ECE30030/ITP30010 Database Systems

Structured Query Language

Reading: Chapter 2

Charmgil Hong

charmgil@handong.edu

Spring, 2023

Handong Global University

Announcement

- Homework assignment #1 is due this Friday
  - Due: By the end of Friday, March 24

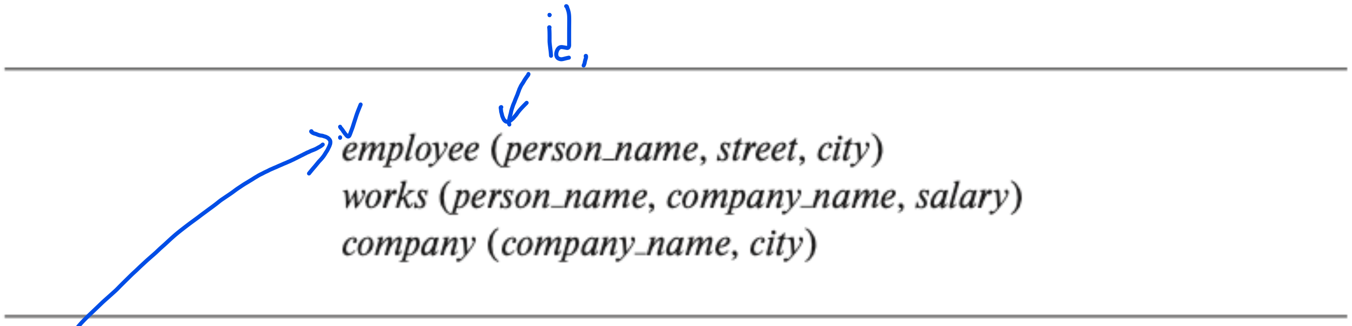
- A video presentation on *how to prepare a Docker environment* is available on LMS
  - TA’s have been working on all the materials!

Announcement

- The Class Docker Image is available
  - A recent version of the MySQL Server official image
  - Contains 8 example databases
    - You may exercise SQL with the example databases
      - You can reproduce the examples used in class on the ‘university_small’ database
    - Assignments using these example databases will be provided

Announcement

- For Problem #4 of Assignment #1, the textbook has a typo in Figure 2.17:



id,

√ *employee* (*person_name*, *street*, *city*)
works (*person_name*, *company_name*, *salary*)
company (*company_name*, *city*)

Figure 2.17 Employee database.

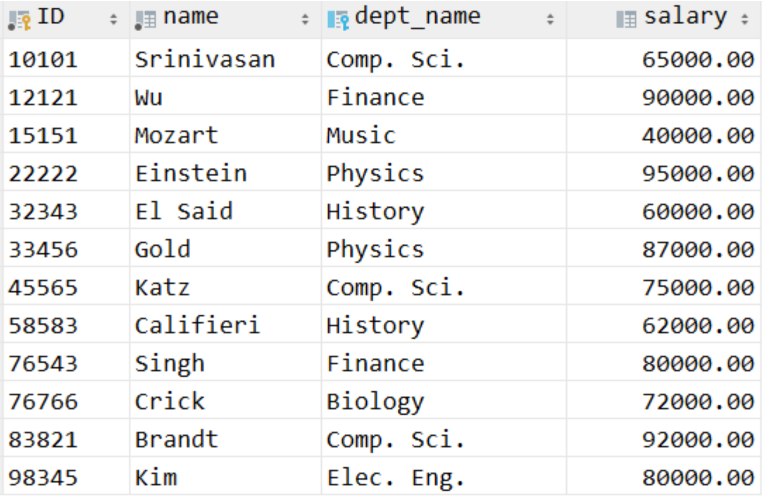
- You need to suppose the *employee* table has an additional column named *id*, such that:

employee (*id*, *person name*, *street*, *city*)

Example Problem

- Find the records of the instructor(s) who get(s) the largest salary
  - List the records of the instructor(s) who do not get less than someone else

