CS50 Section

Week 7

Attendance Sign In: tinyurl.com/sqlcs50rw

Agenda

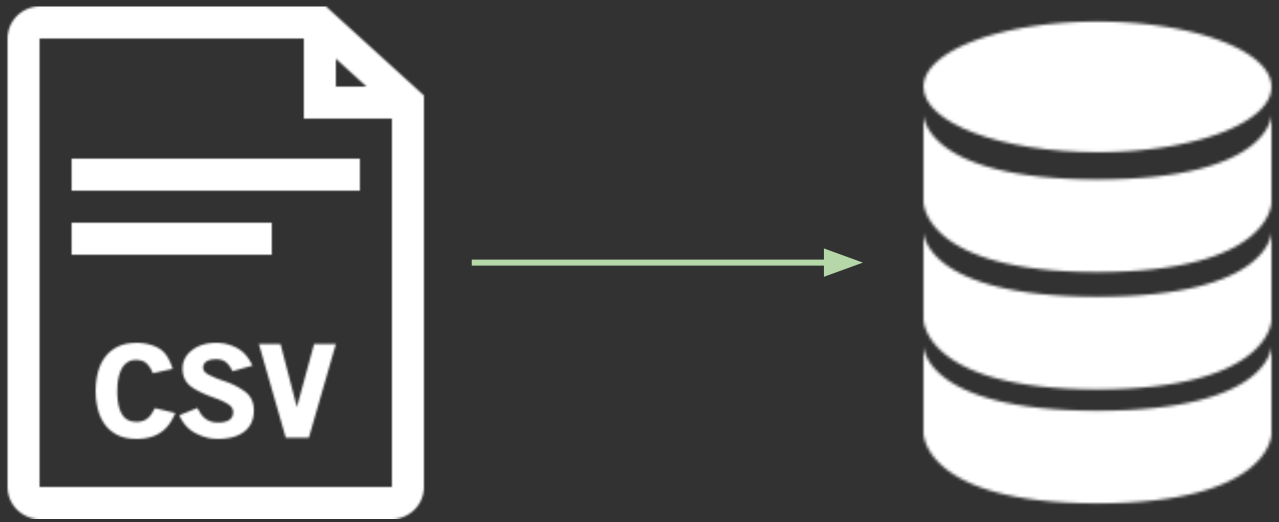
- Week 6: Python - Remaining Questions
- Week 7: SQL

Week 6 Review - What Questions do You Have?

