ECE30030/ITP30010 Database Systems

Structured Query Language

Reading: Chapter 3

Charmgil Hong

charmgil@handong.edu

Spring, 2025

Handong Global University

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”
  - 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
- Integrity – the DDL includes commands for specifying integrity constraints
- View definition – the DDL includes commands for defining views
- Transaction control – includes commands for specifying the beginning and ending of transactions
- Embedded SQL and dynamic SQL – define how SQL statements can be embedded within general-purpose programming language
- Authorization – includes commands for specifying access rights to relations and views

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:
  - SQL-86, SQL-89, SQL-92, SQL:1999, …, SQL:2011, SQL:2016 (current)
  - SQL-92 is supported by the most of database systems

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

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)

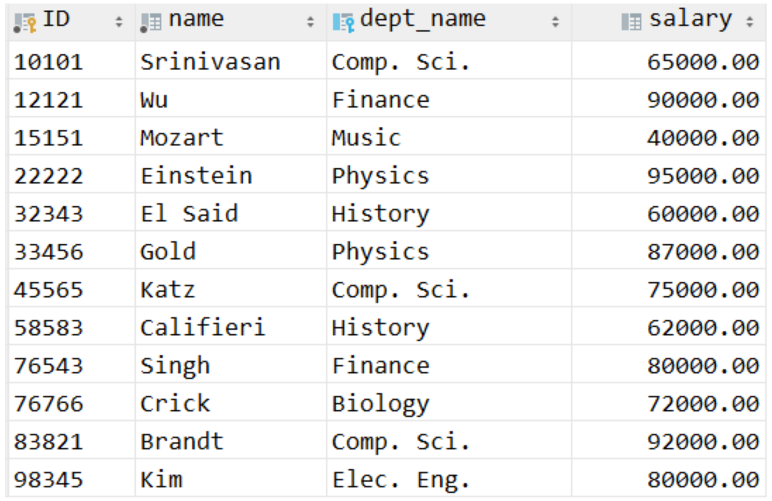
SQL Data Manipulation Language

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

Running Examples

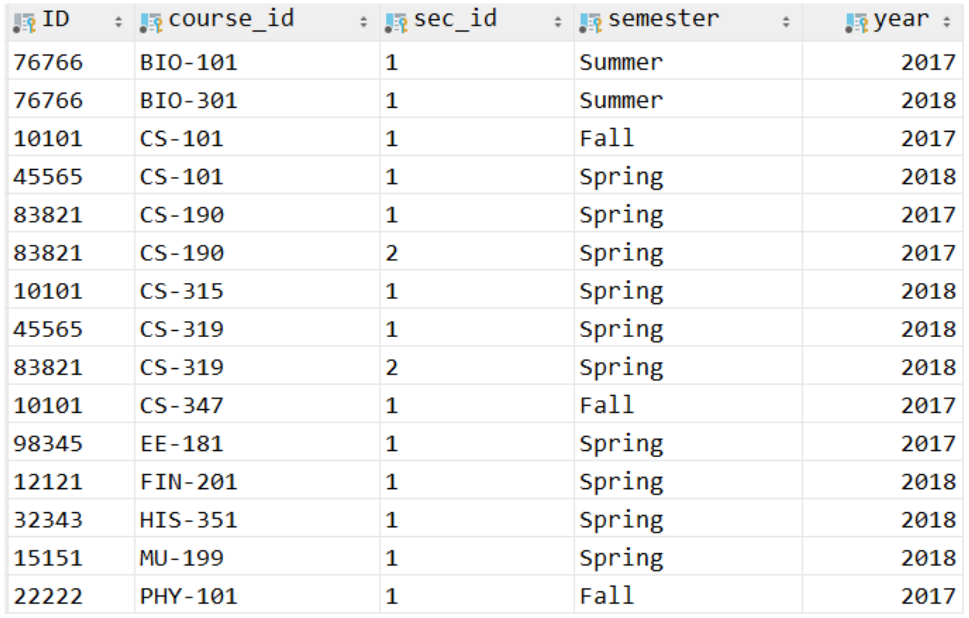
- Relations (tables): *instructor, teaches*

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

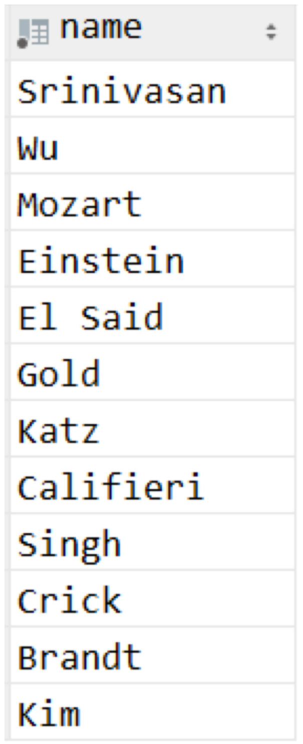
  - A_i represents an attribute
  - R_i represents a relation
  - P is a predicate

- The result of an SQL query is a relation

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

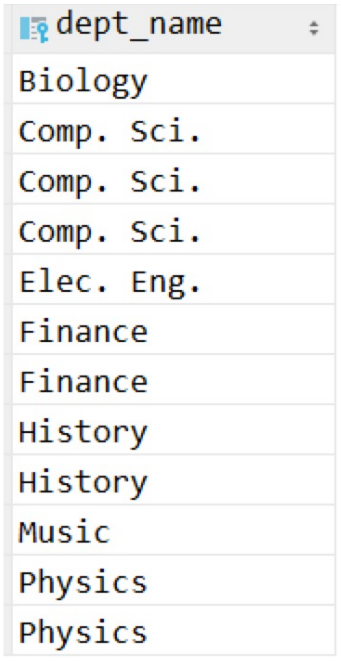
- Note: SQL names are case insensitive
  - *E.g., Name ≡ NAME ≡ name*
  - SQL commands (SELECT, FROM, WHERE, …) are written in upper case (just a convention)

