Database Systems
(CS 355 / CE 373)

Dr. Umer Tariq
Assistant Professor,
Dhanani School of Science & Engineering,
Habib University

Acknowledgements

- Many slides have been borrowed from the official lecture slides accompanying the textbook:

  Database System Concepts, (2019), Seventh Edition,

  Avi Silberschatz, Henry F. Korth, S. Sudarshan

  McGraw-Hill, ISBN 9780078022159

  The original lecture slides are available at:

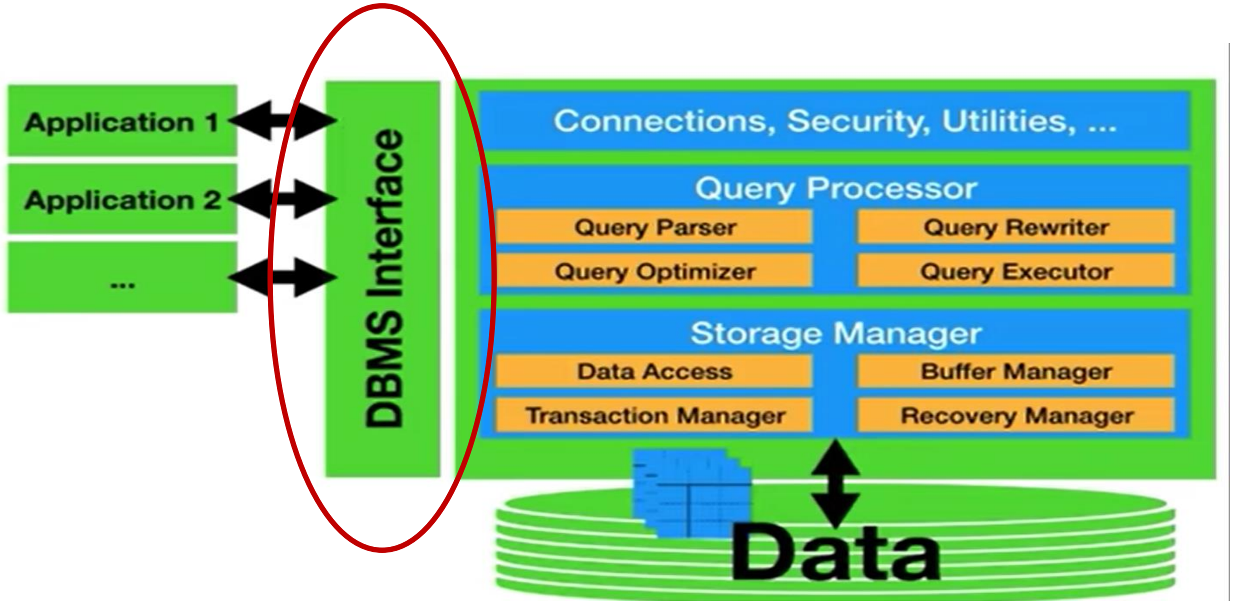
  https://www.db-book.com/

- Some of the slides have been borrowed from the lectures by Dr. Immanuel Trummer (Cornell University). Available at: (www.itrummer.org)

Outline: Week 6

- SQL: An Overview
- Data Definition in SQL
- Basic Structure of SQL Queries
- Aggregate Functions

DBMS-based Approach



- Data Model
  - A collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

The Relational Model

- The relational model uses <u>a collection of tables</u> to represent both data and the relationships among those data.

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

Table/ Relation
instructor

Column / Attribute
dept_name

Row / Tuple
(83821, Brandt, Comp. Sci., 92000)

Figure 2.1 The *instructor* relation.

EXAMPLE :: Find NAMES OF ALL INSTRUCTORS IN "COMP Sci" Department ?

