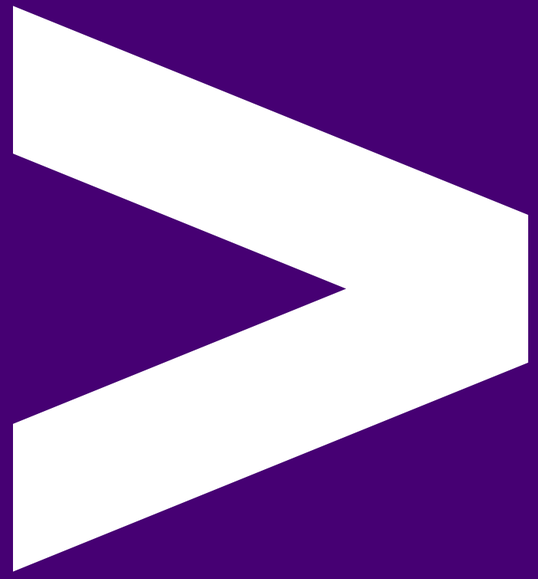
School of Tech
part of accenture

# Databases



School of Tech
part of accenture

# Overview

- What is a database?
- SQL
- DDL and DML
- Using DDL SQL statements
- Using DML SQL statements
- Connecting to a database and using SQL using Python

School of Tech
part of accenture

# Objectives

- Learn the features of a database
- Learn why we need databases
- Get introduced to the differences between relational and non-relational databases
- See some examples of SQL and learn about DDL and DML
- Learn, and practice using, SQL commands to interact with a database
- Learn, and practice using, Python to interact with a database

# What is a Database?

A collection of structured information or data organised for convenient access.

Databases typically follow a client-server architecture:


The client might be:

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

School of Tech
part of accenture

# Aspects of data

Table - holds a collection of columns and rows.

Column - hold different types of data.

Row - an individual entry in the table.

Constraints - automatically restrict what data can appear in a column.

Schema - the information about the structure of the database and any constraints it has.

# Example table

## Student

| student_id | first_name | last_name | birth_date | mentor_id |
|---|---|---|---|---|
| 001 | Bernard | Matthews | 19700101 | 123 |
| 002 | Josh | Leeds | 19410709 | 123 |
| 003 | Vince | Levis | 19930430 | 420 |

# 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.

# Relational vs Non-relational

What is a relation?

A relation is a set of tuples (rows) that share the same attributes (columns). Essentially - a table.

Tables in relational databases can have a connection.

School of Tech
part of accenture

# Database Relationship

In databases, a relationship is a connection between two or more records in different tables, usually established through a shared ID, like `user_id`.

```
User table

| user_id | name  |
|---------|-------|
| 1       | Alice |
```

```
Order table

| order_id | user_id | item   |
|----------|---------|--------|
| 101      | 1       | Laptop |
```

# Non-relational

Anything that does not rely on relations between records.

Most can be broadly put into one of these categories:

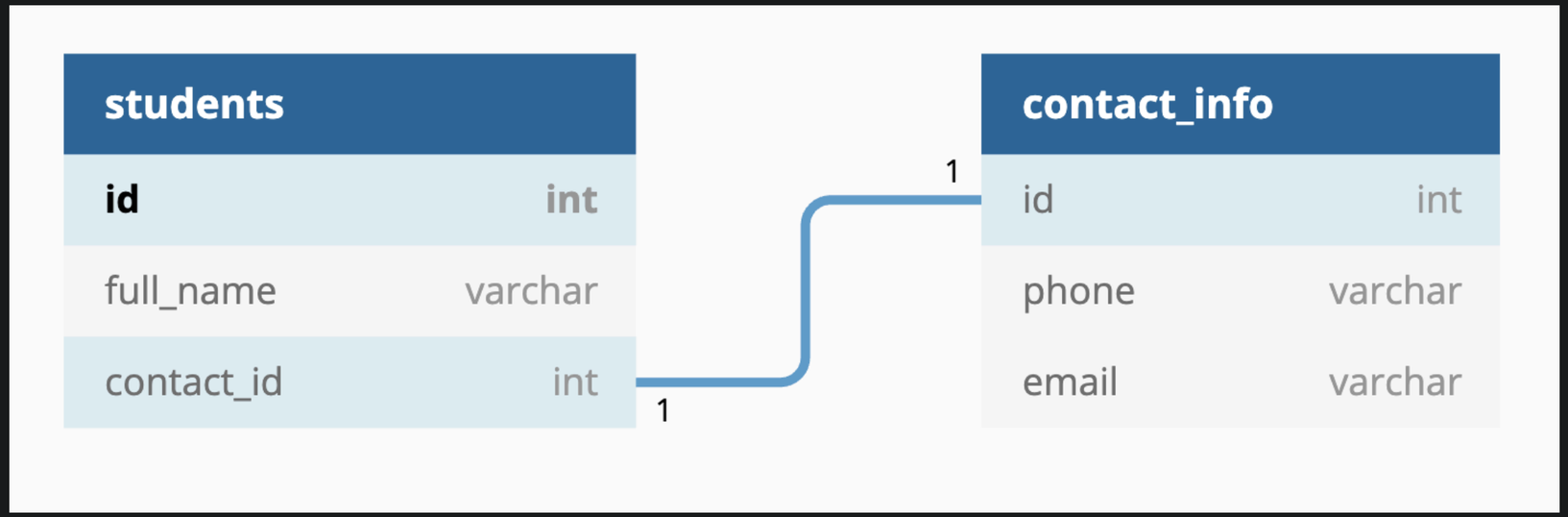
- Key-value stores (like a Dictionary)
- Graph stores (like a mind-map structure)
- Column stores (as opposed to Row stores - potentially faster for analytics)
- Document stores (eg JSON files)