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

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*



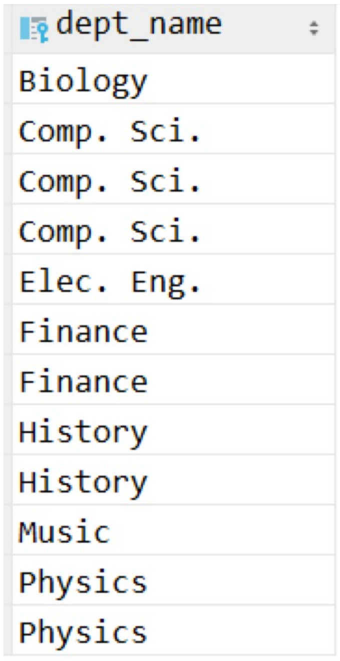
dept_name
Biology
Comp. Sci.
Comp. Sci.
Comp. Sci.
Elec. Eng.
Finance
Finance
History
History
Music
Physics
Physics

The SELECT Clause

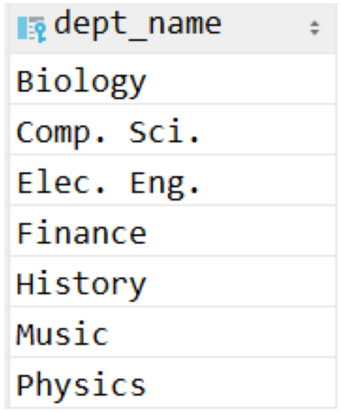
- SQL allows duplicates in relations as well as in query results
  - The keyword ALL specifies that duplicates should not be removed

    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

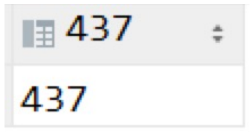


dept_name
Biology
Comp. Sci.
Elec. Eng.
Finance
History
Music
Physics

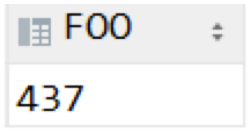
The SELECT Clause

- 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



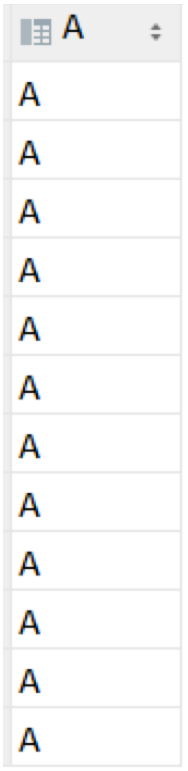
FOO
437

The SELECT Clause

- An attribute can be a literal with FROM clause

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

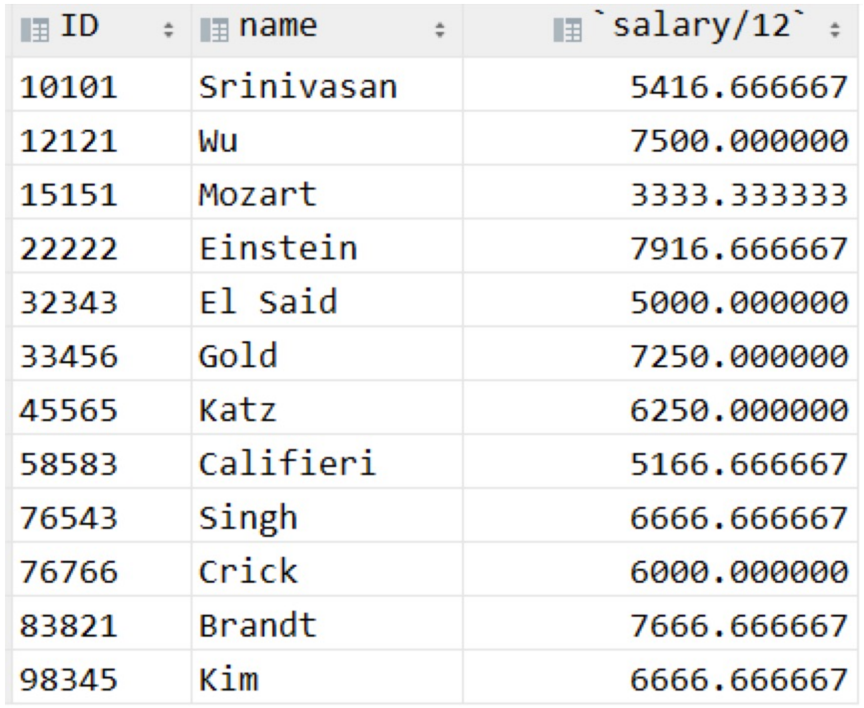
  - Result is a table with one column and *N* rows (number of tuples in the *instructor* table), each row with value “A”



