BUILD X : ANDROID

Lecture Four: Preferences and Databases

Handout:
https://tinyurl.com/android-lecture4-handout

Recap

Handout:
https://tinyurl.com/android-lecture4-handout

Recap – What are we doing?

We're going to build a basic todo list app, **KCL Tech Todo**.

If you want a preview, it is **available on Google Play.**

- Search for "**KCL Tech Todo"** on Google or on the Play Store.

Recap - ListViews

ListViews ...

Adapters ...

Recap - ListViews

ListViews display a list of items in any kind of list format.

They recycle views in the list to save memory, only showing what is needed.

Adapters map data to a ListView, maintaining a separation of concerns.

- These have `getCount()`, `getItem()` and `getView()`.

This means that any type of data that can fit into an **adapter** can be displayed in a ListView.

Storing Data

Handout:
https://tinyurl.com/android-lecture4-handout

Storing Data

There are **many reasons** why you'd want to store data on a user's device.

- Preferences, app content, high scores, game progress, usage logs, etc.

The data is **persistent**, meaning it still exists if the app closes or the device reboots.

There are three main ways to store data…

Storing Data: Plain Files

Your app can **write and read any kind of file** to a private area, accessible only to your app.

This is useful if you have your own **proprietary file format**, or your have **large BLOBs** (Binary Large OBject) of data to store.

We will not be using or studying this method.

Storing Data: Shared Preferences

Shared preferences is a utility built into Android that is primitive, but **very easy to use.**

You can use shared preferences to store simple **key/value information**.

Storing Data: Databases

Your app can **create and use** as many databases as it wants.

Databases are great for storing structured data that needs to be organised, searchable, etc.

Databases are the most **complicated** method, but also the most **powerful**.

Shared Preferences

Handout:
https://tinyurl.com/android-lecture4-handout

Storing Data: Shared Preferences

Shared preferences are designed to store a user's preferences and settings, like whether they have notifications turned on.

However… you can actually use the for **any kind of simple key/value information**.

Storing Data: Shared Preferences

Key	Value
`user_highscore`	`42`
`welcome_tour_finished`	`true`
`notification_alert_sound`	`"bells"`

Coding time!

Handout:
https://tinyurl.com/android-lecture4-handout

Side Note: Models

Not this kind of model.

```java
public class Task {

        private long id;
        private String title;
        private String notes;
        private DateTime dueDate;
        private boolean complete;

        /*==============*
         * Constructors *
         *==============*/

        public Task(String title, String notes, DateTime dueDate, boolean complete) {
                this.title = title;
                this.notes = notes;
                this.dueDate = dueDate;
                this.complete = complete;
        }

        public Task(Cursor input) throws IllegalArgumentException {
                id = input.getLong(input.getColumnIndexOrThrow(Db.id));
                title = input.getString(input.getColumnIndexOrThrow(Db.title));
                notes = input.getString(input.getColumnIndexOrThrow(Db.notes));
                dueDate = new DateTime(input.getLong(input.getColumnIndexOrThrow(Db.dueDate)));
                complete = input.getInt(input.getColumnIndexOrThrow(Db.complete)) == 1;
        }

        /*====================*
         * Getters and setters *
         *====================*/
```

This kind of model!

Models

A model is a **representation** of some item or entity.

In **Java (and therefore Android)**, a model is generally **a class** that represents one of the items or entities your software deals with.

We need a model that **represents a task.**

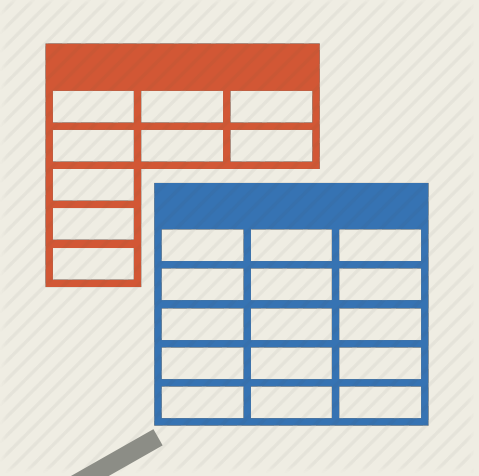
Coding time!

Databases

Handout:
https://tinyurl.com/android-lecture4-handout

Databases: 60-Second Recap

A database contains many **tables**



A table looks like this

Row:
A single object, item, record or entity of your data.

Column:
A property or attribute about your data.

Column

Row

ID	Name	Age	Gender	Email	Phone	Alive
1	Mark	22	M	mar...	123	Y
2	Ana	20	F	ana...	123	Y
3	George	21	M	geo...	123	Y
4	Maria	21	F	mar...	123	Y
5	Alan	42	M	ala...	123	N

Databases: SQL and SQLite

Databases often use **SQL**, which stands for **Structured Query Language.**

- A set of questions and commands you can send to a database.
- Allow you to **create, read, update and delete** records.