Instructor relation

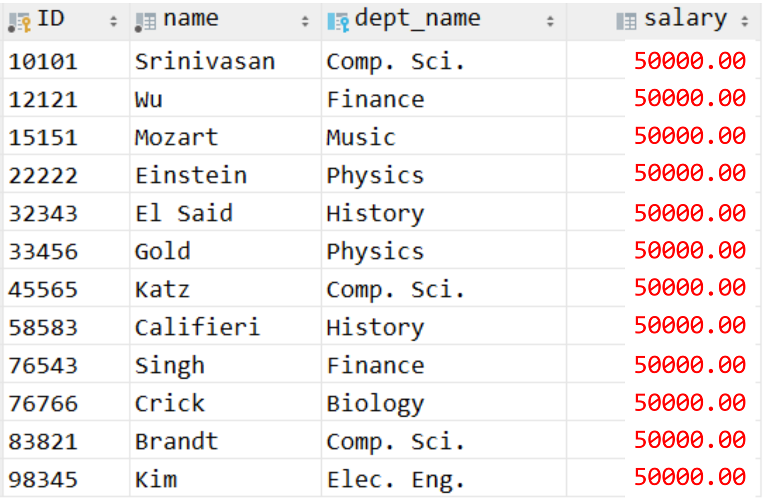


ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766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

Example Problem

- Find the records of the instructor(s) who get(s) the largest salary
  - List the records of the instructor(s) who do not get less than someone else
  - What if your data had uniform salary values?

Instructor relation



ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	50000.00
12121	Wu	Finance	50000.00
15151	Mozart	Music	50000.00
22222	Einstein	Physics	50000.00
32343	El Said	History	50000.00
33456	Gold	Physics	50000.00
45565	Katz	Comp. Sci.	50000.00
58583	Califieri	History	50000.00
76543	Singh	Finance	50000.00
76766	Crick	Biology	50000.00
83821	Brandt	Comp. Sci.	50000.00
98345	Kim	Elec. Eng.	50000.00

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) -- *NEXT CLASS*

Structured Query Language (SQL)

- SQL: Structured Query Language
  - The principal language used to describe and manipulate relational databases
  - Very high-level
    - Say "what to do" rather than "how to do it"
    - SQL is not specifying data-manipulation details
    - DBMSs figure out the "best" way to execute queries
      - Called "query optimization"

graph reduction.
AI.

  - Two aspects to SQL
    - Data definition: for declaring database schemas (DDL)
    - Data manipulation: for querying (asking questions about) databases and for modifying the database (DML)

SQL Parts

- DML – provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database
- DDL – includes commands for defining views
  - The DDL includes commands for specifying integrity constraints

a set of operations: bundle of operations.

ex) Salary, balance cannot be negative.

- Transaction control – includes commands for specifying the beginning and ending of transactions
- Authorization – includes commands for specifying access rights to relations and views

access permission.

- Embedded SQL and dynamic SQL – define how SQL statements can be embedded within general-purpose programming language

programming support Layer.

A Brief History

- IBM SEQUEL (Structured English Query Language) was developed as a part of the System R project (Chamberlin and Boyce, early 1970s)
  - Later on, SEQUEL was renamed SQL (structured query language)
  - System R → System/38 (1979), SQL/DS (1981), DB2 (1983)

- Relational Software, Inc released the first commercial implementation of SQL, Oracle V2 for VAX computers
  - Relational Software, Inc is now Oracle Corporation

- ANSI and ISO standardized SQL:
  - Ver : SQL-86, SQL-89, SQL-92, SQL:1999, …, SQL:2011, SQL:2016 (current)
  - SQL-92 is supported by the most of database systems !!

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)

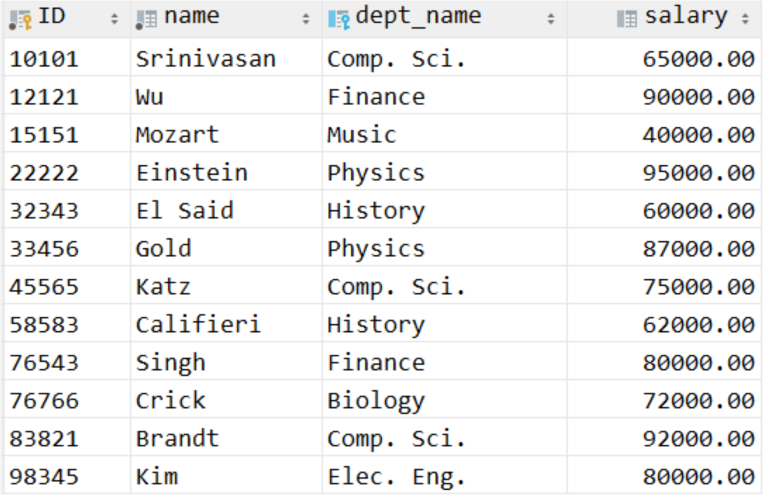
SQL Data Manipulation Language

- The SQL data-manipulation language (DML) allows querying (ask questions about) and modifying the databases