# Using relations

Different kinds of relationships can exist between records:

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

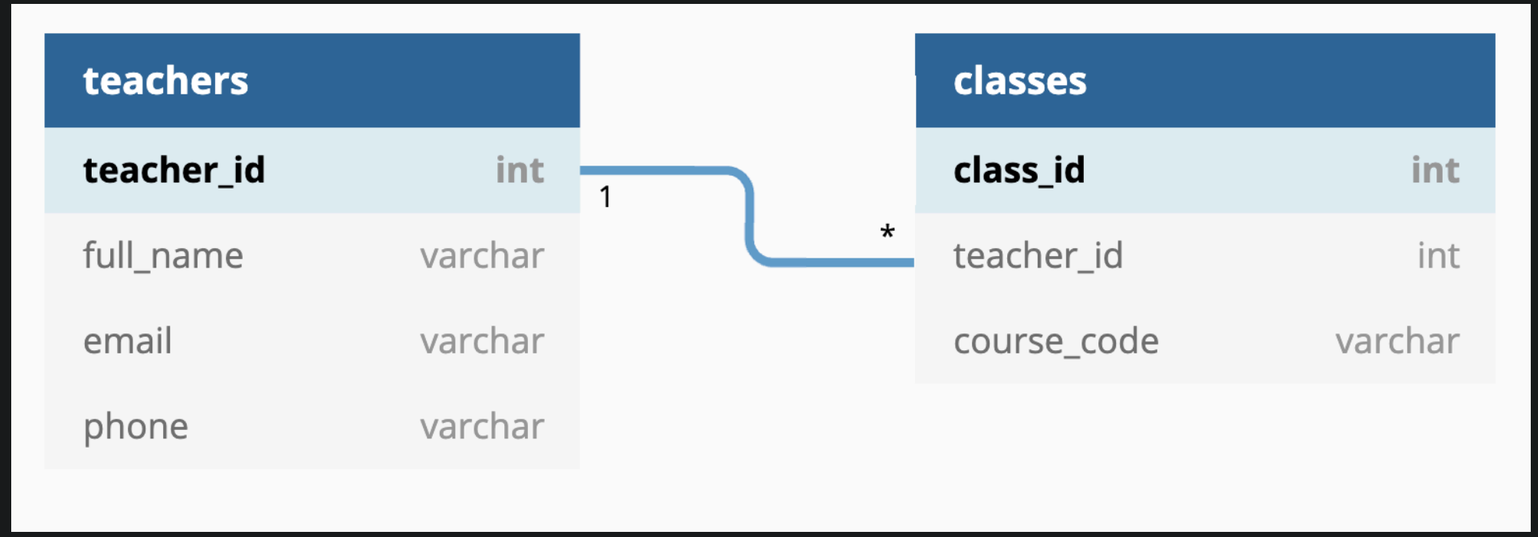
# One to One


students
id int
full_name varchar
contact_id int
1
1
contact_info
id int
phone varchar
email varchar

School of Tech
part of accenture

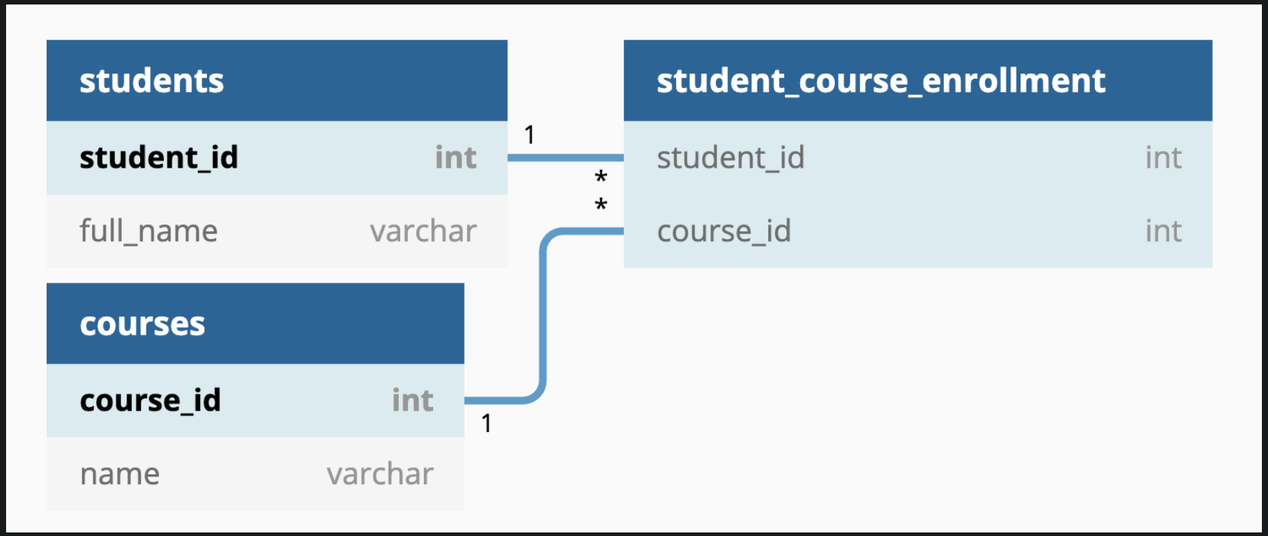
# One to Many

- Is also Many-to-one



School of Tech
part of accenture

# Many to Many



# Pros of Relational stores:

- Reduced data duplication
- Allows our database to enforce integrity for us
- Mature tooling
- How we access the data doesn't matter so much, SQL is very flexible

# Cons of relational stores:

- Rigid schemas are hard to change
- Not all problems map well to relations
- Harder to scale
- Big-Data complex reports or analytics can be faster in Column based stores

# So, how can we utilise it?

`(R)DBMS`

(Relational) Database Management System.

