# Colonial Council Bank

After the success of the Colonial Journey, the Council has decided to establish a bank – Colonial Council Bank. You have been employed by the Council to finish the database layer, which supports basic functionality like importing JSON and XML data and exporting some results.

## Functionality Overview

The Council has hired you as their application developer, to implement the **database layer**. The application should be able to easily **import** hard-formatted data from **XML** and **JSON** and **support functionality** for also **exporting** the imported data. The application is called – **ccb**.

Look at the pictures below to see what must happen:

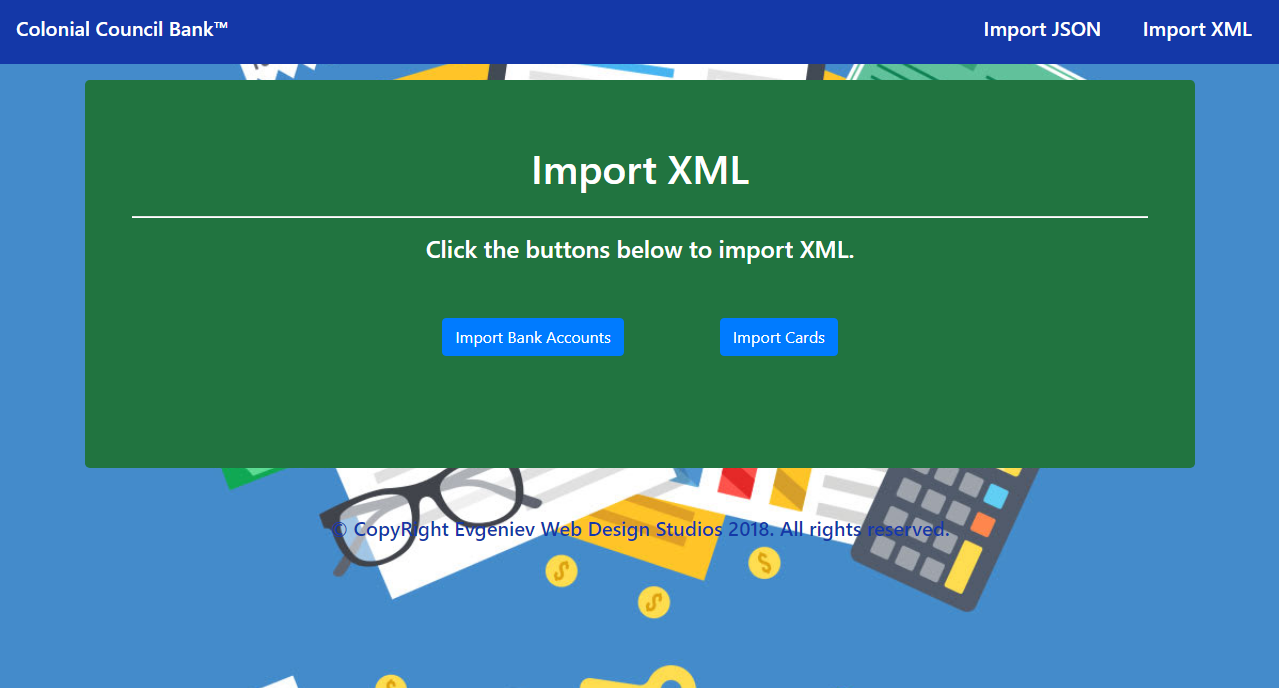
* Home page before importing anything:



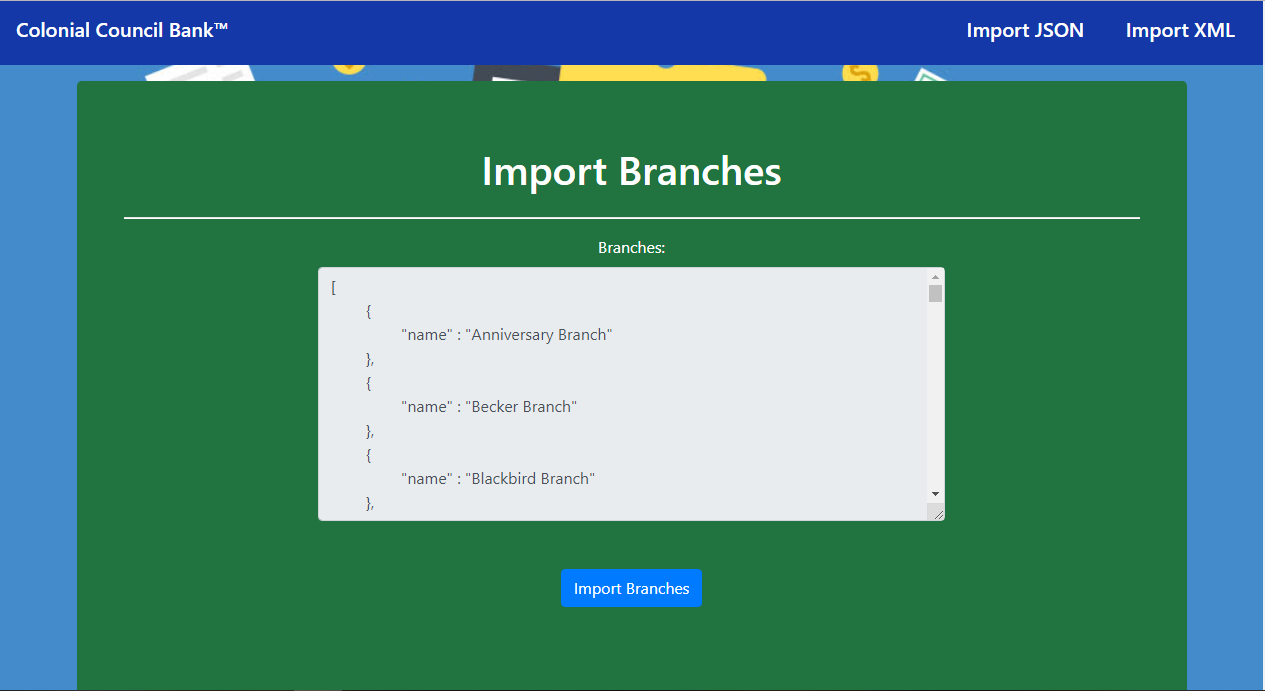
* Import JSON page before importing anything:



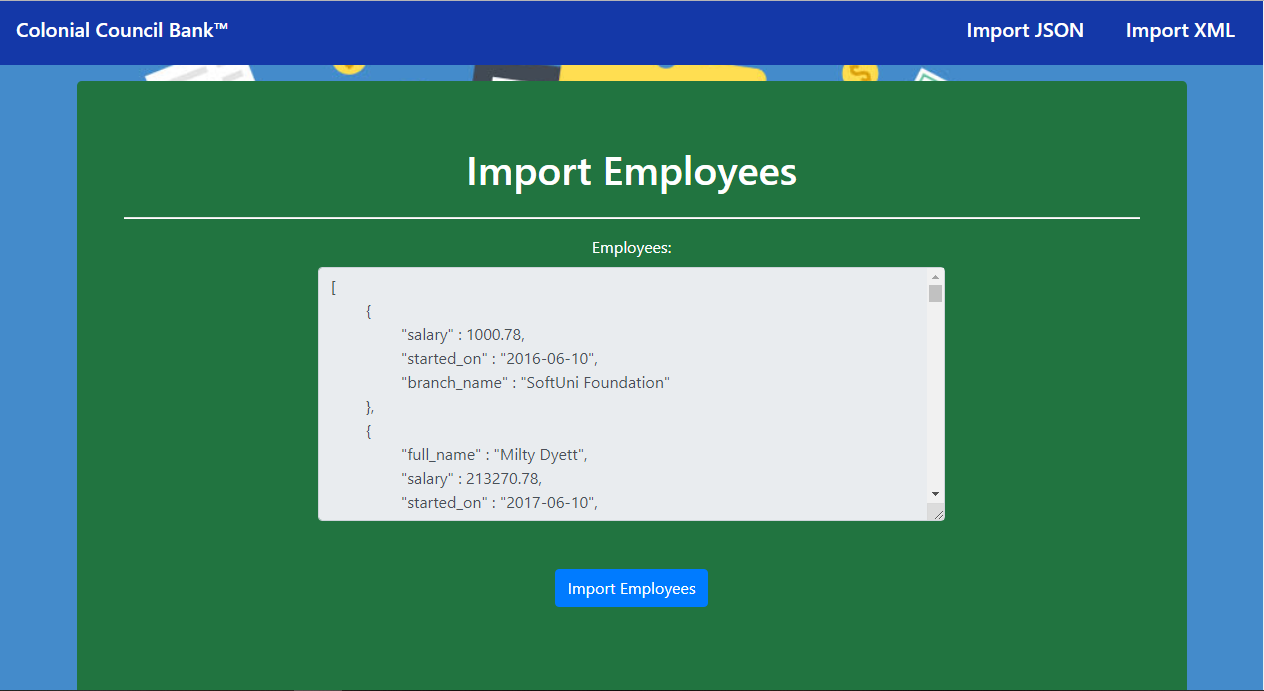
* Import XML page before importing anything:



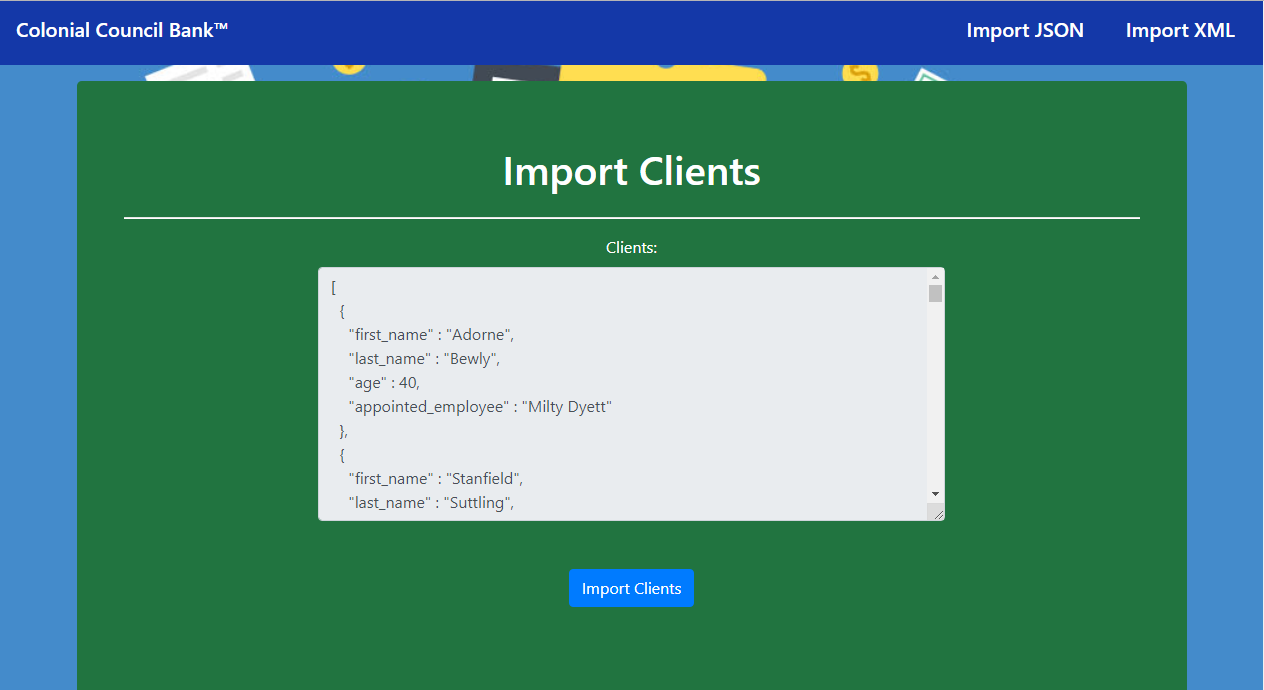
* Import Branches page after reading the **branches.json** file:



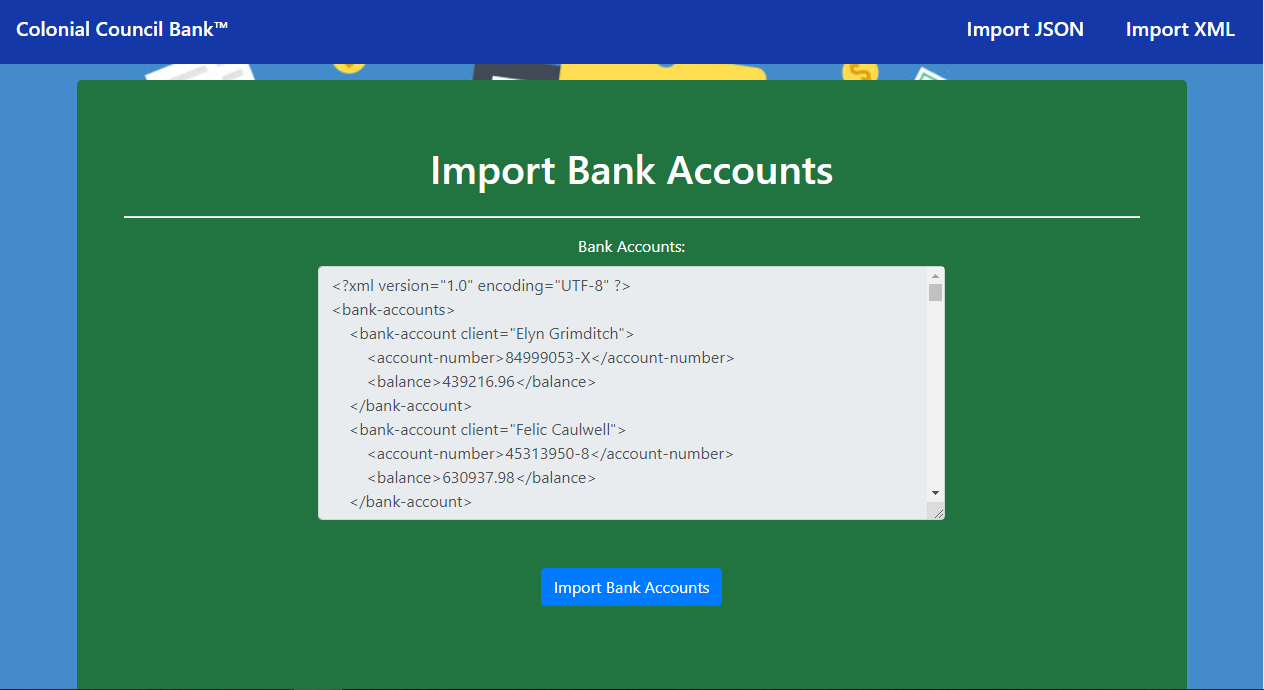
* Import Employees page after reading the **employees.json** file:



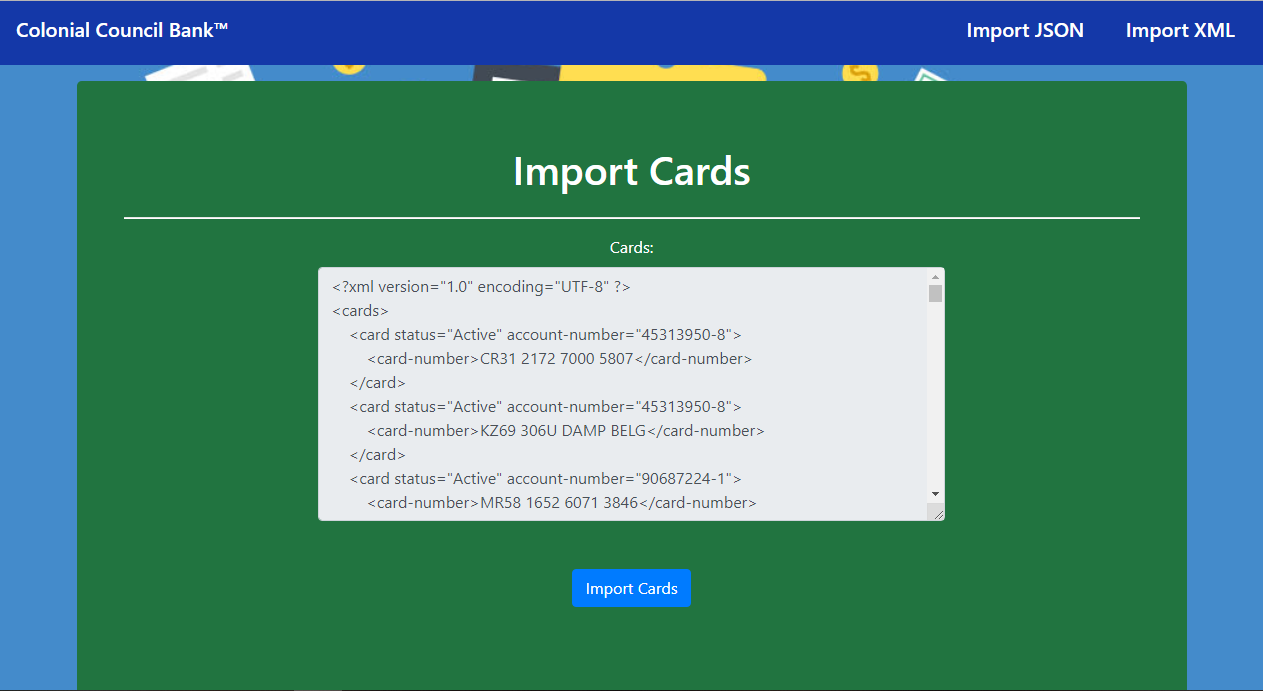
* Import Clients page after reading the **clients.json** file:



* Import Bank Accounts page after reading **bank-accounts.xml** file:



* Import Cards page after reading **cards.xml** file:



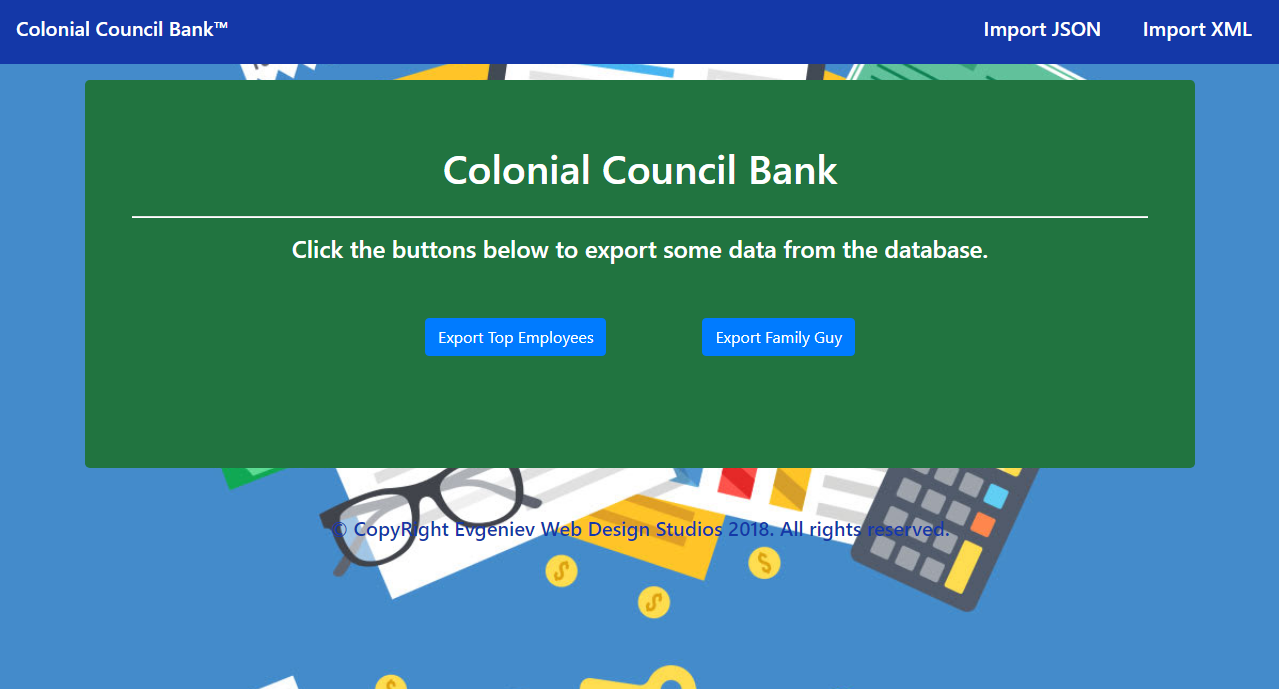
* Import JSON page after importing the given data:



* Import XML page after importing the given data:



* Home page after importing the given data:



* Export Top Employees page:



* Export Family Guy page:



## Project Skeleton Overview

You will be given a **Skeleton**, containing a **certain architecture(MVC)** with **several classes**, some of which – completely empty. The **Skeleton** will include the **files** with which you will **seed** the **database**.

## Model Definition

There are 5 main models that the **ccb** database application should contain in its functionality.

Design them in the **most appropriate** way, considering the following **data constraints**:

### Branch

* id – **integer** number, **primary identification field**.
* name – a **string** (**required**).

### Employee

* id – **integer** number, **primary identification field**.
* first\_name – a **string** (**required**).
* last\_name – a **string** (**required**).
* salary – a **decimal** data type.
* started\_on – a **Date**.
* branch – a Branch entity (**required**).
* clients – a **collection** of Client entity.

### Clients

* id – **integer** number, **primary identification field**.
* full\_name – a **string** (**required**).
* age – an **integer** number.
* bank\_account – a Bank Account entity (**One**).

