

CS6

Practical System Skills

Fall 2019 edition

Leonhard Spiegelberg
lspiegel@cs.brown.edu



Logistics

Midterm II, this Friday 4pm @ CIT 368

→ same procedure as for first Midterm

Midterm II Review lab

→ today 8pm @ CIT 201

Last homework out today

→ due after Thanksgiving, **5th Dec**

NO LECTURE on Thursday

→ I'll hold office hours in CIT 477@4pm-5:20pm instead

20 Databases II

CS6 Practical System Skills

Fall 2019

Leonhard Spiegelberg *lspiegel@cs.brown.edu*

20.01 Mapping relations to tables

⇒ Usually data can be assigned to multiple, related entities.

⇒ there are multiple ways how entities can be related to each other:

Relationship	Examples
one-to-one	one customer has exactly one primary address 1 customer \Leftrightarrow 1 shipping address
one-to-many / many-to-one	one blog entry may have 0 or more comments. 1 blog entry \Leftrightarrow N comments
many-to-many	social media friends (self referencing) N friends \Leftrightarrow M friends N Students are each enrolled in M courses N Students \Leftrightarrow M Courses


20.01 One-to-one

⇒ Use a foreign key and make it unique or place data in the same table

Customer		
id	Customer	Address
100	Tux	115 Waterman St
101	Sealion	123 California Bay
102	Crabby	789 Ocean Rd

⇒ save in one table. Here no constraint on 1:1 relation. Could enforce e.g. by making Customer + Address primary key or putting UNIQUE constraint on both fields jointly.

Customer		Shipping Address	
id	Customer	customerID	Address
100	Tux	<u>100</u>	115 Waterman St
101	Sealion	<u>101</u>	123 California Bay
102	Crabby	<u>102</u>	789 Ocean Rd



⇒ to make it a 1:1 relationship put UNIQUE constraint on customerID.

20.01 One-to-many / Many-to-one

⇒ Use a foreign key and two tables

Blog		
id	Title	text
42	Tux's first blog entry	This is a long entry...
43	Why do fairy tales always start like this?	Once upon a time...
45	Classical 20th century soap opera start	In a galaxy far, far away...

Comments	
blogID	Comment
<u>42</u>	Tux is so right about this!
<u>45</u>	Kudos! Best movie ever
<u>45</u>	Oh Disney, why...?

⇒ no restrictions on foreign key blogID and comment attribute.

20.01 Many-to-many

⇒ use a junction table (also known as associative table)

Courses		
id	Number	Title
1600	CS6	Practical System Skills
1310	CS131	Fundamentals of Computer Systems
1900	CS19	Accelerated Introduction to Computer Science

took	
course ID	studentID
<u>1600</u>	<u>30</u>
<u>1310</u>	<u>30</u>
<u>1600</u>	<u>32</u>
<u>1600</u>	<u>31</u>
<u>1900</u>	<u>30</u>

Students	
id	Name
30	Tux
31	Sealion
32	Crabby

20.01 Associations

⇒ Sometimes additional information for the connection is helpful.
Can be saved in associative/junction table as attributes.

Courses		
id	Number	Title
1600	CS6	Practical System Skills
1310	CS131	Fundamentals of Computer Systems
1900	CS19	Accelerated Introduction to Computer Science

took		
courseID	studentID	grade
<u>1600</u>	<u>30</u>	A
<u>1310</u>	<u>30</u>	A
<u>1600</u>	<u>32</u>	B
<u>1600</u>	<u>31</u>	C
<u>1900</u>	<u>30</u>	B

Students	
id	Name
30	Tux
31	Sealion
32	Crabby

20.02 Joins

⇒ to combine tables, one can join them. I.e. a combined row is constructed from matching attribute values

→ the opposite of joining a table is normalizing it

⇒ this is core part of a Database course

animals	
name	type
Tux	penguin
Tango	penguin
Sam	sealion
Crabby	crab

JOIN

diets	
animal	food
penguin	fish
sealion	penguin
squid	shrimp



*			
name	type	animal	food
Tux	penguin	penguin	fish
Tango	penguin	penguin	fish
Sam	sealion	sealion	fish

20.02 Inner Join

animals	
name	type
Tux	penguin
Tango	penguin
Sam	sealion
Crabby	crab

**INNER
JOIN**

diets	
animal	food
penguin	fish
sealion	penguin
squid	shrimp



*			
name	type	animal	food
Tux	penguin	penguin	fish
Tango	penguin	penguin	fish
Sam	sealion	sealion	fish

⇒ JOIN on some condition via

```
SELECT * FROM animals a JOIN diets d ON a.type = d.animal
```