Android uses **SQLite**, which is a lightweight version of SQL.

- It's a bit less powerful, but also better at running on mobile devices.

Example queries:

```
SELECT Name, Age FROM Users;
SELECT * FROM Users WHERE alive = "Y"
SELECT * FROM Users ORDER BY age DESC;
```

Databases: SQL

Users

ID	Name	Age	Gender	Email	Phone	Alive
1	Shakeel	19	M	shak…	123	Y
2	Jardin	19	M	jar…	123	Y
3	Vishnu	19	M	vish…	123	Y
4	Sarah	22	F	sar..	123	Y
5	Bea	20	F	bea…	123	Y
6	Tupac	25	M	tup…	123	N

```
SELECT Name, Age FROM Users;
```

Databases: SQL

```
SELECT Name, Age FROM Users;
```

Name	Age
Shakeel	19
Jardin	19
Vishnu	19
Sarah	22
Bea	20
Tupac	25


```
SELECT Name, Age FROM Users;
```

Databases: SQL

Users

ID	Name	Age	Gender	Email	Phone	Alive
1	Shakeel	19	M	shak…	123	Y
2	Jardin	19	M	jar…	123	Y
3	Vishnu	19	M	vish…	123	Y
4	Sarah	22	F	sar..	123	Y
5	Bea	20	F	bea…	123	Y
6	Tupac	25	M	tup…	123	N

```
SELECT * FROM Users WHERE alive = “Y”
```

Databases: SQL

```
SELECT * FROM Users WHERE alive = “Y”
```

ID	Name	Age	Gender	Email	Phone	Alive
1	Shakeel	19	M	shak…	123	Y
2	Jardin	19	M	jar…	123	Y
3	Vishnu	19	M	vish…	123	Y
4	Sarah	22	F	sar..	123	Y
5	Bea	20	F	bea…	123	Y


```
SELECT * FROM Users WHERE alive = “Y”
```

Databases: SQL

Users

ID	Name	Age	Gender	Email	Phone	Alive
1	Shakeel	19	M	shak…	123	Y
2	Jardin	19	M	jar…	123	Y
3	Vishnu	19	M	vish…	123	Y
4	Sarah	22	F	sar..	123	Y
5	Bea	20	F	bea…	123	Y
6	Tupac	25	M	tup…	123	N

```
SELECT * FROM Users ORDER BY age DESC;
```

