UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 3380 Spring 2024

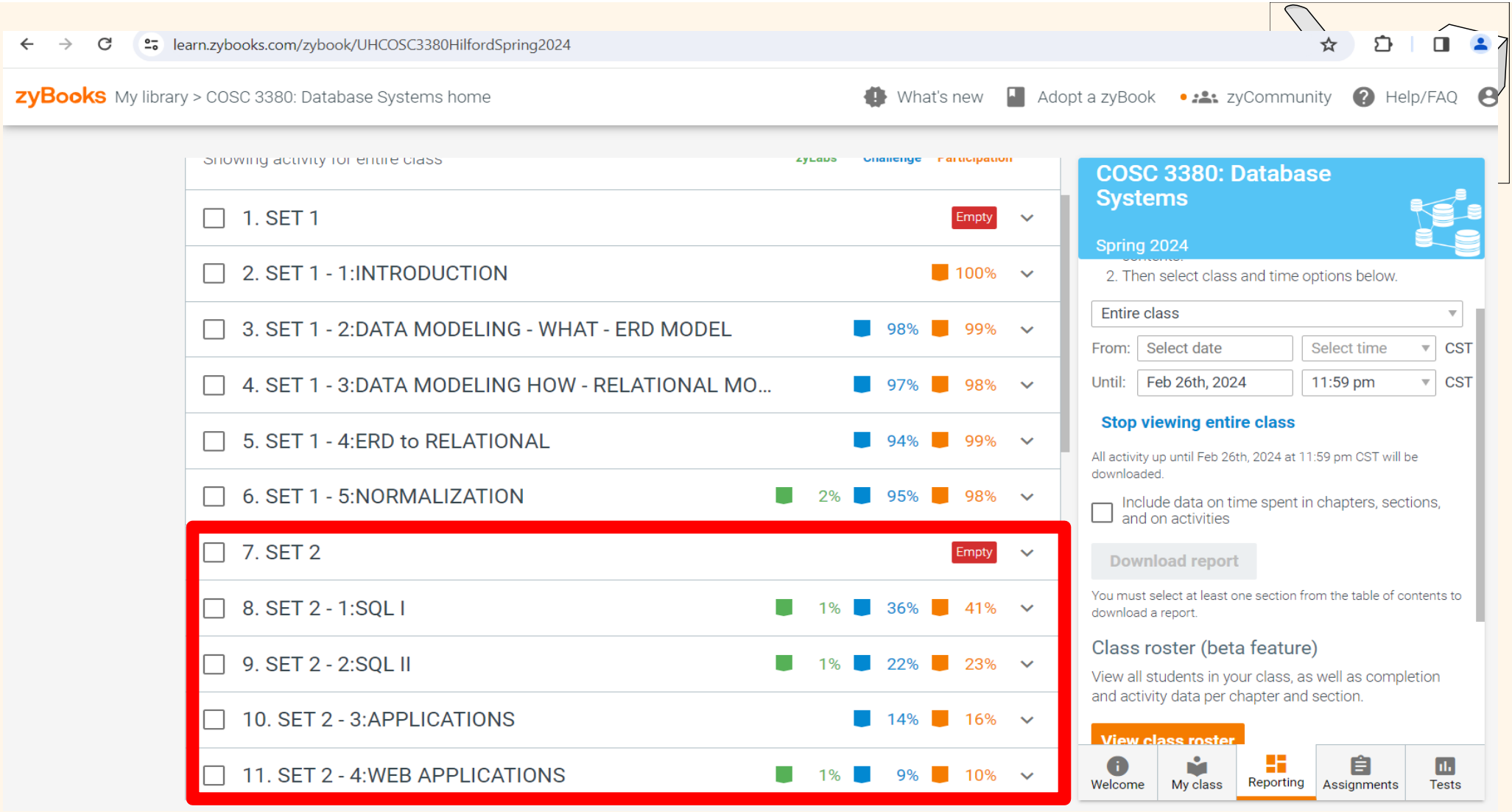
Database Systems

M & W 4:00 to 5:30 PM

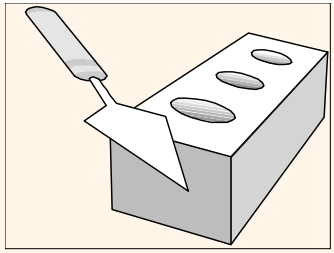
Prof. **Victoria Hilford**

PLEASE TURN your webcam ON (must have)

NO CHATTING during LECTURE



03.04.2024	TA Download	EXAM 2 Review
(14 - Mo)	ZyBook SET 2 Sections	(PART of 20 points)
	(4 PM)	
	(PART of 30 points)	



COSC 3380

4 to 5:30

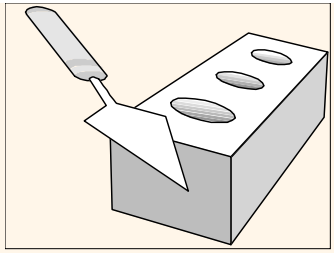
PLEASE

LOG IN

CANVAS

Please close all other windows.

02.26.2024 (12 - Mo)	**ZyBook SET 2 - 4**	**Set 2** LECTURE 11 WEB APPLICATIONS
02.28.2024 (13 - We)		**EXAM 2 Practice** **(PART of 20 points)**
03.04.2024 (14 - Mo)	**TA Download** **ZyBook SET 2 Sections** **(4 PM)** **(PART of 30 points)**	**EXAM 2 Review** **(PART of 20 points)**
03.06.2024 (15 - We)		**EXAM 2** **(PART of 50 points)**

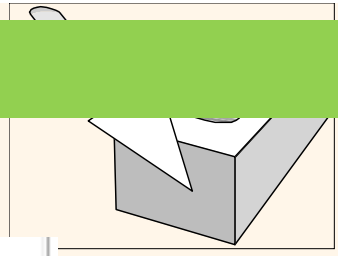


COSC 3380

Class 12

02.26.2024		ZyBook SET 2 - 4	Set 2
(12 - Mo)			LECTURE 11 WEB APPLICATIONS

From 4:00 to 4:07 PM – 5 minutes.



02.26.2024 (12 - Mo)	ZyBook SET 2 - 4	Set 2 LECTURE 11 WEB APPLICATIONS

CLASS PARTICIPATION 20 points — 20% of Total

WEB APPLICATIONS

Class 12 BEGIN PARTICIPATION

Not available until Feb 26 at 4:00pm | Due Feb 26 at 4:07pm | 100 pts

VH, publish

This is a synchronous online class.

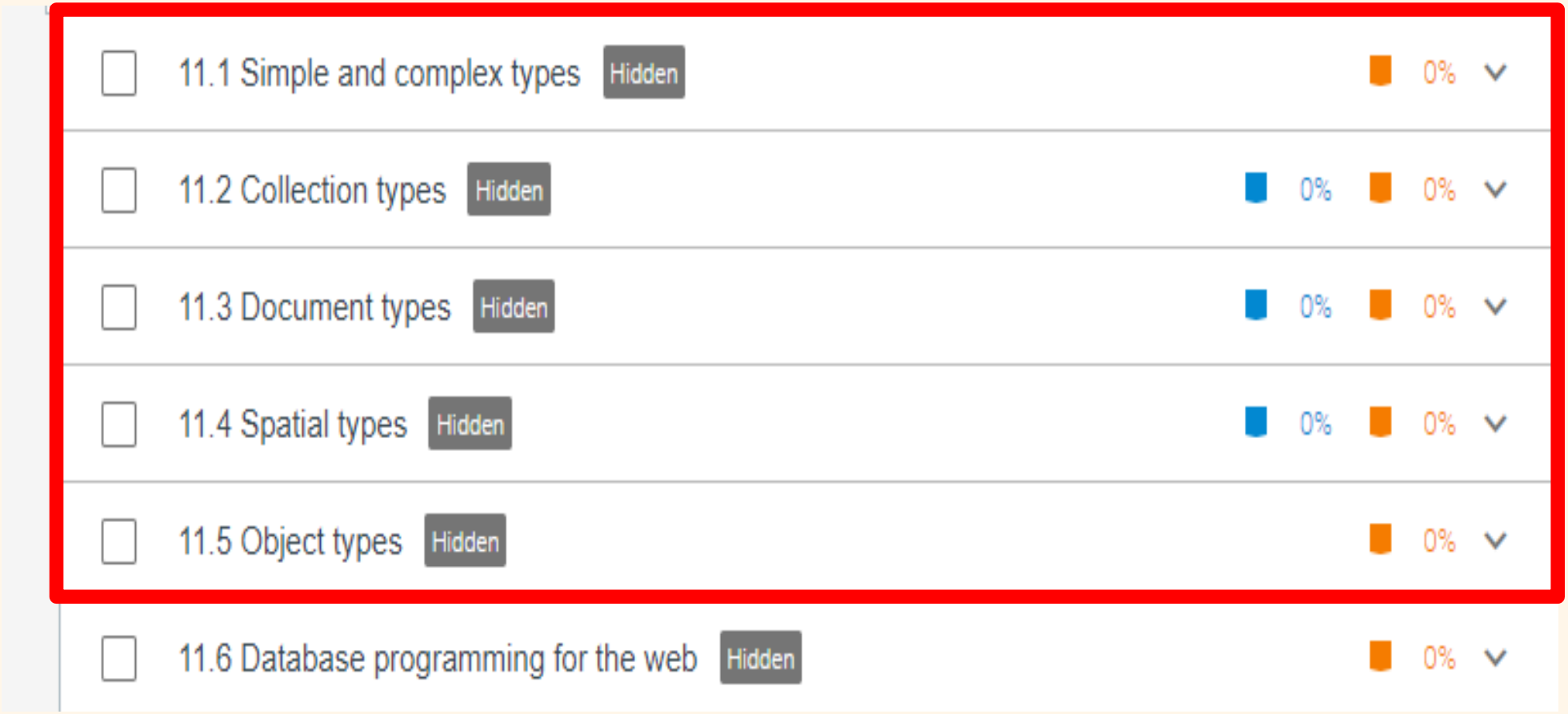
Attendance is required.

Recording or distribution of class materials is prohibited.

1. At the beginning of selected classes there is an **assessment in the first 10 minutes**. (**beige** BOX in the Detailed Syllabus)
2. At the end of selected classes there is an **assessment in the last 10 minutes**. (**blue** BOX in the Detailed Syllabus)
3. ZyBook sections will be downloaded and used for 30% of Total Score on the dates specified in the Detailed Syllabus.
4. EXAMS are in CANVAS. **No late EXAMS**.
5. I have to be **present in TEAMS in order to take any graded assignment assigned during that class.**

From 4:07 to 5:00 PM – 53 minutes.

Relational Databases



Relational Databases

11.1 Simple and complex types

Simple types

Since the 1980s, relational database products have supported six broad categories of data types:

- **Integer** types represent positive and negative integers.
- **Decimal** types represent numbers with fractional values.
- **Character** types represent textual characters. Character types may be either fixed-length or variable-length strings, consisting of either single-byte (ASCII) or double-byte (Unicode) characters.
- **Time** types represent date, time, or both. Some time types include a time zone or specify a time interval.
- **Binary** types store data exactly as the data appears in memory or computer files, bit for bit. Ex: Binary types may be used to store images.
- **Semantic** types are based on other types but have a special meaning and functions. Ex: MONEY has decimal values representing currency. BOOLEAN has values zero and one representing false and true. UUID has string values representing Universally Unique Identifiers, such as **a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11**. ENUM has a fixed set of string values specified by the database designer, such as 'red', 'green', 'blue'.

Relational Databases

11.1 Simple and complex types

Table 11.1.1: Simple type examples implemented by various databases.

	MySQL	Oracle Database	PostgreSQL
Integer	BIT TINYINT SMALLINT MEDIUMINT BIGINT	INT NUMBER	BIT SMALLINT INTEGER BIGINT
Decimal	FLOAT DOUBLE DECIMAL	FLOAT NUMBER	REAL NUMERIC DECIMAL
Character	CHAR VARCHAR TEXT	CHAR VARCHAR2 LONG	CHAR VARCHAR TEXT
Time	DATE DATETIME TIMESTAMP	DATE TIMESTAMP TIMESTAMP WITH TIMEZONE INTERVAL	DATE TIME TIMESTAMP INTERVAL
Binary	TINYBLOB MEDIUMBLOB LONGBLOB	BLOB BFILE RAW	BYTEA
Semantic	ENUM	UROWID	MONEY BOOLEAN UUID

11.1.1: Simple types.

Table 11.1.1: Simple type examples implemented by various databases.

	MySQL	Oracle Database	PostgreSQL
Integer	BIT TINYINT SMALLINT MEDIUMINT BIGINT	INT NUMBER	BIT SMALLINT INTEGER BIGINT
Decimal	FLOAT DOUBLE DECIMAL	FLOAT NUMBER	REAL NUMERIC DECIMAL
Character	CHAR VARCHAR TEXT	CHAR VARCHAR2 LONG	CHAR VARCHAR TEXT
Time	DATE DATETIME TIMESTAMP	DATE TIMESTAMP TIMESTAMP WITH TIMEZONE INTERVAL	DATE TIME TIMESTAMP INTERVAL
Binary	TINYBLOB MEDIUMBLOB LONGBLOB	BLOB BFILE RAW	BYTEA
Semantic	ENUM	UROWID	MONEY BOOLEAN UUID

NUMERIC is a decimal type supported by Oracle Database.

- ○ True
- ◉ False

Correct

NUMERIC is supported by PostgreSQL, not Oracle Database. The Oracle Database NUMBER type is equivalent to the PostgreSQL NUMERIC type.

Your Answer?

1

11.1.1: Simple types.

Table 11.1.1: Simple type examples implemented by various databases.

	MySQL	Oracle Database	PostgreSQL
Integer	BIT TINYINT SMALLINT MEDIUMINT BIGINT	INT NUMBER	BIT SMALLINT INTEGER BIGINT
Decimal	FLOAT DOUBLE DECIMAL	FLOAT NUMBER	REAL NUMERIC DECIMAL
Character	CHAR VARCHAR TEXT	CHAR VARCHAR2 LONG	CHAR VARCHAR TEXT
Time	DATE DATETIME TIMESTAMP	DATE TIMESTAMP TIMESTAMP WITH TIMEZONE INTERVAL	DATE TIME TIMESTAMP INTERVAL
Binary	TINYBLOB MEDIUMBLOB LONGBLOB	BLOB BFILE RAW	BYTEA
Semantic	ENUM	UROWID	MONEY BOOLEAN UUID

1/29/2020 14:30:00 might represent a DATETIME value in MySQL.

- (●) True
- ( ) False

Correct

MySQL supports the DATETIME type. Since 1/29/2020 represents a date and 14:30:00 represents a time, 1/29/2020 14:30:00 might represent a DATETIME value.

Your Answer?

2

11.1.1: Simple types.

Table 11.1.1: Simple type examples implemented by various databases.

	MySQL	Oracle Database	PostgreSQL
Integer	BIT TINYINT SMALLINT MEDIUMINT BIGINT	INT NUMBER	BIT SMALLINT INTEGER BIGINT
Decimal	FLOAT DOUBLE DECIMAL	FLOAT NUMBER	REAL NUMERIC DECIMAL
Character	CHAR VARCHAR TEXT	CHAR VARCHAR2 LONG	CHAR VARCHAR TEXT
Time	DATE DATETIME TIMESTAMP	DATE TIMESTAMP TIMESTAMP WITH TIMEZONE INTERVAL	DATE TIME TIMESTAMP INTERVAL
Binary	TINYBLOB MEDIUMBLOB LONGBLOB	BLOB BFILE RAW	BYTEA
Semantic	ENUM	UROWID	MONEY BOOLEAN UUID

3.1415 can be stored as an INT type in Oracle Database.

- ○ True
- ◉ False

Correct

Oracle Database supports the integer type INT. However, integer types may not have a fractional part. The value 3.1415 must be stored as a decimal type such as FLOAT or NUMBER.

Your Answer?

3

Relational Databases

11.1 Simple and complex types

Complex types

As relational database adoption increased in the 1980s, the need for additional types became apparent. Ex: A spatial type might represent a single point as an X and Y value. Ex: A composite type representing a full name might contain three simple types representing first, middle, and last names. Newer types like spatial and composite have a rich internal structure and are called ***complex data types***.

Most complex types fall into one of four categories:

- **Collection** types include several distinct values of the same base type, organized as a set or an array.
- **Document** types contain textual data in a structured format such as XML or JSON.
- **Spatial** types store geometric information, such as lines, polygons, and map coordinates.
- **Object** types support object-oriented programming constructs, such as composite types, methods, and subtypes.

Research on complex types began in 1986 with the POSTGRES project at the University of California, Berkeley. POSTGRES was released as PostgreSQL in 1997 and now supports complex types in all four categories. In the 1990s, commercial products such as Oracle Database and SQL Server also added support for complex types.

From the perspective of the database system, complex types, like simple types, are atomic and stored as one value per cell.

Set type

	Set	Mulitset	List	Array
MySQL	✓	-	-	-

The MySQL SET type is similar to the ENUM type — both have a base type consisting of character strings. Each ENUM value must contain exactly one element, while each SET value may contain zero, one, or many elements.

A SET type is defined with the SET keyword followed by the base type strings. Ex: `SET('apple', 'banana', orange')` is the type consisting of elements 'apple', 'banana', and 'orange'. A SET value is specified as a string with commas between each element and no blank spaces. Ex: `'apple,orange'` is a value of `SET('apple', 'banana', orange')`.

A SET value can contain no elements. A value with no elements represents the empty set and is not the same as a NULL value.

Internally, each SET value is represented as a series of bits. Each bit corresponds to a specific element. When a bit is one, the corresponding element is included in a value. When a bit is zero, the corresponding element is not included. A base type can have at most 64 elements, so each SET value requires at most 64 bits, or eight bytes.

PARTICIPATION ACTIVITY 11.2.2: MySQL set type.

```
CREATE TABLE Employee (
   ID INTEGER,
   Name VARCHAR(20),
   Language SET('English', 'French', 'Spanish', 'Mandarin', 'Japanese'),
   PRIMARY KEY (ID)
);

INSERT INTO Employee (ID, Name, Language)
VALUES (2538, 'Lisa Ellison', 'English,Spanish'),
       (6381, 'Maria Rodriguez', 'English,Spanish,Japanese'),
       (7920, 'Jiho Chen', 'Mandarin');
```

Employee

ID	Name	Language
2538	Lisa Ellison	English, Spanish
6381	Maria Rodriguez	English, Spanish, Japanese
7920	Jiho Chen	Mandarin

```
SELECT *
FROM Employee
WHERE Language LIKE '%Spanish%';
```

Result

ID	Name	Language
2538	Lisa Ellison	English, Spanish
6381	Maria Rodriguez	English, Spanish, Japanese

Array type

	Set	Mulitset	List	Array
PostgreSQL	-	-	-	✓

PostgreSQL specifies array types by appending a pair of brackets to any base type. A number in the brackets indicates the array size. Ex: `INTEGER[4]` is an array type consisting of four integers. Optionally, the keyword ARRAY can appear between the base type and brackets. Ex: `INTEGER ARRAY[4]` is equivalent to `INTEGER[4]`. If no number appears in the brackets, array size is variable, up to a system maximum.

An array value is specified as a string with comma-separated values within braces. Ex: `'{2, 5, 11, 6}'` is a value of `INTEGER[4]`. An individual array element is specified with an index within brackets. Ex: `WHERE MonthlyHours[2] > 100` selects all rows in which the second element of the MonthlyHours column exceeds 100.

A multidimensional array type can be specified with multiple bracket pairs. Ex: `INTEGER[4][9]` is an integer array with four rows and nine columns. A multidimensional array value is specified with nested braces. Ex: `'{ {2, 5}, {11, 6}, {45, 0} }'` is a value of `INTEGER[3][2]`.

PARTICIPATION ACTIVITY 11.2.4: PostgreSQL array type.

```
CREATE TABLE Employee (
  ID INTEGER,
  Name VARCHAR(20),
  QuarterlySales INTEGER[4],
  PRIMARY KEY (ID)
);

INSERT INTO Employee (ID, Name, QuarterlySales)
VALUES (2538, 'Lisa Ellison', '{ 1450, 2020, 900, 5370 }'),
       (6381, 'Maria Rodriguez', '{ 3340, 800, 1700, 6400 }'),
       (7920, 'Jiho Chen', '{ 0, 3900, 8000, 320 }');
```

1 based

Employee

ID	Name	QuarterlySales [1] [2] [3] [4]
2538	Lisa Ellison	{ 1450, 2020, 900, 5370 }
6381	Maria Rodriguez	{ 3340, 800, 1700, 6400 }
7920	Jiho Chen	{ 0, 3900, 8000, 320 }

```
SELECT *
FROM Employee
WHERE QuarterlySales[2] > 1000;
```

Result

ID	Name	QuarterlySales
2538	Lisa Ellison	{ 1450, 2020, 900, 5370 }
7920	Jiho Chen	{ 0, 3900, 8000, 320 }

TA time (Fernando) – 10 minutes
(CA 11.2.1 Steps 1 - 5 – Collection types)

CHALLENGE ACTIVITY | 11.2.1: Collection types.