⇒ instead of JOIN, can also write INNER JOIN

20.02 Left Outer Join

animals	
name	type
Tux	penguin
Tango	penguin
Sam	sealion
Crabby	crab

**LEFT
OUTER
JOIN**

diets	
animal	food
penguin	fish
sealion	penguin
squid	shrimp



*			
name	type	animal	food
Tux	penguin	penguin	fish
Tango	penguin	penguin	fish
Sam	sealion	sealion	fish
Crabby	crab		

⇒ JOIN on some condition via

```
SELECT * FROM animals a LEFT JOIN diets d ON a.type = d.animal
```

⇒ instead of LEFT JOIN, can also write LEFT OUTER JOIN

20.02 Right Outer Join

animals	
name	type
Tux	penguin
Tango	penguin
Sam	sealion
Crabby	crab

**RIGHT
OUTER
JOIN**

diets	
animal	food
penguin	fish
sealion	penguin
squid	shrimp



*			
name	type	animal	food
Tux	penguin	penguin	fish
Tango	penguin	penguin	fish
Sam	sealion	sealion	fish
		squid	shrimp

⇒ JOIN on some condition via

```
SELECT * FROM animals a RIGHT JOIN diets d ON a.type = d.animal
```

⇒ instead of RIGHT JOIN, can also write RIGHT OUTER JOIN

20.02 Full Outer Join

animals	
name	type
Tux	penguin
Tango	penguin
Sam	sealion
Crabby	crab

**FULL
OUTER
JOIN**

diets	
animal	food
penguin	fish
sealion	penguin
squid	shrimp



*			
name	type	animal	food
Tux	penguin	penguin	fish
Tango	penguin	penguin	fish
Sam	sealion	sealion	fish
Crabby	crab		
		squid	shrimp

⇒ JOIN on some condition via

```
SELECT * FROM animals a FULL JOIN diets d ON a.type = d.animal
```

⇒ instead of FULL JOIN, can also write FULL OUTER JOIN

Web applications & databases

Examples from today available under github.com/browncs6/DBIntro

20.03 Object - relational mapping

- ⇒ You can use a python-database adapter like `psycopg2` directly in your Flask application if you want.
- ⇒ to use a database, you need to write queries and define how to map python data to relational data → Object-relational mapping
 - ⇒ **Problem:** You might want to exchange the database and avoid wasting too much time on defining the mapping/common queries.
 - ⇒ **Solution:** There exist high-level libraries like SQLAlchemy which allow to map python data structures to a relational database (schema)

20.03 SQLAlchemy

- ⇒ is a object relational mapper
- ⇒ flask_sqlalchemy provides flask & SQLAlchemy integration
- ⇒ define objects as python classes, create necessary database tables automatically
 - **Pro:** allows to swap out database on-demand
 - **Pro:** allows to migrate to another database easily
 - **Con:** Only works for simple relationships.
- ⇒ Source for the next slides:

https://docs.sqlalchemy.org/en/13/orm/basic_relationships.html



20.03 Defining objects in SQLAlchemy

```
class Animal(Base):  
    __tablename__ = 'animals'  
  
    id = Column(Integer, primary_key=True)  
    name = Column(String)  
    type = Column(String)  
  
    def __repr__(self):  
        return '{}({})'.format(self.name, self.type)
```

Define name of SQL-table
here

Define attributes with
constraints

You can add whichever
functions you like!

⇒ Use like a python object, i.e.

```
tux = Animal(name='Tux', type='penguin')
```

SQLAlchemy - relations

20.03 SQLAlchemy - One-to-one

```
class Parent(Base):  
    __tablename__ = 'parent'  
    id = Column(Integer, primary_key=True)  
    child = relationship("Child", uselist=False, back_populates="parent")
```

Define name of
SQL-table here

uselist=False makes
it a 1:1 relation

```
class Child(Base):  
    __tablename__ = 'child'  
    id = Column(Integer, primary_key=True)  
    parent_id = Column(Integer, ForeignKey('parent.id'))  
    parent = relationship("Parent", back_populates="child")
```

Define fields here + special fields like
foreign keys referencing attributes of
other objects

Allow easy access to linked
objects! E.g.
`child.parent` to access the
parent object

20.03 SQLAlchemy - One-to-many

```
class Parent(Base):  
    __tablename__ = 'parent'  
    id = Column(Integer, primary_key=True)  
    children = relationship("Child", back_populates="parent")
```

This is probably the most frequently used pattern you'll encounter!

```
class Child(Base):  
    __tablename__ = 'child'  
    id = Column(Integer, primary_key=True)  
    parent_id = Column(Integer, ForeignKey('parent.id'))  
    parent = relationship("Parent", back_populates="children")
```

20.03 SQLAlchemy Many-to-many

```
association_table = Table('association', Base.metadata,
    Column('left_id', Integer, ForeignKey('left.id')),
    Column('right_id', Integer, ForeignKey('right.id'))
)
```

```
class Parent(Base):
    __tablename__ = 'left'
    id = Column(Integer, primary_key=True)
    children = relationship(
        "Child",
        secondary=association_table,
        back_populates="parents")
```

```
class Child(Base):
    __tablename__ = 'right'
    id = Column(Integer, primary_key=True)
    parents = relationship(
        "Parent",
        secondary=association_table,
        back_populates="children")
```

Need to define additional
junction/association table

More complicated
referencing to setup easy
object like access of data

20.04 Using SQLAlchemy in Flask

⇒ detailed information in Chapter 5, Flask book.

⇒ we can use SQLAlchemy directly in Flask!

```
@app.route('/', methods=['GET', 'POST'])
def index():

    # check if post request
    # if so, add new pokemon!
    if request.method == 'POST':
        poke = Pokemon(name=request.form['name'],
                        category=request.form['category'],
                        height_ft=request.form['height_ft'],
                        weight_lbs=request.form['weight_lbs'])

        db.session.add(poke)    # add to database (transaction)
        db.session.commit()    # commit transaction

    pokemon = Pokemon.query.all()
    return render_template('index.html', pokemon = pokemon)
```



Document stores

20.05 Document stores

⇒ So far: Mapping objects to tables.

→ Why not store objects directly?

```
class Comment:
    def __init__(self, text, author):
        self.text = text
        self.author = author

class Blog:
    def __init__(self, title, text, author, comments):
        self.title = title
        self.text = text
        self.author = author
        self.comments = comments
```




JSON representation:

```
{
  "title": "Tux states a truth",
  "text": "Sealions and penguins will never
be friends!",
  "author": "Tux",
  "comments": [
    {
      "author": "Sealion",
      "text": "This is not true, we had fish
together last Sunday."
    },
    {
      "author": "Crabby",
      "text": "I so agree, these sealions
think they can get away with anything."
    }
  ]
}
```


20.05 Document stores

Relational database	Document store
table	collection
rows/tuples	documents
columns/attributes	fields



name	category
Pikachu	mouse
Bulbasaur	seed
Charmander	lizard

collection

document

```
[  
  {  
    "name": "Pikachu",  
    "category": "mouse"  
  },  
  {  
    "name": "Bulbasaur",  
    "category": "seed"  
  },  
  {  
    "name": "Charmander",  
    "category": "lizard"  
  }  
]
```

field

20.06 MongoDB

Widely adopted document store.

Good for

- (1) Web applications
- (2) Real-time analytics & high-speed logging
- (3) Caching

⇒ **Don't use** when SQL is a better fit or for highly transactional applications

In MongoDB there are

- no joins
- no multi-document transactions (i.e. can't have ACID with multiple tables!)



20.06 Working with MongoDB

⇒ Mongo Shell or Python adapter

→ there is also a flask package! `pip3 install Flask-PyMongo`

⇒ Think of a collection as holding zero or more JSON like documents, access/update of these also works in dict-like fashion

⇒ primary resource for how to work with MongoDB:

<https://api.mongodb.com/python/current/api/pymongo/collection.html>

⇒ Jupyter notebook in materials!

Data analytics

20.07 Data analytics via SQL

⇒ so far: Storing & Retrieving data from tables.

⇒ We can use SQL also to aggregate data!

SELECT ..., AGG(...)

Define what to select,
what aggregates, e.g.
AVG/MIN/MAX/SUM/...

FROM ...

WHERE ...

condition rows to be
aggregated must satisfy

GROUP BY ...

aggregate over attribute
groups

ORDER BY AGG(...)

sort result

HAVING ...

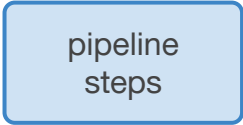
condition to filter over
groups based on
aggregates

20.08 Data analytics in MongoDB

⇒ MongoDB also provides aggregations over multiple documents

⇒ uses aggregation pipelines consisting of multiple steps

```
db.flights.aggregate([  
    { '$match' : { 'ORIGIN_CITY_NAME' : { '$regex' : 'New York' } } },  
    { '$group' : { '_id' : '$OP_UNIQUE_CARRIER',  
                    'total': { '$sum' : 1 } } },  
    { '$sort' : { 'total' : -1 } }  
])
```



20.09 Data analytics - which tool to use?

⇒ So far we've learned many tools

→ Which tool to actually use for analytics?

1. use bash tools & scripting: uniq/sort/awk/sed/grep/...!
2. write a python script!
3. load data into SQL database and analyze!
4. load data into a document store and analyze!

⇒ This is **HW10!** Explore which tool to use for data analytics.

End of lecture.

Next class: Fri, 4pm @ CIT 368 Midterm II