

# COMS W4111-002/V02, Spring 22: Take Home Final

---

## Overview

## Instructions

- There is an [Ed discussion thread \(https://edstem.org/us/courses/18760/discussion/1464598\)](https://edstem.org/us/courses/18760/discussion/1464598) that will specify completion and submission instructions.

## Due Date, Completing the Exam and Rules

- The final exam is due at 11:59 PM on Monday, 08-May-2022. **You are not allowed to use late days.**
- You may use on-line information and sources to answer questions. But,
  - You cannot simply cut and paste answers or code. Your answer must demonstrate that you understood the material and are capable of producing an answer from your understanding.
  - You must cite any sources of information that you used. This can simply be a comment in a text/markdown cell in your answer. For example, (Note: I used [https://www.w3schools.com/sql/sql\\_check.asp](https://www.w3schools.com/sql/sql_check.asp) to help me with the syntax for adding a check constraint).
  - You do NOT need to cite lecture notes, recordings, slides, ... You do not need to cite information from the recommended textbook or textbook slides.
- You **MUST NOT** collaborate with ANYONE, including other students. You MAY speak with the professor or a TA to discuss the exam.
- If you have questions, post them as PRIVATE question on Ed discussion and use the Category Exams -> Final.
- There is a pinned Ed discussion thread [Midterm Clarifications \(https://edstem.org/us/courses/18760/discussion/1464595\)](https://edstem.org/us/courses/18760/discussion/1464595) that the professor and TA will use to communicate updates and clarifications. **Students are responsible for checking this post.**
- Individual questions provide explanations for how to complete the question.

# Environment Setup and Test

1. This section tests your environment.
2. You will need to change the URLs and password in some of the cells below to match your configuration.
3. You may need to load data and copy databases. The relevant questions provide information.
4. **You must read the comments in the setup cells. You will need objects and functions in the cells for some questions.**

## MySQL

In [1]:

```
%load_ext sql
```

In [2]:

```
# Connect to MySQL. Replace dbuser:dbuserdbuser with your MySQL user ID and password.  
%sql mysql+pymysql://root:dbuserdbuser@localhost
```

Out[2]:

```
'Connected: root@None'
```

In [3]:

```
from sqlalchemy import create_engine
```

In [4]:

```
# Create an SQL Alchemy engine for using MySQL. Replace dbuser:dbuserdbuser with your user  
sql_engine = create_engine("mysql+pymysql://root:dbuserdbuser@localhost")
```

In [5]:

```
import pandas as pd
```

In [6]:

```
import pymysql
```

In [7]:

```
def get_mysql_connection(user_id, password, autocommit=True):
    """
    Creates and return a connection to the local MySQL database.

    :param user_id: The user ID for the connection.
    :param password: Corresponding password.
    :param autocommit: Should this connection use autocommit for executed statements. See t
        for explanation:
        - https://pymysql.readthedocs.io/en/latest/modules/connections.html
        - https://dev.mysql.com/doc/refman/5.6/en/innodb-autocommit-commit-rollback.html
    """
    sql_conn = pymysql.connect(
        user=user_id,
        password=password,
        host="localhost",
        port=3306,
        cursorclass=pymysql.cursors.DictCursor,
        autocommit=autocommit)

    return sql_conn

def close_connection(sql_connection):
    """
    Closes a connection.

    :param sql_connection: The connection to close.
    """
    if sql_connection:
        sql_connection.close()
```

- The following cell(s) load information about CU courses.

In [8]:

```
# You must set these parameters to the values for you final submission and databases.
# Replace dff9 with your UNI
#
mysql_url = "mysql+pymysql://root:dbuserdbuser@localhost"
final_schema = "sm4940_s22_final"
course_info_table = "course_info"
course_info_file = "./course_info.json"

def read_and_save_json_file(file_name, schema_name, table_name):

    df = pd.read_json(file_name)
    eng = create_engine(mysql_url)
    df.to_sql(table_name, schema=schema_name, con=eng, index=False, if_exists="replace")
```

In [9]:

```
%sql create database if not exists sm4940_s22_final
```

```
* mysql+pymysql://root:***@localhost  
1 rows affected.
```

Out[9]:

```
[]
```

In [10]:

```
read_and_save_json_file(course_info_file, final_schema, course_info_table)
```

- The following cell tests your load. You must replace the schema name with your schema.

In [11]:

```
%sql use $final_schema
```

```
* mysql+pymysql://root:***@localhost  
0 rows affected.
```

Out[11]:

```
[]
```

In [12]:

```
%%sql
select
    callnumber, schoolcode, coursetitle, coursesubtitle,
    instructor1name, typename,
    course, prefixname,
    enrollmentstatus,
    maxsize, numenrolled,
    callnumber,
    prefixlongname,
    departmentcode, campuscode,
    divisionname, departmentname,
    typecode, term,
    schoolname, meets1
from course_info
where instructor1name like "%Ferguson%Donald%"
order by callnumber;
```

```
* mysql+pymysql://root:***@localhost
7 rows affected.
```

Out[12]:

callnumber	schoolcode	coursetitle	coursesubtitle	instructor1name	typename	
11009	SEAS	INTRODUCTION TO DATABASES	INTRODUCTION TO DATABASES	FERGUSON, DONALD F	LECTURE	COMS41
11038	SEAS	TOPICS IN SOFTWARE ENGINEERING	CLOUD COMPUTING	FERGUSON, DONALD F	LECTURE	COMS61
12423	SEAS	INTRODUCTION TO DATABASES	INTRODUCTION TO DATABASES	FERGUSON, DONALD F	LECTURE	COMS41
13390	SEAS	TOPICS IN SOFTWARE ENGINEERING	CLOUD COMPUTING	FERGUSON, DONALD F	LECTURE	COMS61
13911	SEAS	INTRODUCTION TO DATABASES	INTRODUCTION TO DATABASES	FERGUSON, DONALD F	LECTURE	COMS41
16112	SEAS	TOPICS IN SOFTWARE ENGINEERING	CLOUD COMPUTING	FERGUSON, DONALD F	LECTURE	COMS61
18408	SEAS	INTRODUCTION TO DATABASES	INTRODUCTION TO DATABASES	FERGUSON, DONALD F	LECTURE	COMS41

## MongoDB

In [51]:

```
mongodb_url = "mongodb+srv://Siddartha_db:y1BwtF5LT5KMa51@cluster0.2jot5.mongodb.net/myFir
```

