SQLite Databases

By Sindya and Hai De

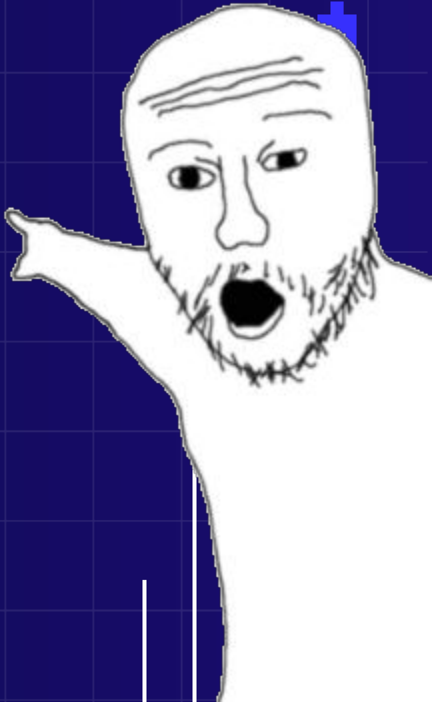
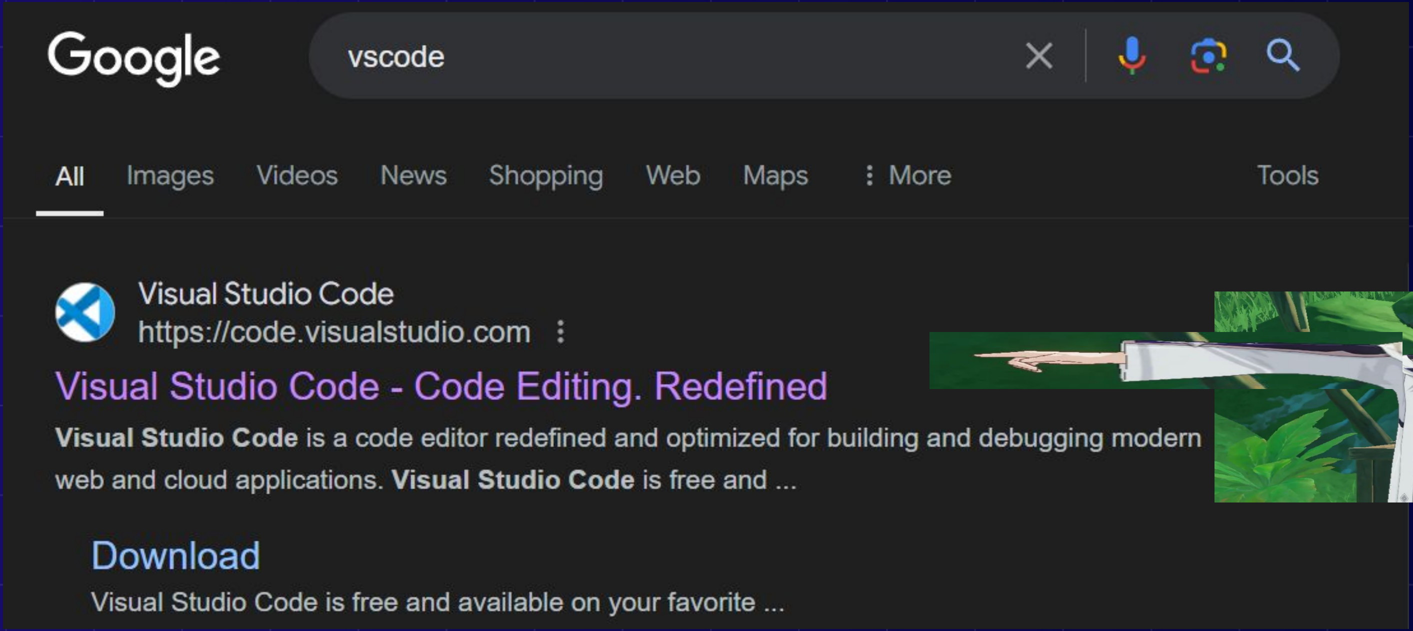


TABLE OF CONTENTS

01

Set Up :)

download vscode from website



vscode

Visual Studio Code
https://code.visualstudio.com

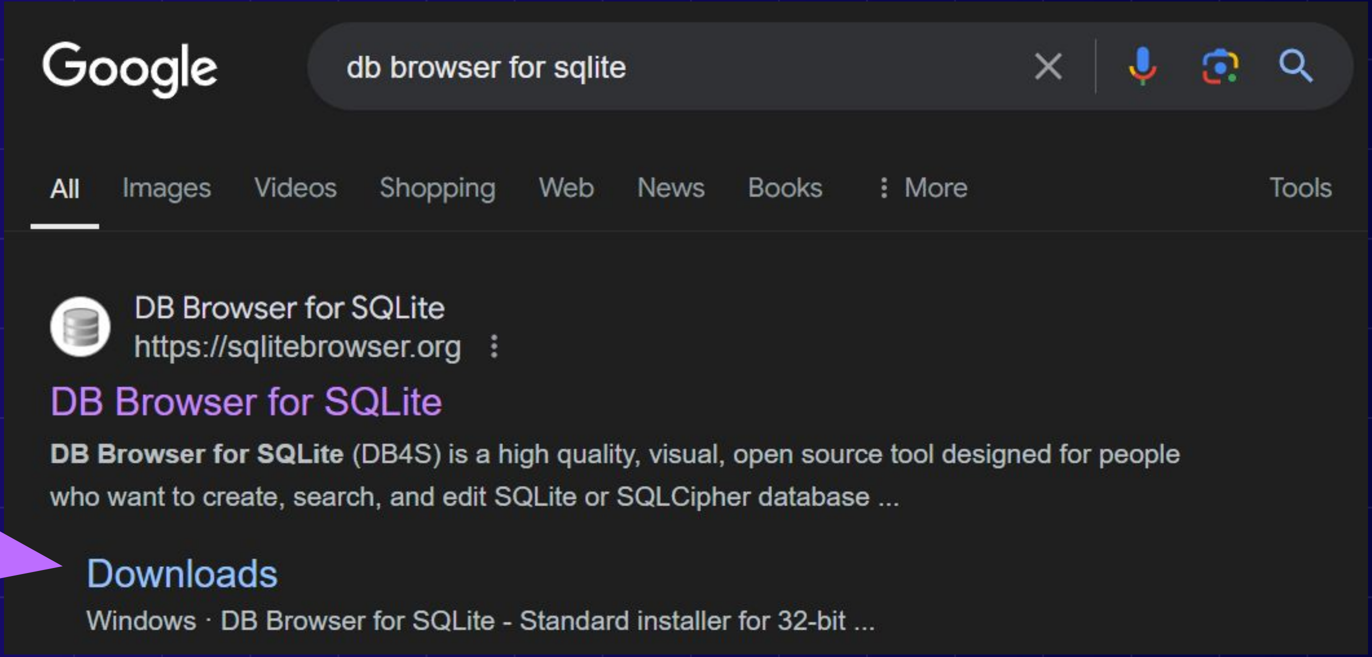
Visual Studio Code - Code Editing. Redefined

Visual Studio Code is a code editor redefined and optimized for building and debugging modern web and cloud applications. **Visual Studio Code** is free and ...

Download

Visual Studio Code is free and available on your favorite ...

download db browser from website



DB Browser for SQLite
https://sqlitebrowser.org

DB Browser for SQLite

DB Browser for SQLite (DB4S) is a high quality, visual, open source tool designed for people who want to create, search, and edit SQLite or SQLCipher database ...

Downloads

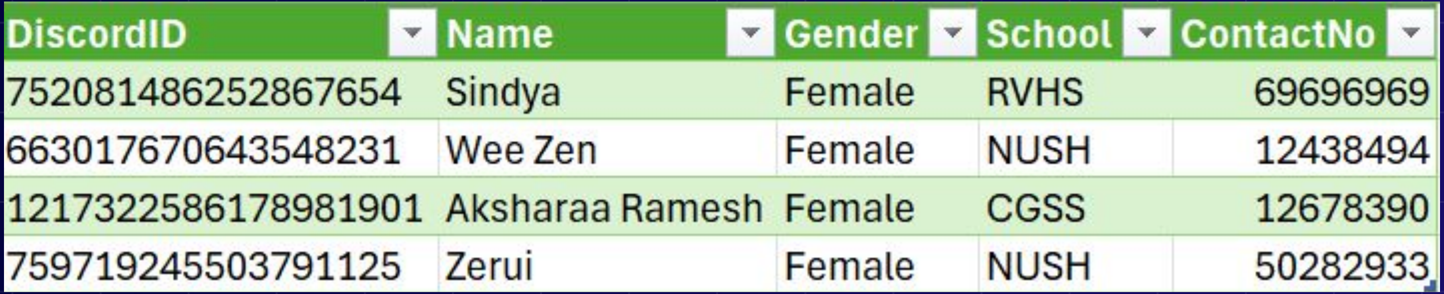
Windows · DB Browser for SQLite - Standard installer for 32-bit ...

02

Intro to Databases

Databases

A database is an organized collection of data, generally stored and accessed electronically from a computer system.



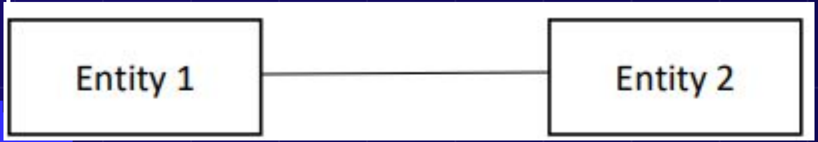
DiscordID	Name	Gender	School	ContactNo
752081486252867654	Sindya	Female	RVHS	69696969
663017670643548231	Wee Zen	Female	NUSH	12438494
1217322586178981901	Aksharaa Ramesh	Female	CGSS	12678390
759719245503791125	Zerui	Female	NUSH	50282933

Databases

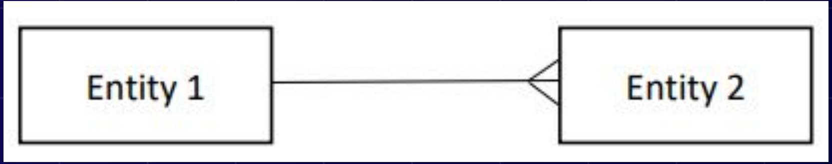
- ★ A widely used technique for designing database is the use of entity – relationship diagram to establish from who and what kind of data are collected.
- ★ A database model can then be chosen to determine the logical structure of a database how they can be stored, organised, and manipulated.

Entity-Relationship diagram

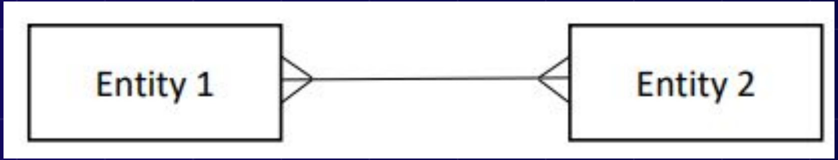
One to one



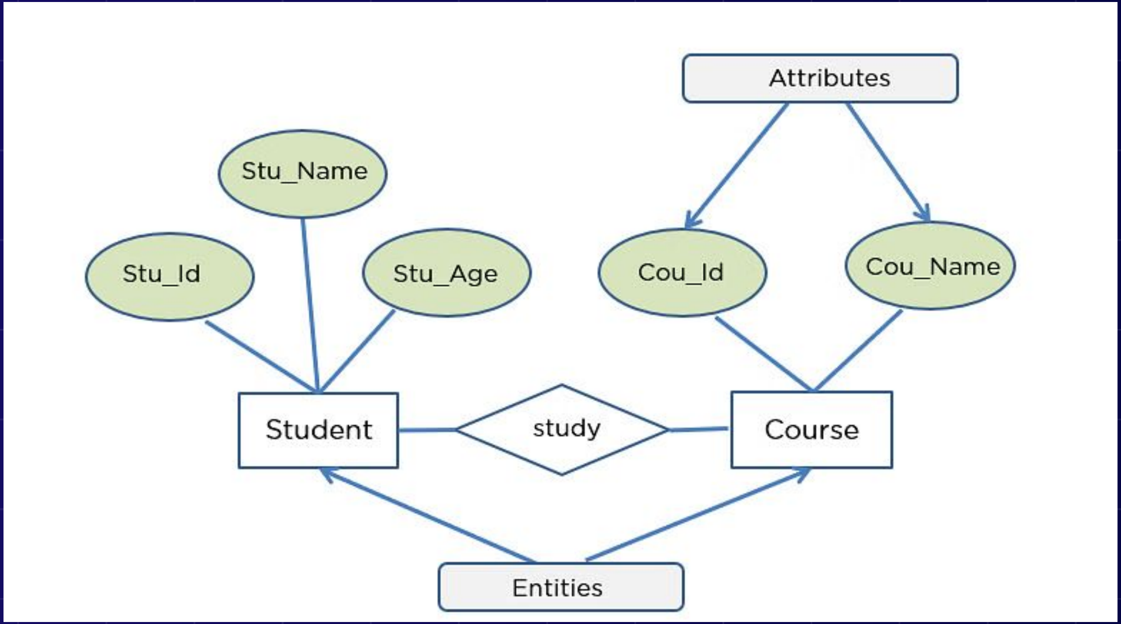
One to many



Many to many



Entity-Relationship diagram



Databases

★ The 4S can be used to describe the characteristics of database models: structure, size, speed, scalability.

★ DBMS is a software package needed to manipulate data in a database. It has ACID properties which will ensure that database transactions are processed reliably.