All Steps

The following creates a MySQL table:

```
CREATE TABLE Traveler (
   ID INTEGER,
   Name VARCHAR(20),
   Country SET('Serbia', 'Sudan', 'Guinea', 'Chad', 'Guyana'),
   PRIMARY KEY (ID)
);
```

Complete the query below to insert values Guinea, Chad, Serbia, and Sudan.

```
INSERT INTO Traveler (ID, Name, Country)
VALUES (1234, 'Rob Roy',    /* Your code goes here */ );
```

1 | 2 | 3 | 4 | 5

(6) COSC 3380 23578 M &

teams.microsoft.com/_#/modern-calling/

Relaunch to update

Try the new Teams

Search

25:21

Chat | People 117 | Raise | View | Apps | More | Camera | Mic | Share | Leave

Activity
Chat 6
Teams
Assignments
Calendar
Calls
Files
Shifts
Apps
Help

Feedback?

CHALLENGE ACTIVITY | 11.2.1: Collection types.

544874.3144414.qx3zqy7

Jump to level 1

The following creates a MySQL table:

```
CREATE TABLE Traveler (
   ID INTEGER,
   Name VARCHAR(20),
   Country SET('Cyprus', 'Spain', 'Japan', 'Mali', 'Iraq'),
   PRIMARY KEY (ID)
);
```

Complete the query below to insert values Cyprus and Mali.

```
INSERT INTO Traveler (ID, Name, Country)
VALUES (1234, 'Dan Tran', 'Cyprus,Mali' );
```

1 2 3 4 5

Check Next

✔ Expected: 'Cyprus,Mali'

SET values are specified as strings containing elements in any order, separated by commas, and containing no blanks.

Ramirez, Fernando

1/8

FR

Ramirez, Ferna...

View all

In this meeting (117) Mute all

AP Pham, An T
QP Pham, Quoc Hung
NP Phun
ZR Rahm
ZR Rama
FR Ramirez, Fernando
CR Robles, Cristian J
AR Rodrigues, Albamaria V
MR Rodriguez, Miguel A
GR Romero Ramirez, Gabriela
VR Ruiz, Vance
SS Sabu, Sheiin K

Mute participant
Pin for me
Spotlight for everyone
Make an attendee
Remove from meeting

Type here to search

71°F Sunny

4:14 PM
2/26/2024

(6) COSC 3380 23578 M &

teams.microsoft.com/_#/modern-calling/

Relaunch to update

Try the new Teams

Search

26:11

Chat | People 117 | Raise | View | Apps | More | Camera | Mic | Share | Leave

Activity
Chat 6
Teams
Assignments
Calendar
Calls
Files
Shifts
Apps
Help

Arc File Edit View Spaces Tabs Archive Extensions Window Help Mon Feb 26 4:15:43 PM

CHALLENGE ACTIVITY | 11.2.1: Collection types.

544874.3144414.qx3zqy7

Jump to level 1

The following creates a MySQL table:

```
CREATE TABLE Traveler (
   ID INTEGER,
   Name VARCHAR(20),
   Country SET('Kenya', 'Fiji', 'India', 'China', 'Greece'),
   PRIMARY KEY (ID)
);
```

Complete the query below to select all travelers that have visited Kenya.

```
SELECT *
FROM Traveler
WHERE Country LIKE '%Kenya%' ;
```

1 2 3 4 5

Check Next

✔ Expected:

```
LIKE '%Kenya%'
```

SET values can be compared to another string with the LIKE operator. '%' is interpreted as a wildcard. Since 'Kenya' may appear in the middle of a Country string, '%' must appear both before and after 'Kenya'.

Ramirez, Fernando

1/8

FR Ramirez, Ferna...

View all

In this meeting (117) Mute all

AP Pham, An T
QP Pham, Quoc Hung
NP Phun
ZR Rahm
ZR Rama
FR Ramirez, Fernando
CR Robles, Cristian J
AR Rodrigues, Albamaria V
MR Rodriguez, Miguel A
GR Romero Ramirez, Gabriela
VR Ruiz, Vance
SS Sabu, Sheiin K

Mute participant
Pin for me
Spotlight for everyone
Make an attendee
Remove from meeting

Type here to search

71°F Sunny

4:15 PM
2/26/2024

(6) COSC 3380 23578 M &

teams.microsoft.com/_#/modern-calling/

Relaunch to update

Try the new Teams

Search

27:14

Chat | People 116 | Raise | View | Apps | More | Camera | Mic | Share | Leave

Activity
Chat 6
Teams
Assignments
Calendar
Calls
Files
Shifts
Apps
Help

Arc File Edit View Spaces Tabs Archive Extensions Window Help Mon Feb 26 4:16:46 PM

CHALLENGE ACTIVITY | 11.2.1: Collection types.

544874.3144414.qx3zqy7

Jump to level 1

Complete the query below to create a PostgreSQL table.

Runner

ID	Name	SixMonthMileage
8878	Abe Cruz	{503, 403}
2571	Ari Dean	{486, 495}
1944	Noa Chen	{487, 403}
8829	Ken Khan	{417, 424}
3332	Ada Ross	{309, 380}
5996	Sue Shaw	{511, 467}

```
CREATE TABLE Runner (
  ID INTEGER,
  Name VARCHAR(20),
  SixMonthMileage INTEGER[2],
  PRIMARY KEY (ID)
);
```

1 2 3 4 5

Check Next

✔ Expected:

```
INTEGER ARRAY[2]
or
INTEGER[2]
```

The SixMonthMileage column is an array of two integers, representing a runner's mileage every six months.

Ramirez, Fernando

1/8

FR Ramirez, Ferna...

View all

In this meeting (116) Mute all

Pham, Quoc Hung
Phung, Nam K
Rahm
Rama
Rami
Roble
Rodrigues, Albamaria V
Rodriguez, Miguel A
Romero Ramirez, Gabriela
Ruiz, Vance
Sabu, Shejin K
Salim, Bagir

Mute participant
Pin for me
Spotlight for everyone
Make an attendee
Remove from meeting

Type here to search

71°F Sunny

4:16 PM
2/26/2024

(6) COSC 3380 23578 M &

teams.microsoft.com/_#/modern-calling/

Relaunch to update

Try the new Teams

Search

29:47

Chat | People 115 | Raise | View | Apps | More | Camera | Mic | Share | Leave

Activity
Chat 6
Teams
Assignments
Calendar
Calls
Files
Shifts
Apps
Help

Arc File Edit View Spaces Tabs Archive Extensions Window Help Mon Feb 26 4:19:18 PM

CHALLENGE ACTIVITY | 11.2.1: Collection types.

544874.3144414.qx3zqy7

Jump to level 1

Consider the PostgreSQL table and SELECT query.

Runner

ID	Name	SixMonthMileage
4784	Ken Leon	{439, 481}
3879	Bob Vega	{379, 384}
8660	Avi Khan	{513, 522}
2162	Tia Baca	{636, 511}
7065	Meg Park	{634, 627}
8187	Guy Tran	{673, 651}

```
SELECT *
FROM Runner
WHERE SixMonthMileage[1] > 500;
```

The result table has 4 runners.

1 2 3 4 5

Check Next

Expected: 4

SixMonthMileage[1] refers to the first element of each value in the SixMonthMileage column. Runners with IDs 8660, 2162, 7065, and 8187 have SixMonthMileage[1] > 500, so 4 runners.

View solution (Instructors only)

Ramirez, Fernando

1/8

FR Ramirez, Ferna...

View all

In this meeting (115) Mute all

NP Phung, Nam K
ZR Rahman, Zayed
ZR Ramadan, Zeyad
FR Ramirez, Fernando
CR Robles, Cristian J
AR Rodrigues, Albamaria V
MR Rodriguez, Miguel A
GR Romero Ramirez, Gabriela
VR Ruiz, Vance
SS Sabu, Shejin K
BS Salim, Baqir
ES Sehic, Edin

Type here to search

71°F Sunny

4:19 PM 2/26/2024

(6) COSC 3380 23578 M &

teams.microsoft.com/_#/modern-calling/

Relaunch to update

Try the new Teams

Search

32:46

Chat | People 115 | Raise | View | Apps | More | Camera | Mic | Share | Leave

Activity
Chat 6
Teams
Assignments
Calendar
Calls
Files
Shifts
Apps
Help

Arc File Edit View Spaces Tabs Archive Extensions Window Help Mon Feb 26 4:22:17 PM

CHALLENGE ACTIVITY | 11.2.1: Collection types.

544874.3144414.qx3zqy7

Jump to level 1

The following refers to the PostgreSQL array type.
Ex: '{{2, 3, 4, 5}, {11, 12, 13, 14}}' is a value of `INTEGER[2][4]`.

Select all valid values of type `INTEGER[3][4]`

- (1) {{9, 23, 11, 17}, {10, 18, 21, 6}, {4, 22, 20, 7}}
- (2) '{17, 7, 1, 2, 16, 22, 5, 23, 24, 13, 11, 12}'
- (3) '{{12, 11, 24, 9}, {3, 21, 23, 17}, {7, 25, 14, 19}}'
- (4) '{{19, 0, 17}, {18, 20, 22}, {8, 11, 1}, {13, 15, 5}}'
- (5) {{13, 0, 20, 17}, {14, 16, 24, 23}, {11, 12, 10, 7}}
- (6) '{{23, 5, 0}, {7, 16, 25}, {20, 17, 6}, {19, 13, 9}}'

1 2 3 4 5

Check Next **Done**. Click any level to practice more. Completion is preserved.

✔ Expected: (3)

(3) is an integer array with 4 columns and 3 rows, so is a value of `INTEGER[3][4]`.

Invalid values:
(1) and (5) are not strings with comma-separated values within nested braces

Ramirez, Fernando

1/8

FR Ramirez, Ferna...

View all

In this meeting (115) Mute all

- NP Phung, Nam K
- ZR Rahman, Zayed
- ZR Ramadan, Zeyad
- FR Ramirez, Fernando
- CR Robles, Cristian J
- AR Rodrigues, Albamaria V
- MR Rodriguez, Miguel A
- GR Romero Ramirez, Gabriela
- VR Ruiz, Vance
- SS Sabu, Shejin K
- BS Salim, Baqir
- ES Sehic, Edin

Type here to search

71°F Sunny

4:22 PM
2/26/2024

11.3 Document types

Document types

Structured data is stored as a fixed set of named data elements, organized in groups. A type is explicitly declared for each element. Each group has the same number of elements with the same names and types. Ex: Spreadsheets and relational tables contain structured data.

Semistructured data is similar to structured data, except each group may have a different number of elements with different names. Types are not explicitly declared. Instead, elements are stored as characters, and type is inferred from the data. Ex: `<Temperature>98.6</Temperature>` represents an element 98.6 named Temperature with a decimal type.

Unstructured data is stored as elements embedded in a continuous string of characters or bits. Element names and types are not declared. Ex: A statistical report contains numerous data elements, but individual elements are not explicitly separated or named. Ex: A bitmap image may contain images of people, but sophisticated algorithms are necessary to identify each individual.

Semistructured data is stored in a ***document*** as text in a flexible format, such as XML or JSON. Most relational databases support XML or JSON document types. Each value of a document type is a complete XML or JSON document and may contain many elements.

Table 11.3.1: Document type support.

	XML	JSON
MySQL	-	✓
Oracle Database	✓	-
PostgreSQL	✓	✓
SQL Server	✓	-
DB2	✓	-
Informix	-	-

XML format

XML stands for eXtensible Markup Language and uses tags instead of columns. A ***tag*** is a name enclosed in angle brackets < >. An ***XML element*** consists of a start tag, data, and an end tag. The end tag is the start tag with a forward slash / before the name. Ex: `<Department>Accounting</Department>` is an element with tag name 'Department' and data 'Accounting'.

XML elements can be nested by embedding one pair of start and end tags within another. An XML document must have a ***root*** element that contains all other elements.

XML documents have an optional first line, called the ***declaration***, that specifies document processing information such as the XML version and character encoding. Ex: `<?xml version = "2.0" encoding = "UTF-8"?>` indicates the document is formatted with XML version 2.0 and the UTF-8 Unicode encoding.

PARTICIPATION ACTIVITY 11.3.2: XML format.

```
<?xml version = "2.0" encoding = "UTF-16"?>
<Customer>
   <Name>Maria Rodriguez</Name>
   <Vehicle>
      <Make>Ford</Make>
      <Model>F-150</Model>
      <Year>2008</Year>
   </Vehicle>
   <Vehicle>
      <Make>Toyota</Make>
      <Model>Camry</Model>
      <Year>2019</Year>
   </Vehicle>
   <Budget></Budget>
   <PreviousCustomer>true</PreviousCustomer>
   <FamilyMember>Jose</FamilyMember>
   <FamilyMember>Felicia</FamilyMember>
   <FamilyMember>Isabella</FamilyMember>
   <Notes>Shopping for a new sports car. Interested in leasing.</Notes>
</Customer>
```

JSON format

JSON stands for JavaScript Object Notation and is commonly pronounced 'JAY-sun'. JSON format is similar to XML but more compact.

A ***JSON element*** consists of a name and associated data, written as `"name":data`. Ex: The JSON element `"Department":"Accounting"` is equivalent to the XML element `<Department>Accounting</Department>`. The element name is not repeated, reducing document size compared to XML.

Unlike XML, JSON elements have a type. The data following an element name must be one of six types:

- String — a series of characters enclosed in double quotes
- Number — a series of digits with an optional decimal point
- Boolean — the strings `true` or `false`
- Null — the string `null`
- Array — multiple data values enclosed in brackets. Ex: `["Arabic","English","Spanish"]`.
- Object — multiple elements enclosed in braces. Ex: `{"Employee":"Sam Snead","Salary":55000}`.

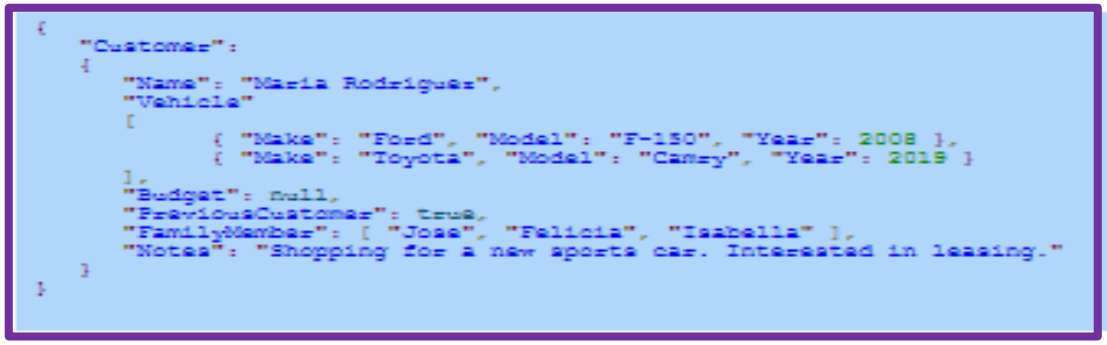
Hierarchical data is represented in JSON by nesting elements, as in XML. Ex: First and Last elements are nested in a Name element, and Name and Salary elements are nested in an Employee element in the following JSON:
`"Employee":{"Name":{"First":"Sam", "Last":"Snead"},"Salary":55000}`.

XML is an early document type, developed as internet use proliferated in the 1990s. JSON became popular about ten years later, replacing XML in applications when document size was important. JSON is commonly used to transmit data over the internet — the compact format reduces transmission time and improves application performance.

Terminology

The name of a JSON element is commonly called a key and the data is commonly called a value. This section uses the terms name and data to avoid confusion with the primary key of a table and the value in a cell of a table.

PARTICIPATION ACTIVITY | 11.3.4: JSON format.



```
{
   "Customer":
   {
      "Name": "Maria Rodriguez",
      "Vehicle"
      [
            { "Make": "Ford", "Model": "F-150", "Year": 2008 },
            { "Make": "Toyota", "Model": "Camry", "Year": 2019 }
      ],
      "Budget": null,
      "PreviousCustomer": true,
      "FamilyMember": [ "Jose", "Felicia", "Isabella" ],
      "Notes": "Shopping for a new sports car. Interested in leasing."
   }
}
```

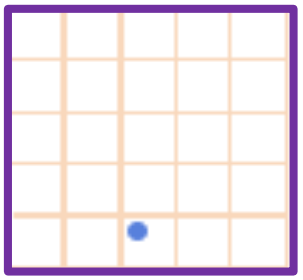
11.4 Spatial types

Spatial data

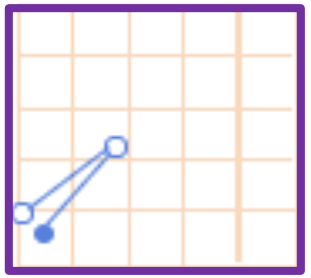
Spatial data is a geometric object, such as a point, line, polygon, or sphere, specified as coordinates in an N-dimensional space. Two-dimensional data is used in mapping applications, such as Google Maps. Three-dimensional data is used in computer-aided design (CAD) systems to engineer airplanes, buildings, and integrated circuits. Weather applications may use two-dimensional data to describe temperature on the surface of the earth, or three-dimensional data to describe atmospheric conditions.

Spatial data is commonly written in a format called ***Well-Known Text*** (***WKT***). WKT format specifies a shape name followed by vertex coordinates.

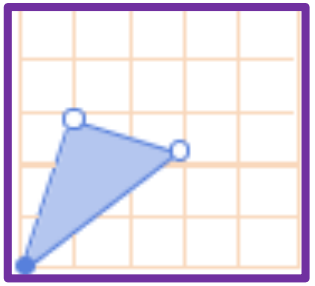
PARTICIPATION ACTIVITY | 11.4.1: WKT format.



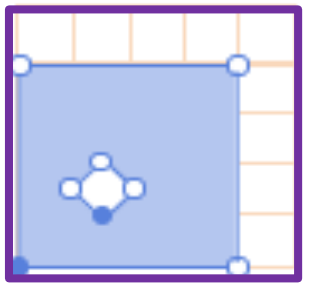
POINT(22 8)



LINESTRING(5 8, 18 21, 0 10)



POLYGON((0 0, 29 21, 10 29, 0 0))



POLYGON((0 0, 40 0, 40 40, 0 40, 0 0),
(15 10, 20 15, 15 20, 10 15, 15 10))

11.5 Object types

Object-orientation

The relational model was developed in the 1970s. Relational products were introduced and adopted throughout the 1980s. Object-oriented programming languages like Java and C++ became popular in the 1990s. As a result, most relational products did not initially support object-oriented capabilities, such as:

- ***Composite types*** combine several properties in one type.
- ***Methods*** are functions or procedures associated with a type.
- ***Subtypes*** are derived from an existing type, called a ***supertype***. The subtype automatically inherits all supertype properties and methods. The subtype may also have additional properties and methods.
- A subtype's method may ***override***, or redefine, the behavior of the supertype's inherited method.

A composite type is called a ***class*** in object-oriented programming languages. A subtype is called a ***derived class***, and a supertype is called a ***base class***.

As object-oriented programming became mainstream, some relational databases added composite types, methods, and subtypes. The goal was twofold — extend database capabilities and simplify database programming with object-oriented languages.

PARTICIPATION ACTIVITY | 11.5.1: Object-oriented programming in Python.

```
class Address:
    def printAddress(self):
        print(self.street)
        print(self.city + ', ' +
              self.stateCode + ' ' +
              self.postalCode)

address = Address()
address.street = '440 Maple Street'
address.city = 'Chicago'
address.stateCode = 'IL'
address.postalCode = '60620'
address.printAddress()
```

440 Maple Street
Chicago, IL 60620

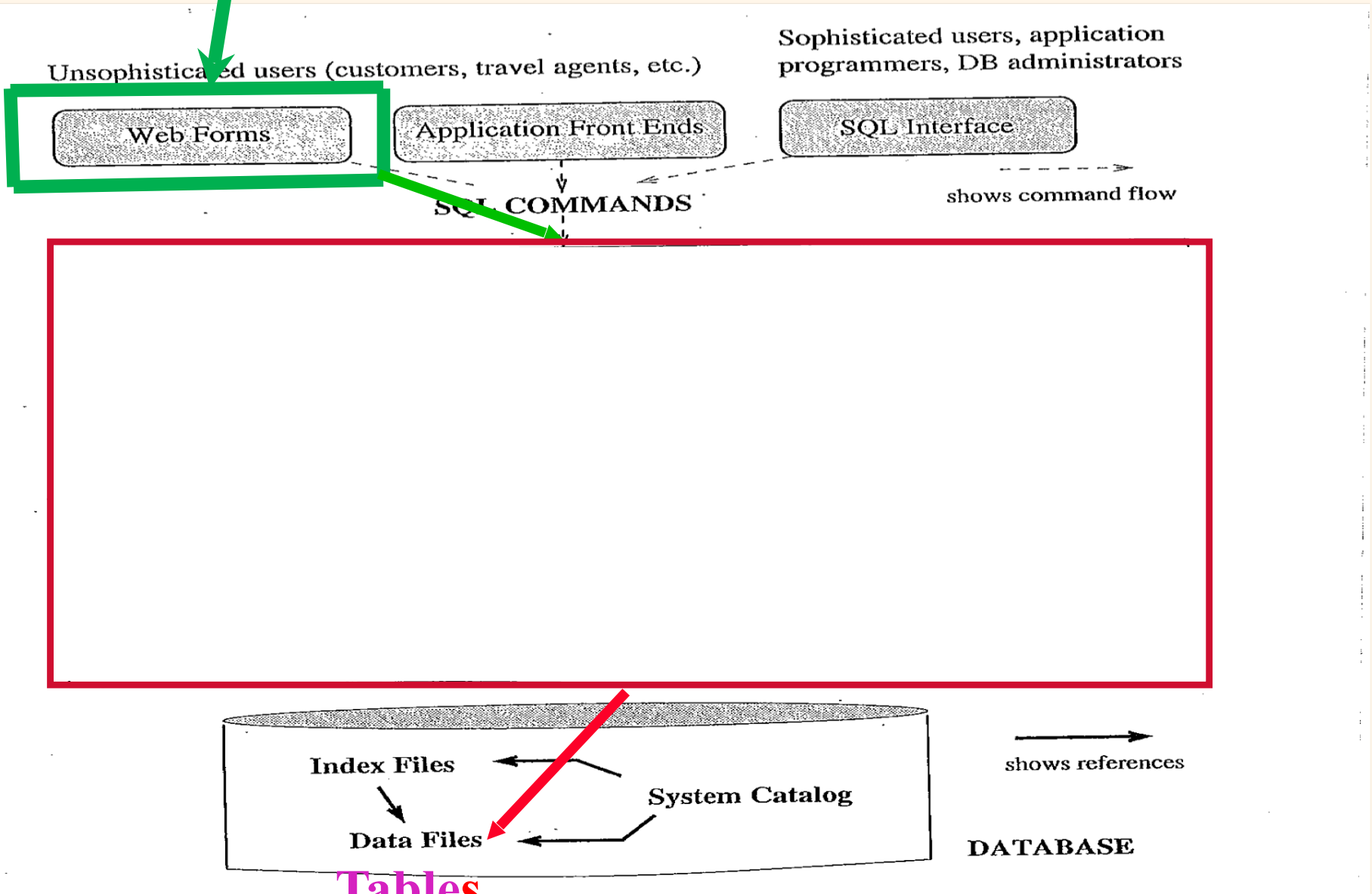
```
class GlobalAddress(Address):
    def printAddress(self):
        Address.printAddress(self)
        print(self.country)

globalAddress = GlobalAddress()
globalAddress.street = '32 Oxford Avenue'
globalAddress.city = 'Vancouver'
globalAddress.stateCode = 'BC'
globalAddress.postalCode = 'V5H 3Z7'
globalAddress.country = 'Canada'
globalAddress.printAddress()
```

32 Oxford Avenue
Vancouver, BC V5H 3Z7
Canada

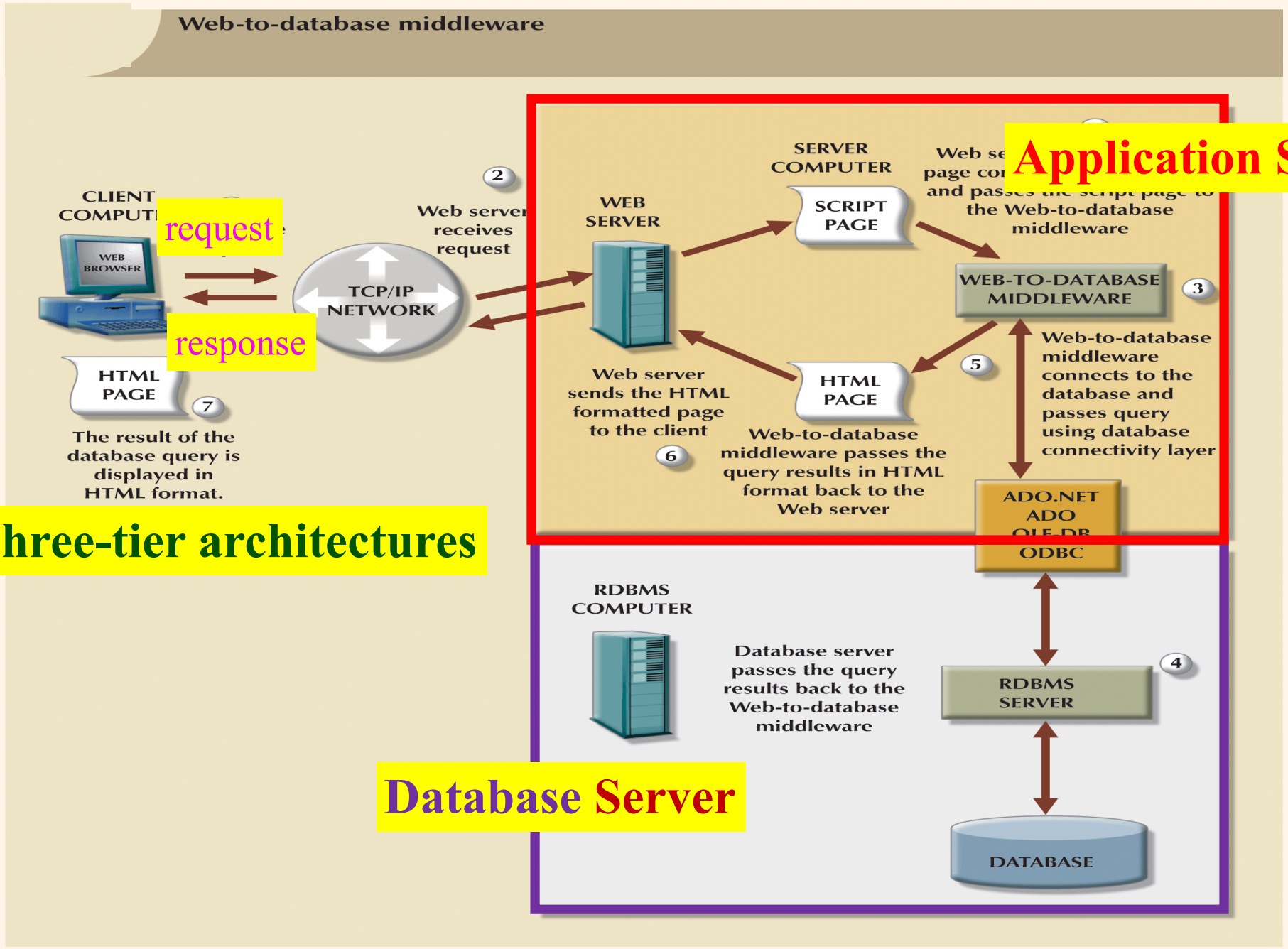
Lecture 10

Internet Applications
with a database back-end

DBMS Internet Applications



Database Application Development

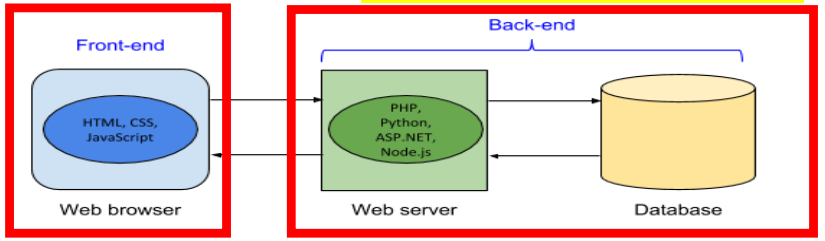


11.1 Full-stack development (Node)

Overview of front-end and back-end development

Most websites and web applications require the development of client-side technologies that interact with server-side technologies. ***Client-side*** (or ***front-end***) refers to those technologies that run in the web browser like HTML, CSS, and JavaScript. ***Server-side*** (or ***back-end***) refers to those technologies that run on the web server like PHP, Python, Node.js, etc. and databases. Ex: Amazon uses server-side technologies to store information on millions of products and a client-side search interface that interacts with the web server so customers can find and purchase products.

Figure 11.1.1: Front-end and back-en **Application Development**



A ***front-end developer*** is a developer that is proficient in client-side technologies. A ***back-end developer*** is a developer that is proficient in server-side technologies. Many developers strive to be proficient in both front-end and back-end technologies and how the two sides work together. A ***full-stack developer*** is a developer who has expertise in all aspects of a website or web application's development, including client technologies, server technologies, data modeling, and user interfaces. The "stack" in "full-stack" refers to the various layers that compose websites and web applications. Technology stacks have increased in complexity over the years, so even "full-stack" developers typically specialize in a few areas of the technology stack.

Overview

- Internet Concepts
- Web data formats
  - HTML, XML, DTDs
- Introduction to three-tier architectures
- The presentation layer
  - HTML forms; HTTP Get and POST, URL encoding; Javascript; Stylesheets; XSLT
- The middle tier
  - CGI, application servers, Servlets, JavaServerPages, ASP, PHP, passing arguments, maintaining state (cookies)

Uniform Resource Identifiers

- Uniform naming schema to identify ***resources* on the Internet**
- A **resource** can be anything:
  - **index.html**
  - **mysong.mp3**
  - **picture.jpg**

- Example URIs:

http://**www.cs.wisc.edu**/~dbbook/**index.html**

mailto:**webmaster@bookstore.com**

HyperText Transfer Protocol

What is a communication protocol?

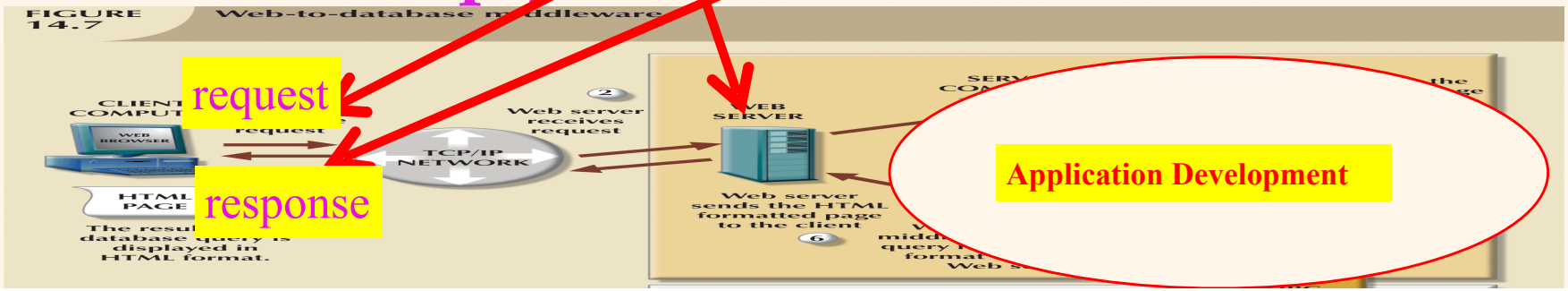
- Set of **standards** that defines the structure of **messages**
- Examples: TCP, IP, HTTP, HTTPS

What happens if you click on www.cs.wisc.edu/~dbbook/index.html?

Client (web browser) sends HTTP **request** to Web Server

Web Server receives **request** and **replies**

Client receives **reply** from Web Server; makes new **requests**



HTTP

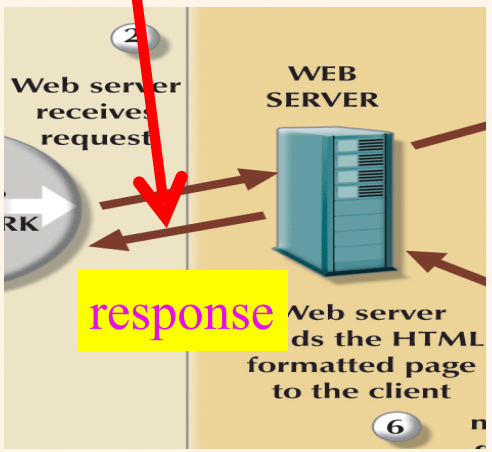
Client request to Web Server:

GET ~/**index.html** HTTP/1.1
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg

Web Server replies **response** to **Client**:

HTTP/1.1 200 OK
Date: Wed, 26 Feb 2014 16:00:00 GMT
Server: Apache/1.3.0 (Linux)
Last-Modified: Wed, 04 Oct 2006 16:30:24 GMT
Content-Length: 1024
Content-Type: text/html

```
<HTML>
<HEAD></HEAD>
<BODY>
<h1>Barns and Nobble Internet Bookstore</h1>
Our inventory:
<h3>Science</h3>
<b>The Character of Physical Law</b>
...

</HTML>
```



*Some Remarks About **HTTP***

HTTP is stateless

- No "sessions"
- Every **message** is completely self-contained
- No previous interaction is "**remembered**" by the protocol
- Tradeoff between ease of implementation and ease of Application development: Other functionality has to be built on top

Implications for Applications:

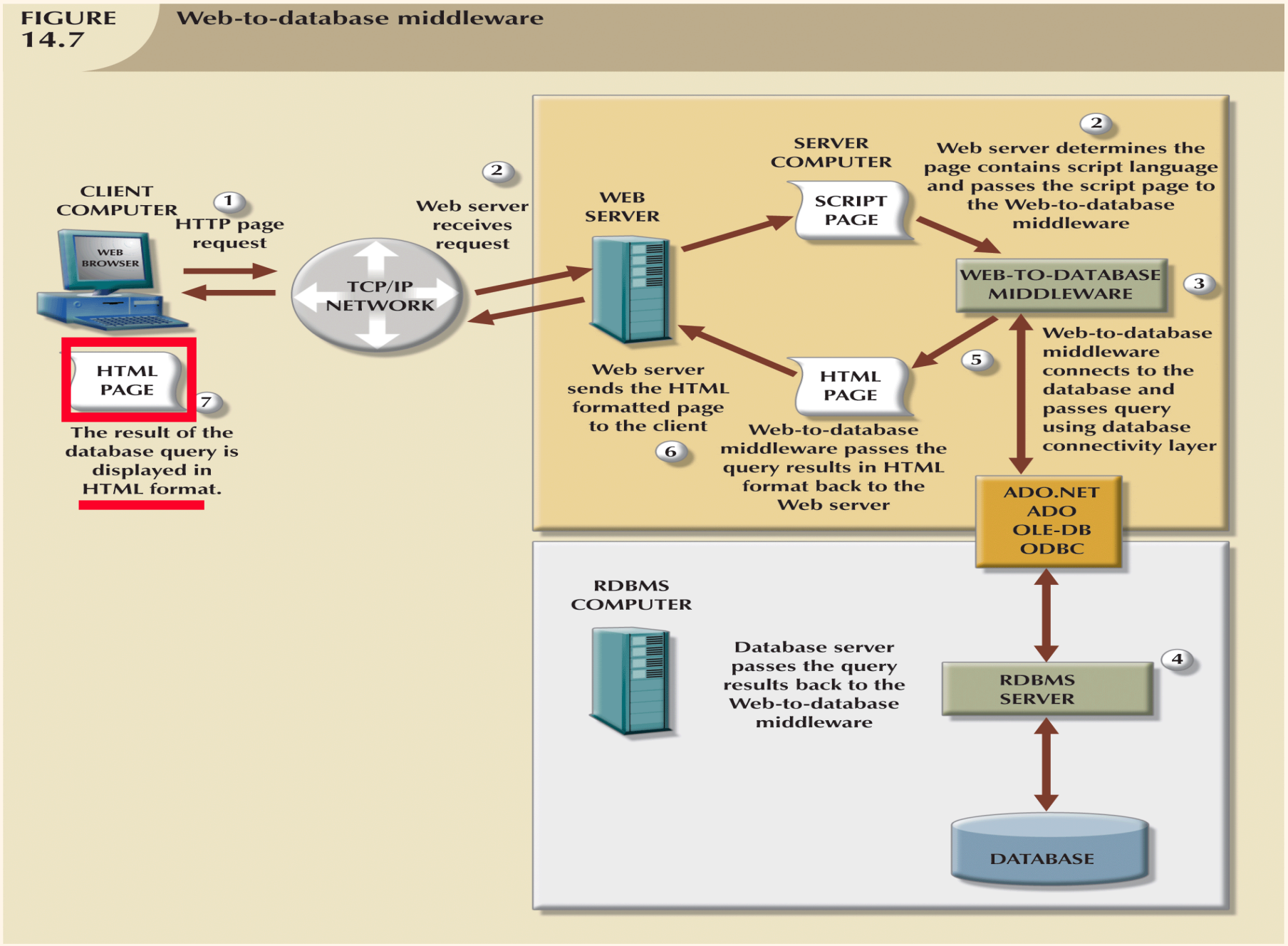
- Any state information (shopping carts, user login-information) need to be encoded in every HTTP **request** and **response**!
- Popular methods on how to maintain state:
  - Cookies
  - Dynamically generate unique URL's at the server level

Overview

- Internet Concepts
- Web **data** formats
  - HTML, XML, DTDs
- Introduction to three-tier architectures
- The presentation layer
  - HTML forms; HTTP Get and POST, URL encoding; Javascript; Stylesheets; XSLT
- The middle tier
  - CGI, application servers, Servlets, JavaServerPages, ASP, PHP, passing arguments, maintaining state (cookies)

Web Database Application Development

FIGURE 14.7 Web-to-database middleware



*Web **Data** Formats*

- HTML (HyperText Markup Language)
  - The **Presentation - View (V)** Language for the Internet

- XML (Extensible Markup Language)
  - A self-describing, hierarchal **data** model (**M**)
  - DTD (Document Type Definition)
    - Standardizing schemas for **XML**

In 2001 Bill Gates called XML the "*lingua franca* of the Internet age"

HTML: An Example

```
<HTML>
  <HEAD></HEAD>
  <BODY>
  <h1>Barns and Nobble Internet Bookstore</h1>
  Our inventory:

  <h3>Science</h3>
  <b>The Character of Physical Law</b>
  <UL>
    <LI>Author: Richard Feynman</LI>
    <LI>Published 1980</LI>
    <LI>Hardcover</LI>
  </UL>
  <h3>Fiction</h3>
  <b>Waiting for the Mahatma</b>
  <UL>
    <LI>Author: R.K. Narayan</LI>
    <LI>Published 1981</LI>
  </UL>
  <b>The English Teacher</b>
  <UL>
    <LI>Author:  R.K. Narayan</LI>
    <LI>Published 1980</LI>
    <LI>Paperback</LI>
  </UL>
  </BODY>
</HTML>
```

HTML: Sample Commands

- <HTML>:
- <UL>: unordered list
- <LI>: list entry
- <h1>: largest heading
- <h2>: second-level heading, <h3>, <h4> analogous
- <B>Title</B>: Bold

XML: An Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BOOKLIST>
  <BOOK genre="Science" format="Hardcover">
    <AUTHOR>
      <FIRSTNAME>Richard</FIRSTNAME><LASTNAME>Feynman</LASTNAME>
    </AUTHOR>
    <TITLE>The Character of Physical Law</TITLE>
    <PUBLISHED>1980</PUBLISHED>
  </BOOK>
  <BOOK genre="Fiction">
    <AUTHOR>
      <FIRSTNAME>R.K.</FIRSTNAME><LASTNAME>Narayan</LASTNAME>
    </AUTHOR>
    <TITLE>Waiting for the Mahatma</TITLE>
    <PUBLISHED>1981</PUBLISHED>
  </BOOK>
  <BOOK genre="Fiction">
    <AUTHOR>
      <FIRSTNAME>R.K.</FIRSTNAME><LASTNAME>Narayan</LASTNAME>
    </AUTHOR>
    <TITLE>The English Teacher</TITLE>
    <PUBLISHED>1980</PUBLISHED>
  </BOOK>
</BOOKLIST>
```

DTD – Document Type Definition

- A **DTD** is a **Schema (type)** for **XML data**
- **(type variable) (int x)**

- **XML** protocols and Languages can be standardized with **DTD** files

- A **DTD** says what Elements and Attributes are required or optional
  - Defines the **formal structure** of the Language

DTD

```
<!DOCTYPE BOOKLIST [
<!ELEMENT BOOKLIST (BOOK)*>
    <!ELEMENT BOOK (AUTHOR,TITLE,PUBLISHED?)>
        <!ELEMENT AUTHOR (FIRSTNAME,LASTNAME)>
            <!ELEMENT FIRSTNAME (#PCDATA)>
            <!ELEMENT LASTNAME (#PCDATA)>
        <!ELEMENT TITLE (#PCDATA)>
        <!ELEMENT PUBLISHED (#PCDATA)>
    <!ATTLIST BOOK GENRE (Science|Fiction) #REQUIRED>
    <!ATTLIST BOOK FORMAT (Paperback|Hardcover) "Paperback">
]>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BOOKLIST>
  <BOOK genre="Science" format="Hardcover">
    <AUTHOR>
      <FIRSTNAME>Richard</FIRSTNAME><LASTNAME>Feynman</LASTNAME>
    </AUTHOR>
    <TITLE>The Character of Physical Law</TITLE>
    <PUBLISHED>1980</PUBLISHED>
  </BOOK>
```

Overview

- Internet Concepts
- Web data formats
  - HTML, XML, DTDs
- Introduction to three-tier architectures
- The presentation layer
  - HTML forms; HTTP Get and POST, URL encoding; Javascript; Stylesheets; XSLT
- The middle tier
  - CGI, application servers, Servlets, JavaServerPages, ASP, PHP, passing arguments, maintaining state (cookies)

*Components of **Data**-Intensive* Systems

Three separate types of functionality:

- **Data** management (**Model**)
- Application logic (**Controller**)
- Presentation (View)

- The **system architecture** determines whether these three components reside on a single system (**tier**) or are distributed across several **tiers**

The Three Layers (MVC)

Presentation tier (V)

- Primary interface to the User
- Needs to adapt to different display devices (PC, PDA, cell phone, voice access?)

Middle tier (C)

- Implements **business logic** (implements complex actions, maintains state between different steps of a workflow)
- Accesses different data management systems

Data Management tier (M)

- One or more standard **Database** Management Systems

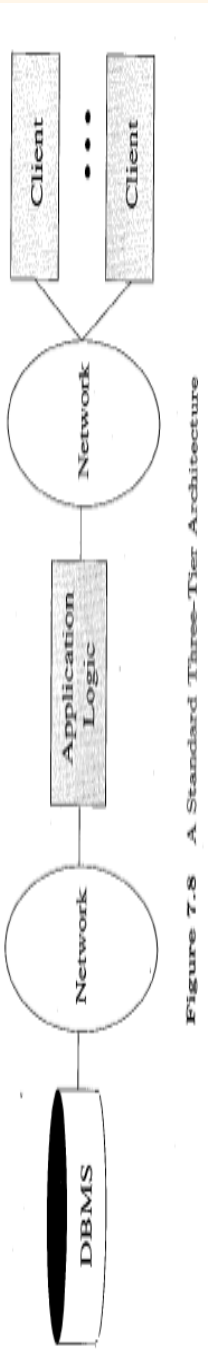


Figure 7.8 A Standard Three-Tier Architecture

3-tier Technologies (MVC)

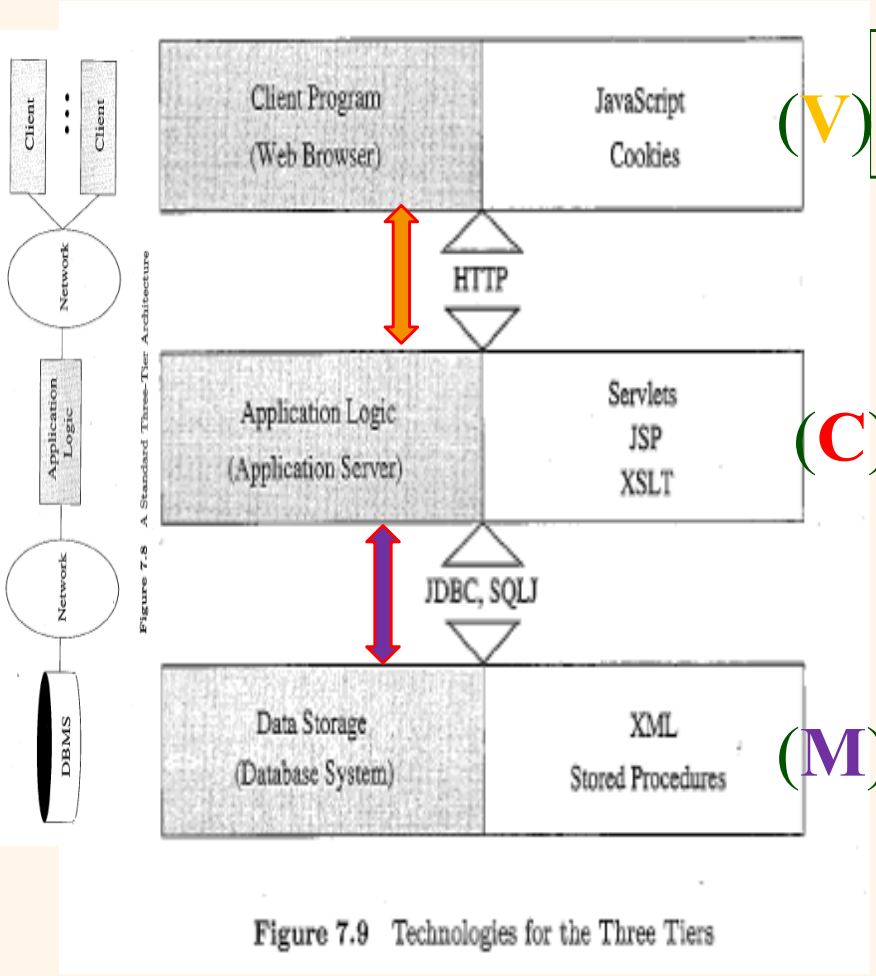


Figure 7.9 Technologies for the Three Tiers

(V) Client
(Web Browser)

HTML
Javascript
CSS
XSLT
Cookies

(C) Application Server
(Tomcat, Apache, IIS)

JSP
Servlets
CGI
ASP.NET
PHP

(M) Database Server
(DB2)

XML

Stored Procedures

Overview

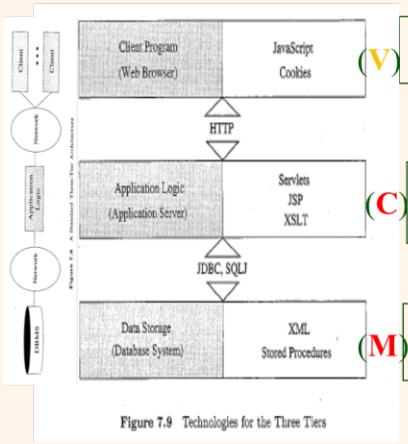
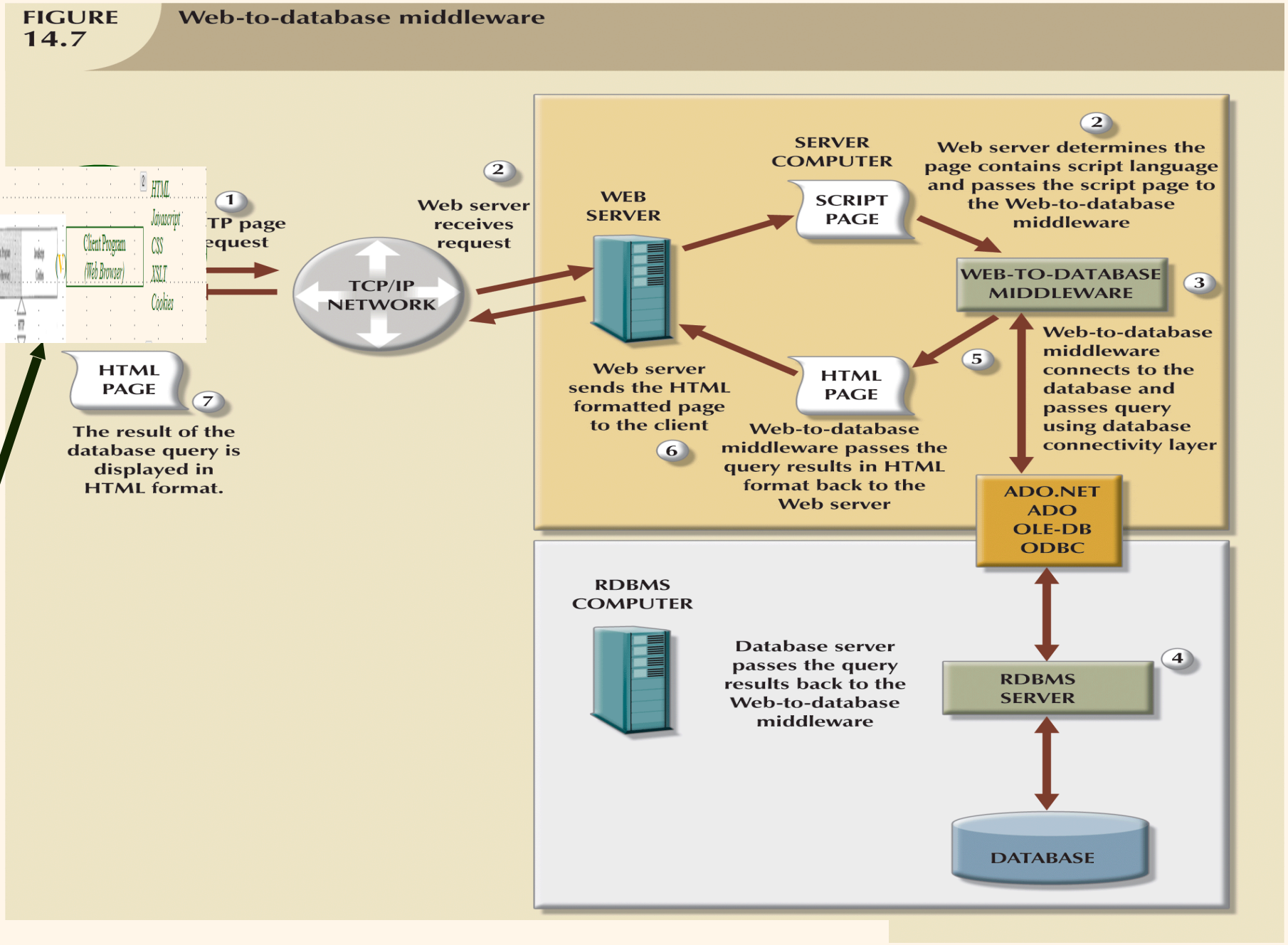


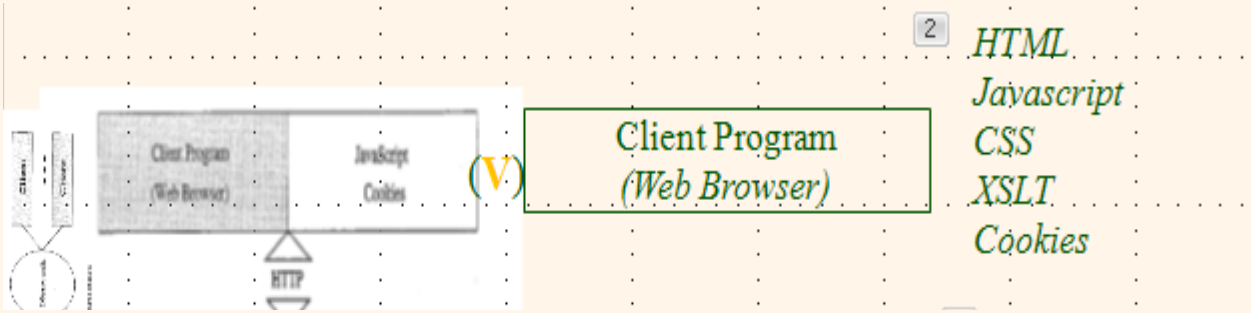
Figure 7.9 Technologies for the Three Tiers

- Internet Concepts
- Web data formats
  - HTML, XML, DTDs
- Introduction to three-tier architectures
- The **Presentation** layer
  - HTML forms (**Input Forms**) and human-readable output (**Output Reports**) ; HTTP Get and POST, URL encoding; Javascript; Stylesheets; XSLT
- The middle tier
  - CGI, application servers, Servlets, JavaServerPages, ASP, PHP, passing arguments, maintaining state (cookies)

Web Database Application Development (MVC)



Overview of the Presentation Tier (V)



Recall: Functionality of the **Presentation** tier

- Primary interface to the **User**
- Needs to adapt to different display devices (PC, PDA, cell phone, voice access?)
- Simple functionality, such as field validity checking

We will cover:

- **HTML Forms (Input Forms)** : How to pass **data** to the middle tier
- **JavaScript** (ECMA): Simple functionality at the **Presentation** tier
- **Style sheets**: Separating **data** from formatting

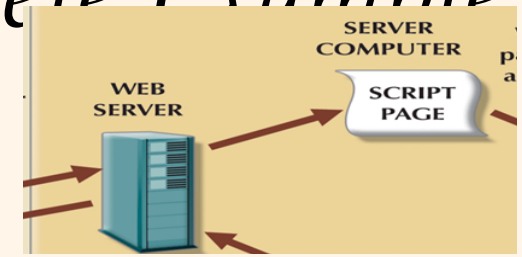
HTML Forms (Input Forms)

- input tag
  - Attributes:
    - type: text (text input field), password (text input field where input is, reset (resets all input fields), submit (submit Form parameters)
    - name: symbolic name, used to identify field value at the middle tier (C)
    - value: default value

- Example form:

```
<form method="POST" action="TableOfContents.jsp">
    <input type="text"      name="userid">
    <input type="password"  name="password">
    <input type="submit"    name="submit"     value="Login" >
    <input type="reset"                       value="Clear">
</form>
```

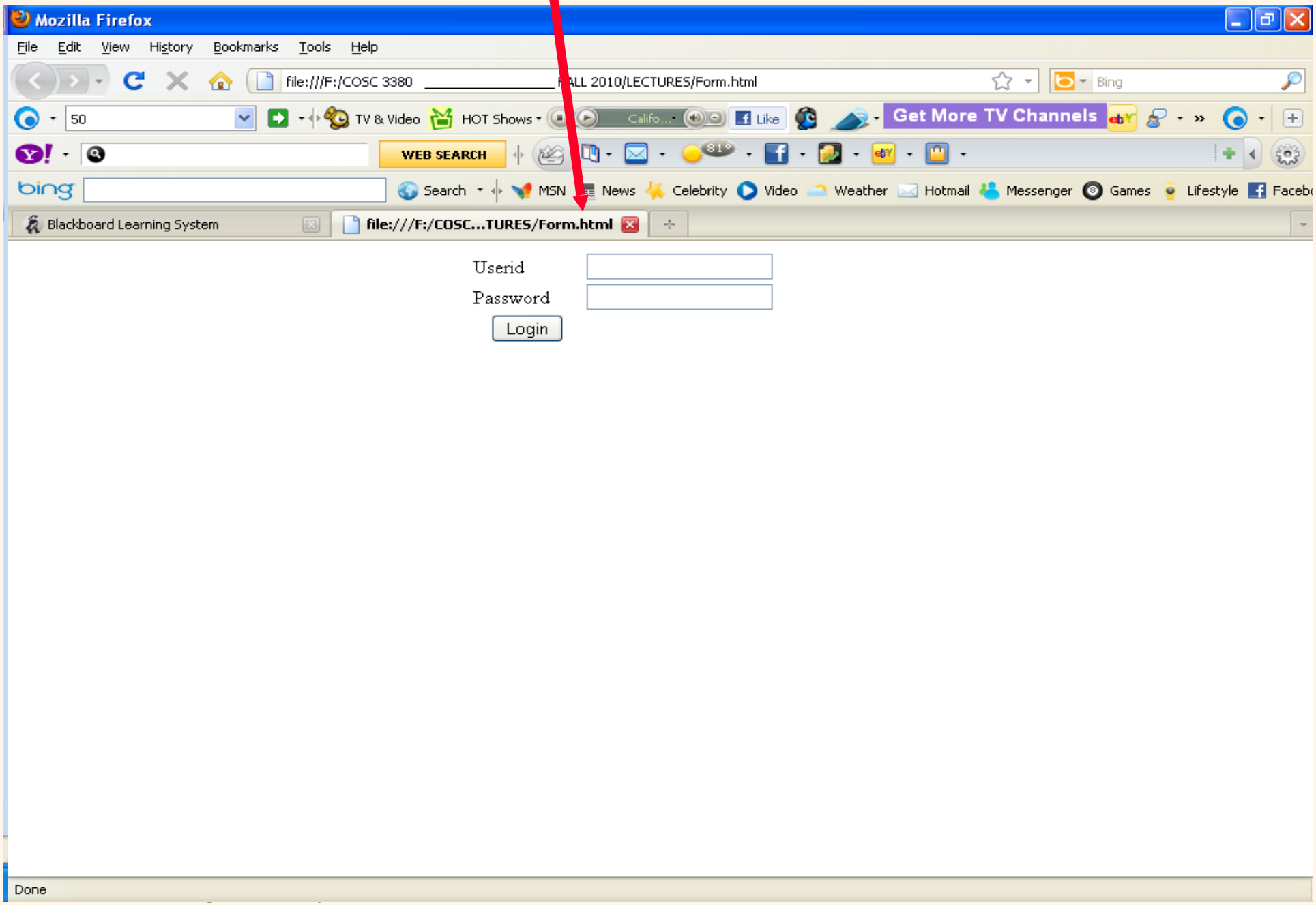
HTML Forms (Input Forms) : A Complete Example




```
form method="POST" action="TableOfContents.jsp">
  <table align = "center" border="0" width="300">
  <tr>
        <td>Userid</td>
        <td><input type="text" name="userid" size="20"></td>
  </tr>
  <tr>
        <td>Password</td>
        <td><input type="password" name="password" size="20"></td>
  </tr>
  <tr>
        <td align = "center"><input type="submit" name="submit " value="Login"></td>
  </tr>
  </table>
/form>
```

Place this in a Form.html
Open Form.html with a browser!

HTML Forms: A Complete Example



Hey, what's another programming language?!

Goal: Add functionality to the Presentation tier. (V)

Sample Applications:

- Detect browser type and load browser-specific page
- Form validation: Validate Form input fields
- Browser control: Open new windows, close existing windows (example: pop-up ads)

Usually embedded directly inside the HTML with the <SCRIPT> … </SCRIPT> tag.

<SCRIPT> tag has several attributes:

- LANGUAGE: specifies language of the script (such as javascript)
- SRC: external file with script code
- Example:

```
<SCRIPT LANGUAGE="JavaScript" SRC="validate.js>
</SCRIPT>
```

JavaScript

- If <SCRIPT> tag does not have a SRC attribute, then the **JavaScript** is directly in the **HTML** file.

- Example:

```
<SCRIPT LANGUAGE="JavaScript">
        <!-- alert("Welcome to our bookstore") //-->
</SCRIPT>
```

- Two different commenting styles
  - <!-- comment for **HTML**, since the following **JavaScript** code should be ignored by the **HTML** processor
  - // comment for **JavaScript** in order to end the **HTML** comment

JavaScript

```
<SCRIPT language="javascript">
function testLoginEmpty(loginForm)
{
 if ((loginForm.userid.value == "") ||(loginForm.password.value == ""))
 {
   alert('Please enter values for userid and password.');
   return false;
 }
 else
   return true;
}
</SCRIPT>
```

JavaScript is a complete scripting language

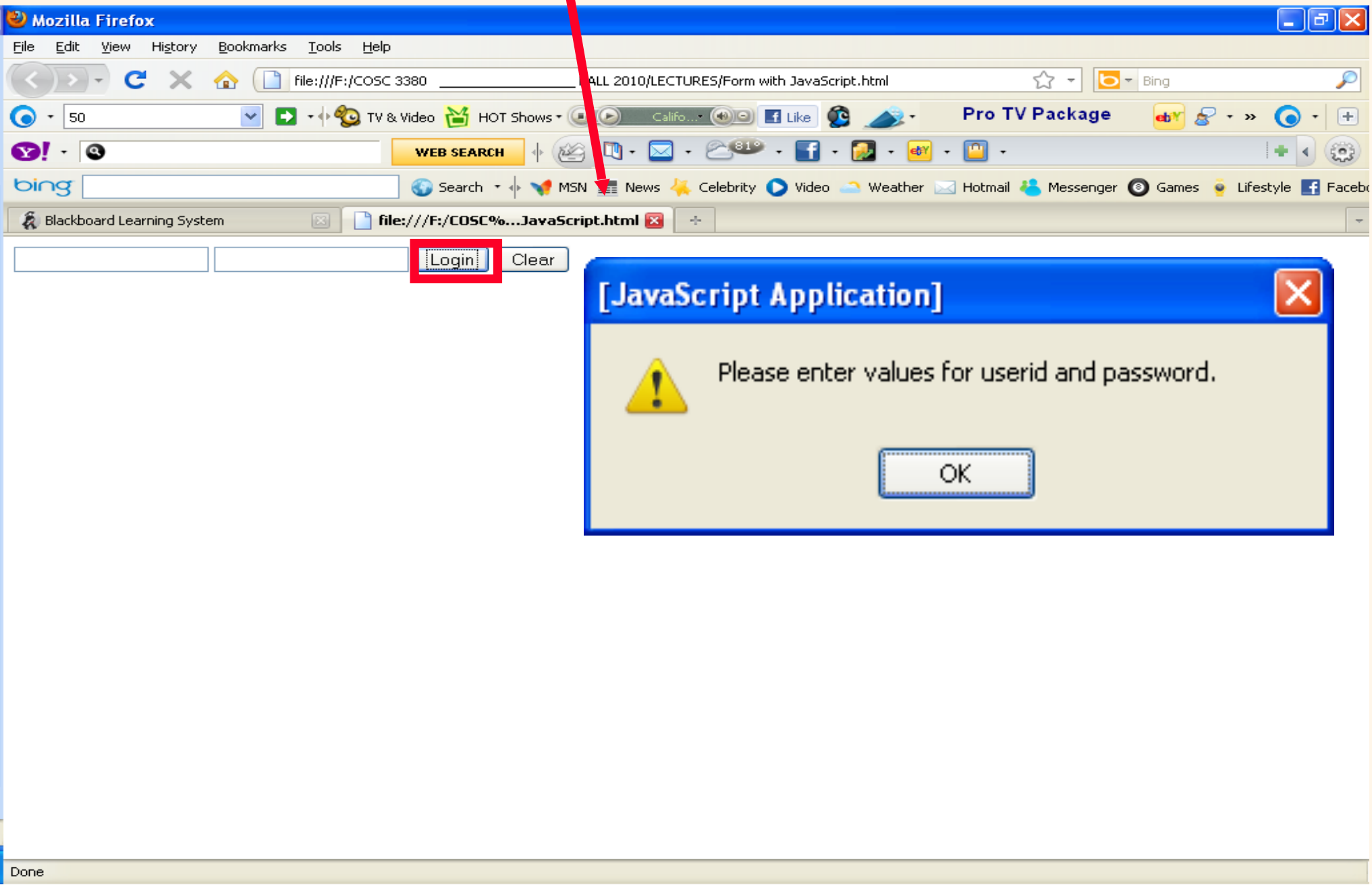
- Variables
- Assignments (=, +=, …)
- Comparison operators (<,>,…), boolean operators (&&, ||, !)
- Statements
  - if (condition) {statements;} else {statements;}
  - for loops, do-while loops, and while-loops
- Functions with return values
  - Create functions using the function keyword
  - f(arg1, …, argk) {statements;}

JavaScript: A Complete Example

```
<HTML>
<HEAD>

      <SCRIPT language="javascript">
      function testLoginEmpty(loginForm)
      {
       if ((loginForm.userid.value == "") ||(loginForm.password.value == ""))
       {
          alert('Please enter values for userid and password.');
          return false;
       }
       else
          return true;
      }
      </SCRIPT>

</HEAD>

<BODY>
      <form method="POST"  action="TableOfContents.jsp">
        <input type="text"           name="userid">
        <input type="password"    name="password">
        <input type="button"       name="submit"       value="Login"   onClick="testLoginEmpty(this.form)">
        <input type="reset"                                value="Clear">
      </form>

</BODY>
</HTML>
```

Place this in a Form with JavaScript.html
Open it with a browser!

JavaScript: A Complete Example



Java Script

Click the buttons. Try adding three more buttons for ratings 3, 2, and 1. Try replacing the * with a star (★). To specify a star, use ★, which is the HTML entity for displaying a star.

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta charset="UTF-8">
      <title>Kyoto Kaiseki Restaurant Review</title>
      <script>
         function updateRating(newRating) {
            let star1 = document.getElementById("rating1");
            let star2 = document.getElementById("rating2");
            let star3 = document.getElementById("rating3");
            let star4 = document.getElementById("rating4");
            let star5 = document.getElementById("rating5");

            if (newRating == 5) {
               star5.style.color = "blue";
               star4.style.color = "blue";
               star3.style.color = "blue";
               star2.style.color = "blue";
               star1.style.color = "blue";
            }
            else if (newRating == 4) {
               star5.style.color = "lightgray";
               star4.style.color = "blue";
               star3.style.color = "blue";
               star2.style.color = "blue";
               star1.style.color = "blue";
            }
         }
      </script>
   </head>
   <body>
      <h1>Kyoto Kaiseki Restaurant Review</h1>
      <p><strong>Rating:</strong>
         <span id="rating1">*</span>
         <span id="rating2">*</span>
         <span id="rating3">*</span>
         <span id="rating4">*</span>
         <span id="rating5">*</span>
      </p>

      <p><strong>Update rating:</strong>
         <button type="button" onclick="updateRating(4)">Rate 4</button>
         <button type="button" onclick="updateRating(5)">Rate 5</button>
      </p>
```

Kyoto Kaiseki Restaurant Review

Rating: * * * * *

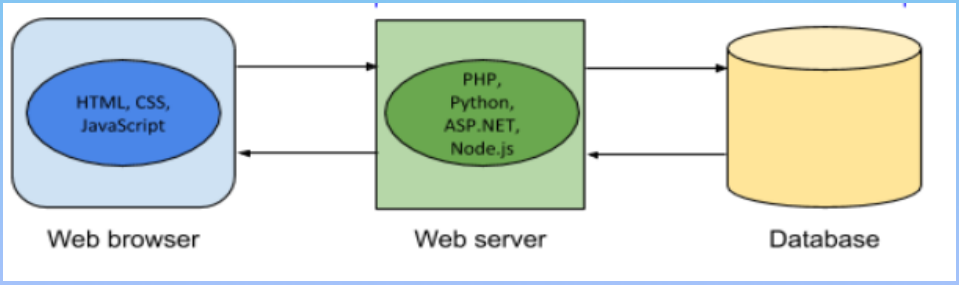
Update rating: Rate 4 Rate 5

Favorite dish: Mixed sashimi

7. JavaScript in the Browser

7.8 JavaScript Object Notation (JSON)

Communicating data between the server and browser is a significant task for modern web applications. Initial attempts to do so included unstructured text documents and heavily structured XML documents, both of which required significant effort to convert to a usable format. **JavaScript Object Notation**, or **JSON**, is an efficient, structured format for data based on a subset of the JavaScript language. JSON (pronounced "Jason") is intended to be easily readable by humans and computers. Debugging communication that uses JSON is easy because humans can read JSON. Communication is efficient because computers can transmit and parse JSON quickly. As a result, JSON has rapidly become the dominant format of data transfer between web browsers and servers.



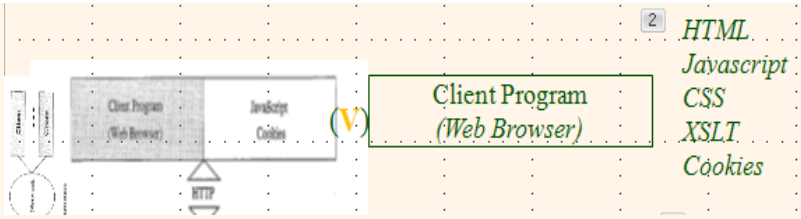
PARTICIPATION ACTIVITY | 11.3.2: XML format.

```
<?xml version = "2.0" encoding = "UTF-16"?>
<Customer>
    <Name>Maria Rodriguez</Name>
    <Vehicle>
        <Make>Ford</Make>
        <Model>F-150</Model>
        <Year>2008</Year>
    </Vehicle>
    <Vehicle>
        <Make>Toyota</Make>
        <Model>Camry</Model>
        <Year>2019</Year>
    </Vehicle>
    <Budget></Budget>
    <PreviousCustomer>true</PreviousCustomer>
    <FamilyMember>Jose</FamilyMember>
    <FamilyMember>Felicia</FamilyMember>
    <FamilyMember>Isabella</FamilyMember>
    <Notes>Shopping for a new sports car. Interested in leasing.</Notes>
</Customer>
```

PARTICIPATION ACTIVITY | 11.3.4: JSON format.

```
{
  "Customer":
  {
    "Name": "Maria Rodriguez",
    "Vehicle"
    [
        { "Make": "Ford", "Model": "F-150", "Year": 2008 },
        { "Make": "Toyota", "Model": "Camry", "Year": 2019 }
    ],
    "Budget": null,
    "PreviousCustomer": true,
    "FamilyMember": [ "Jose", "Felicia", "Isabella" ],
    "Notes": "Shopping for a new sports car. Interested in leasing."
  }
}
```

Stylesheets



- Idea: Separate **display** from **contents**, and adapt display to different **Presentation** formats

- Two aspects:
  - Document transformations to decide what parts of the document to **display** in what order
  - Document rending to decide how each part of the document is **displayed**

- Why use stylesheets?
  - **REUSE** of the same document for different **displays**
  - Tailor **display** to **User**'s preferences
  - **REUSE** of the same document in different contexts

- Two Stylesheet languages
  - **C**ascading **S**tyle **S**heets (CSS): for **HTML** documents
  - E**X**tensible **S**tyle Sheet **L**anguage (XSL): for **XML** documents

CSS: Cascading Style Sheets

- Defines how to display **HTML** documents
- Many **HTML** documents can refer to the same CSS
  - Can change format of a website by changing a single **S**tyle **S**heet
  - Example:
    <LINK REL="style sheet" TYPE="text/css" HREF="**books.css**" />

Each line in **CSS** consists of three parts:

selector {property: value}

- **selector**: **Tag** whose format is defined
- **property**: Tag's **Attribute** whose value is set
- **value**: value of the **Attribute**

```
<HTML>
  <HEAD></HEAD>
  <BODY>
  <h1>Barns and Nobble Internet Bookstore</h1>
  Our inventory:

  <h3>Science</h3>
  <b>The Character of Physical Law</b>
  <UL>
    <LI>Author: Richard Feynman</LI>
    <LI>Published 1980</LI>
    <LI>Hardcover</LI>
  </UL>
   <h3>Fiction</h3>
  <b>Waiting for the Mahatma</b>
  <UL>
    <LI>Author: R.K. Narayan</LI>
    <LI>Published 1981</LI>
  </UL>
  <b>The English Teacher</b>
  <UL>
    <LI>Author: R.K. Narayan</LI>
    <LI>Published 1980</LI>
    <LI>Paperback</LI>
  </UL>
  </BODY>
</HTML>
```

CSS: Cascading Style Sheets

Each line in CSS consists of three parts:

selector {property: value}

- **selector**: Tag whose format is defined
- **property**: Tag's **Attribute** whose value is set
- **value**: value of the **Attribute**

Example Style Sheet:

```
body {background-color: yellow}
h1 {font-size: 36pt}
h3 {color: blue}
p {margin-left: 50px; color: red}
```

The first line has the same effect as:

<body background-color="yellow">

```
<HTML>
  <HEAD></HEAD>
  <BODY>
  <h1>Barns and Nobble Internet Bookstore</h1>
  Our inventory:
  <h3>Science</h3>
  <b>The Character of Physical Law</b>
  <UL>
    <LI>Author: Richard Feynman</LI>
    <LI>Published 1980</LI>
    <LI>Hardcover</LI>
  </UL>
  <h3>Fiction</h3>
  <b>Waiting for the Mahatma</b>
  <UL>
    <LI>Author: R.K. Narayan</LI>
    <LI>Published 1981</LI>
  </UL>
  <b>The English Teacher</b>
  <UL>
    <LI>Author: R.K. Narayan</LI>
    <LI>Published 1980</LI>
    <LI>Paperback</LI>
  </UL>
  </BODY>
</HTML>
```

CSS: Cascading Style Sheets

Figure 2.6.1: HTML code without and with CSS rules.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>For sale: 2012 Ducati Streetfighter</title>
  </head>
  <body>
    <h1>Ducati Streetfighter - $9000</h1>
    <p>year: <strong>2012</strong></p>
    <p>make and model:
    <strong>Ducati Streetfighter 848</strong></p>
    <p>condition: <strong>excellent</strong></p>
    <p>odometer: <strong>9500</strong></p>
  </body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>For sale: 2012 Ducati Streetfighter</title>
    <style>
    h1 {
      color: green;
      background-color: lightgray;
    }
    p {
      font-family: arial;
      margin-left: 10px;
    }
    strong {
      background-color: lightgreen;
      padding: 5px;
    }
    </style>
  </head>
  <body>
    <h1>Ducati Streetfighter - $9000</h1>
    <p>year: <strong>2012</strong></p>
    <p>make and model:
    <strong>Ducati Streetfighter 848</strong></p>
    <p>condition: <strong>excellent</strong></p>
    <p>odometer: <strong>9500</strong></p>
  </body>
</html>
```

Ducati Streetfighter - $9000

Ducati Streetfighter - $9000

XSL - EXtensible Style Sheet Language

- Language for expressing **S**tyle **S**heets.
- It is, like **CSS**, a file that describes how to **display** an **XML** document **of a given type** **DTD**.
- It exceeds the capabilities of **CSS**.
- It contains the XSL **Transformation** Language, or **XSLT**, a Language that allows us to **transform** the input **XML** document into a **XML** document with **another structure**.

Web Database Application Development

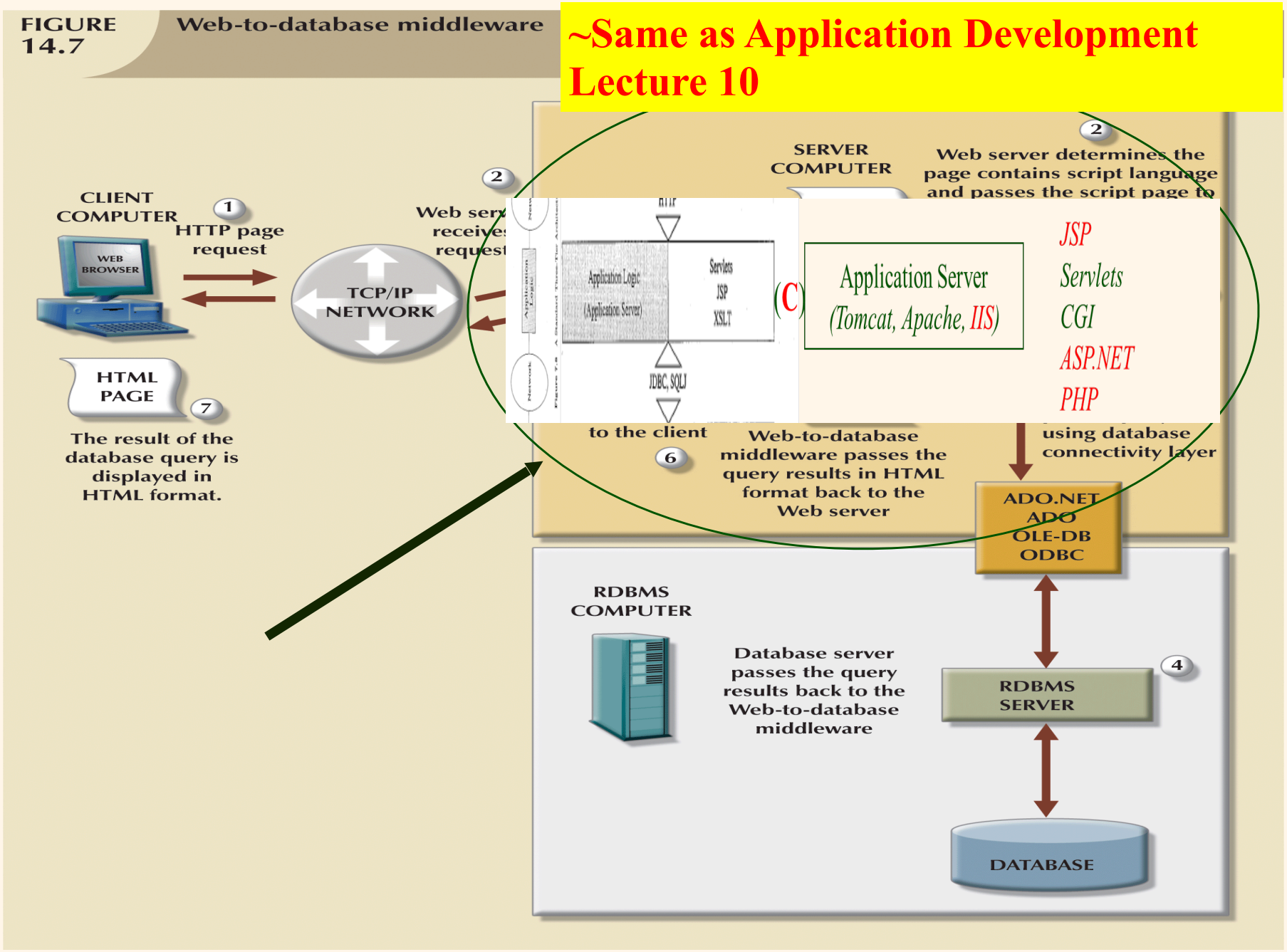
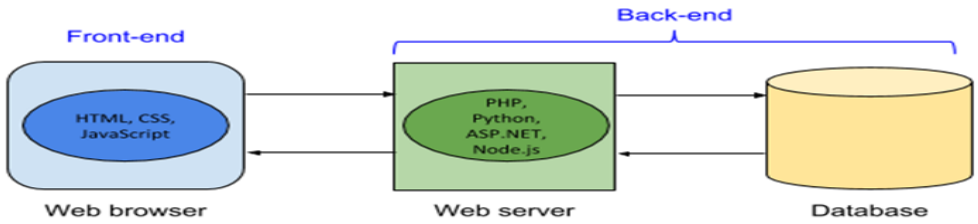


Figure 11.1.1: Front-end and back-end technologies.



Server-side programming

Web developers have a wide range of options when choosing a server-side programming platform or language. When choosing a server-side programming platform, developers must consider:

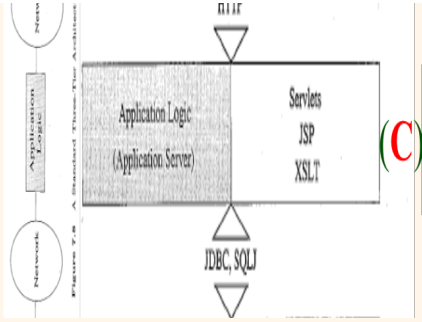
- Server platform: Some web servers support certain languages and not others. Ex: IIS supports ASP.NET, and Apache supports PHP.
- Tool support: Some tools are ideal for working with certain programming languages. Ex: PhpStorm is ideal for PHP development, and Visual Studio is ideal for ASP.NET.
- Developer experience: JavaScript developers may choose Node.js instead of learning a new language like C#. Developers who are new to web development might already know Java or Python and prefer those languages.
- Library support: Some languages may have pre-built libraries that support some web applications better than others.

(C) Application Server *(Tomcat, Apache, IIS)*

JSP
Servlets
CGI
ASP.NET
PHP

Node.js

Overview



(C)

Application Server
(Tomcat, Apache, IIS)

JSP
Servlets
CGI
ASP.NET
PHP

- Internet Concepts
- Web data formats
  - HTML, XML, DTDs
- Introduction to three-tier architectures
- The presentation layer
  - HTML forms; HTTP Get and POST, URL encoding; Javascript; Styleshe XSLT
- The middle tier
  - Common Gateway Interface (CGI),
  - Application Servers,
  - Servlets,
  - Java Server Pages (JSP),
  - Active Server Pages (ASP),
  - PHP
  - passing arguments,
  - maintaining state (cookies)

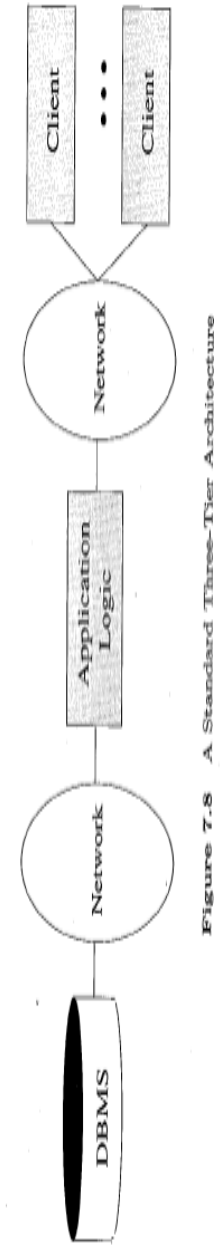
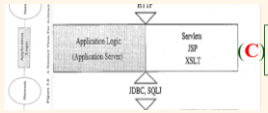


Figure 7.8 A Standard Three-Tier Architecture

*Overview of the **Middle** Tier*



(C) Application Server *(Tomcat, Apache, IIS)*

JSP
Servlets
CGI
ASP.NET
PHP

- Recall: Functionality of the **Middle** Tier
  - Encodes business logic (C)
  - Connects to **database system(s)** (M)
  - Accepts **Form input** from the **Presentation** tier (V)
  - Generates **output** for the **Presentation** tier
- We will cover
  - CGI: Protocol for passing arguments to programs running at the **middle** tier
  - Application Servers: Runtime environment at the middle tier
  - Servlets: Java programs at the **middle** tier
  - JSP Java Server Pages: Java scripts at the **middle** tier
  - ASP: **A**synchronous **S**erver **P**age
  - PHP
  - Maintaining state: How to maintain state at the **middle** tier.

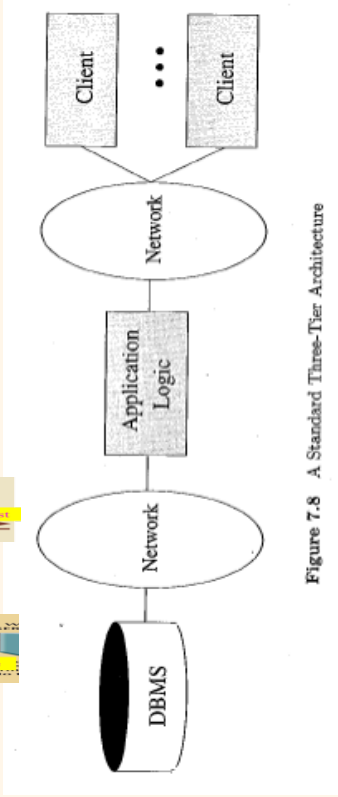
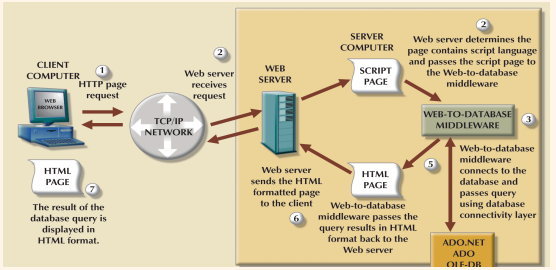


Figure 7.8 A Standard Three-Tier Architecture

Application Server: Process Structure (C)



- Main pool of threads of processes
- Manage connections (M)
- Enable access to heterogeneous data sources
- Other functionality such as **API**s for session management

(V)

Web Browser ⟷ Web Server

HTTP

(C)

Application Server

Pool of Servlets

JDBC ⟷ DBMS 1

ODBC ⟷ DBMS 2

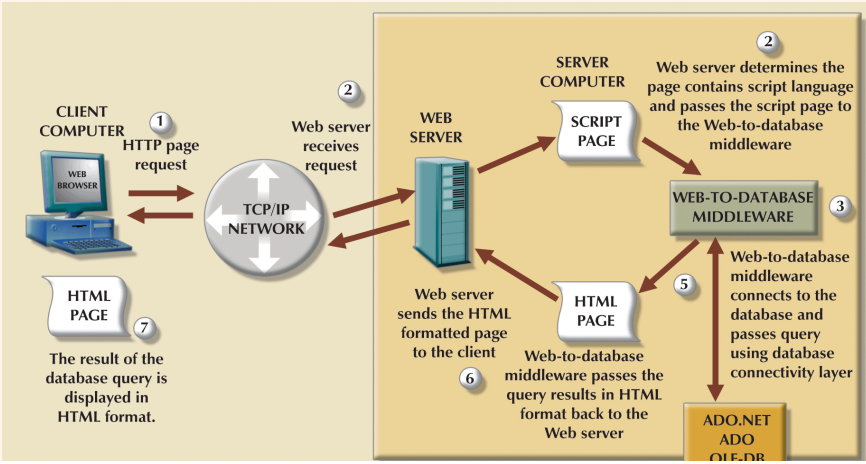
C++ Application

JavaBeans

(M)

Java Server Pages (JSP)

- Java Server Pages
  - Written in **HTML**, **JAVA** Code embedded in the **HTML**
  - Webpage first, Code second **HTML** first!



Java Server Pages: Example


```
<html>
<head><title>Welcome to B&N</title></head>
<body>
   <h1>Welcome back!</h1>
```

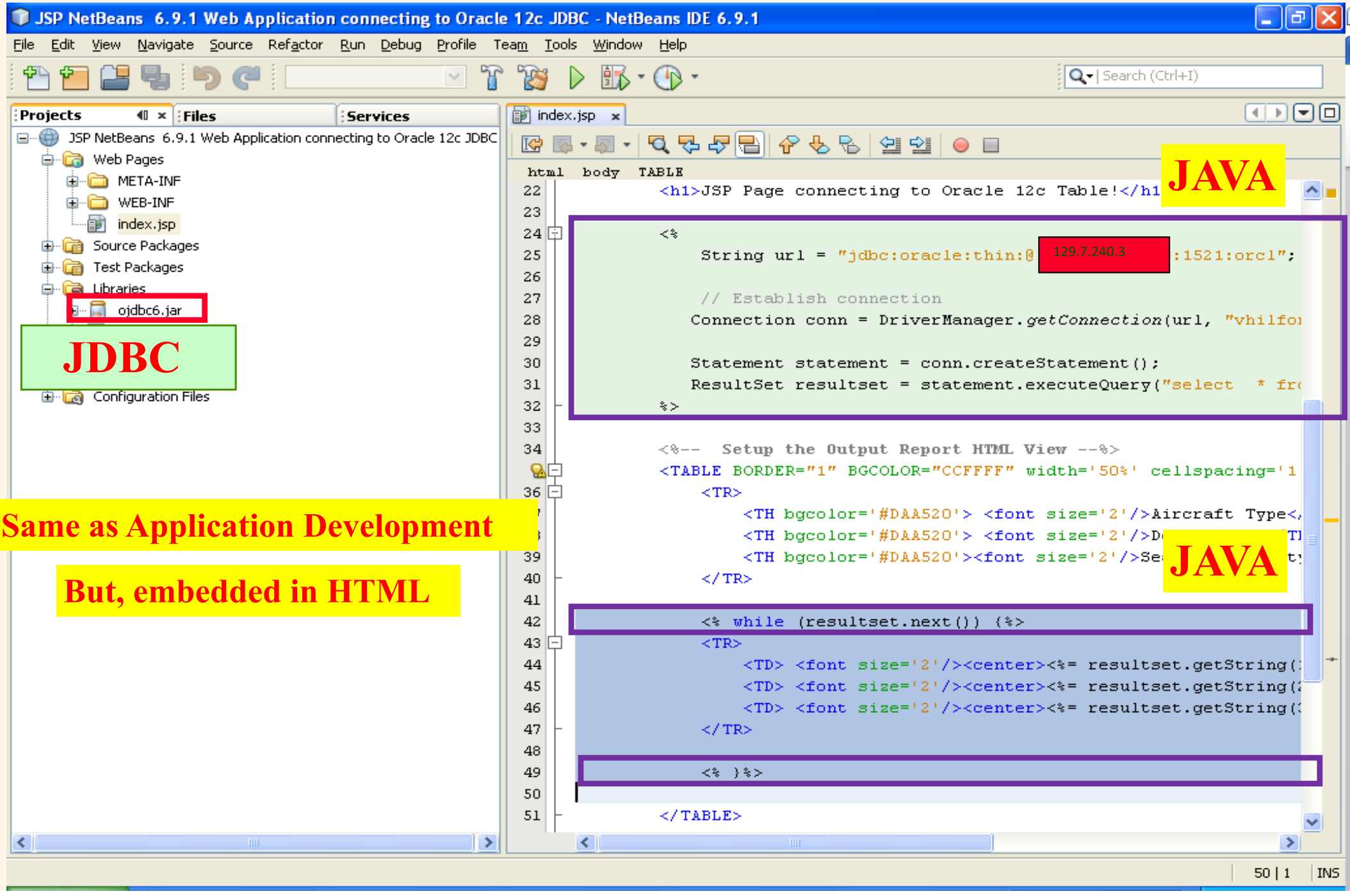
JAVA

```
   <% String name="NewUser";
         if (request.getParameter("username") != null)
         {
              name=request.getParameter("username");
         }
   %>
   You are logged on as user <%=name%>
```

JAVA

```
</body>
</html>
```

JSP to oracle 12c: A Complete Example

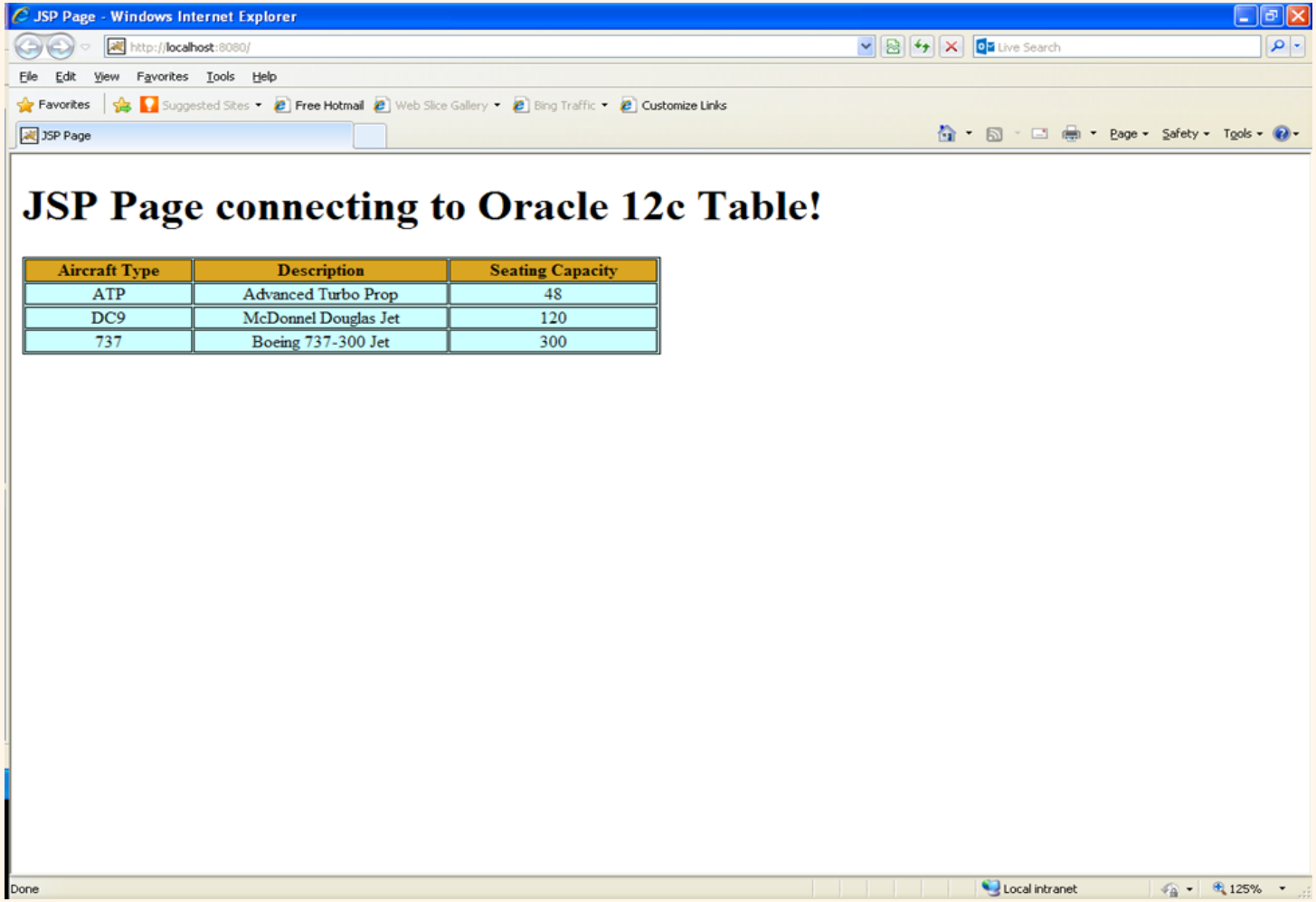


JSP to oracle 12c: A Complete Example

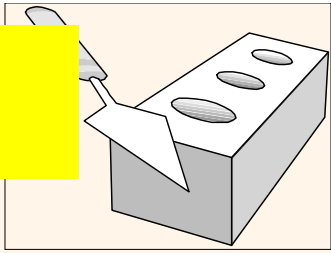
select * from VHILFORD.AI... ×

Page Size: 20 | Total Rows: 3 Page: 1 of 1 | Matching Rows:

#	AIRCRAFTTYPE	ADESCRIPTION	SEATCAPACITY
1	ATP	Advanced Turbo Prop	48
2	DC9	McDonnel Douglas Jet	120
3	737	Boeing 737-300 Jet	300

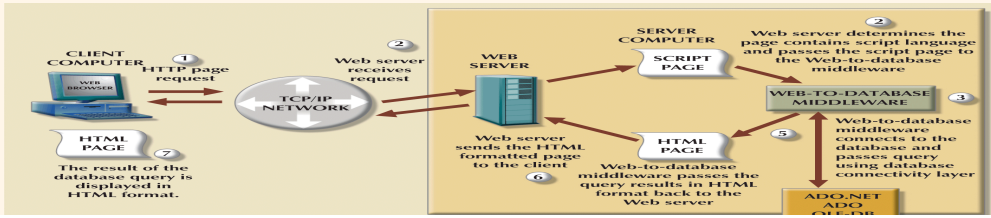


Active Server Pages (ASP)

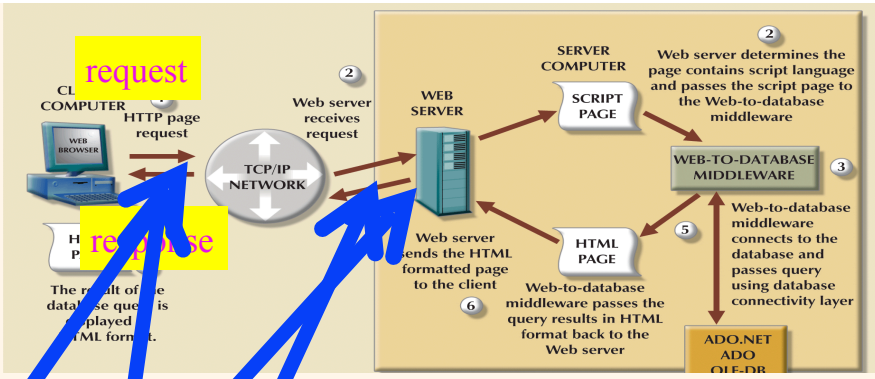


http://msdn.microsoft.com/en-us/library/ms972337.aspx

- Active Server Pages
- A series of **objects** and components that are executed *on the web server* IIS
- Uses a suite of technologies that allows **dynamically-generated** content
- Control of how content is generated *from* the **Server** to the **Browsers**



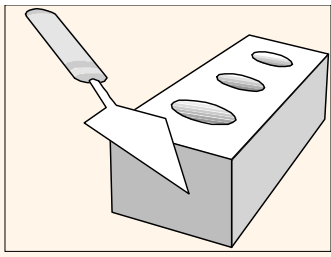
ASP Page Execution



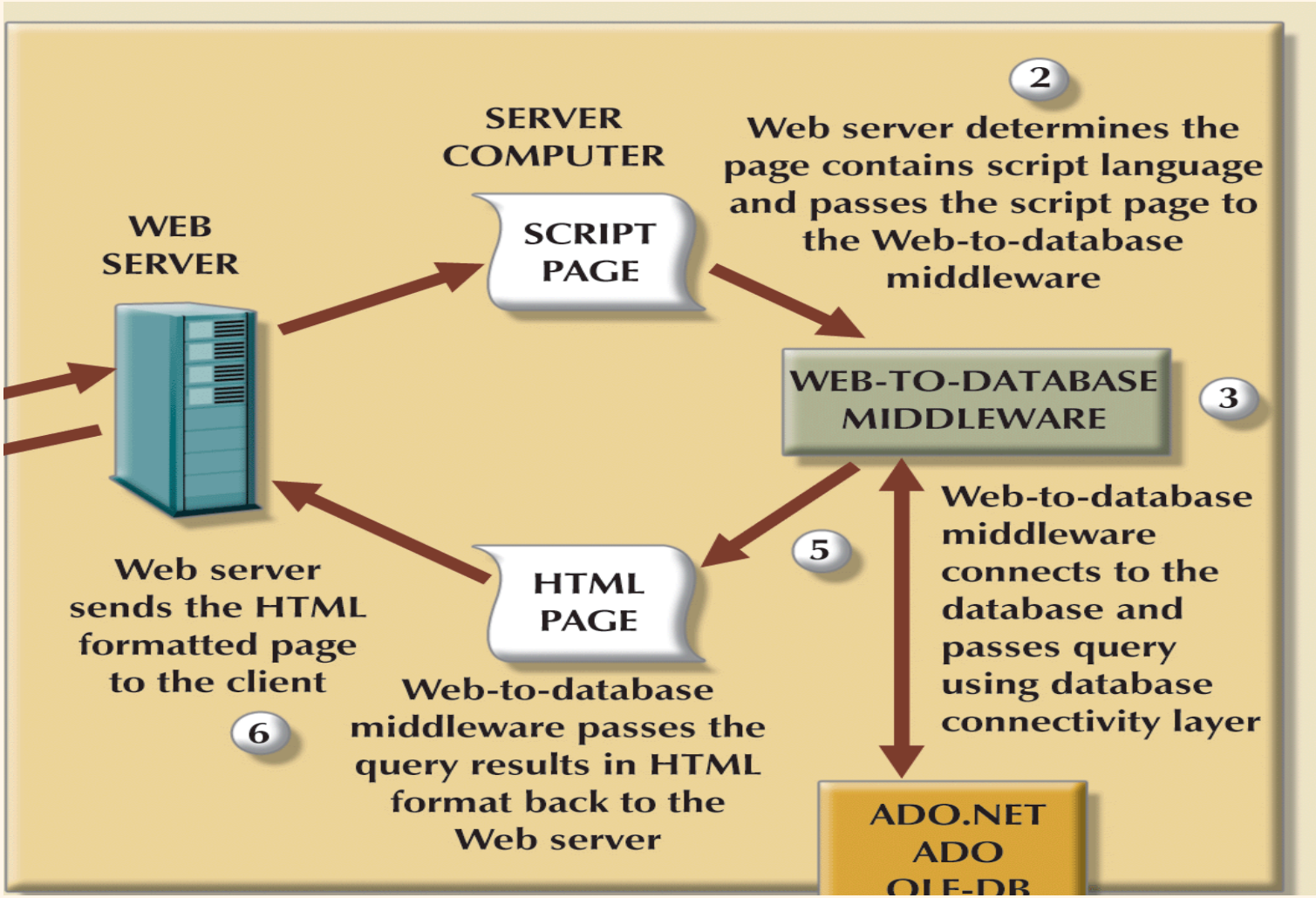
You Request an ASP Page

www.ocstc.org/default.aspx

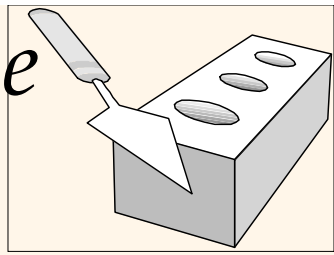
ASP Compared to Other Languages



- **ASP** and /**PHP**/Perl/**CGI**/**JSP** execution are roughly equivalent in execution sequence



ASP to MS SQL Server : A Complete Example



Project3ASP - Microsoft Visual Studio

File Edit View Qt Refactor Website Build Debug Team Data Tools Visual Assert Test Window Help

http://localhost:13202/Project3ASP/Default.aspx - Windows Internet Explorer provided by MSN & Bing

http://localhost:13202/Project3ASP/Default.aspx

File Edit View Favorites Tools Help

Google Search More » Sign In

Favorites Suggested Sites Free Hotmail Web Slice Gallery Bing Traffic

http://localhost:13202/Project3ASP/Default.aspx

Page Safety Tools

Log In

User Name: vhilford

Password: ••••••••••

Remember me next time.

Log In

vhilford mypassword

http://localhost:13202/Project3ASP/Home.aspx - Windows Internet Explorer provided by MSN & Bing

http://localhost:13202/Project3ASP/Home.aspx

File Edit View Favorites Tools Help

Google Search More » Sign In

Favorites Suggested Sites Free Hotmail Web Slice Gallery Bing Traffic

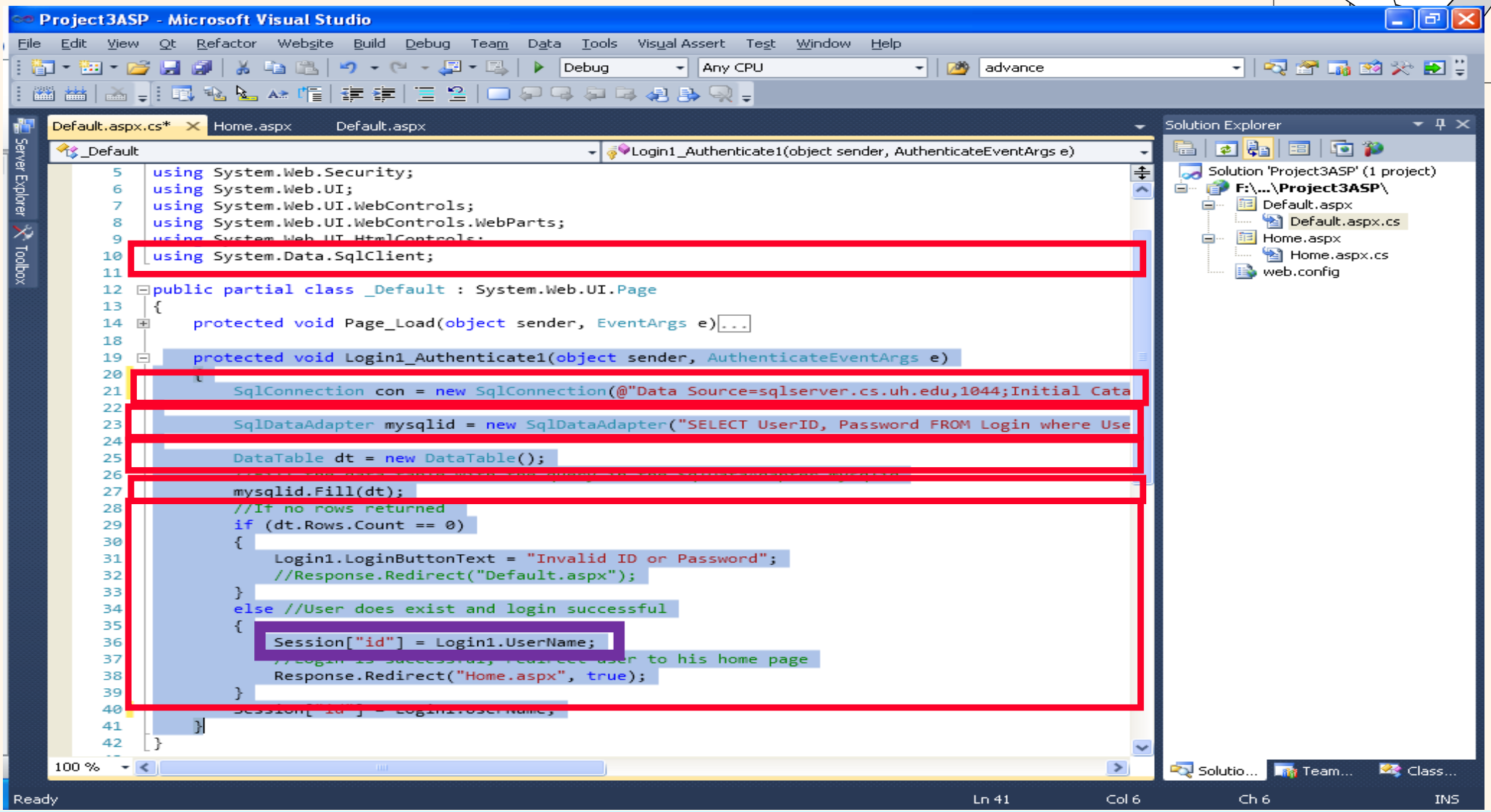
http://localhost:13202/Project3ASP/Home.aspx

Page Safety Tools

Welcome, you have SUCCESSFULLY logged in, vhilford !

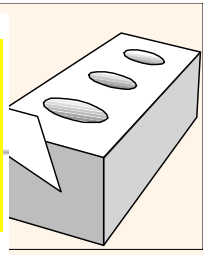
Ready Ln 41 Col 6 Ch 6 INS

ASP to MS SQL Server : A Complete Example



Default.aspx contains the user interface component of the application; the corresponding **Default.aspx.cs** file contains any Visual C# prgramming code that may be included.

Personal Home Pages (PHP)



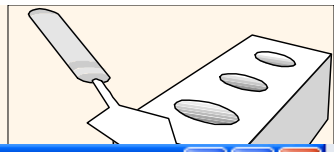
This article is about the scripting language. For other uses, see PHP (disambiguation).

PHP is a general-purpose server-side scripting language originally designed for Web development to produce dynamic Web pages. It is one of the first developed server-side scripting languages to be embedded into an HTML source document rather than calling an external file to process data. The code is interpreted by a Web server with a PHP processor module which generates the resulting Web page. It also has evolved to include a command-line interface capability and can be used in standalone graphical applications.[2] PHP can be deployed on most Web servers and also as a standalone shell on almost every operating system and platform free of charge.[3] A competitor to Microsoft's Active Server Pages (ASP) server-side script engine[4] and similar languages, PHP is installed on more than 20 million Web sites and 1 million Web servers.[5] Software that uses PHP includes MediaWiki, Joomla, Wordpress, Concrete5, MyBB, and Drupal.

PHP was originally created by Rasmus Lerdorf in 1995. The main implementation of PHP is now produced by The PHP Group and serves as the formal reference to the PHP language.[6] PHP is free software released under the PHP License, which is incompatible with the GNU General Public License (GPL) due to restrictions on the usage of the term *PHP*.[7]

While PHP originally stood for *Personal Home Page*, it is now said to stand for *PHP: Hypertext Preprocessor*, a recursive acronym.[8]

PHP to mySQL Server : A Complete Example



Project3PHP - NetBeans IDE 6.9.1

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default>

Search (Ctrl+I)

P... Files Services

Project3PHP
- Source Files
  - index.php
- Include Path

index.php

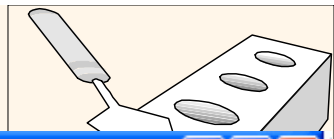
html head title

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2  <html>
3      <head>
4          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5          <title>Project3PHP</title>
6      </head>
7      <body>
8          <?php
9          error_reporting(E_ALL); //turns on all error reporting, useful for debugging
10
11         class GuestbookDb {
12
13             function __construct($db, $user='root', $password='', $host='localhost') {
14                 mysql_connect($host, $user, $password);
15                 mysql_select_db($db);
16                 //mysql_query('delete from address');
17             }
18
19             function addRecord($name, $note) {
20                 $query = 'insert into guests values ("'.$name.'", "' .$note.'")'; //insert
21                 mysql_query($query);
22             }
23
24             function printRecords() {
25                 $query = 'select * from guests'; //select all records
26                 $result = mysql_query($query);
27
28                 $guests = mysql_fetch_row($result); //gets array of a row of the results
29                 while ($guests == true) {
30                     echo $guests[0] . '<br>' . $guests[1] . '<br><br>';
```

PHP

5 | 27 INS

PHP to mySQL Server : A Complete Example



Project3PHP - NetBeans IDE 6.9.1

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default> | Search (Ctrl+I)

Files | Ser...

- Project3PHP
  - Source Files
    - index.php
  - Include Path

index.php

PHP

action= " "

```
36
37      if (isset($_POST['name']) & isset($_POST['note'])) {//checks if form has been submitted
38          $guestBook = new GuestbookDb('guestbook');
39          $guestBook->addRecord($_POST['name'], $_POST['note']);
40          $guestBook->printRecords();
41          $header="Location:index.php" //file to redirect to. Will change depending on name of
42          header($header);//redirects browser to original page. By redirecting, we avoid multip
43          exit; //ends execution
44      }
45
46      $guestBook = new GuestbookDb('guestbook');
47      $guestBook->printRecords(); //Print records
48      ?>
49      <!-- Guestbook Form-->
50      <b>Please sign our guest book!</b><br>
51      <form action="" method="post" enctype="application/x-www-form-urlencoded">
52          Name: <input type="text" size="50" maxlength="100" name="name" /><br>
53          Note: <textarea rows="6" cols="40"    name="note" ></textarea><br>
54
55
56          <br>
57          <input type="submit" value="Submit" />
58      </form>
59
60      </body>
61  </html>
62
```

HTML Form

11.6 Database programming for the web

Present Note

Connections

This section describes database programming for the web using PHP and MySQL. ***PHP*** is the most widely used programming language used to create dynamic websites. Other alternatives to PHP include Python, Java, Node.js, and ASP.NET. This section assumes familiarity with HTML, the language used to build web pages, and PHP.

A PHP application uses ***PHP Data Objects*** (***PDO***), an API to interact with a wide range of databases, including MySQL. To use PDO for a specific database, a database-specific PDO driver is required. Most PHP installations include a PDO driver for MySQL.

PHP scripts must connect to a database prior to executing queries. A PHP script establishes a connection by creating a ***PDO object***. The PDO constructor specifies a DSN and a MySQL username and password. A ***Data Source Name*** (***DSN***) is a string containing database connection information:

- DSN prefix - "mysql:" for MySQL
- host - Database server's hostname or address
- port - Database server's port number (if the default port number is not used)
- dbname - Database name

If the PDO constructor fails to connect to MySQL successfully, the PDO constructor throws a ***PDOException***, which results in a fatal error. PHP can be configured to handle fatal errors in various ways, including outputting the error message to the web page.

PARTICIPATION ACTIVITY 11.6.1: Connecting to MySQL.

Start 2x speed

db.php

```
<?php
$dsn = "mysql:host=localhost;dbname=Reservation";
$username = "samanead";
$password = "*jka1726";
$pdo = new PDO($dsn, $username, $password);

// Use $pdo to perform database operations

?>
```

Python & mySQL

Same as Application Development

Executing statements

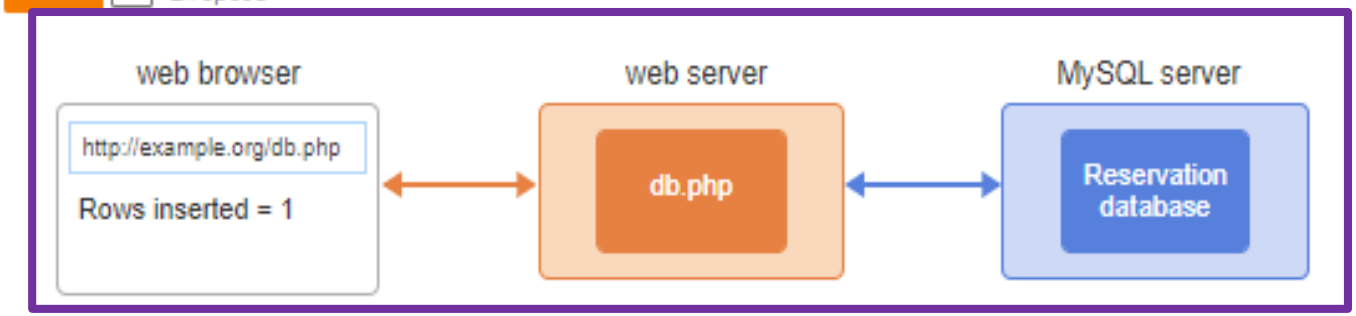
The PDO object's ***query()*** method executes an SQL statement. The query() method returns a ***PDOStatement*** object or FALSE if an error occurs executing the statement.

Instead of checking the query() return value for FALSE, good practice is to call setAttribute() to make the PDO object throw a PDOException when an SQL error occurs. The PDOException provides details about the SQL error. The PDO object's ***setAttribute()*** method sets a PDO attribute to a value. Ex: `$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION)` sets the error mode to throw a PDOException.

The PDOStatement method ***rowCount()*** returns the number of rows that are inserted, updated, or deleted.

PARTICIPATION ACTIVITY 11.6.3: Insert a new flight.

Start 2x speed



Python & mySQL

```
$dsn = "mysql:host=localhost;dbname=Reservation";
$pdo = new PDO($dsn, "samsnead", "^jksi72$");

$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$sql = "INSERT INTO Flight " .
       "(FlightNumber, AirlineName, AirportCode) " .
       "VALUES (280, 'China Airlines', 'PEK')";

$statement = $pdo->query($sql);

echo "Rows inserted = " . $statement->rowCount();
```

Same as Application Development

Prepared statements

An SQL statement may be created dynamically from user input. A ***prepared statement*** is an SQL statement that can be customized with parameter values retrieved from user input. Prepared statements are immune to most SQL injection attacks.

Prepared statements are created and executed in four steps:

1. **Define** - A prepared statement uses an SQL statement with parameter identifiers as placeholders for values. A ***parameter identifier*** can be a question mark (`?`) or a name following a colon (`:name`).
2. **Prepare** - The PDO method ***prepare()*** prepares a parameterized SQL statement for execution and returns a PDOStatement.
3. **Bind** - The PDOStatement method ***bindValue()*** binds a value to a parameter identifier in the SQL statement. Ex: `$statement->bindValue(1, "Bob")` binds "Bob" to the first question mark, and `$statement->bindValue("name", "Bob")` binds "Bob" to the :name parameter identifier.
4. **Execute** - The PDOStatement method ***execute()*** executes the SQL statement with the bound parameters and throws a PDOException if an error occurs.

PARTICIPATION ACTIVITY | 11.6.5: Prepared statement inserts a new flight.

Start | 2x speed

addflight.php

```
$sql = "INSERT INTO Flight " .
       "(FlightNumber, AirlineName, AirportCode) " .
       "VALUES (?, ?, ?)";

$statement = $pdo->prepare($sql);
$statement->bindValue(1, $_POST["flightNum"]);
$statement->bindValue(2, $_POST["airlineName"]);
$statement->bindValue(3, $_POST["airlineCode"]);
$statement->execute();

echo "Flight inserted.";
```

$_POST

flightNum	105
airlineName	Air India
airlineCode	DEL

http://example.org/addflight.php

Flight inserted.

Python & mySQL

Same as Application Development

Fetching values

When executing a PDOStatement with a SELECT statement, the execute() method creates a cursor object. Query results are obtained from the cursor with the fetch() method. The PDOStatement method ***fetch()*** returns an array containing data from one row or FALSE if no row is selected. The row values are indexed in two ways:

- By column name. Ex: ["FlightNumber"] => 350, ["DepartureTime"] => "08:15:00", ["AirportCode"] => "PEK".
- By column number. Ex: [0] => 350, [1] => "08:15:00", [2] => "PEK".

If a SELECT statement returns more than one row, fetch() may be called in a loop. Each call to fetch() returns the next row from the result table. When no more rows exist, fetch() returns FALSE.

The fetch() method has an optional style parameter that returns the fetched row in other formats. Common style values include:

- `PDO::FETCH_NUM` - Returns only an array indexed by column number. Ex: `$row[0]` is the data from the first column.
- `PDO::FETCH_ASSOC` - Returns only an associative array indexed by column name. Ex: `$row["colName"]` is the data from the colName column.
- `PDO::FETCH_OBJ` - Returns an object with property names that match column names. Ex: `$row->colName` is the data from the colName column.

PARTICIPATION ACTIVITY 11.6.7: Fetching flight results.

Start 2x speed

```
$sql = "SELECT FlightNumber, DepartureTime FROM Flight " .
       "WHERE AirportCode = :code AND AirlineName = :name";
$statement = $pdo->prepare($sql);
$statement->bindValue("code", "PEK");
$statement->bindValue("name", "China Airlines");
$statement->execute();

while ($row = $statement->fetch()) {
   echo "<p>Flight $row[FlightNumber] departs at $row[DepartureTime]</p>";
}
```

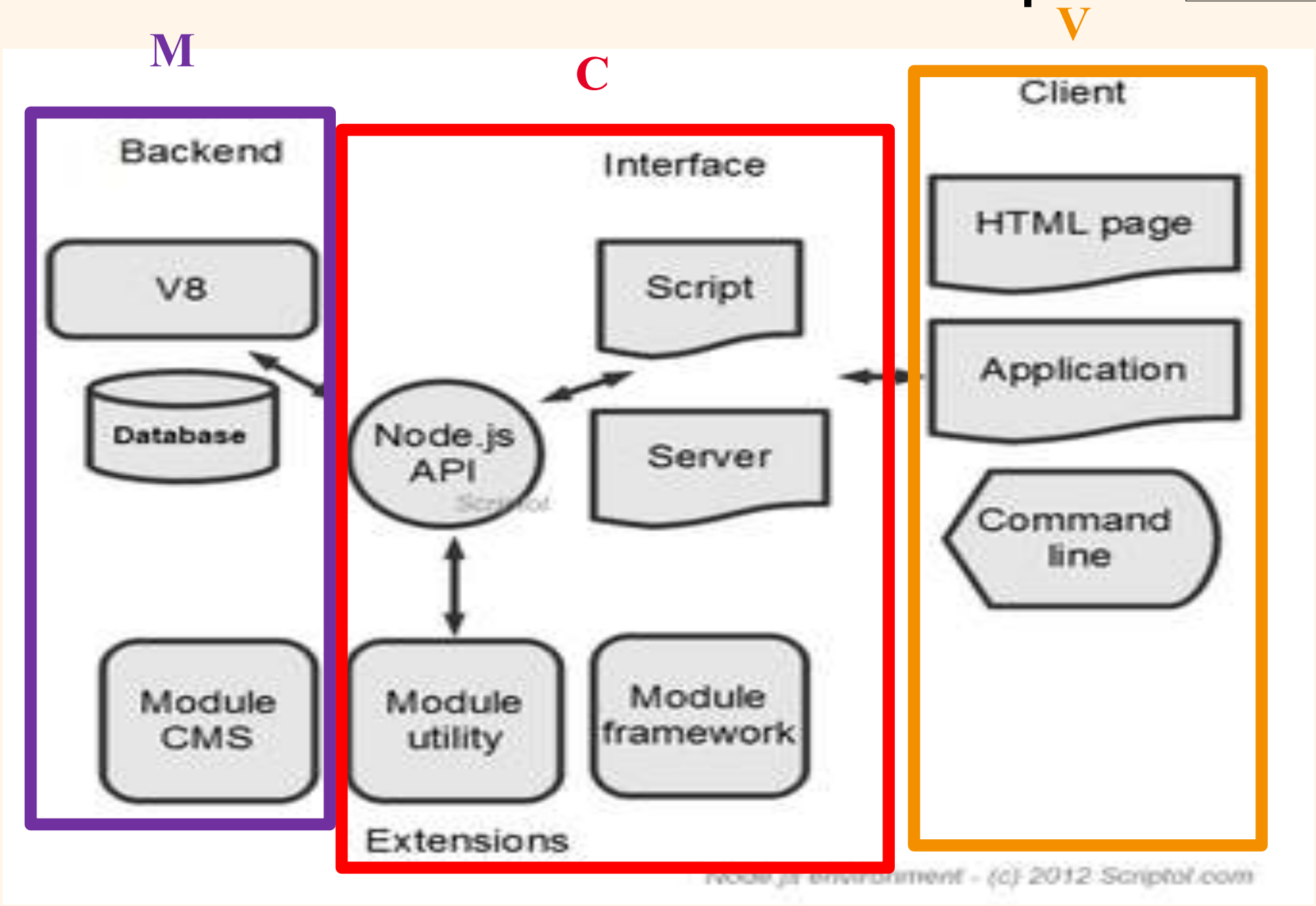
http://example.org/flights.php

Flight 107 departs at 13:30:00
Flight 350 departs at 08:15:00
Flight 482 departs at 10:07:00

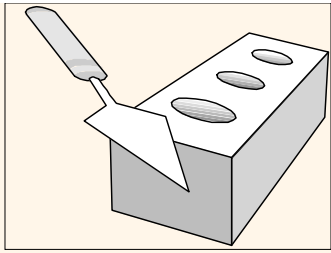
Same as Application Development

Python & mySQL

Node.Js
Server Side Javascript



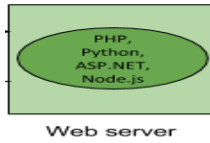
Introduction Node.js



- **Node.js** was created by **Ryan Dahl** starting in 2009, and its growth is sponsored by Joyent, his employer.

- JavaScript used in client-side but **Node.js** puts the **JavaScript** on **server-side** thus making communication between client and server happen in same language

Getting Started with Node.js



Node.js is a JavaScript runtime environment that is primarily used to run server-side web applications. Node.js has many benefits:

- The event-driven, non-blocking I/O architecture of Node.js allows Node.js to handle high loads.
- Node.js allows developers to write JavaScript on the server and client, simplifying some development tasks.
- Node.js provides a simple mechanism to create and distribute modules. A ***Node.js module*** is a self-contained collection of JavaScript code.
- Node.js works seamlessly with MongoDB, a document database that stores JSON and uses JavaScript as a query language. Web development is greatly simplified when JSON is used between the client and server, and between the server and database.

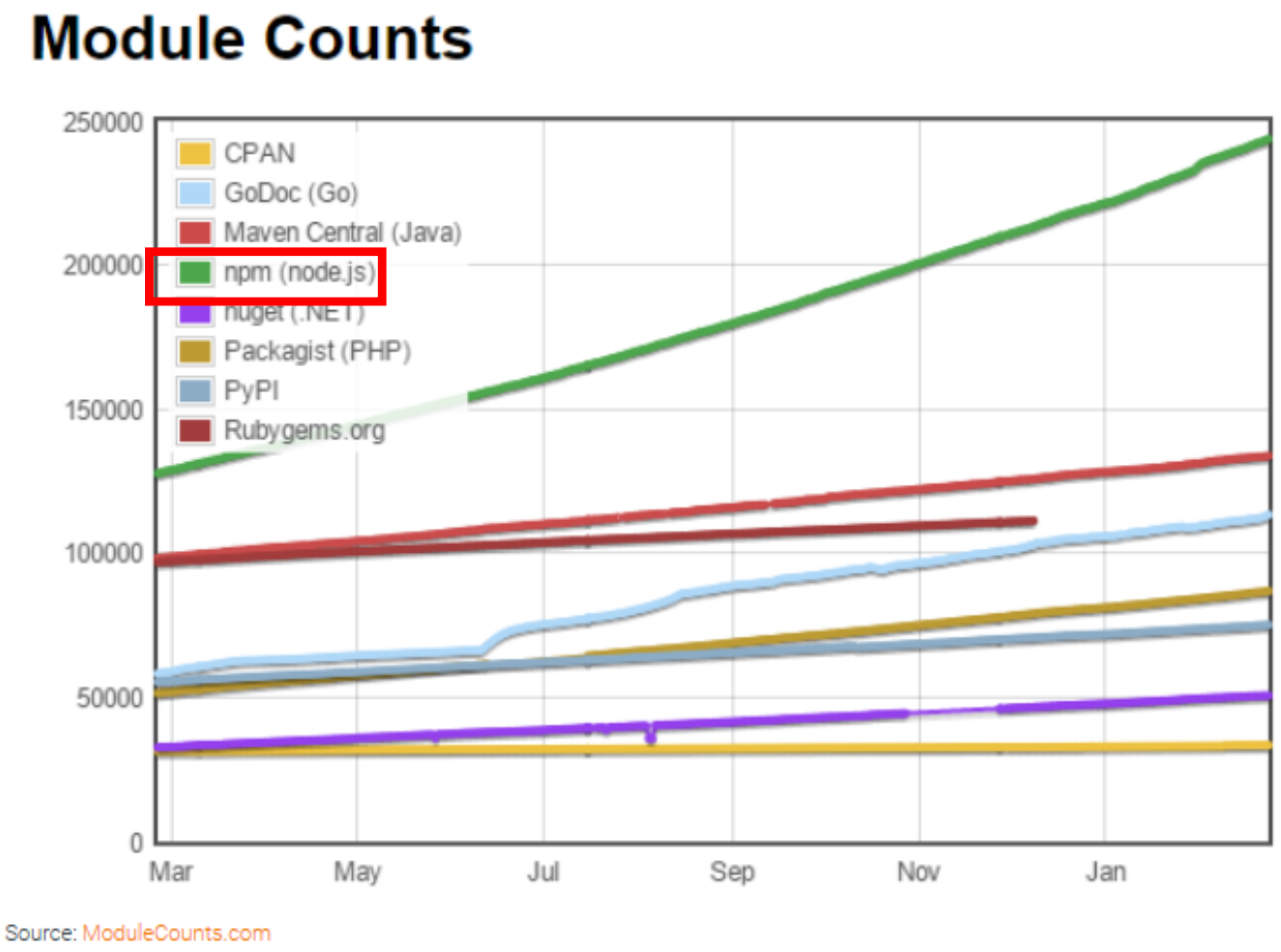
Companies using Node.js include Netflix, Walmart, Ebay, and LinkedIn. Adoption by these companies helped validate the Node.js approach and spur development of more Node.js packages.

JSON – JavaScript Object Notation

node

Introduction to Node.js

Number of modules for the Node.js package manager (npm) far exceeds other languages.



Source: ModuleCounts.com

Web Applications Practice Questions

Node.js programs are always written in JavaScript.

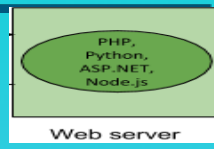
- (●) True
- ( ) False

Node.js uses Google's V8 JavaScript engine to run JavaScript programs.

Your Answer?

8

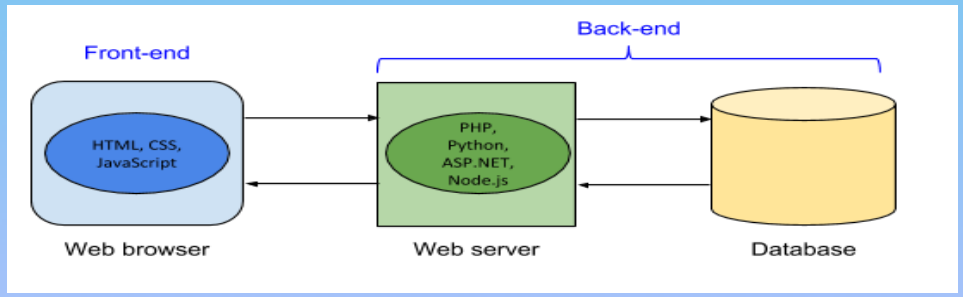
Introduction to Node.js.



Node.js programs run in the web browser.

- ○ True
- ◉ False

Node.js programs run on the web server.



Your Answer?

9

11. SET 2 - 4:WEB APPLICATIONS

A complex type ____.

a. has multiple values per table cell, from the perspective of the database system

b. cannot be used in a join query

c. is defined by the user, not the database system

d. has a richer internal structure than a simple type

From the perspective of the database system, complex types, like simple types, are atomic and stored as one value per cell.

?????

10

11. SET 2 - 4:WEB APPLICATIONS

Which type is complex?

a. BLOB

b. JSON

c. DATE

d. TEXT

?????

11

11. SET 2 - 4:WEB APPLICATIONS

A user-defined type is defined in terms of a _____ type.

a. base

b. simple

c. complex

d. built-in

User-defined types

A ***system-defined type***, also known as a ***built-in type***, is provided by the database as a reserved keyword. Ex: FLOAT, CHAR, ENUM, SET, JSON, and POINT are MySQL system-defined types.

A ***user-defined type*** is created by a database designer or administrator with the CREATE TYPE statement. The ***CREATE TYPE*** statement specifies the type name and a ***base type*** that defines the implementation. The base type can be either system-defined or user-defined. Ex: `CREATE TYPE Meters AS REAL` creates a new type named Meters with base type REAL. Although Meters is implemented as REAL, the two types are different and cannot be directly compared.

User-defined data types appear after column names in CREATE TABLE statements, just like system-defined types. User-defined types can be either simple or complex. Ex: CREATE TYPE is used to create simple enumerated types in PostgreSQL and complex array types in Oracle Database.

CREATE TYPE is specified in the SQL standard and supported in Oracle Database, PostgreSQL, SQL Server, and DB2. However, syntax and capabilities vary. MySQL does not support the CREATE TYPE statement.

12

11. SET 2 - 4:WEB APPLICATIONS

In MySQL, a colum
many bytes does

a. 1

b. 2

c. 4

d. 8

13

11.2 Collection types

Present Note

Collection types

A collection type is defined in terms of a base type. Each collection type value contains zero, one, or many base type values. Ex: `QuarterlySales INTEGER ARRAY[4]` defines a column QuarterlySales with the collection type ARRAY and base type INTEGER. Base type values are called ***elements***.

Collections include four complex types:

- Elements of a ***set*** value cannot be repeated and are not ordered.
- Elements of a ***multiset*** value can be repeated and are not ordered.
- Elements of a ***list*** value can be repeated and are ordered.
- An ***array*** is an indexed list. Each element of an array value can be accessed with a numeric index.

The SQL standard includes multiset and array types but not set and list types. Most databases support only one or two collection types, and implementations vary greatly. Ex: In the MySQL set type, the base type must be a fixed group of character strings. In Informix, the base type can be any type, including complex types. A set value can be NULL in MySQL but not in Informix.

Table 11.2.1: Collection type support.

	Set	Mulitset	List	Array
MySQL	✓	-	-	-
Oracle Database	-	-	-	✓
PostgreSQL	-	-	-	✓
SQL Server	-	-	-	-
DB2	-	-	-	-
Informix	✓	✓	✓	-

The PosgreSQL AP

a. object

b. simple

c. collection

d. semantic

11.2 Collection types

Present Note

Collection types

A collection type is defined in terms of a base type. Each collection type value contains zero, one, or many base type values. Ex: `QuarterlySales INTEGER ARRAY[4]` defines a column QuarterlySales with the collection type ARRAY and base type INTEGER. Base type values are called ***elements***.

Collections include four complex types:

- Elements of a ***set*** value cannot be repeated and are not ordered.
- Elements of a ***multiset*** value can be repeated and are not ordered.
- Elements of a ***list*** value can be repeated and are ordered.
- An ***array*** is an indexed list. Each element of an array value can be accessed with a numeric index.

The SQL standard includes multiset and array types but not set and list types. Most databases support only one or two collection types, and implementations vary greatly. Ex: In the MySQL set type, the base type must be a fixed group of character strings. In Informix, the base type can be any type, including complex types. A set value can be NULL in MySQL but not in Informix.

Table 11.2.1: Collection type support.

	Set	Mulitset	List	Array
MySQL	✓	-	-	-
Oracle Database	-	-	-	✓
PostgreSQL	-	-	-	✓
SQL Server	-	-	-	-
DB2	-	-	-	-
Informix	✓	✓	✓	-

14

11. SET 2 - 4:WEB APPLICATIONS

Elements of the

a. SET

b. MULTISET

c. LIST

d. ARRAY

11.2 Collection types

Present Note

Collection types

A collection type is defined in terms of a base type. Each collection type value contains zero, one, or many base type values. Ex: `QuarterlySales INTEGER ARRAY[4]` defines a column QuarterlySales with the collection type ARRAY and base type INTEGER. Base type values are called ***elements***.

Collections include four complex types:

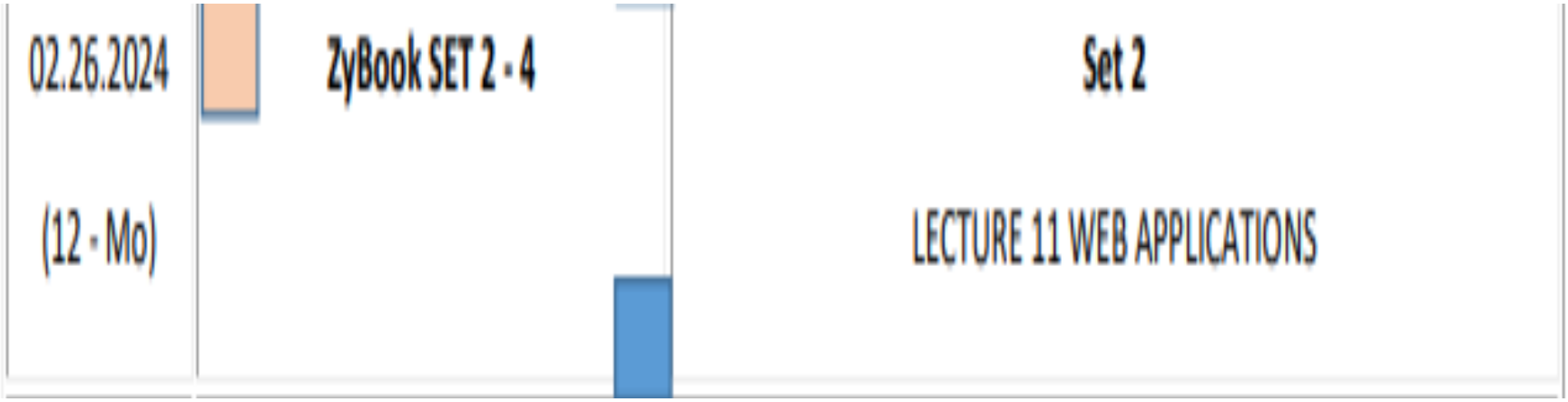
- Elements of a ***set*** value cannot be repeated and are not ordered.
- Elements of a ***multiset*** value can be repeated and are not ordered.
- Elements of a ***list*** value can be repeated and are ordered.
- An ***array*** is an indexed list. Each element of an array value can be accessed with a numeric index.

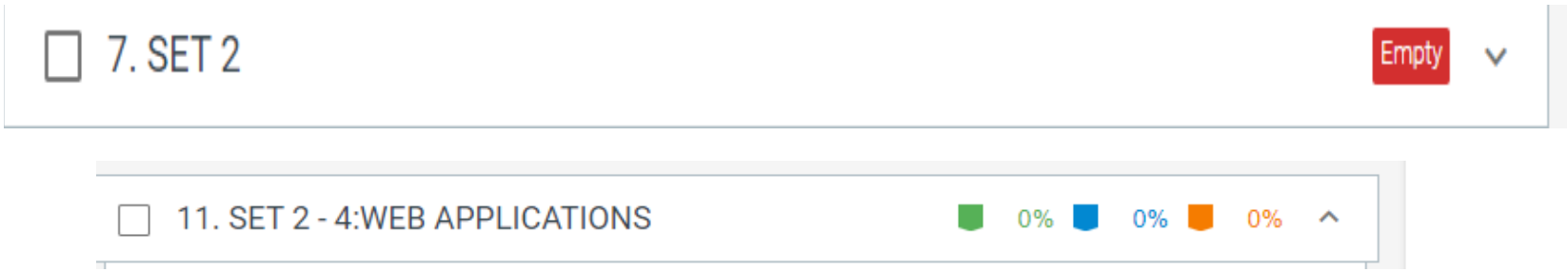
The SQL standard includes multiset and array types but not set and list types. Most databases support only one or two collection types, and implementations vary greatly. Ex: In the MySQL set type, the base type must be a fixed group of character strings. In Informix, the base type can be any type, including complex types. A set value can be NULL in MySQL but not in Informix.

Table 11.2.1: Collection type support.

	Set	Mulitset	List	Array
MySQL	✓	-	-	-
Oracle Database	-	-	-	✓
PostgreSQL	-	-	-	✓
SQL Server	-	-	-	-
DB2	-	-	-	-
Informix	✓	✓	✓	-

At 5:00 PM .

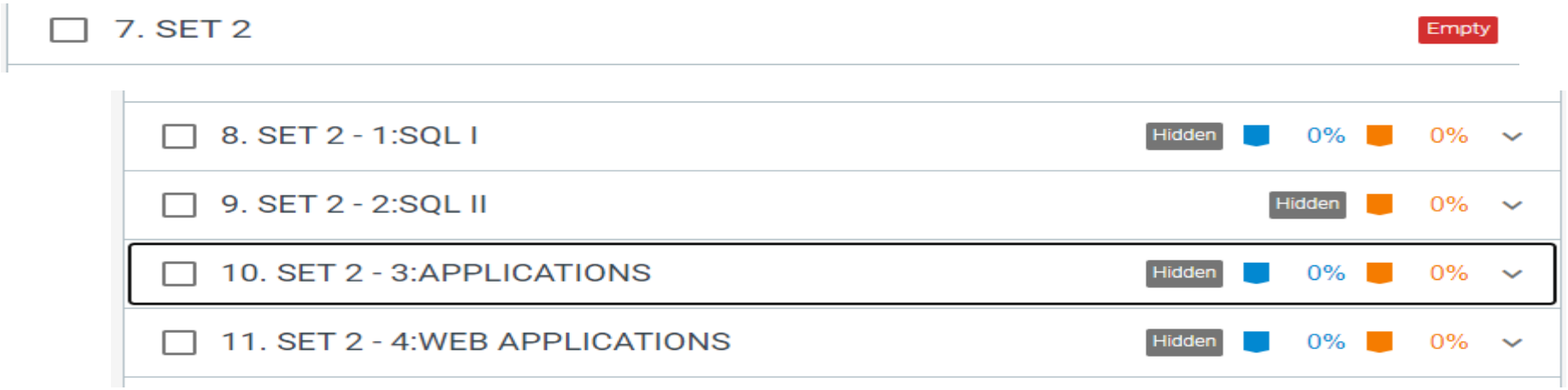




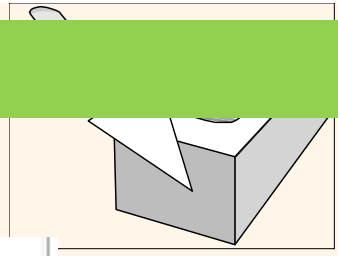
VH work on
SET 2 – 4: WEB APPLICATIONS

Next

02.28.2024 (13 - We)		EXAM 2 Practice (PART of 20 points)



From 5:05 to 5:15 PM – 5 minutes.



02.26.2024 (12 - Mo)	ZyBook SET 2 - 4	Set 2 LECTURE 11 WEB APPLICATIONS

CLASS PARTICIPATION 20 points — 20% of Total

WEB APPLICATIONS

Class 12 END PARTICIPATION
Not available until Feb 26 at 5:05pm | Due Feb 26 at 5:15pm | 100 pts

VH, publish

This is a synchronous online class.

Attendance is required.

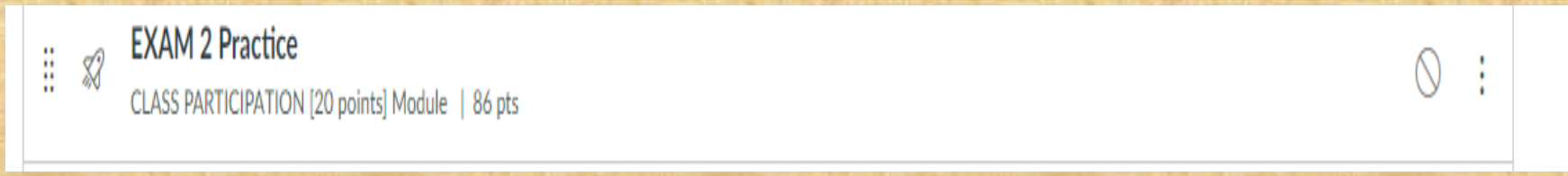
Recording or distribution of class materials is prohibited.

1. At the beginning of selected classes there is an **assessment in the first 10 minutes**. (**beige** BOX in the Detailed Syllabus)
2. At the end of selected classes there is an **assessment in the last 10 minutes**. (**blue** BOX in the Detailed Syllabus)
3. ZyBook sections will be downloaded and used for 30% of Total Score on the dates specified in the Detailed Syllabus.
4. EXAMS are in CANVAS. **No late EXAMS**.
5. I have to be **present in TEAMS in order to take any graded assignment assigned during that class.**

At 5:15 PM.

End Class 12

VH, Download Attendance Report
Rename it:
2.26.2024 Attendance Report FINAL



VH, upload Class 12 to CANVAS.