Running Examples

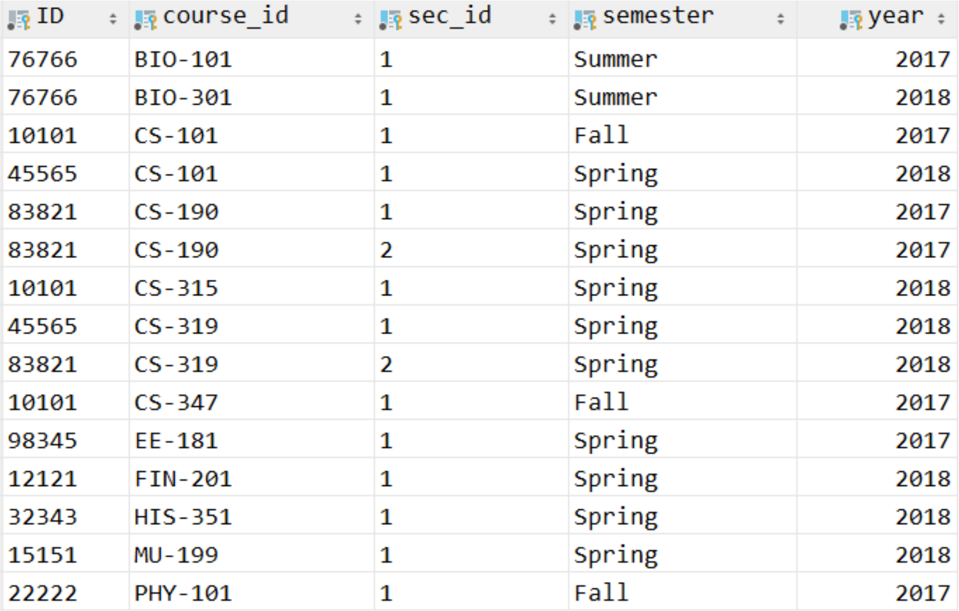
- Relations (tables): *instructor, teaches*

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

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 : table .
  - P is a predicate .

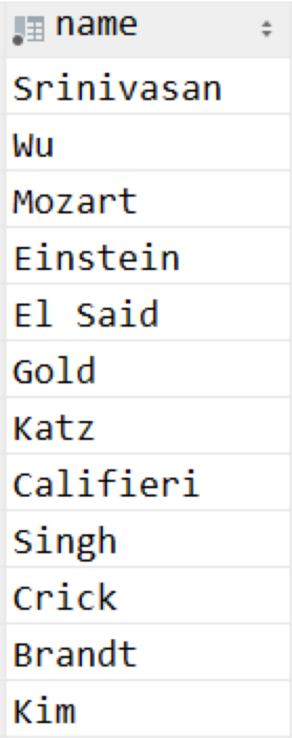
- The result of an SQL query is a relation : table.

SQL	Relational Algebra
Select	π
From	( )
Where	σ

The SELECT Clause

- 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
  - SQL: **SELECT** *name* **FROM** *instructor*;

