INFINITYWORKS ACADEMY

Generation

Databases

Overview

- What is a database?
- SQL
- DDL / DML
- Modelling
- Connecting to a database from Python

Learning Objectives

- Explain what a database is and how to utilise one
- Describe SQL and it's categories, DDL and DML
- Identify modelling relationships in data
- Setup your own database and interact with it using Python

What is a Database?

A structured set of data held in a computer, especially one that is accessible in various ways.

Databases typically follow a client-server architecture:

Client
Database

The client might be:

- A command line tool
- GUI tool
- A library for a programming language!

Aspects of a Database

Schema: The information about the structure of the database and how things relate to each other

Table: Holds a collection of columns and rows

Column: A set of values of a particular data type (e.g. string or int)

Row: An entry in a table (sometimes called a record)

Cell: A single value at the intersection between a row and a column (sometimes called a field)

Constraints: Automatically restrict what data can appear in a column

Why do we need a database?

- Storing data in memory risks data loss
- Big files are slow to read and write
- Small files adds complexity to your application
- Integrity of data is easier to maintain
- Data security is easier to manage

So, how can we utilise one?

We can use an RDBMS. Otherwise known as a Relational DataBase Management System.

They allow us to work with tables and the relationships between them, typically through an application.

RDBMS Examples

- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL

We will be using MySQL for this module.

SQL

- Structured Query Language
- THE language for querying relational databases
- Declarative, not imperative
- i.e tell it what to do, not how to do it

SQL Data Types

Like Python, SQL has data types:

CHAR: A fixed-length string with a 0-255 character size that is defined on table creation.

VARCHAR: A variable-length string with a 0-65535 size.

BOOL: Zero is false, one is true.

INT: An integer value.

DECIMAL: An exact fixed-point number.

SQL Data Types

DATE: A date format.

TIME: A time format.

DATETIME: A date-time format.

All of these types are for MySQL. Different SQL providers can have different data types so be careful.

There are also plenty more data types out there, but we don't need to know them all right now.

DDL

Data Definition Language

- Deals with database schemas and descriptions, and how the data should reside in the database

DDL handles these commands:

- CREATE - create database/table
- ALTER - alters the structure of existing database/table
- DROP - deletes tables from a database
- TRUNCATE - remove all records from a table
- COMMENT - add comments similar to Python
- RENAME - rename a table

Create Database

```sql
CREATE DATABASE name_of_db;
```

Create Table

Structure

```
CREATE TABLE < table name > (
column1 < type >,
column2 < type >,
.
.
.
);
```

Example

```sql
CREATE TABLE person (
  person_id INT NOT NULL AUTO_INCREMENT,
  first_name VARCHAR(100) NOT NULL,
  last_name VARCHAR(100) NOT NULL,
  age INT,
  PRIMARY KEY(person_id)
);
```

Keywords

`NOT NULL`: Do not allow this field to be null when inserting a new record.

`AUTO_INCREMENT`: Tells MySQL to add the next available number when inserting a new record. This will make more sense on the next slide.

INFINITYWORKS ACADEMY

Generation

ALTER TABLE

If you need to amend a table's design, you can do the following:

```sql
ALTER TABLE person
ADD email varchar(255);
```

You can add or drop columns for instance.

DROP TABLE

You can remove a table entirely:

```
DROP TABLE person;
```

This is an operation which isn't used frequently, so be careful!

Exercise

Complete the steps described in part 1 of the handout.

DML

Data Manipulation Language

- Deals with data manipulation, and includes most common SQL statements
- Used to store, modify, retrieve, delete and update data in database.

DML handles these commands (and more):

- SELECT - retrieve data from one or more tables
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - delete all records from a table

INSERT

Inserts a row into your table:

```
INSERT INTO person (first_name, last_name, age)
  VALUES ('Jane', 'Bloggs', 32)
INSERT INTO person (first_name, last_name, age)
  VALUES ('Joe', 'Bloggs', 28)
```

- Any non-null fields are required as a value
- Any field not provided with a value will default to null

UPDATE

A way to change a row that is already stored:

```sql
UPDATE person SET age = 25 WHERE first_name = 'Joe'
```

DELETE

Deletes a row given certain conditions:

```
DELETE FROM person WHERE first_name = 'Joe'
```

SELECTing Data

This is the skeleton command for selecting data:

```
SELECT ...
FROM ...
WHERE ...
ORDER BY ...
LIMIT ...
```

Only `SELECT` and `FROM` are required, the rest are optional but allow for more refined queries.