In [52]:

```
from pymongo import MongoClient
```

In [53]:

```
mongo_client = MongoClient(mongodb_url)
```

In [54]:

```
# Note that your list of database will be different.
# You should see the databases from your MongoDB Atlas account.
#
dbs = mongo_client.list_databases()
list(dbs)
```

Out[54]:

```
[{'name': 'GoT', 'sizeOnDisk': 561152, 'empty': False},
 {'name': 'sample_airbnb', 'sizeOnDisk': 56639488, 'empty': False},
 {'name': 'sample_analytics', 'sizeOnDisk': 9613312, 'empty': False},
 {'name': 'sample_geospatial', 'sizeOnDisk': 1458176, 'empty': False},
 {'name': 'sample_guides', 'sizeOnDisk': 40960, 'empty': False},
 {'name': 'sample_mflix', 'sizeOnDisk': 49610752, 'empty': False},
 {'name': 'sample_restaurants', 'sizeOnDisk': 6967296, 'empty': False},
 {'name': 'sample_supplies', 'sizeOnDisk': 1200128, 'empty': False},
 {'name': 'sample_training', 'sizeOnDisk': 57790464, 'empty': False},
 {'name': 'sample_weatherdata', 'sizeOnDisk': 2940928, 'empty': False},
 {'name': 'admin', 'sizeOnDisk': 376832, 'empty': False},
 {'name': 'local', 'sizeOnDisk': 12952899584, 'empty': False}]
```

## Neo4j

In [31]:

```
#
# You must replace the following with your AuraDB Neo4j information.
neo_url = "neo4j+s://747cc46b.databases.neo4j.io:7687"
neo_auth = ("neo4j", "0MiCjx0Bi7QyiMA6f172TDlu3l0_KZbq_5pqDBcv_ng")
```

In [32]:

```
from py2neo import data, Graph, NodeMatcher, Node, Relationship, RelationshipMatcher
```

In [33]:

```
neo_graph = Graph(neo_url, auth=neo_auth)
```

In [34]:

```
#
# The following code assumes that you have loaded the sample movie database in Neo4j
tom = NodeMatcher(neo_graph).match(name="Tom Hanks")
for t in tom:
    print(dict(t))
```

```
{'born': 1956, 'name': 'Tom Hanks'}
```

## Written Questions

### W1: Serializability

- Briefly define *conflict serializable*. Relative to *conflict serializable*, define *\_conflicting operations*.

Answer: A schedule can be called conflict serializable if it can be transformed into a serial schedule by performing non conflicting operations.

An operation is said to be conflicting if all the three conditions are satisfied:

- both are operating on same data item
- They belong to different transactions
- One of them is a write operation

- Is the following schedule *conflict serializable*? Why?

T1	T2
R(A)	
R(B)	
	R(A)
	R(B)
	W(B)
W(A)	

Answer:

No, This schedule is not conflict serializable as we can't have T1 operations done before T2 as W(A) from T1 cant be moved up and we cant have T2 operations done before T1 as W(B) of T2 cant be moved above T1.

### W2: Locking

- Briefly define *Two-Phase Locking* and *Strict Two-Phase Locking*. What condition does *Strict Two-Phase Locking* prevent?

Answer: Two phase locking ensures conflict serializable schedules, where in growing phase, A transaction can

obtain locks but not release them and in the shrinking phase, a transaction can release locks but cannot obtain them.

Strict two-phase locking prevents cascading roll-back as it mandates a transaction to hold all its exclusive locks till it commits/aborts

Type *Markdown* and LaTeX:  $\alpha^2$

## W3: Metadata

- Define *metadata*. Give an example of metadata for a SQL database. What is the name of the database/schema where MySQL stores its metadata?

Answer:

Metadata is the data about data. In the SQL database the data like constraints, data types, foreign keys, columns is called meta data. MySQL stores the metadata in the database mysql

## W4: Locking

- Briefly define *Two-Phase Locking* and *Strict Two-Phase Locking*. What condition does *Strict Two-Phase Locking* prevent?

Answer: Two phase locking ensures conflict serializable schedules, where in growing phase, A transaction can obtain locks but not release them and in the shrinking phase, a transaction can release locks but cannot obtain them.

Strict two-phase locking prevents cascading roll-back as it mandates a transaction to hold all its exclusive locks till it commits/aborts

## W5: Functional Dependency

- Briefly define *functional dependency*. What changes to a schema do you make to eliminate functional dependencies.

Answer:

Functional dependency is when an attribute can uniquely identify another attribute, generally it is the relation between PK and non-key attributes.

Type *Markdown* and LaTeX:  $\alpha^2$

## W6: Entity Types

- Briefly define *associative entity* and *weak entity*.



Answer: An associative entity establishes association between two entities to resolve many to many relationships. A weak entity is the one that cannot be uniquely identified with its own attributes and needs to use some foreign key in conjunction to its attributes to become identifiable.

Type *Markdown* and LaTeX:  $\alpha^2$

## W7: RAID

- Briefly define and explain the differences between *RAID 0*, *RAID 1* and *RAID 5*.

Answer:

RAID is a data storage visualization technology which stands for redundant array of independent disk that combines multiple physical disk drive components into single logical unit for the purpose of reliability, data redundancy and performance improvement. RAID 0 uses disk striping, without mirroring or parity. RAID 1 uses data mirroring without parity or striping and RAID 5 uses block level striping with distributed parity.

## W8: Column versus Row Storage

- Briefly explain *row-oriented storage* versus *column-oriented storage*. Explain a scenario/database query pattern where column-oriented storage is beneficial.

Answer: Row-oriented storage is the storage where all the associated fields of the record are stored together whereas in column-oriented storage all the values in a field are stored together. Generally, any query that requires aggregation/decision will benefit from column-oriented storage. For ex: if we are asked to find the min/max of an attribute, column-oriented storage will have much faster response.

Type *Markdown* and LaTeX:  $\alpha^2$

## W9: Indexes

- Briefly define:
  - Sparse index
  - Dense index
  - Clustered index

Answer:

**Sparse index:** Sparse index stores only partial amount of key values from the table for indexing. So, a set of values are associated to an index. It needs less space compared to dense index.

**Dense index:** When an index is created for every search key value in the table then that index is called dense index, it is very fast but would require large memory to store large indexing table.

**Clustered index:** clustered indexes are indexes whose order of rows in the data pages correspond to the order of rows in the index

ref: <https://www.ibm.com/docs/en/ias?topic=indexes-clustered-non-clustered> (<https://www.ibm.com/docs/en/ias?topic=indexes-clustered-non-clustered>)

Type *Markdown* and LaTeX:  $\alpha^2$

## W10: Query Processing

- Briefly define:
  - Access path
  - Join order optimization
  - Equivalent queries/expressions.

