Advanced Data Management (CSCI 640/490)

Structured Data

Dr. David Koop

Relational Algebra

- Definition: A procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.
- Six basic operators
  - select: σ
  - project: ∏
  - union: ∪
  - set difference: –
  - Cartesian product: x
  - rename: ρ

[A. Silberschatz et al.]

Select Operation

- The select operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- p is called the selection predicate
- Example: select those tuples of the `instructor` relation where the instructor is in the "Physics" department.
  - Query: $\sigma_{\text{dept\_name}=\text{“Physics”}}(\texttt{instructor})$
  - Result:

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

[A. Silberschatz et al.]

Project Operation

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

- Example: eliminate the `dept_name` attribute of instructor
- Query: $\Pi_{\text{ID, name, salary}}$ (`instructor`)

[A. Silberschatz et al.]

Cartesian-Product Operation

- The **Cartesian-product** operation (denoted by X) allows us to combine information from any two relations.
- Example: the Cartesian product of the relations `instructor` and `teaches` is written as: `instructor` X `teaches`
- We construct a tuple of the result out of **each possible pair** of tuples: one from the instructor relation and one from the teaches relation
- Since the instructor ID appears in both relations we distinguish between these attribute by attaching to the attribute the name of the relation from which the attribute originally came.
  - `instructor.ID` and `teaches.ID`

[A. Silberschatz et al.]

Join Operation

- The Cartesian-Product `instructor` X `teaches` associates every tuple of instructor with every tuple of teaches.
  - Most of the resulting rows have information about instructors who **did not** teach a particular course.
- To get only those tuples of `instructor` X `teaches` that pertain to instructors and the courses that they taught, we write:

$$\sigma_{\text{instructor.id} = \text{teaches.id}} (\texttt{instructor} \times \texttt{teaches})$$

  - We get only those tuples of `instructor` X `teaches` that pertain to instructors and the courses that they taught.

[A. Silberschatz et al.]

Equivalent Queries

- Example: Find information about courses taught by instructors in the Physics department
- Query 1:

$$\sigma_{\text{dept\_name=“Physics”}} (\texttt{instructor} \bowtie_{\text{instructor.ID = teaches.ID}} \texttt{teaches})$$

- Query 2

$$(\sigma_{\text{dept\_name=“Physics”}} (\texttt{instructor})) \bowtie_{\text{instructor.ID = teaches.ID}} \texttt{teaches}$$

- The **order** of joins is one focus of some of the work on query optimization

[A. Silberschatz et al.]

Components of SQL

- **Data Definition Language (DDL)**: the specification of information about relations, including schema, types, integrity constraints, indices, storage
- **Data Manipulation Language (DML)**: provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
- **Integrity**: the DDL includes commands for specifying integrity constraints.
- **View definition**: The DDL includes commands for defining views.
- Also: **Transaction control**, **embedded and dynamic SQL**, **authorization**

[A. Silberschatz et al.]

Create Table

- An SQL relation is defined using the create table command:

  `create table r` $(A_1\ D_1, A_2\ D_2, \ldots, A_n\ D_n, (C_1), \ldots, (C_k))$

  - `r` is the **name** of the relation
  - each A_i is an **attribute name** in the schema of relation `r`
  - D_i is the **data type** of values in the domain of attribute A_i

- Example:

```
create table instructor(
        ID              char(5),
        name            varchar(20),
        dept_name       varchar(20),
        salary          numeric(8,2));
```

C_i are integrity constraints: keys, foreign keys

[A. Silberschatz et al.]

Basic Query Structure

- A typical SQL query has the form:

  `select` $A_1, A_2, \ldots, A_n$

  `from` $r_1, r_2, \ldots, r_m$

  `where` P

  - A_i represents an **attribute**
  - r_i represents a **relation**
  - P is a **predicate**.

- The result of an SQL query is a **relation**

[A. Silberschatz et al.]

Select

- The **select** clause lists the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: Find the names of all instructors
  - ```
    select name
    from instructor;
    ```
- Note: SQL names are **case insensitive**
  - `Name` and `NAME` and `name` are equivalent
  - Some people use upper case for language keywords (e.g. `SELECT`)

[A. Silberschatz et al.]
    ```

Where

- The operands can be expressions with operators <, <=, >, >=, =, and **<>**
- SQL allows the use of the logical connectives `and`, `or`, and `not`
- Comparisons can be applied to results of arithmetic expressions
- Example: Find all instructors in Comp. Sci. with salary > 70000
  -
```
select name
from instructor
where dept_name = 'Comp. Sci.' and salary > 70000
```

name
Katz
Brandt

[A. Silberschatz et al.]

From

- The **from** clause lists the relations involved in the query
  - Corresponds to the **Cartesian Product** operation in relational algebra
- Find the Cartesian product `instructor` X `teaches`
  - ```
    select *
    from instructor, teaches;
    ```
  - All possible `instructor` – `teaches` pair, with all attributes from both
  - Shared attributes (e.g., `ID`) are renamed (e.g., `instructor.ID`)
- Not very useful directly but useful combined with where clauses.

