IN4MATX 133: User Interface Software

Lecture 16:
Databases and Local Storage

Today’s goals

By the end of today, you should be able to…

- Differentiate relational from non-relational databases
- Explain the advantages of each style of database
- Use IndexedDB to implement a non-relational database

Today is a crash course in databases
CS 122A and 122B provide substantially more depth

Data storage

- What happens when we refresh the A4 sleep tracking app?
  - We lose all of the data we logged
- This is obviously not ideal
  - We have to tell the browser, app, etc. to store it

Data storage

- Data can be stored locally on a device
  - Android and iOS allow apps to store some data
  - Capacitor provides (good) libraries for using local storage

Local Storage

- In Ionic, can store key-value pairs
  - Keys must be strings, values can be any type
- This is actually a non-relational database!
  - More on this in a few slides

https://capacitorjs.com/docs/apis/storage

Local Storage

- Capacitor provides a cross-platform storage API

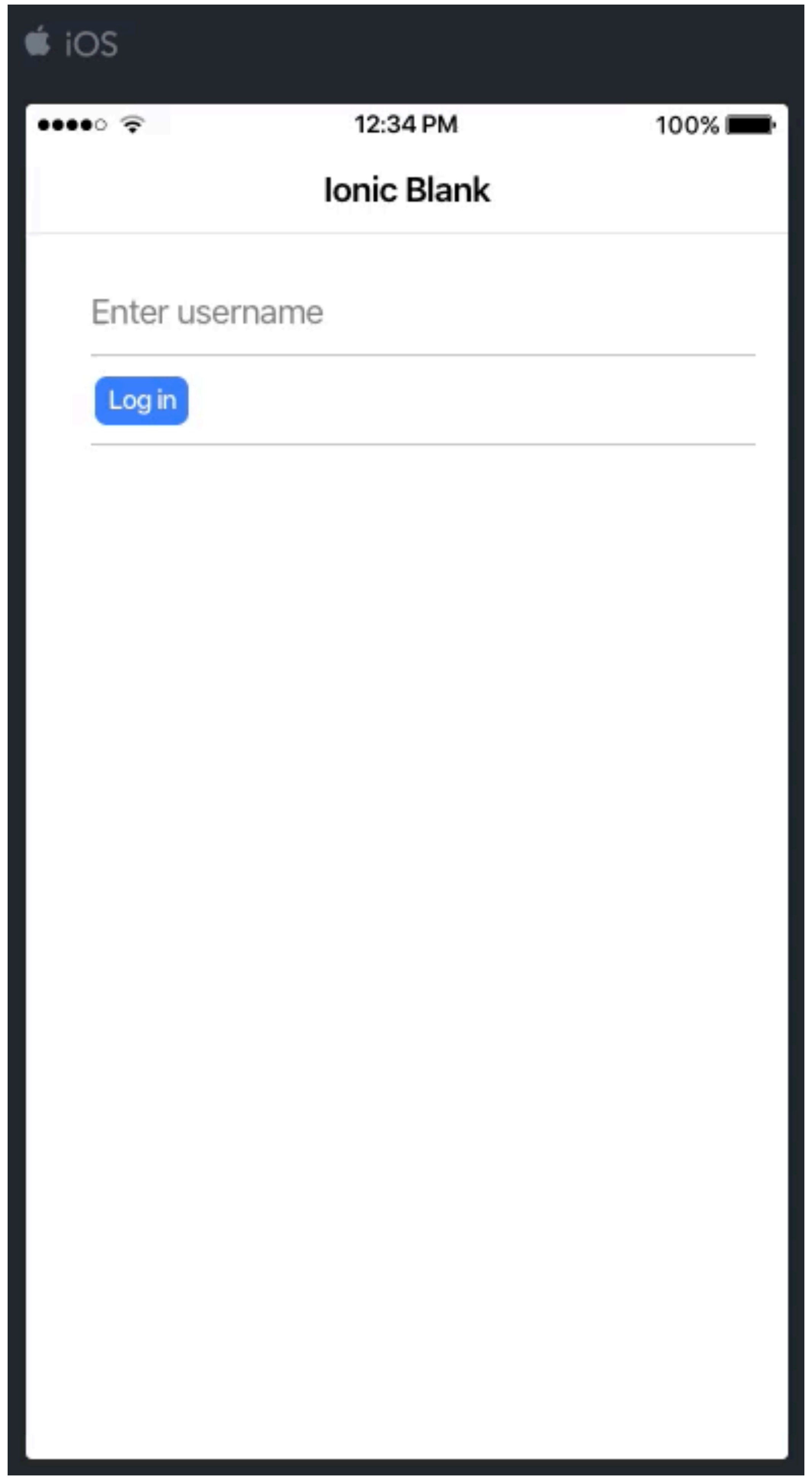
```
import { Storage } from '@capacitor/storage';

Storage.get({key:’keyName’}).then((data) => {
  console.log(data.value);
});

Storage.set({key:’keyName’, value:’value’}).then(() => {
  console.log(“set value”);
});
```

https://capacitorjs.com/docs/apis/storage

Local Storage



If we can store data on devices, why do we need databases?

Databases

- Provide reliability
  - You can get your data back if your phone dies or you get a new phone
- Provide cross-device support
  - Allow you to see and modify the same data across a phone and a desktop, for example

Databases

- Are more than files stored in the cloud
  - Can be “queried” efficiently to get subsets of data
- Two main approaches to making databases
  - Relational databases: MySQL, Postgres
  - Non-relational databases: MongoDB, Firebase, IndexedDB
- Transaction: any add/delete/update/etc. made to a database

Databases

Relational databases

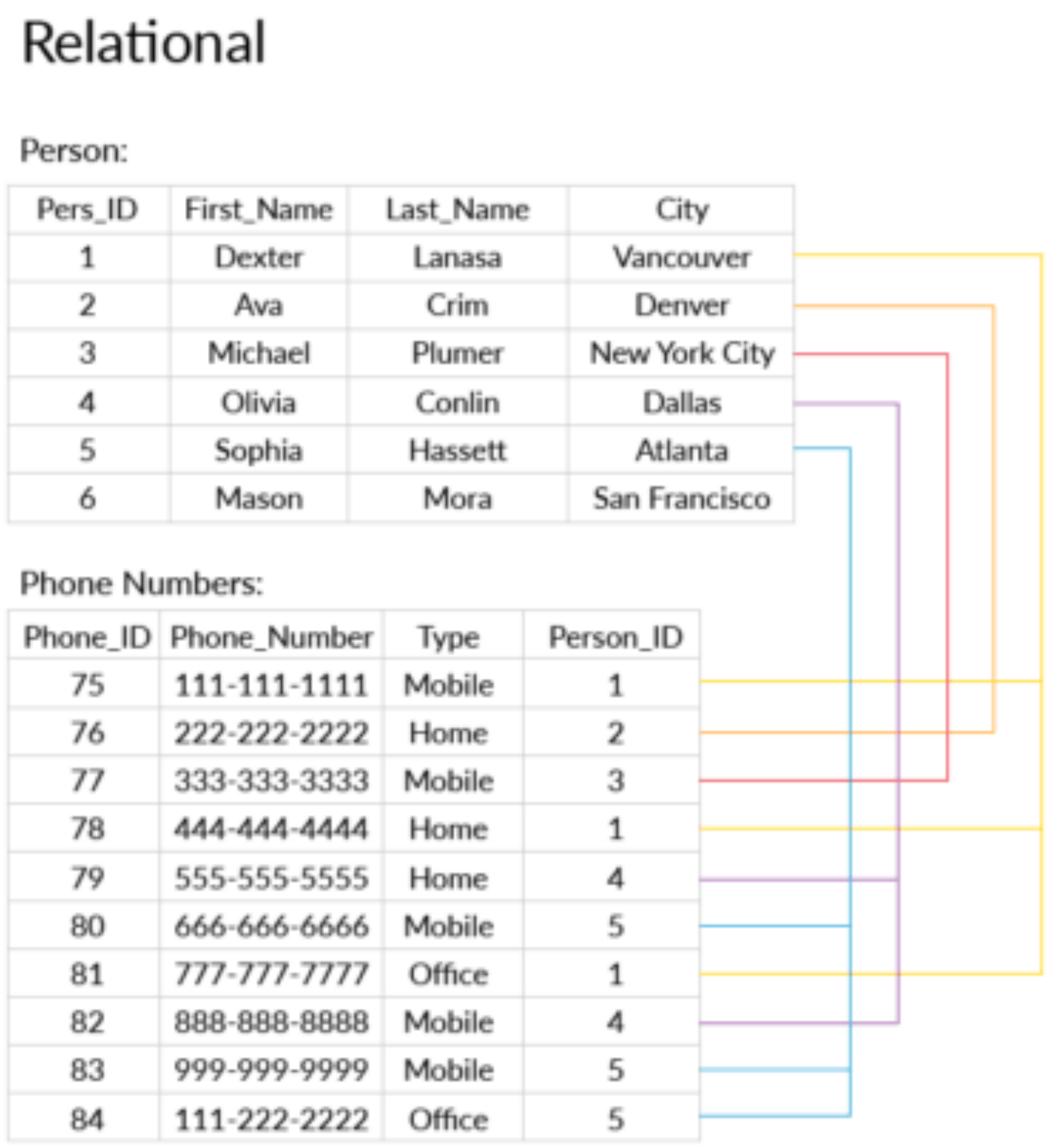
- Everything is organized into tables
- Tables contain columns with predefined names and data types
- Tables “relate” to one another by having overlapping or similar columns
  - Minimizes redundancy and keeps order
- Every data entry is a row of a table

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/

https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Relational databases



Relational

Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/

https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Relational databases

```sql
CREATE TABLE IF NOT EXISTS tasks (
    task_id INT AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    start_date DATE,
    due_date DATE,
    status TINYINT NOT NULL,
    priority TINYINT NOT NULL,
    description TEXT,
    PRIMARY KEY (task_id)
)  ENGINE=INNODB;
```

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/

https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Non-relational databases