Answer:

Access path: It is the path to be followed to access a variable/file/directory etc

Join order optimization: It is the process of identifying the best possible sequence of joining tables to maximize the performance

Equivalent queries or expressions: These are calculated every time before query execution and are very important as a complex query might be equivalent to a simple query which can have high performance.

Type *Markdown* and LaTeX:  $\alpha^2$

## Data Modeling Question

- Consider the [IMDB datasets](https://www.imdb.com/interfaces/). (<https://www.imdb.com/interfaces/>)
- The following queries run on my laptop show the structure of some of the data. These queries will not run on your computer. So, if you execute the cell, you will lose the out.

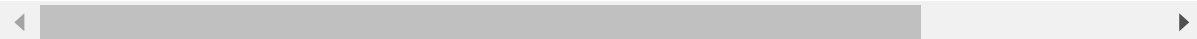
In [13]:

```
%sql select * from imdb_raw_fast.name_basics limit 10;
```

```
* mysql+pymysql://dbuser:***@localhost
10 rows affected.
```

Out[13]:

nconst	primaryName	birth_year	death_year	primaryProfession	
nm0000001	Fred Astaire	1899	1987	soundtrack,actor,miscellaneous	tt0053137,tt
nm0000002	Lauren Bacall	1924	2014	actress,soundtrack	tt0038355,tt
nm0000003	Brigitte Bardot	1934	None	actress,soundtrack,music_department	tt0049189,tt
nm0000004	John Belushi	1949	1982	actor,soundtrack,writer	tt0072562,tt
nm0000005	Ingmar Bergman	1918	2007	writer,director,actor	tt0060827,tt
nm0000006	Ingrid Bergman	1915	1982	actress,soundtrack,producer	tt0034583,tt
nm0000007	Humphrey Bogart	1899	1957	actor,soundtrack,producer	tt0033870,tt
nm0000008	Marlon Brando	1924	2004	actor,soundtrack,director	tt0070849,tt
nm0000009	Richard Burton	1925	1984	actor,soundtrack,producer	tt0057877,tt
nm0000010	James Cagney	1899	1986	actor,soundtrack,director	tt0031867,tt



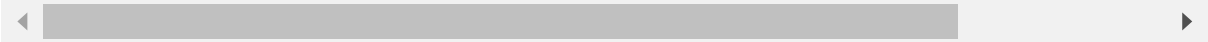
In [14]:

```
%sql select * from imdb_raw_fast.title_basics limit 10;
```

\* mysql+pymysql://dbuser:\*\*\*@localhost  
10 rows affected.

Out[14]:

tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	
tt0000001	short	Carmencita	Carmencita	0	1894	\N	1	
tt0000002	short	Le clown et ses chiens	Le clown et ses chiens	0	1892	\N	5	
tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	4	Anin
tt0000004	short	Un bon bock	Un bon bock	0	1892	\N	12	
tt0000005	short	Blacksmith Scene	Blacksmith Scene	0	1893	\N	1	
tt0000006	short	Chinese Opium Den	Chinese Opium Den	0	1894	\N	1	
tt0000007	short	Corbett and Courtney Before the Kinetograph	Corbett and Courtney Before the Kinetograph	0	1894	\N	1	
tt0000008	short	Edison Kinetoscopic Record of a Sneeze	Edison Kinetoscopic Record of a Sneeze	0	1894	\N	1	
tt0000009	short	Miss Jerry	Miss Jerry	0	1894	\N	40	
tt0000010	short	Leaving the Factory	La sortie de l'usine Lumière à Lyon	0	1895	\N	1	



In [15]:

```
%sql select * from imdb_raw_fast.title_principals limit 10;
```

```
* mysql+pymysql://dbuser:***@localhost
10 rows affected.
```

Out[15]:

tconst	ordering	nconst	category	job	characters
tt0000001	1	nm1588970	self	\N	["Self"]
tt0000001	2	nm0005690	director	\N	\N
tt0000001	3	nm0374658	cinematographer	director of photography	\N
tt0000002	1	nm0721526	director	\N	\N
tt0000002	2	nm1335271	composer	\N	\N
tt0000003	1	nm0721526	director	\N	\N
tt0000003	2	nm1770680	producer	producer	\N
tt0000003	3	nm1335271	composer	\N	\N
tt0000003	4	nm5442200	editor	\N	\N
tt0000004	1	nm0721526	director	\N	\N

- Given this information and the [descriptions of the data \(https://www.imdb.com/interfaces/\)](https://www.imdb.com/interfaces/), produce a Crow's Foot diagram of a data model you would use to hold the data.
- Your design will require transforming the data, which you do not need to do. But, you should do a good data model. That is, define a "to be" data model.

Answer:

In [81]:

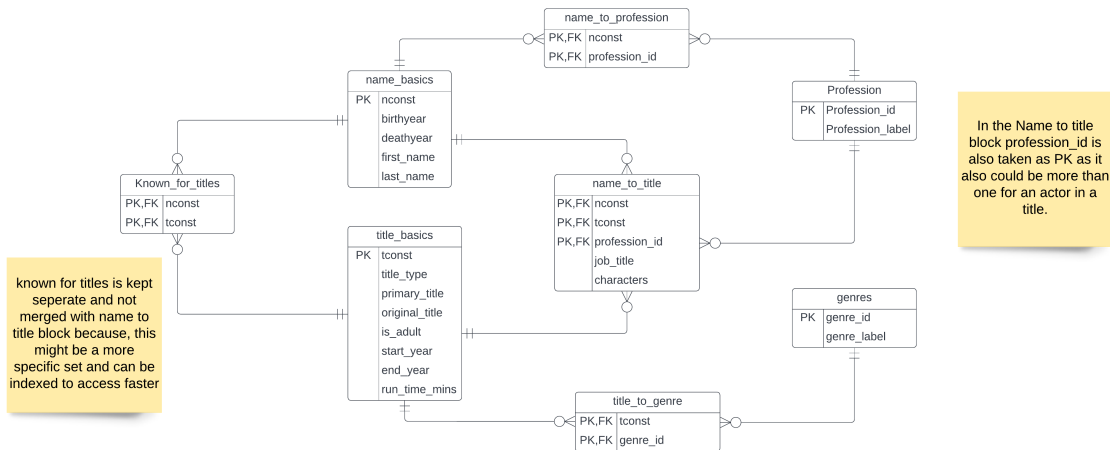
```

file_name = 'IMDB_data_model.png'

print("\n")
from IPython.display import Image
Image(filename=file_name)

```

Out[81]:

Type *Markdown* and LaTeX:  $\alpha^2$ 

## Data Loading and Transformation

- In the setup for MySQL, you tested the setup by loading information from the [Columbia Open Data Service](https://opendataservice.columbia.edu/). (<https://opendataservice.columbia.edu/>)

In [46]:

```
%%sql
select
    *
from course_info
where instructor1name like "%Ferguson%Donald%"
order by callnumber;
```

\* mysql+pymysql://dbuser:\*\*\*@localhost  
7 rows affected.