ACID properties

ACID

★ stands for Atomicity, Consistency, Isolation, Durability

★ is a standard set of properties that guarantee database transactions are processed reliably

More on ACID

Why is ACID needed?

- ★ ACID properties are designed as principles of transaction-oriented database recovery.
- ★ ACID provides the principles that database transactions should adhere to, to ensure that data doesn’t become corrupt as a result of a failure of some sort.

More on ACID

ACID definition

ACID will ensure that only successful transactions are processed. If a failure occurs before a transaction completes, no data will be changed.

$\downarrow$

maintains data integrity

Relational Databases

Relational databases are tables which are linked together, making them easier to store, modify and retrieve.

Relational Database model

In the relational database model, data is stored in relations and represented in the form of rows and collected in a table. A relational database is a collection of relational tables.

Relational Database model

A table description:

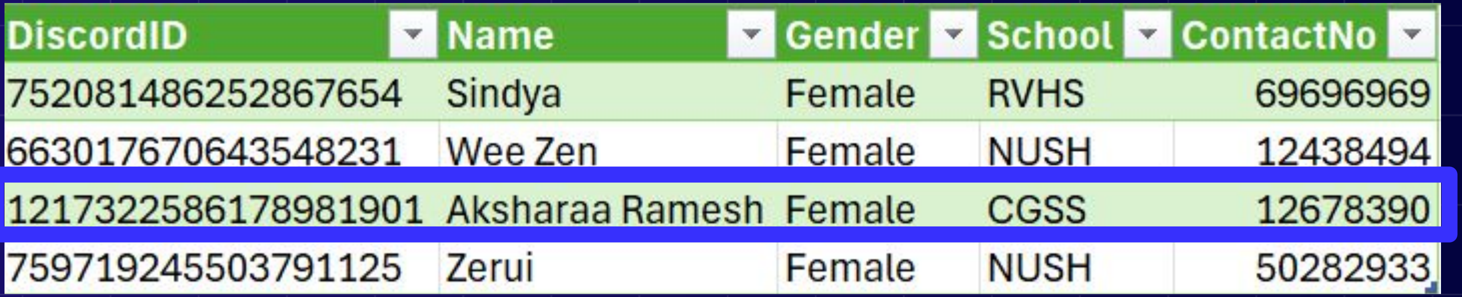
Tablename (Attribute1, Attribute2, Attribute3..)
For example,
Buildingblocs (DiscordID, Name, Gender, School, ContactNo)

Relational Database model

A table is a relational database if:

- ★ Values are atomic (cannot be divided)
- ★ Columns are of the same kind
- ★ Rows are unique
- ★ The order of columns is insignificant
- ★ Each column must have a unique name

Relational Database model

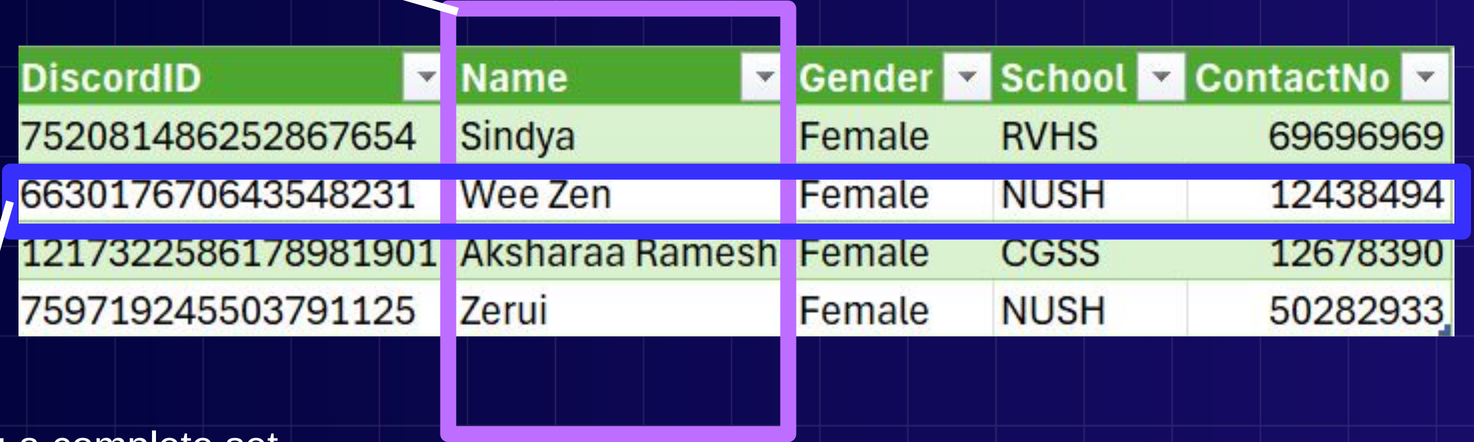


DiscordID	Name	Gender	School	ContactNo
752081486252867654	Sindya	Female	RVHS	69696969
663017670643548231	Wee Zen	Female	NUSH	12438494
1217322586178981901	Aksharaa Ramesh	Female	CGSS	12678390
759719245503791125	Zerui	Female	NUSH	50282933

And that’s where SQLite comes in!

Try relating it to Excel!

Field: an attribute of a record in a database table

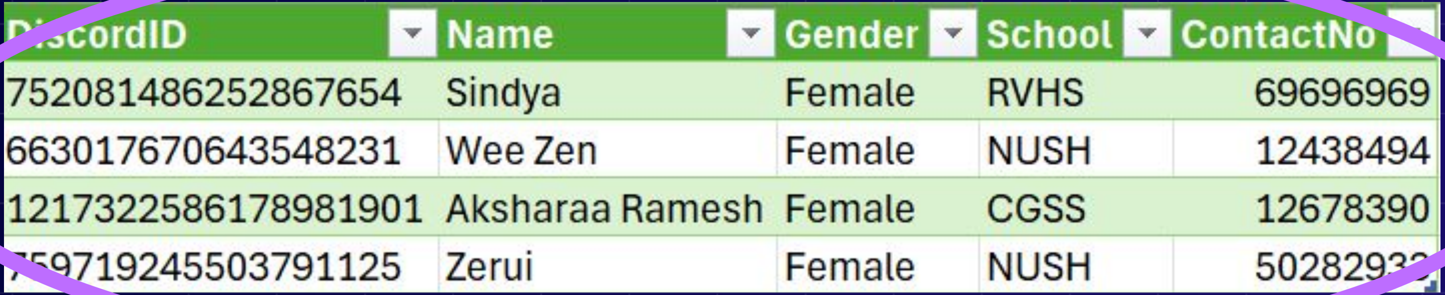


DiscordID	Name	Gender	School	ContactNo
752081486252867654	Sindya	Female	RVHS	69696969
663017670643548231	Wee Zen	Female	NUSH	12438494
1217322586178981901	Aksharaa Ramesh	Female	CGSS	12678390
759719245503791125	Zerui	Female	NUSH	50282933

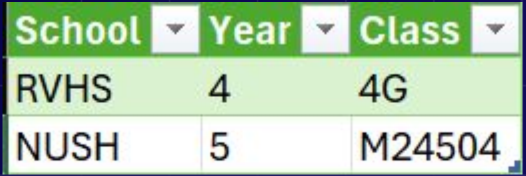
Record: a complete set of data about a single item

Test your understanding: How many records does the table above have?

Tables



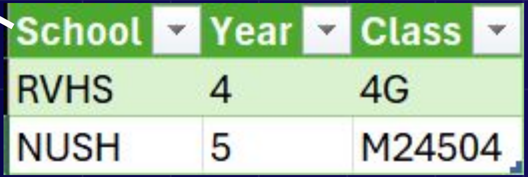
DiscordID	Name	Gender	School	ContactNo
752081486252867654	Sindya	Female	RVHS	69696969
663017670643548231	Wee Zen	Female	NUSH	12438494
1217322586178981901	Aksharaa Ramesh	Female	CGSS	12678390
759719245503791125	Zerui	Female	NUSH	50282932



School	Year	Class
RVHS	4	4G
NUSH	5	M24504

More terms

This is another table about our schools. “School” is chosen as the primary key for this table, and it can be linked to “school” in the next table as a foreign key

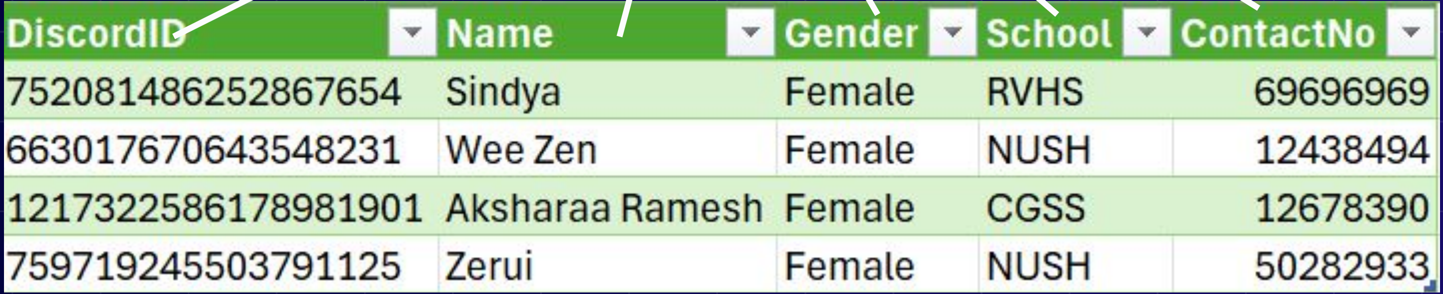


School	Year	Class
RVHS	4	4G
NUSH	5	M24504

A foreign key is an attribute (field) in one table that refers to the primary key in another table

More terms

Candidate keys: should never be null or empty, can have more than one and can be a combination of more than one field



DiscordID	Name	Gender	School	ContactNo
752081486252867654	Sindya	Female	RVHS	69696969
663017670643548231	Wee Zen	Female	NUSH	12438494
1217322586178981901	Aksharaa Ramesh	Female	CGSS	12678390
759719245503791125	Zerui	Female	NUSH	50282933

Primary Key: main key, it uniquely identifies each record in a table and should not change over time

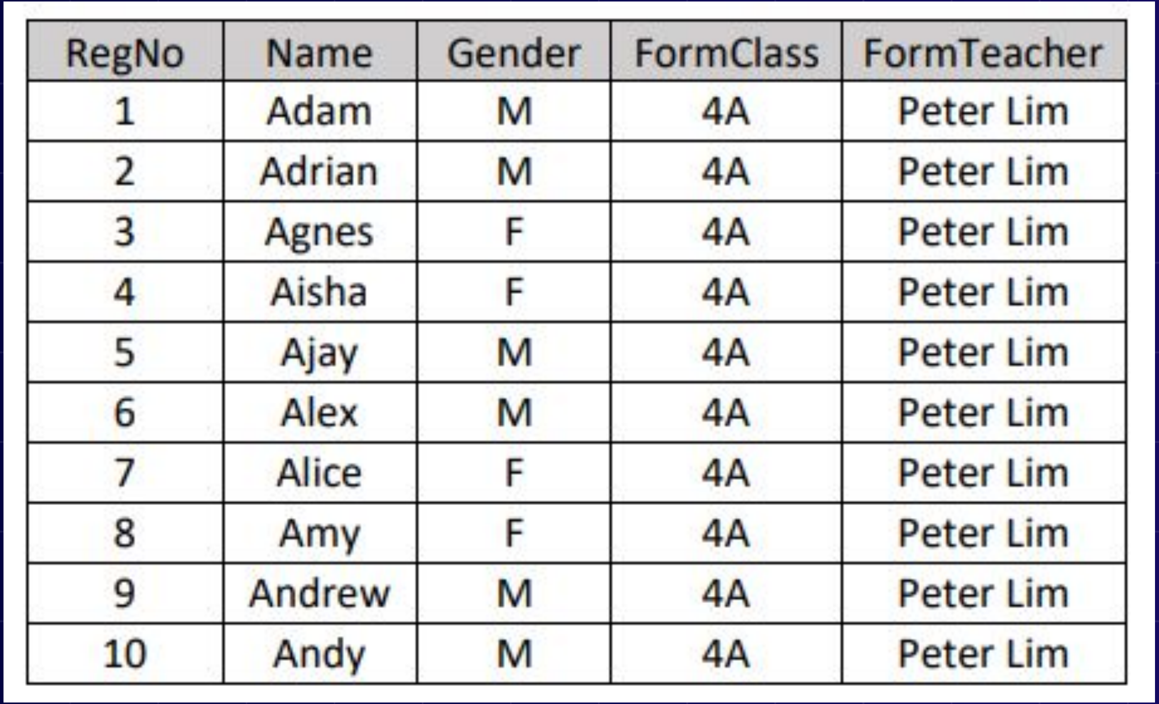
Secondary Key: candidate keys that are not selected as primary key

So, data.

Data Redundancy

Data redundancy refers to the same data being stored more than once. This will cause issues when inserting, updating and deleting data from the database.

Why is Data Redundancy an issue?