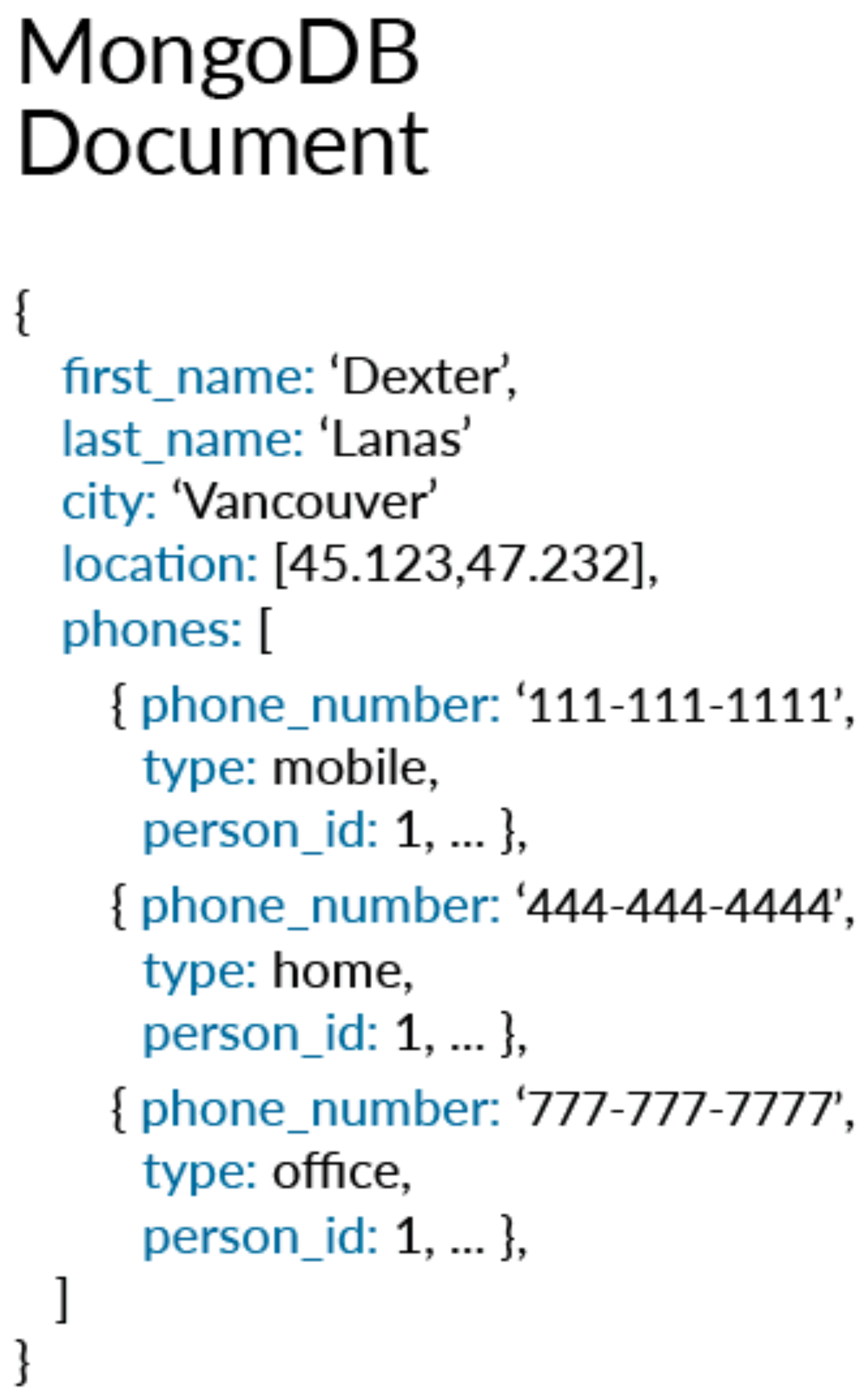
- Everything is organized into objects
- There are no restrictions on how objects are structured
- Every data entry is an object, or “document”
  - Documents may be structured differently from one another

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/

https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Non-relational databases



MongoDB
Document

```
{
  first_name: ‘Dexter’,
  last_name: ‘Lanas’
  city: ‘Vancouver’
  location: [45.123,47.232],
  phones: [
    { phone_number: ‘111-111-1111’,
      type: mobile,
      person_id: 1, ... },
    { phone_number: ‘444-444-4444’,
      type: home,
      person_id: 1, ... },
    { phone_number: ‘777-777-7777’,
      type: office,
      person_id: 1, ... },
  ]
}
```

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/

https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Non-relational databases

- There is no well-defined enforced structure
- That said, flatter structures are generally better

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/

https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Non-relational databases

```
{
  // This is a poorly nested data architecture, because iterating the children
  // of the "chats" node to get a list of conversation titles requires
  // potentially downloading hundreds of megabytes of messages
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "messages": {
        "m1": { "sender": "ghopper", "message": "Relay malfunction found. Cause: moth." },
        "m2": { ... },
        // a very long list of messages
      }
    },
    "two": { ... }
  }
}
```

https://firebase.google.com/docs/database/ios/structure-data

Databases

Non-relational databases

```
{
  // Chats contains only meta info about each conversation stored under the chats's unique ID
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "lastMessage": "ghopper: Relay malfunction found. Cause: moth."
    },
    "two": { ... }
  },
  // Messages are separate from data we may want to iterate quickly but still easily paginated and queried,
  // and organized by chat conversation ID
  "messages": {
    "one": {
      "m1": {
        "name": "eclarke",
        "message": "The relay seems to be malfunctioning."
      },
      "m2": { ... }
    },
    "two": { ... }
  }
}
```

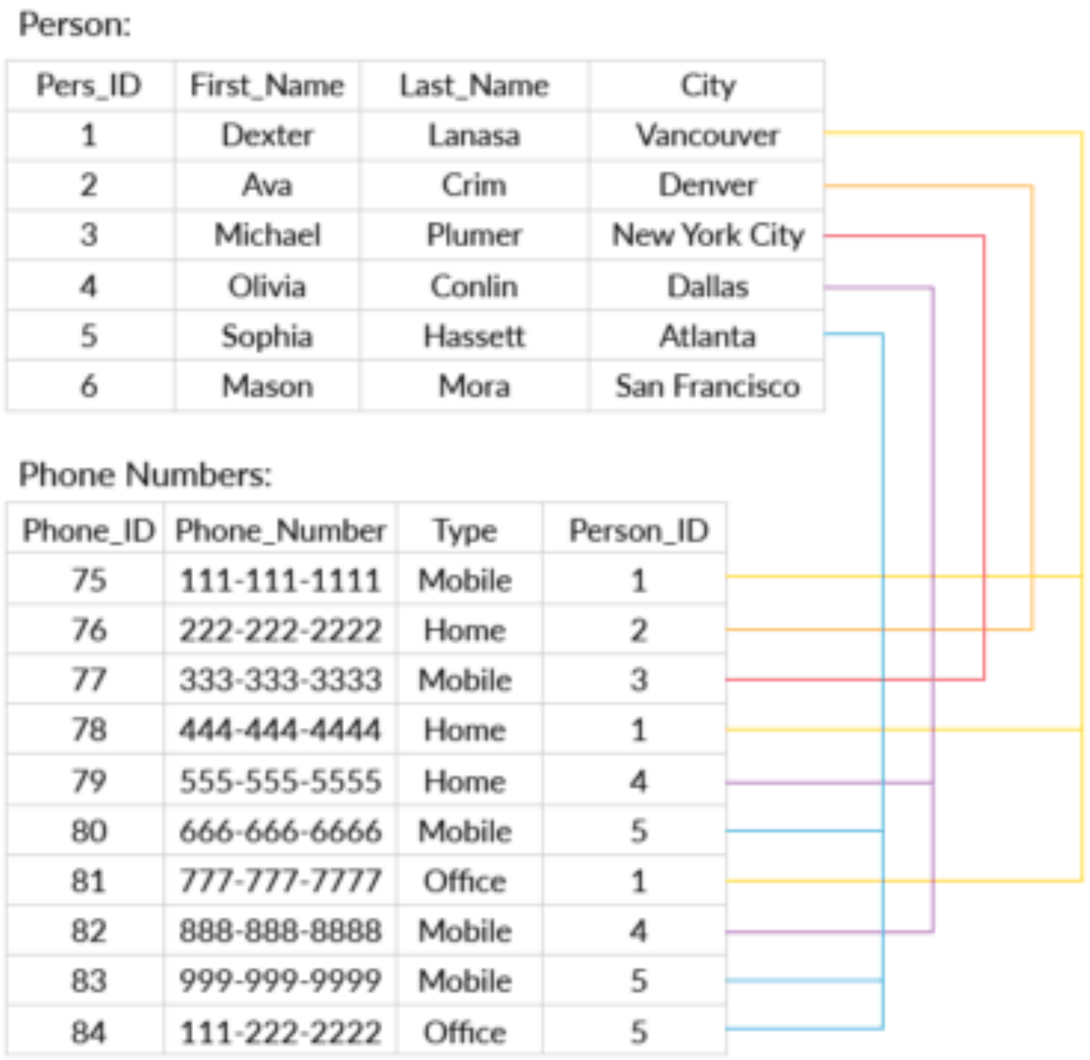
https://firebase.google.com/docs/database/ios/structure-data

Question

Which database structure will be best for retrieving all first names?

- Ⓐ The relational database
- Ⓑ The non-relational database
- Ⓒ They will be about the same
- Ⓓ I’m not sure
- Ⓔ [space intentionally left blank]

Relational



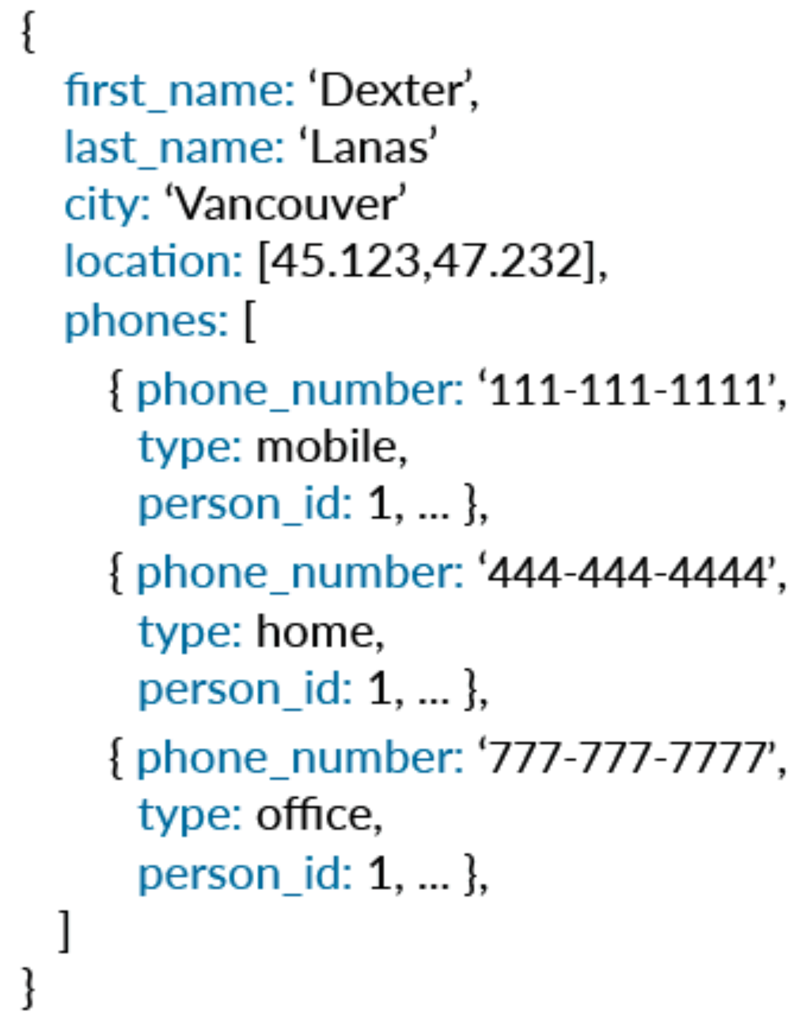
Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