SELECT

Specifies which fields to return and takes a comma-separated list of field names:

```
SELECT first_name, last_name, age
```

`*` represents all columns in the table:

```
SELECT *
```

FROM

Specifying which table you're querying against:

```
SELECT first_name, last_name, age
FROM person
```

WHERE

Specifying a predicate that evaluates whether a row should be returned. Can take in wildcard matching:

```
SELECT *
FROM person
WHERE age = 25
```

INFINITYWORKS ACADEMY

Generation

Complex WHERE

You can build where clauses that use boolean operators.

You can compound the boolean operators for even more fun!

```
SELECT * FROM person
WHERE first_name = "Mike"
AND (surname = "Goddard" OR age >= 28);
```

ORDER BY

Specifying the order in which the data is returned.

Optionally, the direction of the order can be appended.

Can add multiple columns on which to order:

```
SELECT *
FROM person
WHERE age = 25
ORDER BY first_name DESC
```

LIMIT

Limits the number of records returned:

```
SELECT *
FROM person
WHERE age = 25
ORDER BY first_name DESC
LIMIT 10
```

Primary Key

- A special table column (or combination of columns) that uniquely identify each record
- Every row must have a PK and must be unique
- Often a self-incrementing number
- Cannot be `null`

```
CREATE TABLE person (
  person_id INT NOT NULL AUTO_INCREMENT,
  PRIMARY KEY(person_id)
);
```

Foreign Key

- A column (or combination of columns) that provides a link between two tables
- Acts as a cross-reference between two tables as it references the PK of another table
- Must always reference a PK

```
CREATE TABLE contact_info(
  id INT NOT NULL AUTO_INCREMENT,
  person_id INT,
  phone VARCHAR(15),
  email VARCHAR(100),
  PRIMARY KEY(id),
  FOREIGN KEY(person_id) REFERENCES person(person_id)
);
```

Exercise

Complete the steps described in part 2 of the handout.

Quiz Time! 🤓

Which of these describes an entry in a table?

1. Field
2. Column
3. Record
4. Table

Answer: 3

Which of these best describes DDL?

1. Deals with database schemas and descriptions, and how the data should reside in the database.
2. Deals with storage of data, modifications, retrievals, deletes and updates in a database.
3. Specifies which fields to return in a query and takes a comma-separated list of field names.
4. A special table column (or combination of columns) that uniquely identify each record.

Answer: 1

JOINs

A `JOIN` clause is used to combine rows from two or more tables, based on a relation between them.

Example 1

```sql
SELECT first_name, email
FROM person
JOIN contact_info
ON person.person_id = contact_info.person_id
```

Example 2

```sql
SELECT p.first_name, c.email
FROM person p
JOIN contact_info c
ON p.person_id = c.person_id
```

You can alias table names variables which are typically first-letter abbreviations.