Databases: SQL

```
SELECT * FROM Users ORDER BY age DESC;
```

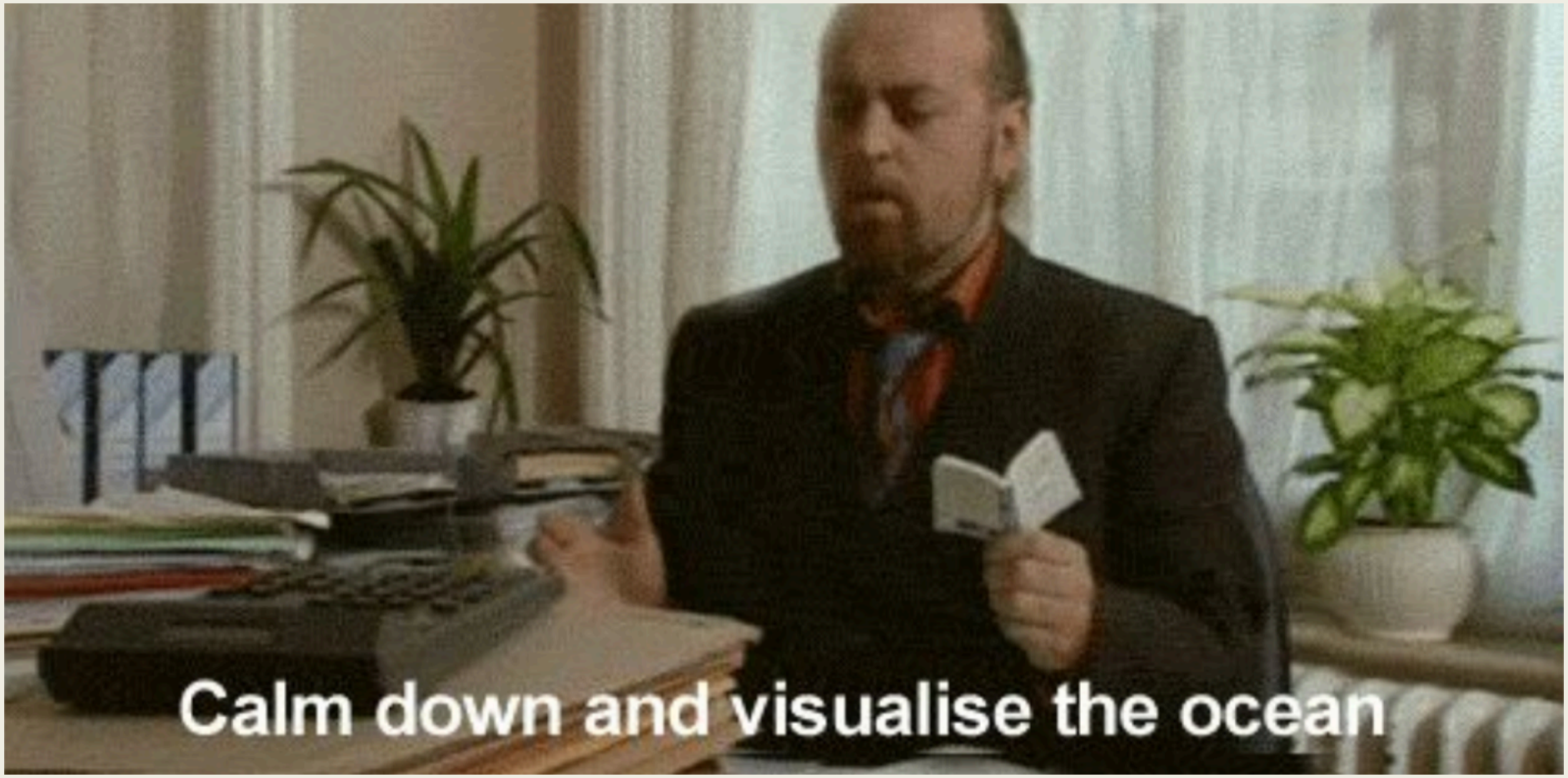
ID	Name	Age	Gender	Email	Phone	Alive
6	Tupac	25	M	tup…	123	N
4	Sarah	22	F	sar..	123	Y
5	Bea	20	F	bea…	123	Y
1	Shakeel	19	M	shak…	123	Y
2	Jardin	19	M	jar…	123	Y
3	Vishnu	19	M	vish…	123	Y


```
SELECT * FROM Users ORDER BY age DESC;
```

Databases

We will cover four parts to using databases in Android;

- Creating a database handler/helper
- Creating a table
- Inserting a task
- Reading tasks



Break! 10 mins

Handout:
https://tinyurl.com/android-lecture4-handout

Databases: Creating a Helper

Handout:
https://tinyurl.com/android-lecture4-handout

Databases: Creating a Helper

A **helper/handler** is a tool that makes working with SQLite databases easier.

Remember how we made an adapter by extending `BaseAdapter`?

To make our own helper, we extend `SQLiteOpenHelper`.

Coding time!

Databases:
Creating a Table

Databases – Creating a Table

We will use an SQL command to create a table.

```
CREATE TABLE Tasks (
    id INTEGER PRIMARY KEY,
    title TEXT,
    notes TEXT,
    due_date INTEGER,
    is_complete INTEGER
);
```

Coding time!

Databases:
Inserting a Task

Databases: Inserting a Task

We **could** use SQL, but there's a **better way**.

If we can convert our task into a `ContentValues` object, we can insert that directly.

A **ContentValues** object simply stores key/value pairs of information, just like a Bundle (lecture three).

We're going to handle **create and update** of a Task in the same method by telling the database to **replace** any task that has the same ID.

Coding time!

Databases:
Reading Tasks

Database: Reading Tasks

This time we will use SQL – using two different queries.

To get all uncomplete tasks:

```
SELECT * FROM Tasks WHERE is_complete = 0 ORDER BY due_date ASC;
```