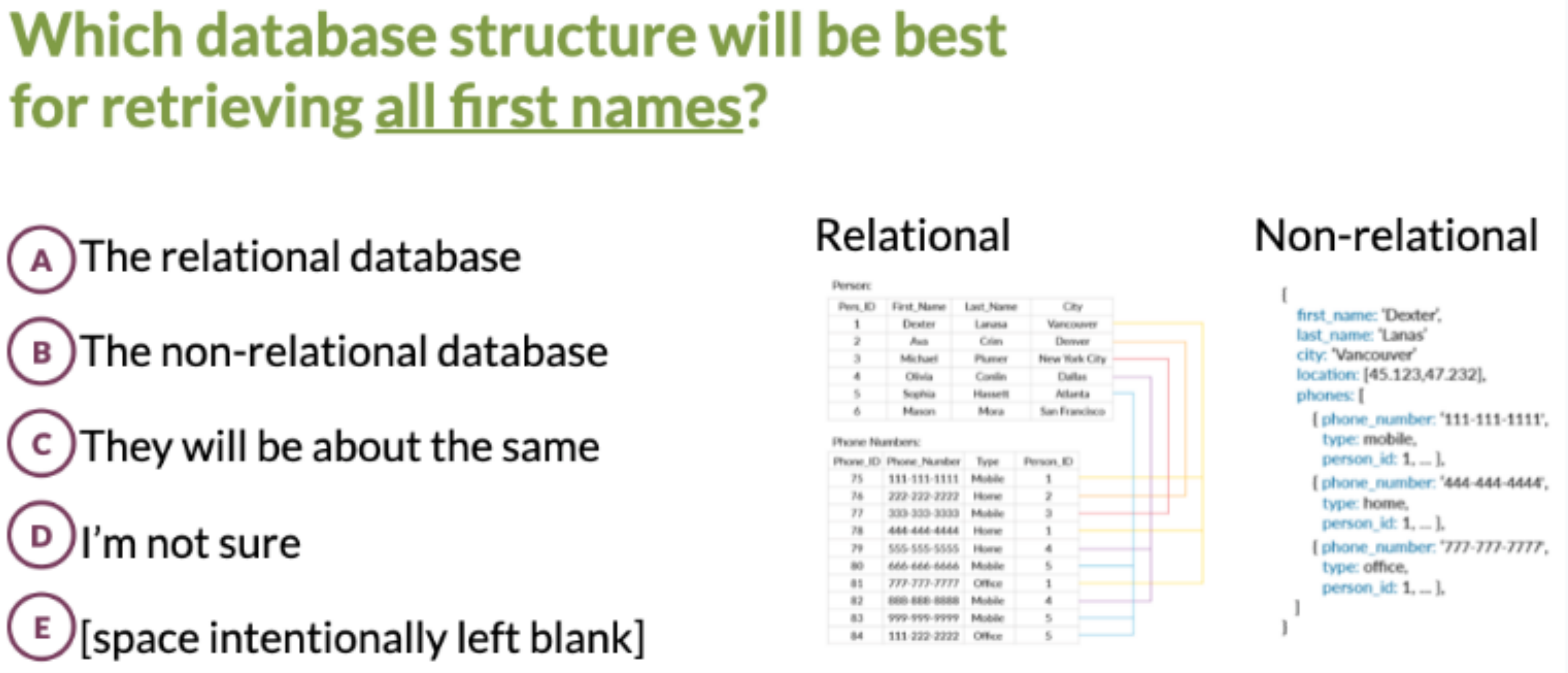
Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational



```
{
   first_name: ‘Dexter’,
   last_name: ‘Lanas’
   city: ‘Vancouver’
   location: [45.123,47.232],
   phones: [
      { phone_number: ‘111-111-1111’,
        type: mobile,
        person_id: 1, ... },
      { phone_number: ‘444-444-4444’,
        type: home,
        person_id: 1, ... },
      { phone_number: ‘777-777-7777’,
        type: office,
        person_id: 1, ... },
   ]
}
```



Which database structure will be best for retrieving all first names?

- Ⓐ The relational database
- Ⓑ The non-relational database
- Ⓒ They will be about the same
- Ⓓ I'm not sure
- Ⓔ [space intentionally left blank]

Relational

Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational

```
{
  first_name: 'Dexter',
  last_name: 'Lanas'
  city: 'Vancouver'
  location: [45.123,47.232],
  phones: [
    { phone_number: '111-111-1111',
      type: mobile,
      person_id: 1, ... },
    { phone_number: '444-444-4444',
      type: home,
      person_id: 1, ... },
    { phone_number: '777-777-7777',
      type: office,
      person_id: 1, ... },
  ]
}
```

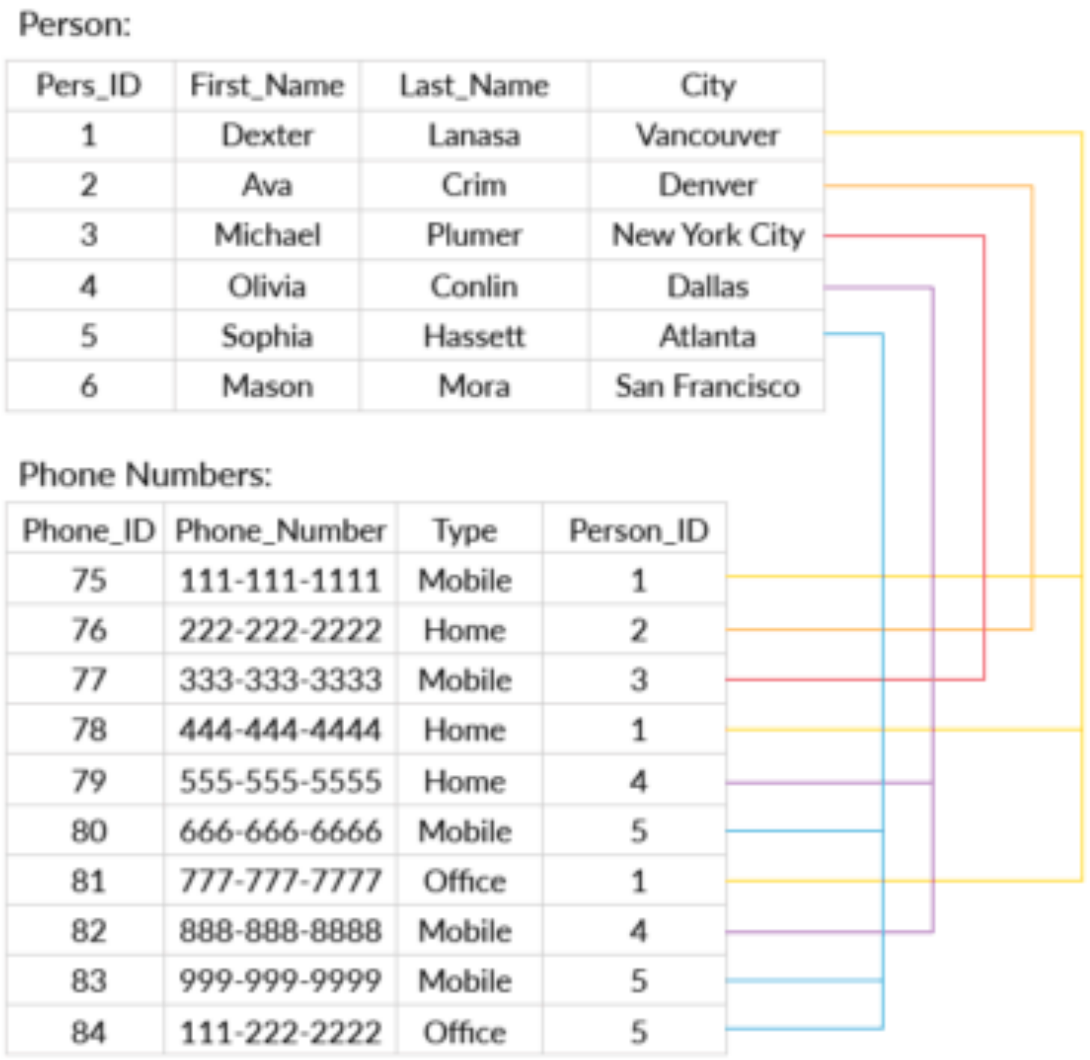
A	0%
B	0%
C	0%
D	0%
E	0%

Question

Which database structure will be best for retrieving all first names?