Out[46]:

Instructor3Name	DivisionCode	ChargeAmt1	SchoolCode	CourseSubtitle	Meets5	Approval	I
	IF		SEAS	INTRODUCTION TO DATABASES			
	EP		SEAS	CLOUD COMPUTING			
	IF		SEAS	INTRODUCTION TO DATABASES			
	EP		SEAS	CLOUD COMPUTING			
	IF		SEAS	INTRODUCTION TO DATABASES			
	EP	395.00	SEAS	CLOUD COMPUTING			
	IF	395.00	SEAS	INTRODUCTION TO DATABASES			

- The columns in the table are:

In [80]:

```
%sql show columns from sm4940_s22_final.course_info;
```

```
* mysql+pymysql://root:***@localhost
44 rows affected.
```

Out[80]:

Field	Type	Null	Key	Default	Extra
Instructor3Name	text	YES		None	
DivisionCode	text	YES		None	
ChargeAmt1	text	YES		None	
SchoolCode	text	YES		None	
CourseSubtitle	text	YES		None	
Meets5	text	YES		None	
Approval	text	YES		None	
Instructor1Name	text	YES		None	
Meets6	text	YES		None	
NumFixedUnits	text	YES		None	
TypeName	text	YES		None	
SubtermCode	text	YES		None	
ClassNotes	text	YES		None	
Meets3	text	YES		None	
Course	text	YES		None	
PrefixName	text	YES		None	
EnrollmentStatus	text	YES		None	
MaxSize	text	YES		None	
NumEnrolled	text	YES		None	
ChargeMsg1	text	YES		None	
CallNumber	bigint	YES		None	
BulletinFlags	text	YES		None	
SubtermName	text	YES		None	
PrefixLongname	text	YES		None	
Meets4	text	YES		None	
DepartmentCode	text	YES		None	
CampusCode	text	YES		None	
DivisionName	text	YES		None	
CourseTitle	text	YES		None	
DepartmentName	text	YES		None	
ExamDate	text	YES		None	
ChargeMsg2	text	YES		None	
CampusName	text	YES		None	



Field	Type	Null	Key	Default	Extra
ChargeAmt2	text	YES		None	
Instructor2Name	text	YES		None	
MaxUnits	bigint	YES		None	
TypeCode	text	YES		None	
Term	bigint	YES		None	
Instructor4Name	text	YES		None	
SchoolName	text	YES		None	
MinUnits	bigint	YES		None	
Meets1	text	YES		None	
Meets2	text	YES		None	
ExamMeet	text	YES		None	

- For the purposes of the final, you only need to consider the columns in the following query.

In [44]:

```
%%sql
select
    Instructor3Name,
    DivisionCode,
    SchoolCode,
    CourseSubtitle,
    Approval,
    Instructor1Name,
    TypeName,
    ClassNotes,
    Course,
    PrefixName,
    EnrollmentStatus,
    MaxSize,
    NumEnrolled,
    CallNumber,
    BulletinFlags,
    PrefixLongname,
    DepartmentCode,
    CampusCode,
    DivisionName,
    CourseTitle,
    DepartmentName,
    CampusName,
    Instructor2Name,
    TypeCode,
    Term,
    Instructor4Name,
    SchoolName
from
    course_info order by callnumber
limit 10;
```

```
* mysql+pymysql://dbuser:***@localhost
10 rows affected.
```

Out[44]:

Instructor3Name	DivisionCode	SchoolCode	CourseSubtitle	Approval	Instructor1Name
	BC	BCOL	BEFORE 'BLACK LIVES MATTE		CHRISTIANSE YVETTE
	BC	BCOL	GLOBAL LONG- FORM PHOTOGRA		MATAR, DIANA
	BC	BCOL	EUROPEAN HISTORY 1500-178	EP	VALENZE DEBORAH
	BC	BCOL	SENIOR SEMINAR		
	BC	BCOL	MIGRATION & ECONOMIC CHAN		TIMMER ASHLEY

BC	BCOL	INTRO AM HIST TO 1865	EP	LIPMAN ANDREW C
BC	BCOL	INTERDISCIPLINARY SENIOR	EP	CLEARY, SKYE
BC	BCOL	BAD LOVE		HAMILTON ROSS T
BC	BCOL	GLOBAL QUEER CINEMAS		ULA, DUYGU
BC	BCOL	PLTCS CRIME& POLICING IN	EP	VAZ, MATTHEW

- Your task is to "clean up" the data. This will involve:
  - Splitting/decomposing into multiple tables.
  - Setting types for columns.
  - Creating constraints and indexes, etc.
- In the following cells, include and execute:
  - The create and alter table statements.
  - The query statements for copying and transforming the data.

Answer:

In [ ]:

```
#
# ALL of your DDL statements must go here. You must group all statements for a table togeth
# with one cell containing statements for a table.
#
```