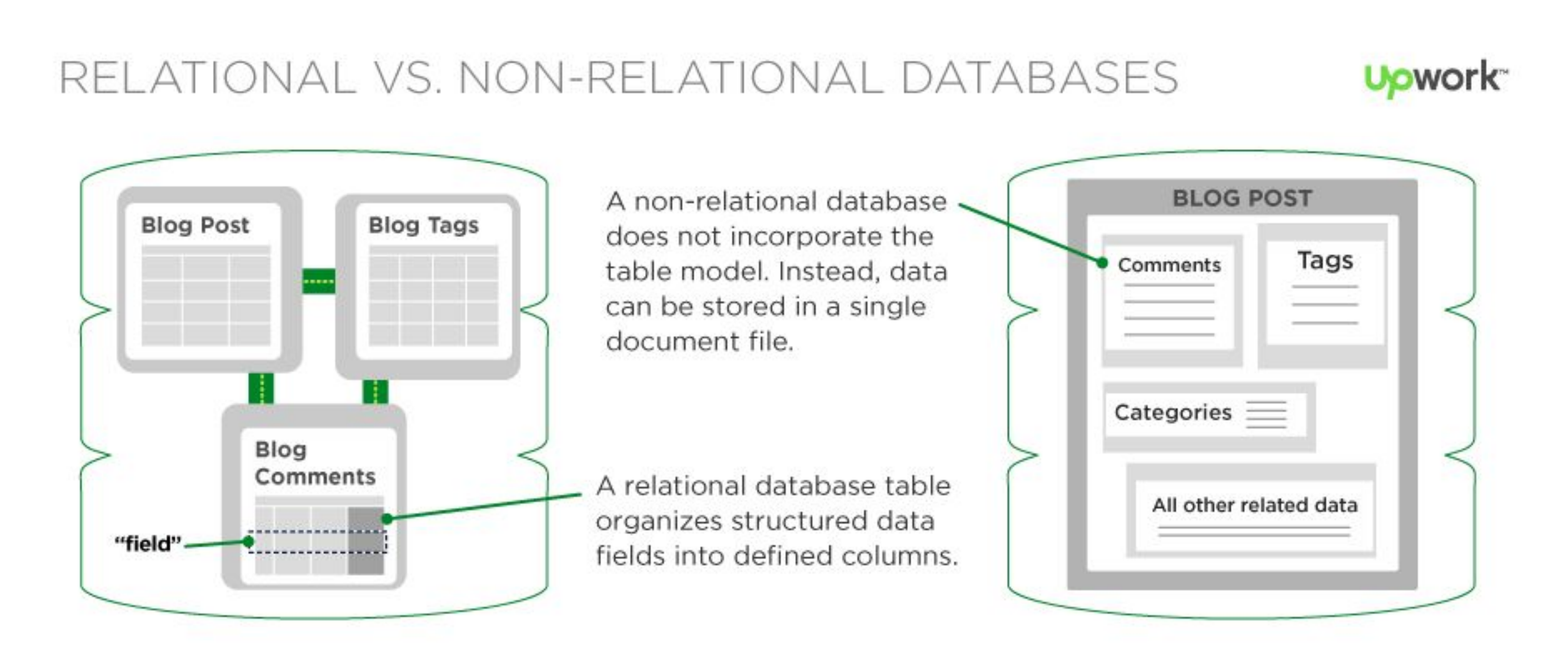
- For Loops
  - range()
  - len()
- While Loops
- Tuples: ()
- Lists: []
- Dictionaries: {}
- Strings
  - Substring Slicing: s[i:j]

Week 7

MOVING BEYOND THE CSV FILE



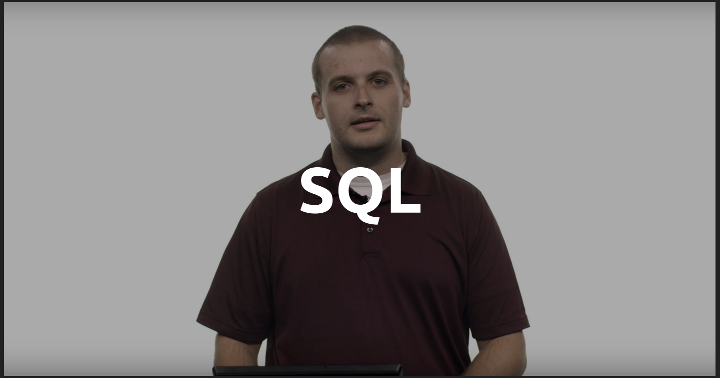
MOVING BEYOND THE CSV FILE



A non-relational database does not incorporate the table model. Instead, data can be stored in a single document file.

A relational database table organizes structured data fields into defined columns.

SQL



https://youtu.be/nfGiGSCEYRI

- As a relational DB, SQL is centered around **tables**.
- For each table, you specify all of the **columns** in the table.
- When new information is added to the database, the new information (typically) goes into a new **row**.
- There are many data types that can be stored in a SQL database. This is just a small sample.

`INT`	`SMALLINT`	`TINYINT`	`MEDIUMINT`	`BIGINT`
`DECIMAL`	`FLOAT`	`BIT`	`DATE`	`TIME`
`DATETIME`	`TIMESTAMP`	`CHAR`	`VARCHAR`	`BINARY`
`BLOB`	`TEXT`	`ENUM`	`GEOMETRY`	`LINESTRING`

- As a relational DB, SQL is centered around **tables**.
- For each table, you specify all of the **columns** in the table.
- When new information is added to the database, the new information (typically) goes into a new **row**.
- In SQLite, which we'll use in this course, we can consolidate these various datatypes into a few more general classes (though underlying types still exist)

`NULL`	`INTEGER`	`REAL`	`TEXT`	`BLOB`

DATABASE DESIGN PRINCIPLES

- Why do we use types when designing our databases?