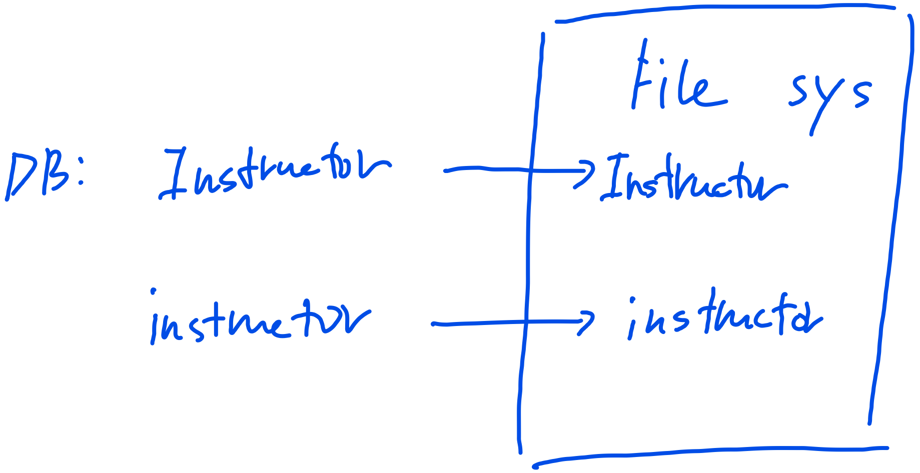


name
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

Note

- Note: SQL names are case insensitive
  - *E.g., Name ≡ NAME ≡ name*
  - SQL commands (SELECT, FROM, WHERE, …) are written in upper case (just a convention)

- MySQL has an option flag, `lower_case_table_names`
  - Link: https://dev.mysql.com/doc/refman/8.0/en/identifier-case-sensitivity.html

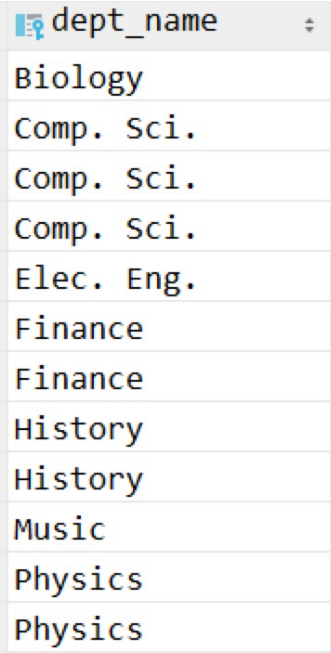


depends if the OS is case insensitive or not.

The SELECT Clause

- SQL allows duplicates in relations as well as in query results
  - The keyword ALL specifies that duplicates should not be removed

    SELECT ALL *dept_name*
    FROM *instructor*



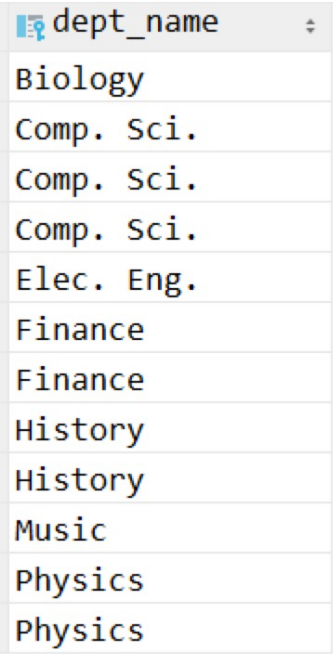
```
dept_name
Biology
Comp. Sci.
Comp. Sci.
Comp. Sci.
Elec. Eng.
Finance
Finance
History
History
Music
Physics
Physics
```

The SELECT Clause

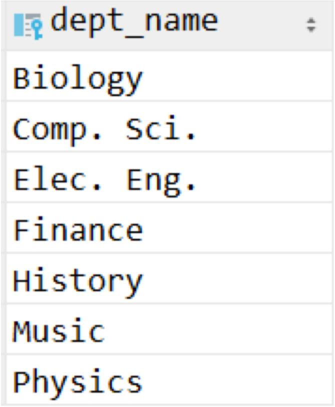
- SQL allows duplicates in relations as well as in query results
  - The keyword ALL specifies that duplicates should not be removed

    SELECT ALL *dept_name*
    FROM *instructor*
  - To force the elimination of duplicates, insert the keyword DISTINCT after SELECT
    - Find the department names of all instructor, removing duplicates:

      SELECT DISTINCT *dept_name*
      FROM *instructor*;



dept_name
Biology
Comp. Sci.
Comp. Sci.
Comp. Sci.
Elec. Eng.
Finance
Finance
History
History
Music
Physics
Physics



dept_name
Biology
Comp. Sci.
Elec. Eng.
Finance
History
Music
Physics

The SELECT Clause

- An asterisk in the select clause denotes “all attributes”