- A The relational database
- B The non-relational database
- C They will be about the same
- D I’m not sure
- E [space intentionally left blank]

Relational



Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational

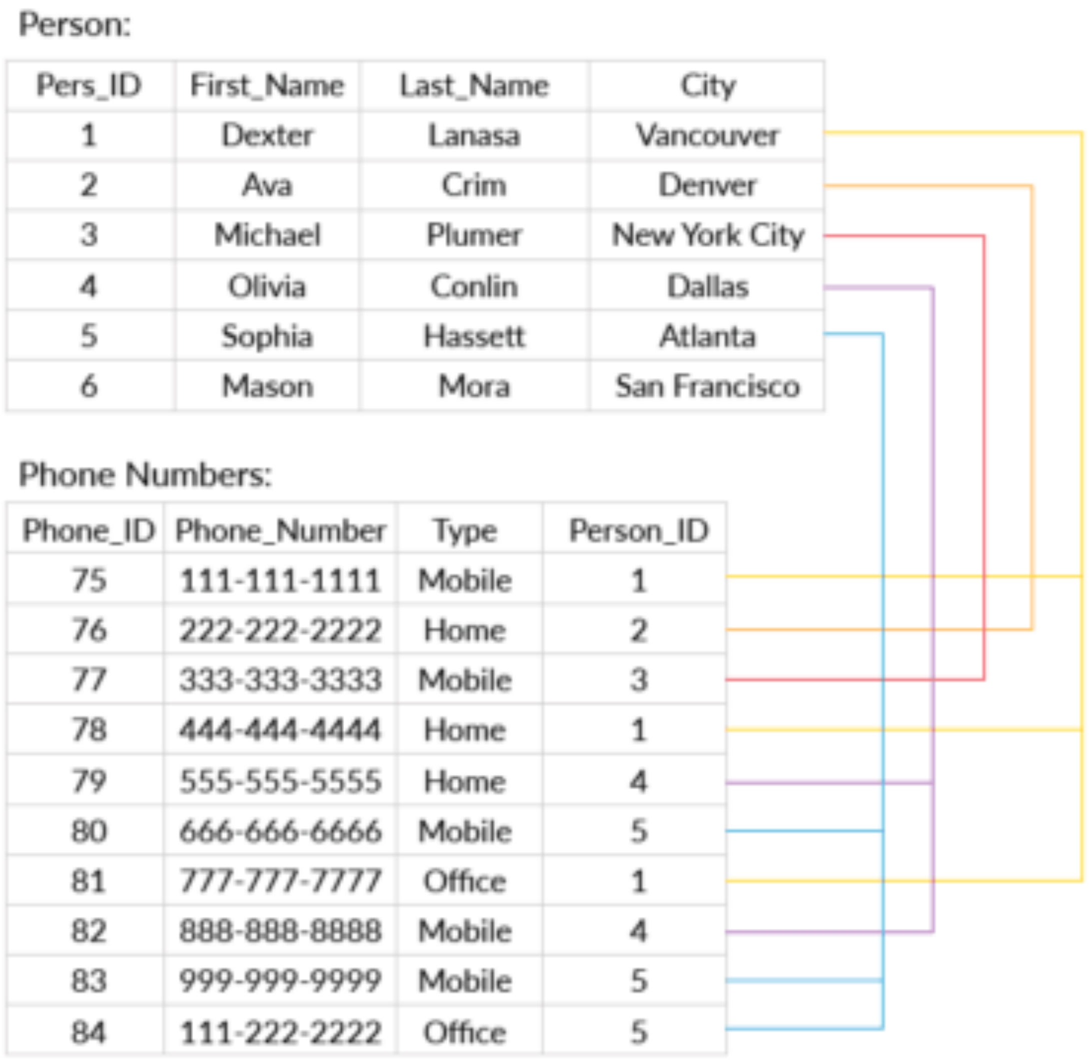
```
{
   first_name: ‘Dexter’,
   last_name: ‘Lanas’
   city: ‘Vancouver’
   location: [45.123,47.232],
   phones: [
      { phone_number: ‘111-111-1111’,
        type: mobile,
        person_id: 1, ... },
      { phone_number: ‘444-444-4444’,
        type: home,
        person_id: 1, ... },
      { phone_number: ‘777-777-7777’,
        type: office,
        person_id: 1, ... },
   ]
}
```

Question

Which database structure will be best for retrieving all phone numbers?

- (A) The relational database
- (B) The non-relational database
- (C) They will be about the same
- (D) I'm not sure
- (E) [space intentionally left blank]

Relational



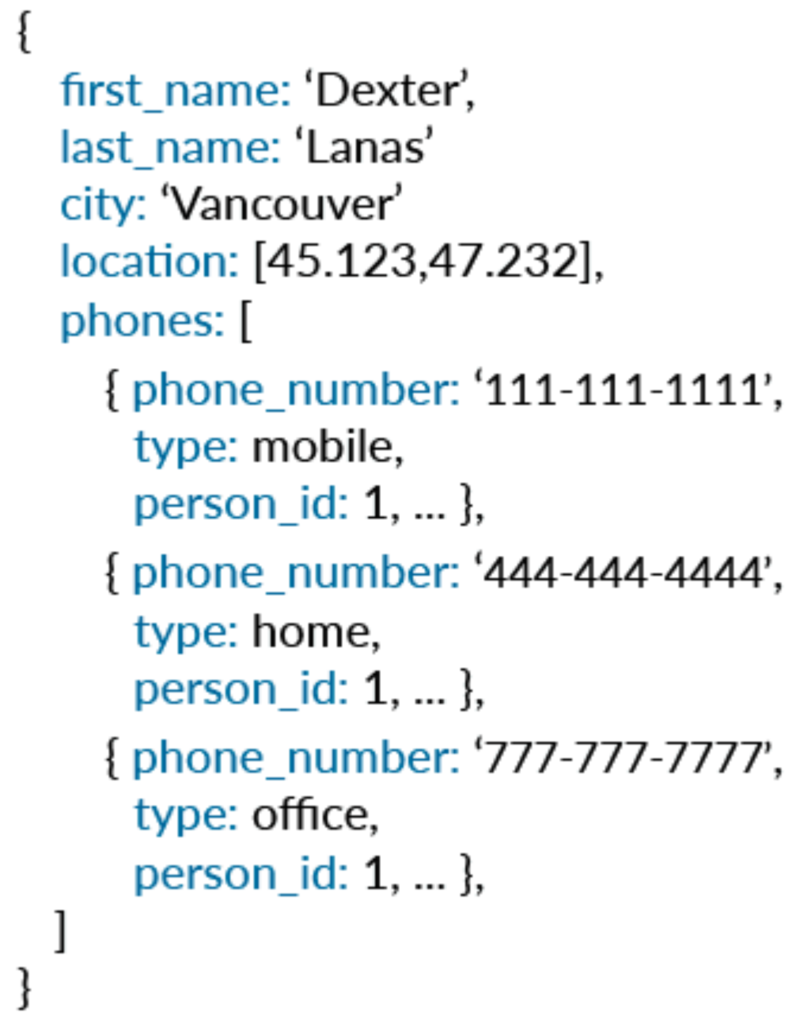
Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

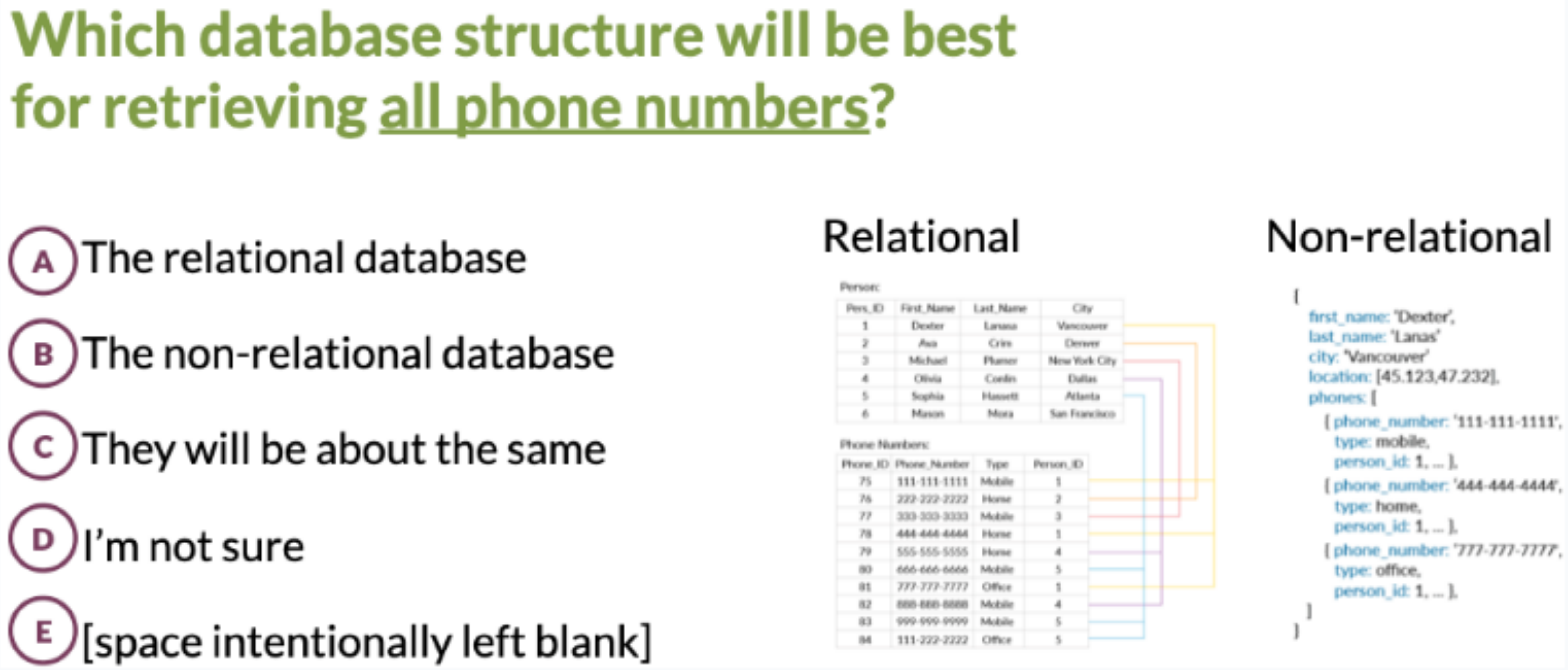
Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational



```
{
  first_name: 'Dexter',
  last_name: 'Lanas'
  city: 'Vancouver'
  location: [45.123,47.232],
  phones: [
    { phone_number: '111-111-1111',
      type: mobile,
      person_id: 1, ... },
    { phone_number: '444-444-4444',
      type: home,
      person_id: 1, ... },
    { phone_number: '777-777-7777',
      type: office,
      person_id: 1, ... },
  ]
}
```



Which database structure will be best for retrieving all phone numbers?

Ⓐ The relational database

Ⓑ The non-relational database

Ⓒ They will be about the same

Ⓓ I’m not sure

Ⓔ [space intentionally left blank]

Relational

Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Cordin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational

```
{
  first_name: 'Dexter',
  last_name: 'Lanas'
  city: 'Vancouver'
  location: [45.123,47.232],
  phones: [
    { phone_number: '111-111-1111',
      type: mobile,
      person_id: 1, ... },
    { phone_number: '444-444-4444',
      type: home,
      person_id: 1, ... },
    { phone_number: '777-777-7777',
      type: office,
      person_id: 1, ... },
  ]
}
```

A 0%

B 0%

C 0%

D 0%

E 0%

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Question

Which database structure will be best for retrieving all phone numbers?