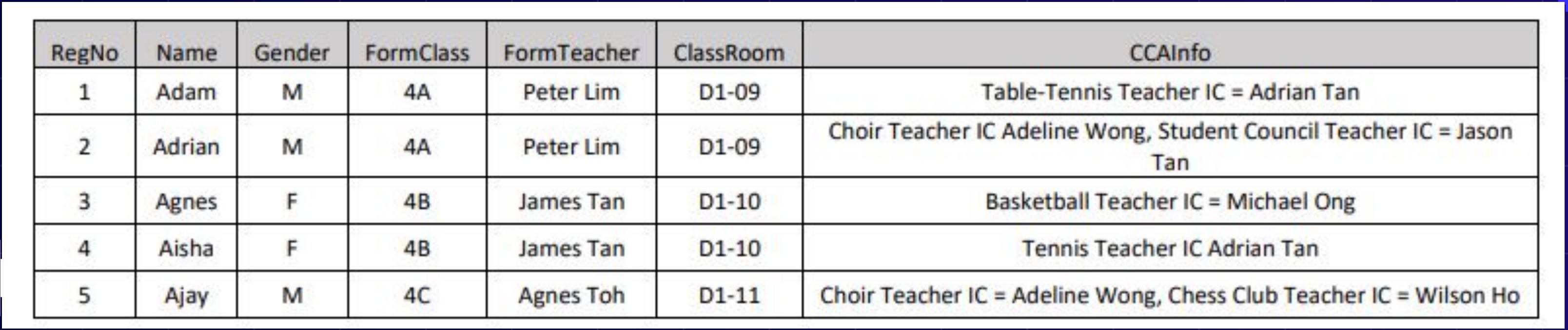
RegNo	Name	Gender	FormClass	FormTeacher
1	Adam	M	4A	Peter Lim
2	Adrian	M	4A	Peter Lim
3	Agnes	F	4A	Peter Lim
4	Aisha	F	4A	Peter Lim
5	Ajay	M	4A	Peter Lim
6	Alex	M	4A	Peter Lim
7	Alice	F	4A	Peter Lim
8	Amy	F	4A	Peter Lim
9	Andrew	M	4A	Peter Lim
10	Andy	M	4A	Peter Lim

Data Normalisation

Normalisation is the process of organising the tables in a database to reduce data redundancy and prevent inconsistent data.

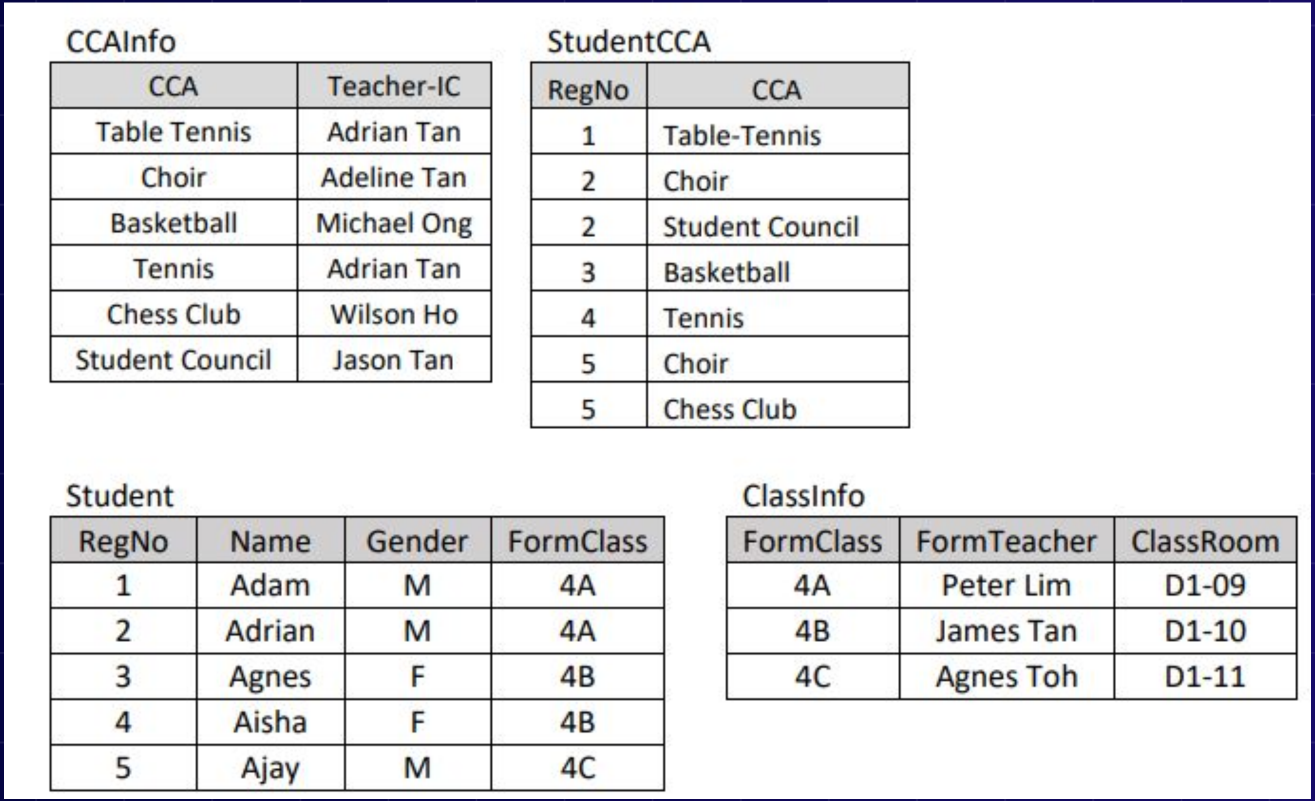
- all columns must be atomic
- every non-key attribute must be fully dependent on the entire primary key
- the table should not have transitive dependencies, which occurs when some non-key attribute determines some other attribute

consider this table:



RegNo	Name	Gender	FormClass	FormTeacher	ClassRoom	CCAInfo
1	Adam	M	4A	Peter Lim	D1-09	Table-Tennis Teacher IC = Adrian Tan
2	Adrian	M	4A	Peter Lim	D1-09	Choir Teacher IC Adeline Wong, Student Council Teacher IC = Jason Tan
3	Agnes	F	4B	James Tan	D1-10	Basketball Teacher IC = Michael Ong
4	Aisha	F	4B	James Tan	D1-10	Tennis Teacher IC Adrian Tan
5	Ajay	M	4C	Agnes Toh	D1-11	Choir Teacher IC = Adeline Wong, Chess Club Teacher IC = Wilson Ho

after normalisation:



CCAInfo

CCA	Teacher-IC
Table Tennis	Adrian Tan
Choir	Adeline Tan
Basketball	Michael Ong
Tennis	Adrian Tan
Chess Club	Wilson Ho
Student Council	Jason Tan

StudentCCA

RegNo	CCA
1	Table-Tennis
2	Choir
2	Student Council
3	Basketball
4	Tennis
5	Choir
5	Chess Club

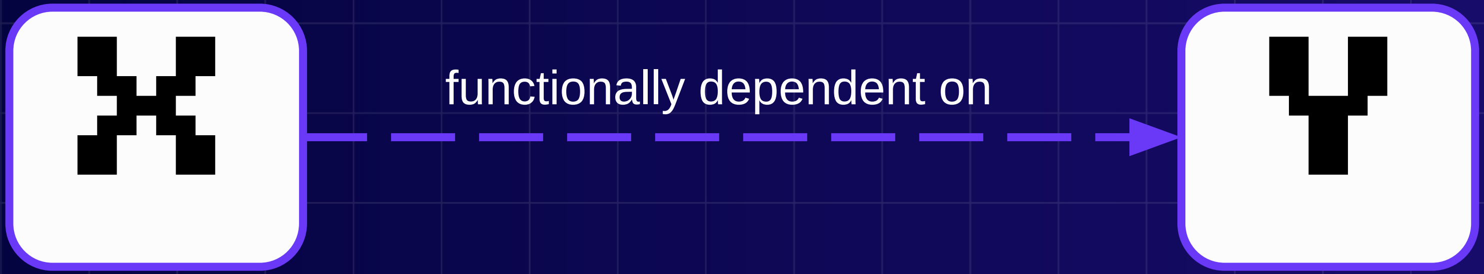
Student

RegNo	Name	Gender	FormClass
1	Adam	M	4A
2	Adrian	M	4A
3	Agnes	F	4B
4	Aisha	F	4B
5	Ajay	M	4C

ClassInfo

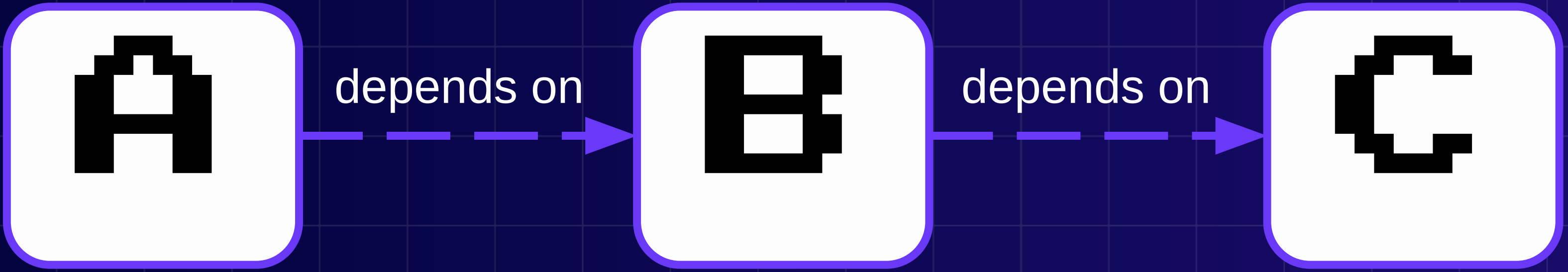
FormClass	FormTeacher	ClassRoom
4A	Peter Lim	D1-09
4B	James Tan	D1-10
4C	Agnes Toh	D1-11

Functional Dependency



for every valid instance of X, the value of X uniquely determines the value of Y

Transistive Dependency



A depends on B
A is transistively dependent on C

03

Intro to SQLite

Type affinities (yay)

Type affinities

INTEGER

TEXT

REAL

NUMERIC

BLOB

Integers

★ used to store a signed integer value

★ the value is stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value

Type affinities

INTEGER

TEXT

REAL

NUMERIC

BLOB

Text

★ Used to store a text string using the database encoding (UTF-8, UTF-16BE or UTF16LE)

Type affinities

INTEGER

TEXT

REAL

NUMERIC

BLOB

Real

★ Used to store a floating point value, as an 8-byte IEEE floating point number

Type affinities

INTEGER

TEXT

REAL

NUMERIC

BLOB

Numeric

★ A column with NUMERIC affinity may contain values using any of these five storage classes:

1. Integer
2. Float
3. NULL
4. BLOB
5. Text (stored with conversion)

Type affinities

INTEGER

TEXT

REAL

NUMERIC

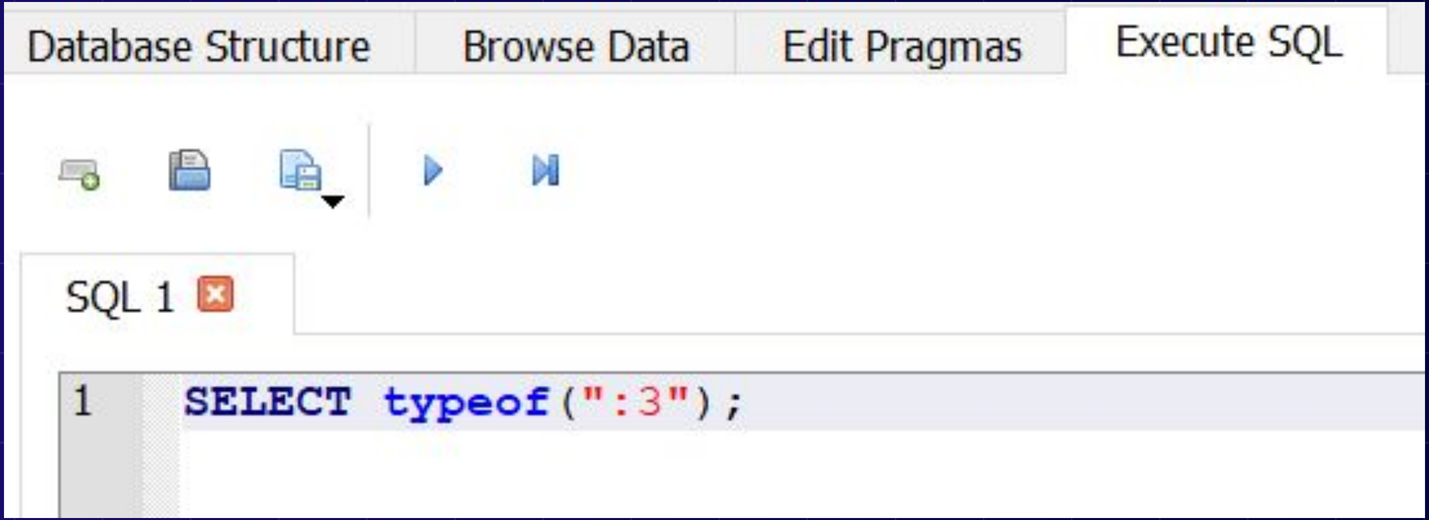
BLOB

Blob

★ BLOB stands for Binary Large Object (BLOB) which is used to store large binary data, such as images or multimedia in a database.