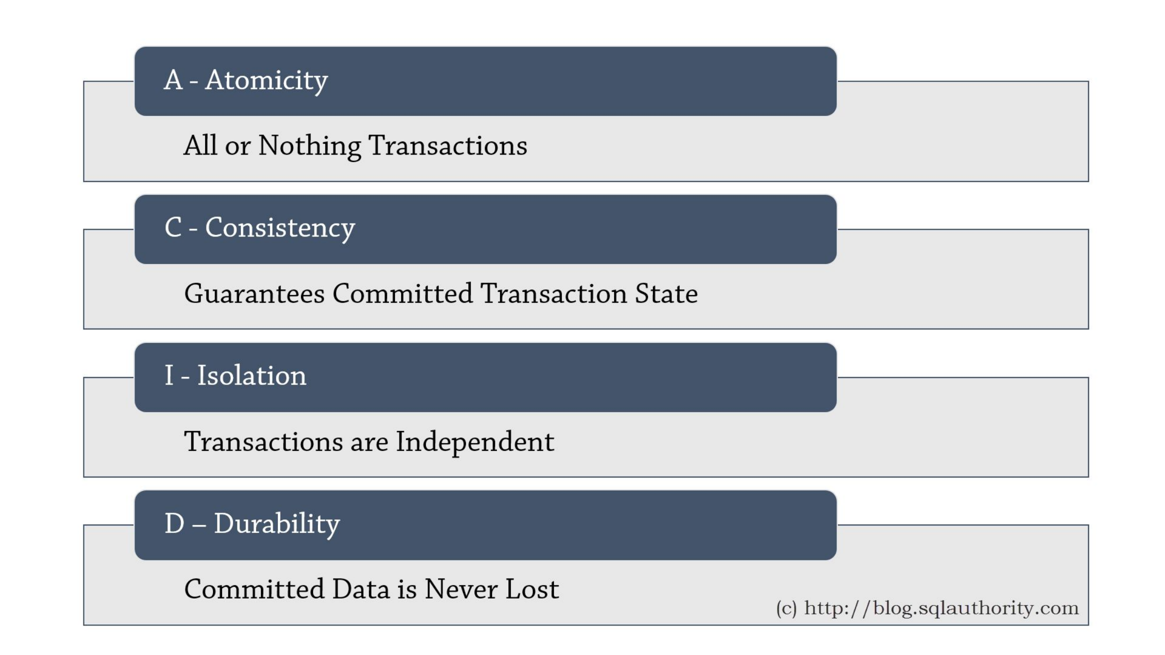
DATABASE DESIGN PRINCIPLES

- Why do we use types when designing our databases?

They allow us to increase the efficiency of data storage/retrieval AND they improve data integrity.

DATABASE DESIGN PRINCIPLES

- Why do we use types when designing our databases?



A - Atomicity

All or Nothing Transactions

C - Consistency

Guarantees Committed Transaction State

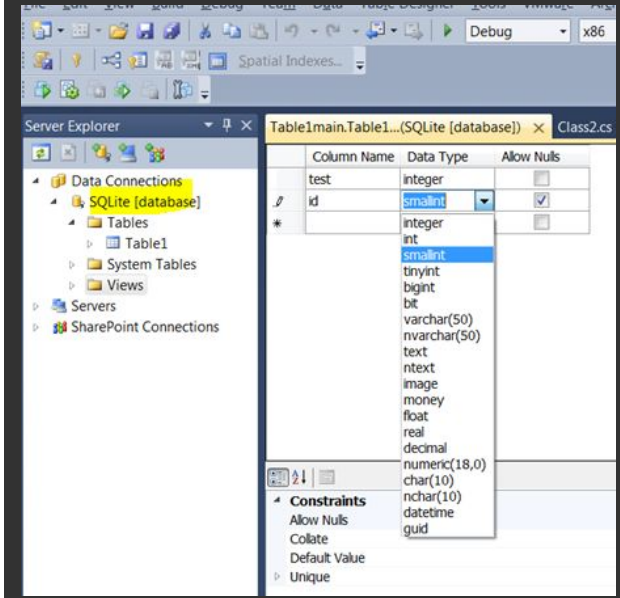
I - Isolation

Transactions are Independent

D – Durability

Committed Data is Never Lost

(c) http://blog.sqlauthority.com



Column Name	Data Type	Allow Nulls
test	integer	□
id	smallint	☑
		□

RELATIONAL DATABASE TERMINOLOGY

- **Database** - One discrete data system; The largest structure we’ll be working with