In [38]:

```
%%sql

# create instructor table
CREATE TABLE Instructor(
    Instructor_id int NOT NULL AUTO_INCREMENT,
    First_name varchar(64),
    Last_name varchar(64),
    PRIMARY KEY(Instructor_id)
);

%%sql

CREATE TABLE Division(
    Division_id int NOT NULL AUTO_INCREMENT,
    DivisionCode char(2),
    DivisionName varchar(64),
    PRIMARY KEY(Division_id)
);

CREATE TABLE Campus(
    Campus_id int NOT NULL AUTO_INCREMENT,
    CampusCode varchar(8),
    CampusName varchar(64),
    PRIMARY KEY(Campus_id)
);

CREATE TABLE Department(
    Department_id int NOT NULL AUTO_INCREMENT,
    DepartmentCode varchar(8),
    DepartmentName varchar(64),
    PRIMARY KEY(Department_id)
);

CREATE TABLE School(
    School_id int NOT NULL AUTO_INCREMENT,
    SchoolCode varchar(8),
    SchoolName varchar(64),
    PRIMARY KEY(School_id)
);

CREATE TABLE courses_20213(
    course_id varchar(12) NOT NULL,
    CourseTitle varchar(64) NOT NULL,
    CourseSubTitle varchar(32),
    Approval varchar(2),
    TypeName varchar(12),
    ClassNotes varchar(64),
    Prefixname varchar(32),
    EnrollmentStatus char(1),
    Maxsize int,
    NumEnrolled int,
    CallNumber int,
    BulletinFlags varchar(12),
    PrefixLongname varchar(32),
    TypeCode char(2),
    Instructor1_id int,
    FOREIGN KEY (Instructor1_id) REFERENCES Instructor(Instructor_id),
    Instructor2_id int,
    FOREIGN KEY (Instructor2_id) REFERENCES Instructor(Instructor_id),
```

```

    Instructor3_id int,
    FOREIGN KEY (Instructor3_id) REFERENCES Instructor(Instructor_id),
    Instructor4_id int,
    FOREIGN KEY (Instructor4_id) REFERENCES Instructor(Instructor_id),
    Division_id int,
    FOREIGN KEY (Division_id) REFERENCES Division(Division_id),
    Department_id int,
    FOREIGN KEY (Department_id) REFERENCES Department(Department_id),
    Campus_id int,
    FOREIGN KEY (Campus_id) REFERENCES Campus(Campus_id),
    School_id int,
    FOREIGN KEY (School_id) REFERENCES School(School_id),
    PRIMARY KEY (course_id)
    CHECK (NumEnrolled >= 0 AND Maxsize >= 0 AND NumEnrolled <= Maxsize)
);

```

```

CREATE TABLE courses_20221(
    course_id varchar(12) NOT NULL,
    CourseTitle varchar(64) NOT NULL,
    CourseSubTitle varchar(32),
    Approval varchar(2),
    TypeName varchar(12),
    ClassNotes varchar(64),
    Prefixname varchar(32),
    EnrollmentStatus char(1),
    Maxsize int,
    NumEnrolled int,
    CallNumber int,
    BulletinFlags varchar(12),
    PrefixLongname varchar(32),
    TypeCode char(2),
    Instructor1_id int,
    FOREIGN KEY (Instructor1_id) REFERENCES Instructor(Instructor_id),
    Instructor2_id int,
    FOREIGN KEY (Instructor2_id) REFERENCES Instructor(Instructor_id),
    Instructor3_id int,
    FOREIGN KEY (Instructor3_id) REFERENCES Instructor(Instructor_id),
    Instructor4_id int,
    FOREIGN KEY (Instructor4_id) REFERENCES Instructor(Instructor_id),
    Division_id int,
    Department_id int,
    Campus_id int,
    School_id int,
    PRIMARY KEY (course_id),
    FOREIGN KEY (Division_id) REFERENCES Division(Division_id),
    FOREIGN KEY (Department_id) REFERENCES Department(Department_id),
    FOREIGN KEY (Campus_id) REFERENCES Campus(Campus_id),
    FOREIGN KEY (School_id) REFERENCES School(School_id)
    CHECK (NumEnrolled >= 0 AND Maxsize >= 0 AND NumEnrolled <= Maxsize)
);

```

```

CREATE TABLE courses_20222(
    course_id varchar(12) NOT NULL,
    CourseTitle varchar(64) NOT NULL,
    CourseSubTitle varchar(32),
    Approval varchar(2),
    TypeName varchar(12),
    ClassNotes varchar(64),
    Prefixname varchar(32),
    EnrollmentStatus char(1),
    Maxsize int,

```

```

    NumEnrolled int,
    CallNumber int,
    BulletinFlags varchar(12),
    PrefixLongname varchar(32),
    TypeCode char(2),
    Instructor1_id int,
    FOREIGN KEY (Instructor1_id) REFERENCES Instructor(Instructor_id),
    Instructor2_id int,
    FOREIGN KEY (Instructor2_id) REFERENCES Instructor(Instructor_id),
    Instructor3_id int,
    FOREIGN KEY (Instructor3_id) REFERENCES Instructor(Instructor_id),
    Instructor4_id int,
    FOREIGN KEY (Instructor4_id) REFERENCES Instructor(Instructor_id),
    Division_id int,
    Department_id int,
    Campus_id int,
    School_id int,
    PRIMARY KEY (course_id),
    FOREIGN KEY (Division_id) REFERENCES Division(Division_id),
    FOREIGN KEY (Department_id) REFERENCES Department(Department_id),
    FOREIGN KEY (Campus_id) REFERENCES Campus(Campus_id),
    FOREIGN KEY (School_id) REFERENCES School(School_id)
    CHECK (NumEnrolled >= 0 AND Maxsize >= 0 AND NumEnrolled <= Maxsize)
);

CREATE TABLE courses_20223(
    course_id varchar(12) NOT NULL,
    CourseTitle varchar(64) NOT NULL,
    CourseSubTitle varchar(32),
    Approval varchar(2),
    TypeName varchar(12),
    ClassNotes varchar(64),
    Prefixname varchar(32),
    EnrollmentStatus char(1),
    Maxsize int,
    NumEnrolled int,
    CallNumber int,
    BulletinFlags varchar(12),
    PrefixLongname varchar(32),
    TypeCode char(2),
    Instructor1_id int,
    FOREIGN KEY (Instructor1_id) REFERENCES Instructor(Instructor_id),
    Instructor2_id int,
    FOREIGN KEY (Instructor2_id) REFERENCES Instructor(Instructor_id),
    Instructor3_id int,
    FOREIGN KEY (Instructor3_id) REFERENCES Instructor(Instructor_id),
    Instructor4_id int,
    FOREIGN KEY (Instructor4_id) REFERENCES Instructor(Instructor_id),
    Division_id int,
    Department_id int,
    Campus_id int,
    School_id int,
    PRIMARY KEY (course_id),
    FOREIGN KEY (Division_id) REFERENCES Division(Division_id),
    FOREIGN KEY (Department_id) REFERENCES Department(Department_id),
    FOREIGN KEY (Campus_id) REFERENCES Campus(Campus_id),
    FOREIGN KEY (School_id) REFERENCES School(School_id)
    CHECK (NumEnrolled >= 0 AND Maxsize >= 0 AND NumEnrolled <= Maxsize)
)

```

```
* mysql+pymysql://root:***@localhost
0 rows affected.
```

Out[38]:

[]

In [47]:

```
#
#
# Put your cleanup and load statements in the following cells.
#
```

