Assignment 2

Data set analysis

Based on the given data set, 5 collections with its own purpose were created as follows:

- Collection **user**: contains personal information of each user such as age, gender, native country, and race.
- Collection **education**: contains information about user's education such as education level and number of years of education.
- Collection **occupation:** contains information of user's work class, occupation, and hours of working per week.
- Collection **relationship:** contains information of user's marital status and relationship with owner.
- Collection **finance**: contains information of user's financial situation such as total income, income level, capital gain and capital loss.

ERD design

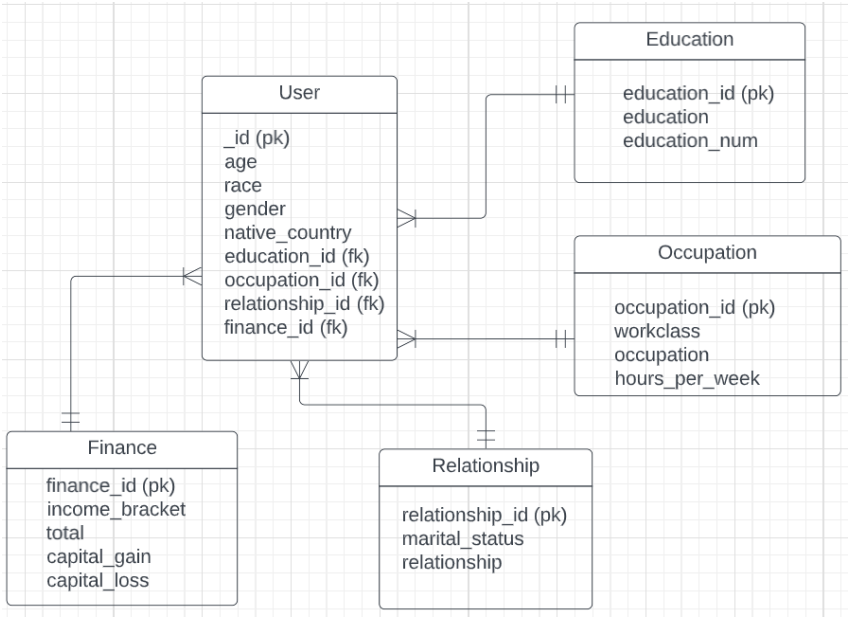


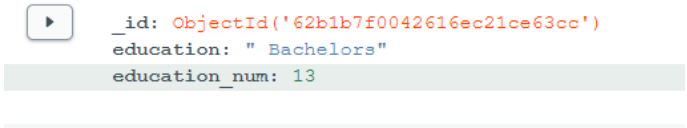
FIGURE 1. Database ERD design

Relationship among between collections:

- Users-Relationship: 1-M, meaning that many users have one same kind of relationship. One user has only 1 kind of relationship.
- Users-Finance: 1-M, meaning that many users have one same kind of financial situation. One user has only 1 financial state.

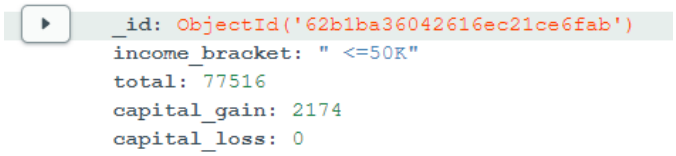
- Users-Occupation: 1-M, meaning that many users have one same kind of relationship. One user has only one kind of occupation.
- Users-Education: 1-M, meaning that many users have one same kind of education. One user has only 1 kind of education.

Education collection: 16 documents



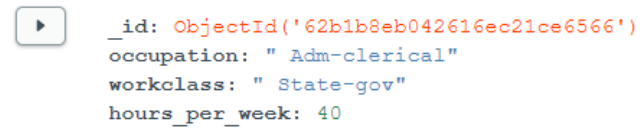
- _id: unique classifier
- education: type of highest education
- education_num: years of education

Finance collection: 25757 documents



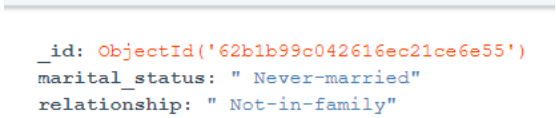
- _id: unique classifier
- income_bracket: either > 50k or <= 50k
- total: total amount in balance
- capital_gain: plus changes in account
- capital_loss: minus changes in account

Occupation collection: 1858 documents



- _id: unique classifier
- occupation: occupation name
- workclass: work class name
- hours_per_weeks: number of working hours per week