A
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

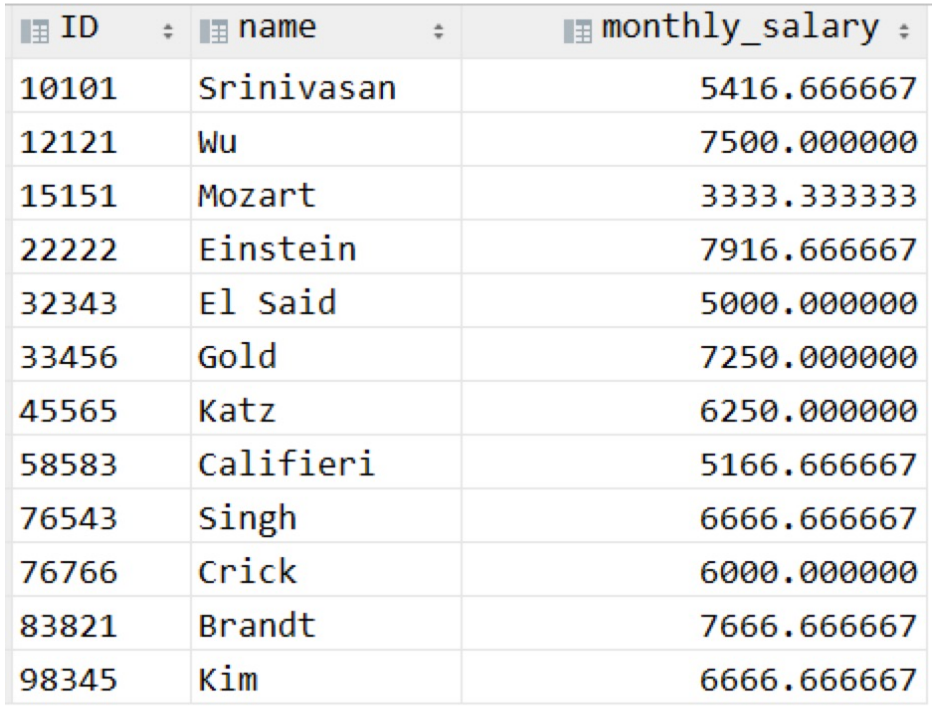


ID	name	\`salary/12\`
10101	Srinivasan	5416.666667
12121	Wu	7500.000000
15151	Mozart	3333.333333
22222	Einstein	7916.666667
32343	El Said	5000.000000
33456	Gold	7250.000000
45565	Katz	6250.000000
58583	Califieri	5166.666667
76543	Singh	6666.666667
76766	Crick	6000.000000
83821	Brandt	7666.666667
98345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

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*

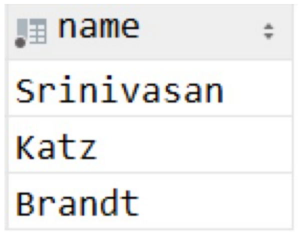


ID	name	monthly_salary
10101	Srinivasan	5416.666667
12121	Wu	7500.000000
15151	Mozart	3333.333333
22222	Einstein	7916.666667
32343	El Said	5000.000000
33456	Gold	7250.000000
45565	Katz	6250.000000
58583	Califieri	5166.666667
76543	Singh	6666.666667
76766	Crick	6000.000000
83821	Brandt	7666.666667
98345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



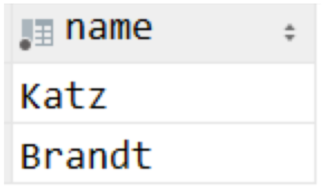
name
Srinivasan
Katz
Brandt

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)
- Comparisons can be applied to results of arithmetic expressions

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

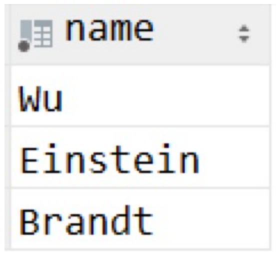
  SELECT *name* **FROM** *instructor*
  WHERE *dept_name* = 'Comp. Sci.' **AND** *salary* > 70000;



name
Katz
Brandt

The WHERE Clause

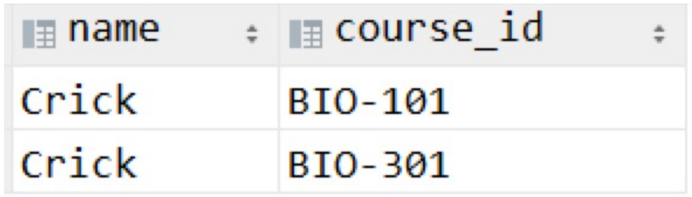
- SQL includes a **BETWEEN** comparison operator
- Example: Find the names of all instructors with salary between \$90,000 and \$100,000 (that is, ≥ \$90,000 and ≤ \$100,000)
  - **SELECT** *name*
    FROM *instructor*
    WHERE *salary* **BETWEEN** 90000 **AND** 100000



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');



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*;

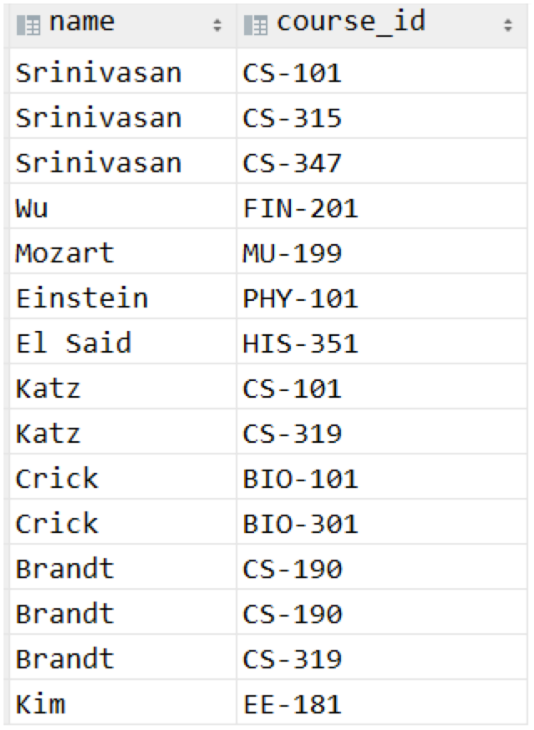
instructor.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	76766	BIO-101	1	Summer	2017
12121	Wu	Finance	90000	76766	BIO-101	1	Summer	2017
15151	Mozart	Music	40000	76766	BIO-101	1	Summer	2017
22222	Einstein	Physics	95000	76766	BIO-101	1	Summer	2017
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*