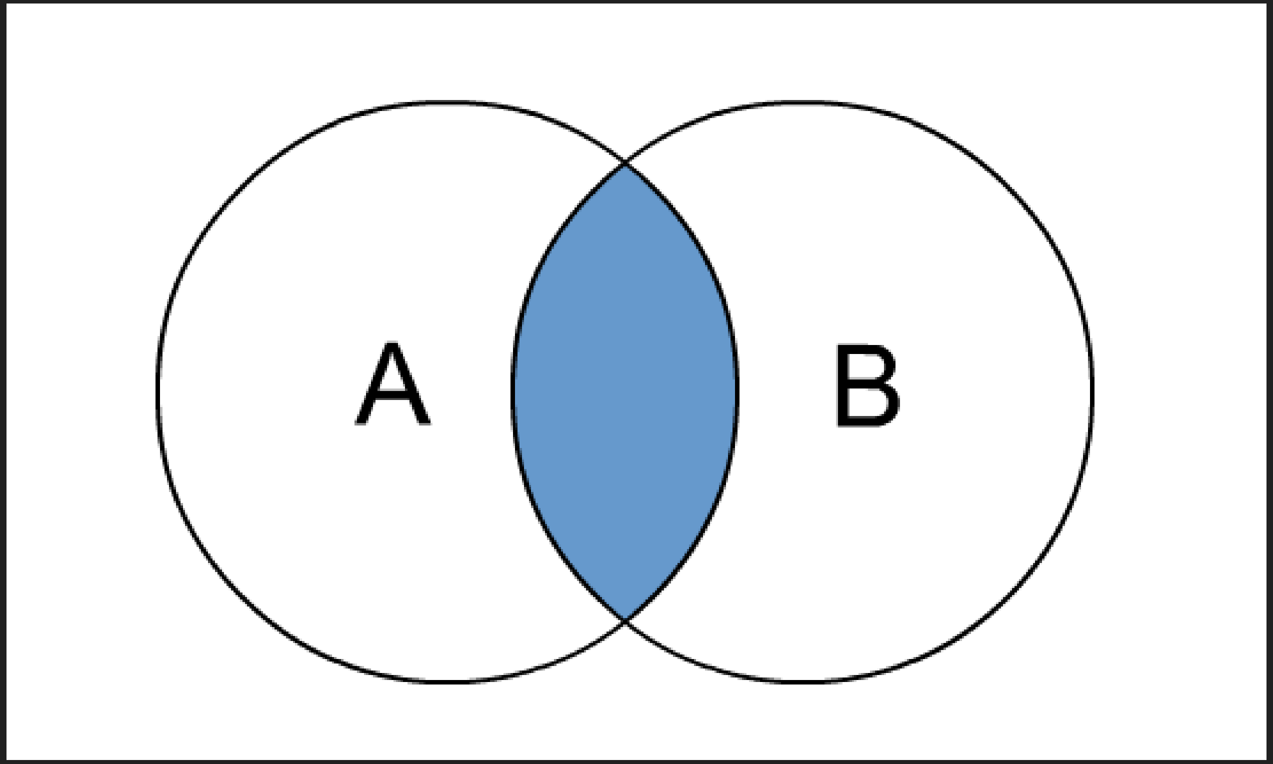
Example 3

```sql
SELECT p.first_name, c.email, d.veggie
FROM person p
JOIN contact_info c ON p.person_id = c.person_id
JOIN dietary_reqs d ON p.person_id = d.person_id
```

You can use multiple joins.