To get a single, specific task:

```
SELECT * FROM Tasks WHERE id = 0;
```

Databases: Reading Tasks/Cursors

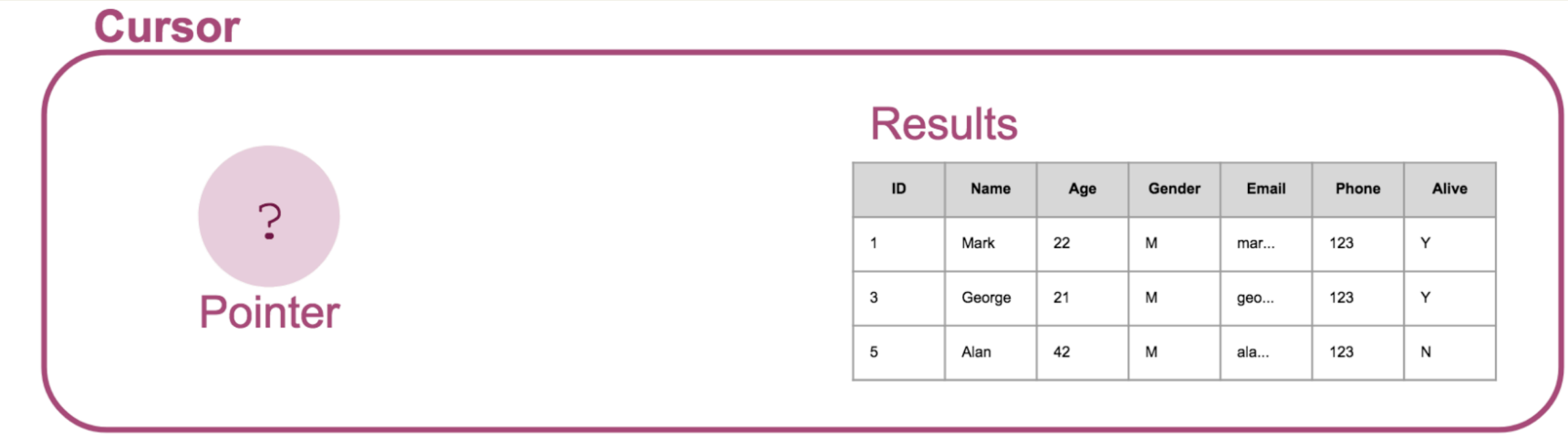
Both of those queries return a `Cursor`. Cursors are special…

A cursor is **one object** that can represent **any number of rows** from your database.

It does this by only "looking at" **one record at a time** and using a **pointer** to remember which one it is currently looking at.

Databases: Reading Tasks/Cursors

Here is an **example result set**, and our **pointer**. To start with, it doesn't point to anything.



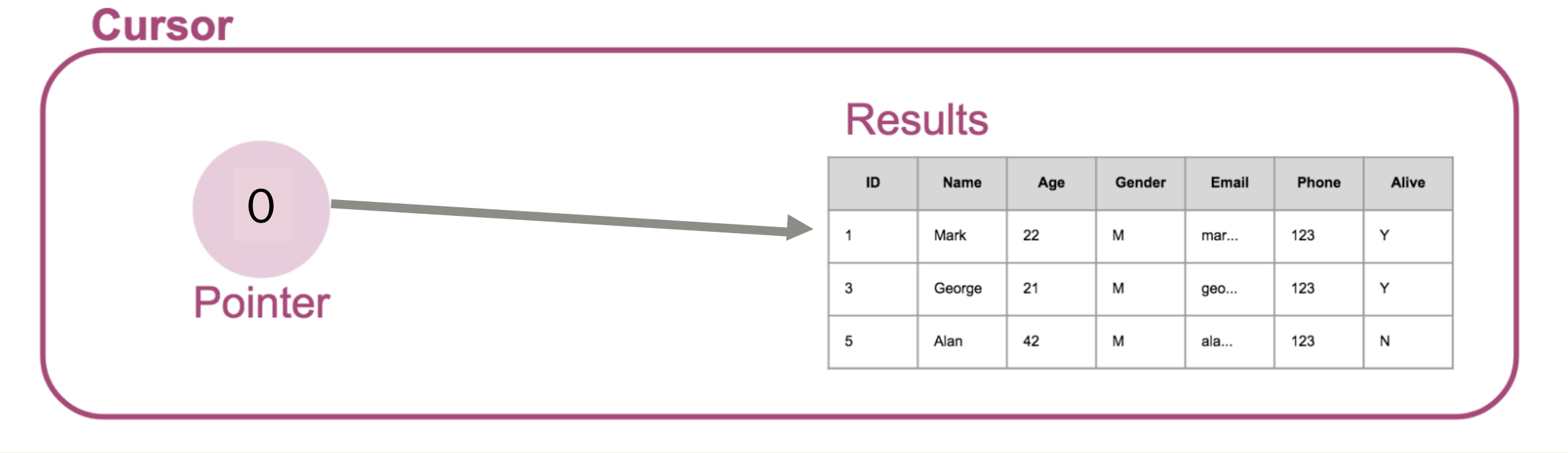
Databases: Reading Tasks/Cursors

We can call `moveToFirst()` to start looking at the first result.

This returns false if there **isn't** a first result (i.e. the result set is empty).