In [72]:

```
%%sql

INSERT INTO Instructor (First_name,Last_name)
select substring_index(Instructor1Name,',',-1),
substring_index(Instructor1Name,',',1)
from
(select distinct(Instructor1Name) from course_info
UNION
select distinct(Instructor2Name) from course_info
UNION
select distinct(Instructor3Name) from course_info
UNION
select distinct(Instructor4Name) from course_info) as a
```

```
* mysql+pymysql://root:***@localhost
6324 rows affected.
```

Out[72]:

[]

In [73]:

```
%%sql
```

```
select * from Instructor limit 10
```

```
* mysql+pymysql://root:***@localhost
10 rows affected.
```

Out[73]:

Instructor_id	First_name	Last_name
1	JESSICA	SOURSOURIAN
2	AMIR	ZIV
3	.	FACULTY
4	MATTHIAS	BREUER
5	WEI	CAI
6	RADHA	RADHAKRISHNA
7	ROBERT	STOUMBOS
8	SHIVARAM	RAJGOPAL
9	THOMAS E	BOURVEAU
10	ANDREAS	OHL

In [74]:

```
%%sql
```

```
INSERT INTO Division(DivisionName,DivisionCode)
select distinct(DivisionName) as DivisionName, DivisionCode from course_info
where DivisionCode in (select distinct(DivisionCode) from course_info)
```

```
* mysql+pymysql://root:***@localhost
35 rows affected.
```

Out[74]:

```
[]
```

In [75]:

```
%%sql
```

```
INSERT INTO Department(DepartmentName,DepartmentCode)
select distinct(DepartmentName) as DepartmentName, DepartmentCode from course_info
where DepartmentCode in (select distinct(DepartmentCode) from course_info)
```

```
* mysql+pymysql://root:***@localhost
192 rows affected.
```

Out[75]:

```
[]
```



In [76]:

```
%%sql

INSERT INTO Campus(CampusName,CampusCode)
select distinct(CampusName) as CampusName, CampusCode from course_info
where CampusCode in (select distinct(CampusCode) from course_info)
```

```
* mysql+pymysql://root:***@localhost
22 rows affected.
```

Out[76]:

```
[]
```

In [77]:

```
%%sql

INSERT INTO School(SchoolName,SchoolCode)
select distinct(SchoolName) as SchoolName, SchoolCode from course_info
where SchoolCode in (select distinct(SchoolCode) from course_info)
```

```
* mysql+pymysql://root:***@localhost
20 rows affected.
```

Out[77]:

```
[]
```

In [17]:

```
%%sql

UPDATE course_info
SET Maxsize = 1
WHERE Maxsize = ''
```

```
* mysql+pymysql://root:***@localhost
67 rows affected.
```

Out[17]:

```
[]
```

In [15]:

```
%%sql

UPDATE course_info
SET NumEnrolled = 0
where NumEnrolled <=0 or NumEnrolled = ''
```

```
* mysql+pymysql://root:***@localhost
5039 rows affected.
```

Out[15]:

```
[]
```

In [13]:

```
%%sql
```

```
UPDATE course_info  
SET Maxsize = NumEnrolled  
WHERE Maxsize < NumEnrolled
```

```
* mysql+pymysql://root:***@localhost  
4630 rows affected.
```

Out[13]:

```
[]
```

In [49]:

```
%%sql
```

```
INSERT INTO courses_20213(course_id,CourseTitle,CourseSubTitle,Approval,TypeName,ClassNotes  
                          Maxsize, NumEnrolled, CallNumber, BulletinFlags, PrefixLongname, T  
                          Instructor2_id,Instructor3_id,Instructor4_id,Division_id,Campus_id  
SELECT Course,CourseTitle,CourseSubTitle,Approval,TypeName,ClassNotes,Prefixname,Enrollment  
       Maxsize, NumEnrolled, CallNumber, BulletinFlags, PrefixLongname, T  
       Instructor_id_1, Instructor_id_2,Instructor_id_3,Instructor_id_4,Di  
       School_id,Department_id  
  
from course_info  
where Term = 20213
```

```
* mysql+pymysql://root:***@localhost  
8898 rows affected.
```

Out[49]:

```
[]
```

In [50]:

```
%%sql

INSERT INTO courses_20223(course_id,CourseTitle,CourseSubTitle,Approval,TypeName,ClassNotes
                        Maxsize, NumEnrolled, CallNumber, BulletinFlags, PrefixLongname, T
                        Instructor2_id,Instructor3_id,Instructor4_id,Division_id,Campus_id
SELECT Course,CourseTitle,CourseSubTitle,Approval,TypeName,ClassNotes,Prefixname,Enrollment
                        Maxsize, NumEnrolled, CallNumber, BulletinFlags, PrefixLongname, T
                        Instructor_id_1, Instructor_id_2,Instructor_id_3,Instructor_id_4,Di
                        School_id,Department_id

    from course_info
    where Term = 20223;

INSERT INTO courses_20222(course_id,CourseTitle,CourseSubTitle,Approval,TypeName,ClassNotes
                        Maxsize, NumEnrolled, CallNumber, BulletinFlags, PrefixLongname, T
                        Instructor2_id,Instructor3_id,Instructor4_id,Division_id,Campus_id
SELECT Course,CourseTitle,CourseSubTitle,Approval,TypeName,ClassNotes,Prefixname,Enrollment
                        Maxsize, NumEnrolled, CallNumber, BulletinFlags, PrefixLongname, T
                        Instructor_id_1, Instructor_id_2,Instructor_id_3,Instructor_id_4,Di
                        School_id,Department_id

    from course_info
    where Term = 20222;

INSERT INTO courses_20221(course_id,CourseTitle,CourseSubTitle,Approval,TypeName,ClassNotes
                        Maxsize, NumEnrolled, CallNumber, BulletinFlags, PrefixLongname, T
                        Instructor2_id,Instructor3_id,Instructor4_id,Division_id,Campus_id
SELECT Course,CourseTitle,CourseSubTitle,Approval,TypeName,ClassNotes,Prefixname,Enrollment
                        Maxsize, NumEnrolled, CallNumber, BulletinFlags, PrefixLongname, T
                        Instructor_id_1, Instructor_id_2,Instructor_id_3,Instructor_id_4,Di
                        School_id,Department_id

    from course_info
    where Term = 20221

* mysql+pymysql://root:***@localhost
4728 rows affected.
1810 rows affected.
8704 rows affected.
```

Out[50]:

[]

In [34]:

```
%%sql

ALTER TABLE course_info
ADD Instructor_id_1 int;

ALTER TABLE course_info
ADD Instructor_id_2 int;

ALTER TABLE course_info
ADD Instructor_id_3 int;

ALTER TABLE course_info
ADD Instructor_id_4 int;

* mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
```

Out[34]:

[]

In [87]:

```
%%sql

INSERT INTO course_info(Instructor_id_1)
select Instructor_id
from Instructor a
JOIN course_info b
on a.First_name = substring_index(b.Instructor2Name,',',-1) and
a.Last_name = substring_index(b.Instructor2Name,',',1)

* mysql+pymysql://root:***@localhost
24140 rows affected.
```

Out[87]:

[]

In [73]:

```
%%sql
select * from course_info
limit 10
```

\* mysql+pymysql://root:\*\*\*@localhost  
10 rows affected.