A Sample Relational Model

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

(a) The *instructor* table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

1) FROM

Utilizing Relational DBMS: Lifecycle

1) 1. Design relational schema (and tell the system about it)

2) 2. Populate tables/relations

3) 3. Write "queries" to get information back from tables

“Database Languages” Supported by DBMS Interface

- The DBMS interface supports the following types of languages:

- Data Definition Language (DDL)
  - Used to define the Database Schema

- Data Manipulation Language (DML)
  - Used to Retrieve / manipulate data

“Database Languages” Supported by DBMS Interface

- The DBMS interface supports the following types of languages:
- Data Definition Language (DDL)
  - Used to define the Database Schema
- Data Manipulation Language (DML)
  - Used to Retrieve / manipulate data
- In practice, the DDL and DML are not two separate languages: instead they simply form parts of a single database language.
- For relational databases, the most popular database language is **Structured Query Language (SQL)**

Structured Query Language (SQL)

- The standard to access/retrieve/manipulate data in a **relational database**

- Examples of a Data Definition Language (DDL) Component

```
create table department
    (dept_name   char (20),
     building    char (15),
     budget      numeric (12,2));
```

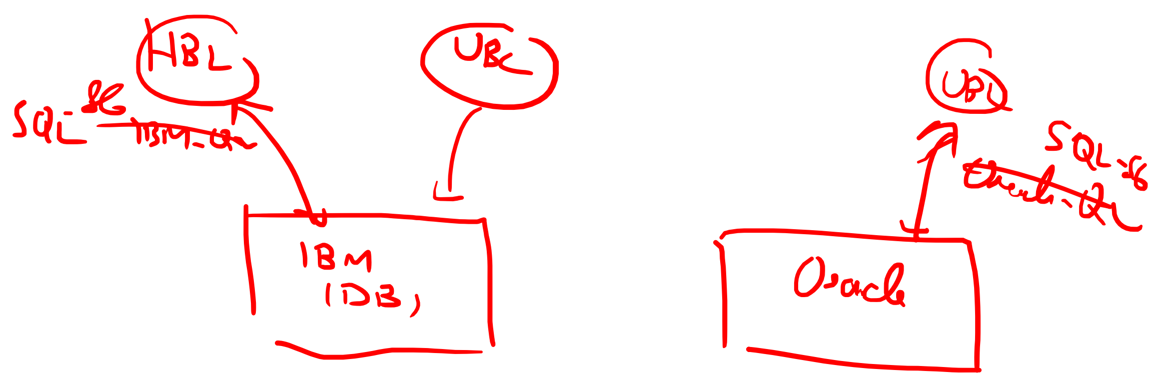
- Examples of a Data Manipulation Language (DML) Component

```
select instructor.name
from instructor
where instructor.dept_name = 'History';
```

```
select instructor.ID, department.dept_name
from instructor, department
where instructor.dept_name= department.dept_name and
      department.budget > 95000;
```

SQL: The History

- IBM developed the original version of SQL, originally called Sequel, in the early 1970s.
- SQL clearly established itself as the standard relational database language.
- In 1986, ANSI and ISO published an SQL standard, called SQL-86.
- Follow-up standards
  - SQL-89
  - SQL-92
  - SQL:1999
  - SQL:2003
  - SQL:2006
  - SQL:2008
  - SQL:2011
  - SQL:2016



SQL: Overview

- The SQL has two main parts:

Data-definition language (**DDL**). The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.

Data-manipulation language (**DML**). The SQL DML provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.

SQL: Data Definition

- Create Tables
- Domain Types
- Identify constraints (primary key / foreign key)

Creating Tables

The general form of the **create table** command is:

create table r
(A_1 D_1,
A_2 D_2,
...,
A_n D_n,
⟨integrity-constraint$_1$⟩,
...,
⟨integrity-constraint$_k$⟩);

where r is the name of the relation, each A_i is the name of an attribute in the schema of relation r, and D_i is the domain of attribute A_i; that is, D_i specifies the type of attribute A_i along with optional constraints that restrict the set of allowed values for A_i.

```
create table department
   (dept_name    varchar (20),
    building     varchar (15),
    budget       numeric (12,2),
    primary key (dept_name));
```

```
create table instructor
   (ID           varchar (5),
    name         varchar (20) not null,
    dept_name    varchar (20),
    salary       numeric (8,2),
    primary key (ID),
    foreign key (dept_name) references department);
```

