions

> For range operators like `<` and `>`, do the values in the database have to be integers?
>
> No, the values can be integers or floating-point (i.e., “decimal” or “real”) numbers. While creating a database, there are ways to set these data types for columns.

ORDER BY

- The `ORDER BY` keyword allows us to organize the returned rows in some specified order.
- The following query selects the bottom 10 books in our database by rating.

```
SELECT "title", "rating"
FROM "longlist"
ORDER BY "rating" LIMIT 10;
```

- Note that we get the bottom 10 books because `ORDER BY` chooses ascending order by default.

- Instead, to select the top 10 books

```
SELECT "title", "rating"
FROM "longlist"
ORDER BY "rating" DESC LIMIT 10;
```

  Note the use of the SQL keyword `DESC` to specify the descending order. `ASC` can be used to explicitly specify ascending order.

- To select the top 10 books by rating and also include number of votes as a tie-break, we can run

```
SELECT "title", "rating", "votes"
FROM "longlist"
ORDER BY "rating" DESC, "votes" DESC
LIMIT 10;
```

  Note that for each column in the `ORDER BY` clause, we specify ascending or descending order.

Questions

> To sort books by title alphabetically, can we use `ORDER BY`?

Yes, we can. The query would be

```
SELECT "title"
FROM "longlist"
ORDER BY "title";
```

Aggregate Functions

- `COUNT`, `AVG`, `MIN`, `MAX`, and `SUM` are called aggregate functions and allow us to perform the corresponding operations over multiple rows of data. By their very nature, each of the following aggregate functions will return only a single output—the aggregated value.
- To find the average rating of all books in the database

```
SELECT AVG("rating")
FROM "longlist";
```

- To round the average rating to 2 decimal points

```
SELECT ROUND(AVG("rating"), 2)
FROM "longlist";
```

- To rename the column in which the results are displayed

```
SELECT ROUND(AVG("rating"), 2) AS "average rating"
FROM "longlist";
```

  Note the use of the SQL keyword `AS` to rename columns.

- To select the maximum rating in the database

```
SELECT MAX("rating")
FROM "longlist";
```

- To select the minimum rating in the database

```
SELECT MIN("rating")
FROM "longlist";
```

- To count the total number of votes in the database

```
SELECT SUM("votes")
FROM "longlist";
```

- To count the number of books in our database

```
SELECT COUNT(*)
FROM "longlist";
```

    Remember that we used `*` to select every row and column from the database. In this case, we are trying to count every row in the database and hence we use the `*`.

- To count the number of translators

```
SELECT COUNT("translator")
FROM "longlist";
```

    We observe that the number of translators is fewer than the number of rows in the database. This is because the `COUNT` function does not count `NULL` values.

- To count the number of publishers in the database

```
SELECT COUNT("publisher")
FROM "longlist";
```

- As with translators, this query will count the number of publisher values that are not `NULL`. However, this may include duplicates. Another SQL keyword, `DISTINCT`, can be used to ensure that only distinct values are counted.

```
SELECT COUNT(DISTINCT "publisher")
FROM "longlist";
```

Questions

> Would using `MAX` with the title column give you the longest book title?

No, using `MAX` with the title column would give you the “largest” (or in this case, last) title alphabetically. Similarly, `MIN` will give the first title alphabetically.

Fin

- This brings us to the conclusion of Lecture 0 about Querying in SQL! To exit the SQLite prompt, you can type in the SQLite keyword `.quit` and this should take you back to the regular terminal.
- Until next time!