[✓ A] The relational database

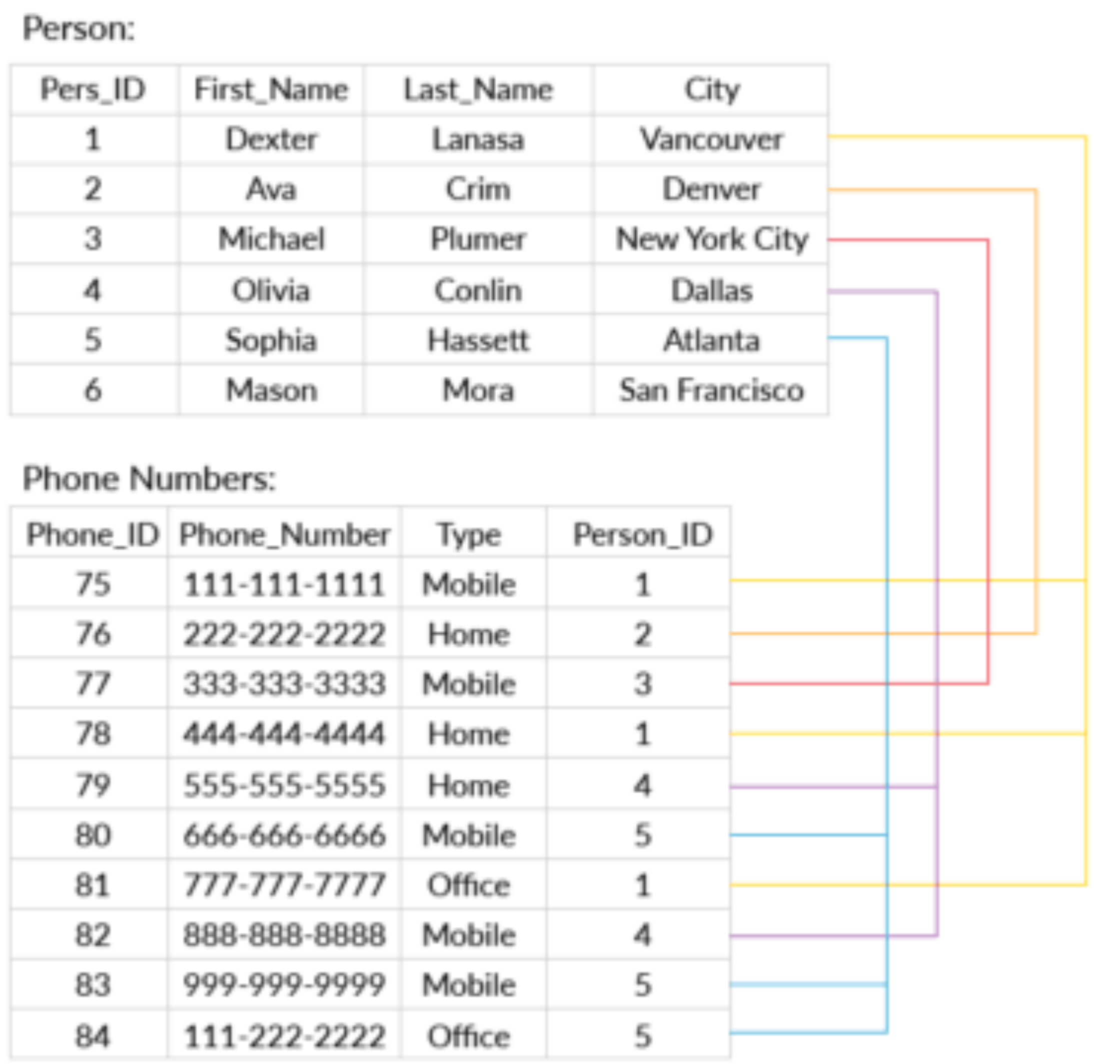
B The non-relational database

C They will be about the same

D I’m not sure

E [space intentionally left blank]

Relational



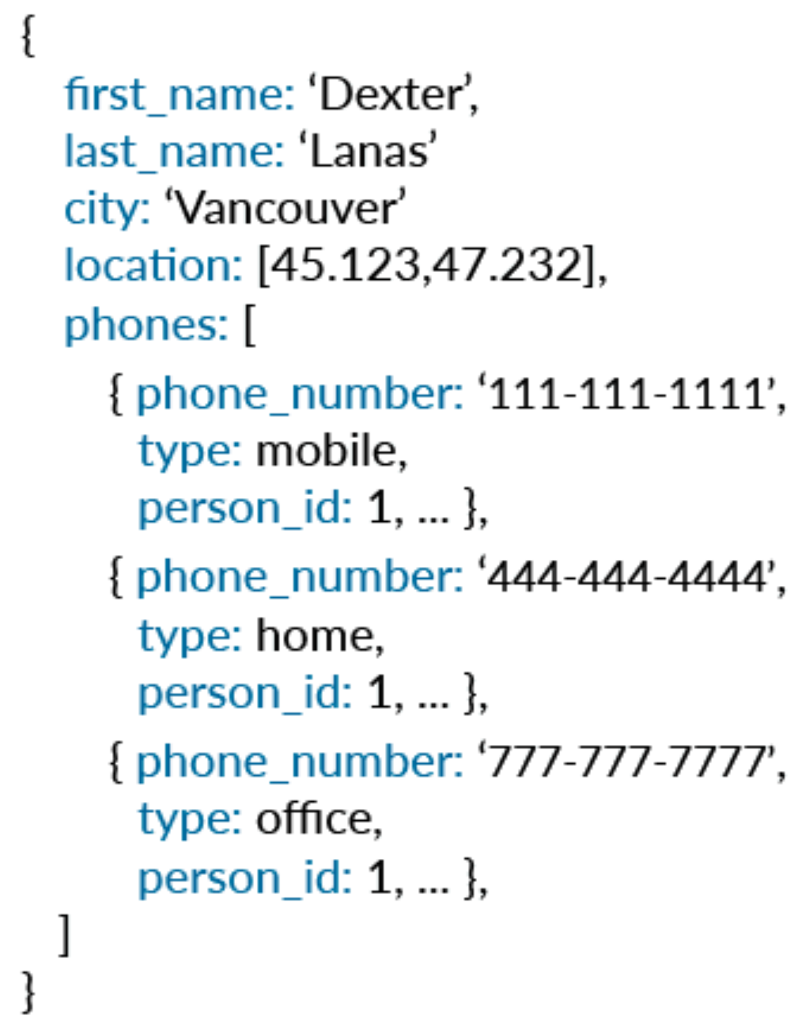
Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational



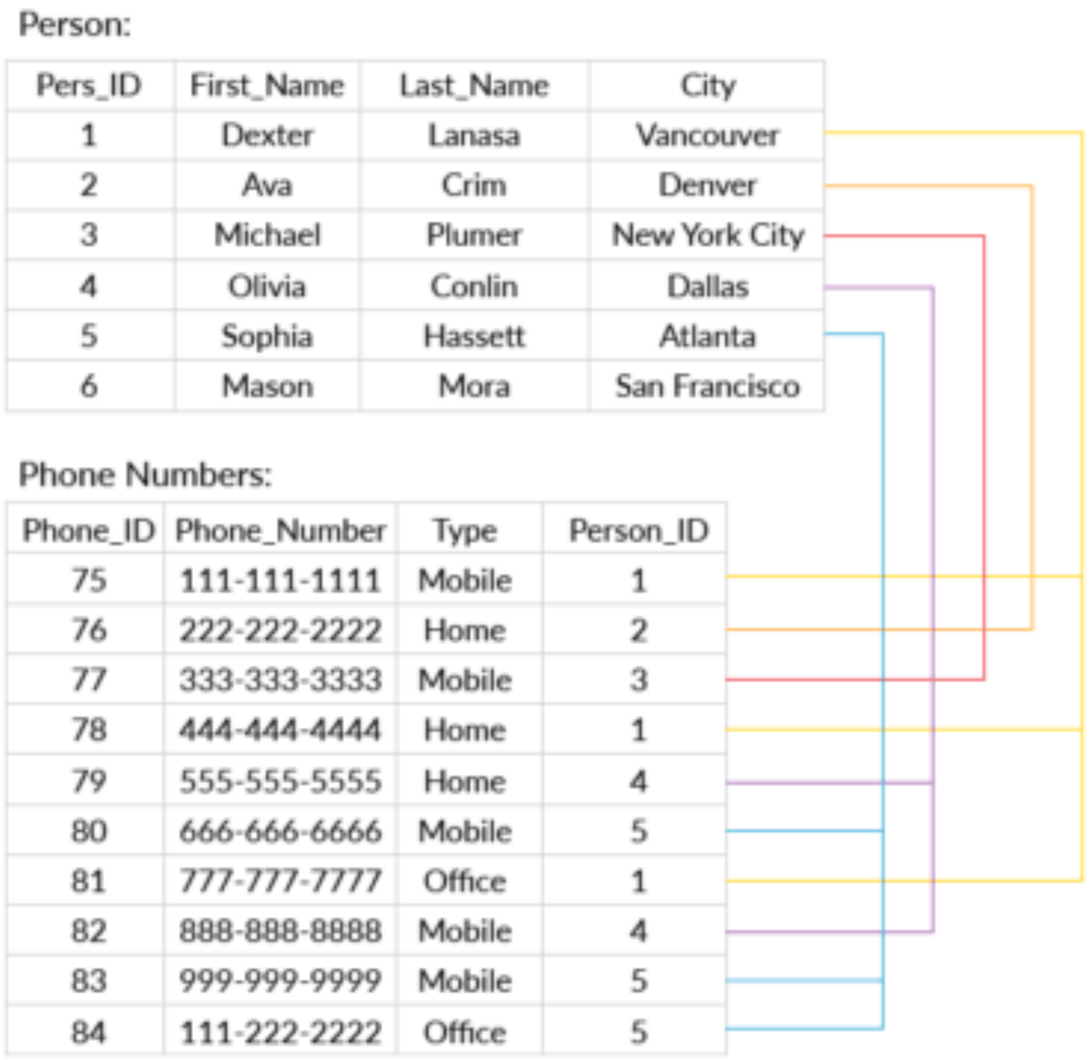
```
{
   first_name: ‘Dexter’,
   last_name: ‘Lanas’
   city: ‘Vancouver’
   location: [45.123,47.232],
   phones: [
      { phone_number: ‘111-111-1111’,
        type: mobile,
        person_id: 1, ... },
      { phone_number: ‘444-444-4444’,
        type: home,
        person_id: 1, ... },
      { phone_number: ‘777-777-7777’,
        type: office,
        person_id: 1, ... },
   ]
}
```

Question

Which database structure will be best for retrieving all data?