Works with tables and the relationships between them.

Create custom views on our data.

Program custom scripts (Stored Procedures) to query and manipulate our data.

# Some relational databases you may have heard of...

- Oracle
- MySQL
- PostgreSQL
- Microsoft SQL Server
- Microsoft Access (sort of)
- SQLite

# Emoji Check:

On a high level how do you feel about the use of databases?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

School of Tech
part of accenture

# Task - 10 mins

- In groups try and come up with a design for a database for a school
- Think about what data needs storing and the relationships between that data
- Draw out your design and we'll share them shortly

# SQL

Structured Query Language (SQL)

THE language for querying relational databases.

Declarative, not imperative (i.e tell it what to do, not how to do it).

# DDL vs DML

Data Definition Language:

- Create, alter and drop tables to create a schema

Data Manipulation Language:

- Working with the data in your tables

# DDL

## Structure

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

# Table Example

```
CREATE TABLE person_details (
    person_id INTEGER GENERATED ALWAYS AS IDENTITY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    age INTEGER,
    PRIMARY KEY(person_id)
);
```

- `GENERATED ALWAYS AS IDENTITY` is an auto-incrementing number
- `PRIMARY KEY` tells us what is unique for each row
- `VARCHAR` is a variable-length text field.

# Table Example

```
                                Table "public.person_details"
   Column   |          Type          | Collation | Nullable |           Default
------------+------------------------+-----------+----------+-------------------------------
 person_id  | integer                |           | not null | generated always as identity
 first_name | character varying(100) |           | not null |
 last_name  | character varying(100) |           | not null |
 age        | integer                |           |          |
Indexes:
    "person_details_pkey" PRIMARY KEY, btree (person_id)
```

# Tables and linking data

What if we want a second table that references the first? For example, contact info that is tied to a person?

Then we create the the second table and link back with a Constraint.

We must create the matched columns in both tables i.e. `person_id`, then link them with a `FOREIGN KEY CONSTRAINT`.

# Foreign Keys example

```
CREATE TABLE contact_info (
   info_id INTEGER GENERATED ALWAYS AS IDENTITY,
   person_id INT,
   phone VARCHAR(15),
   email VARCHAR(100),
   PRIMARY KEY(info_id),
   CONSTRAINT fk_person
     FOREIGN KEY(person_id)
     REFERENCES person(person_id)
);
```

- The `CONSTRAINT` clause requires the `FOREIGN KEY` and `REFERENCES` parts too

# Table Example

```
                                Table "public.contact_info"
  Column   |          Type          | Collation | Nullable |           Default
-----------+------------------------+-----------+----------+-------------------------------
 info_id   | integer                |           | not null | generated always as identity
 person_id | integer                |           |          |
 phone     | character varying(15)  |           |          |
 email     | character varying(100) |           |          |
Indexes:
    "contact_info_pkey" PRIMARY KEY, btree (info_id)
Foreign-key constraints:
    "fk_person" FOREIGN KEY (person_id) REFERENCES person(person_id)
```

School of Tech
part of accenture

# ALTER TABLE

We can manipulate the structure of existing tables;

```
ALTER TABLE person_details ADD date_of_birth DATE;
```

We can add and remove whole columns, and add and remove constraints.

```
                                Table "public.person_details"
    Column     |          Type          | Collation | Nullable |           Default
---------------+------------------------+-----------+----------+-----------------------------
 person_id     | integer                |           | not null | generated always as identity
 first_name    | character varying(100) |           | not null |
 last_name     | character varying(100) |           | not null |
 age           | integer                |           |          |
 date_of_birth | date                   |           |          |
Indexes:
    "person_details_pkey" PRIMARY KEY, btree (person_id)
```

# DROP TABLE

We can remove existing tables:

```
DROP TABLE contact_info;
```

(We're not expecting the use of 'DROP TABLE' often)

# Emoji Check:

How do you feel about using SQL to create tables?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

# Using Podman to run PostgreSQL

> We are going to now setup a PostgreSQL container running via the Podman service.
>
> We are using this as a tool to all get the same db running, we can go into how Podman works another time.

# Cleanup old containers 1/2

First, we need to make sure Podman is running in Windows.

In a Powershell terminal, run these commands to check:

```
podman machine list

podman machine start
> Started
```

# Clean up old containers 2/2

> Ensure no old containers are running. We need to clean up from any previous work.

- Open a Powershell terminal
- Run `podman ps -a`
  - This will show a list of container ids (e.g. `a1b2c3`) and names (e.g. `funny_einstein`)
- If you have anything running, stop it with `podman stop <container_name>`
- Remove all old containers with `podman rm <container_name>`
- Run `podman ps -a` again - you should have zero containers now!

# Setup new PostgreSQL container

On the next slides we have instructions to get your PostgreSQL container running, so we can use it to practice our SQL.

- If on Windows, try Option A first in Powershell
  - If that does not work, try Option C in GitBash
- If on WSL-Ubuntu / Mac / Unix, use Option B

# Option A - Podman Compose in Windows

> If using podman in Windows natively (not WSL), we can get up and running with podman compose.