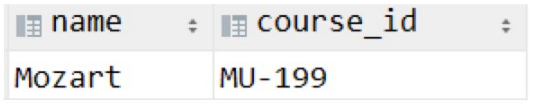
    SELECT *name, course_id*
    FROM *instructor , teaches*
    WHERE *instructor.ID = teaches.ID*



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'



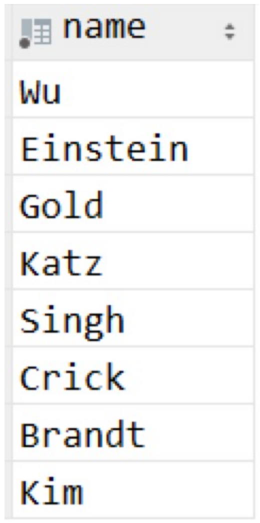
name	course_id
Mozart	MU-199

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

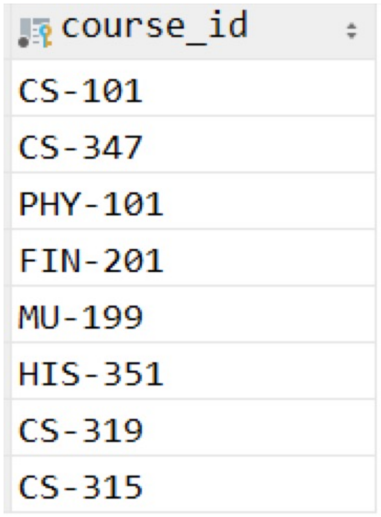
- Set operations **UNION, INTERSECT,** and **EXCEPT**
  - Each of the above operations automatically eliminates duplicates

- 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)

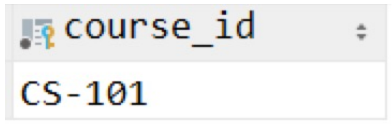


course_id
CS-101
CS-347
PHY-101
FIN-201
MU-199
HIS-351
CS-319
CS-315

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*;

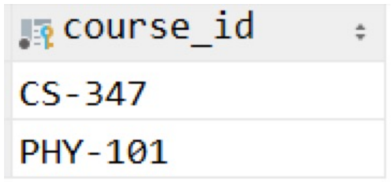


course_id
CS-101

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

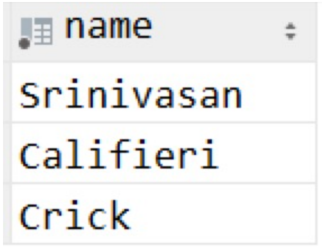
- Structured query language (SQL)
- **SQL data manipulation language (DML)**
  - SELECT, FROM, WHERE
  - NULL values
  - Set operations
  - **String operations, ordering**
  - Aggregate functions, aggregation
- SQL data definition language (DDL)

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** ‘\’

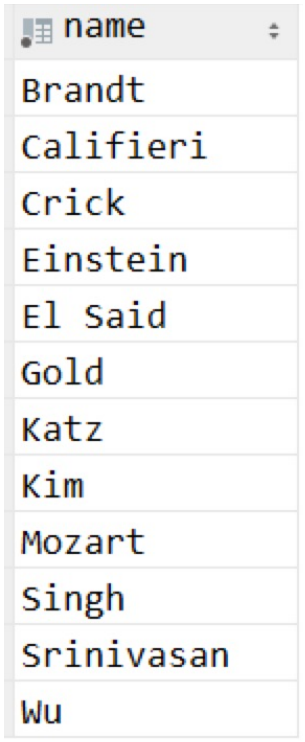
String Operations

- 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

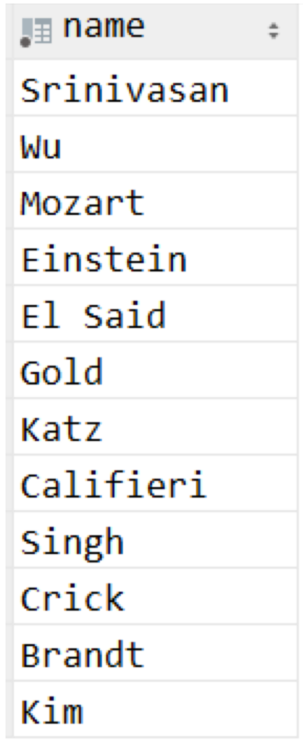
- SQL supports a variety of string operations such as
  - concatenation (using “||”)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