INNER JOIN

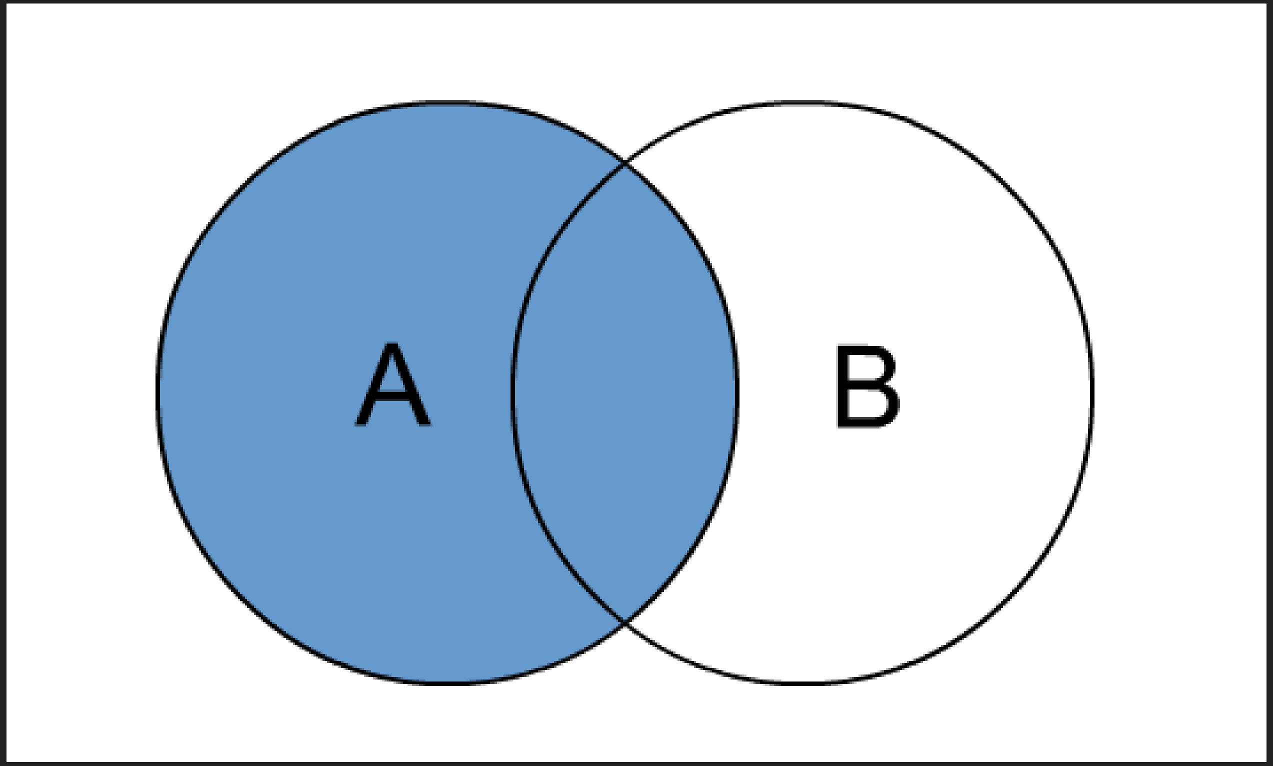
Returns records that have a matching value on both tables



LEFT/RIGHT OUTER JOIN

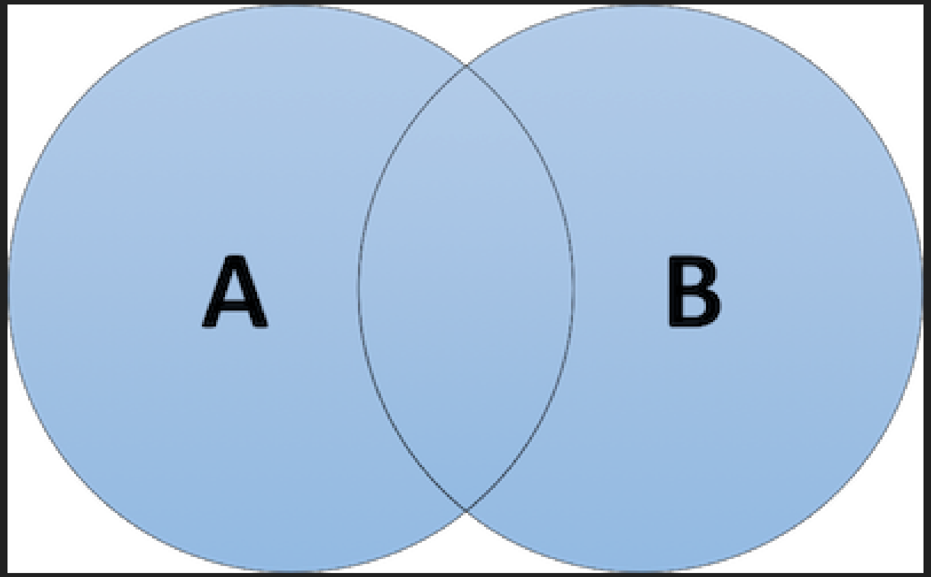
Returns all records from the left table, and the matched records on the right table.

And vice versa!



FULL OUTER JOIN

Returns all records when there is a match in either left or right table.



Exercise

We will go through some examples of joins in the joins handout.