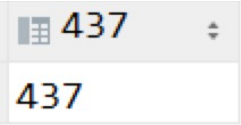
  SELECT * **FROM** *instructor*;

- An attribute can be a literal with no FROM clause

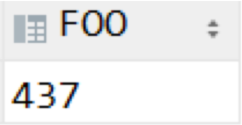
  SELECT ‘437’;

  - Result is a table with one column and a single row with value “437”
  - Can give the column a name using **AS**:

    SELECT ‘437’ **AS** FOO



437
437



FOO
437

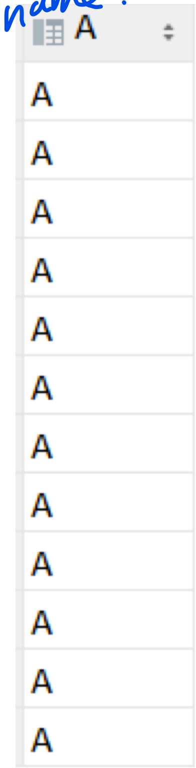
The SELECT Clause

- An attribute can be a literal with FROM clause

  SELECT 'A' **FROM** *instructor*

  - Result is a table with one column and *N* rows (number of tuples in the *instructor* table), each row with value “A”

'A' as a column name.



A
A
A
A
A
A
A
A
A
A
A
A
A

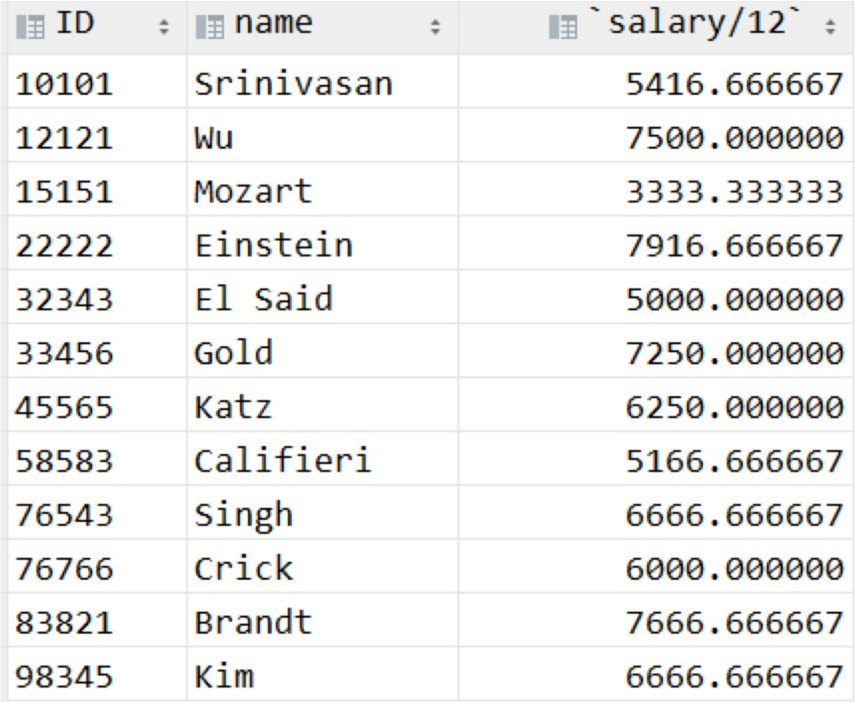
The SELECT Clause

- The SELECT clause can contain arithmetic expressions involving the operation, +, –, ∗, and /, and operating on constants or attributes of tuples
  - The query: **SELECT** *ID, name, salary/12*
    FROM *instructor*

    would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12

col name is also changed.

Round (salary/12) as montlysalary.