Relationship collection: 29 documents



- _id: unique classifier
- marital_status: marital status
- relationship: relationship with owner

User collection: 32429 documents


```
_id: ObjectId('62b1bdac042616ec21d0e3fd')
age: 39
gender: " Male"
native_country: " United-States"
race: " White"
education_id: ObjectId('62b1b7f0042616ec21ce63cc')
finance_id: ObjectId('62b1ba36042616ec21ce6fab')
relationship_id: ObjectId('62b1b99c042616ec21ce6e55')
occupation_id: ObjectId('62b1b8eb042616ec21ce6566')
```


- _id: unique classifier
- age: age
- gender: gender
- native_country: native country
- race: race
- education_id: references _id in education collection
- finance_id: references _id in finance collection
- relationship_id: references _id in relationship collection
- occupation_id: references _id in occupation collection

Create physical database

1. Import file in mongoDB compass into database named “DEPx302_Asm2” under collection named “whole”. The idea is load data into the database without separating it into different collections up to this point.

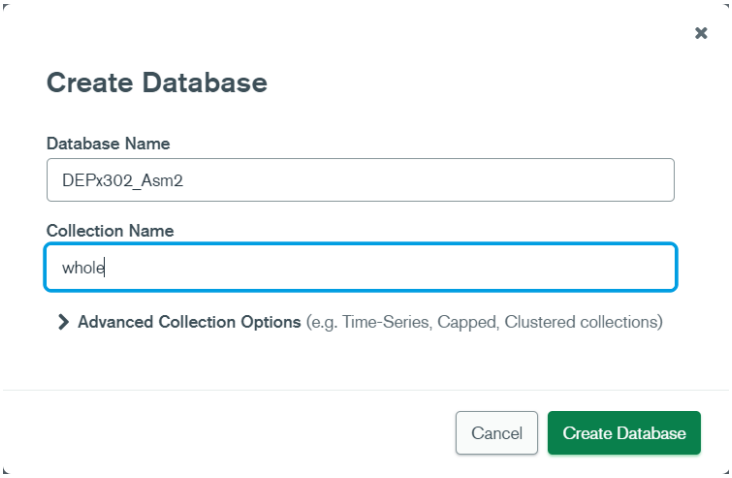
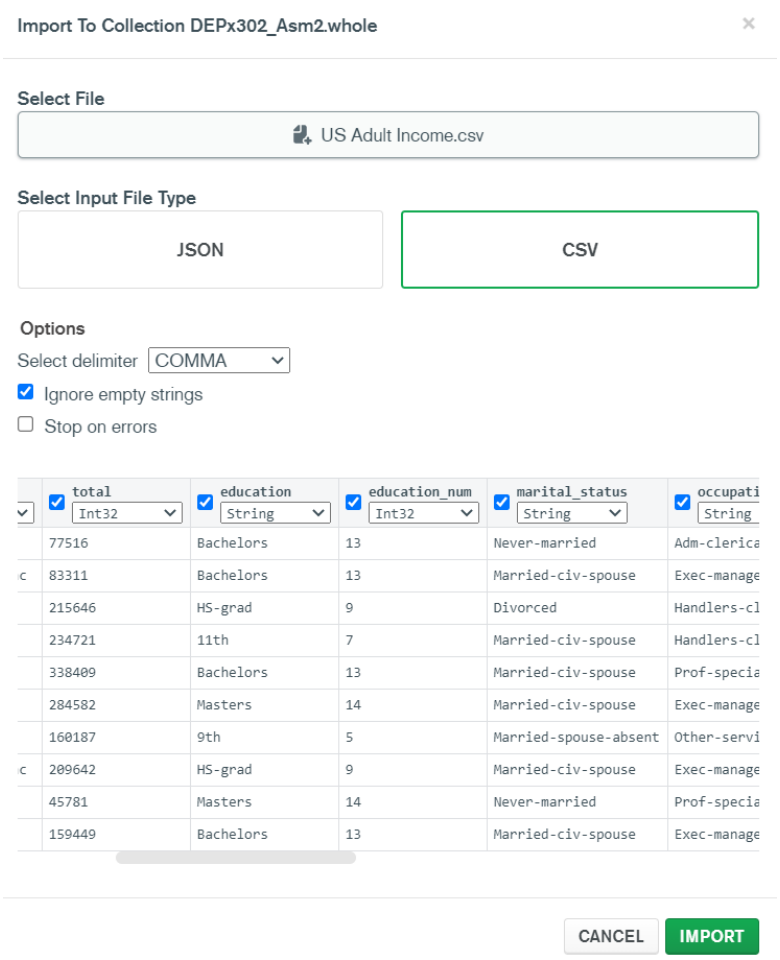


FIGURE 2. Create database and collection

2. While loading data, some variables’ data types have been changed into int32 including age, total, capital_gain, capital_loss, hours_per_week, and education-num.