Ordering the Display of Tuples

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



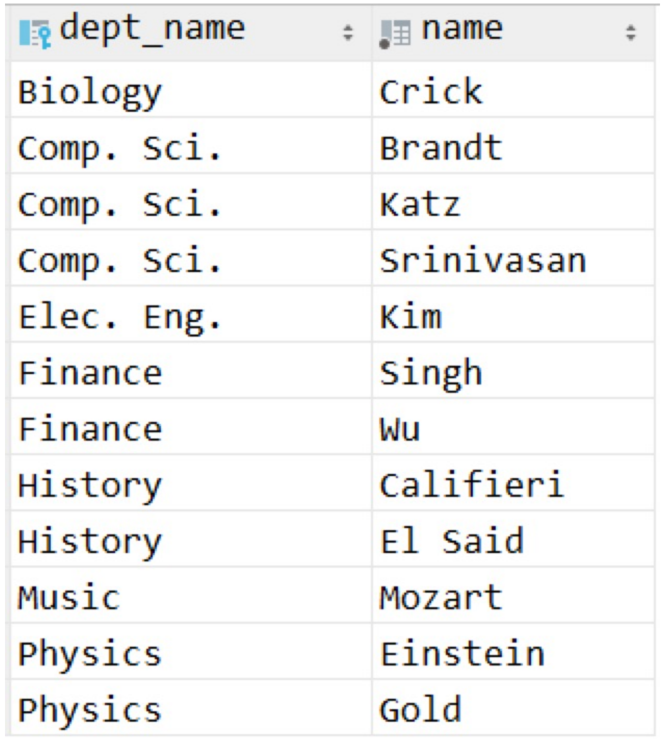
name
Brandt
Califieri
Crick
Einstein
El Said
Gold
Katz
Kim
Mozart
Singh
Srinivasan
Wu



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

Ordering the Display of Tuples

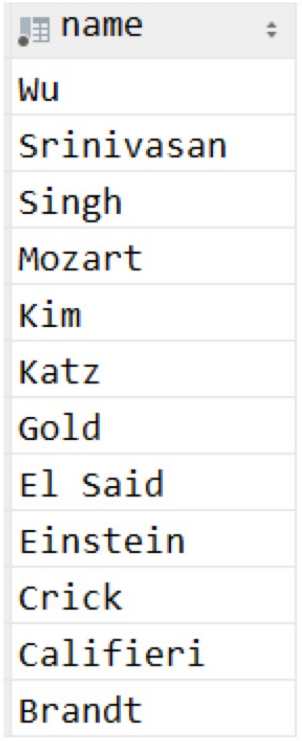
- Can sort on multiple attributes
  - *E.g.,* **SELECT** *dept_name, name*
    FROM *instructor*
    ORDER BY *dept_name, name*



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

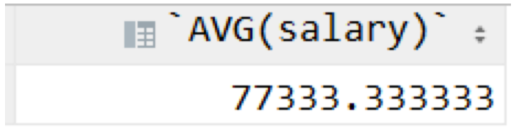
- Structured query language (SQL)
- **SQL data manipulation language (DML)**
  - SELECT, FROM, WHERE
  - NULL values
  - Set operations
  - String operations, ordering
  - **Aggregate functions, aggregation**
- SQL data definition language (DDL)