Functions

Some useful functions you may want to use:

SUBSTRING

```
SELECT SUBSTRING(string, start, length) AS ExtractString;
```

AVERAGE, MIN, MAX

```
SELECT MIN(price) AS SmallestPrice FROM products;
```

Functions

CURRENT_DATE

```
CURDATE()
```

COUNT

```
SELECT COUNT(product_id) AS NumberOfProducts FROM products;
```

Again, there are plenty of functions that aren't covered here.

Exercise

Instead of reinventing the wheel, W3 has an excellent resource on exercises for join statements:

https://www.w3resource.com/sql-exercises/sql-joins-exercises.php

Relationships

When creating a database, we use separate tables for different types of entities (*customer*, *order*, *item*).

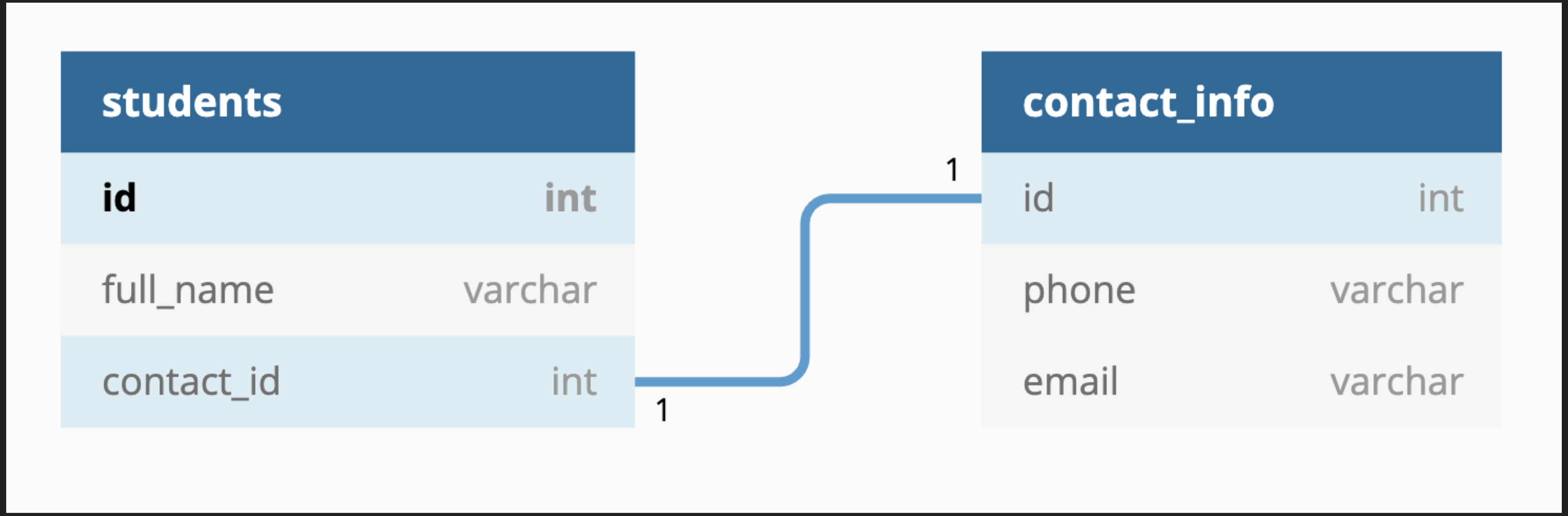
We also need to have relationships between these tables. For instance, customers make orders, and orders contain items. These relationships need to be represented in the database.

Different kinds of relationships can exist between records:

- One-to-one
- One-to-many
- Many-to-many

One-to-One

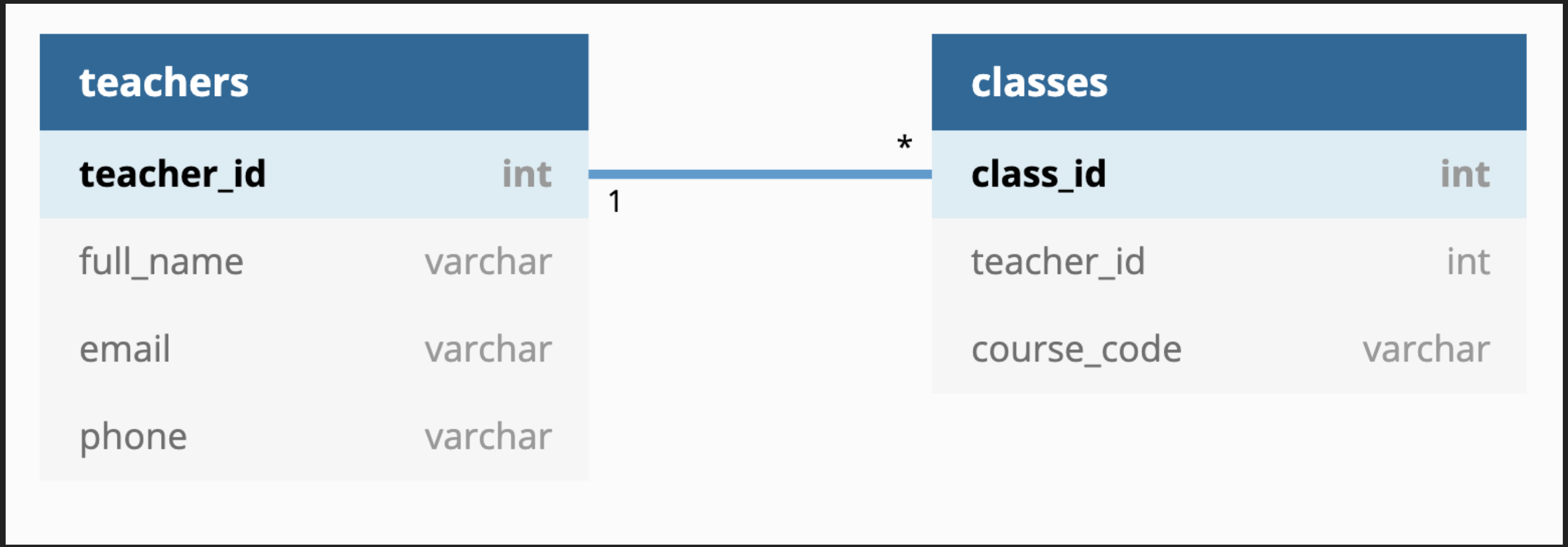
Imagine we have a table of students and another table of their contact information:



- One student has one set of contact details
- The `id` fields in each table are the primary keys
- The `contact_id` field in `students` is a foreign key on `contact_info`

One-to-Many

Imagine we have a table of teachers and a table of classes:

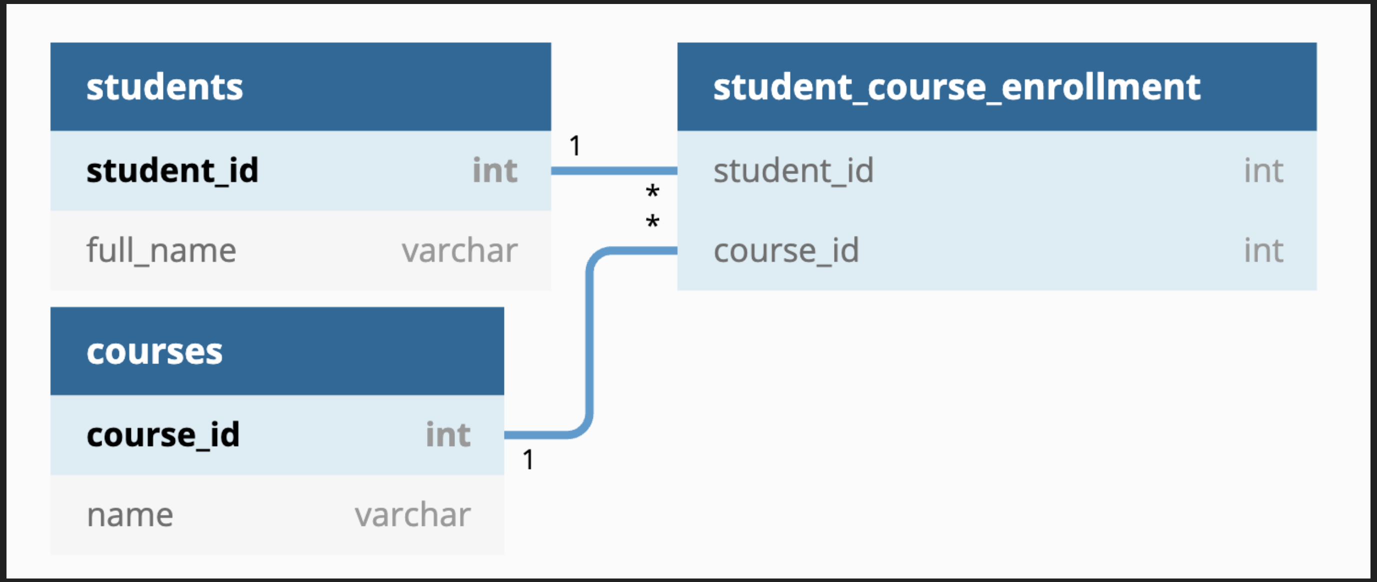


- One teacher can teach many classes
- A teacher can teach many classes, but a class can only have one teacher (in this scenario)

Many-to-Many

Imagine we have a table of students and a table of courses.

- Many students can take many courses
- Many-to-Many relationships are tricky to construct, so one way to simplify it is to have an intermediary table for course enrolment



Theory

There are some key database principles we need to look at before we move on.

NoSQL

- Refers to any non-relational DB
- Came about in the late 2000s as cost of storage vary dramatically
- As it decreased, it allowed us to store way more data, which could be structured or unstructured
- Cloud computing has driven NoSQL for it's ability to distribute data across multiple servers and regions around the world

NoSQL DB Types

Document DB

- Stores data in documents similar to JSON objects
- Each document contains pairs of fields and values, like dictionaries in Python
- Documents map to objects in your code easily
- No need to decompose data across multiple tables, or use JOINs
- Dynamic schema, can change as you go along
- Faster querying of data (not always though)
- Distributed at their core
- MongoDB and CouchDB are popular choices

Key-Value DB

- Each item contains a key and values
- A value can only be retrieved by referencing its key
- Great for storing large amounts of data but don't need to perform complex queries to retrieve it
- Redis and DynamoDB are popular choices

Wide-Column Store

- Stores data in tables, rows and dynamic columns
- Provide a lot of flexibility over relational DBs because each row is not required to have the same columns
- Great for when you need to store large amounts of data and you can predict what your query patterns will be
- Casssandra and HBase are popular choices

Graph DB

- Stores data in nodes and edges
- Nodes store information about people, places etc.
- Edges contain information about the relationships between nodes
- Great for when you need to traverse relationships to look for patterns like social networks and fraud detection
- Example: Twitter followers or LinkedIn connections
- Neo4j and JanusGraph are popular choices

So why do we need them?

Dynamic schema - no set structure

Auto-sharding - distribution of the data across servers

Auto replication - allows for high availability

The move to smaller distributed services means not relying on one central DB

Easily scalable - more servers means more replication and load balancing

They're not the silver bullet

- Sacrificing strong guarantees (which we will see next slide)
- Transactions aren't always supported
- You can end up stuck in the "relational" way of thinking when using them

ACID

Acid refers to a standard set of properties that guarantee database transactions are processed reliably.

More importantly, it is concerned with how a database recovers from any failure during a transaction.

ACID-compliant databases ensure that the database remains accurate and consistent.

More on this, but first - *what is a transaction*?

Transaction

A unit of work that is treated as a whole. It either has to happen in full or not at all.

Example: Imagine transferring money from one bank to another. This requires multiple steps:

1. Withdraw from the source account
2. Deposit it to the destination account

The operation has to succeed in full. Stopping halfway could mean money is lost.

ACID Acronyms Explained

Atomicity

Guarantees that all of the transaction succeeds, or none of it does.

An atomic system must guarantee atomicity in every situation, including power failures, errors and crashes.