FIGUR3. Import data into collection “whole”

There are 32,561 documents imported successfully to collection “whole”.

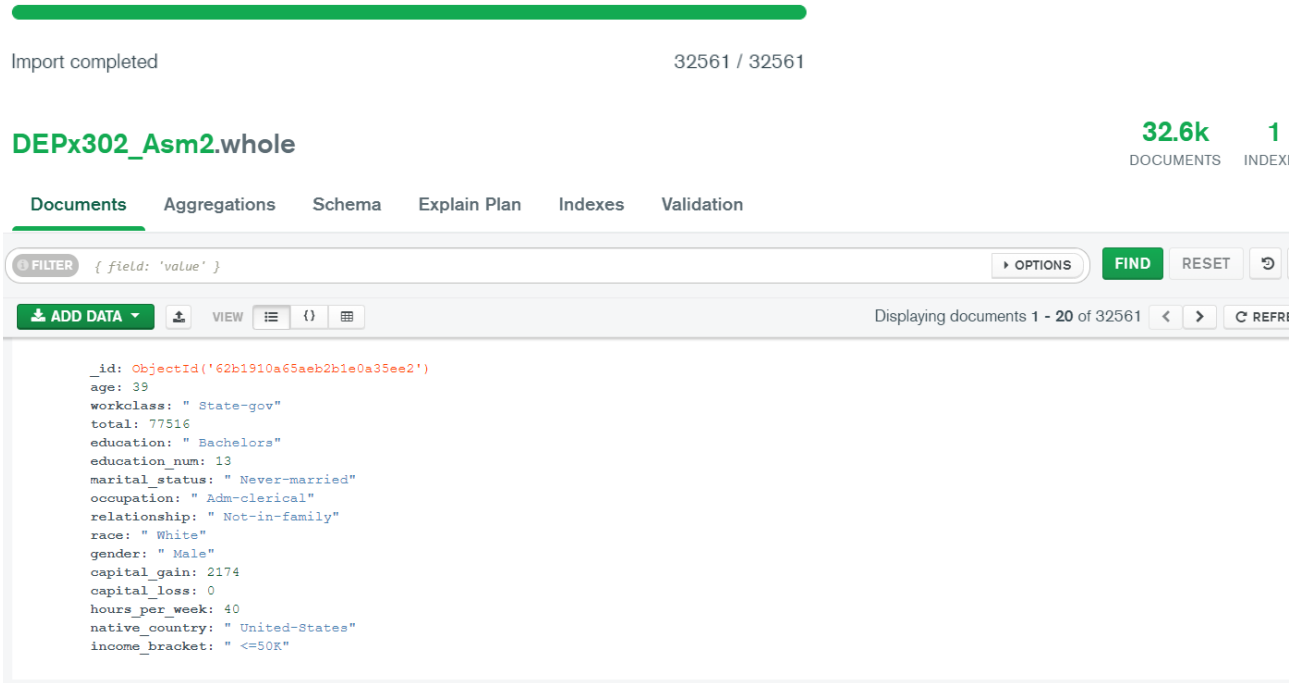


FIGURE 4. A document inside collection “whole”

3. Sequentially import data into these 5 predefined collections. Source code can be found in file "Import data to collections.js" that is also included under the same zip folder.

```
// load data into collection "relationship"
function loadRelationship() {
    const bulkInsert = db.relationship.initializeUnorderedBulkOp();
    // Get all Documents in 'full' Collection
    const documents = db.whole.find({});

    // Process each document
    documents.forEach(function (doc) {
        const element = {
            marital_status: doc.marital_status,
            relationship: doc.relationship
        };
        // Upsert into education Document
        bulkInsert.find(element).upsert().replaceOne(element);
    });
    bulkInsert.execute();
    return true;
}
```

FIGURE 5. Example of source code to load data into collection named relationship

Business questions:

1. How many people that are female and work more than 30 hours per week?

```
{ femaleAndMoreThan30Hours: 7642 }
```

2. How many people in America that had salary that is more than >50k annually?

```
AmericaAnd>50k: 7152
```

3. Total sum of money in balance of people in America

```
_id: " United-States"
totalIncome: 5434891017
```

4. Total sum of working hours of people that had annual salary that is below or equal to 50k.

```
_id: null
totalWorkinghours: 913329
```

5. Find number of people that had more than 100k in account and work less than 55 hours per week.

```
workLessButMuchMoney: 25284
```

Query answers for these business questions

In file "queries.txt" file.