Databases: Reading Tasks/Cursors

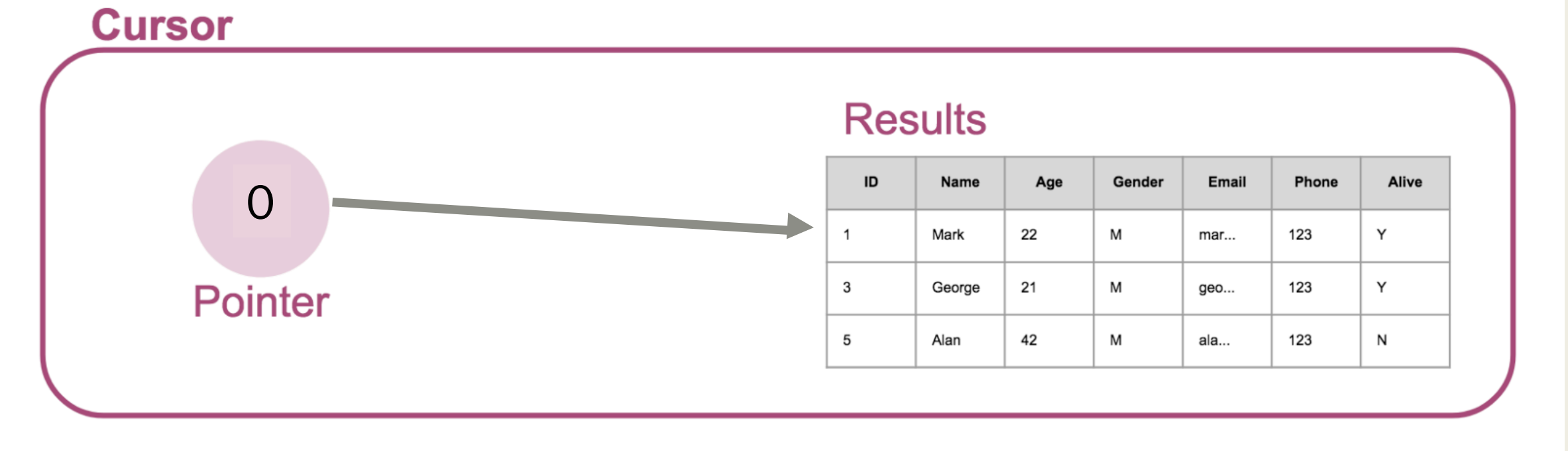
We can now use `getInt(…)`, `getString(…)` to read information about the active result.



Databases: Reading Tasks/Cursors

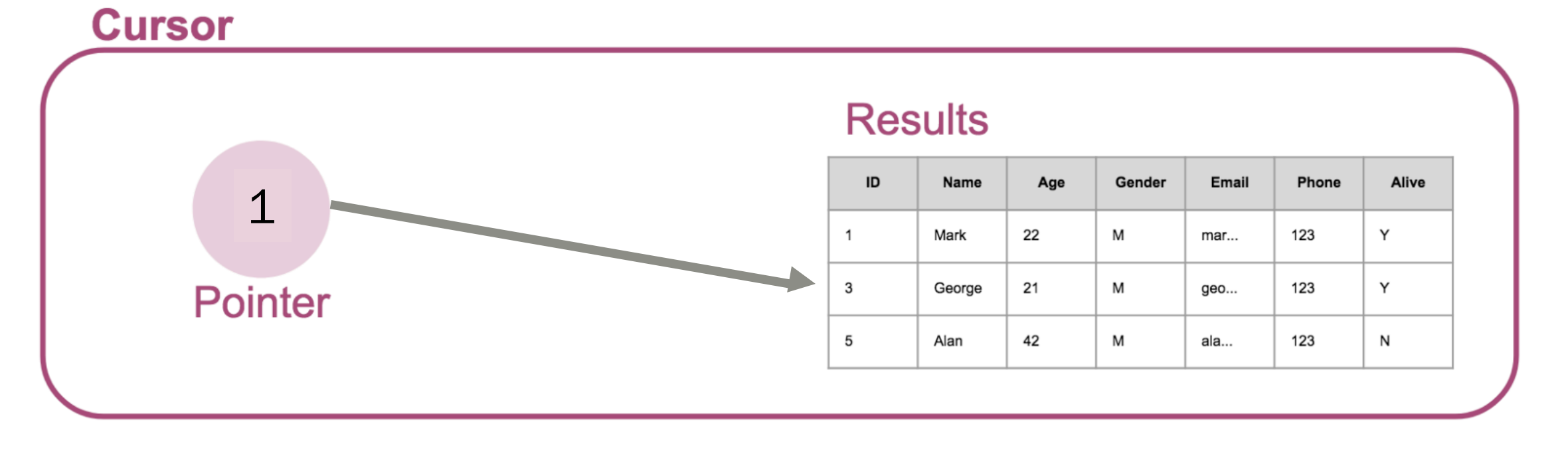
These methods take the column number as an argument, starting at zero.

