# Databases Advanced Retake Exam

# Royal United Kingsman

Royal United Kingsman Bank or most widely known as R.U.K. Bank is a new bank founded by Darkman Nakov. You have been employed by the bank to design a database application infrastructure, which supports basic functionality like importing and exporting JSON and XML data.

## Functionality Overview

R.U.K. has hired you as their database developer (personal request by Darkman), to implement a **database application**. The application should be able to easily **import** hard-formatted data from **XML** and **JSON** and **support functionality** for also **exporting** into the same formats. The application is called – **ruk**.

## Project Skeleton Overview

You will be given a **Skeleton**, containing a **certain architecture** with **several classes**, some of which – completely empty. The **Skeleton** will include the **files** with which you will **seed** the **database**. You will also be given a bunch of **Unit Tests** which will assure that you **don’t write something broken**, because we are talking about a Bank Project.

## Model Definition (50 pts)

There are 5 main models that the **R.U.K.** 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

Darkman Nakov 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 (25 pts)

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 “Succesfully 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 R.U.K. have prepared some XML data for you to import too. Don’t worry, its 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 (25 pts)

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, In the corresponding folders and files.

### JSON Export

#### Top Employees (topEmployees.json)

**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.
* **Export** the **data** in **JSON** format.

|  |
| --- |
| topEmployees**.json** |
| [  {  "full\_name" : "Trula Glasscott",  "salary" : 506242.15,  "started\_on" : "2017-08-23",  "clients" : [  "Baxy David",  "Rahel Tinham",  "Ody Tottman",  . . .  ]  },  {  "full\_name" : "Nial Lonergan",  "salary" : 647924.73,  "started\_on" : "2017-12-21",  "clients" : [  . . .  ]  },  . . .  ] |

### XML Export

#### Family Guy (family-guy.xml)

**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**.
* **Export** the **data** in **XML** format.

|  |
| --- |
| **family-guy.xml** |
| <?xml version="1.0" encoding="utf-8"?>  <family-guy full-name="Baxy David" age="41">  <bank-account account-number="73279927-2">  <cards>  <card>  <card-number>PS69 KNTI I5Q8 4DPA</card-number>  <card-status>Active</card-status>  </card>  . . .  </cards>  </bank-account>  </family-guy> |