- Open a Powershell terminal in the `exercises` directory
- Run `podman compose up -d`
- You should get something similar to the following output

```
Creating postgres_container ... done
Creating adminer_container ... done
```

# Option B - Podman script in WSL / Mac / Unix

> If using podman in WSL, we use some helper scripts instead of compose:

- Open a terminal in the `exercises` directory
- Run the command `./setup-all-podman.sh`
  - This will start the ./exercises/setup-all-podman.sh script

# Option C - Podman script in GitBash

> If using podman in Windows, we can use GitBash, and we use a helper script.

- Open GitBash
- Change directory to the `exercises` directory.
- Run the command `./setup-all-podman.sh`
    - This will start the ./exercises/setup-all-podman.sh script

# Check postgres is running

Run the following:

```
podman ps -a
```

...it should show you that PostgreSQL and Adminer are running.

# Connect to your server

There are two ways to do this in general:

- Via a shell in the postgres container
- Install some psql tools locally and use those

> We will use containers

Details on following slides.

School of Tech
part of accenture

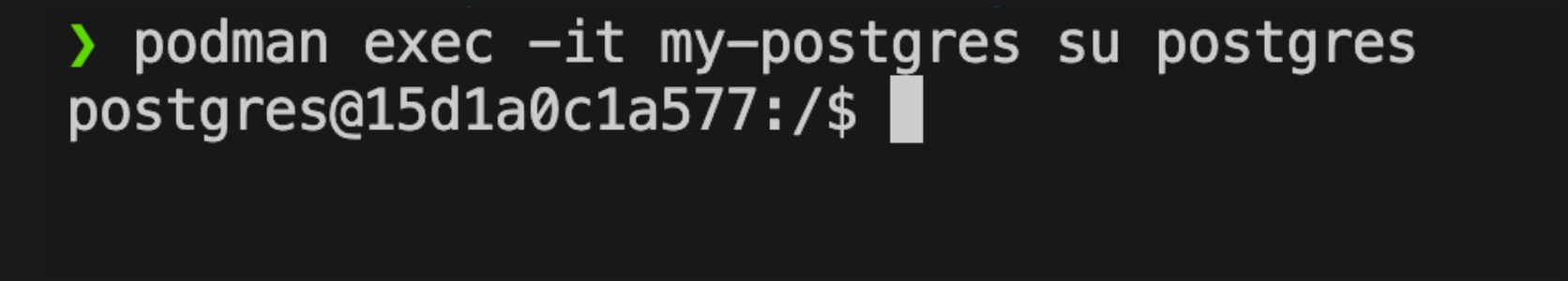
# Task - connect via the container

In your terminal run this:

```
podman exec -it my-postgres su postgres
```

...which will open a shell inside the postgres container as the `postgres` user.

You should see a prompt like `postgres@15d1a0c1a577:/$`:



School of Tech
part of accenture

# Task - run the interactive PSQL tool

Then, run the `psql` tool from within the container:

```
psql
```

You should see a prompt like `psql=#` or `postgres=#`

```
❯ podman exec -it my-postgres su postgres
postgres@15d1a0c1a577:/$ psql
psql (16.2 (Debian 16.2-1.pgdg120+2))
Type "help" for help.

postgres=#
```

# Running SQL in PSQL

When we run SQL in the PSQL interactive tool, e.g. `SELECT * FROM person;` you should see output a bit like this:

```
postgres=# SELECT * FROM person;
 person_id | first_name | last_name | age |      email
-----------+------------+-----------+-----+-----------------
         1 | Jane       | Bloggs    |  32 | jane@email.com
         2 | Alice      | Babbs     |  45 | alice@email.com
         3 | Mark       | Smith     |  51 | mark@email.com
         4 | Grace      | Matthews  |  68 | grace@email.com
(4 rows)
```

# List databases and tables

Some Postgres handy commands can be found on postgrescheatsheet.com

For now at our psql prompt we can:

- List the databases with `\l` or `\list` (there will only be the default ones yet)
- List your tables with the describe tables command `\dt` (the default is `postgres`)

# Who loves a semi-colon?

> Databases do!

Statements in sql need finishing like this;

Each semicolon; Makes a separate statement;

The commands like `\dt` are postgres-only commands, not SQL, so don't need one.

# Task - groups - 15 mins

- Working in your groups create tables matching your design
- Warning: every statement must end with a `;`
- Make a file named e.g. `create_<table_name>.sql` in VSCode and write your SQL in there - this will show you context highlighting
- When done, copy-and-paste the SQL into PSQL to check it works
- Share the files with your group afterwards

School of Tech
part of accenture

# Emoji Check:

How do you feel about writing SQL statements?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

# School Example Teacher table

```
CREATE TABLE teacher (
    teacher_id INTEGER GENERATED ALWAYS AS IDENTITY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    PRIMARY KEY(teacher_id)
);
```

# School Example Course table

```
CREATE TABLE course(
   course_id INTEGER GENERATED ALWAYS AS IDENTITY,
   teacher_id INT,
   course_name VARCHAR(255),
   course_length VARCHAR(100),
   PRIMARY KEY(course_id),
   CONSTRAINT fk_person
      FOREIGN KEY(teacher_id)
     REFERENCES teacher(teacher_id)
);
```

# Who loves a single-quote?

> Postgres does!

DML Statements in PostgreSQL need single quotes `'`, not double-quotes `"`!

E.g. `INSERT INTO person (first_name, last_name, age) VALUES ('Mike', 'Goddard', 28);`

School of Tech
part of accenture

# INSERTing Data

As you would expect, this inserts (adds) a row into your table

```
INSERT INTO person (first_name, last_name, age)
    VALUES ('Mike', 'Goddard', 28);
INSERT INTO person (first_name, last_name, age)
    VALUES ('Emily', 'Birch', 52);
```

```
INSERT INTO contact_info (person_id, email)
    VALUES (1, 'mike@dummy.com');
INSERT INTO contact_info (person_id, email)
    VALUES (null, 'edward@dummy.com');
```

Any non-null fields are required as a value.