- `getInt(0) -> 1`
- `getString(1) -> "Mark"`
- `getInt(2) -> 22`



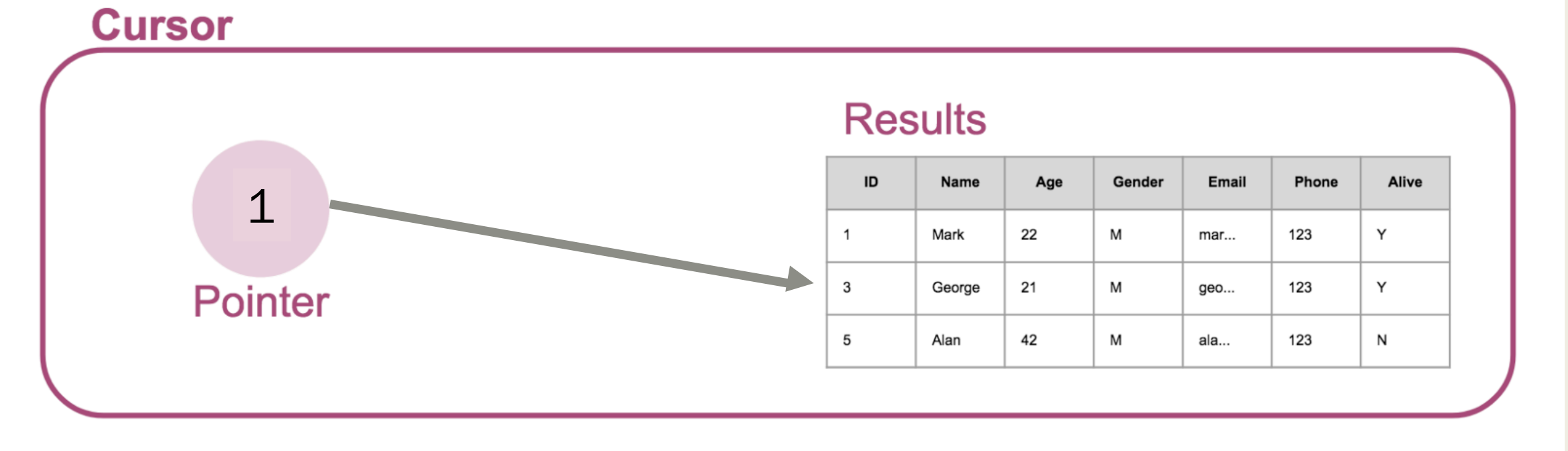
Databases: Reading Tasks/Cursors

To move onwards, we can call `moveToNext()`.



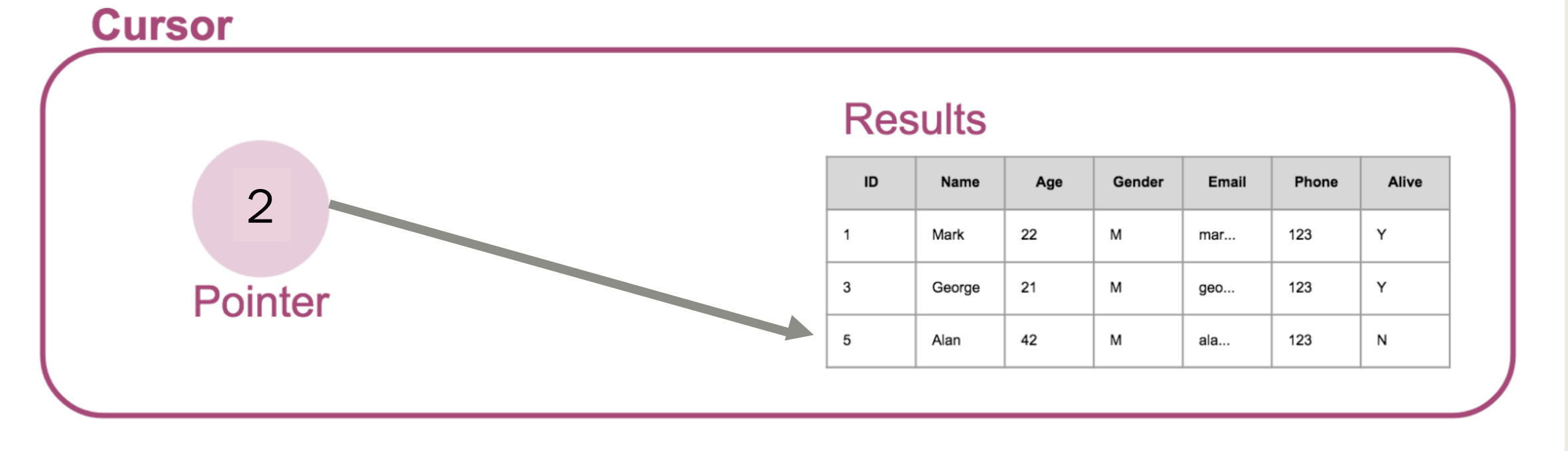
Databases: Reading Tasks/Cursors

If we call `moveToNext()` again...