[A. Silberschatz et al.]
    ```

Assignment 2

- Same questions as Assignment 1 but using pandas, duckdb, and ibis

Group By

- Find the average salary of instructors in each department
  - ```
    select dept_name, avg(salary) as avg_salary
    from instructor
    group by dept_name;
    ```

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

[A. Silberschatz et al.]
    ```

Group By

- Find the average salary of instructors in each department
  -
```
select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name;
```

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

[A. Silberschatz et al.]

Group By

- Find the average salary of instructors in each department
  - ```
    select dept_name, avg(salary) as avg_salary
    from instructor
    group by dept_name;
    ```

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

[A. Silberschatz et al.]
    ```

Having Clause

- Filter groups based on predicates
- Predicates in the having clause are applied **after** the formation of groups whereas predicates in the where clause are applied **before** forming groups
- Example: Find the names and average salaries of all departments whose average salary is greater than 42,000
  -
```
select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name
having avg(salary) > 42000;
```

[A. Silberschatz et al.]

Modification of the Database

- Deleting tuples from a given relation.
- Inserting new tuples into a given relation
- Updating values in some tuples in a given relation

[A. Silberschatz et al.]

Deletion

- Delete all instructors: `delete from instructor;`
- Delete all instructors from the Finance department
  -
```
delete from instructor
where dept_name= 'Finance';
```
- Delete all tuples in the instructor relation for those instructors associated with a department located in the Watson building
  -
```
delete from instructor
where dept_name in (select dept_name
                    from department
                    where building = 'Watson');
```

[A. Silberschatz et al.]

Deletion

- Delete all instructors: `delete from instructor;`
- Delete all instructors from the Finance department
  -
```
delete from instructor
where dept_name= 'Finance';
```
- Delete all tuples in the instructor relation for those instructors associated with a department located in the Watson building
  -
```
delete from instructor
where dept_name in (select dept_name
                    from department
                    where building = 'Watson');
```

[A. Silberschatz et al.]

Insertion

- Add a new tuple to course
  - `insert into course values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);`
- or…
  - `insert into course(course_id, title, dept_name, credits) values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);`
- Add a new tuple to `student` with `tot_creds` set to `null`
  - `insert into student values ('3003', 'Green', 'Finance', null);`

[A. Silberschatz et al.]

Insertion

- Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of $18,000.
  -
```
insert into instructor
    select ID, name, dept_name, 18000
    from student
    where dept_name = 'Music' and total_cred > 144;
```
- The select-from-where statement is evaluated fully before any of its results are inserted into the relation.
- If not queries like

```
insert into table1 select * from table1
```

  would cause problems

[A. Silberschatz et al.]

Updates

- Give a 5% salary raise to all instructors
  -
```
update instructor
set salary = salary * 1.05
```
- Give a 5% salary raise to those instructors who earn less than 70000
  -
```
update instructor
set salary = salary * 1.05
where salary < 70000;
```
- Give a 5% salary raise to instructors whose salary is less than average
  -
```
update instructor
set salary = salary * 1.05
where salary < (select avg(salary) from instructor);
```

[A. Silberschatz et al.]

Updates

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Use two update statements:
  - ```
    update instructor
    set salary = salary * 1.03
    where salary > 100000;
    ```
  - ```
    update instructor
    set salary = salary * 1.05
    where salary <= 100000;
    ```
  - What happens if we swap the order of these statements?

[A. Silberschatz et al.]

Joins

- Join operations take two relations and return another relation.
- From relational algebra, this is a Cartesian product + selection
- Want tuples in the two relations to match (under some condition)
- The join operations typically used as subquery expressions in the from clause
- Three types of joins:
  - Natural join
  - Inner join
  - Outer join

[A. Silberschatz et al.]

Natural Join

- Natural join matches tuples with the same values for all **common** attributes, and retains only **one copy** of each common column.
- List the names of instructors along with the course ID of the courses that they taught
  - ```
    select name, course_id
    from students, takes
    where student.ID = takes.ID;
    ```
- Same query in SQL with “natural join” construct
  - ```
    select name, course_id
    from student natural join takes;
    ```

[A. Silberschatz et al.]

Example: Student Schedules

ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

ID	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	*null*

[A. Silberschatz et al.]

Example: Natural Join

ID	*name*	*dept_name*	*tot_cred*	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2017	A
00128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2017	A-
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2017	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2017	A
12345	Shankar	Comp. Sci.	32	CS-315	1	Spring	2018	A
12345	Shankar	Comp. Sci.	32	CS-347	1	Fall	2017	A
19991	Brandt	History	80	HIS-351	1	Spring	2018	B
23121	Chavez	Finance	110	FIN-201	1	Spring	2018	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2017	B-
45678	Levy	Physics	46	CS-101	1	Fall	2017	F
45678	Levy	Physics	46	CS-101	1	Spring	2018	B+
45678	Levy	Physics	46	CS-319	1	Spring	2018	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2017	A-
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2017	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2018	A-
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2017	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2018	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2017	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2017	C-
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2018	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2017	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2018	*null*

[A. Silberschatz et al.]

Natural Join Danger

- Beware of unrelated attributes with same name which get equated incorrectly
- Example: List the names of students instructors along with the titles of courses that they have taken
  - ```
    select name, title
    from student natural join takes natural join course;
    ```
- Wrong… only lists courses when the student took courses in their department (major)
- Correct:
  - ```
    select name, title
    from student natural join takes, course
    where takes.course_id = course.course_id;
    ```

[A. Silberschatz et al.]

Outer Join

- Joins so far are inner joins
- Outer joins returns tuples from one (or both) relations that do not match tuples in the other relation
- Fills in missing values with null
- Three forms of outer join:
  - **left** outer join
  - **right** outer join
  - full **outer** join

[A. Silberschatz et al.]

Join Examples

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

`course`

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

`prereq`

Left Join

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

Right Join

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

[A. Silberschatz et al.]

Join Examples

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

`course`

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

`prereq`

(Full) Outer Join

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

Inner Join

course_id	*title*	*dept_name*	*credits*	*prereq_id*	*course_id*
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

[A. Silberschatz et al.]

Dataframe Model

Data Frame

```
df = pd.read_csv('penguins_lter.csv')
```

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7
...	...	...	...	...	...	...	...	...	...	...
339	PAL0910	120	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N38A2	No	12/1/09	NaN
340	PAL0910	121	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A1	Yes	11/22/09	46.8
341	PAL0910	122	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A2	Yes	11/22/09	50.4
342	PAL0910	123	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A1	Yes	11/22/09	45.2
343	PAL0910	124	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A2	Yes	11/22/09	49.9

344 rows × 17 columns

Data Frame

Column Names

```
df = pd.read_csv('penguins_lter.csv')
```

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7
...	...	...	...	...	...	...	...	...	...	...
339	PAL0910	120	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N38A2	No	12/1/09	NaN
340	PAL0910	121	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A1	Yes	11/22/09	46.8
341	PAL0910	122	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A2	Yes	11/22/09	50.4
342	PAL0910	123	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A1	Yes	11/22/09	45.2
343	PAL0910	124	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A2	Yes	11/22/09	49.9

344 rows × 17 columns

Data Frame

```
df = pd.read_csv('penguins_lter.csv')
```

Column Names

Index

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7
...	...	...	...	...	...	...	...	...	...	...
339	PAL0910	120	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N38A2	No	12/1/09	NaN
340	PAL0910	121	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A1	Yes	11/22/09	46.8
341	PAL0910	122	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A2	Yes	11/22/09	50.4
342	PAL0910	123	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A1	Yes	11/22/09	45.2
343	PAL0910	124	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A2	Yes	11/22/09	49.9

344 rows × 17 columns

Data Frame

```
df = pd.read_csv('penguins_lter.csv')
```

Column Names

Index

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7
...	...	...	...	...	...	...	...	...	...	...
339	PAL0910	120	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N38A2	No	12/1/09	NaN
340	PAL0910	121	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A1	Yes	11/22/09	46.8
341	PAL0910	122	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A2	Yes	11/22/09	50.4
342	PAL0910	123	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A1	Yes	11/22/09	45.2
343	PAL0910	124	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A2	Yes	11/22/09	49.9

344 rows × 17 columns

Column: `df['Island']`

Data Frame

```
df = pd.read_csv('penguins_lter.csv')
```

Column Names

Row: `df.loc[2]`

Index

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7
...	...	...	...	...	...	...	...	...	...	...
339	PAL0910	120	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N38A2	No	12/1/09	NaN
340	PAL0910	121	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A1	Yes	11/22/09	46.8
341	PAL0910	122	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A2	Yes	11/22/09	50.4
342	PAL0910	123	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A1	Yes	11/22/09	45.2
343	PAL0910	124	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A2	Yes	11/22/09	49.9

344 rows × 17 columns

Column: `df['Island']`

Data Frame

```
df = pd.read_csv('penguins_lter.csv')
```

Column Names

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7
...	...	...	...	...	...	...	...	...	...	...
339	PAL0910	120	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N38A2	No	12/1/09	NaN
340	PAL0910	121	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A1	Yes	11/22/09	46.8
341	PAL0910	122	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A2	Yes	11/22/09	50.4
342	PAL0910	123	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A1	Yes	11/22/09	45.2
343	PAL0910	124	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A2	Yes	11/22/09	49.9

344 rows × 17 columns

Row: `df.loc[2]`

Index

Cell: `df.loc[341,'Species']`

Column: `df['Island']`

Data Frame

```
df = pd.read_csv('penguins_lter.csv')
```

Column Names

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	
...	...	...	...	...	...	...	...	...	...	...
339	PAL0910	120	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N38A2	No	12/1/09	NaN
340	PAL0910	121	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A1	Yes	11/22/09	46.8
341	PAL0910	122	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A2	Yes	11/22/09	50.4
342	PAL0910	123	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A1	Yes	11/22/09	45.2
343	PAL0910	124	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A2	Yes	11/22/09	49.9

344 rows × 17 columns

Row: `df.loc[2]`

Index

Cell: `df.loc[341,'Species']`

Missing Data

Column: `df['Island']`

Arrays

What is the difference between an array and a list (or a tuple)?

Arrays

- Usually a fixed size—lists are meant to change size
- Are mutable—tuples are not
- Store only one type of data—lists and tuples can store anything
- Are faster to access and manipulate than lists or tuples
- Can be multidimensional:
  - Can have list of lists or tuple of tuples but no guarantee on shape
  - Multidimensional arrays are rectangles, cubes, etc.

Why NumPy?

- Fast **vectorized** array operations for data munging and cleaning, subsetting and filtering, transformation, and any other kinds of computations
- Common array algorithms like sorting, unique, and set operations
- Efficient descriptive statistics and aggregating/summarizing data
- Data alignment and relational data manipulations for merging and joining together heterogeneous data sets
- Expressing conditional logic as array expressions instead of loops with `if-elif-else` branches
- Group-wise data manipulations (aggregation, transformation, function application).

[W. McKinney, Python for Data Analysis]

```
import numpy as np
```

PyData Notebooks

- https://github.com/wesm/pydata-book/
- ch04.ipynb
- Click the raw button and save that file to disk
- …or download/clone the entire repository

Creating arrays

- `data1 = [6, 7, 8, 0, 1]`
  `arr1 = np.array(data1)`
- `data2 = [[1.5,2,3,4],[5,6,7,8]]`
  `arr2 = np.array(data2)`
- `data3 = np.array([6, "abc", 3.57]) # !!! check !!!`
- Can check the type of an array in `dtype` property
- Types:
  - `arr1.dtype # dtype('int64')`
  - `arr3.dtype # dtype('<U21'), unicode plus # chars`

Types

- "But I thought Python wasn't stingy about types…"
- numpy aims for speed
- Able to do array arithmetic
- int16, int32, int64, float32, float64, bool, object
- Can specify type explicitly
  - `arr1_float = np.array(data1, dtype='float64')`
- `astype` method allows you to convert between different types of arrays:

```
arr = np.array([1, 2, 3, 4, 5])
arr.dtype
float_arr = arr.astype(np.float64)
```

numpy data types (dtypes)

Type	Type code	Description
`int8, uint8`	`i1, u1`	Signed and unsigned 8-bit (1 byte) integer types
`int16, uint16`	`i2, u2`	Signed and unsigned 16-bit integer types
`int32, uint32`	`i4, u4`	Signed and unsigned 32-bit integer types
`int64, uint64`	`i8, u8`	Signed and unsigned 64-bit integer types
`float16`	`f2`	Half-precision floating point
`float32`	`f4 or f`	Standard single-precision floating point; compatible with C float
`float64`	`f8 or d`	Standard double-precision floating point; compatible with C double and Python `float` object
`float128`	`f16 or g`	Extended-precision floating point
`complex64, complex128, complex256`	`c8, c16, c32`	Complex numbers represented by two 32, 64, or 128 floats, respectively
`bool`	`?`	Boolean type storing `True` and `False` values
`object`	`O`	Python object type; a value can be any Python object
`string_`	`S`	Fixed-length ASCII string type (1 byte per character); for example, to create a string dtype with length 10, use `'S10'`
`unicode_`	`U`	Fixed-length Unicode type (number of bytes platform specific); same specification semantics as `string_` (e.g., `'U10'`)

[W. McKinney, Python for Data Analysis]