ID	name	\`salary/12\`
10101	Srinivasan	5416.666667
12121	Wu	7500.000000
15151	Mozart	3333.333333
22222	Einstein	7916.666667
32343	El Said	5000.000000
33456	Gold	7250.000000
45565	Katz	6250.000000
58583	Califieri	5166.666667
76543	Singh	6666.666667
76766	Crick	6000.000000
83821	Brandt	7666.666667
98345	Kim	6666.666667

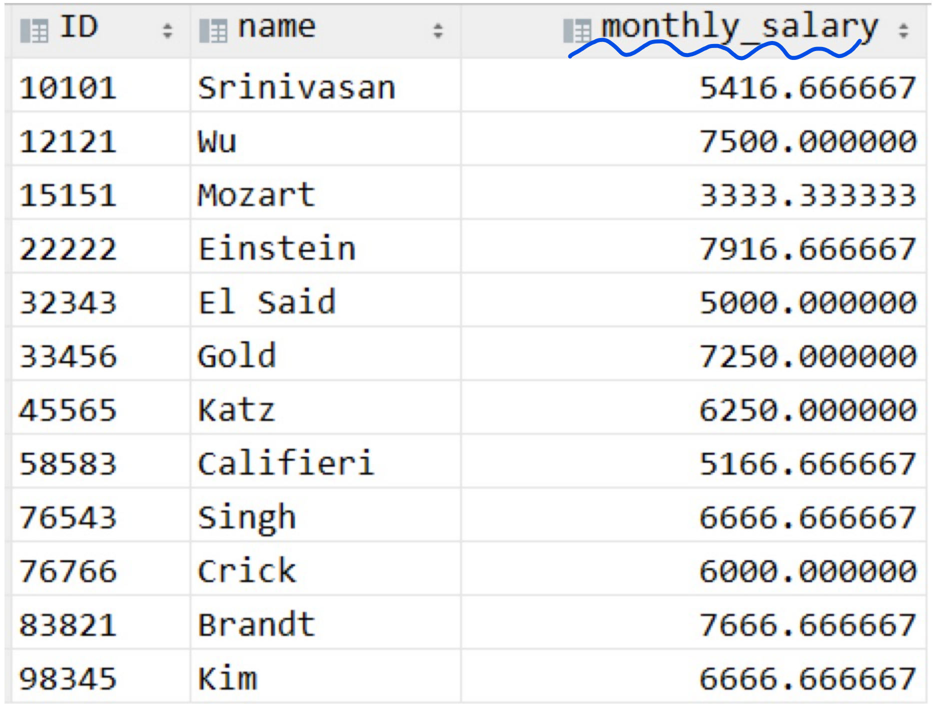
montlysalaly

5417
7500
3333
⋮
6667

The SELECT Clause

- The SELECT clause can contain arithmetic expressions involving the operation, +, –, ∗, and /, and operating on constants or attributes of tuples
  - Can rename "*salary/12*" using the **AS** clause:

    SELECT *ID, name, salary/12* **AS** *monthly_salary*
    FROM *instructor*



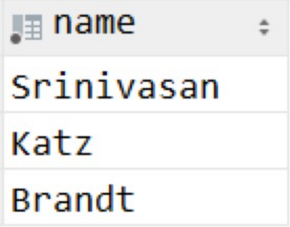
ID	name	monthly_salary
10101	Srinivasan	5416.666667
12121	Wu	7500.000000
15151	Mozart	3333.333333
22222	Einstein	7916.666667
32343	El Said	5000.000000
33456	Gold	7250.000000
45565	Katz	6250.000000
58583	Califieri	5166.666667
76543	Singh	6666.666667
76766	Crick	6000.000000
83821	Brandt	7666.666667
98345	Kim	6666.666667

The WHERE Clause

- The WHERE clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra

- *E.g.*, To find all *instructors* in Comp. Sci. dept:

```
SELECT name FROM instructor
WHERE dept_name = 'Comp. Sci.';
```



name
Srinivasan
Katz
Brandt

The WHERE Clause

- SQL allows the use of the logical connectives **AND**, **OR**, and **NOT**
- The operands of the logical connectives can be expressions involving the comparison operators <, <=, >, >=, =, and <>
  - <> means not equal (there is no != in SQL)

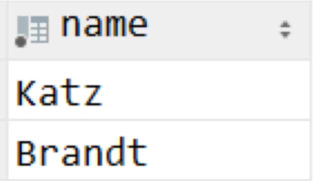
not ==

same as !=

- Comparisons can be applied to results of arithmetic expressions

- *E.g.*, To find all *instructors* in Comp. Sci. with *salary* > 70,000:

  SELECT *name* **FROM** *instructor*
  WHERE *dept_name* = 'Comp. Sci.' **AND** *salary* > 70000;

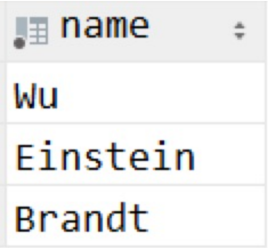


name
Katz
Brandt

The WHERE Clause

- SQL includes a **BETWEEN** comparison operator ,, wow!!
- Example: Find the names of all instructors with salary between $90,000 and $100,000 (that is, ≥ $90,000 and ≤ $100,000)
  - **SELECT** *name*
    FROM *instructor*
    WHERE *salary* **BETWEEN** 90000 **AND** 100000

Inclusive.



name
Wu
Einstein
Brandt

The WHERE Clause

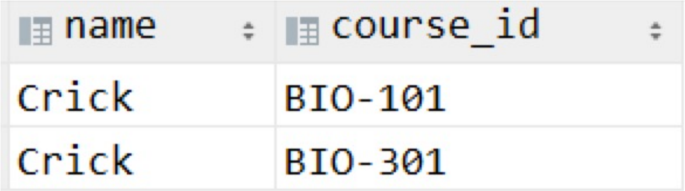
- Tuple comparison: makes comparisons per tuple
  - **SELECT** *name, course_id*
    FROM *instructor, teaches*
    WHERE (*instructor.ID, dept_name*) = (*teaches.ID*, 'Biology');

literal.

기본적으로 table 이름을 앞에 붙여줄 수 있다.

instructor name from Instructor

from teaches.



name	course_id
Crick	BIO-101
Crick	BIO-301

The FROM Clause

- The FROM clause lists the relations involved in the query
  - Corresponds to the Cartesian-product operation of the relational algebra

- Find the Cartesian-product *instructor* × *teaches*

  SELECT * **FROM** *instructor, teaches*;

  - Generates every possible instructor-teaches pairs, with all attributes from both relations
  - For common attributes (*e.g.*, *ID*), the attributes in the resulting table are renamed using the relation name (*e.g.*, *instructor.ID*)

The FROM Clause

- Find the Cartesian-product *instructor X teaches*

  SELECT * FROM *instructor, teaches*;

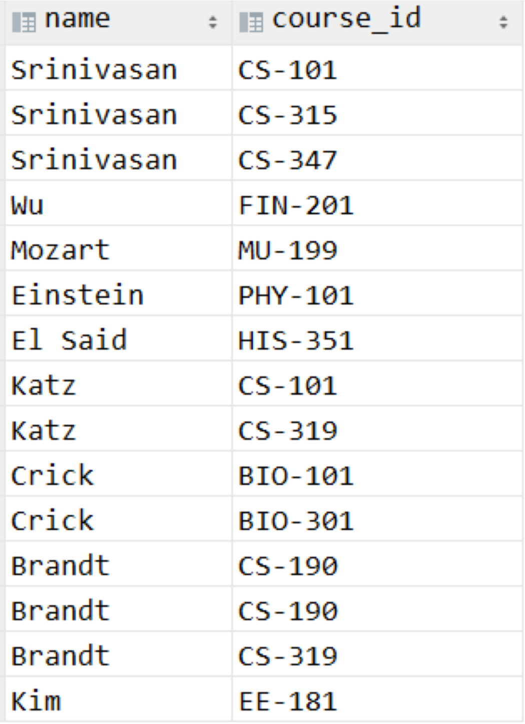
instructor.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	76766	BIO-101	1	Summer	2017
12121	Wu	Finance	90000	76766	BIO-101	1	Summer	2017
15151	Mozart	Music	40000	76766	BIO-101	1	Summer	2017
22222	Einstein	Physics	95000	76766	BIO-101	1	Summer	2017
32343	El Said	History	60000	76766	BIO-101	1	Summer	2017
…	…	…	…	…	…	…	…	…
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
32343	El Said	History	60000	10101	CS-101	1	Fall	2017
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
10101	Srinivasan	Comp. Sci.	65000	83821	CS-190	2	Spring	2017
12121	Wu	Finance	90000	83821	CS-190	2	Spring	2017
15151	Mozart	Music	40000	83821	CS-190	2	Spring	2017
…	…	…	…	…	…	…	…	…
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…