Create index for collections

1. Create index in collection “finance” collection:
   Type: compound index
   Syntax: db.finance.createIndex({income_bracket:1, total: 1})
   Reason: Faster to retrieve data, meaning that the mongodb did not have to scan all of the documents, and only those matching documents had to be pulled into memory. This results in a very efficient query.

```
db.finance.createIndex({income_bracket:1, total:1})
'income_bracket_1_total_1'
```

```
> db.finance.getIndexes()
< [
    { v: 2, key: { _id: 1 }, name: '_id_' },
    {
      v: 2,
      key: { income_bracket: 1, total: 1 },
      name: 'income_bracket_1_total_1'
    }
  ]
```

Query performance:

Without index: have to scan through 25,757 documents to get 15,540 results

```
db.finance.find({income_bracket: {$eq: " <=50K"}, total: {$gt: 100000}}).explain("executionStats")
{ explainVersion: '1',
  queryPlanner:
   { namespace: 'DEPx302_Asm2.finance',
     indexFilterSet: false,
     parsedQuery:
      { '$and':
         [ { income_bracket: { '$eq': ' <=50K' } },
           { total: { '$gt': 100000 } } ] },
     maxIndexedOrSolutionsReached: false,
     maxIndexedAndSolutionsReached: false,
     maxScansToExplodeReached: false,
     winningPlan:
      { stage: 'COLLSCAN',
        filter:
         { '$and':
            [ { income_bracket: { '$eq': ' <=50K' } },
              { total: { '$gt': 100000 } } ] },
        direction: 'forward' },
     rejectedPlans: [] },
  executionStats:
   { executionSuccess: true,
     nReturned: 15540,
     executionTimeMillis: 20,
     totalKeysExamined: 0,
     totalDocsExamined: 25757,
```

After including index: Only have to go through exactly 15,540 documents to get exactly 15,540 result.

```
> db.finance.find({income_bracket: {$eq: " <=50K"}, total: {$gt: 100000}}).explain("executionStats")
< { explainVersion: '1',
    queryPlanner:
     { namespace: 'DEPx302_Asm2.finance',
       indexFilterSet: false,
       parsedQuery:
        { '$and':
           [ { income_bracket: { '$eq': ' <=50K' } },
             { total: { '$gt': 100000 } } ] },
       maxIndexedOrSolutionsReached: false,
       maxIndexedAndSolutionsReached: false,
       maxScansToExplodeReached: false,
       winningPlan:
        { stage: 'FETCH',
          inputStage:
           { stage: 'IXSCAN',
             keyPattern: { income_bracket: 1, total: 1 },
             indexName: 'income_bracket_1_total_1',
             isMultiKey: false,
             multiKeyPaths: { income_bracket: [], total: [] },
             isUnique: false,
             isSparse: false,
             isPartial: false,
             indexVersion: 2,
             direction: 'forward',
             indexBounds:
              { income_bracket: [ '[" <=50K", " <=50K"]' ],
                total: [ '(100000, inf.0]' ] } } },
       rejectedPlans: [] },
    executionStats:
     { executionSuccess: true,
       nReturned: 15540,
       executionTimeMillis: 39,
       totalKeysExamined: 15540,
       totalDocsExamined: 15540,
```

2. The same applies to other collections, where shows potential for creating index other than _id.

- "finance" collection: single field index for only income_bracket field, so that later if we want to retrieve data only for this field in this collection or in join with other connections to get result related to this field.

```
> db.finance.createIndex({income_bracket:1})
< 'income_bracket_1'
```

- "occupation" collection: single field index for only "work_class" field, since it has only 6 values.

```
> db.occupation.createIndex({workclass: 1})
< 'workclass_1'
```

- "relationship" collection: compound index for all fields in this collection.

```
> db.relationship.createIndex({_id:1, marital_status: 1, relationship:1})
< '_id_1_marital_status_1_relationship_1'
```

- "education" collection: single field index for only "education" field, since it has only small amount of values and it would faster to retrieve data when using group() aggregation or simply find().

```
> db.education.createIndex({education:1})
< 'education_1'
```

- "user" collection: compound index for field "gender" and "native_country" since these 2 field are very common personal information.

```
> db.user.createIndex({gender:1, native_country: 1})
< 'gender_1_native_country_1'
```