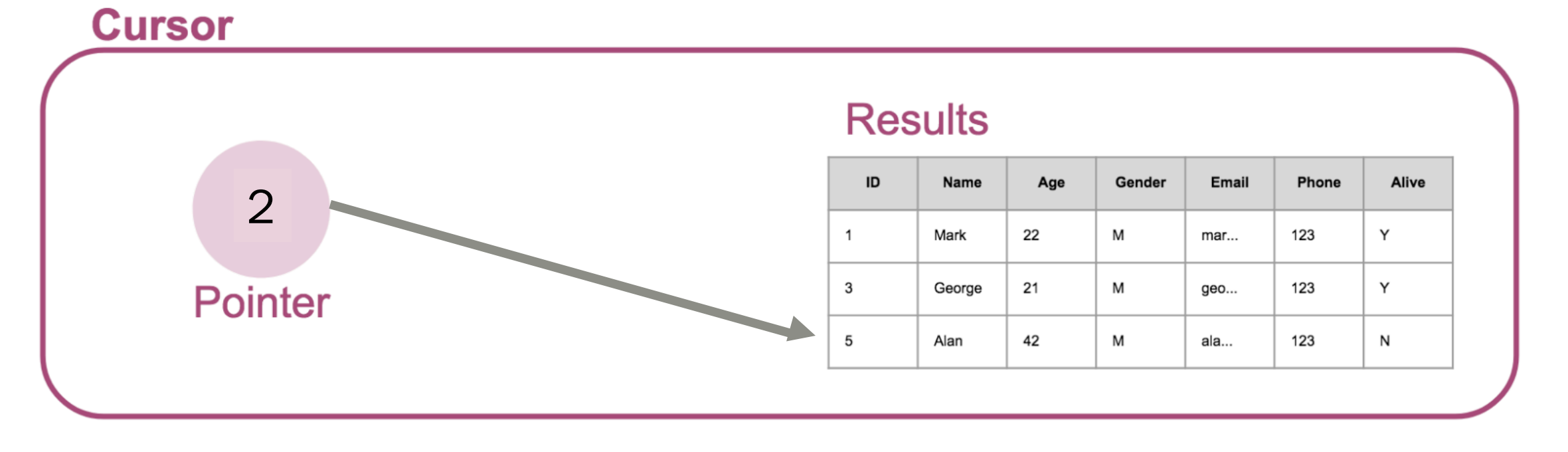
Databases: Reading Tasks/Cursors

If we call `moveToNext()` again...



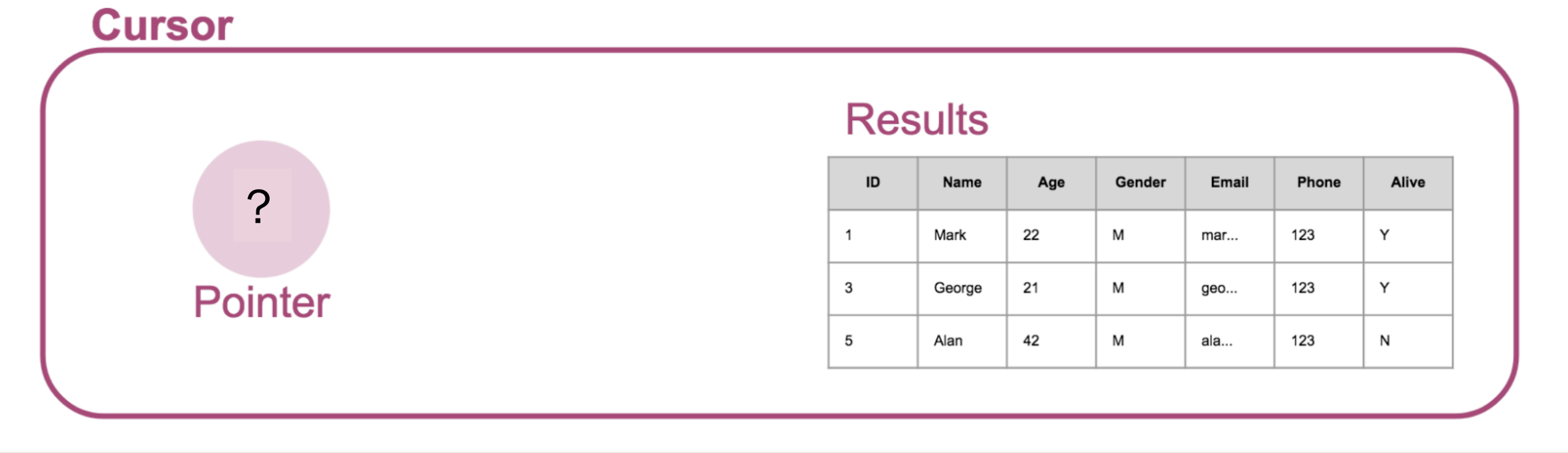
Databases: Reading Tasks/Cursors

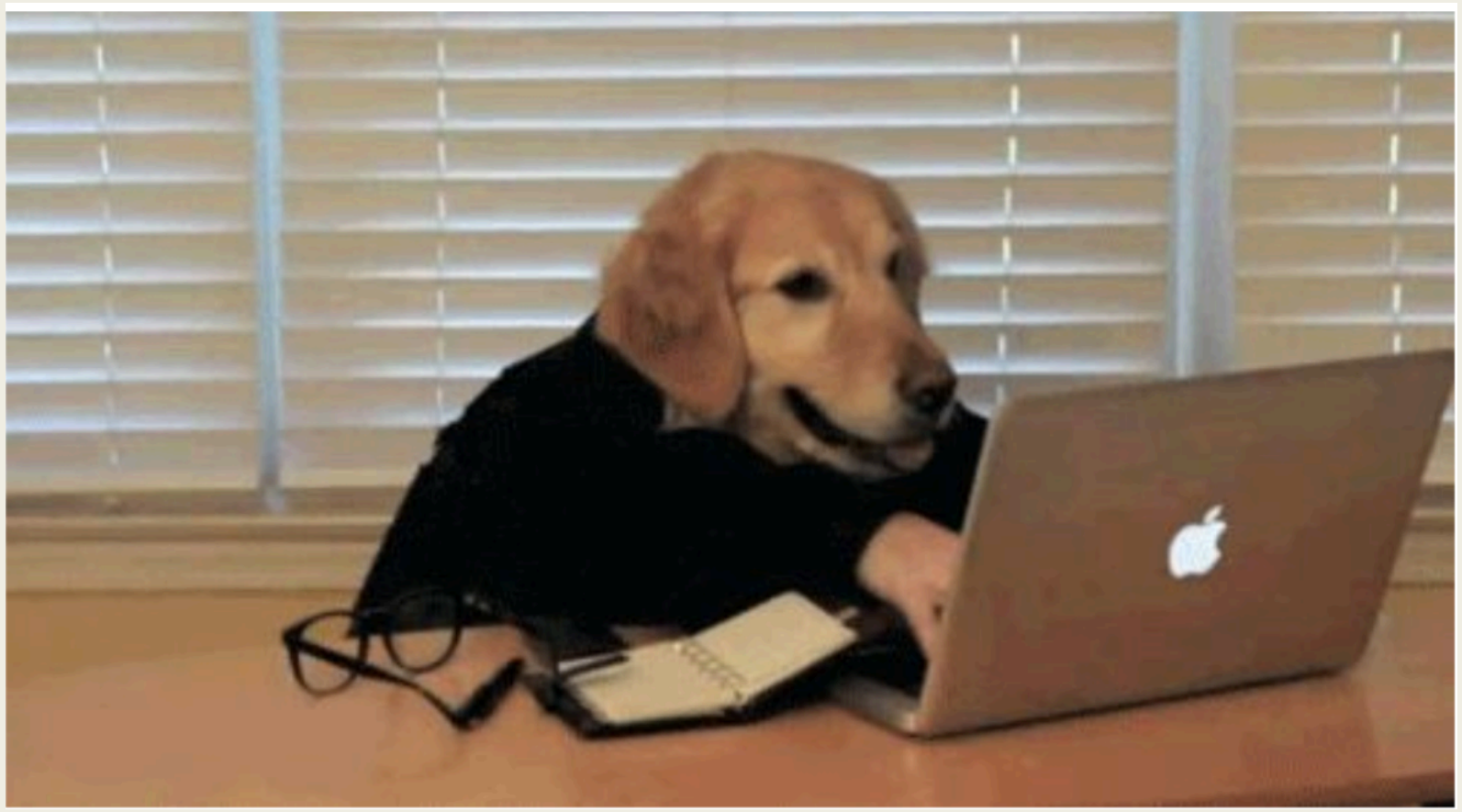
If we call `moveToNext()` once more we run out of records, so it returns false to tell us there wasn't a "next" record.



Databases: Reading Tasks/Cursors

If we call `moveToNext()` once more we run out of records, so it returns false to tell us there wasn't a "next" record.





Coding time!

Next Week: Putting it Together!

We will put everything we've learned so far together into a **complete working app!**

Make sure you are up to date with everything we've done so far.

- I'll upload all resources, slides and "work so far" code later this week for you all to look at.

Done!