Out[73]:

Instructor3Name	DivisionCode	ChargeAmt1	SchoolCode	CourseSubtitle	Meets5	Approval	In
	BU		SBUS	Financial Accounting			S
	BU		SBUS	Accounting I: Financial A			
	BU		SBUS	Financial Accounting			
	BU		SBUS	Exemption: Accounting I - Fina			
	BU		SBUS	Financial Accounting			
	BU		SBUS	Financial Accounting			
	BU		SBUS	Financial Accounting			
	BU		SBUS	Financial Accounting			
	BU		SBUS	Financial Accounting			

In [37]:

```
%%sql

INSERT INTO course_info(Instructor_id_2)
select Instructor_id
from Instructor a
JOIN course_info b
on a.First_name = substring_index(b.Instructor2Name,',',-1) and
a.Last_name = substring_index(b.Instructor2Name,',',1)

* mysql+pymysql://root:***@localhost
24140 rows affected.
```

Out[37]:

[]

In [38]:

```
%%sql

INSERT INTO course_info(Instructor_id_3)
select Instructor_id
from Instructor a
JOIN course_info b
on a.First_name = substring_index(b.Instructor3Name,',',-1) and
a.Last_name = substring_index(b.Instructor3Name,',',1);

INSERT INTO course_info(Instructor_id_4)
select Instructor_id
from Instructor a
JOIN course_info b
on a.First_name = substring_index(b.Instructor4Name,',',-1) and
a.Last_name = substring_index(b.Instructor4Name,',',1);

* mysql+pymysql://root:***@localhost
24140 rows affected.
24140 rows affected.
```

Out[38]:

[]

In [39]:

```
%%sql
```

```
ALTER TABLE course_info  
ADD Division_id int;
```

```
ALTER TABLE course_info  
ADD Department_id int;
```

```
ALTER TABLE course_info  
ADD Campus_id int;
```

```
ALTER TABLE course_info  
ADD School_id int;
```

```
* mysql+pymysql://root:***@localhost  
0 rows affected.  
0 rows affected.  
0 rows affected.  
0 rows affected.
```

Out[39]:

```
[]
```

In [41]:

```
%%sql
```

```
INSERT INTO course_info(Division_id)  
select a.Division_id  
from Division a  
JOIN course_info b  
on a.DivisionName = b.DivisionName and  
a.DivisionCode = b.DivisionCode;
```

```
* mysql+pymysql://root:***@localhost  
24140 rows affected.
```

Out[41]:

```
[]
```

In [42]:

```
%%sql

INSERT INTO course_info(Department_id)
select a.Department_id
from Department a
JOIN course_info b
on a.DepartmentName = b.DepartmentName and
a.DepartmentCode = b.DepartmentCode;

INSERT INTO course_info(Campus_id)
select a.Campus_id
from Campus a
JOIN course_info b
on a.CampusName = b.CampusName and
a.CampusCode = b.CampusCode;

INSERT INTO course_info(School_id)
select a.School_id
from School a
JOIN course_info b
on a.SchoolName = b.SchoolName and
a.SchoolCode = b.SchoolCode;
```

```
* mysql+pymysql://root:***@localhost
24140 rows affected.
24140 rows affected.
24140 rows affected.
```

Out[42]:

[]

**The Solution is complete if the Insert Into blocks above to insert rows into course\_info work, for some reason they are not writing values into course\_info as expected**

#



In [66]:

```
%%sql

select * from course_info
limit 10
```

\* mysql+pymysql://root:\*\*\*@localhost  
10 rows affected.

Out[66]:

Instructor3Name	DivisionCode	ChargeAmt1	SchoolCode	CourseSubtitle	Meets5	Approval	In
	BU		SBUS	Financial Accounting			S
	BU		SBUS	Accounting I: Financial A			
	BU		SBUS	Financial Accounting			
	BU		SBUS	Exemption: Accounting I - Fina			
	BU		SBUS	Financial Accounting			
	BU		SBUS	Financial Accounting			
	BU		SBUS	Financial Accounting			
	BU		SBUS	Financial Accounting			
	BU		SBUS	Financial Accounting			

# NoSQL

## Introduction

- We studied inheritance/generalization/specialization when data modeling.
- We modeled person, student, employee in a HW. The folder containing the notebook has CSV files with data for employee and student.

In [17]:

```
student_df = pd.read_csv('./students.csv')  
employee_df = pd.read_csv('./employee.csv')
```

In [18]:

```
student_df.head(10)
```

Out[18]:

Unnamed: 0		id	first_name	middle_name	last_name	email
0	0	103edf18-a882-11ec-a3ad-edb7cbd2bb34	Zared	NaN	Fenelon	afenelona@theforest.net
1	1	103f6b22-a882-11ec-a3ad-edb7cbd2bb34	Bibbye	Annabal	Guesford	aguesfordb@tumblr.com
2	2	10402f4e-a882-11ec-a3ad-edb7cbd2bb34	Duffy	Auberon	Pounder	apounder2h@reuters.com
3	3	10406b1c-a882-11ec-a3ad-edb7cbd2bb34	Base	NaN	Baybutt	bbaybutty@tmall.com
4	4	1040de26-a882-11ec-a3ad-edb7cbd2bb34	Jenine	Berry	Habberjam	bhabberjam2k@examiner.com
5	5	104153d8-a882-11ec-a3ad-edb7cbd2bb34	Reece	Corbett	Caps	ccapsz@telegraph.co.uk
6	6	10419168-a882-11ec-a3ad-edb7cbd2bb34	Barry	Cullin	Elias	celias1k@scribd.com
7	7	104203d2-a882-11ec-a3ad-edb7cbd2bb34	Kerwin	Conrade	Foort	cfoortw@vinaora.com
8	8	1042383e-a882-11ec-a3ad-edb7cbd2bb34	Brigida	Cameron	Maclean	cmaclean13@mac.com
9	9	10426bc4-a882-11ec-a3ad-edb7cbd2bb34	Woodrow	Camile	Moughtin	cmoughtin17@illinois.edu