### Bank Account

* id – **integer** number, **primary identification field**.
* account\_number – a **string** (**required**).
* balance – a **decimal** data type.
* client – a Client entity (One).
* cards – a **collection** of Card entity.

### Card

* id – **integer** number, **primary identification field**.
* card\_number – a **string** (**required**).
* card\_status – a **string** (**required**).
* bank\_account – a Bank Account entity.

**NOTE**: Name the entities and their class members, **exactly** in the **format stated** above. Do not name them in snake case with the dashes, of course. But if a field is specified as bank\_account, you are to name it bankAccount.

#### Relationships

The Council decided to give you a little hint about the more complex relationships in the database, so that you can implement it correctly.

One Employee may have only one Branch, and one Branch may have many Employees.

One Employee may have many Clients, and one Client may be appointed to many Employees.

A Client can have only one Bank Account, and one Bank Account can have only one Client.

One Card can have only one Bank Account, and one Bank Account can have many Cards.

## Data Import

Use the provided **JSON** and **XML** files to populate the database with data. Import all the information from those files into the database.

**You are not allowed to modify the provided JSON and XML files.**

**ANY INCORRECT** data should be **ignored** and a message “Error: Incorrect Data!” should be printed.

* **NOTE**: An incorrect data input is an input which is **missing required fields**.
* There are **no other validation criteria**.

**ANY SUCCESSFUL** data import should **result** in a message “Successfully imported {entityClass} – {entityField}.”.

The entityField depends on the entityClass:

* For Branch – {name}.
* For Employee – a string **composed** in the following format – “{first\_name} {last\_name}”.
* For Client – {full\_name}.
* For BankAccount – {account\_number}.
* For Card – {card\_number}.

### JSON Import

#### branches (branches.json)

|  |
| --- |
| **branches.json** |
| [  { "name" : "Anniversary Branch" },  { "name" : "Becker Branch" },  . . .  ] |

|  |
| --- |
| Successfully imported Branch - Anniversary Branch.  Successfully imported Branch - Becker Branch.  . . . |

#### Employees (employees.json)

|  |
| --- |
| **employees.json** |
| [  {  "full\_name" : "Milty Dyett",  "salary" : 213270.78,  "started\_on" : "2017-06-10",  "branch\_name" : "Mendota Branch",  },  {  "full\_name" : "Ermentrude Crenshaw",  "salary" : 641140.25,  "started\_on" : "2017-11-26",  "branch\_name" : "Grasskamp Branch",  },  . . .  ] |

|  |
| --- |
| Successfully imported Employee - Milty Dyett.  Successfully imported Employee - Ermentrude Crenshaw.  . . . |

#### Clients (clients.json)

|  |
| --- |
| **clients.json** |
| [  {  "first\_name" : "Adorne",  "last\_name" : "Bewly",  "age" : 34,  "appointed\_employee" : "Milty Dyett"  },  . . .  ] |

|  |
| --- |
| Successfully imported Client - Adorne Bewly.  . . . |

### XML Import

The **ccb** have prepared some XML data for you to import too. Don’t worry, it’s not too much.

#### Bank Accounts (bank-accounts.xml)

|  |
| --- |
| **bank-accounts.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <bank-accounts>  <bank-account client="Elyn Grimditch">  <account-number>84999053-X</account-number>  <balance>439216.96</balance>  </bank-account>  ...  </bank-accounts> |

|  |
| --- |
| Successfully imported Bank Account - 84999053-X.  . . . |

#### Cards (cards.xml)

|  |
| --- |
| **cards.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <cards>  <card status="Active" account-number="45313950-8">  <card-number>CR31 2172 7000 5807</card-number>  </card>  <card status="Active" account-number="45313950-8">  <card-number>KZ69 306U DAMP BELG</card-number>  </card>  <card status="Active" account-number="90687224-1">  <card-number>MR58 1652 6071 3846</card-number>  </card>  ...  </cards> |

|  |
| --- |
| Successfully imported Card - CR31 2172 7000 5807.  . . . |

## Data Export

Get ready to export the data you’ve imported in the previous task. Here you will have some pretty complex database querying. Export the data in the formats specified below.

#### Top Employees

**Export all employees** which have **any** **clients** in them:

* Extract from the database, the employees and their clients.
* **Order** them **descending** by count of clients, and **ascending** by employee id.

#### Family Guy

**Export** the **client** with the **most cards** in his **bank account**.

* Export the **client’s** **full** **name**, **age, bank\_account** and his **cards**.
* For **each** of his **cards**, **export** the **card\_number** and the **status**.