

Principles of Database Systems (CS307)

Lecture 9.1: Relational Algebra

Zhong-Qiu Wang

Department of Computer Science and Engineering
Southern University of Science and Technology

- Most contents are from slides made by Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.
- The slides are largely based on the slides provided by Dr. Yuxin Ma

Relational Algebra

- A procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result
 - Form the basis for the widely used SQL query language
 - Can express SQL statements in a mathematical way
- 6 basic operators:
 - select: σ
 - project: Π
 - rename: ρ
 - union: \cup
 - set difference: $-$
 - Cartesian product: \times
- Unary (operating on one relation) and binary (operating on pairs of relations)

Select Operation

- The select operation selects tuples that satisfy a given predicate (谓词)
 - Notation: $\sigma_p(r)$
 - p is called the selection predicate (选择谓词)
 - Lowercase Greek letter - σ
- Example
 - Select those tuples of the *instructor* relation where the instructor is in the “Physics” department

$$\sigma_{dept_name = "Physics"}(instructor)$$

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Select Operation

- We allow comparisons using $=, \neq, >, \geq, <, \leq$ in the selection predicate
- We can combine several predicates into a larger predicate by using the connectives:

\wedge (**and**), \vee (**or**), \neg (**not**)

- E.g., find the instructors in Physics with a salary greater \$90,000, we write:

$$\sigma_{dept_name = "Physics"} \wedge salary > 90,000 (instructor)$$

- The select predicate may include comparisons between two attributes
 - E.g., find all departments whose name is the same as their building name:

$$\sigma_{dept_name=building} (department)$$

Project Operation

- A unary operation that returns its argument relation, with certain attributes left out

Notation: $\Pi_{A_1, A_2, A_3 \dots A_k}(r)$

where A_1, A_2, \dots, A_k are attribute names, r is a relation name, and Π is uppercase Greek letter pi

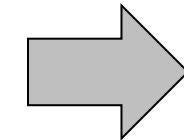
- The result is defined as **the relation of k columns** obtained by **erasing the columns** that are not listed
- Duplicate rows removed from result, since relations are sets

Project Operation

- Example: eliminate the *dept_name* attribute of instructor
 - Query:

$$\Pi_{ID, name, salary} (instructor)$$

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
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



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

Project Operation

- Example: eliminate the *dept_name* attribute of instructor and calculate average salary per month
 - Query:

$$\Pi_{ID, name, salary/12} (\text{instructor})$$

Composition of Relational Operations

- The result of a relational-algebra operation is relation
 - ... and therefore, relational-algebra operations can be composed together into a relational-algebra expression
- Example: Find the names of all instructors in the Physics department

$$\Pi_{name}(\sigma_{dept_name = "Physics"}(instructor))$$

- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation

Cartesian-Product Operation

- The Cartesian-product operation (denoted by a cross, \times) allows us to combine information from any two relations

- E.g., the Cartesian product of the relations `instructor` and `teaches` is written as:

`instructor` \times `teaches`

- We **construct a tuple of the result out of each possible pair of tuples**

- ... one from the `instructor` relation and one from the `teaches` relation (see next slide)
 - Since the `instructor` ID appears in both relations we distinguish between these attribute by attaching to the attribute the name of the relation from which the attribute originally came
 - `instructor.ID` and `teaches.ID`

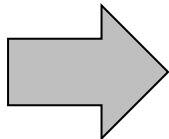
- Relation schema of `instructor` \times `teaches`

- (`instructor.ID`, `instructor.name`, `instructor.dept name`, `instructor.salary`, `teaches.ID`, `teaches.course id`,
`teaches.sec id`, `teaches.semester`, `teaches.year`)
 - (`instructor.ID`, `name`, `dept name`, `salary`, `teaches.ID`, `course id`, `sec id`, `semester`, `year`)

Can remove the prefix if
not producing ambiguity

The “instructor teaches” table

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
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



<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>
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

Join Operation

- **Problem:** The Cartesian-Product “instructor \times teaches” associates **every tuple of instructor** with **every tuple of teaches**
 - Most of the resulting rows have information about instructors who did NOT teach a particular course

<i>instructor.ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>	<i>teaches.ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017

Join Operation

- To get only those tuples of “instructor \times teaches” that pertain to instructors and the courses that they taught, we write:

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

- We get only those tuples of “instructor \times teaches” that pertain to instructors and the courses that they taught
 - i.e., those tuples where $instructor.id = teaches.id$

Join Operation

- The table corresponding to $\sigma_{instructor.id = teaches.id} (instructor \times teaches)$:

<i>instructor.ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>	<i>teaches.ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
32343	El Said	History	60000	32343	HIS-351	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-101	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-319	1	Spring	2018
76766	Crick	Biology	72000	76766	BIO-101	1	Summer	2017
76766	Crick	Biology	72000	76766	BIO-301	1	Summer	2018
83821	Brandt	Comp. Sci.	92000	83821	CS-190	1	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-190	2	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-319	2	Spring	2018
98345	Kim	Elec. Eng.	80000	98345	EE-181	1	Spring	2017

Join Operation

- The table corresponding to $\sigma_{instructor.id = teaches.id} (instructor \times teaches)$:

<i>instructor.ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>	<i>teaches.ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
32343	El Said	History	60000	32343	HIS-351	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-101	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-319	1	Spring	2018
76766	Crick	Biology	72000	76766	BIO-101	1	Summer	2017
76766	Crick	Biology	72000	76766	BIO-301	1	Summer	2018
83821	Brandt	Comp. Sci.	92000	83821	CS-190	1	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-190	2	Spring	2017
83821	Brandt	Comp. Sci.	92000	10101	Srinivasan	Comp. Sci.	65000	12121
98345	Kim	Eng. E	...	10101	Srinivasan	Comp. Sci.	65000	15151
				10101	Srinivasan	Comp. Sci.	65000	22222
				10101	Srinivasan	Comp. Sci.	65000	PHY-101

... will NOT include such tuples (rows) with different IDs

Recall: The Old Way of Writing Joins

- Use commas to separate the tables
 - Example: The solution for the same question in the previous slide
- A little bit history:
 - join was introduced in SQL-1999 (later than this original way)
- Problem:
 - If you forget a comma, it will still work sometimes (interpreted as “renaming”)



```
select m.title, c.credited_as,  
       p.first_name, p.surname  
  from movies m,  
       credits c,  
       people p  
 where c.movieid = m.movieid  
   and p.peopleid = c.peopleid  
   and m.country = 'cn'
```



The SQL syntax was derived from the form of cartesian products in relational algebra

$$\sigma_{movies.id = credits.id \wedge people.peopleid = credits.peopleid \wedge movies.country = "cn"} (Movies \times Credits \times People)$$

Recall: The Old Way of Writing Joins

- Use commas to separate the tables
 - Example: The solution for the same question in the previous slide
- A little bit history:
 - join was introduced in SQL-1999 (later than this original way)
- Problem:
 - The “select operation” is written as the “where” clause here
 - If you forget a comma, it will still work sometimes (interpreted as “renaming”)

$$\sigma_{movies.id = credits.id \wedge people.peopleid = credits.peopleid \wedge movies.country = "cn"}$$

```
select m.title, c.credited_as,  
       p.first_name, p.surname  
from   movies m,  
       credits c,  
       people p  
where  c.movieid = m.movieid  
       and p.peopleid = c.peopleid  
       and m.country = 'cn'
```

Use commas as the “multiplication signs”



The SQL syntax was derived from the form of cartesian products in relational algebra

(Movies \times Credits \times People)

Join Operation

- The join operation allows us to combine a select operation and a Cartesian-Product operation into a single operation
 - Consider relations $r(R)$ and $s(S)$:
 - Let “**theta (θ)**” be a predicate on attributes in the schema R “union” S. The join operation $r \bowtie_{\theta} s$ is defined as follows:

$$r \bowtie_{\theta} s = \sigma_{\theta} (r \times s)$$

- Thus, $\sigma_{instructor.id = teaches.id} (instructor \times teaches)$ can equivalently be written as:
 $instructor \bowtie_{Instructor.id = teaches.id} teaches$

Union Operation

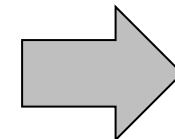
- The union operation allows us to combine two relations
 - Notation: $r \cup s$
- For $r \cup s$ to be valid:
 - r, s must have the same **arity** (same number of attributes)
 - The attribute domains must be compatible
 - Example: 2nd column of r deals with the same type of values as does the 2nd column of s

Union Operation

- Example: To find all courses taught in the Fall 2017 semester, or in the Spring 2018 semester, or in both

$$\Pi_{course_id} (\sigma_{semester="Fall"} \wedge year=2017(section)) \cup \Pi_{course_id} (\sigma_{semester="Spring"} \wedge year=2018(section))$$

<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>	<i>building</i>	<i>room_number</i>	<i>time_slot_id</i>
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



<i>course_id</i>
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Duplicates are removed

Set-Intersection Operation

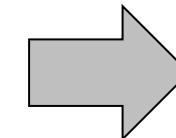
- The set-intersection operation allows us to find tuples that are in both the input relations
 - Notation: $r \cap s$
- Assume (same as Union):
 - r, s have the same arity
 - Attributes of r and s are compatible

Set-Intersection Operation

- Example: Find the set of all courses taught in both the Fall 2017 and the Spring 2018 semesters

$$\Pi_{course_id} (\sigma_{semester = "Fall"} \wedge year = 2017 (section)) \cap \Pi_{course_id} (\sigma_{semester = "Spring"} \wedge year = 2018 (section))$$

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



course_id
CS-101

Set Difference Operation

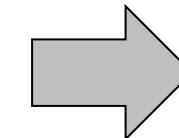
- The set-difference operation allows us to **find tuples that are in one relation but are not in another**
 - Notation: $r - s$
- Assume (same as Union and Set Intersection):
 - r, s have the same **arity**
 - Attributes of r and s are compatible

Set Difference Operation

- Example: Find all courses taught in the Fall 2017 semester, but not in the Spring 2018 semester

$$\Pi_{course_id} (\sigma_{semester="Fall" \wedge year=2017}(section)) - \Pi_{course_id} (\sigma_{semester="Spring" \wedge year=2018}(section))$$

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



course_id
CS-347
PHY-101

The Assignment Operation

- It is convenient at times to write a relational-algebra expression by **assigning parts of it to temporary relation variables**
 - The assignment operation is denoted by \leftarrow and works like **assignment** in a programming language
- Example: Find all instructors in the Physics and Music department

$$Physics \leftarrow \sigma_{dept_name = "Physics"}(instructor)$$
$$Music \leftarrow \sigma_{dept_name = "Music"}(instructor)$$
$$Physics \cup Music$$

- With the assignment operation, a query can be **written as a sequential program consisting of a series of assignments** followed by **an expression** whose value is displayed as the result of the query

The Rename Operation

- Some queries require the same relation to be used more than once in the query
 - The rename operation can be used to give unique names to the different occurrences of the same relation
- The results of relational-algebra expressions do not have a name that we can use to refer to them
 - The rename operator, ρ , is provided for that purpose
 - The expression $\rho_x(E)$ returns **the result of expression E under the name x**
 - Another form of the rename operation which also renames the columns:
 - $\rho_{x(A1, A2, \dots An)}(E)$

Equivalent Queries

- There is more than one way to write a query in relational algebra
 - Example: Find information about courses taught by instructors in the Physics department with salary greater than 90,000
 - Query 1

$$\sigma_{dept_name = "Physics" \wedge salary > 90,000} (instructor)$$

- Query 2

$$\sigma_{dept_name = "Physics"} (\sigma_{salary > 90.000} (instructor))$$

- The two queries are not identical
 - they are, however, **equivalent** -- they give the same result on any database

Equivalent Queries

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

$$\sigma_{dept_name = "Physics"}(instructor \bowtie_{instructor.ID = teaches.ID} teaches)$$

- (Join first, then select)
 - Query 2
 - (Select first, then join)

Equivalent Queries

- Application of Relational Algebra: Query Optimization
 - Transform queries into equivalent ones with less computational cost

Principles of Database Systems (CS307)

Lecture 9.2: Application Development

Zhong-Qiu Wang

Department of Computer Science and Engineering
Southern University of Science and Technology

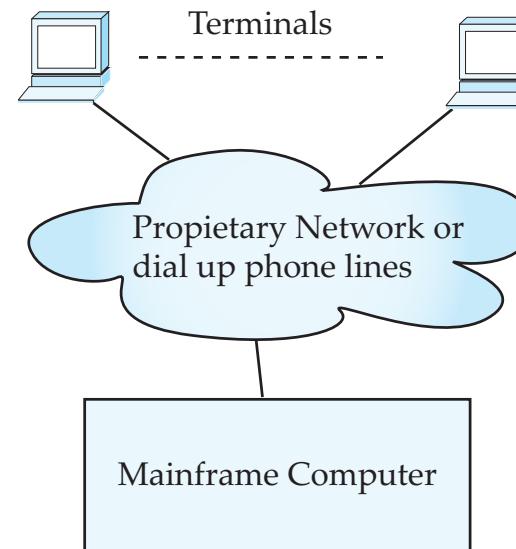
- Most contents are from slides made by Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.
- The slides are largely based on the slides provided by Dr. Yuxin Ma

Application Programs and User Interfaces

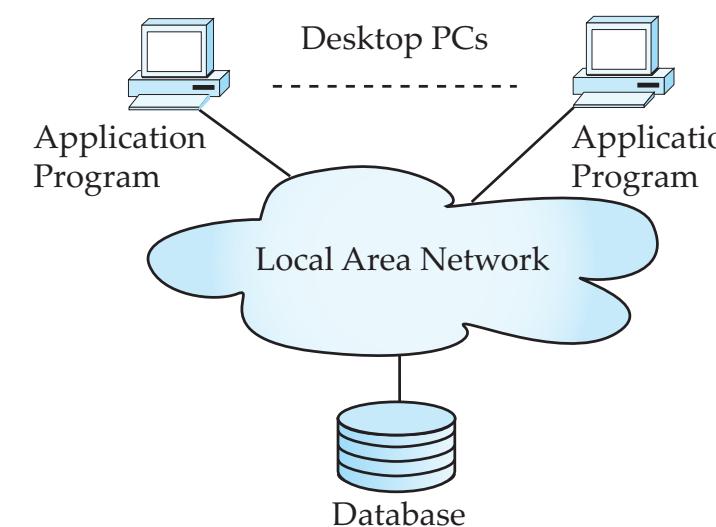
- Most database users *do not* use a query language like SQL
- All use of databases occurs from within application programs
- An application program acts as the intermediary between users and the database
 - Applications split into
 - front-end
 - middle layer
 - backend
- Front-end: user interface
 - Form-based interface
 - Graphical user interfaces
 - Many interfaces are Web-based

Application Architecture Evolution

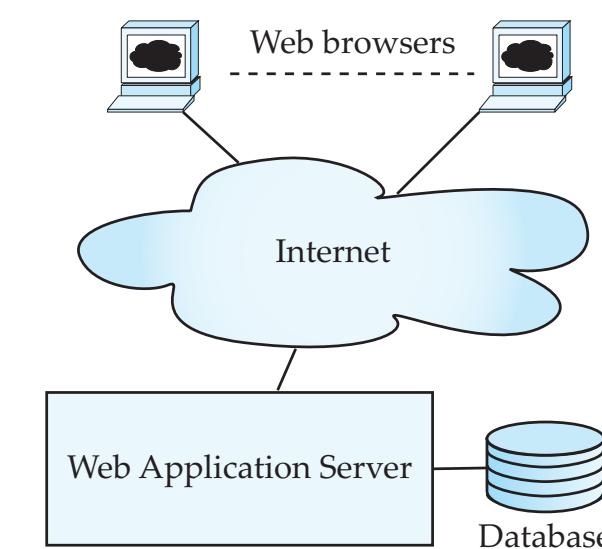
- Three distinct era's of application architecture
 - Mainframe (1960' s and 70' s)
 - Personal computer era (1980' s)
 - Web era (mid 1990' s onwards)
 - Web and Smartphone era (2010 onwards)



(a) Mainframe Era



(b) Personal Computer Era



(c) Web era

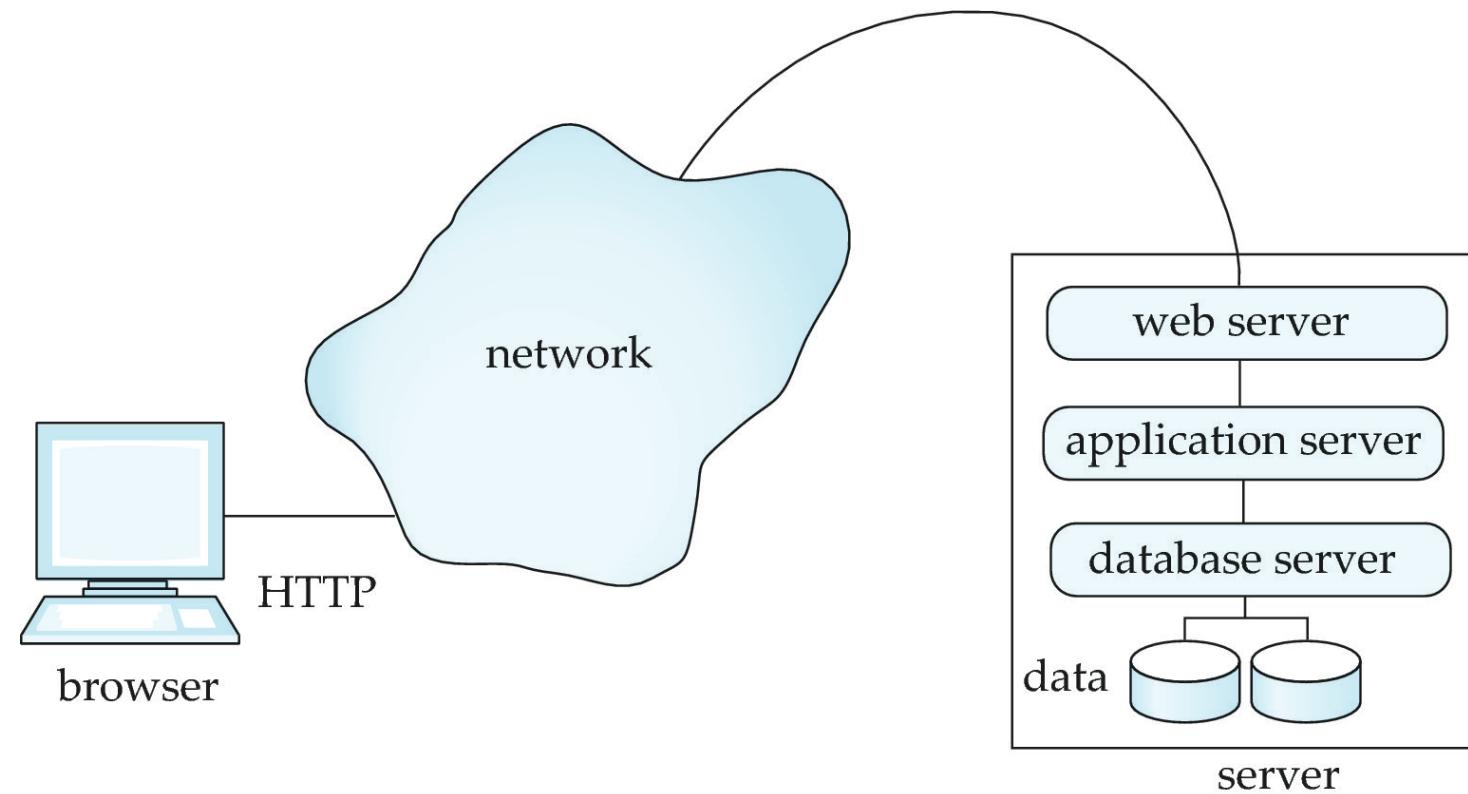
Web Interface

- Web browsers have become the de-facto standard user interface to databases
 - Enable large numbers of users to access databases from anywhere
 - Avoid the need for downloading/installing specialized code, while providing a good graphical user interface
 - JavaScript, Flash and other scripting languages run in browser, but are downloaded transparently
 - Examples: banks, airline and rental car reservations, university course registration and grading, and so on.

The World Wide Web

- The Web is a distributed information system based on hypertext (超文本)
 - Hypertext is a kind of formatted text that provides a link to other content
- Most Web documents are hypertext documents formatted via the HyperText Markup Language (HTML, 超文本标记语言)
- HTML documents contain
 - text along with font specifications, and other formatting instructions
 - hypertext links to other documents, which can be associated with regions of the text.
 - forms, enabling users to enter data which can then be sent back to the Web server

Three-Layer Web Architecture



HTML and HTTP

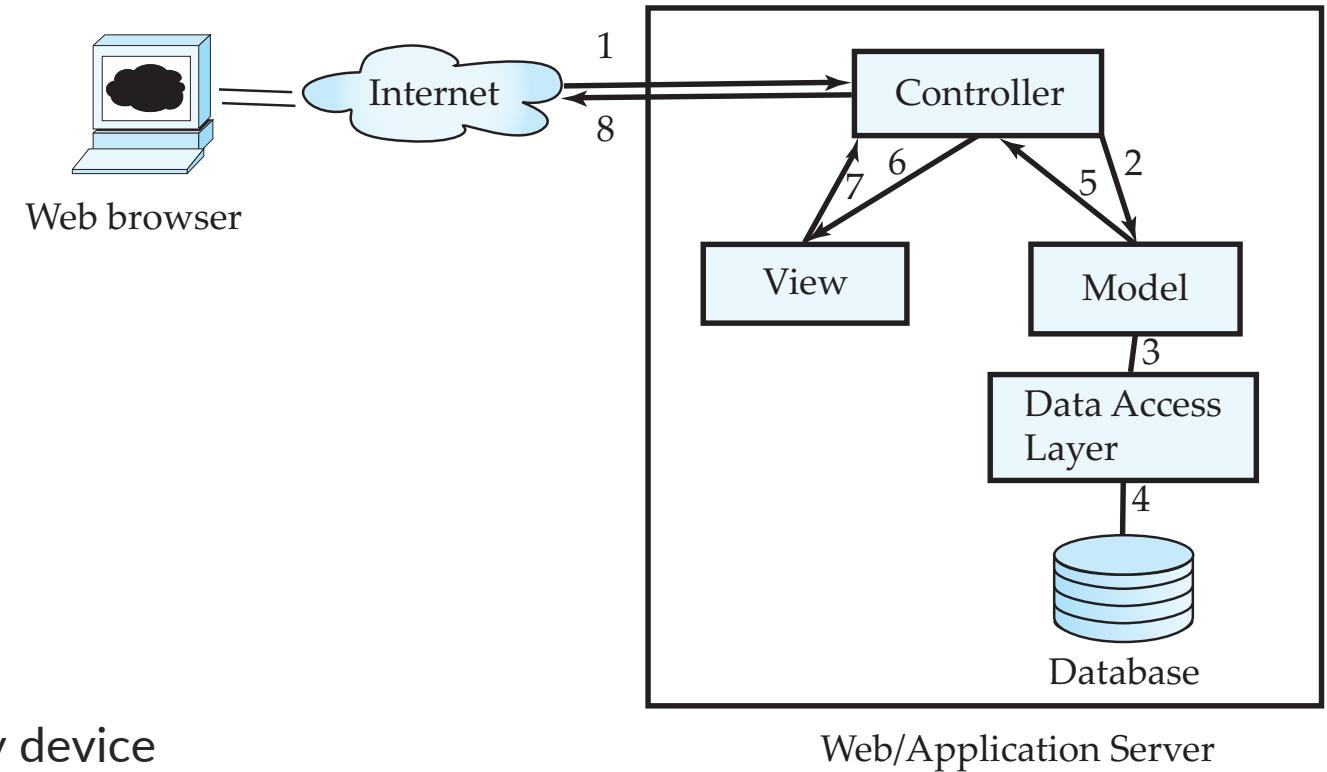
- HTML provides formatting, hypertext link, and image display features
 - including tables, stylesheets (to alter default formatting), etc.
- HTML also provides input features
 - Select from a set of options
 - Pop-up menus, radio buttons, check lists
 - Enter values
 - Text boxes
 - Filled in input sent back to the server, to be acted upon by an executable at the server
- HyperText Transfer Protocol (HTTP) used for communication with the Web server

JavaScript

- JavaScript very widely used
 - Forms basis of new generation of Web applications (called Web 2.0 applications) offering rich user interfaces
- JavaScript functions can
 - Check input for validity
 - Modify the displayed Web page, by altering the underling document object model (DOM) tree representation of the displayed HTML text
 - Communicate with a Web server to fetch data and modify the current page using fetched data, without needing to reload/refresh the page
 - Forms basis of AJAX technology used widely in Web 2.0 applications
 - E.g. on selecting a country in a drop-down menu, the list of states in that country is automatically populated in a linked drop-down menu

Application Architectures

- Application layers
 - Presentation or user interface
 - model-view-controller (MVC) architecture
 - model: business logic
 - view: presentation of data, depends on display device
 - controller: receives events, executes actions, and returns a view to the user
 - business-logic layer
 - provides high level view of data and actions on data
 - often using an object data model
 - hides details of data storage schema
 - data access layer
 - interfaces between business logic layer and the underlying database
 - provides mapping from object model of business layer to relational model of database



Business Logic Layer

- Provides abstractions of entities
 - E.g., students, instructors, courses, etc
- Enforces business rules for carrying out actions
 - E.g., student can enroll in a class only if she has completed prerequisites, and has paid her tuition fees
- Supports workflows which define how a task involving multiple participants is to be carried out
 - E.g., how to process application by a student applying to a university
 - Sequence of steps to carry out task
 - Error handling
 - E.g. what to do if recommendation letters not received on time

Object-Relational Mapping

- Allows application code to be written on top of object-oriented data model, while storing data in a traditional relational database
 - Alternative: implement object-oriented or object-relational database to store object model
 - Has not been commercially successful
- Schema designer must provide a mapping between object data and relational schema
 - E.g., Java class Student mapped to relation student, with corresponding mapping of attributes
 - An object can map to multiple tuples in multiple relations
- Application opens a session, which connects to the database
- Objects can be created and saved to the database using `session.save(object)`
 - Mapping used to create appropriate tuples in the database
- Query can be run to retrieve objects satisfying specified predicates

Web Services

- Allow data on Web to be accessed using remote procedure call mechanism
- Two approaches are widely used
 - Representation State Transfer (REST): allows use of standard HTTP request to a URL to execute a request and return data
 - Returned data is encoded either in XML, or in JavaScript Object Notation (JSON)
 - Big Web Services:
 - Uses XML representation for sending request data, as well as for returning results
 - Standard protocol layer built on top of HTTP

Self Study

- Key points:
 - Techniques / libraries / APIs to connect to a database (e.g. PostgreSQL) and run SQL querys in a program
 - ODBC, JDBC?
 - SQLAlchemy? Spring Boot? Hibernate?
 - (Web) Backend Frameworks
 - Spring Boot, Flask, Django
 - NodeJS
 - Frontend Frameworks
 - React, Vue