In [19]:

```
employee_df.head(10)
```

Out[19]:

Unnamed: 0		id	first_name	middle_name	last_name	email
0	0	103e6a74-a882-11ec-a3ad-edb7cbd2bb34	Sanders	Arline	Breckell	abreckell1x@fotki.com
1	1	103f2a54-a882-11ec-a3ad-edb7cbd2bb34	Ethelin	NaN	Fidele	afidele12@google.ru
2	2	103fa696-a882-11ec-a3ad-edb7cbd2bb34	Xenia	Ardella	Kief	akieft@free.f
3	3	103fe1ec-a882-11ec-a3ad-edb7cbd2bb34	Cari	Andriana	Leask	aleask1n@devhub.com
4	4	10409f88-a882-11ec-a3ad-edb7cbd2bb34	Lemmy	Burr	Bradnocke	bbradnockek@nifty.com
5	5	10411828-a882-11ec-a3ad-edb7cbd2bb34	Sibylle	Bearnard	Lalley	blalley2d@rediff.com
6	6	1041cc50-a882-11ec-a3ad-edb7cbd2bb34	Lu	Cinnamon	Flaxman	cflaxman1b@cdbaby.com
7	7	1043122c-a882-11ec-a3ad-edb7cbd2bb34	Hobart	Dominic	Croal	dcroalx@purevolume.com
8	8	10434364-a882-11ec-a3ad-edb7cbd2bb34	Marylin	Darcy	Favey	dfavey2p@mozilla.com
9	9	1043a598-a882-11ec-a3ad-edb7cbd2bb34	Ailbert	Danie	Warmisham	dwarmishame@soundcloud.com

- For the purposes of the final, you only need to worry about:

- uni
- first\_name, middle\_name, last\_name
- email
- employee\_type
- enrollment\_year

# Neo4j

- The task is to load the data into Neo4j. You do not need to worry about relationships. This question is solely about creating nodes.
- The question is focusing on how you decide to represent inheritance, e.g. that the nodes are either
  - Person and Student
  - Person and Employee
- Put your code for loading the data in the cells below.

In [16]:

```
cypher_q = "create (g:person:student {uni: $uni, first_name: $fn, middle_name: $mn, last_name: $ln, enrollment_year: $ey}) return g"
```

In [38]:

```
for i in range(len(student_df)):
    result = neo_graph.run(cypher_q, uni=str(student_df.iloc[i].uni), fn=str(student_df.iloc[i].first_name), mn = str(student_df.iloc[i].middle_name), ln = student_df.iloc[i].last_name, em = student_df.iloc[i].email, ey = str(student_df.iloc[i].enrollment_year))
```

In [39]:

```
cypher_q1 = "create (g:person:employee {uni: $uni, first_name: $fn, middle_name: $mn, last_name: $ln, employment_type: $et}) return g"
```

In [42]:

```
for i in range(len(employee_df)):
    result = neo_graph.run(cypher_q1, uni=employee_df.iloc[i].uni, fn=employee_df.iloc[i].first_name, mn = employee_df.iloc[i].middle_name, ln = employee_df.iloc[i].last_name, em = employee_df.iloc[i].email, et = employee_df.iloc[i].employment_type)
```

- Put code that shows a few nodes in the cells below.

In [43]:

```
cypher_ret = "Match (n:person {first_name:'Sanders'}) return n"
res = neo_graph.run(cypher_ret, None)
```

In [44]:

```
# result showing both the labels employee and person when filtered only with person
res
```

Out[44]:

n

```
(_705:employee:person {email: 'abreckell1x@fotki.com', employment_type: 'Professor', first_name: 'Sanders', last_name: 'Breckell', middle_name: 'Arline', uni: 'SAB1'})
```

In [45]:

```
cypher_ret = "Match (n:person {first_name:'Zared'}) return n"
res = neo_graph.run(cypher_ret, None)
```

In [46]:

```
# result showing both the labels student and person and can also be filtered using the label
res
```

Out[46]:

n

```
(_647:person:student {email: 'afenelona@theforest.net', enrollment_year: '2021', first_name: 'Zared',
last_name: 'Fenelon', middle_name: 'nan', uni: 'ZF1'})
```

## MongoDB

- The task is the same for MongoDB.
- You need to figure out how to model inheritance in a document DB. Specifically, how to model the fact that an entity is either both a Person-Employee or Person-Student.
- Put your code to load the data here.

In [49]:

```
student_dict = [{ 'uni':student_df.iloc[i].uni, 'fn':student_df.iloc[i].first_name,
                  'mn' : student_df.iloc[i].middle_name, 'ln' : student_df.iloc[i].last_name,
                  'email' : student_df.iloc[i].email,
                  'enrollment_year' : str(student_df.iloc[i].enrollment_year)} for i in range(len(student_df))]
```

In [56]:

```
for s in student_dict:
    mongo_client['person']['student'].insert_one(s)
```

In [57]:

```
employee_dict = [{ 'uni':employee_df.iloc[i].uni, 'fn':employee_df.iloc[i].first_name,
                  'mn' : employee_df.iloc[i].middle_name, 'ln' : employee_df.iloc[i].last_name,
                  'email' : employee_df.iloc[i].email,
                  'employee_type' : str(employee_df.iloc[i].employee_type)} for i in range(len(employee_df))]
```

In [60]:

```
for e in employee_dict:
    mongo_client['person']['employee'].insert_one(e)
```

- Put some simple queries that show you structure below.

In [64]:

```
res = mongo_client['person']['employee'].aggregate([{'$match':{'fn':'Sanders'}}])  
list(res)
```

Out[64]:

```
[{'_id': ObjectId('62790012bb0d72f2cfd15f51'),  
  'uni': 'SAB1',  
  'fn': 'Sanders',  
  'mn': 'Arline',  
  'ln': 'Breckell',  
  'email': 'abreckell1x@fotki.com',  
  'employee_type': 'Professor'}]
```

In [65]:

```
res = mongo_client['person']['student'].aggregate([{'$match':{'fn':'Zared'}}])  
list(res)
```

Out[65]:

```
[{'_id': ObjectId('6278ff68bb0d72f2cfd15f17'),  
  'uni': 'ZF1',  
  'fn': 'Zared',  
  'mn': nan,  
  'ln': 'Fenelon',  
  'email': 'afenelona@themeforest.net',  
  'enrollment_year': '2021'}]
```

In [ ]: