ECE30030/ITP30010 Database Systems

Structured Query Language

Reading: Chapter 3

Charmgil Hong

charmgil@handong.edu

Spring, 2023

Handong Global University

Announcement

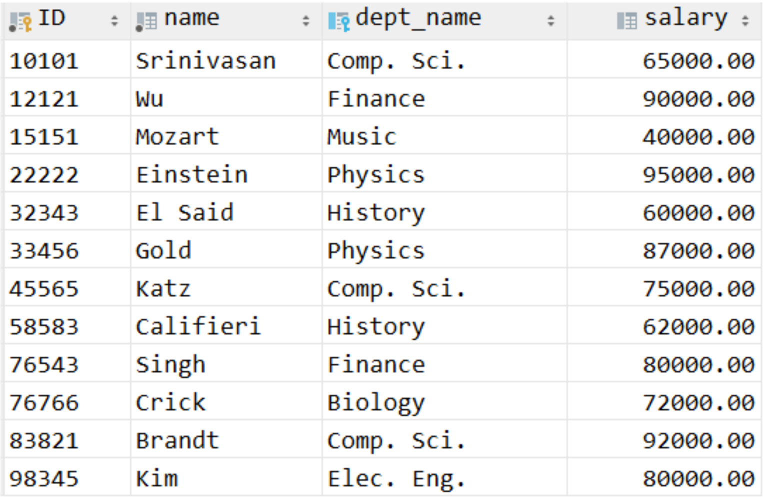
- Homework assignment #2 is out
  - Due: By the end of Friday, April 7
  - Please start early

- Rules related to late submissions
  - Late submissions will be accepted within 24 hours after the deadline with a penalty of -20% of the assignment grade
    - Submissions made after 24 hours from the deadline will be rejected
  - For additional extensions, reasonable excuses should be submitted before the deadline

Running Examples

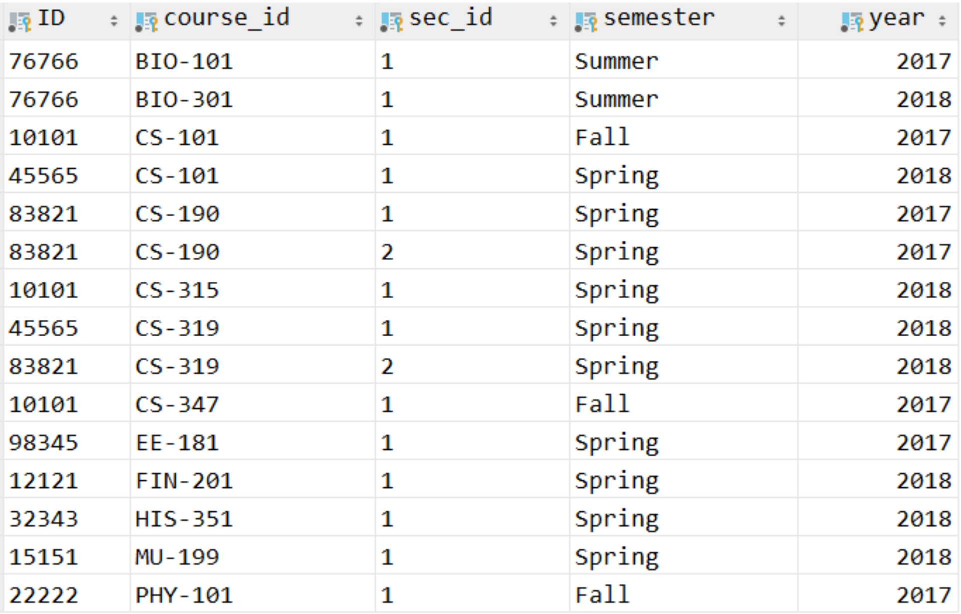
- Relations (tables): *instructor, teaches*

Instructor relation



ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

teaches relation

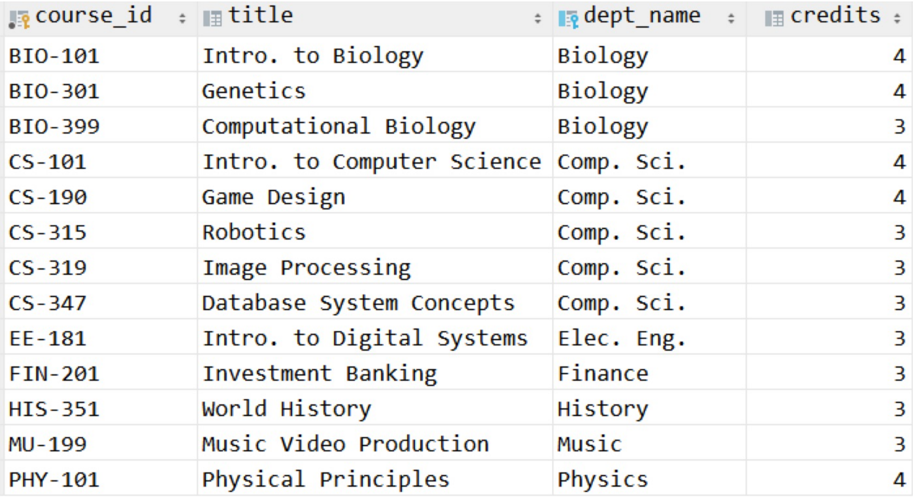


ID	course_id	sec_id	semester	year
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
10101	CS-101	1	Fall	2017
45565	CS-101	1	Spring	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
10101	CS-315	1	Spring	2018
45565	CS-319	1	Spring	2018
83821	CS-319	2	Spring	2018
10101	CS-347	1	Fall	2017
98345	EE-181	1	Spring	2017
12121	FIN-201	1	Spring	2018
32343	HIS-351	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017

Running Examples

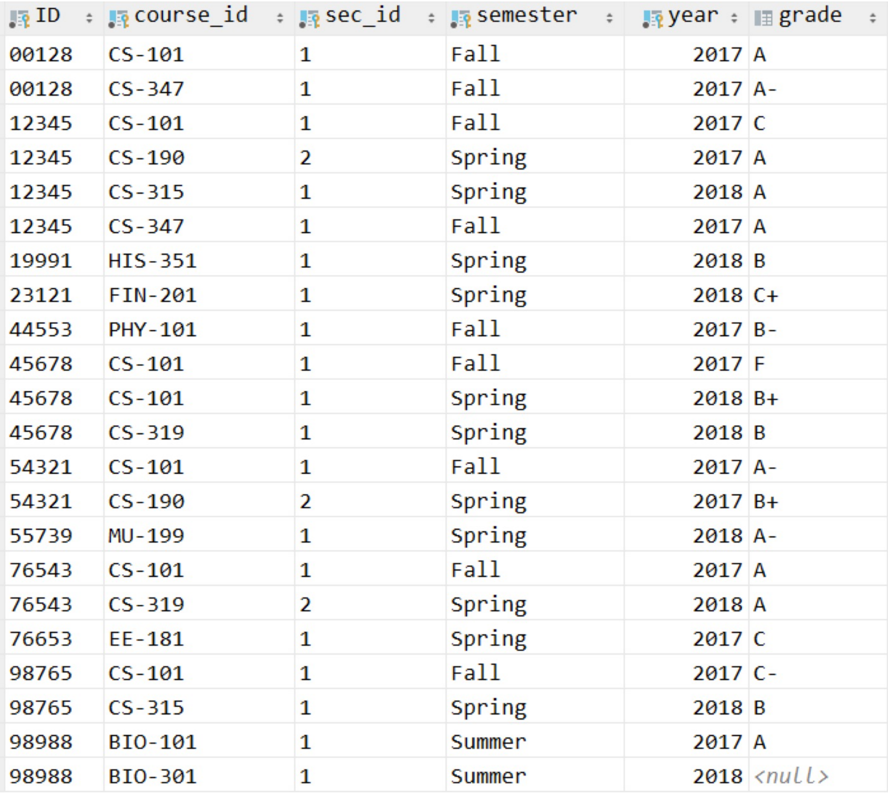
- Relations (tables): *course*, *takes*

course relation



course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

takes relation

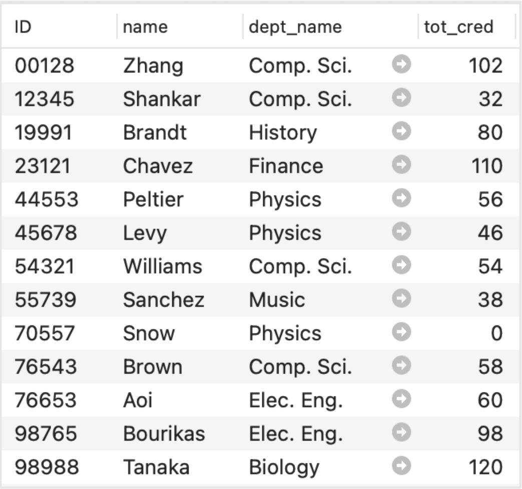


ID	course_id	sec_id	semester	year	grade
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	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>

Running Examples

- Relations (tables): *student*

student relation



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

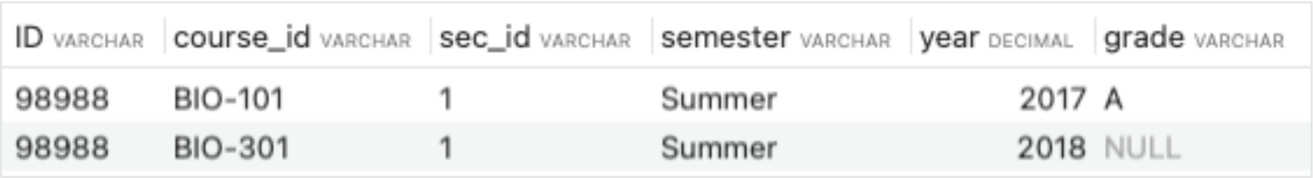
Agenda

- Structured query language (SQL)
- **SQL data manipulation language (DML)**
  - SELECT, FROM, WHERE
  - **NULL values**
  - Set operations
  - String operations, ordering
  - Aggregate functions, aggregation
- SQL data definition language (DDL)

NULL Values

- It is possible for tuples to have a NULL value for some of their attributes
  - NULL signifies an unknown value or that a value does not exist

- The result of any arithmetic expression involving NULL is NULL
  - *E.g.,* 5 + NULL returns NULL



ID VARCHAR	course_id VARCHAR	sec_id VARCHAR	semester VARCHAR	year DECIMAL	grade VARCHAR
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	NULL

IS NULL / IS NOT NULL

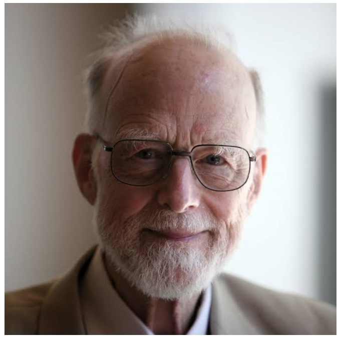
- The predicate IS NULL can be used to check for NULL values
  - *E.g.*, Find all instructors whose salary is null

    SELECT *name*
    FROM *instructor*
    WHERE *salary* IS NULL

- The predicate IS NOT NULL succeeds if the value on which it is applied is not null

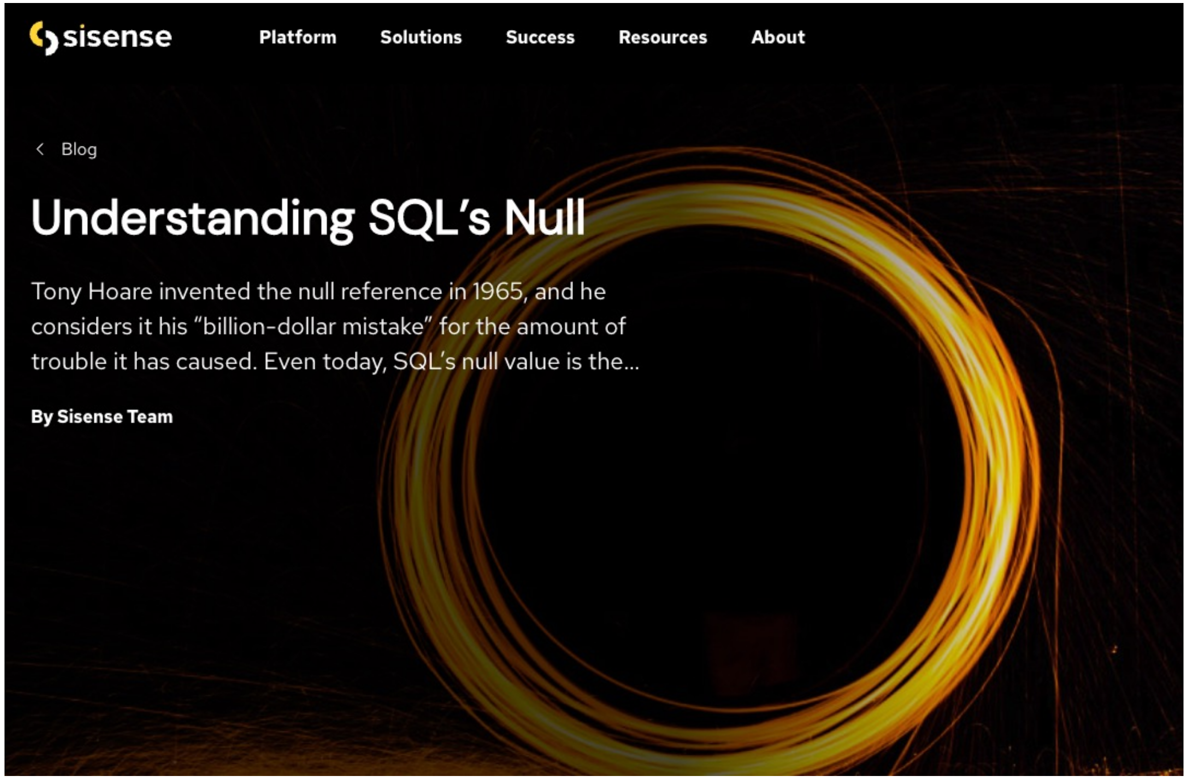
NULL Values in CS

- Invented by Sir Charles Antony Richard Hoare (Tony Hoare)
  - *“I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object-oriented language (ALGOL W).*
    My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”



NULL Values in CS

- Interesting discussion
  - Understanding SQL's Null
    https://www.sisense.com/blog/understanding-sql-null/?utm_source=trendemon&utm_medium=content&utm_campaign=flow



Understanding SQL's Null

Tony Hoare invented the null reference in 1965, and he considers it his "billion-dollar mistake" for the amount of trouble it has caused. Even today, SQL's null value is the...

By Sisense Team

Agenda

- Structured query language (SQL)
- **SQL data manipulation language (DML)**
  - SELECT, FROM, WHERE
  - NULL values
  - **Set operations**
  - String operations, ordering
  - Aggregate functions, aggregation
- SQL data definition language (DDL)

Set Operations

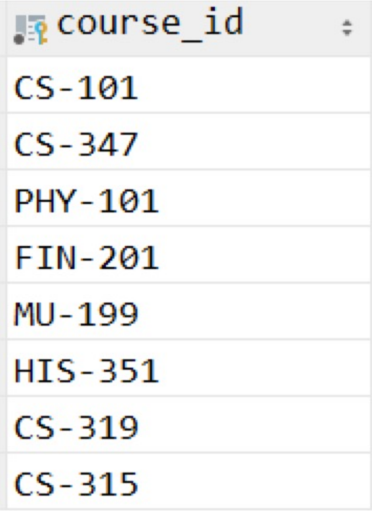
- Set operations **UNION, INTERSECT,** and **EXCEPT**
  - Each of the above operations automatically eliminates duplicates

- To retain all duplicates, use ALL:
  - **UNION ALL**
  - **INTERSECT ALL**
  - **EXCEPT ALL**

- *C.f.*, SELECT retains all duplicates by default

Set Operations: UNION

- Find courses that ran in Fall 2017 or in Spring 2018
  - (**SELECT** *course_id* **FROM** *teaches* **WHERE** *semester* = 'Fall' **AND** *year* = 2017)
    UNION
    (**SELECT** *course_id* **FROM** *teaches* **WHERE** *semester* = 'Spring' **AND** *year* = 2018)

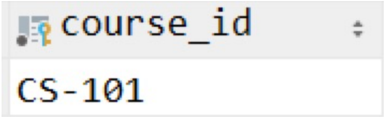


course_id
CS-101
CS-347
PHY-101
FIN-201
MU-199
HIS-351
CS-319
CS-315

Set Operations: INTERSECT

- Find courses that ran in Fall 2017 and in Spring 2018
  - (**SELECT** *course_id* **FROM** *teaches* **WHERE** *semester* = 'Fall' **AND** *year* = 2017)
    INTERSECT
    (**SELECT** *course_id* **FROM** *teaches* **WHERE** *semester* = 'Spring' **AND** *year* = 2018)

  - *C.f.*, MySQL does NOT support INTERSECT
    - *One can emulate INTERSECT using JOIN (we'll study JOIN later)*
    - **SELECT** *LT.course_id*
      FROM (**SELECT** *course_id* **FROM** *teaches* **WHERE** *semester* = 'Fall' **AND** *year* = 2017)
      AS *LT*
      JOIN (**SELECT** *course_id* **FROM** *teaches* **WHERE** *semester* = 'Spring' **AND** *year* = 2018) **AS** *RT*
      ON *LT.course_id=RT.course_id*;

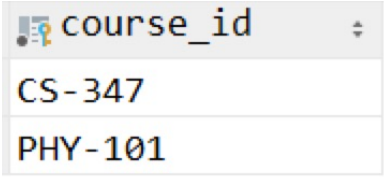


course_id
CS-101

Set Operations: EXCEPT

- Find courses that ran in Fall 2017 but not in Spring 2018
  - (**SELECT** *course_id* **FROM** *teaches* **WHERE** *semester* = 'Fall' **AND** *year* = 2017)
    EXCEPT
    (**SELECT** *course_id* **FROM** *teaches* **WHERE** *semester* = 'Spring' **AND** *year* = 2018)

  - *C.f.*, MySQL does NOT support EXCEPT
    - *One can emulate EXCEPT using NOT IN*
    - **SELECT** *course_id* **FROM** *teaches* **WHERE** *semester* = 'Fall' **AND** *year* = 2017
      AND *course_id* **NOT IN**
      (**SELECT** *course_id* **FROM** *teaches*
      WHERE *semester* = 'Spring' **AND** *year* = 2018);



course_id
CS-347
PHY-101

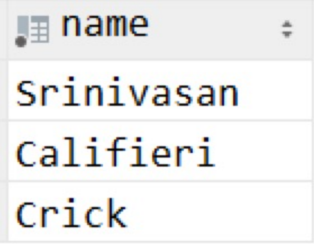
Agenda

- Structured query language (SQL)
- **SQL data manipulation language (DML)**
  - SELECT, FROM, WHERE
  - NULL values
  - Set operations
  - **String operations, ordering**
  - Aggregate functions, aggregation
- SQL data definition language (DDL)

String Operations

- SQL includes a string-matching operator for comparisons on character strings
- The operator **LIKE** uses patterns that are described using two special characters:
  - percent (%) – The % character matches any substring
  - underscore (_) – The _ character matches any character
- Find the names of all instructors whose name includes the substring “ri”

  SELECT *name*
  FROM *instructor*
  WHERE *name* **LIKE** ‘%ri%'



name
Srinivasan
Califieri
Crick

String Operations

- Escape character: Use backslash (\) as the escape character
  - *E.g.*, Match the string “100%”
    LIKE '100 \%’ **ESCAPE** ‘\’

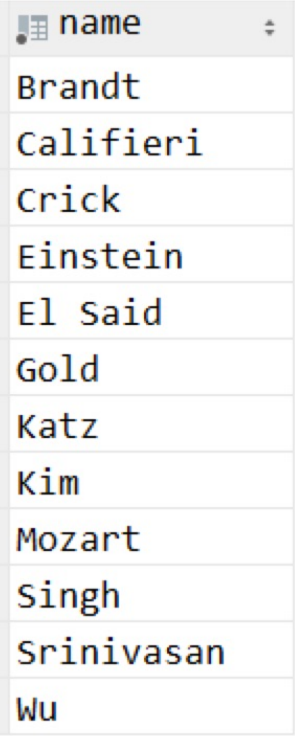
String Operations

- Patterns are case sensitive
- Pattern matching examples:
  - 'Intro%' matches any string beginning with “Intro”
  - '%Comp%' matches any string containing “Comp” as a substring
  - '_ _ _' matches any string of exactly three characters
  - '_ _ _ %' matches any string of at least three characters

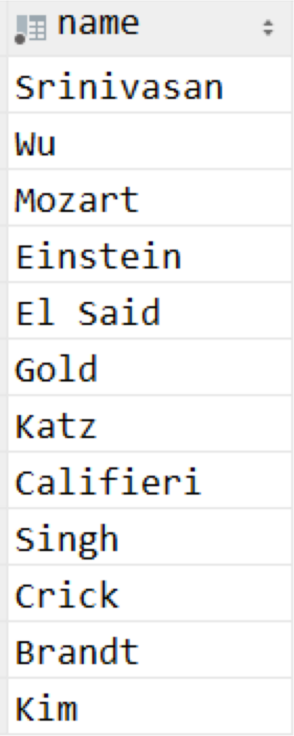
- SQL supports a variety of string operations such as
  - concatenation (using “||”)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

Ordering the Display of Tuples

- List in alphabetic order the names of all instructors
  - **SELECT DISTINCT** *name*
    FROM *instructor*
    ORDER BY *name*



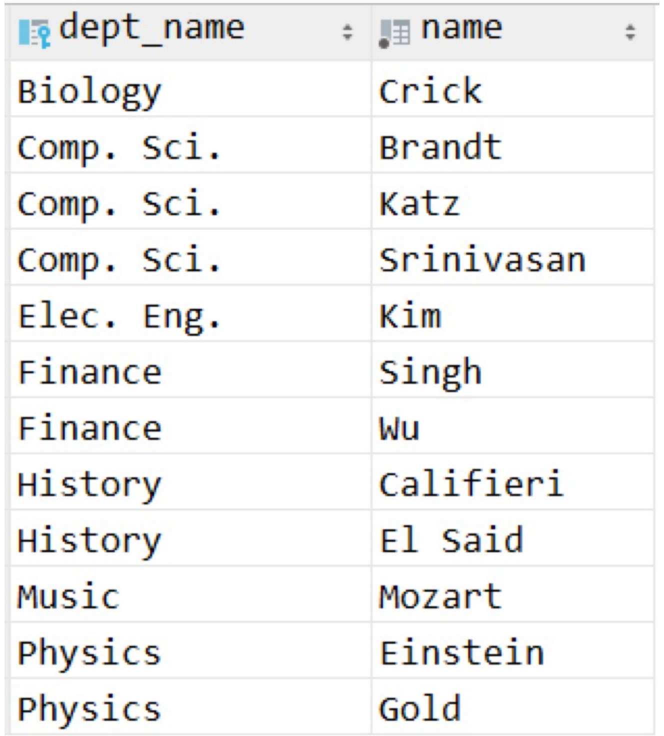
name
Brandt
Califieri
Crick
Einstein
El Said
Gold
Katz
Kim
Mozart
Singh
Srinivasan
Wu



name
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

Ordering the Display of Tuples

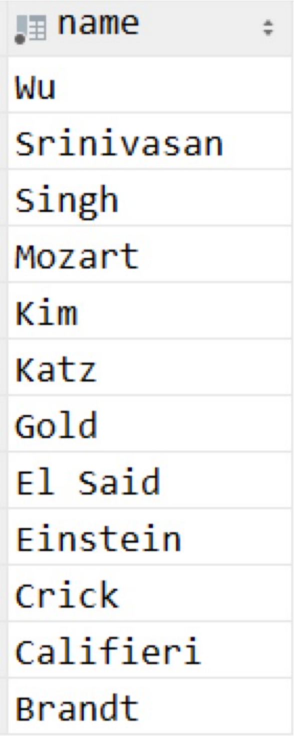
- Can sort on multiple attributes
  - *E.g.*, **SELECT** *dept_name, name*
    FROM *instructor*
    ORDER BY *dept_name, name*



dept_name	name
Biology	Crick
Comp. Sci.	Brandt
Comp. Sci.	Katz
Comp. Sci.	Srinivasan
Elec. Eng.	Kim
Finance	Singh
Finance	Wu
History	Califieri
History	El Said
Music	Mozart
Physics	Einstein
Physics	Gold

Ordering the Display of Tuples

- We may specify DESC for descending order or ASC for ascending order, for each attribute; *ascending order is the default*
  - *E.g.*, **ORDER BY** *name* **DESC**



name
Wu
Srinivasan
Singh
Mozart
Kim
Katz
Gold
El Said
Einstein
Crick
Califieri
Brandt

Agenda

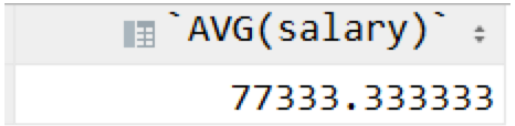
- Structured query language (SQL)
- **SQL data manipulation language (DML)**
  - SELECT, FROM, WHERE
  - NULL values
  - Set operations
  - String operations, ordering
  - **Aggregate functions, aggregation**
- SQL data definition language (DDL)

Aggregate Functions

- These functions operate on the multiset of values of a column of a relation, and return a value
  - **AVG:** average value
  - **MIN:** minimum value
  - **MAX:** maximum value
  - **SUM:** sum of values
  - **COUNT:** number of values

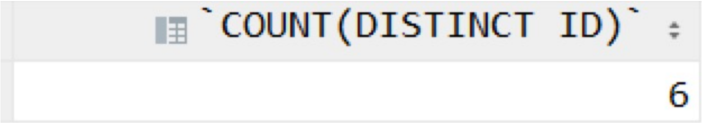
Aggregate Functions Examples

- Find the average salary of instructors in the Computer Science department
  - **SELECT AVG**(*salary*)
FROM *instructor*
WHERE *dept_name*= 'Comp. Sci.';



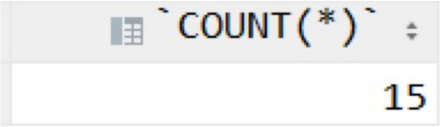
```
`AVG(salary)`
77333.333333
```

- Find the total number of instructors who teach a course in the Spring 2018 semester
  - **SELECT COUNT**(**DISTINCT** *ID*)
FROM *teaches*
WHERE *semester* = 'Spring' **AND** *year* = 2018;



```
`COUNT(DISTINCT ID)`
                   6
```

- Find the number of tuples in the *teaches* relation
  - **SELECT COUNT** (*)
FROM *teaches*;

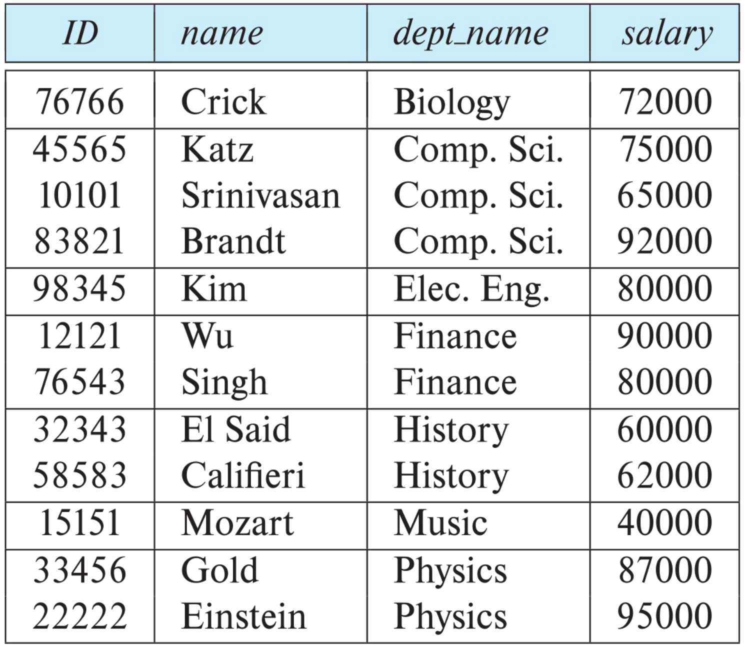


```
`COUNT(*)`
        15
```

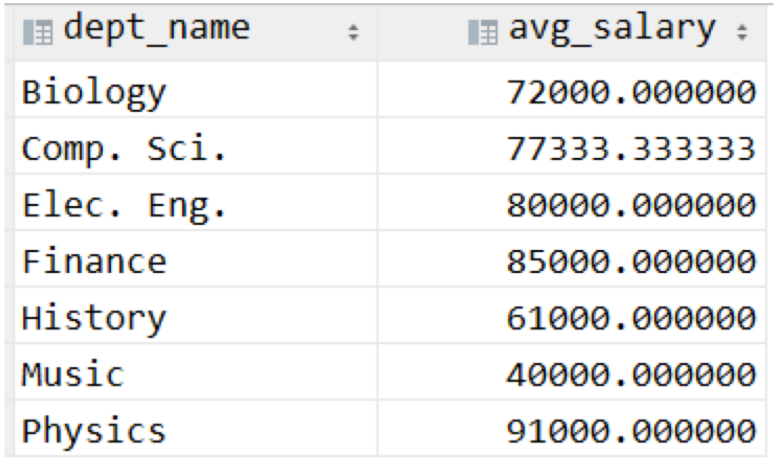
Aggregate Functions: Group By

- Find the average salary of instructors in each department

  -
    ```
    SELECT dept_name, AVG(salary) AS avg_salary
    FROM instructor
    GROUP BY dept_name;
    ```



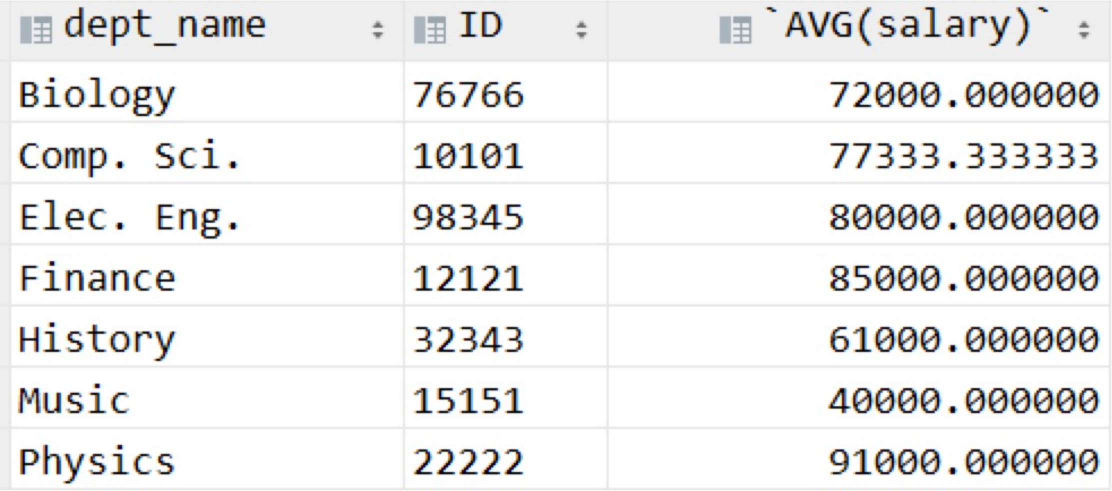
ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music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



dept_name	avg_salary
Biology	72000.000000
Comp. Sci.	77333.333333
Elec. Eng.	80000.000000
Finance	85000.000000
History	61000.000000
Music	40000.000000
Physics	91000.000000

Aggregation

- Attributes in **SELECT** clause outside of aggregate functions must appear in **GROUP BY** list
  - /* erroneous query */
    SELECT *dept_name*, *ID*, **AVG**(*salary*)
    FROM *instructor*
    GROUP BY *dept_name*;

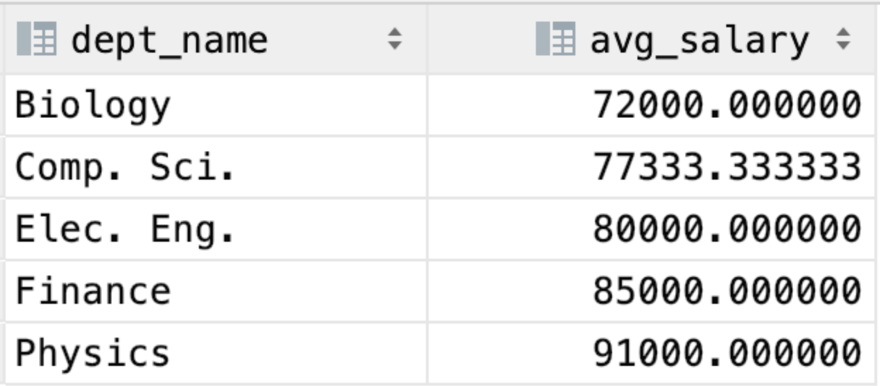


dept_name	ID	`AVG(salary)`
Biology	76766	72000.000000
Comp. Sci.	10101	77333.333333
Elec. Eng.	98345	80000.000000
Finance	12121	85000.000000
History	32343	61000.000000
Music	15151	40000.000000
Physics	22222	91000.000000

Aggregate Functions – Having Clause

- Find the names and average salaries of all departments whose average salary is greater than 65000

  -
    ```
    SELECT dept_name, AVG(salary) AS avg_salary
    FROM instructor
    GROUP BY dept_name
    HAVING AVG(salary) > 65000;
    ```



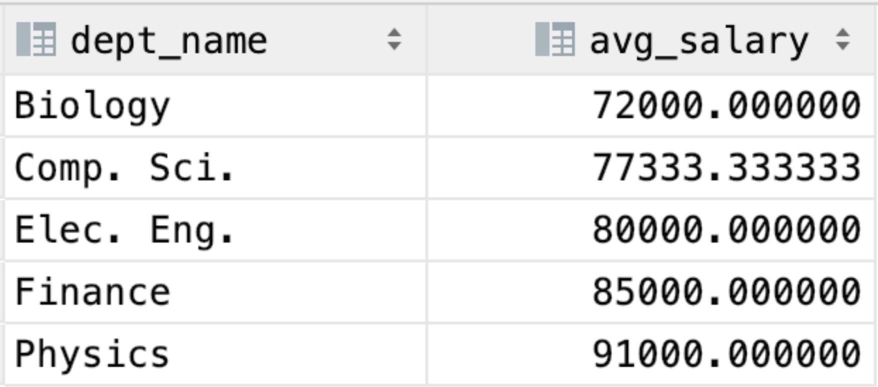
dept_name	avg_salary
Biology	72000.000000
Comp. Sci.	77333.333333
Elec. Eng.	80000.000000
Finance	85000.000000
Physics	91000.000000

Aggregate Functions – Having Clause

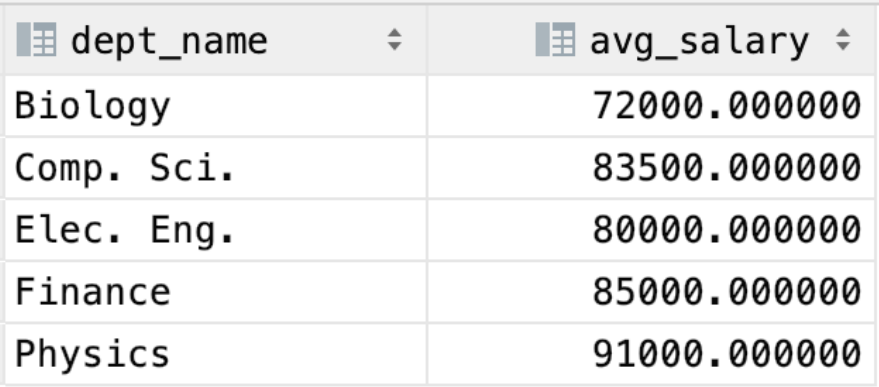
- Note: predicates in the **HAVING** clause are applied after the formation of groups whereas predicates in the **WHERE** clause are applied before forming groups

```
SELECT dept_name, AVG(salary) AS avg_salary
FROM instructor
GROUP BY dept_name
HAVING AVG(salary) > 65000;
```

```
SELECT dept_name, AVG(salary) AS avg_salary
FROM instructor
WHERE salary > 65000
GROUP BY dept_name;
```



dept_name	avg_salary
Biology	72000.000000
Comp. Sci.	77333.333333
Elec. Eng.	80000.000000
Finance	85000.000000
Physics	91000.000000



dept_name	avg_salary
Biology	72000.000000
Comp. Sci.	83500.000000
Elec. Eng.	80000.000000
Finance	85000.000000
Physics	91000.000000

SQL Order of Execution

Order	Clause	Function
1	FROM	Choose and join tables to get base data
2	WHERE	Filters the base data
3	GROUP BY	Aggregates the base data
4	HAVING	Filters the aggregated data
5	SELECT	Returns the final data
6	ORDER BY	Sorts the final data
7	LIMIT	Limits the returned data to a row count

EOF

- Coming next:
  - SQL data manipulation language (DML)
  - Nested subqueries
  - Set membership (SOME, ALL, EXISTS)