RELATIONAL DATABASE TERMINOLOGY

- **Database** - One discrete data system; The largest structure we’ll be working with
- **Table** - A set of data contained inside a database; Consists of columns and rows

RELATIONAL DATABASE TERMINOLOGY

- **Database** - One discrete data system; The largest structure we'll be working with
- **Table** - A set of data contained inside a database; Consists of columns and rows
- **Record** - One individual set of fields in the database (i.e. a single row)

RELATIONAL DATABASE TERMINOLOGY

- **Database** - One discrete data system; The largest structure we’ll be working with
- **Table** - A set of data contained inside a database; Consists of columns and rows
- **Record** - One individual set of fields in the database (i.e. a single row)
- **Key** - A field in a table used to retrieve records (*more later*)

RELATIONAL DATABASE TERMINOLOGY

- **Database** - One discrete data system; The largest structure we'll be working with
- **Table** - A set of data contained inside a database; Consists of columns and rows
- **Record** - One individual set of fields in the database (i.e. a single row)
- **Key** - A field in a table used to retrieve records (*more later*)
- **Schema** - A representation of the underlying structure behind the database; Serves as a template of the database

KEYS AND SQL

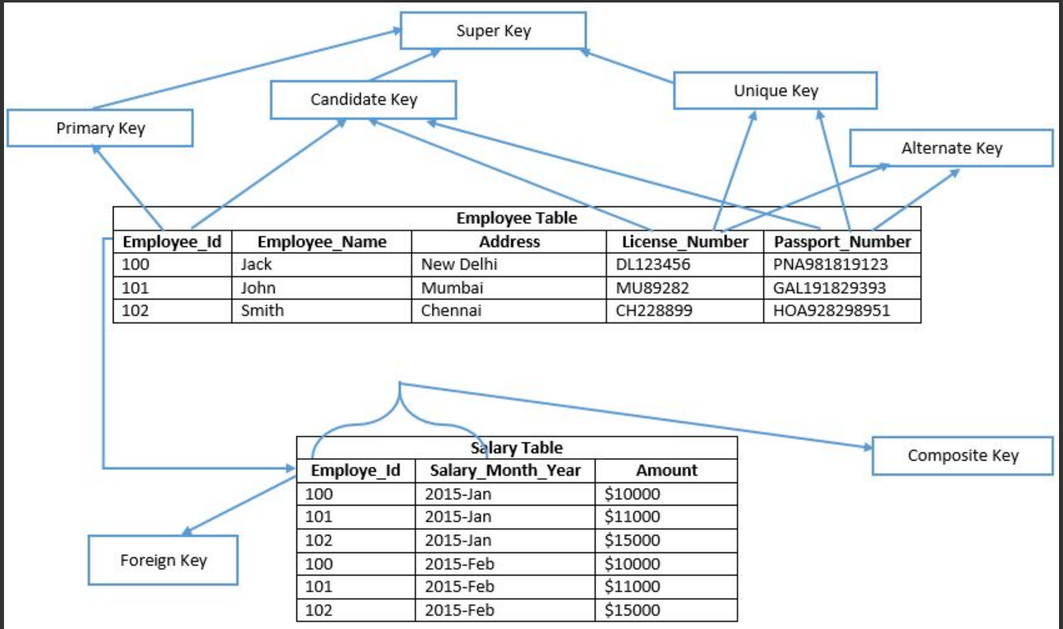
- Keys are just a way to retrieve records from our tables
- They should be unique *within* the table
  - Example: You have an **`employee`** table. What types of keys could you use to identify your records?

KEYS AND SQL

- Keys are just a way to retrieve records from our tables
- They should be unique *within* the table
  - Example: You have an `employee` table. What types of keys could you use to identify your records?
    - **An employee ID**
    - **Driver's license number**
    - **A full legal name (possibly—often causes conflicts!)**

KEYS AND SQL

- There are many different types of keys (we use **primary keys** in CS50):



Employee Table

Employee_Id	Employee_Name	Address	License_Number	Passport_Number
100	Jack	New Delhi	DL123456	PNA981819123
101	John	Mumbai	MU89282	GAL191829393
102	Smith	Chennai	CH228899	HOA928298951

Salary Table

Employe_Id	Salary_Month_Year	Amount
100	2015-Jan	$10000
101	2015-Jan	$11000
102	2015-Jan	$15000
100	2015-Feb	$10000
101	2015-Feb	$11000
102	2015-Feb	$15000

STRUCTURED QUERY LANGUAGE (SQL)

- We’ll be working with SQL using the **CRUD** paradigm:
  - **Create** new databases, tables, and records (INSERT)
  - **Read** existing records from the database (SELECT)
  - **Update** records and database schema (UPDATE)
  - **Delete** records or database schema (DELETE)

STRUCTURED QUERY LANGUAGE (SQL)

- Really get to know these commands for the Pset:
    - **INSERT**: Inserts values into a table
    - **SELECT**: Selects values from a table
    - **UPDATE**: Updates records and database schema
    - **DELETE**: Deletes records from a table

STRUCTURED QUERY LANGUAGE (SQL)

- SQL is the underlying language that standardizes how we run statements to implement the CRUD paradigm
  - There exists many different **database engines** that allow us to execute SQL
  - We utilize SQLite in CS50, but an extremely popular alternative is MySQL
- We also have access to a tool called **`DB Browser`** to give us a graphical user interface (GUI) for running different SQL commands

EXAMPLES: THE CONTEXT

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza

INSERT

- An `INSERT` query adds information to a table.

```
INSERT INTO
<table>
(<columns>)
VALUES
(<values>)
```

INSERT

- An `INSERT` query adds information to a table.

```
INSERT INTO
users
(username, password, fullname)
VALUES
('newman', 'USMAIL', 'Newman')
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza

INSERT

- When defining the column that ultimately is your primary key, it’s usually a good idea for that column to be an integer.

- Moreover, you can configure that column to **autoincrement**, so it will pre-populate that column for you automatically when rows are added, eliminating the risk that you’ll accidentally try to insert something with a duplicate value.

INSERT

- An INSERT query adds information to a table.

```
INSERT INTO
moms
(username, mother)
VALUES
('kramer', 'Babs Kramer')
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

SELECT

- A SELECT query extracts information from a table.

```
SELECT
<columns>
FROM
<table>
WHERE
<predicate>
```

SELECT

- A SELECT query extracts information from a table.

```
SELECT
idnum, fullname
FROM
users
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

SELECT

- A SELECT query extracts information from a table.

```
SELECT
password
FROM
users
WHERE
idnum < 12
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

SELECT

- A SELECT query extracts information from a table.

```
SELECT
*
FROM
moms
WHERE
username = 'jerry'
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

JOIN

- Databases empower us to organize information into tables efficiently.

- We don’t always need to store every possible relevant piece of information in the same table, but rather we can use relationships across tables to connect all the pieces of data we need.

- Let’s imagine we need to get a user’s full name (from the *users* table) and their mother’s name (from the *moms* table).

JOIN

- A `SELECT (JOIN)` query extracts information from multiple tables.

```
SELECT
<columns>
FROM
<table1>
JOIN
<table2>
ON
<predicate>
```

JOIN

- A SELECT (JOIN) query extracts information from multiple tables.

```
SELECT
users.fullname, moms.mother
FROM
users
JOIN
moms
ON
users.username = moms.username
```

JOIN

- A SELECT (JOIN) query extracts information from multiple tables.

```
SELECT
users.fullname, moms.mother
FROM
users
JOIN
moms
ON
users.username = moms.username
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

users & moms

users.idnum	users.username moms.username	users.password	users.fullname	moms.mother
10	jerry	fus!ll!	Jerry Seinfeld	Helen Seinfeld
11	gcostanza	b0sc0	George Costanza	Estelle Costanza

users & moms

users.idnum	users.username moms.username	users.password	users.fullname	moms.mother
10	jerry	fus!ll!	Jerry Seinfeld	Helen Seinfeld
11	gcostanza	b0sc0	George Costanza	Estelle Costanza

UPDATE

- An `UPDATE` query modifies information in a table.

```
UPDATE
<table>
SET
<column> = <value>
WHERE
<predicate>
```

UPDATE

- An UPDATE query modifies information in a table.

```
UPDATE
users
SET
password = 'yadayada'
WHERE
idnum = 10
```

users

idnum	username	password	fullname
10	jerry	yadayada	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

DELETE

- A DELETE query removes information from a table.

```
DELETE FROM
<table>
WHERE
<predicate>
```

DELETE

- A `DELETE` query removes information from a table.

```
DELETE FROM
users
WHERE
username = 'newman'
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

users

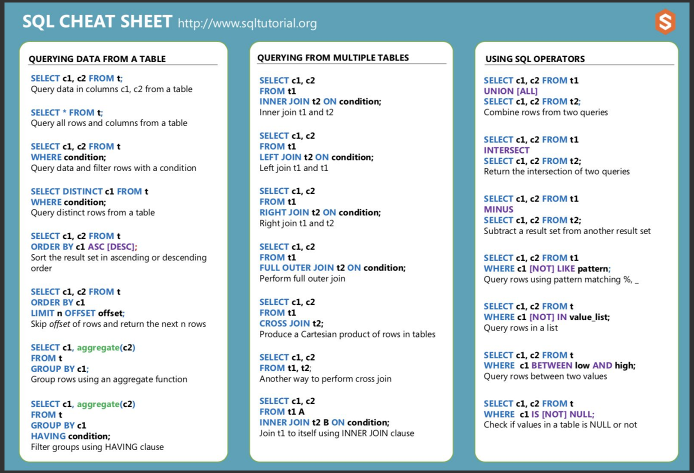
idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

STRUCTURED QUERY LANGUAGE (SQL)

SQL Cheat Sheet



SQL CHEAT SHEET http://www.sqltutorial.org

QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
```
Query data in columns c1, c2 from a table

```
SELECT * FROM t;
```
Query all rows and columns from a table

```
SELECT c1, c2 FROM t
WHERE condition;
```
Query data and filter rows with a condition

```
SELECT DISTINCT c1 FROM t
WHERE condition;
```
Query distinct rows from a table

```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```
Sort the result set in ascending or descending order

```
SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
```
Skip *offset* of rows and return the next n rows

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```
Group rows using an aggregate function

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```
Filter groups using HAVING clause

QUERYING FROM MULTIPLE TABLES

```
SELECT c1, c2
FROM t1
INNER JOIN t2 ON condition;
```
Inner join t1 and t2

```
SELECT c1, c2
FROM t1
LEFT JOIN t2 ON condition;
```
Left join t1 and t1

```
SELECT c1, c2
FROM t1
RIGHT JOIN t2 ON condition;
```
Right join t1 and t2

```
SELECT c1, c2
FROM t1
FULL OUTER JOIN t2 ON condition;
```
Perform full outer join

```
SELECT c1, c2
FROM t1
CROSS JOIN t2;
```
Produce a Cartesian product of rows in tables

```
SELECT c1, c2
FROM t1, t2;
```
Another way to perform cross join

```
SELECT c1, c2
FROM t1 A
INNER JOIN t2 B ON condition;
```
Join t1 to itself using INNER JOIN clause

USING SQL OPERATORS

```
SELECT c1, c2 FROM t1
UNION [ALL]
SELECT c1, c2 FROM t2;
```
Combine rows from two queries

```
SELECT c1, c2 FROM t1
INTERSECT
SELECT c1, c2 FROM t2;
```
Return the intersection of two queries