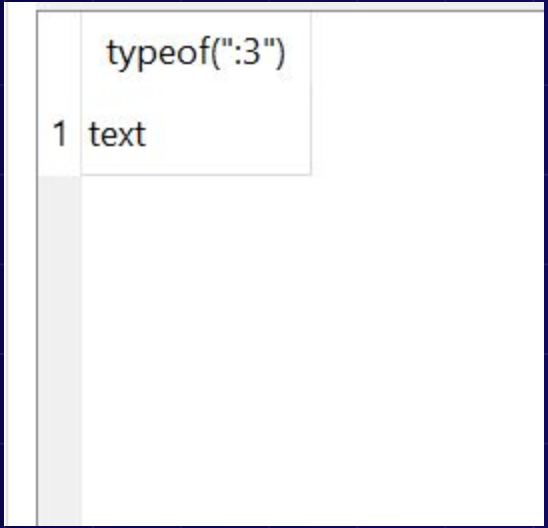
typeof() function

How to use



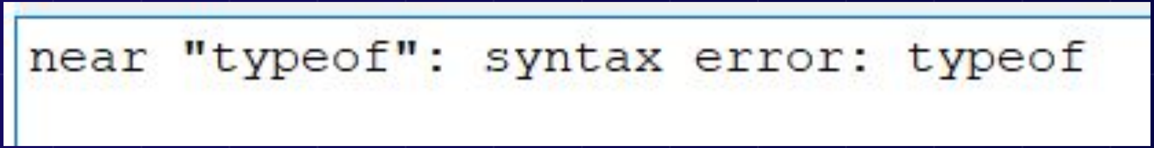
```
SELECT typeof(":3");
```

So what's the datatype of ":3"?



```
  typeof(":3")
1 text
```

What if you do typeof()?



```
near "typeof": syntax error: typeof
```

Statements

What are SQL statements?

An SQL statement is a command used to interact with a relational database, allowing you to perform actions like retrieving, adding, modifying, and deleting data within database tables.

These statements can be broadly classified into 4 different categories:

1. Data Query Language (DQL)
2. Data Manipulation Language (DML)
3. Data Definition Language (DDL)
4. Data Control Language (DCL)

Basic Statement Syntax/Commands

btw, SQL is not case-sensitive ;p
(i.e. **bruh** is the same as **BRUH**)

DDL

Data Definition Language

CREATE

CREATE

- ★ Create database or its objects (table, index, function, views, store procedure, and triggers)
- ★ Syntax:

CREATE TABLE table_name (column, data_type, …..)

ALTER

ALTER

- ★ Alters the structure of the database
- ★ Syntax:

```
ALTER TABLE table_name ADD COLUMN column_name
datatype;
```

DROP

DROP

- ★ Deletes a table from the database
- ★ Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!
- ★ Syntax:

```
DROP TABLE table_name
```

DQL

Data Query Language

SELECT

SELECT

- ★ Used to select data from a database
- ★ Query returns the data which fulfills the filters specified afterwards
- ★ Syntax:

```
SELECT column1, column2, …
FROM table_name;
```

Filtering in SELECT

BASIC OPERATORS

- ★ SELECT DISTINCT column_name FROM table_name → used to return only distinct (different) values
- ★ SELECT * FROM table_name → return all columns, without specifying every column name
- ★ SELECT COUNT(*) FROM table_name → find total number of columns
- ★ SELECT COUNT (DISTINCT column_name) FROM table_name → return number of unique columns

DML

Data Manipulation Language

INSERT

INSERT

- ★ Copies data from one table and inserts it into another table
- ★ It requires that the data types in source and target tables match
- ★ Syntax:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition
```

- ★ this is for when you only want to copy some columns from one table into another table:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, …
FROM table1
WHERE condition
```

UPDATE

UPDATE

- ★ Used to modify the existing records in a table
- ★ The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated.
- ★ Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, …
WHERE condition
```

DELETE

DELETE

- ★ Used to delete existing records in a table
- ★ Syntax:

```
DELETE FROM table_name WHERE condition
```

DCL

Data Control Language

GRANT

GRANT

- ★ Grants specific access rights to certain users
- ★ Syntax:

```
GRANT privilege ON table_name TO user
```

REVOKE

REVOKE

- ★ The opposite of GRANT
- ★ Revokes access rights to certain users
- ★ Syntax:

```
REVOKE privilege ON table_name FROM user
```

But we missed out a category…

do you all remember what we missed out :p
(smh why aren’t y’all paying attention)

Trick Question!

TCL

Transaction Control Language

COMMIT

COMMIT

- Permanently save any transaction into the DB
- Syntax:

```
COMMIT
```

ROLLBACK

ROLLBACK

- ★ Restores the database to last committed state; OR
- ★ Can be used with SAVEPOINT command to jump to a savepoint in a transaction
- ★ Syntax:

```
ROLLBACK
```

SAVEPOINT

SAVEPOINT

★ Temporarily save a transaction so that you can rollback to that point whenever necessary

★ Syntax:

```
SAVEPOINT
```

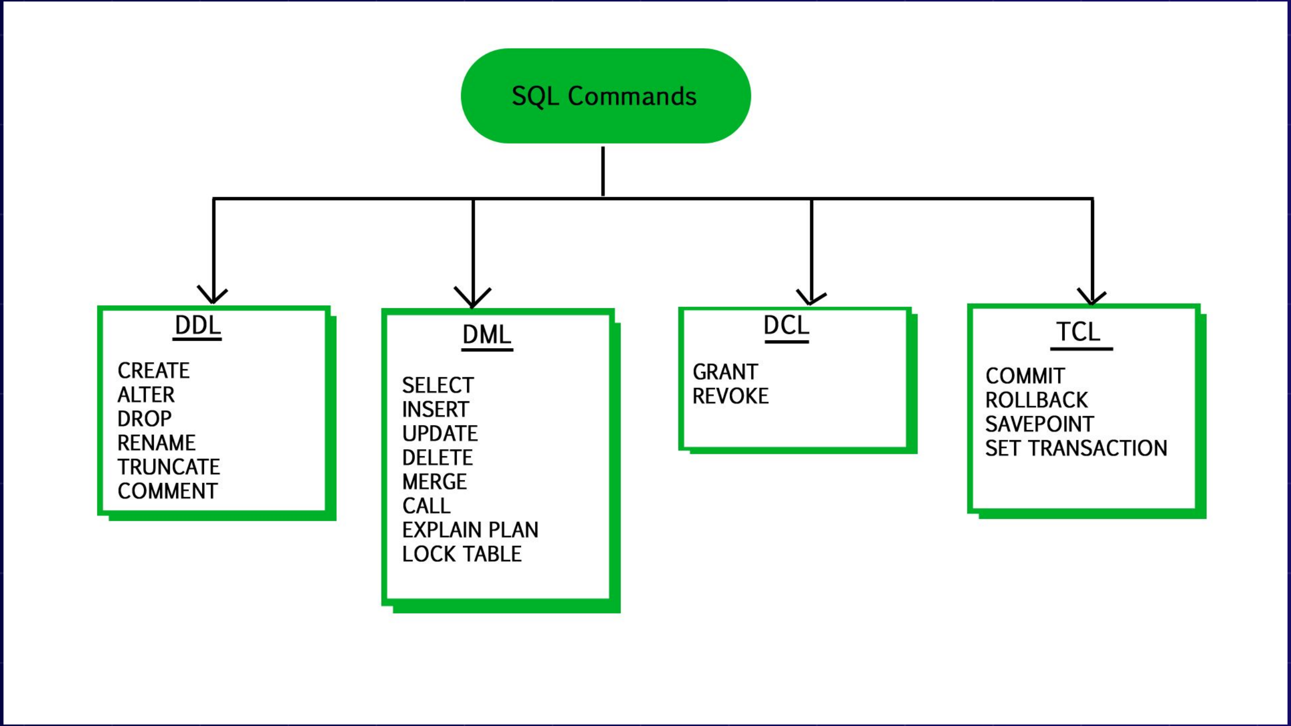
"So, when do I use COMMIT/ SAVEPOINTs?"

General Advice

- **COMMIT** when it is a “finalised” version, done with transaction
- **SAVEPOINT** as a temporary checkpoint that you can always rollback/ ”undo” to

*** when you **COMMIT**, the changes are **PERMANENTLY SAVED**, and **CANNOT** be directly reversed ***

Summary


SQL Commands
DDL
CREATE
ALTER
DROP
RENAME
TRUNCATE
COMMENT
DML
SELECT
INSERT
UPDATE
DELETE
MERGE
CALL
EXPLAIN PLAN
LOCK TABLE
DCL
GRANT
REVOKE
TCL
COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION

04

SQLite & Python + Activity!

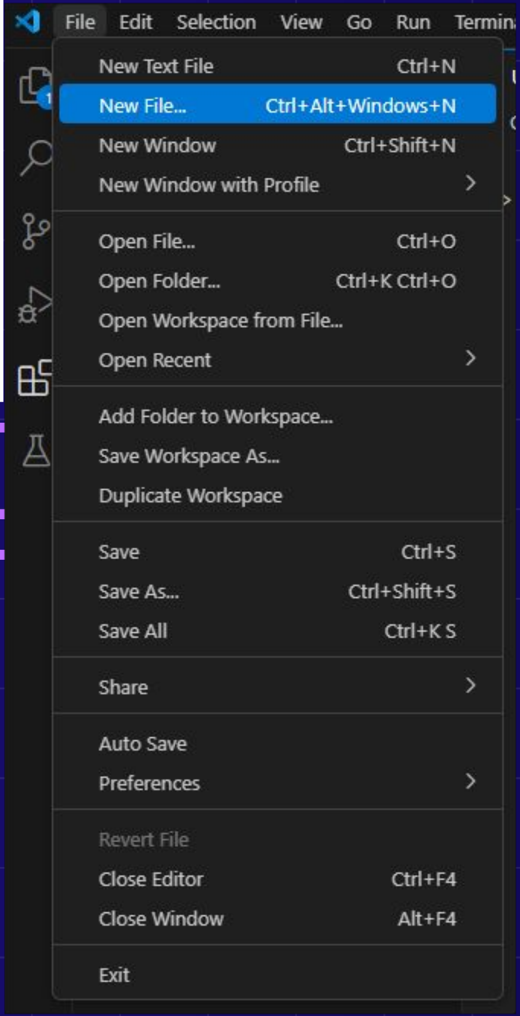
Set up the Environment

go to VSCode! (or any Python Environment)

VSCode

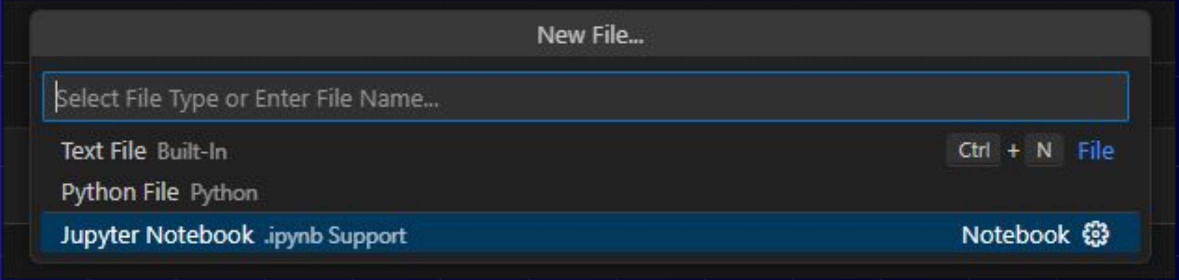
Open a new jupyter file!

VSCode



File Edit Selection View Go Run Termin

New Text File	Ctrl+N
New File...	Ctrl+Alt+Windows+N
New Window	Ctrl+Shift+N
New Window with Profile	>
Open File...	Ctrl+O
Open Folder...	Ctrl+K Ctrl+O
Open Workspace from File...	
Open Recent	>
Add Folder to Workspace...	
Save Workspace As...	
Duplicate Workspace	
Save	Ctrl+S
Save As...	Ctrl+Shift+S
Save All	Ctrl+K S
Share	>
Auto Save	
Preferences	>
Revert File	
Close Editor	Ctrl+F4
Close Window	Alt+F4
Exit	



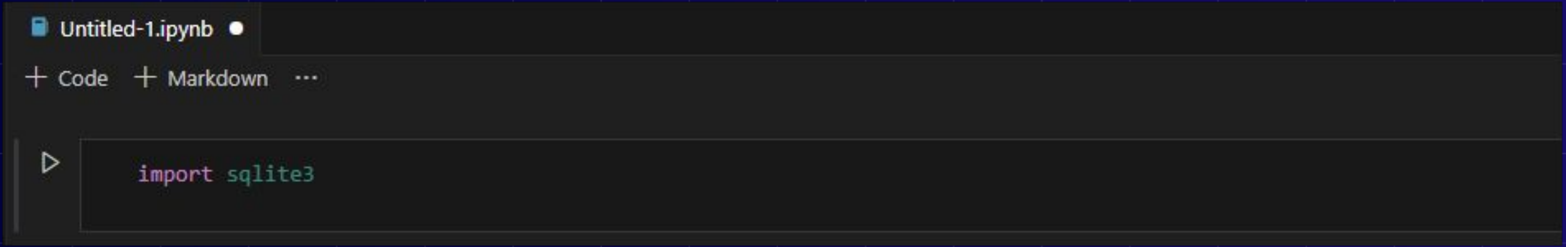
New File...

Select File Type or Enter File Name...

Text File Built-In	Ctrl + N File
Python File Python	
Jupyter Notebook .ipynb Support	Notebook

VSCode

import sqlite into vscode