Any field not provided with a value will default to null.

# INSERT with RETURNING

We can get back the id that the server created with the `RETURNING` keyword:

```
INSERT INTO person (first_name, last_name)
    VALUES ('Oscar', 'Cooper') RETURNING person_id;
```

This is especially useful when writing code (like python) that is adding the data for us.

# Bulk INSERT clause (aka "multi-values")

You can insert multiple rows at once, which is very efficient. Here is an example;

```
INSERT INTO drink (name, type, temperature)
    VALUES
        ('americano', 'coffee', 'hot'),
        ('iced latte', 'coffee', 'cold'),
        ('filter', 'coffee', 'hot');
```

You can also use `RETURNING` with this syntax.

School of Tech
part of accenture

# Task - groups - 10 mins

- Working in your groups, have a go at inserting some data into your tables
  - `INSERT` as single row
  - `INSERT` multiple rows at once
- Use the `RETURNING` keyword (with one or more columns)
- Make a file named e.g. `insert_<table_name>.sql` in VSCode and write your SQL in there - this will show you context highlighting
- When done, copy-and-paste the SQL into PSQL to check it works
- Share the files with your group

# Emoji Check:

How do you feel about inserting data?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

# School Examples

```
INSERT INTO teacher(first_name, last_name)
    VALUES ('Gatsby', 'Woodhead') RETURNING teacher_id;
INSERT INTO teacher(first_name, last_name)
    VALUES ('Wiggins', 'Pickering') RETURNING teacher_id;

SELECT * FROM teacher; /* shows us the ids */

INSERT INTO course(teacher_id, course_name, course_length)
    VALUES (1, 'Physics', '3 Years') RETURNING course_id;
INSERT INTO course(teacher_id, course_name, course_length)
    VALUES (2, 'Philosophy', '4 Years') RETURNING course_id;
```

School of Tech
part of accenture

# SELECTing Data

This is how we get our data back out;

SELECT

FROM

WHERE

ORDER BY

LIMIT

# SELECT

Specifies which fields to return, and any criteria for the rows.

Takes a comma-separated list of field names:

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

`*` represents everything (all fields):

```
SELECT *
```

# FROM

Specifying which table you're querying against

```
SELECT person_id, first_name FROM person;
```

# WHERE

Specifying a predicate that evaluates whether a row should be returned

```
SELECT * FROM person WHERE first_name = 'Mike';
```

Can take in wildcard matching

```
SELECT * FROM person WHERE first_name like '%ily%';
```

# 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 (last_name = 'Goddard' OR age >= 20);
```

# 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 - this is like a primary and secondary sort (and so on)

```
SELECT * FROM contact_info
    ORDER BY email DESC, phone_number ASC;
```

# LIMIT

The number of results can be limited

```
SELECT * FROM contact_info
WHERE first_name = 'Mike'
ORDER BY last_name ASC
LIMIT 1;
```

# Task - groups - 5 mins

- Use `SELECT` to pull the data back out of your tables to check the `INSERT` worked correctly
- Make a file named e.g. `select_<table_name>.sql` in VSCode and write your SQL in there - this will show you context highlighting
- When done, copy-and-paste the SQL into PSQL to check it works
- Share the files with your group

# Emoji Check:

How do you feel about selecting data?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

School of Tech
part of accenture

# Modifying Data

# UPDATE

A way to change a row that is already stored:

```
UPDATE person SET age = 25 WHERE first_name = 'Mike';
```

Best practice is to update by the primary key id(s):

```
UPDATE person SET age = 25 WHERE person_id = 123;
```

# Question: UPDATE

> What happens if we miss off the `WHERE`?

```
UPDATE person SET age = 25;
```

Answer: We change everyone at once!

# DELETE

As you'd expect, delete a row given certain conditions:

```
DELETE FROM person WHERE first_name = 'Emily';
```

Best practice is to delete by the primary key id(s):

```
DELETE FROM person WHERE person_id = 456;
```

# Question: DELETE

> What happens if we miss off the `WHERE`?

```
DELETE FROM person;
```

Answer: We delete everyone at once! Possibly a P45 too!

# Task - groups - 10 mins

- Use UPDATE to update a row
- Use DELETE to delete a row
- Make a file named e.g. `<command>_<table_name>.sql` in VSCode and write your SQL in there - this will show you context highlighting
- When done, copy-and-paste the SQL into PSQL to check it works
- Share the files with your group

# Emoji Check:

How do you feel about updating and deleting data?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

School of Tech
part of accenture

# Discussion - 2 mins

Did anyone run into problems?

# Referential Integrity

If you try and delete a row that is linked by a foreign key Postgres will complain. It's enforcing data integrity for us!