- Ⓐ The relational database
- Ⓑ The non-relational database
- Ⓒ They will be about the same
- Ⓓ I’m not sure
- Ⓔ [space intentionally left blank]

Relational



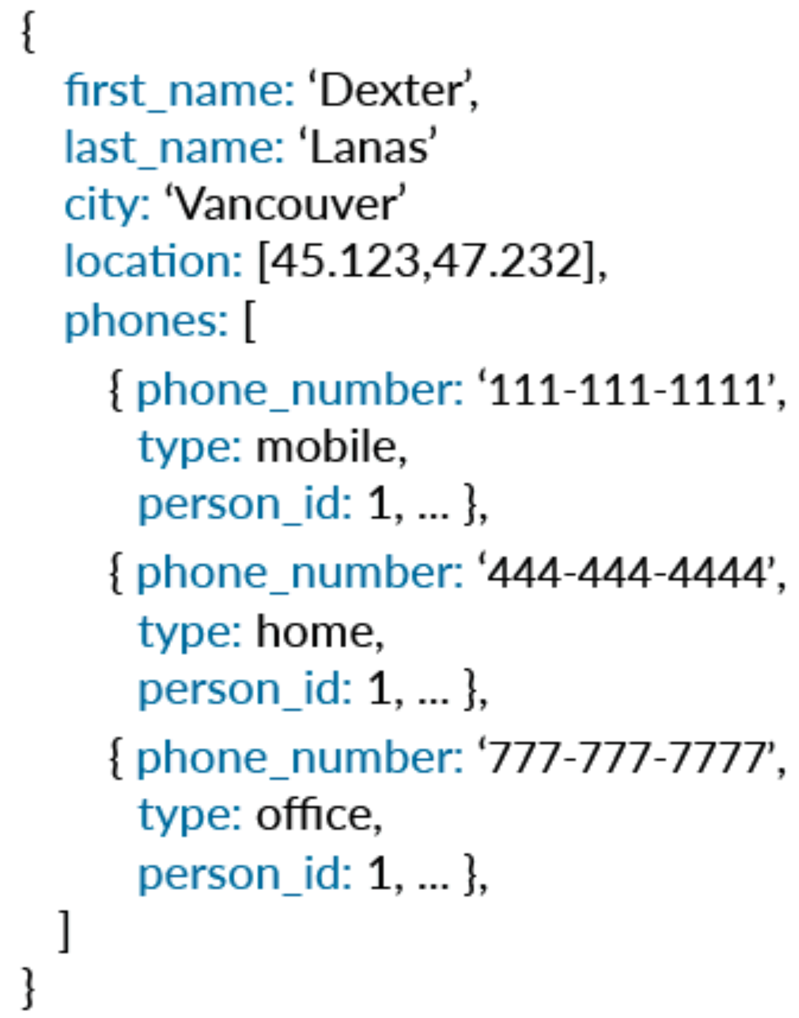
Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

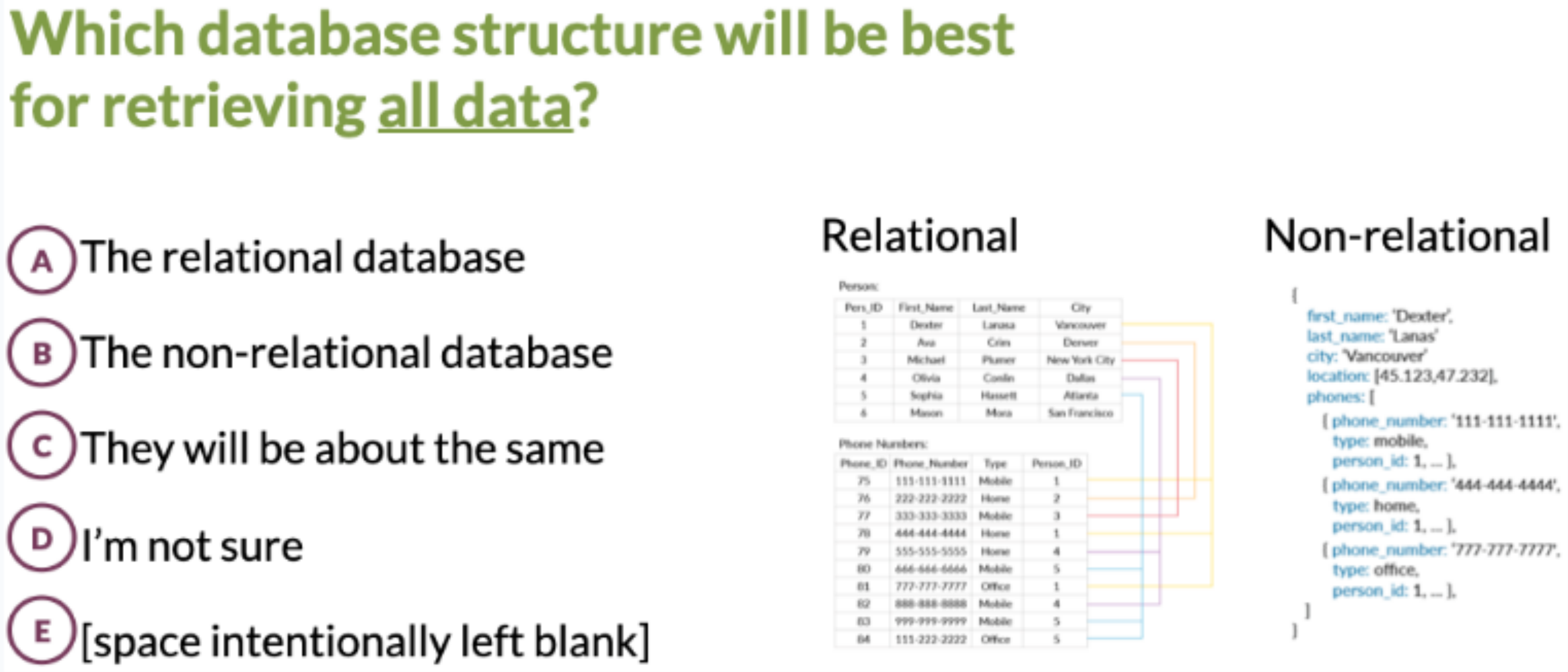
Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational



```
{
  first_name: ‘Dexter’,
  last_name: ‘Lanas’
  city: ‘Vancouver’
  location: [45.123,47.232],
  phones: [
    { phone_number: ‘111-111-1111’,
      type: mobile,
      person_id: 1, … },
    { phone_number: ‘444-444-4444’,
      type: home,
      person_id: 1, … },
    { phone_number: ‘777-777-7777’,
      type: office,
      person_id: 1, … },
  ]
}
```



Which database structure will be best for retrieving all data?

- Ⓐ The relational database
- Ⓑ The non-relational database
- Ⓒ They will be about the same
- Ⓓ I’m not sure
- Ⓔ [space intentionally left blank]

Relational

Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational

```
{
  first_name: 'Dexter',
  last_name: 'Lanas'
  city: 'Vancouver'
  location: [45.123,47.232],
  phones: [
    { phone_number: '111-111-1111',
      type: mobile,
      person_id: 1, ... },
    { phone_number: '444-444-4444',
      type: home,
      person_id: 1, ... },
    { phone_number: '777-777-7777',
      type: office,
      person_id: 1, ... },
  ]
}
```

A	0%
B	0%
C	0%
D	0%
E	0%

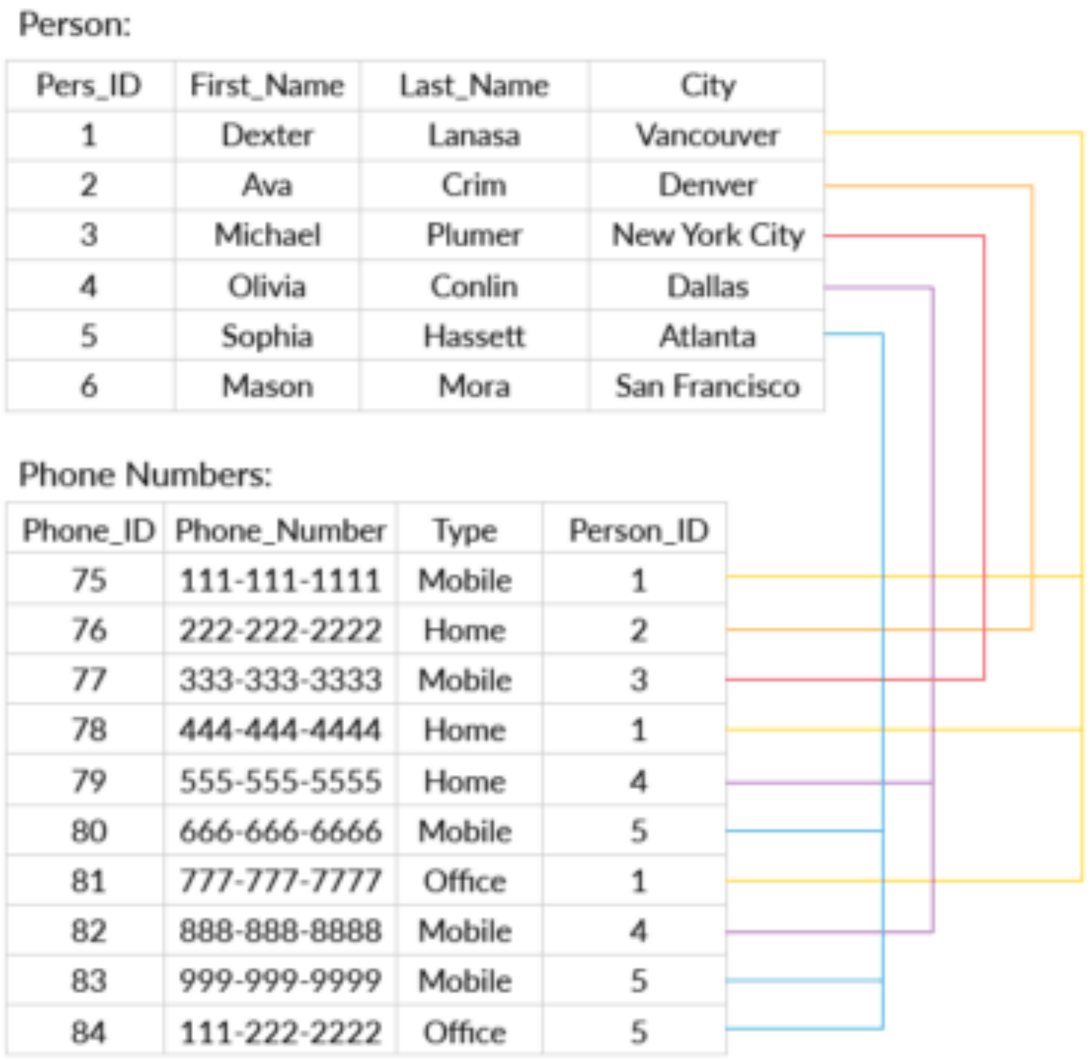
Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Question

Which database structure will be best for retrieving all data?