```
import sqlite3
```

What is SQLite3?

Essentially, it is a built-in C-language library/module to work with SQLite databases in Python. (so yay, no need to install)

These are just the extremely basic syntax!!
Read more at https://www.sqlite.org/lang.html :)

Things to note!

- ★ remember to import sqlite3 first
- ★ try except are very useful (you will learn it as we go)
- ★ statements are the same as what we taught just now
- ★ cursor, importing csv into a database, how to execute an sqlite statement in python (you will also learn)

Let us begin!

Why import?

- ★ Your code won’t work if you don’t import sqlite3
- ★ Import allows you to use the sqlite3 module in python
- ★ So it’s very important or you can’t even use SQLite

Why Try Except?

- ★ The ***try*** block lets you test a block of code for errors.
- ★ The ***except*** block lets you handle the error.
- ★ The error can be handled by printing something in place of it. However, the error still exists.

How to use the sqlite3 module in Python

1. Connect to the database
2. Create tables in the database
3. Create a cursor object, and call execute on the SQL statements
4. Commit the transaction into the database
5. Close the database connection

Activity Time!

Oh no! Santa has fallen off his sleigh and injured his back! He has sustained a spinal cord injury, and has sent toys out to a factory in China operating on child labour elves in order to get the childrens’ Christmas gifts packed and delivered, while minimising cost!

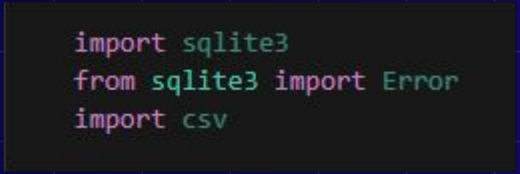
But oh no, the toys were messed up during shipping! There are some potential threats and contamination in the toys!

Your job as Santa's ~~child labourers~~ elves is to gather all the toys in the factory, categorise them and get rid of the threats, receive new orders for toys and finally get them ready to be gifted to the children! This is totally not unpaid labour!

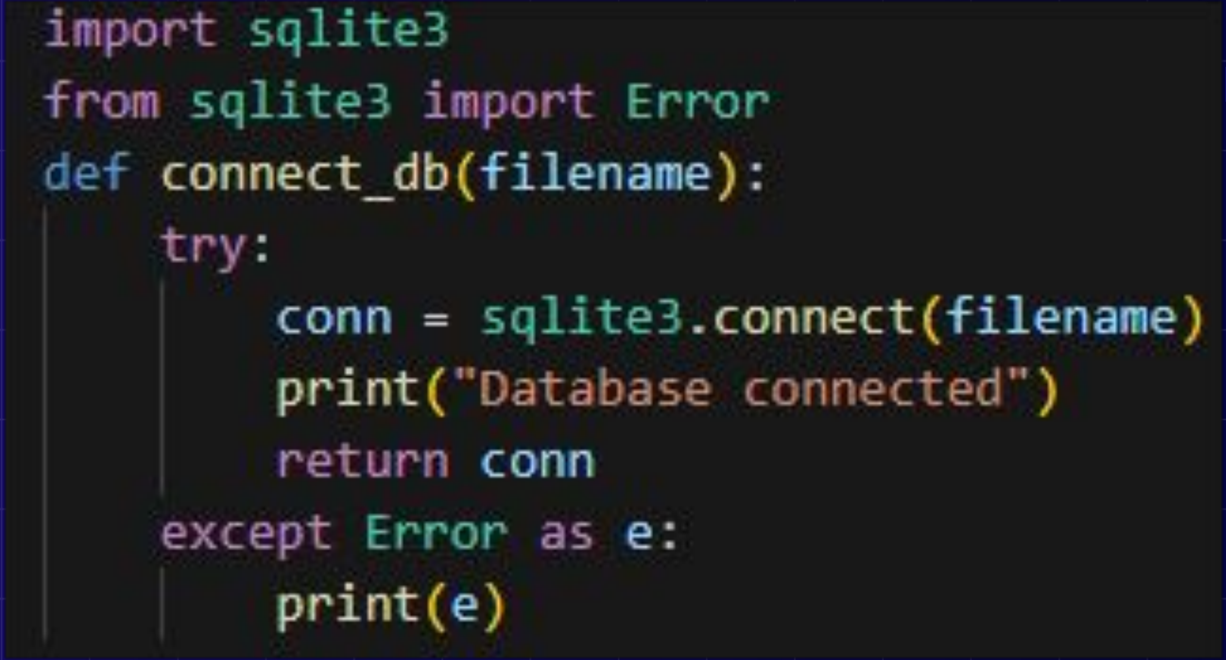
Connecting to the database

1. Import sqlite3 and Error (“error” is wrong)
2. try sqlite3.connect
3. except Error as e

Tip: just place sqlite3, Error and csv right at the beginning like this



```python
import sqlite3
from sqlite3 import Error
import csv
```



```python
import sqlite3
from sqlite3 import Error
def connect_db(filename):
    try:
        conn = sqlite3.connect(filename)
        print("Database connected")
        return conn
    except Error as e:
        print(e)
```

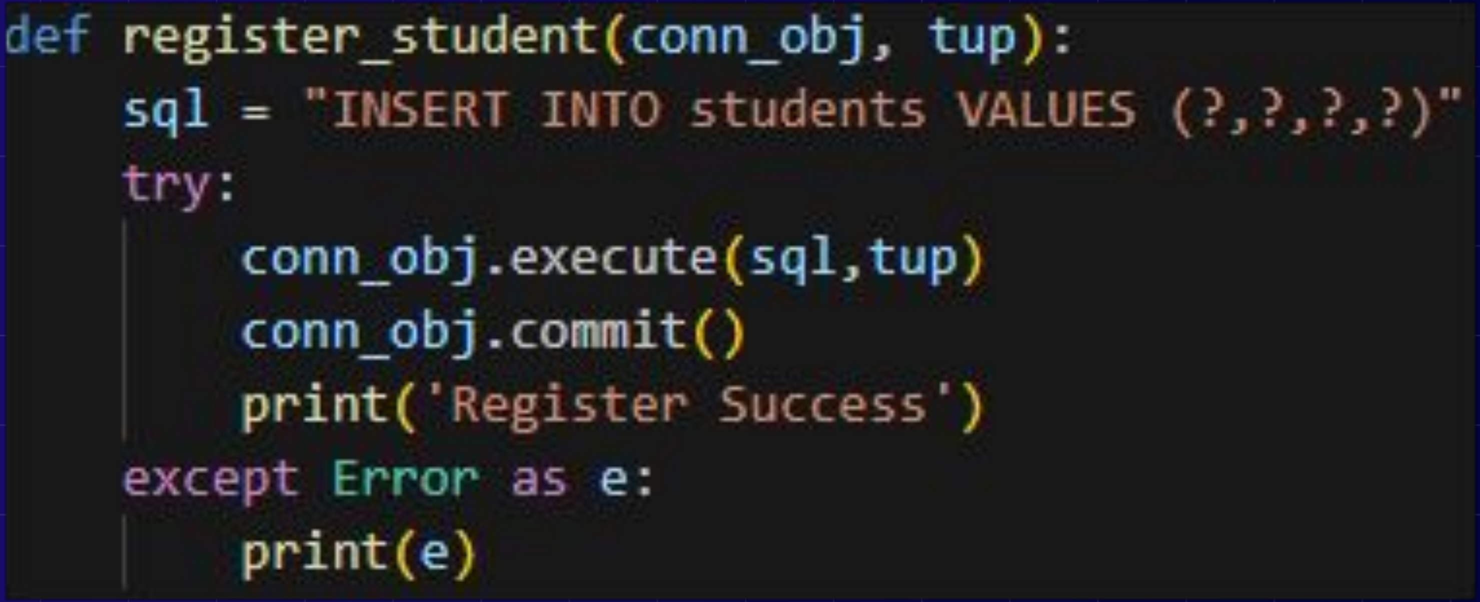
Task 1!

As his child labourers elves, your first task is to create a database, `santas_workshop` to compile all the toys that are supposed to be packed up and gifted to the children!

hint: recall what you learnt about connect db!

Inserting data

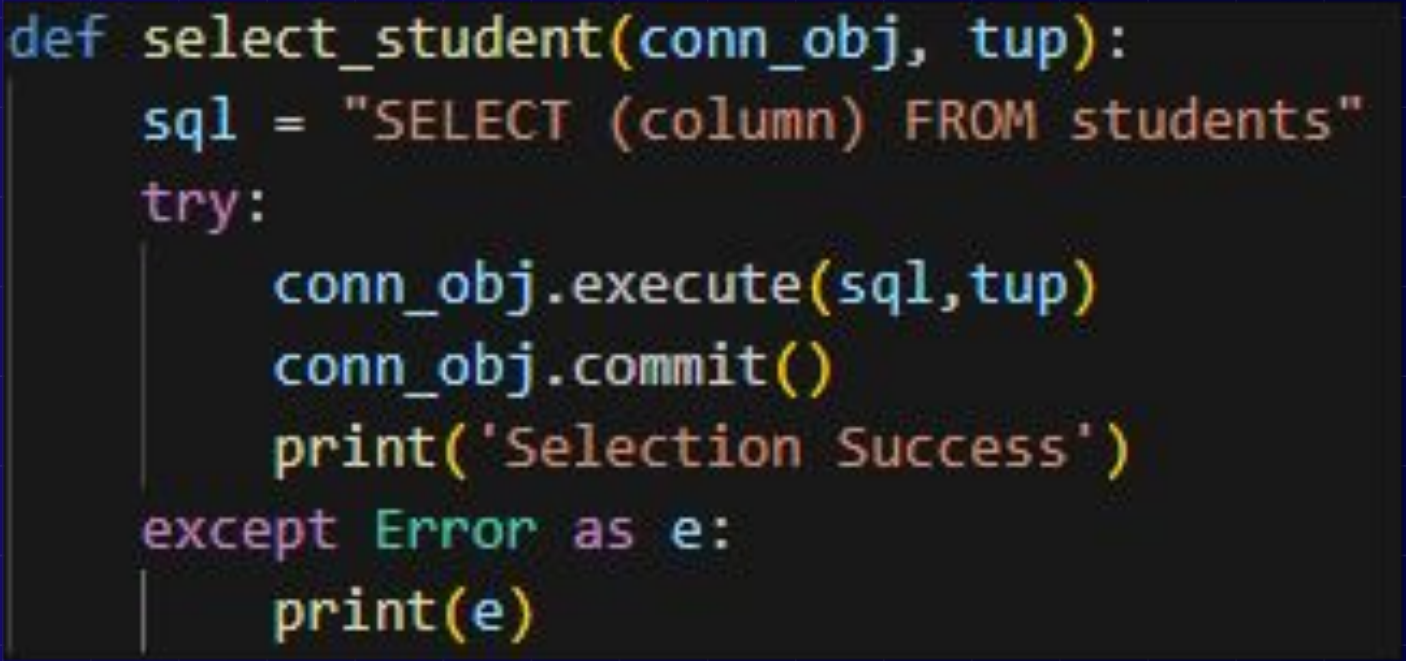
1. Use INSERT INTO {table_name} VALUES [...] statement (values are based on the number of fields in the table)
2. execute
3. commit
4. try except



```
def register_student(conn_obj, tup):
    sql = "INSERT INTO students VALUES (?,?,?,?)"
    try:
        conn_obj.execute(sql,tup)
        conn_obj.commit()
        print('Register Success')
    except Error as e:
        print(e)
```

Selecting data

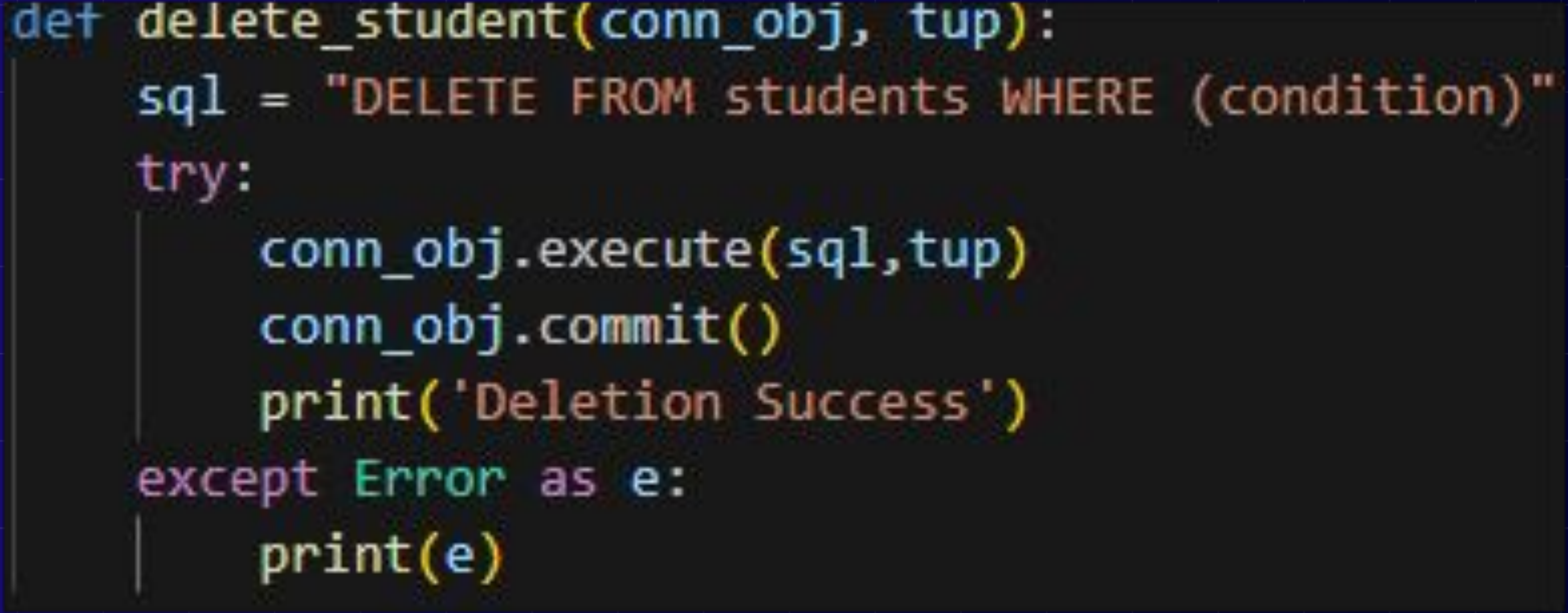
1. Use SELECT [column1, column2…] FROM {table_name}
2. execute
3. commit
4. try except