```
SELECT c1, c2 FROM t1
MINUS
SELECT c1, c2 FROM t2;
```
Subtract a result set from another result set

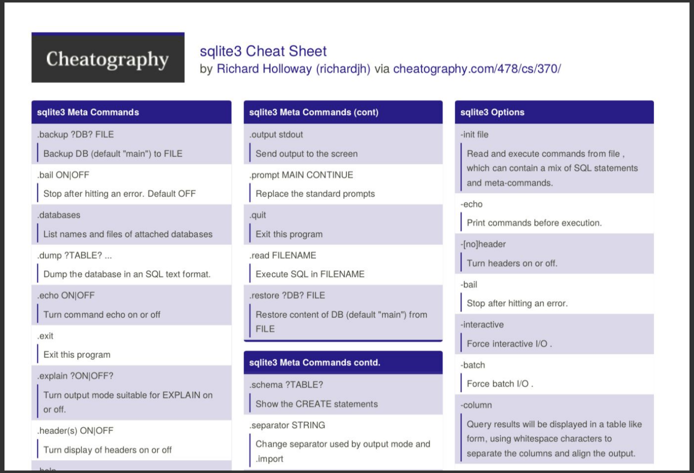
```
SELECT c1, c2 FROM t1
WHERE c1 [NOT] LIKE pattern;
```
Query rows using pattern matching %, _

```
SELECT c1, c2 FROM t
WHERE c1 [NOT] IN value_list;
```
Query rows in a list

```
SELECT c1, c2 FROM t
WHERE  c1 BETWEEN low AND high;
```
Query rows between two values

```
SELECT c1, c2 FROM t
WHERE  c1 IS [NOT] NULL;
```
Check if values in a table is NULL or not

sqlite3 Cheat Sheet



Cheatography

sqlite3 Cheat Sheet
by Richard Holloway (richardjh) via cheatography.com/478/cs/370/

sqlite3 Meta Commands	
.backup ?DB? FILE	Backup DB (default "main") to FILE
.bail ON|OFF	Stop after hitting an error. Default OFF
.databases	List names and files of attached databases
.dump ?TABLE? ...	Dump the database in an SQL text format.
.echo ON|OFF	Turn command echo on or off
.exit	Exit this program
.explain ?ON|OFF?	Turn output mode suitable for EXPLAIN on or off.
.header(s) ON|OFF	Turn display of headers on or off

sqlite3 Meta Commands (cont)	
.output stdout	Send output to the screen
.prompt MAIN CONTINUE	Replace the standard prompts
.quit	Exit this program
.read FILENAME	Execute SQL in FILENAME
.restore ?DB? FILE	Restore content of DB (default "main") from FILE

sqlite3 Meta Commands contd.	
.schema ?TABLE?	Show the CREATE statements
.separator STRING	Change separator used by output mode and .import

sqlite3 Options	
-init file	Read and execute commands from file , which can contain a mix of SQL statements and meta-commands.
-echo	Print commands before execution.
-[no]header	Turn headers on or off.
-bail	Stop after hitting an error.
-interactive	Force interactive I/O .
-batch	Force batch I/O .
-column	Query results will be displayed in a table like form, using whitespace characters to separate the columns and align the output.

Demo

Creating a Database

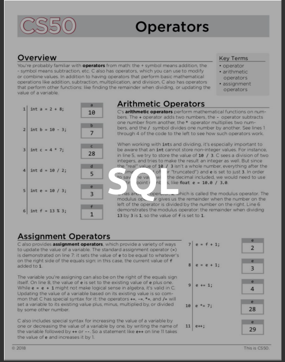
Now Let’s Do Some Exercises!

http://bit.ly/2oojaps

Exercise Solutions

https://github.com/rw5614/cs50sectiontf/blob/master/solution.sql

REFERENCE SHEETS



SQL

CS50 **Operators**

Overview

You're probably familiar with **operators** from math: the + symbol means addition, the - symbol means subtraction, etc. C also has operators, which you can use to modify or combine values. In addition to having operators that perform basic mathematical operations like addition, subtraction, multiplication, and division, C also has operators that perform other functions: like finding the remainder when dividing, or updating the value of a variable.

Key Terms

- operator
- arithmetic operators
- assignment operators

Arithmetic Operators

	Code	Variable	Value
1	`int a = 2 + 8;`	a	10
2	`int b = 10 - 3;`	b	7
3	`int c = 4 * 7;`	c	28
4	`int d = 10 / 2;`	d	5
5	`int e = 10 / 3;`	e	3
6	`int f = 13 % 3;`	f	1

C's **arithmetic operators** perform mathematical functions on numbers. The **+** operator adds two numbers, the **-** operator subtracts one number from another, the ***** operator multiplies two numbers, and the **/** symbol divides one number by another. See lines 1 through 4 of the code to the left to see how such operators work.

When working with **int**s and dividing, it's especially important to be aware that an **int** cannot store non-integer values. For instance, in line 5, we try to store the value of **10 / 3**. C sees a division of two integers, and tries to make the result an integer as well. But since the "real" value of **10 / 3** isn't a whole number, everything after the decimal [illegible] off (or "truncated") and **e** is set to just **3**. In order to save the value with the decimal included, we would need to use [illegible] point [illegible], like **float e = 10.0 / 3.0**.

C has another operator, **%**, which is called the modulus operator. The modulus operator gives us the remainder when the number on the left of the operator is divided by the number on the right. Line 6 demonstrates the modulus operator: the remainder when dividing **13** by **3** is **1**, so the value of **f** is set to **1**.

Assignment Operators

C also provides **assignment operators**, which provide a variety of ways to update the value of a variable. The standard assignment operator (**=**) is demonstrated on line 7: it sets the value of **e** to be equal to whatever's on the right side of the equals sign: in this case, the current value of **f** added to **1**.

The variable you're assigning can also be on the right of the equals sign itself. On line 8, the value of **e** is set to the existing value of **e** plus one. While **e = e + 1** might not make logical sense in algebra, it's valid in C. Updating the value of a variable based on its existing value is so common that C has special syntax for it: the operators **+=**, **-=**, ***=**, and **/=** will set a variable to its existing value plus, minus, multiplied by, or divided by some other number.

C also includes special syntax for increasing the value of a variable by one or decreasing the value of a variable by one, by writing the name of the variable followed by **++** or **--**. So a statement like **e++** on line 11 takes the value of **e** and increases it by 1.

	Code	Variable	Value
7	`e = f + 1;`	e	2
8	`e = e + 1;`	e	3
9	`e += 1;`	e	4
10	`e *= 7;`	e	28
11	`e++;`	e	29

© 2018 This is CS50.

https://www.dropbox.com/sh/5y662ey1hc4sde4/AACjgHN3NtSKk4ShsRDFd_Sja?dl=0&m=&preview=SQL.pdf

THE WEEK AHEAD

Final Live Lecture: Fri 11/1

Pset 7: Sun 11/3 @ 11:59pm

- `Movies`
- `Houses`

Test:

- Review Session: Mon 11/4 @ 3-5pm, Sanders (Also Posted Online)
- Released Mon 11/4 @ 7:30pm
- Due Fri Nov 8 @ 11:59pm

TEST REVIEW TIPS

TEST (11/4 - 11/8)

- Covers Week 0 Through 7
- Open book, *Conceptually-Focused*
- Released via Course Website
- Submitted via submit50
- Expect to spend several hours (not days!)
- Preparation Guidelines: https://cs50.harvard.edu/college/test/

TIPS FOR THE TEST

- Review lecture notes, these slides, reference sheets, and other conceptually-focused materials
- We've spent a lot of time discussing context and *why* in section—Apply that mindset to the test
  - Ask: Why do we use a certain algorithm? What role does this play in my code? Is this syntax or convention?
- Work out the Test problems on paper and type them up after the fact
  - Remember our whiteboarding strategy
  - If you're trying to come up with an algorithm, run through examples, **edge cases**, etc.

ONE MORE THING

It’s been a privilege to be your Teaching Fellow these weeks.

Here to be a resource for you all now and *after* CS50:

- raymond_wang@college.harvard.edu
- (617) 949-6919

THANKS FOR BEING A FANTASTIC SECTION!