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

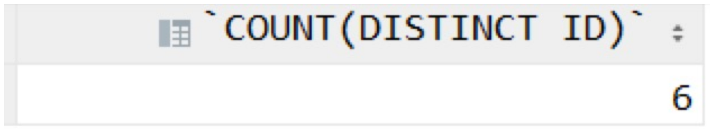
Aggregate Functions Examples

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



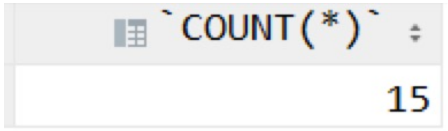
```
`AVG(salary)`
77333.333333
```

- 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
```

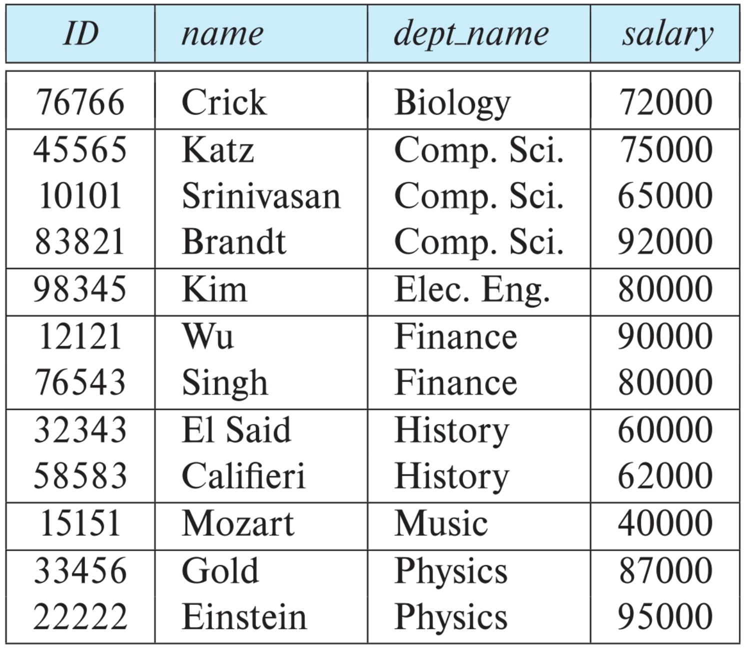
- Find the number of tuples in the *teaches* relation
  - **SELECT COUNT** (*)
    FROM *teaches*;



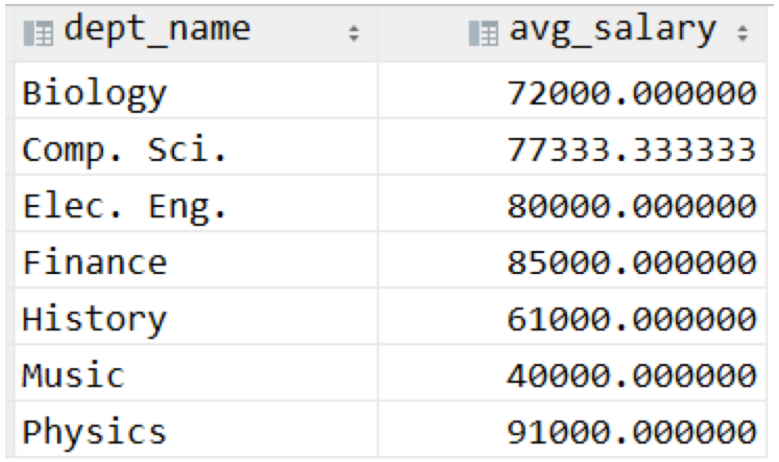
```
`COUNT(*)`
15
```

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*;



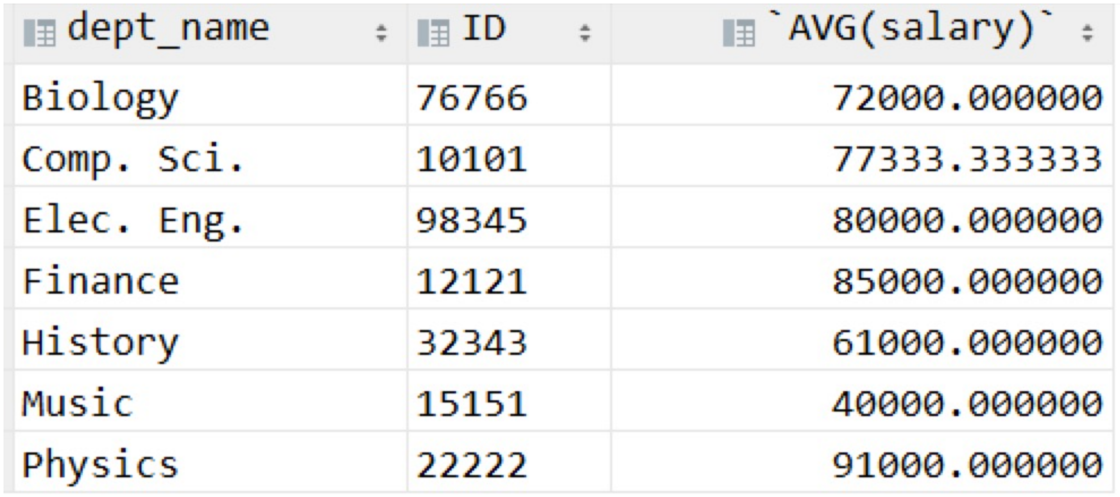
ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music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

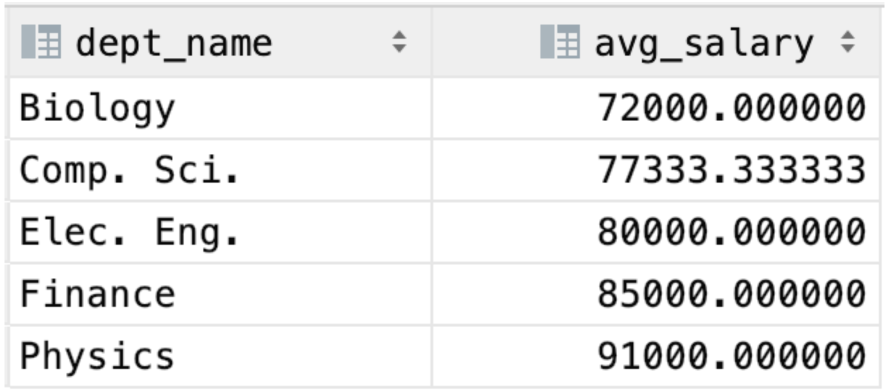
- Attributes in **SELECT** clause outside of aggregate functions must appear in **GROUP BY** list
  - /* erroneous query */
    SELECT *dept_name*, *ID*, **AVG**(*salary*)
    FROM *instructor*
    GROUP BY *dept_name*;



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;



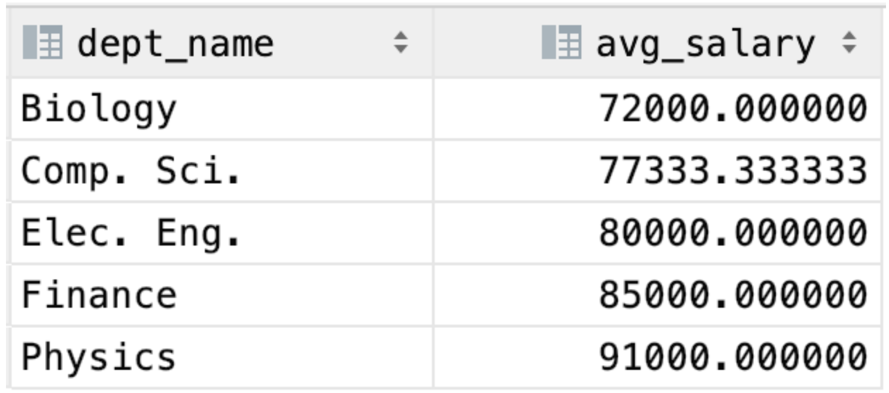
dept_name	avg_salary
Biology	72000.000000
Comp. Sci.	77333.333333
Elec. Eng.	80000.000000
Finance	85000.000000
Physics	91000.000000

Aggregate Functions – Having Clause

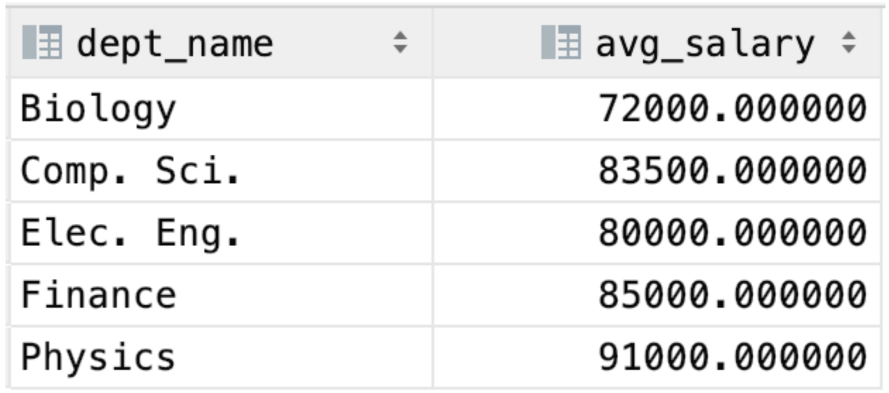
- Note: predicates in the **HAVING** clause are applied after the formation of groups whereas predicates in the **WHERE** clause are applied before forming groups

```
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;
```

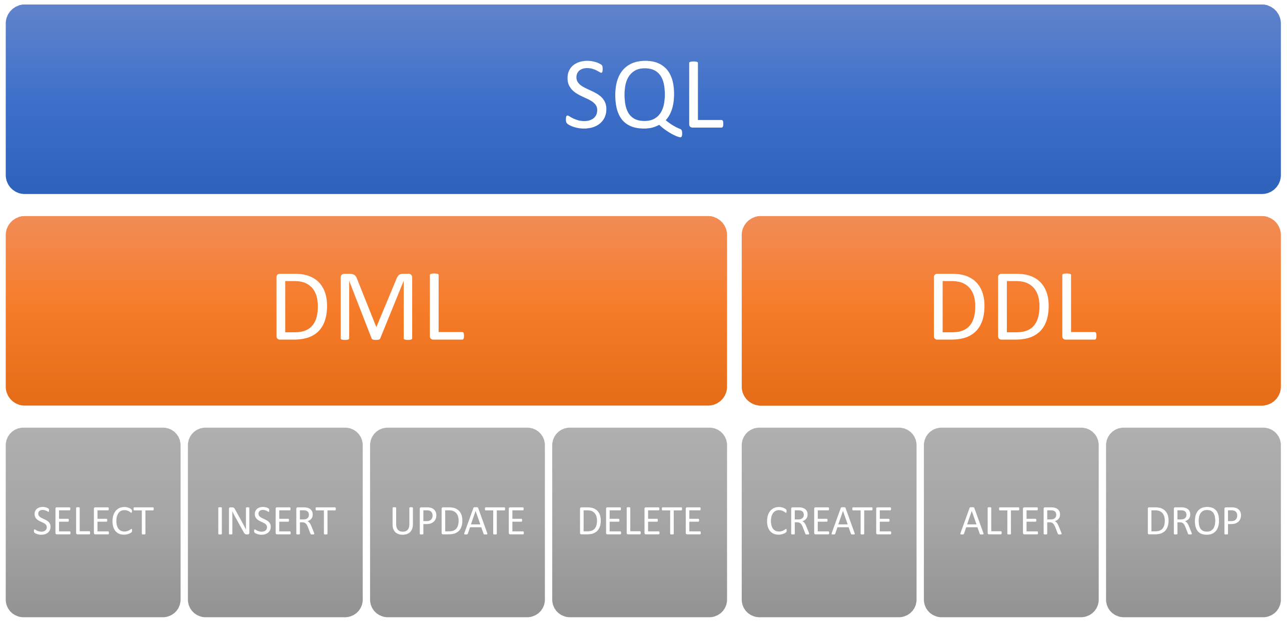


dept_name	avg_salary
Biology	72000.000000
Comp. Sci.	77333.333333
Elec. Eng.	80000.000000
Finance	85000.000000
Physics	91000.000000



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

SQL Commands



INSERT

- Basic syntax
  - Insert data into every column:
    - **INSERT INTO** *tablename*
      VALUES (*col1_value, col2_value, …*)
    - Must list values in the same order as in the table schema
    - If some data values are unknown, must type NULL
    - For character sequences, use quotation marks
      - Single quotation marks are preferred (but double quotation marks are allowed)
      - Value in quotations is case-sensitive
  - Insert data into selected columns
    - **INSERT INTO** *tablename* (*col1_name, col3_name, col4_name, …*)
      VALUES (*col1_value, col3_value, col4_value, …*)

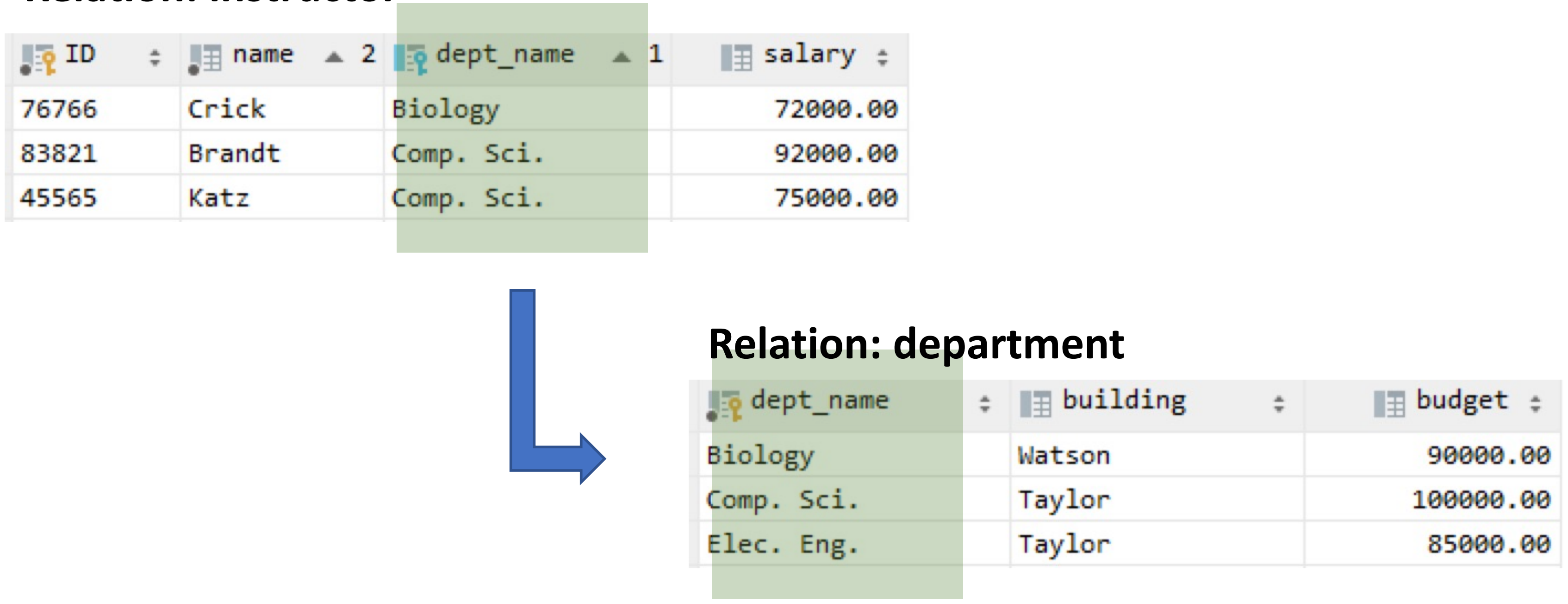
INSERT

- Add a new tuple to *course*
  - **INSERT INTO** *course*
    VALUES ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
- or equivalently
  - **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*);

INSERT

- A foreign key specifies that an attribute from one relation has to map to a tuple in another relation
  - Value in one relation must appear in another relation

Relation: instructor



ID	name	dept_name	salary
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
45565	Katz	Comp. Sci.	75000.00

Relation: department

dept_name	building	budget
Biology	Watson	90000.00
Comp. Sci.	Taylor	100000.00
Elec. Eng.	Taylor	85000.00

INSERT

- A foreign key specifies that an attribute from one relation has to map to a tuple in another relation
  - Value in one relation must appear in another relation

- Make sure all foreign keys that new row references have already been added to database
  - One cannot insert a foreign key value unless the corresponding value exists in the referenced relation

INSERT

- Inserting results of other SELECT query
    - 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
        - Otherwise queries like
          INSERT INTO *table*1 **SELECT** * **FROM** *table*1
          would cause problem

UPDATE

- Basic syntax
  - Updating a table
    - **UPDATE** *tablename*
      SET *col1_name = new_col1_value, col2_name = new_col2_value, …*;
  - Updating a table with conditions
    - **UPDATE** *tablename*
      SET *col1_name = new_col1_value, col2_name = new_col2_value, …*
      WHERE *predicate*;

UPDATE

- 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*);

UPDATE

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Write two UPDATE statements:

    UPDATE *instructor*
    SET *salary* = *salary* * 1.03
    WHERE *salary* > 100000;

    UPDATE *instructor*
    SET *salary* = *salary* * 1.05
    WHERE *salary* <= 100000;

  - The order is important
  - Can be done better using the **case** statement (next slide)

CASE Statement for Conditional Update

- The following query is equivalent to the previous UPDATE queries
  - **UPDATE** *instructor*
    SET *salary* = **CASE**
    WHEN *salary* <= 100000 **THEN** *salary* * 1.05
    ELSE *salary* * 1.03
    END

UPDATE with Scalar Subqueries

- Recompute and update *tot_creds* value for all students
  - **UPDATE** *student S*
    SET *tot_cred* = (**SELECT SUM**(*credits*)
                                  FROM *takes, course*
                                  WHERE *takes.course_id* = *course.course_id* **AND**
                                                  S.ID= *takes.ID* **AND**
                                                  takes.grade <> 'F' **AND**
                                                  takes.grade **IS NOT NULL**);

DELETE

- Basic syntax
  - To remove specific rows
    - **DELETE FROM** *tablename*
      WHERE *predicate*;

  - To remove all rows
    - **DELETE FROM** *tablename*;
    - This is equivalent to **TRUNCATE**:
      TRUNCATE (**TABLE**) *tablename*;

    - One cannot truncate a table with foreign key constraints
      - Must disable the constraints first (we will cover **ALTER** when we study SQL DDL):
        ALTER TABLE *tablename*
        DISABLE CONSTRAINT *constraint_name*;

DELETE

- 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');

DELETE

- Delete all instructors whose salary is less than the average salary of instructors
  - Example: **DELETE FROM** *instructor*
    WHERE *salary* < (**SELECT AVG** (*salary*)
    FROM *instructor*);

- Issue: as we delete tuples from *instructor*, the average salary changes
  - Solution used in SQL:
    1. First, compute **AVG**(*salary*) and find all tuples to delete
    2. Next, delete all tuples found above (without recomputing **AVG** or retesting the tuples)

EOF

- Coming next:
  - More on Structured Query Language