Domain Types

The SQL standard supports a variety of built-in types, including:

- **char**(n): A fixed-length character string with user-specified length n. The full form, **character**, can be used instead.
- **varchar**(n): A variable-length character string with user-specified maximum length n. The full form, **character varying**, is equivalent.
- **int**: An integer (a finite subset of the integers that is machine dependent). The full form, **integer**, is equivalent.
- **smallint**: A small integer (a machine-dependent subset of the integer type).
- **numeric**(p, d): A fixed-point number with user-specified precision. The number consists of p digits (plus a sign), and d of the p digits are to the right of the decimal point. Thus, **numeric**(3,1) allows 44.5 to be stored exactly, but neither 444.5 nor 0.32 can be stored exactly in a field of this type.
- **real, double precision**: Floating-point and double-precision floating-point numbers with machine-dependent precision.
- **float**(n): A floating-point number with precision of at least n digits.

Integrity Constraints

- Types of integrity constraints
  - **primary key** (A_1, ..., A_n)
  - **foreign key** (A_m, ..., A_n) **references** *r*
  - **not null**

- SQL prevents any update to the database that violates an integrity constraint.

SQL Queries: University Database (1/4)

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

Figure 2.1 The *instructor* relation.

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

Figure 2.7 The *teaches* relation.

SQL Queries: University Database (2/4)

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Figure 2.5 The *department* relation.

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A

Figure 2.6 The *section* relation.

SQL Queries: University Database (3/4)

course_id	*prereq_id*
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

Figure 2.3 The *prereq* 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

Figure 2.2 The *course* relation.

SQL Queries: University Database (4/4)

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

Figure 4.1 The *student* 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*

Figure 4.2 The *takes* relation.

SQL: Data Manipulation

- Basic query structure
- Major clauses in an SQL query
  - SELECT
  - FROM
  - WHERE
  - GROUP BY
  - HAVING

Example :: Find names of all instructors in "Comp Sci" Department ?

SQL Queries: Basic Query Structure

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

Figure 2.1 The *instructor* relation.

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

Figure 2.7 The *teaches* relation.

3) SELECT name ('Project' operator in relational algebra)

1) FROM instructor

2) WHERE dept-name = 'Comp. Sci' ('Selection' operator in Relational algebra')

Basic Query Structure

- A typical SQL query has the form:

select $A_1, A_2, ..., A_n$
from $r_1, r_2, ..., r_m$
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.

"Order of Execution" in SQL Queries

In general, the meaning of an SQL query can be understood as follows:

1. Generate a Cartesian product of the relations listed in the **from** clause.
2. Apply the predicates specified in the **where** clause on the result of Step 1.
3. For each tuple in the result of Step 2, output the attributes (or results of expressions) specified in the **select** clause.

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:

  select *name*

  from *instructor*

	name
1	Srinivasan
2	Wu
3	Mozart
4	Einstein
5	El Said
6	Gold
7	Katz
8	Califieri
9	Singh
10	Crick
11	Brandt
12	Kim

- NOTE: SQL names are case insensitive
  - E.g., *Name* ≡ *NAME* ≡ *name*
  - Some people use upper case wherever bold font is used.

SELECT Clause: *distinct* and *all* Keywords

- SQL allows duplicates in relations as well as in query results.

- To force the elimination of duplicates, insert the keyword **distinct** after select**.**
- Find the department names of all instructors, and remove duplicates

  select distinct *dept_name*
  from *instructor*

	dept_name
1	Biology
2	Comp. Sci.
3	Elec. Eng.
4	Finance
5	History
6	Music
7	Physics

- The keyword **all** specifies that duplicates should not be removed.

  select all *dept_name*
  from *instructor*

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

SELECT Clause

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

  select *

  from *instructor*

	ID	name	dept_name	salary
1	10101	Srinivasan	Comp. Sci.	65000.00
2	12121	Wu	Finance	90000.00
3	15151	Mozart	Music	40000.00
4	22222	Einstein	Physics	95000.00
5	32343	El Said	History	60000.00
6	33456	Gold	Physics	87000.00
7	45565	Katz	Comp. Sci.	75000.00
8	58583	Califieri	History	62000.00
9	76543	Singh	Finance	80000.00
10	76766	Crick	Biology	72000.00
11	83821	Brandt	Comp. Sci.	92000.00
12	98345	Kim	Elec. Eng.	80000.00

