Databases

CMSE 890-402

When is a database not a table?

- A database is a *collection* of tables
- A relational database includes *relationships* between tables
- Relationships are defined with *keys*
- Keys are unique identifiers for rows
    - Primary key in the originating table
    - Foreign key when a primary key is linked to another table

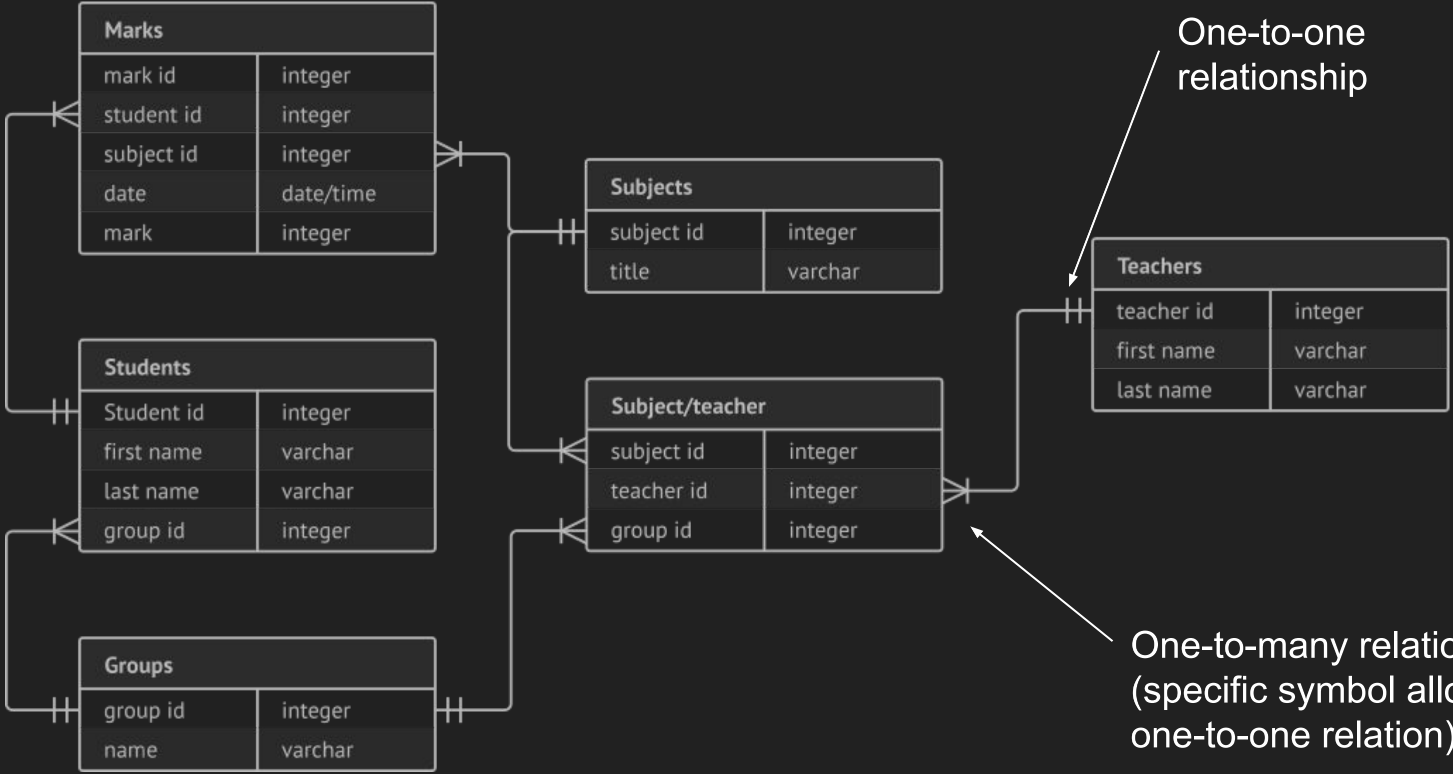
Relationship Types

- One-to-one: there can only be one instance of the key in each related table
- One-to-many: there can be multiple instances of the keys in one of the related tables
- Many-to-many: there can be multiple instances of the keys in both of the related tables

Why use a database?

- Great for data with relationships to other data
- Monolithic (i.e. one file to worry about)
- Easy to query across multiple tables at once

Entity relationship diagrams



Marks	
mark id	integer
student id	integer
subject id	integer
date	date/time
mark	integer

Subjects	
subject id	integer
title	varchar

Teachers	
teacher id	integer
first name	varchar
last name	varchar

Students	
Student id	integer
first name	varchar
last name	varchar
group id	integer

Subject/teacher	
subject id	integer
teacher id	integer
group id	integer

Groups	
group id	integer
name	varchar

One-to-one relationship

One-to-many relationship (specific symbol allows for one-to-one relation)

Cursors and Transactions

- A *cursor* is a process that enables database access
- They “point” to each row to process commands
- Cursors can require additional overhead compared to other methods

- A *transaction* is any unit of work done on a database, such as a command
- Transactions should pass the “ACID test”
  - Atomicity (transactions are single units), Consistency (database state preserves constants), Isolation (transactions can happen simultaneously or sequentially with no difference in result), Durability (completed transactions are guaranteed to apply even with system failure)

SQL

- Structured Query Language
- Designed to be simple and logical
- Query = collection of requests to the database to return data
- Database format
- **Requires a server to function** in most implementations
    - This means you can host a database on the web for remote access
    - High security
    - High scalability
    - Reliable (if you put the work in)

SQLite

- An *implementation* of SQL written in C that can be used to query SQL databases
- Supports multiple interface languages e.g. Python, Java
- Does **NOT** require a server to run
- Supports all major SQL commands
- Can load databases into memory (fast!)
- Dynamic types for database columns, so any value in any column
    - Something to be careful about!

Basic syntax and commands

COMMAND *argument* ADDITIONAL *argument*

ADDITIONAL *argument;*

SELECT *column1, column2* FROM *table1, table2*

WHERE *condition;*

INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)

VALUES (*value1*, *value2*, *value3*, ...);

WHERE Conditions

- AND, OR, NOT
- LIKE ‘string%’
    - % is a wildcard character that allows for matching, in this case “string” with any characters after
- IN (‘entry1’, ‘entry2’, ‘entry3’)
- BETWEEN value1 AND value2

Other useful commands

```
CREATE DATABASE databasename;

CREATE TABLE table_name (

      column1 datatype,

      column2 datatype,

      column3 datatype,

  ....

);
```

In-class assignment

- Go to https://mystery.knightlab.com/walkthrough.html
- Go through the
- Try to solve the mystery!
- You can also go to https://github.com/NUKnightLab/sql-mysteries
- This goes through installing a local SQLite client to solve the mystery

Homework

- Python 3 has a built-in SQLite module, sqlite3
  - https://docs.python.org/3/library/sqlite3.html
- Using this module, convert your in-class assignment into Python code
- Also complete the optional task of finding the mastermind behind the murder
- Upload the script to the github classroom
- https://classroom.github.com/a/r-QioF-A