Consistency

Guarantees that all data stays consistent, as in, the data will be valid according to all of the rules defined on the database.

A transaction should only bring the database from one valid state to another.

Isolation

Guarantees that all transactions occur in isolation. No transactions should affect any other transactions.

Multiple transactions are often executed concurrently (reading and writing to a table at the same time).

Isolation ensures that this concurrent execution of transaction leaves the database in the same state if the transactions were executed sequentially (one at a time).

For example, a transaction cannot read data from any transaction that is yet to complete.

Durability

Once a transaction is committed, it will remain in the system, even if there's a system crash.

If we tell a user that a transaction succeeded, it must be factual.

This normally means that completed transactions are recorded to long-term memory like a hard drive.

CAP

A theorem that states it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

Consistency: Every read receives the most recent write or an error.

Availability: Every request receives a (non-error) response, without the guarantee that it contains the most recent write

Partition Tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

CAP

No distributed system is safe from network failures, so partition tolerance needs to be tolerated.

We're left with deciding consistency over availability.

CAP

When choosing consistency, the system will return an error if particular information can't be guaranteed to be up to date due to network failure.

When choosing availability, the system will always process the query and try to return the most recent available version of the information, even if it cannot guarantee it is up to date due to network failure.

When there is no network failure, both can be satisfied.

CAP

CAP is frequently misunderstood as if we have to choose to abandon one of the three guarantees at all times.

In fact, the choice is really between consistency and availability only when a network partition or failure happens; at all other times, no trade-off has to be made.

Quiz Time! 🤓

Which type of relationship uses an intermediary table?

1. One-to-One
2. One-to-Many
3. Many-to-One
4. Many-to-Many

Answer: 4

How does a many-to-many relationship differ from the other two kinds of relationships?

1. Primary key fields are used for both sides of the relationship.
2. Three tables are used instead of two tables.
3. You can't use referential integrity.
4. No primary key fields are used.

Answer: 2

Which of these is NOT a NoSQL database?

1. DocumentDB
2. Postgres
3. GraphDB
4. Wide-Colum Store

Answer: 2

Which of the below best describe atomicity?

1. Once a transaction is committed, it will remain in the system, even if there's a system crash.
2. Ensures that concurrent execution of transactions leaves the database in the same state if the transactions were executed sequentially (one at a time).
3. Allowing a transaction to only bring the database from one valid state to another.
4. Guarantees that all of the transaction succeeds, or none of it does.

Answer: 4

Which of the below best describe isolation?

1. Once a transaction is committed, it will remain in the system, even if there's a system crash.
2. Allowing a transaction to only bring the database from one valid state to another.
3. Ensures that concurrent execution of transactions leaves the database in the same state if the transactions were executed sequentially (one at a time).
4. Allowing for concurrent execution of data (reading and writing to a table at the same time).

Answer: 3

SQL in Python

You will need `pymysql`:

```
$ python -m pip install pymysql
```

`pymysql` is a third-party MySQL client library that will allow us to interact with a database purely through Python.

Exercise

Complete the steps described in part 3 of the handout.

Learning Objectives Revisited

- Explain what a database is and how to utilise one
- Describe SQL and it's categories, DDL and DML
- Identify modelling relationships in data
- Setup your own database and interact with it using Python

Terms and Definitions Recap

Database: An organized collection of data, generally stored and accessed electronically from a computer system.

Relational Database: A digital database based on the relational model of data.

SQL: A domain-specific language used in programming and designed for managing data held in a relational database management system.

Query: A precise request for information retrieval with database and information system.

Terms and Definitions Recap

DDL: A data description language is a syntax for creating and modifying database objects such as tables, indexes, and users.

DML: A data manipulation language is a computer programming language used for inserting, deleting, and updating data in a database.

Primary Key: A specific choice of a minimal set of attributes (columns) that uniquely specify a tuple (row) in a relation (table).

Foreign Key: A set of attributes in a table that refers to the primary key of another table.

Further Reading

- Learn SQL with Code Academy
- Database Design
- pymysql
- Database definitions
- Database definitions discussion