- Ⓐ The relational database
- [✓ Ⓑ] The non-relational database
- Ⓒ They will be about the same
- Ⓓ I’m not sure
- Ⓔ [space intentionally left blank]

Relational



Person:

Pers_ID	First_Name	Last_Name	City
1	Dexter	Lanasa	Vancouver
2	Ava	Crim	Denver
3	Michael	Plumer	New York City
4	Olivia	Conlin	Dallas
5	Sophia	Hassett	Atlanta
6	Mason	Mora	San Francisco

Phone Numbers:

Phone_ID	Phone_Number	Type	Person_ID
75	111-111-1111	Mobile	1
76	222-222-2222	Home	2
77	333-333-3333	Mobile	3
78	444-444-4444	Home	1
79	555-555-5555	Home	4
80	666-666-6666	Mobile	5
81	777-777-7777	Office	1
82	888-888-8888	Mobile	4
83	999-999-9999	Mobile	5
84	111-222-2222	Office	5

Non-relational

```
{
  first_name: ‘Dexter’,
  last_name: ‘Lanas’
  city: ‘Vancouver’
  location: [45.123,47.232],
  phones: [
    { phone_number: ‘111-111-1111’,
      type: mobile,
      person_id: 1, … },
    { phone_number: ‘444-444-4444’,
      type: home,
      person_id: 1, … },
    { phone_number: ‘777-777-7777’,
      type: office,
      person_id: 1, … },
  ]
}
```

Databases

Advantages of relational databases

- Relational databases support better querying
  - Provide *languages* for querying, such as Structured Query Language (SQL)
  - Those languages can be used to ask for specific tables or even join data across tables
  - "Give me the first name of every user whose phone number starts with 949"

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/

https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Advantages of relational databases

- Relational databases are more organized
  - Because field types are defined, data reliably follows that structure
- Relational databases are more reliable
  - Structure is enforced when new data is added
  - Transactions are atomic, so it’s easy to “get” the current state of the database

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/

https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Advantages of non-relational databases

- Non-relational databases support more flexibility
    - Structure imposes restrictions
    - Adding a new field (column) can mess up a relational database
- Non-relational databases are faster for simple operations
    - It’s much easier to “watch all the files” than to query and index many rows across multiple tables

https://www.neonrain.com/blog/mysql-vs-mongodb-looking-at-relational-and-non-relational-databases/

https://www.mongodb.com/scale/relational-vs-non-relational-database

Databases

Relational vs. Non-relational

- Relational databases tend to be used in Enterprise, large-scale applications
  - It’s important that data conforms to standards
  - It’s important to robustly query large amounts of data
- Non-relational databases tend to be used in smaller applications
  - Data flexibility is valuable
  - Data is small enough to reliably retrieve and parse
- That said, plenty of large apps use non-relational databases and vice versa

Databases vs. Local Storage

- Who needs access to the data?
  - Just the user, or others?
  - As a developer, do you need access?
- Is the data sensitive?
- Is the data valuable enough that it should not be lost?

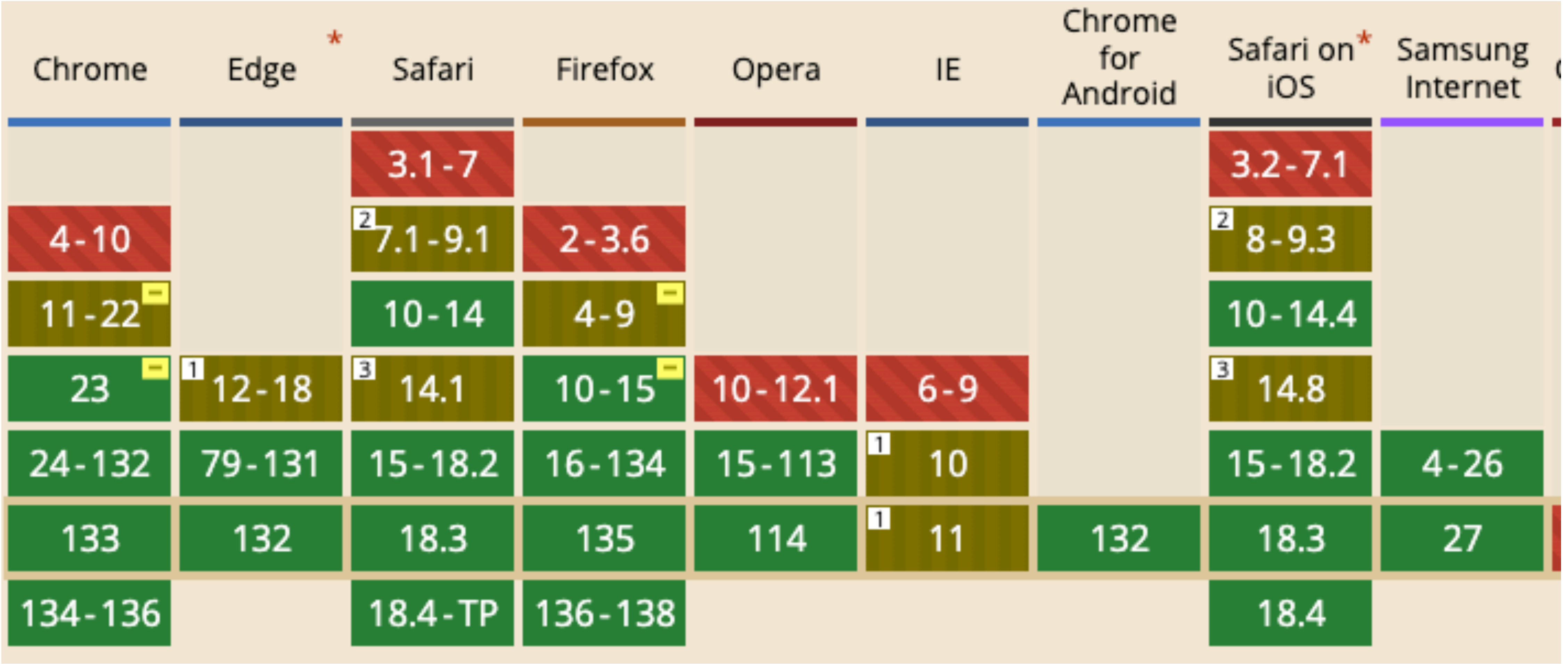
Databases vs. Local Storage

- Databases are crucial if more than the local device needs access
  - Cross-device app: facebook.com and the mobile app need your profile information
  - Developer: to understand habits across users or provide a data-driven service
- Some privacy can be preserved if data is only stored locally
- Which to use depends on the type of data and context

One non-relational database: IndexedDB

IndexedDB

- Proposed in 2015, standardized in 2021
- Native database support in browsers
- Pretty good compatibility these days
- Apple had proposed a relational version (SQLite), but other browsers declined to implement it

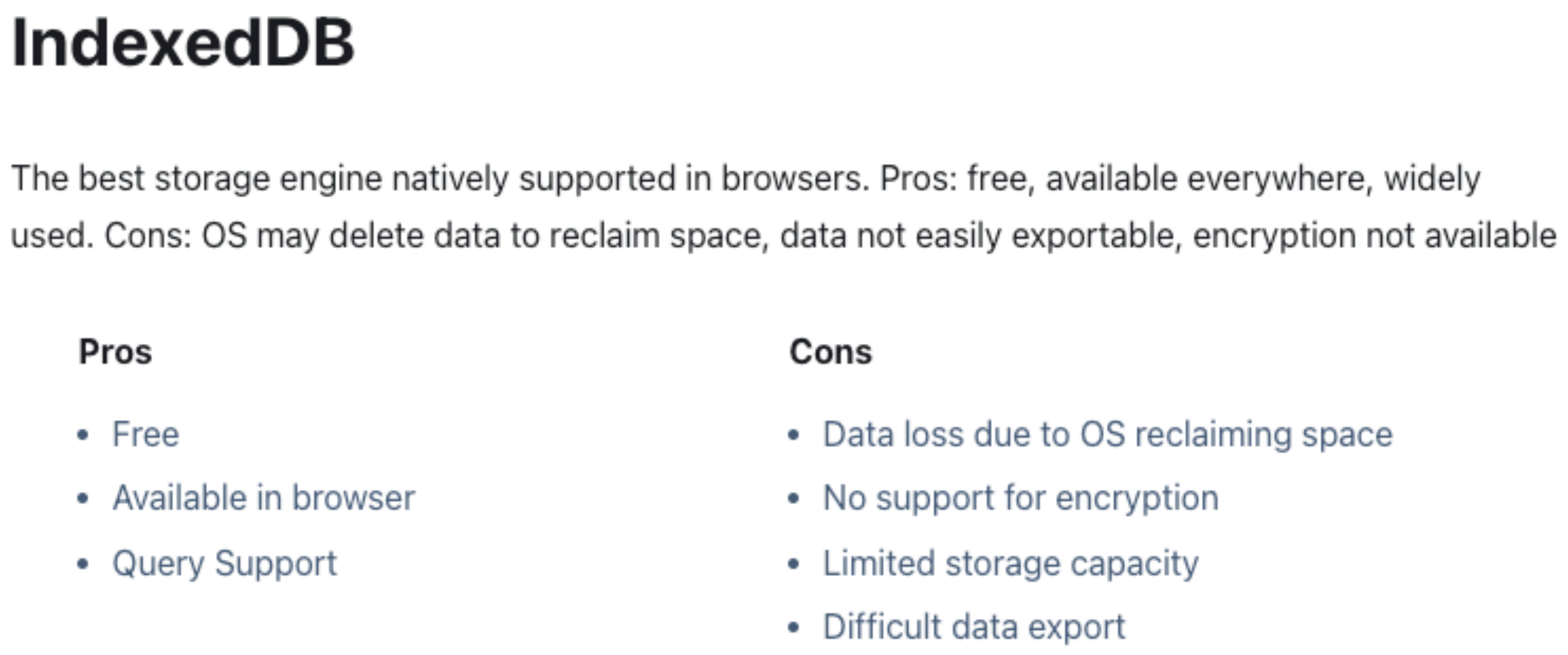


Chrome	Edge *	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS *	Samsung Internet
		3.1-7					3.2-7.1	
4-10		[2] 7.1-9.1	2-3.6				[2] 8-9.3	
11-22 −		10-14	4-9 −				10-14.4	
23 −	[1] 12-18	[3] 14.1	10-15 −	10-12.1	6-9		[3] 14.8	
24-132	79-131	15-18.2	16-134	15-113	[1] 10		15-18.2	4-26
133	132	18.3	135	114	[1] 11	132	18.3	27
134-136		18.4-TP	136-138				18.4	

https://caniuse.com/?search=indexeddb

IndexedDB

- Client-side database
  - No communication across devices, not "cloud" storage
  - If your phone dies, the data is gone
- Some concerns about space, and operating system reclaiming space