```
DELETE FROM person WHERE person_id = 1;

ERROR: update or delete on table "person" violates
    foreign key constraint "fk_person" on table "contact_info"
DETAIL:  Key (person_id)=(1) is still referenced from
    table "contact_info".
```

# Cascade deletes

These tell Postgres we want to delete child records if we delete the parent.

We can't alter constraints so we will need to delete the existing constraint first and then add a new one that will cascade deletes:

```
ALTER TABLE contact_info DROP CONSTRAINT fk_person;

ALTER TABLE contact_info
ADD CONSTRAINT fk_person
FOREIGN KEY(person_id)
REFERENCES person(person_id)
ON DELETE CASCADE;

DELETE FROM person WHERE person_id = 1;
```

# Emoji Check:

How do you feel about concept of referential integrity and cascading deletes?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

# JOINS

As you'd expect, we can join two or more tables in our query to get combined results.

School of Tech
part of accenture

# JOINS

Let's pretend we have some data in tables like this:

Person:

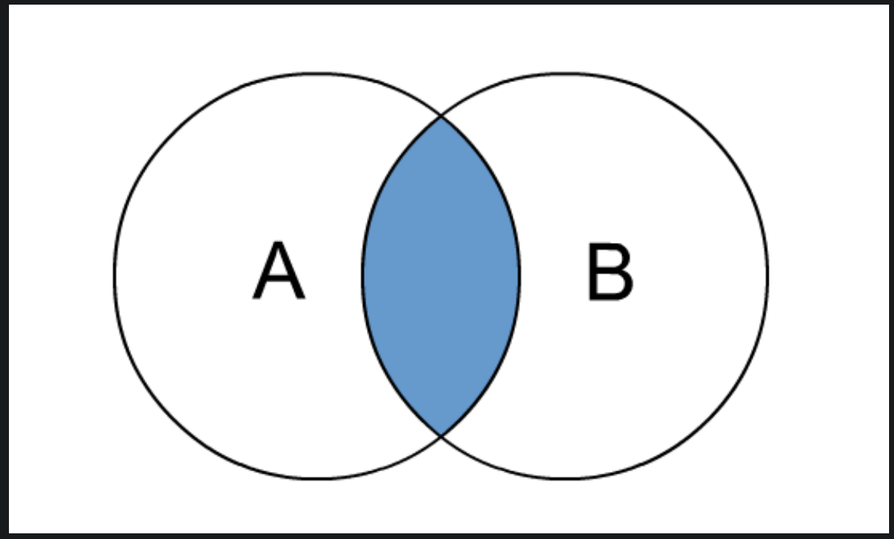
```
person_id | first_name
----------+-----------
 1        | Mike
 2        | Emily # no contact record
```

Contact:

```
contact_id | person_id | email
-----------+-----------+--------------------------
 1         | 1         | mike@dummy.com
 2         | 3         | edward@dummy.com
```

# INNER JOIN

Rows matching the condition of the join will be returned:



# INNER JOIN - Example

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

If we ran that on the sample data on the previous slide, what might we get?

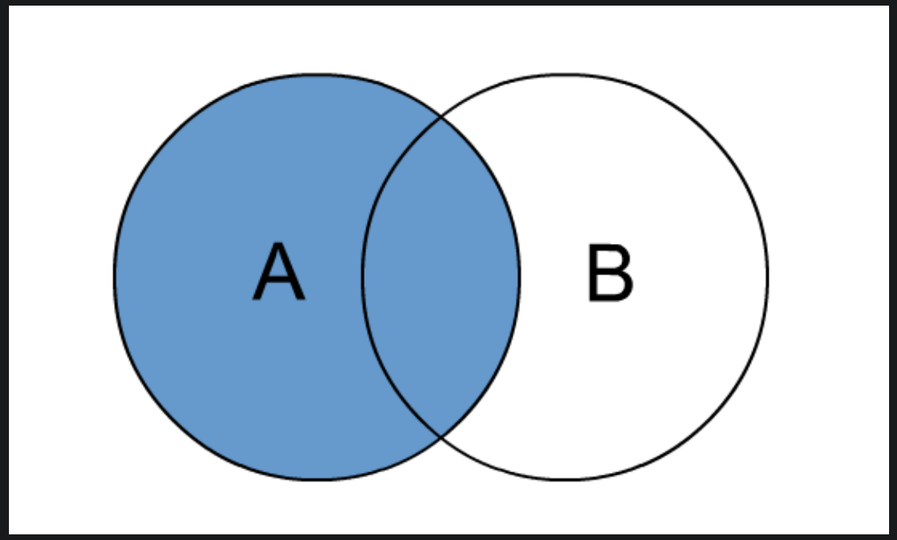
# Answer

Only rows with matching ids in both tables are returned:

```
 first_name | email
------------+----------------
 Mike       | mike@dummy.com
```

# LEFT OUTER JOIN

Returns all the data from A, as well as the matching data in B:



# What will this return?

Given some of the sample data we saw on previous slides, what does this give us?

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

It gives us the full set of people, with blanks where the contacts don't match, and no Edward as he only has Contact info:

```
 first_name | email
------------+-----------------
 Mike       | mike@dummy.com
 Emily      |
```

# How about this?

Given some of the sample data we saw on previous slides, what does this give us?

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

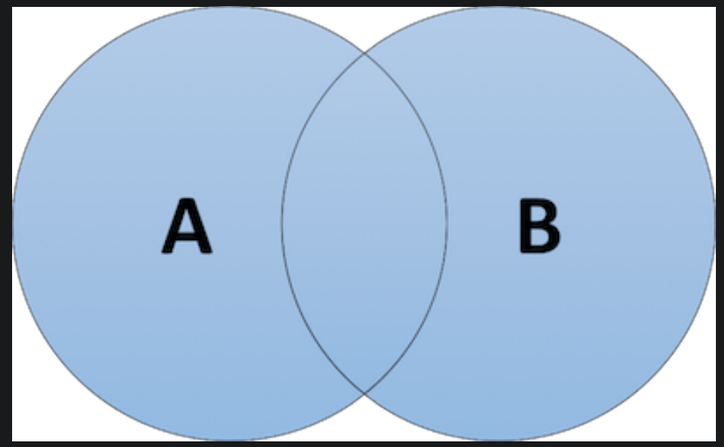
# RIGHT OUTER JOIN

It gives us everything from the RHS (contact_info) with blanks from missing People (so no Emily):

```
 first_name | email
------------+------------------
 Mike       | mike@dummy.com
            | edward@dummy.com
```

# FULL OUTER JOIN

Returns everything from both tables, regardless of if they match:



Given some of the sample data we saw on previous slides, what does this give us?

```
SELECT p.first_name, c.email
FROM person p
FULL OUTER JOIN contact_info c ON p.person_id = c.person_id;
```

It gives us all the rows with blanks inserted where there are no matches:

```
 first_name | email
------------+-------------------
 Mike       | mike@dummy.com
            | edward@dummy.com
 Emily      |
```

# Task - 10 mins

- Use some JOINs to get data out of your database from more than one table at a time
- Make a file named e.g. `join_<table_name>.sql` in VSCode and write your SQL in there - this will show you context highlighting
- When done, copy-and-paste the SQL into PSQL to check it works
- Share the files with your group

# Emoji Check:

How do you feel about writing JOIN statements in SQL?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

# Functions

Many built-in functions exist for each RDBMS you're using.

Some useful functions you may want to use:

- `SUBSTRING`
- `AVERAGE`, `MIN`, `MAX`
- `CURRENT_DATE`
- `COUNT`
- `UPPER`
- `LOWER`

# Some examples

```
SELECT substring(first_name, 1, 3) FROM person;

SELECT substring(first_name FROM 1 FOR 3) FROM person;
```

Gives us:

```
 substring
-----------
 Mik
 Emi
```

School of Tech
part of accenture

# Task - groups - 5 mins

- Use SUBSTRING in a query
- Use UPPER or LOWER in the same query as the SUBSTRING

School of Tech
part of accenture

# Emoji Check:

How do you feel about functions in SQL?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

# Task: Getting the Python exercise files

> For this part we will use the `exercises`, `handouts` and `solutions` folders.
>
> Please check you have them all.

# Connecting to Postgres from Python

We will need our python code to:

- Make a connection to the database
- Query the data from it with a `SELECT` statement (or, later, `INSERT`, or `UPDATE`)
- Present the results to us on-screen

# Connecting postgres from Python

> What data might the python code need in order to connect to the database?

- Address of the server
- Port of the server
- Name of the database
- Optionally - the name of the table

These are usually passed in as environment variables, so that sensitive information is never saved in our code repository.

# SQL in Python

The library `Psycopg` is a pure-Python PostgreSQL client library - we can use it to connect and run our SQL, i.e. allow us to interact with a PostgreSQL database purely through Python.

We do not need install this separately, it is in the `"requirements.txt"` file already.

# Sample requirements.txt

A sample is pre-prepared for us.

> Let's look inside `exercises/requirements.txt`

This contains a sample set of useful libraries we will use.

The binary version is included in case anyone's machine has issues with the raw version.

# Code-along - requirements

- Open a terminal in folder `exercises`
- Activate an existing venv or new virtual env
  - (see the "python-ecosystem" session for details)
- Install the dependencies from `requirements.txt` with:
  - `python3 -m pip install -r requirements.txt` (MacOS / Unix / GitBash);
  - `python -m pip install -r requirements.txt` (Windows)

# Using env vars

> We do not want sensitive information like connection details or passwords in our python files!
>
> Nor in our Git repositories in any other files!

A special file called `.env` can be made with dummy values for our testing, which will have real values in production systems. We then `.gitignore` this file.

In production systems, these values can be created without them ever being saved in our code.

# Using env vars

A standard way to write our code is use the `dotenv` utility to load them from the `.env` file. They can then be accessed in our application like so:

```
import os
from dotenv import load_dotenv

# Load environment variables from .env file
load_dotenv()
host_name = os.environ.get("POSTGRES_HOST")
database_name = os.environ.get("POSTGRES_DB")
user_name = os.environ.get("POSTGRES_USER")
user_password = os.environ.get("POSTGRES_PASSWORD")
```

# Making a connection

The `psycopg` utility has a `connect()` function for creating a connection, which takes a formatted string, like so:

```
import psycopg2 as psycopg
# Other imports
connection = psycopg.connect(f"""
    host={host_name}
    dbname={database_name}
    user={user_name}
    password={user_password}
    """)
```

(This code assumes we have the variables `host_name, database_name, user_name, user_password` from the previous slide.)

School of Tech
part of accenture

# Making a connection "properly"

We should use a context for the connection, like so:

```
import psycopg2 as psycopg
# Other imports
try:
    with psycopg.connect(f"""
        host={host_name}
        dbname={database_name}
        user={user_name}
        password={user_password}
        """) as connection:
        # More code here
        ...

        # The connection will automatically close here
except Exception as ex:
    print('Failed to:', ex)
```

# Opening a cursor

A cursor is an object that lets us manipulate the database, in the context of a database operation:

The function to create one is on the `connection` object, like so:

```
cursor = connection.cursor()
```

We can use this to run queries (SQL) on the database.

School of Tech
part of accenture

# An insert query

For inserting data, we give the `execute()` function some SQL with placeholders in it, and a tuple of all the data, like so:

```
sql = """
    INSERT INTO table_name (column_1, column_2, numeric_3)
    VALUES (%s, %s, %s)
    RETURNING primary_id
"""
data_values = ('value_1', 'value_2', 1234)

cursor.execute(sql, data_values)
```

The `execute()` function will do string substitutions for us in order - but we need to match the correct number of columns in the INSERT clause, VALUES clause, and tuple.

# A bulk insert query

For this we have to use an optional extra from the psycopg2 library called 'execute_values()'; It works a little differently:

```
from psycopg2.extras import execute_values

insert_sql = 'INSERT INTO pets (name) VALUES %s;'
list_of_tuples = [ ('Fido',), ('Catto',)]

execute_values(cursor, insert_sql, list_of_tuples, page_size=1
```

Note that the SQL has a single placeholder `%s` for the whole list.

# Logging the RETURNed value

If we execute a cursor and have used the RETURNING keyword in the SQL, we can find the data that is sent back to us:

```
...
cursor.execute(sql, data_values)
rows = cursor.fetchall()
print('insert result ids = ', rows[0])
```

# Committing the data

"Committing" in database terms means "save any updates".

In code we need to do this ourselves, like so:

```
connection.commit()
```

# Selecting data

To select data we also use a cursor. This will return all the matching data to us:

```
cursor.execute('SELECT column_1, column_2, numeric_1 FROM tabl
rows = cursor.fetchall()
```

We can also use any other valid clauses like `WHERE`, `ORDER BY`, `LIMIT`, and SQL functions.

# Select results

The result we get (`rows` here) will be an array with an entry per row.

Each row will be a tuple of the column values, e.g:

```
rows = [
    ('Jane', 'Bloggs', 32),
    ('Alice', 'Babbs', 45),
]
```

...and so on. The data types in the tuple will depend on that of the columns (i.e. strings, numbers, booleans).

# Tidying up

When we are done with a cursor or connection, we need to close each of them to release the resources they are holding onto:

```
# Closes the cursor so it will be unusable from this point
cursor.close()
```

We only need to manually close the connection if we didn't use a context:

```
# Closes the connection to the DB, ALWAYS do this
connection.close()
```

# Demo time - standard table

> There are some SQL files pre-loaded into your databases - these can be found at `exercises/db-scripts/*.sql`

- These create a table called `person`
- And some sample data INSERTed
- And a SELECT statement to show us the data

If you don't have a table matching this from when we started your postgres database, please run all this sql in the `adminer_container` we also started up. Browse to http://localhost:8080 and ask your instructor for the credentials to log in.

# Demo time - skeleton file

> The instructor will show you the file

- A template file `exercises/my_db_app.py` is provided
  - It already has the imports
  - It already loads the environment variables from an `.env` file
- The file doesn't do anything... yet!
  - It won't compile until it is filled in
  - It contains some `TODO` sections for you to complete

# Exercise - groups: Connect to DB - 10 mins

> Connect to the database

- Edit the `my_db_app.py` file
  - Find the line `# TODO Establish a database connection`
  - Create a connection with a context block
- Run your file to test it, e.g.:
  - `python3 -m my_db_app` (MacOS / Unix / GitBash)
  - `python -m my_db_app` (Windows)

School of Tech
part of accenture

# Discussion

> How did you get on connecting to the database?

# Exercise - groups: Open cursor - 5 mins

> Create a cursor

- Edit the `my_db_app.py` file
  - Find the line `# TODO Add code here to open a cursor`
  - Create a cursor
- Run your file to test it, e.g.:
  - `python3 -m my_db_app` (MacOS / Unix / GitBash)
  - `python -m my_db_app` (Windows)

# Discussion

> How did you get on creating a cursor?

School of Tech
part of accenture

# Exercise - groups: INSERT data - 10 mins

> INSERT some data

- Edit the `my_db_app.py` file
  - Find the line `# TODO Add code here to insert a new record (use RETURNING)`
  - Execute a statement to add some data
  - Try to log the ID that is returned
  - Also find line `# TODO Add code here to commit the INSERT`
  - commit the cursor
- Run your file to test it, e.g.:
  - `python3 -m my_db_app` (MacOS / Unix / GitBash)
  - `python -m my_db_app` (Windows)

# Discussion

> How did you get on inserting some data?

# Exercise - groups: SELECT all data - 10 mins

> SELECT all data

- Edit the `my_db_app.py` file
  - Find the line `# TODO Add code here to select all the records`
  - Execute a statement to select all data
- Run your file to test it, e.g.:
  - `python3 -m my_db_app` (MacOS / Unix / GitBash)
  - `python -m my_db_app` (Windows)

# Discussion

> How did you get on selecting some data?

# Exercise - groups: Print all data - 5 mins

> Print out all data

- Edit the `my_db_app.py` file
  - Find the line `# TODO Add code here to print out all the records`
  - Iterate the rows of data and log each one
- Run your file to test it, e.g.:
  - `python3 -m my_db_app` (MacOS / Unix / GitBash)
  - `python -m my_db_app` (Windows)

School of Tech
part of accenture

# Discussion

> How did you get on displaying the data?

# Exercise - groups: Tidy up - 5 mins

> Close the cursor

- Edit the `my_db_app.py` file
  - Find the line `# TODO close the cursor`
  - Close the cursor
- Run your file to test it, e.g.:
  - `python3 -m my_db_app` (MacOS / Unix / GitBash)
  - `python -m my_db_app` (Windows)

# Discussion

> How did you get on closing the cursor?

# Optional extra task

> In your own project / exercise time, change your code to do a bulk insert with 'execute_values()'.

# Terms and Definitions - recap

Database: An organised 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.

# Objectives - recap

- Learnt about the features of a database
- Learnt why we need databases
- Got introduced to the differences between relational and non-relational
- Saw some examples of SQL and learn about DDL and DML
- Learned, and practiced using, SQL commands to interact with a database
- Learned, and practiced using, Python to interact with a database

School of Tech
part of accenture

# Emoji Check:

On a high level, do you think you understand the main concepts of this session? Say so if not!

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively