Calculating values in a query

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

Math operators

- addition `+`
- subtraction `-`
- multiplication `*`
- division `/`
- modulus `%`
- Work differently on different data types

Calculating difference

```
stmt = select([census.columns.age,
        (census.columns.pop2008 -
        census.columns.pop2000).label('pop_change')
    ])
stmt = stmt.group_by(census.columns.age)
stmt = stmt.order_by(desc('pop_change'))
stmt = stmt.limit(5)
results = connection.execute(stmt).fetchall()
print(results)
```

```
[(61, 52672), (85, 51901), (54, 50808), (58, 45575),
(60, 44915)]
```

Case statement

- Used to treat data differently based on a condition
- Accepts a list of conditions to match and a column to return if the condition matches
- The list of conditions ends with an else clause to determine what to do when a record doesn't match any prior conditions

Case example

```
from sqlalchemy import case

stmt = select([
        func.sum(
            case([
                (census.columns.state == 'New York',
                 census.columns.pop2008)
            ], else_=0))])

results = connection.execute(stmt).fetchall()
print(results)
```

```
[(19465159,)]
```

Cast statement

- Converts data to another type
- Useful for converting...
  - integers to floats for division
  - strings to dates and times
- Accepts a column or expression and the target Type

Percentage example

```
from sqlalchemy import case, cast, Float
stmt = select([
        (func.sum(
            case([
                (census.columns.state == 'New York',
                 census.columns.pop2008)
            ], else_=0)) /
         cast(func.sum(census.columns.pop2008),
              Float) * 100).label('ny_percent')])
results = connection.execute(stmt).fetchall()
print(results)
```

```
[(Decimal('6.4267619765'),)]
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

SQL relationships

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

Relationships

- Allow us to avoid duplicate data
- Make it easy to change things in one place
- Useful to break out information from a table we don't need very often

Relationships

Census

state	sex	age	pop2000	pop2008
New York	F	0	120355	122194
New York	F	1	118219	119661
New York	F	2	119577	116413

State_Fact

name	abbreviation	type
New York	NY	state
Washington DC	DC	capitol
Washington	WA	state

Automatic joins

```
stmt = select([census.columns.pop2008,
               state_fact.columns.abbreviation])

results = connection.execute(stmt).fetchall()
print(results)
```

```
[(95012, u'IL'),
 (95012, u'NJ'),
 (95012, u'ND'),
 (95012, u'OR'),
 (95012, u'DC'),
 (95012, u'WI'),
 ...
```

Join

- Accepts a Table and an optional expression that explains how the two tables are related
- The expression is not needed if the relationship is predefined and available via reflection
- Comes immediately after the `select()` clause and prior to any `where()`, `order_by()` or `group_by()` clauses

select_from()

- Used to replace the default, derived FROM clause with a join
- Wraps the `join()` clause

select_from() example

```
stmt = select([func.sum(census.columns.pop2000)])

stmt = stmt.select_from(census.join(state_fact))

stmt = stmt.where(state_fact.columns.circuit_court == '10')

result = connection.execute(stmt).scalar()
print(result)
```

```
14945252
```

Joining tables without predefined relationship

- Join accepts a Table and an optional expression that explains how the two tables are related
- Will only join on data that match between the two columns
- Avoid joining on columns of different types

select_from() example

```
stmt = select([func.sum(census.columns.pop2000)])
stmt = stmt.select_from(
        census.join(state_fact, census.columns.state
        == state_fact.columns.name))
stmt = stmt.where(
        state_fact.columns.census_division_name ==
        'East South Central')
result = connection.execute(stmt).scalar()
print(result)
```

```
16982311
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Working with hierarchical tables

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

Hierarchical tables

- Contain a relationship with themselves
- Commonly found in:
  - Organizational
  - Geographic
  - Network
  - Graph

Hierarchical tables - example

Employees

id	name	job	manager
1	Johnson	Admin	6
2	Harding	Manager	9
3	Taft	Sales I	2
4	Hoover	Sales I	2

Hierarchical tables - alias()

- Requires a way to view the table via multiple names
- Creates a unique reference that we can use

Querying hierarchical data

```
managers = employees.alias()
stmt = select(
        [managers.columns.name.label('manager'),
         employees.columns.name.label('employee')])
stmt = stmt.select_from(employees.join(
        managers, managers.columns.id ==
        employees.columns.manager)
stmt = stmt.order_by(managers.columns.name)
print(connection.execute(stmt).fetchall())
```

```
[(u'FILLMORE', u'GRANT'),
 (u'FILLMORE', u'ADAMS'),
 (u'HARDING', u'TAFT'), ...
```

group_by and func

- It's important to target `group_by()` at the right alias
- Be careful with what you perform functions on
- If you don't find yourself using both the alias and the table name for a query, don't create the alias at all

Querying hierarchical data

```
managers = employees.alias()
stmt = select([managers.columns.name,
       func.sum(employees.columns.sal)])
stmt = stmt.select_from(employees.join(
       managers, managers.columns.id ==
       employees.columns.manager)
stmt = stmt.group_by(managers.columns.name)
print(connection.execute(stmt).fetchall())
```

```
[(u'FILLMORE', Decimal('96000.00')),
 (u'GARFIELD', Decimal('83500.00')),
 (u'HARDING', Decimal('52000.00')),
 (u'JACKSON', Decimal('197000.00'))]
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON

Handling large ResultSets

INTRODUCTION TO DATABASES IN PYTHON

Jason Myers

Co-Author of Essential SQLAlchemy and Software Engineer

Dealing with large ResultSets

- `fetchmany()` lets us specify how many rows we want to act upon
- We can loop over `fetchmany()`
- It returns an empty list when there are no more records
- We have to close the `ResultProxy` afterwards

Fetching many rows

```
while more_results:

    partial_results = results_proxy.fetchmany(50)

    if partial_results == []:
        more_results = False

    for row in partial_results:
        state_count[row.state] += 1

results_proxy.close()
```

Let's practice!

INTRODUCTION TO DATABASES IN PYTHON