IndexedDB

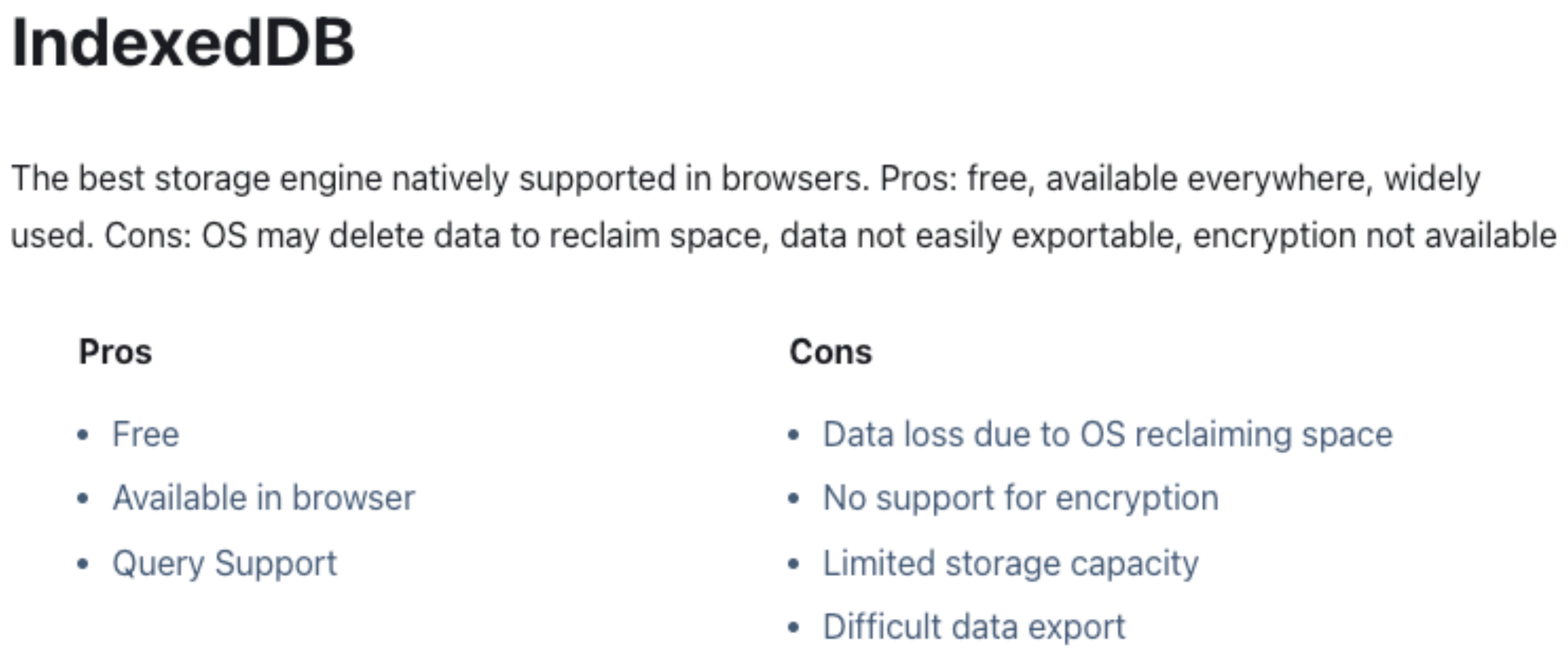
The best storage engine natively supported in browsers. Pros: free, available everywhere, widely used. Cons: OS may delete data to reclaim space, data not easily exportable, encryption not available

Pros	Cons
• Free	• Data loss due to OS reclaiming space
• Available in browser	• No support for encryption
• Query Support	• Limited storage capacity
	• Difficult data export

https://ionic.io/enterprise-guide/data-storage

IndexedDB

- So, what database advantages is it really providing?
  - Some structure and organization, primarily
  - A simpler API for demo purposes
  - Look at other classes for more examples or depth :-)



IndexedDB

The best storage engine natively supported in browsers. Pros: free, available everywhere, widely used. Cons: OS may delete data to reclaim space, data not easily exportable, encryption not available

Pros

- Free
- Available in browser
- Query Support

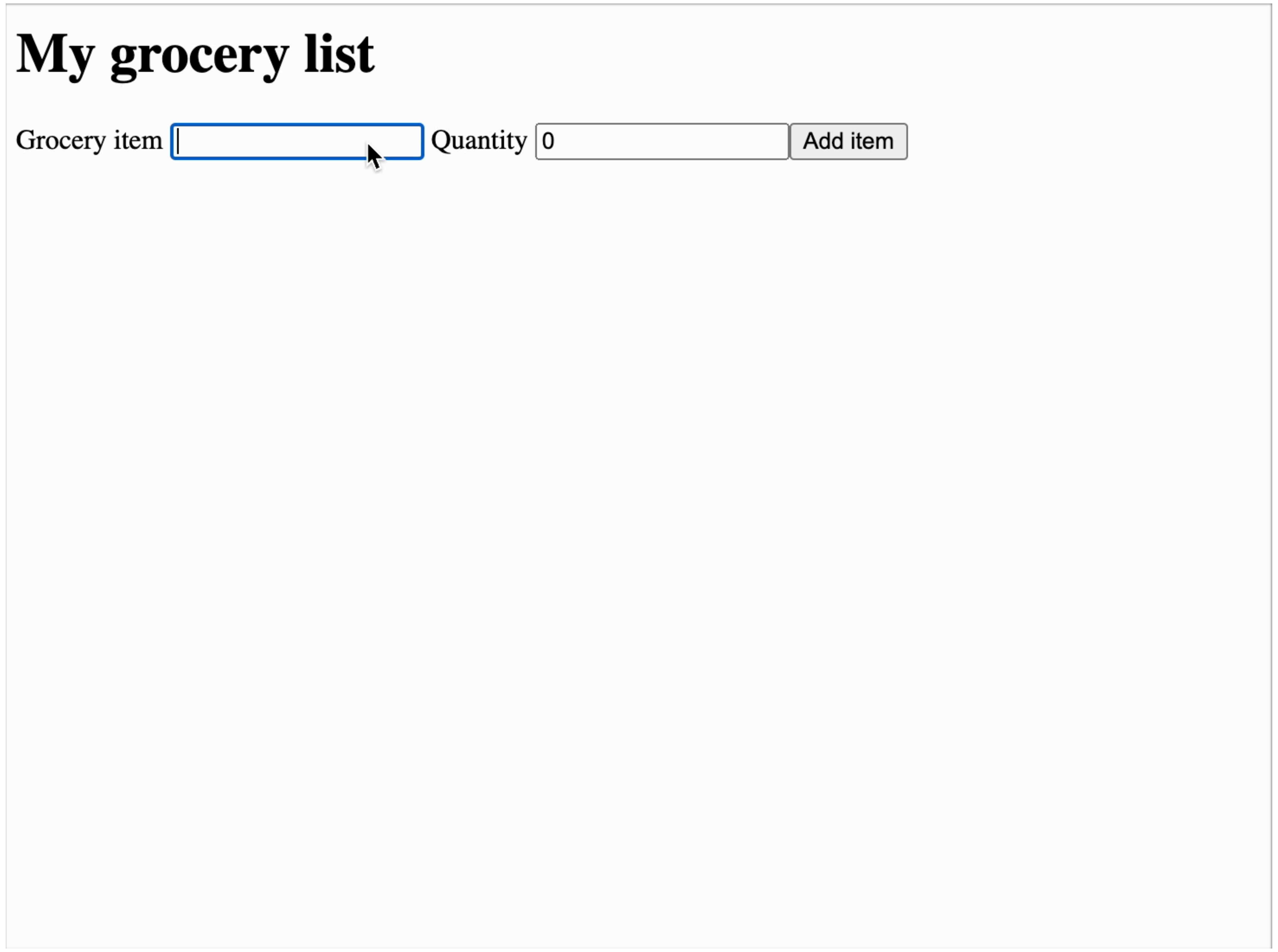
Cons

- Data loss due to OS reclaiming space
- No support for encryption
- Limited storage capacity
- Difficult data export

https://ionic.io/enterprise-guide/data-storage

IndexedDB

Getting some data



My grocery list

Grocery item [ ] Quantity [0] Add item

IndexedDB

Add item

```
this.dbService.add(‘objectStore', {item}).subscribe((value) => {
    console.log(value);
  });
}
```

IndexedDB

Get values

```
this.dbService.getAll(‘objectStore').subscribe((objects) => {
  objects.forEach((object:any) => {
    console.log(object);
  })
});
```

IndexedDB

Other methods

- Other library functions can implement other database methods
  - Update
  - Delete
  - Clear database

https://www.npmjs.com/package/ngx-indexed-db

IndexedDB

Database Configuration

```
Store schema, id field that auto-increments
const dbConfig: DBConfig  = {
  name: 'db',
  version: 1,
  objectStoresMeta: [{
    store: 'objStore',
    storeConfig: { keyPath: 'id', autoIncrement: true },
    storeSchema: [
      { name: 'name', keypath: 'name', options: { unique: false } },
      ...
    ]
  }]
};
```

IndexedDB

Is it really non-relational?

- It is technically an “Object database”, which stores data as objects
    - Think Object-oriented programming
- Relational databases store data as tables
    - Think rows/columns, like Google Sheets or Excel
- Because objects are flexible in structure, object databases are typically considered non-relational
    - But, the config still names fields

https://en.wikipedia.org/wiki/Object_database

Today’s goals

By the end of today, you should be able to…

- Differentiate relational from non-relational databases
- Explain the advantages of each style of database
- Use IndexedDB to implement a non-relational database

IN4MATX 133: User Interface Software

Lecture 16:
Databases and Local Storage