```python
def select_student(conn_obj, tup):
    sql = "SELECT (column) FROM students"
    try:
        conn_obj.execute(sql,tup)
        conn_obj.commit()
        print('Selection Success')
    except Error as e:
        print(e)
```

Deleting data

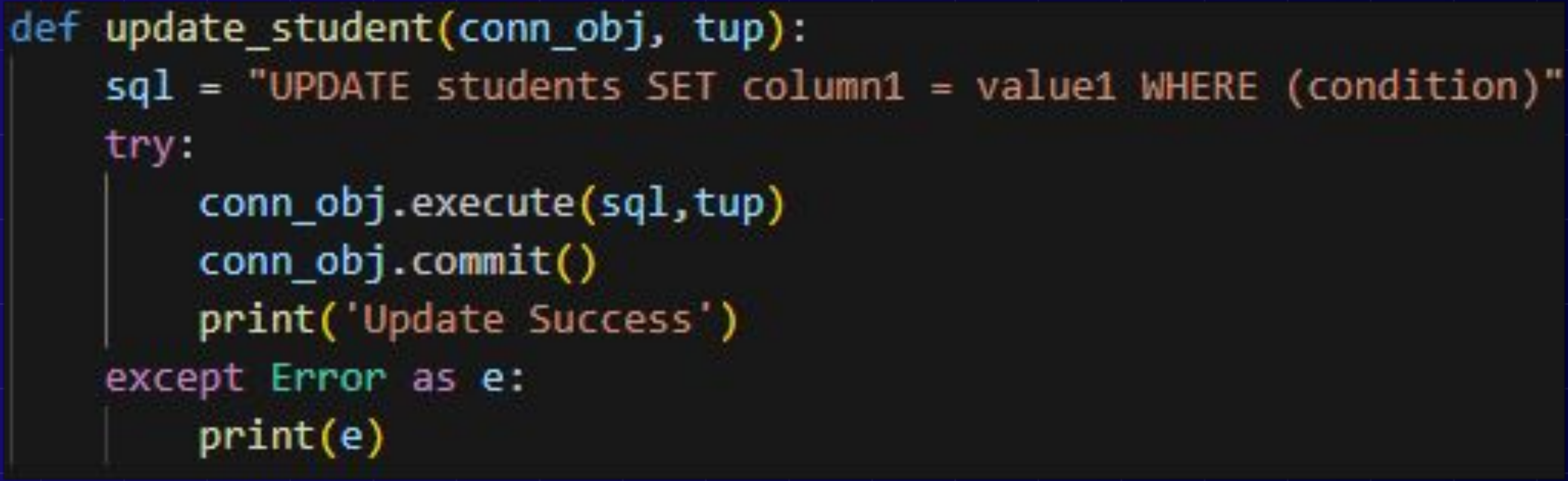
1. Use DELETE FROM {table_name} WHERE [condition]
2. execute
3. commit
4. try except



```
def delete_student(conn_obj, tup):
    sql = "DELETE FROM students WHERE (condition)"
    try:
        conn_obj.execute(sql,tup)
        conn_obj.commit()
        print('Deletion Success')
    except Error as e:
        print(e)
```

Updating data

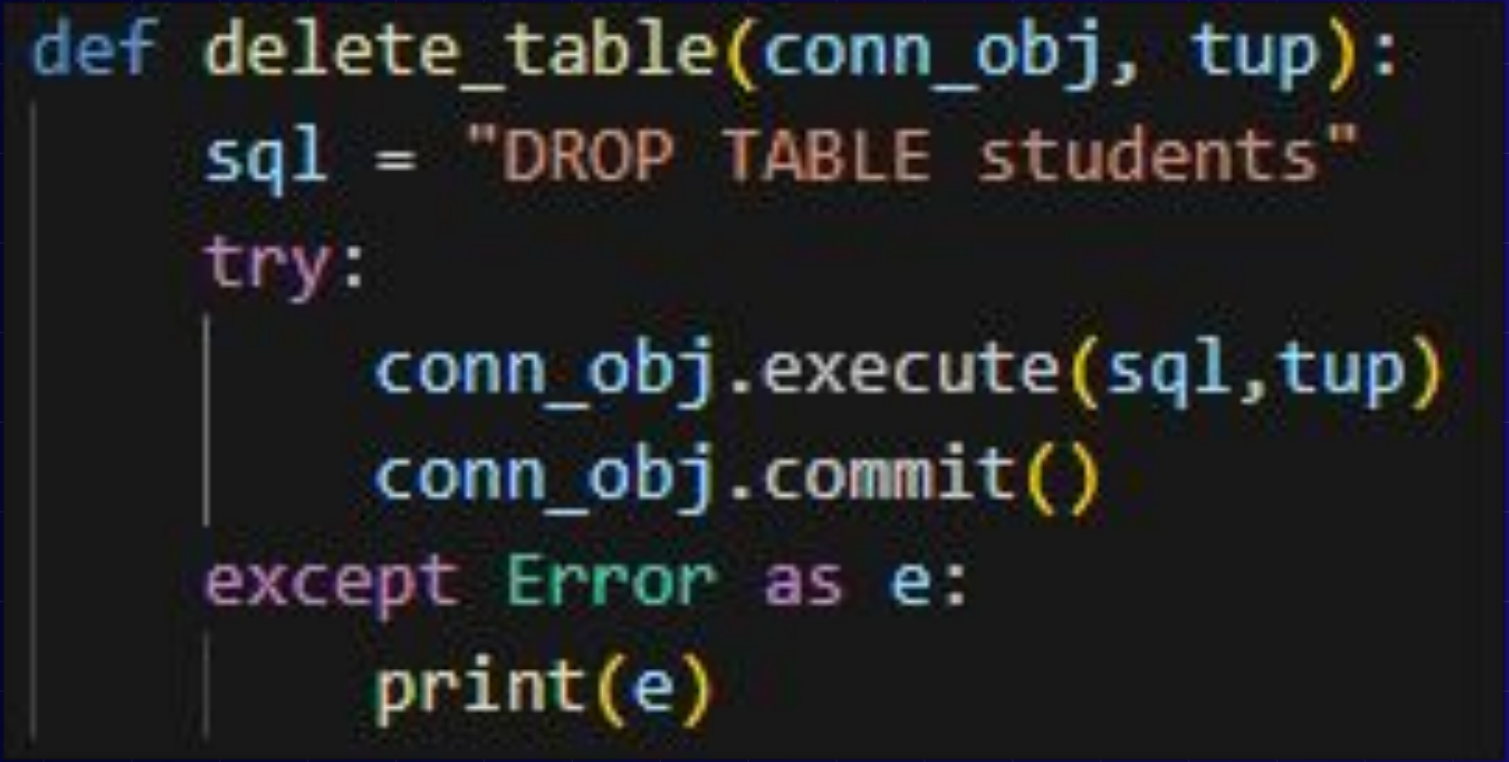
1. UPDATE {table_name}
   SET column1 = value1, column2 = value2....
   WHERE [condition]
2. execute
3. commit
4. try except



```python
def update_student(conn_obj, tup):
    sql = "UPDATE students SET column1 = value1 WHERE (condition)"
    try:
        conn_obj.execute(sql,tup)
        conn_obj.commit()
        print('Update Success')
    except Error as e:
        print(e)
```

Deleting entire tables

1. Use DROP TABLE {table_name}
2. execute
3. commit
4. try except



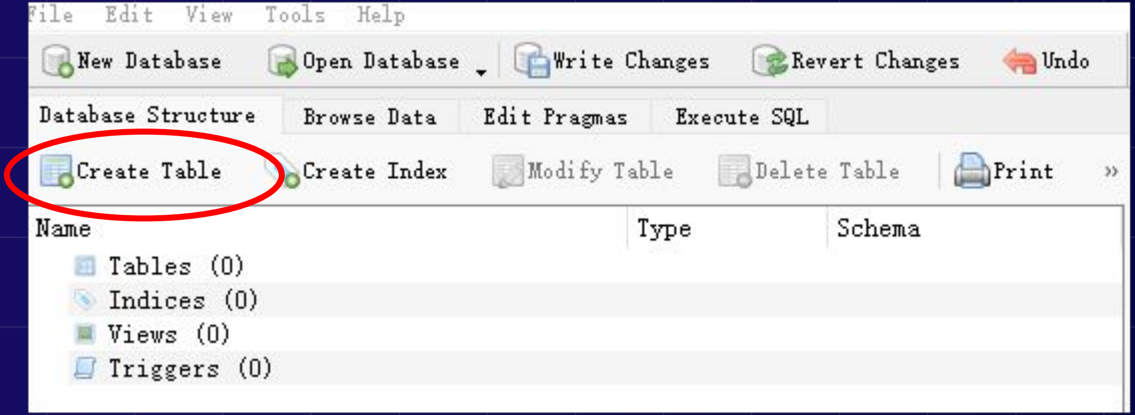
```python
def delete_table(conn_obj, tup):
    sql = "DROP TABLE students"
    try:
        conn_obj.execute(sql,tup)
        conn_obj.commit()
    except Error as e:
        print(e)
```

Creating tables in the database

1. Go to DB browser
2. Create a new database
3. Click “Create Table”
4. Add the needed fields, select constraints and data types
5. Copy the text into python

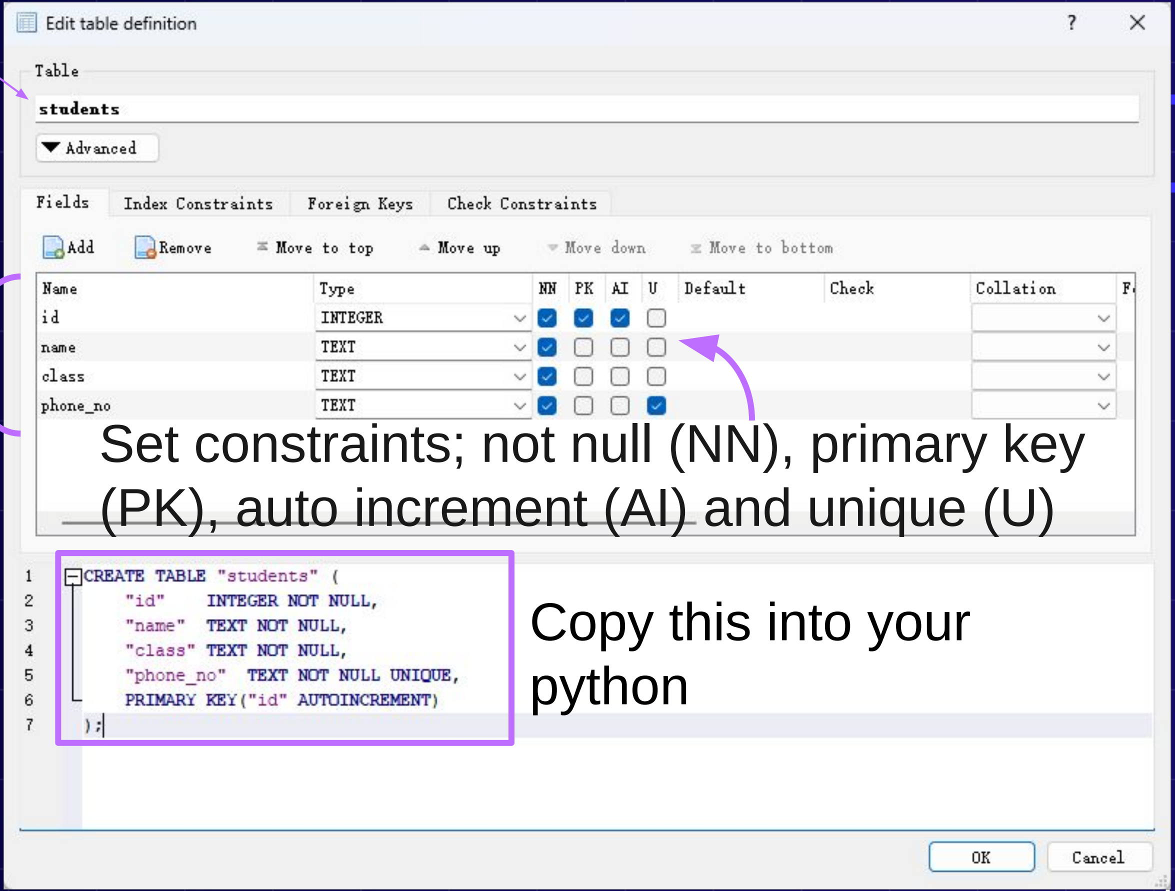
Let’s say you wanna create a “students” table

In DB Browser, click create table



Here you can enter your table name

The fields in your table



Table

students

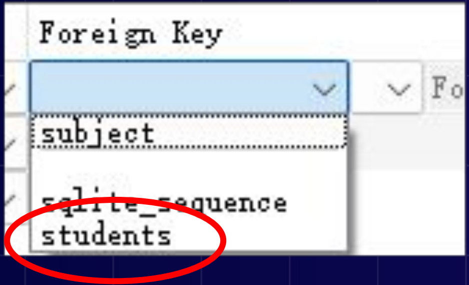
Name	Type	NN	PK	AI	U	Default	Check	Collation
id	INTEGER	✓	✓	✓				
name	TEXT	✓						
class	TEXT	✓						
phone_no	TEXT	✓			✓			

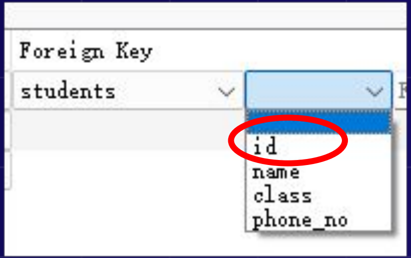
Set constraints; not null (NN), primary key (PK), auto increment (AI) and unique (U)

```
CREATE TABLE "students" (
	"id"	INTEGER NOT NULL,
	"name"	TEXT NOT NULL,
	"class"	TEXT NOT NULL,
	"phone_no"	TEXT NOT NULL UNIQUE,
	PRIMARY KEY("id" AUTOINCREMENT)
);
```

Copy this into your python

For linking tables together with a foreign key, first, scroll to the side of your new table till you see "Foreign Key"





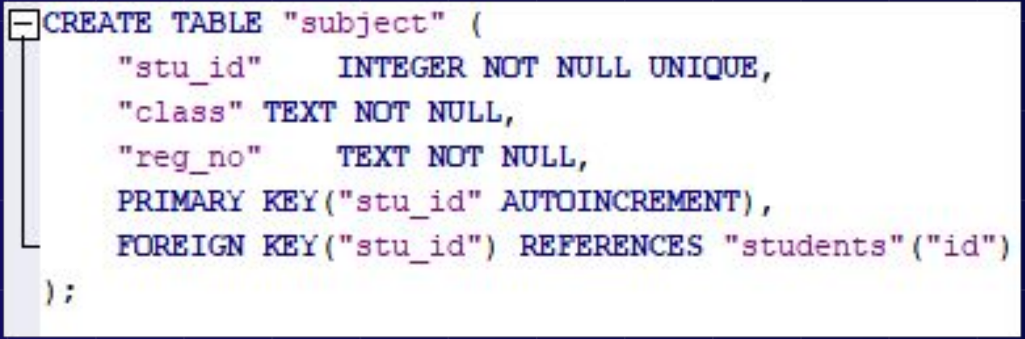
Click which table you want to reference

In my case, it's the students table

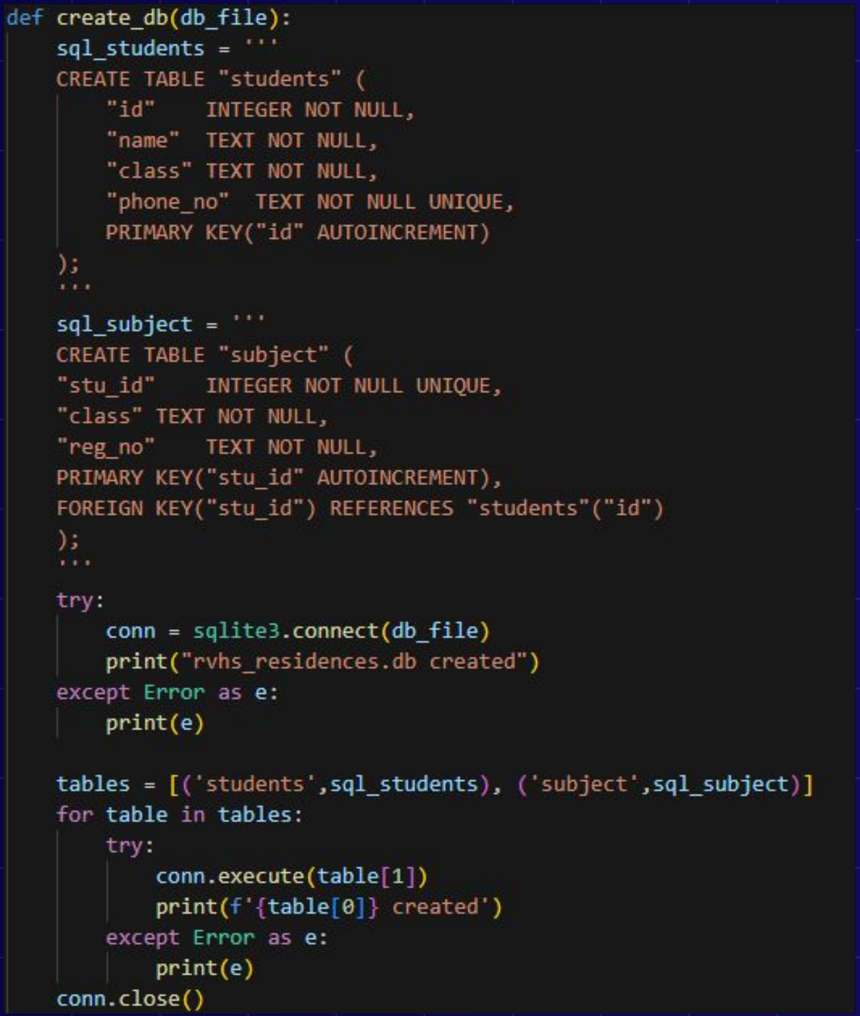
Then select the field that you're going to reference

In my case, it's the id field

This is what it should look like



```
CREATE TABLE "subject" (
    "stu_id"    INTEGER NOT NULL UNIQUE,
    "class" TEXT NOT NULL,
    "reg_no"    TEXT NOT NULL,
    PRIMARY KEY("stu_id" AUTOINCREMENT),
    FOREIGN KEY("stu_id") REFERENCES "students"("id")
);
```



```python
def create_db(db_file):
    sql_students = '''
    CREATE TABLE "students" (
        "id"    INTEGER NOT NULL,
        "name"  TEXT NOT NULL,
        "class" TEXT NOT NULL,
        "phone_no"  TEXT NOT NULL UNIQUE,
        PRIMARY KEY("id" AUTOINCREMENT)
    );
    '''
    sql_subject = '''
    CREATE TABLE "subject" (
    "stu_id"    INTEGER NOT NULL UNIQUE,
    "class" TEXT NOT NULL,
    "reg_no"    TEXT NOT NULL,
    PRIMARY KEY("stu_id" AUTOINCREMENT),
    FOREIGN KEY("stu_id") REFERENCES "students"("id")
    );
    '''
    try:
        conn = sqlite3.connect(db_file)
        print("rvhs_residences.db created")
    except Error as e:
        print(e)

    tables = [('students',sql_students), ('subject',sql_subject)]
    for table in tables:
        try:
            conn.execute(table[1])
            print(f'{table[0]} created')
        except Error as e:
            print(e)
    conn.close()
```

Task 2!

Once you have finished task 1, Santa would like you to create tables to compile the toys, as well as the recipients of the toys and the status of the delivery

The information for toys is stored as such:

Field	Description
`toy_id`	`UNIQUE, NOT NULL` id expressed in `TEXT` to identify each individual toy used as the `PRIMARY KEY` for this table
`toy_type`	type of toy expressed in `TEXT, NOT NULL`
`toy_name`	name of toy expressed in `TEXT, NOT NULL`

The information for the recipients is stored as such:

Field	Description
`recipient_id`	`UNIQUE` id expressed in `TEXT` to identify each individual recipient used as the `PRIMARY KEY` for this table
`recipient_toy_id`	`UNIQUE`, `NOT NULL` id expressed in `TEXT` to identify each individual reference to `toy_id` in the `toy` table
`recipient_name`	name of recipient in `TEXT`, `NOT NULL`
`delivery_status`	delivery status of the toys in `TEXT`, `NOT NULL`

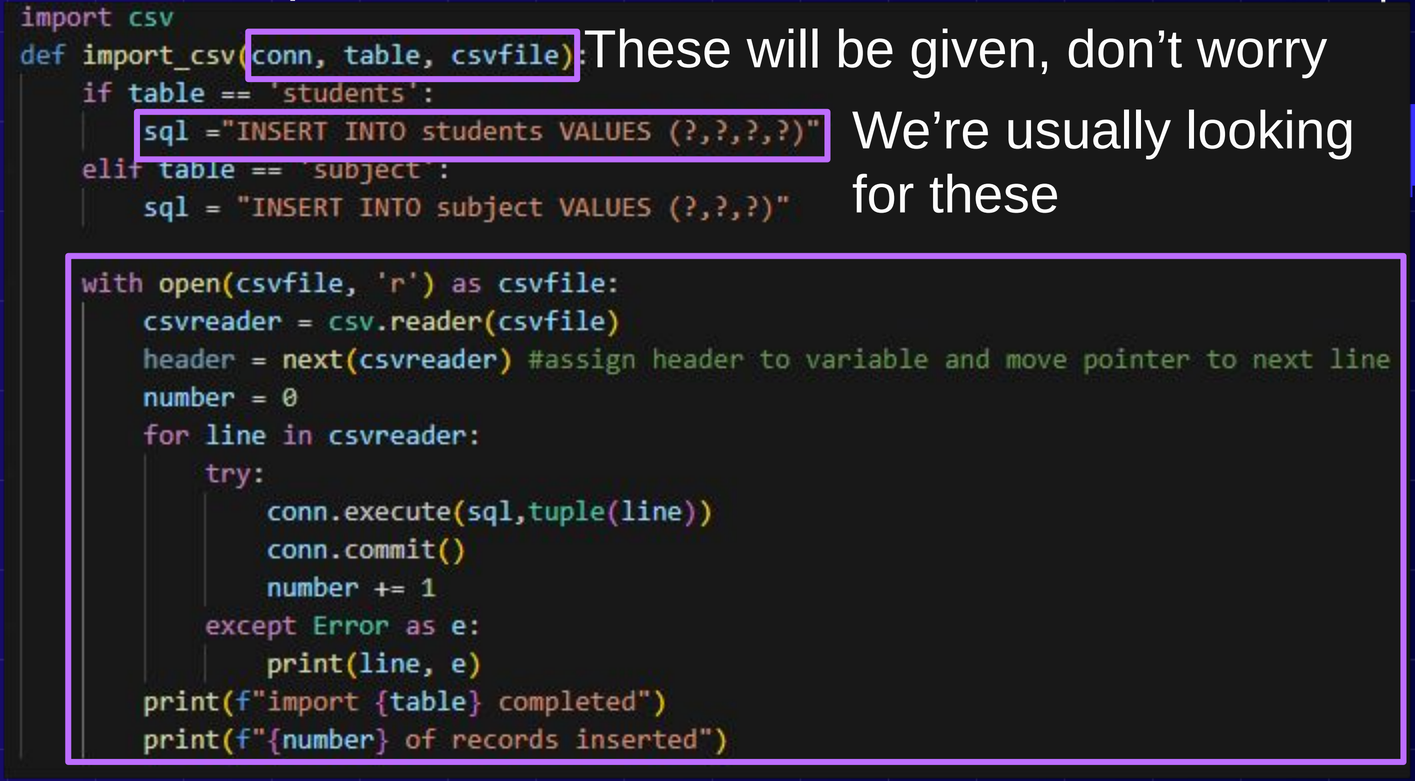
hint: recall what you learnt about creating tables!

How to import csv into database

1. Import csv (important!)
2. Use INSERT INTO {table_name} {values} statement (values are based on the number of fields in the table)
3. Execute the statement under where you imported the csv into
4. Commit the data to the table
5. Except Error as e

Usually try is for executing the statement, except is for error

This might look a bit complicated, but this is in the case that you have to import into more than one table



```
import csv
def import_csv(conn, table, csvfile):
    if table == 'students':
        sql ="INSERT INTO students VALUES (?,?,?,?)"
    elif table == 'subject':
        sql = "INSERT INTO subject VALUES (?,?,?)"

    with open(csvfile, 'r') as csvfile:
        csvreader = csv.reader(csvfile)
        header = next(csvreader) #assign header to variable and move pointer to next line
        number = 0
        for line in csvreader:
            try:
                conn.execute(sql,tuple(line))
                conn.commit()
                number += 1
            except Error as e:
                print(line, e)
        print(f"import {table} completed")
        print(f"{number} of records inserted")
```

These will be given, don't worry

We're usually looking for these

Task 3!

Oh! Santa has just received a new order of toys! Please import this csv into your toy table. Please also print the number of toys imported!

hint: recall what you learnt about importing csv into database!

Task 4!

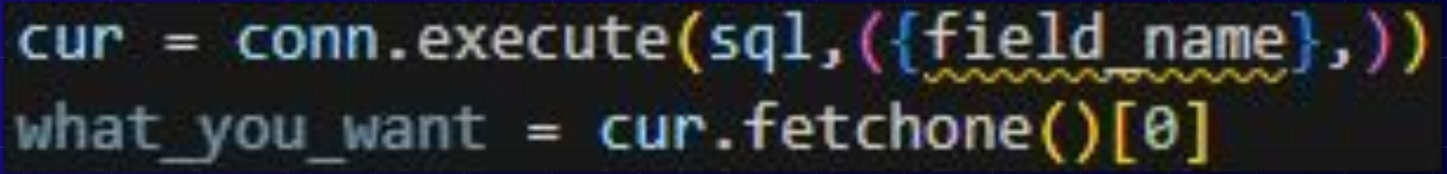
We've just received a list of recipients as well! Please import this csv into your recipients table. Please print the number of recipients imported.

hint: recall what you learnt about importing csv into database!

cursor!

What's cursor?

The sqlite3.Cursor class is an instance where you can invoke methods that execute SQLite statements and fetch data from the result sets of the queries. You can create a Cursor object like this:



```
cur = conn.execute(sql,({field_name},))
what_you_want = cur.fetchone()[0]
```

How to use cursor

Some cursor functions that I’ll be teaching are:

1. fetchone()
2. fetchall()

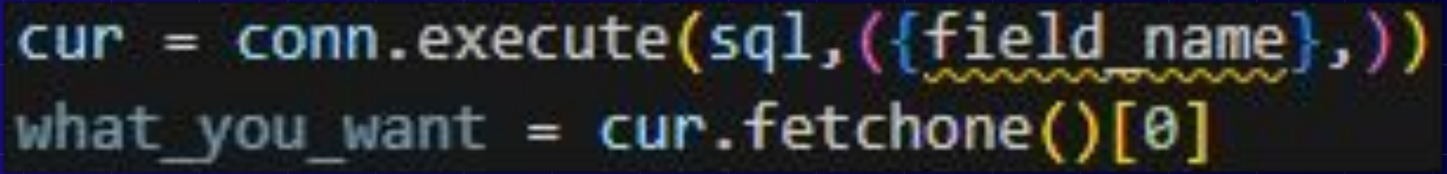
To run these, you just do cur.{function}

so how does it work?

- imagine your mouse cursor. it’s like scrolling through a list and selecting what you want.

cur.fetchone()

- This method fetches the next row of a query result set, returning a single sequence, or None when no more data is available.

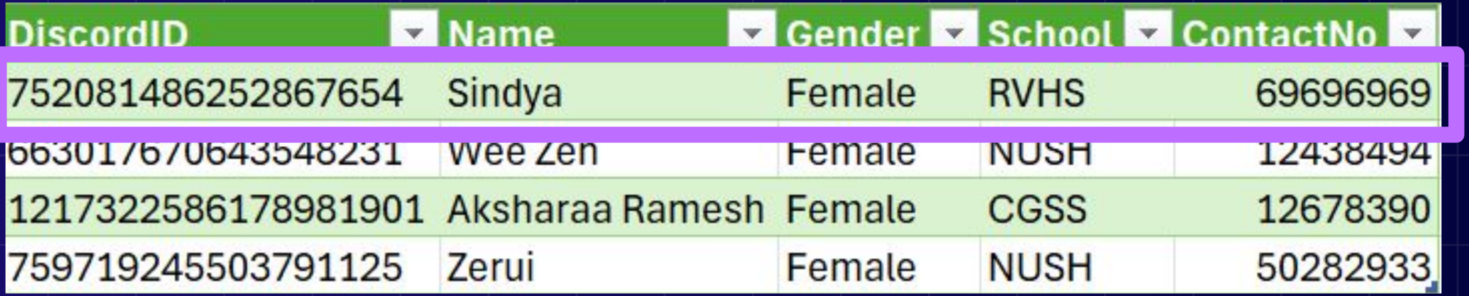


```
cur = conn.execute(sql,({field_name},))
what_you_want = cur.fetchone()[0]
```

cur.fetchone()

For example you've got this table of students, and you run cur.fetchone()

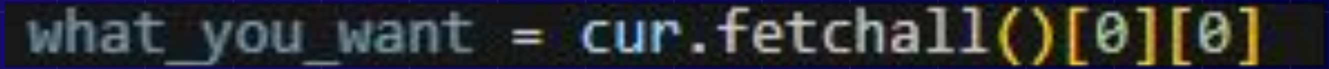
this will return one row of the table. to select specifics, you can do cur.fetchone()[0] which will return the DiscordID in this table.



DiscordID	Name	Gender	School	ContactNo
752081486252867654	Sindya	Female	RVHS	69696969
663017670643548231	Wee Zen	Female	NUSH	12438494
1217322586178981901	Aksharaa Ramesh	Female	CGSS	12678390
759719245503791125	Zerui	Female	NUSH	50282933

cur.fetchall()

This routine fetches all (remaining) rows of a query result, returning a list. An empty list is returned when no rows are available.

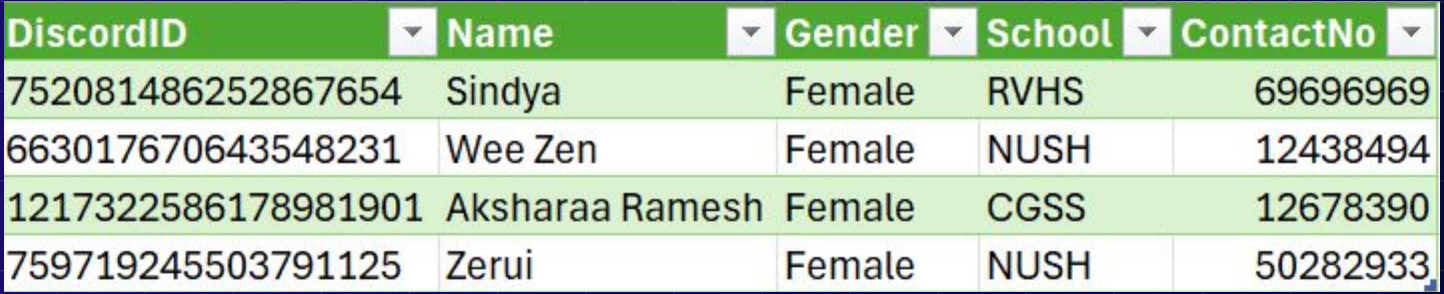


```
what_you_want = cur.fetchall()[0][0]
```

cur.fetchall()

Using the same table from just now, cur.fetchall() returns all of the rows in this table.

to get one specific value, use cur.fetchall()[0][0] to get it.



DiscordID	Name	Gender	School	ContactNo
752081486252867654	Sindya	Female	RVHS	69696969
663017670643548231	Wee Zen	Female	NUSH	12438494
1217322586178981901	Aksharaa Ramesh	Female	CGSS	12678390
759719245503791125	Zerui	Female	NUSH	50282933

Task 5!

There are some threats and contamination in the form of organs and weapons among the toys, please help to filter these out.

Task 5!

The contamination is in the form of Organs, Bombs and Guns. Use cursor to find out the toy id and toy name.

hint #1: recall what you learnt about cursor!

hint #2: use cur.fetchall()!

Task 6!

Please help to filter out the names of the people who have asked for the dangerous items using cursor.

hint #1: recall what you learnt about cursor!

hint #2: use cur.fetchall()!

Deleting specific records

1. Use DELETE FROM {table_name} WHERE condition
2. execute
3. commit
4. try except

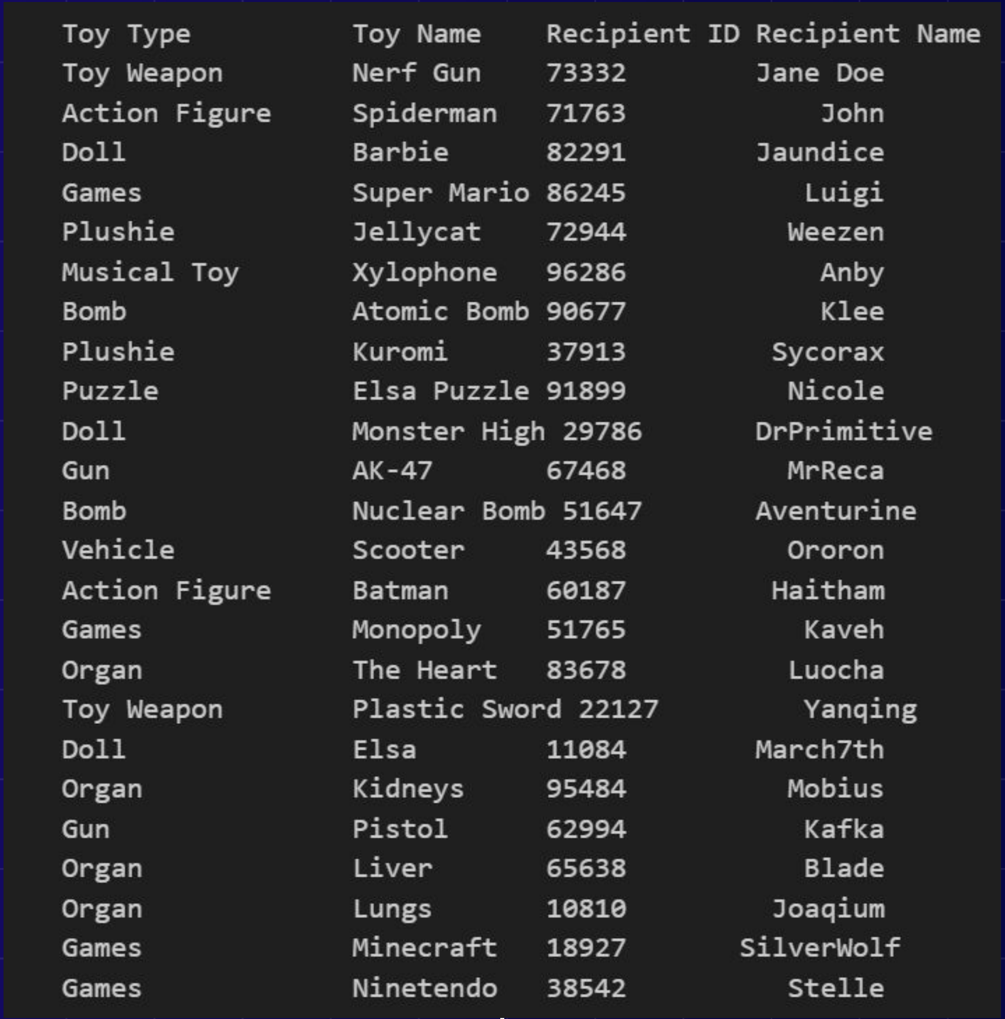
Task 7!

Please help to delete the records of the people who have asked for dangerous items from the recipients table.

hint: recall what you learnt about deleting records!

Task 8!

Please help to delete the records of the dangerous items from the toys table.

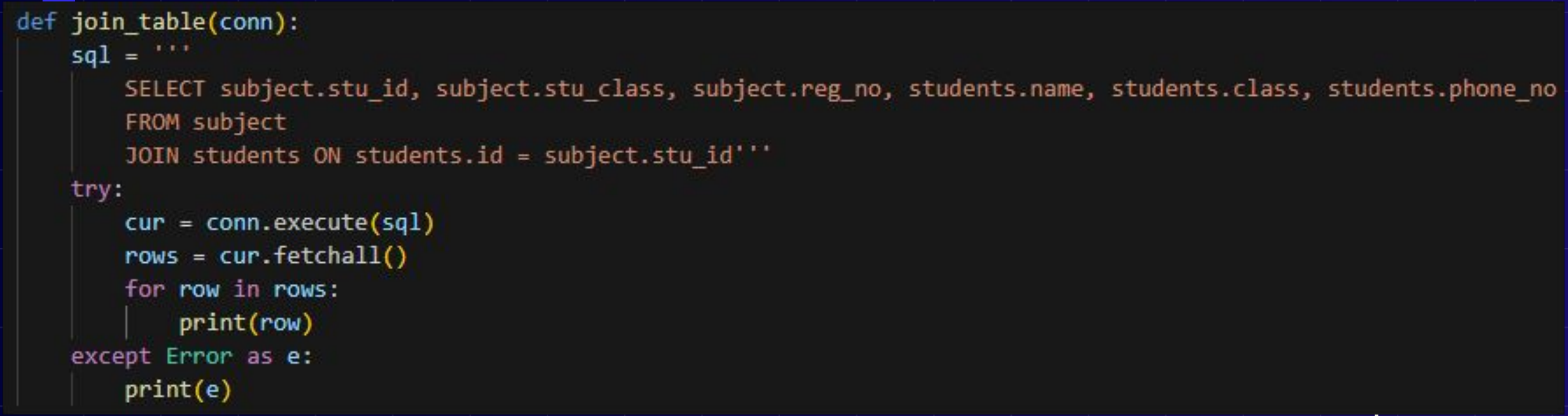


```
Toy Type          Toy Name    Recipient ID Recipient Name
Toy Weapon        Nerf Gun    73332        Jane Doe
Action Figure     Spiderman   71763            John
Doll              Barbie      82291        Jaundice
Games             Super Mario 86245           Luigi
Plushie           Jellycat    72944          Weezen
Musical Toy       Xylophone   96286            Anby
Bomb              Atomic Bomb 90677            Klee
Plushie           Kuromi      37913         Sycorax
Puzzle            Elsa Puzzle 91899          Nicole
Doll              Monster High 29786       DrPrimitive
Gun               AK-47       67468          MrReca
Bomb              Nuclear Bomb 51647       Aventurine
Vehicle           Scooter     43568          Ororon
Action Figure     Batman      60187         Haitham
Games             Monopoly    51765           Kaveh
Organ             The Heart   83678          Luocha
Toy Weapon        Plastic Sword 22127         Yanqing
Doll              Elsa        11084        March7th
Organ             Kidneys     95484          Mobius
Gun               Pistol      62994           Kafka
Organ             Liver       65638           Blade
Organ             Lungs       10810         Joaqium
Games             Minecraft   18927       SilverWolf
Games             Ninetendo   38542          Stelle
```

hint: recall what you learnt about deleting records!

Joining tables in python

1. using the WHERE clause, JOIN ON the primary key of one table and the foreign key of the other table
2. you’ll need cursor for this one, if you’re fetching values.
3. return the value



```python
def join_table(conn):
    sql = '''
        SELECT subject.stu_id, subject.stu_class, subject.reg_no, students.name, students.class, students.phone_no
        FROM subject
        JOIN students ON students.id = subject.stu_id'''
    try:
        cur = conn.execute(sql)
        rows = cur.fetchall()
        for row in rows:
            print(row)
    except Error as e:
        print(e)
```

Task 9!

Finally, please help to join both databases together! That way it will make deliveries easier!

Task 9!

If anyone has prior python knowledge you can print it out in table format too :D hint: recall what you learnt about joining tables!

Congratulations! You have successfully finished your ~~unpaid overtime~~ work!

You are free to take a 5 second break (just kidding) before you go back to work!

THANKS!

Any questions?