- An attribute can also be renamed using ***as***

  select name **as** *nom*

  from *instructor*

	nom
1	Srinivasan
2	Wu
3	Mozart
4	Einstein
5	El Said
6	Gold
7	Katz
8	Califieri
9	Singh
10	Crick
11	Brandt
12	Kim

SELECT Clause: Operators

- The **select** clause can contain arithmetic expressions involving the operation, +, –, ∗, and /, and operating on constants or attributes of tuples.

select *ID, name, salary/12* **as** *monthly_salary*
from instructor

	ID	name	monthly_salary
1	10101	Srinivasan	5416.666666
2	12121	Wu	7500.000000
3	15151	Mozart	3333.333333
4	22222	Einstein	7916.666666
5	32343	El Said	5000.000000
6	33456	Gold	7250.000000
7	45565	Katz	6250.000000
8	58583	Califieri	5166.666666
9	76543	Singh	6666.666666
10	76766	Crick	6000.000000
11	83821	Brandt	7666.666666
12	98345	Kim	6666.666666

2)

WHERE Clause

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

- To find all instructors in Comp. Sci. dept

  select *name*
  from *instructor*
  where *dept_name* = 'Comp. Sci.'

	name
1	Srinivasan
2	Katz
3	Brandt

WHERE Clause: Logical Connectives

- 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 <>. (!=)
- Comparisons can be applied to results of arithmetic expressions
- To find all instructors in Comp. Sci. dept with salary > 70000

  select *name*
  from *instructor*
  where *dept_name* = 'Comp. Sci.' **and** *salary* > 70000

	name
1	Katz
2	Brandt

3) 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 X teaches*

  2) **select ***
  1) **from** *instructor, teaches*

  - generates every possible instructor – teaches pair, with all attributes from both relations.

FROM Clause: Example

select *
from *instructor, teaches*

- Partial Result of the Cartesian Product:

	ID	name	dept_name	salary	ID	course_id	sec_id	semester	year
1	10101	Srinivasan	Comp. Sci.	65000.00	10101	CS-101	1	Fall	2017
2	10101	Srinivasan	Comp. Sci.	65000.00	10101	CS-315	1	Spring	2018
3	10101	Srinivasan	Comp. Sci.	65000.00	10101	CS-347	1	Fall	2017
4	10101	Srinivasan	Comp. Sci.	65000.00	12121	FIN-201	1	Spring	2018
5	10101	Srinivasan	Comp. Sci.	65000.00	15151	MU-199	1	Spring	2018
6	10101	Srinivasan	Comp. Sci.	65000.00	22222	PHY-101	1	Fall	2017
7	10101	Srinivasan	Comp. Sci.	65000.00	32343	HIS-351	1	Spring	2018
8	10101	Srinivasan	Comp. Sci.	65000.00	45565	CS-101	1	Spring	2018
9	10101	Srinivasan	Comp. Sci.	65000.00	45565	CS-319	1	Spring	2018
10	10101	Srinivasan	Comp. Sci.	65000.00	76766	BIO-101	1	Summer	2017
11	10101	Srinivasan	Comp. Sci.	65000.00	76766	BIO-301	1	Summer	2018
12	10101	Srinivasan	Comp. Sci.	65000.00	83821	CS-190	1	Spring	2017
13	10101	Srinivasan	Comp. Sci.	65000.00	83821	CS-190	2	Spring	2017
14	10101	Srinivasan	Comp. Sci.	65000.00	83821	CS-319	2	Spring	2018
15	10101	Srinivasan	Comp. Sci.	65000.00	98345	EE-181	1	Spring	2017
16	12121	Wu	Finance	90000.00	10101	CS-101	1	Fall	2017
17	12121	Wu	Finance	90000.00	10101	CS-315	1	Spring	2018
18	12121	Wu	Finance	90000.00	10101	CS-347	1	Fall	2017
19	12121	Wu	Finance	90000.00	12121	FIN-201	1	Spring	2018

FROM Clause: Example

- Cartesian product is not very useful directly, but it is useful when combined with where-clause condition (selection operation in relational algebra).

- Find the names of all instructors who have taught some course and the course_id

  3) **select** *name, course_id*
  1) **from** *instructor , teaches*
  2) **where** *instructor.ID = teaches.ID*

	name	course_id
1	Srinivasan	CS-101
2	Srinivasan	CS-315
3	Srinivasan	CS-347
4	Wu	FIN-201
5	Mozart	MU-199
6	Einstein	PHY-101
7	El Said	HIS-351
8	Katz	CS-101
9	Katz	CS-319
10	Crick	BIO-101
11	Crick	BIO-301
12	Brandt	CS-190
13	Brandt	CS-190
14	Brandt	CS-319
15	Kim	EE-181

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

- 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

- Match the string "100%"

like '100 \%' **escape** '\'

in that above we use backslash (\) as the escape character.

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.

course_id
CS-101
CS-102
⋮

Set Operations

course_id
CS-101
CS-312
⋮

- Find courses that ran in Fall 2017 or in Spring 2018

  (**select** *course_id* **from** *section* **where** *semester* = 'Fall' **and** *year* = 2017)
  union
  (**select** *course_id* **from** *section* **where** *semester* = 'Spring' **and** *year* = 2018)

- Find courses that ran in Fall 2017 and in Spring 2018

  (**select** *course_id* **from** *section* **where** *semester* = 'Fall' **and** *year* = 2017)
  intersect
  (**select** *course_id* **from** *section* **where** *semester* = 'Spring' **and** *year* = 2018)

- Find courses that ran in Fall 2017 but not in Spring 2018

  (**select** *course_id* **from** *section* **where** *semester* = 'Fall' **and** *year* = 2017)
  except
  (**select** *course_id* **from** *section* **where** *semester* = 'Spring' **and** *year* = 2018)

Set Operations

- Set operations **union, intersect,** and **except**
  - Each of the above operations automatically eliminates duplicates

- To retain all duplicates use the
  - **union all**,
  - **intersect all**
  - **except all**.

Null Values

- It is possible for tuples to have a null value, denoted by **null**, 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**
  - Example: 5 + **null** returns **null**

- The predicate **is null** can be used to check for null values.
  - Example: 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

- SQL treats as **unknown** the result of any comparison involving a null value (other than predicates **is null** and **is not null**).
  - Example*: 5* < **null** or **null** <> **null** or **null** = **null**

- The predicate in a **where** clause can involve Boolean operations (**and**, **or**, **not**); thus the definitions of the Boolean operations need to be extended to deal with the value **unknown**.
  - **and** : *(true* **and** *unknown) = unknown,*
    (false **and** *unknown) = false,*
    (unknown **and** *unknown) = unknown*
  - **or:** (*unknown* **or** *true*) = *true,*
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown) = unknown*

- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

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

- Find the average salary of instructors in the Computer Science department

Aggregate Functions

- Find the average salary of instructors in the Computer Science department

3) SELECT avg(salary)

1) FROM instructor

2) WHERE dept_name = 'Comp Sci'

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

Figure 2.1 The *instructor* relation.

Aggregate Functions

- Find the average salary of instructors in the Computer Science department

select avg (*salary*)
from *instructor*
where *dept_name*= 'Comp. Sci.';

avg (salary)
77333.33333333333

Aggregate Functions

- Find the total number of instructors who teach a course in the Spring 2018 semester

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

Figure 2.7 The *teaches* relation.

distinct

3) SELECT count (ID)

1) FROM teaches

2) WHERE semester='Spring' and year=2018

Aggregate Functions

- 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

Aggregate Functions

- Find the number of tuples in the *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

Figure 2.2 The *course* relation.

3) SELECT count (*)

1) FROM course

2) WHERE

Aggregate Functions

- Find the number of tuples in the *course* relation

select count (*)
from *course*;

count (*)
13