Implementing JOIN

- Cartesian-product is not very useful directly; but useful combined with WHERE-clause condition (selection operation in relational algebra)
  - Cartesian-product + selection = join

  - *E.g.*, Find the names of all instructors who have taught some course and the *course_id*

```
SELECT name, course_id
FROM instructor , teaches
WHERE instructor.ID = teaches.ID
```



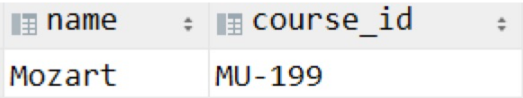
name	course_id
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

Implementing JOIN

- Cartesian-product is not very useful directly; but useful combined with WHERE-clause condition (selection operation in relational algebra)
  - Cartesian-product + selection = join

  - Find the names of all instructors in the Music department who have taught some course and the *course_id*

```
SELECT name, course_id
FROM instructor , teaches
WHERE instructor.ID = teaches.ID
          AND  instructor. dept_name = 'Music'
```



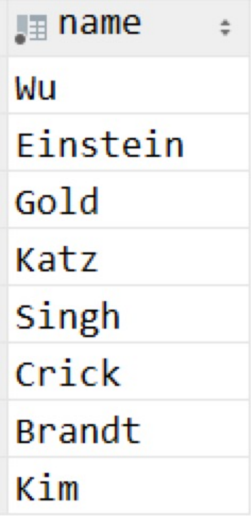
name	course_id
Mozart	MU-199

The Rename Operation

- The SQL allows renaming relations and attributes using the **AS** clause:

  old-name **AS** *new-name*

- Find the names of all instructors who have a higher salary than some instructor in ‘Comp. Sci.’
  - **SELECT DISTINCT** *T.name*
    FROM *instructor* **AS** *T, instructor* **AS** *S*
    WHERE *T.salary > S.salary* **AND** *S.dept_name = 'Comp. Sci.'*



name
Wu
Einstein
Gold
Katz
Singh
Crick
Brandt
Kim

The Rename Operation

- The SQL allows renaming relations and attributes using the **AS** clause:

  old-name **AS** *new-name*

- Find the names of all instructors who have a higher salary than some instructor in ‘Comp. Sci.’
  - **SELECT DISTINCT** *T.name*
    FROM *instructor* **AS** *T, instructor* **AS** *S*
    WHERE *T.salary > S.salary* **AND** *S.dept_name = 'Comp. Sci.'*

- Keyword **AS** is optional and may be omitted

  instructor **AS** *T* ≡ *instructor T*

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

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

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

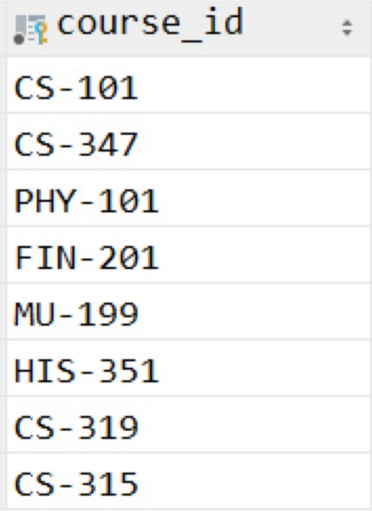
- Set operations **UNION, INTERSECT,** and **EXCEPT**
  - Each of the above operations automatically eliminates duplicates

- 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)

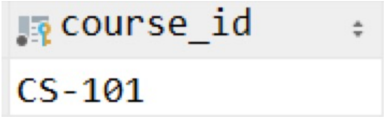


course_id
CS-101
CS-347
PHY-101
FIN-201
MU-199
HIS-351
CS-319
CS-315

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*;

LT JOIN RT on LT.course_id = RT.course_id

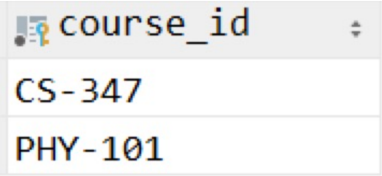


course_id
CS-101

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

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