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'.

The above types have relatively simple internal structures and thus are called ***simple data types***. Ex: A character value consists of a series of individual characters. Ex: A date value has three parts, year, month, and day.

Database functions can decompose the internal structure into separate values. Ex: If BirthDate has type date, the SQL function `month(BirthDate)` might return the birth month. From the perspective of arithmetic and comparison operators, however, the internal structure is ignored and each value is considered atomic.

Table 11.1.1: Simple type examples.

	MySQL	Oracle Database	PostgreSQL	SQL Server
Integer	BIT TINYINT SMALLINT	INT NUMBER	BIT SMALLINT INTEGER	BIT TINYINT SMALLINT

	MEDIUMINT BIGINT		BIGINT	MEDIUMINT BIGINT
Decimal	FLOAT DOUBLE DECIMAL	FLOAT NUMBER	REAL NUMERIC DECIMAL	FLOAT REAL NUMERIC DECIMAL
Character	CHAR VARCHAR TEXT	CHAR VARCHAR2 LONG	CHAR VARCHAR TEXT	CHAR VARCHAR TEXT
Time	DATE DATETIME TIMESTAMP	DATE TIMESTAMP TIMESTAMP WITH TIMEZONE INTERVAL	DATE TIME TIMESTAMP INTERVAL	DATE DATETIME TIME DATETIMEOFFSET
Binary	TINYBLOB MEDIUMBLOB LONGBLOB	BLOB BFILE RAW	BYTEA	BINARY IMAGE
Semantic	ENUM BOOLEAN	UROWID BOOLEAN	MONEY BOOLEAN UUID	MONEY UNIQUEIDENTIFIER

PARTICIPATION ACTIVITY 11.1.1: Simple types.

1) MySQL supports the semantic type MONEY.

- True
- False

2) 3.1415 can be stored as an INT type in Oracle Database.

- True
- False

3) 1/29/20 14:30:00 might represent a DATETIME value in MySQL.

- True
- False

4) Photographs are stored with the BYTEA type in PostgreSQL.
   - True
   - False

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.

Table 11.1.2: Complex type examples.

	MySQL	Oracle Database	PostgreSQL	SQL Server
Collection	SET	CREATE TYPE TypeName AS VARRAY(n) OF	*basetype*[n] *basetype* ARRAY[N]	*none*

		basetype		
Document	JSON	XMLTYPE JSON	XML JSON	XML
Spatial	POINT MULTIPOINT POLYGON MULTIPOLYGON	SDO_GEOMETRY SDO_GEORASTER	POINT LINE POLYGON CIRCLE	GEOMETRY GEOGRAPHY

Object	*none*	CREATE TYPE TypeName AS OBJECT . . .
		CREATE TYPE TypeName AS BODY . . .
			CREATE TYPE TypeName AS . . .	*none*

PARTICIPATION ACTIVITY 11.1.2: Complex types.

1) Which database might have an unordered set of values { 'apple', 'orange', 'banana' } in a cell of a table?

- MySQL
- Oracle Database
- PostgreSQL

2) What kind of type is the following value?

```
<menu>
  <selection>
    <name>Greek salad</name>
    <price>$13.90</price>
    <text>Cucumbers, tomatoes,
onions, and feta cheese</text>
  </selection>
  <selection>
    <name>Turkey
sandwich</name>
    <price>$9.00</price>
    <text>Turkey, lettuce,
tomato on choice of
bread</text>
  </selection>
</menu>
```

- ○ Collection
- ○ Document
- ○ Object

3) Which database has both XML and JSON types?
- ○ MySQL
- ○ Oracle Database only
- ○ Oracle Database and PostgreSQL

4) Which database does not support object types?
- ○ MySQL
- ○ Oracle Database
- ○ PostgreSQL

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.

PARTICIPATION ACTIVITY | 11.1.3: PostgreSQL CREATE TYPE statement.

Animation captions:

PARTICIPATION ACTIVITY | 11.1.4: User-defined types.

1) Which database does not support the CREATE TYPE statement?

- MySQL
- Oracle Database
- PostgreSQL

2) User-defined types can be:

- Simple types only
- Complex types only
- Either simple or complex types.

3) System-defined types can be:

- Simple types only
- Complex types only
- Either simple or complex types

Exploring further:

- MySQL types
- Oracle Database types
- PostgreSQL types
- SQL Server types

11.2 Collection types

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	Multiset	List	Array
MySQL	✔	-	-	-

Oracle Database	-	-	-	✔
PostgreSQL	-	-	-	✔
SQL Server	-	-	-	-
DB2	-	-	-	-
Informix	✔	✔	✔	-

PARTICIPATION ACTIVITY | 11.2.1: Collection types.

1) The only difference between the multiset and list types is that elements are ordered in a list value and not in a multiset value.
   - True
   - False

2) In the Informix set type, the base type must be a group of character strings.
   - True
   - False

3) In all implementations, the base type of a collection can be any supported type.
   - True
   - False

4) In some implementations, the base type of a collection can be a complex type.
   - True
   - False

5) Collection type implementations always conform to the SQL standard.

○ True
○ False

Set type

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.

Animation captions:

PARTICIPATION ACTIVITY | 11.2.3: Set type.

Refer to the MySQL table created by this statement:

```
CREATE TABLE Part (
   PartNumber INTEGER,
   Material SET ('Steel', 'Copper', 'Aluminum', 'Zinc'),
   PRIMARY KEY (PartNumber)
);
```

1) Which string is a correct value of the Material column?

- ○ 'Steel, Copper, Zinc'
- ○ 'Steel,Copper,Zinc'
- ○ ('Steel', 'Copper', 'Zinc')

2) Which query selects all parts containing copper?

○
```
SELECT PartNumber
FROM Part
WHERE Material =
'Copper';
```

○
```
SELECT PartNumber
FROM Part
WHERE Material =
'%Copper%';
```

○
```
SELECT PartNumber
FROM Part
WHERE Material LIKE
'%Copper%';
```

3) If a SET type has 25 elements, how

5) If a SET type has 25 elements, how many bytes does each set value require?

- ○ Two bytes
- ○ Four bytes
- ○ Eight bytes

Array type

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 }');
```

Employee

ID	Name	QuarterlySales
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 }

Animation content:

Static figure:
Two SQL statements appear.
Begin SQL code:
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 }');
End SQL code.

The Employee table appears below the SQL statements. Employee has columns ID, Name, and QuarterlySales. Employee has three rows:
2538, Lisa Ellison, {1450, 2020, 900, 5370}
6381, Maria Rodriguez, {3340, 800, 1700, 6400}
7920, Jiho Chen, {0, 3900, 8000, 320}

A third SQL statement appears below Employee.
Begin SQL code:
SELECT *
FROM Employee
WHERE QuarterlySales[2] > 1000;
End SQL code.

The Result table appears below the third SQL statement. Result has columns ID, Name, and

QuarterlySales. Result has two rows:
2538, Lisa Ellison, {1450, 2020, 900, 5370}
7920, Jiho Chen, {0, 3900, 8000, 320}

Step 1: The QuarterlySales column is an array of four integers, representing employee sales in each of four quarters. The CREATE TABLE statement appears. The QuarterlySales column definition is highlighted. The Employee table appears with no rows.

Step 2: Array values are specified as elements separated by commas and within braces. The INSERT statement appears. The three Employee rows appear.

Step 3: QuarterlySales[2] refers to the second element of each value in the QuarterlySales column. The SELECT statement appears. The WHERE clause is highlighted. The Result table appears.

Animation captions:

1. The QuarterlySales column is an array of four integers, representing employee sales in each of four quarters.
2. Array values are specified as elements separated by commas and within braces.
3. QuarterlySales[2] refers to the second element of each value in the QuarterlySales column.

In Oracle Database, an array is a user-defined type, created with the CREATE TYPE statement. Ex: The statement `CREATE TYPE Numbers VARRAY(4) OF INTEGER` creates a four-integer array type called Numbers.

PARTICIPATION ACTIVITY 11.2.5: Array type.

Match the PostgreSQL fragment with the description.

If unable to drag and drop, refresh the page.

{ 'Copper', 'Iron', 'Copper', 'Zinc', 'Zinc' } **TEXT [10] [100]** **DATE [ ]**

{ { 'Copper', 'Iron' }, { 'Copper', 'Zinc' }, { 'Zinc', 'Aluminum' } } **DATE [100]**

An array type containing 100 elements

	A two-dimensional array type
	An array type containing an unspecified number of elements
	A one-dimensional array value
	A two-dimensional array value

Reset

CHALLENGE ACTIVITY 11.2.1: Collection types.

544874.3500394.qx3zqy7

Start

The following creates a MySQL table:

```
CREATE TABLE Traveler (
   ID INTEGER,
   Name VARCHAR(20),
   City SET('Maseru', 'Beirut', 'Bissau', 'Nassau', 'Ankara'),
   PRIMARY KEY (ID)
);
```

Complete the query below to insert values Beirut and Bissau.

```
INSERT INTO Traveler (ID, Name, City)
VALUES (1234, 'Dan Lee', /* Your code goes here */);
```

1 2 3 4 5

Check Next

Exploring further:

- MySQL set type
- PostgreSQL array type

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	-	-

PARTICIPATION ACTIVITY | 11.3.1: Structured, semistructured, and unstructured data.

Match the data category to the example.

If unable to drag and drop, refresh the page.

Semistructured Unstructured Structured

	The Library of Congress scans all books and documents in the collection. Scanned images are converted to characters with optical character recognition software. Each book is associated with a title and author.
	A file contains 80,000 lines of data. Each line is separated by a new line character and contains 12 data values separated by commas. A second file contains descriptions of each of the 12 values in a line.
	Bloomberg News provides a financial information service. The service streams financial data as a series of records. Each record contains current price and identifier for a financial instrument, such as a stock or bond. Additional information in the record depends on the type of the financial instrument, as described in a reference manual.

Reset

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.

Animation captions:

XML tags are similar to relational column names but have several advantages:

- *Readable*. Tag names and embedded data are legible in an XML document and easy to understand.
- *Flexible*. Tags can be easily added or dropped as elements are inserted into or removed from a document. Flexibility is important for semistructured data, such as user click sequences on a website.
- *Hierarchical*. Hierarchical data is easily represented by nesting elements within elements. In comparison, the relational representation of hierarchical data requires multiple tables, foreign keys, and referential integrity rules.

The primary disadvantage of XML is document size. Each element appears between a pair of tags, and each pair is repeated for every value. Ex: A ten-character tag repeated over a million elements requires 20 million bytes (10 characters × 2 tags per element × 1,000,000 elements). By comparison, only a few bytes of overhead are necessary for each relational column.

PARTICIPATION ACTIVITY 11.3.3: XML format.

Refer to this XML:

```
<Library>
   <Book>
      <Title>For Whom the Bell Tolls</Title>
      <Author>Ernest Hemingway</Author>
      <Publisher>Random House</Publisher>
      <Copyright>1940</Copyright>
   <___A___>
   <Book>
      <Title>The Map of Knowledge</Title>
      <Author>Violet Moller</Author>
      <Publisher>Doubelday</Publisher>
      <Copyright>2019</Copyright>
   </Book>
   <Movie>
      <Title>La La Land</Title>
      <Director>Damien Chazelle</Director>
      <Producer>Fred Berger</Producer>
      <Producer>Jordan Horowitz</Producer>
      <Producer>Gary Gilbert</Producer>
      <Producer>Mark Platt</Producer>
      <___B___>
         <Budget>$30,000,000</Budget>
         <Revenue>$446,100,000</Revenue>
      </Financial>
      <Copyright>2016</Copyright>
      <RunTime>128 minutes</RunTime>
   </Movie>
```

____C____

1) What is A?

Check Show answer

2) What is B?

Check Show answer

3) What is C?

Check Show answer

4) How many producers did the movie La La Land have?

Check Show answer

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.

Animation captions:

PARTICIPATION ACTIVITY 11.3.5: JSON format.

Refer to following JSON document:

```
[
  { "Name": "oreo",
    "Type": "cookie",
    "Flavors": ["chocolate", "vanilla"],
    "Favorite": false,
    "Created": 1912
  },
  { "Name": "snickers",
    "Type": "candy bar",
    "Flavors": ["chocolate", "peanuts", "caramel", "nougat"],
    "Favorite": true,
    "Created": 1930
  },
  { "Name": "malt",
    "Type": "frozen dairy",
    "Flavors": ["vanilla", "chocolate", "strawberry"],
    "Favorite": false,
    "Created": 1922
  }
]
```

1) What type of data is created by the outer brackets in the JSON document?

- ○ array
- ○ boolean
- ○ object

2) How many objects are created by the JSON document?

- ○ 1
- ○ 3
- ○ 4

3) What is the data type of `Favorite`?

- ○ boolean
- ○ object
- ○ string

4) What is the data type of `Created`?

- ○ number
- ○ object
- ○ string

XML and JSON types

Most relational databases support XML or JSON types, but capabilities and internal format vary greatly.

MySQL supports the JSON type. A JSON value is stored as an internal binary format rather than a character string. The format is optimized so that, given an element name or array index, MySQL can quickly find element data. If the document were stored as a character string, the database would have to read and parse the entire document to find an element, which is relatively slow.

MySQL supports comparisons of JSON values with the < > and = operators but not BETWEEN and IN. In addition, MySQL provides roughly 30 functions that manipulate JSON values. Ex:

- ***JSON_ARRAY()*** formats string or numeric data as a JSON array.
- ***JSON_DEPTH()*** determines the maximum number of levels in a JSON document hierarchy.
- ***JSON_EXTRACT()*** returns data from a JSON document.
- ***JSON_OBJECT()*** converts element names and data to a JSON document.
- ***JSON_PRETTY()*** prints a JSON document in a format that is easy to read, with one element per line.

Like document type implementations in most databases, MySQL checks JSON values for correct syntax and does not store invalid documents.

PARTICIPATION ACTIVITY | 11.3.6: MySQL JSON type.

Animation captions:

PARTICIPATION ACTIVITY | 11.3.7: XML and JSON types.

1) Functions that manipulate XML and JSON values are similar in most relational databases.

- ○ True
- ○ False

2) JSON documents can be stored as a VARCHAR type in MySQL.

- ○ True
- ○ False

3) When a document is inserted to a column with XML or JSON type, databases generate an error if the syntax is incorrect.

- ○ True
- ○ False

4) In MySQL, tables can be joined on columns with type JSON.

- ○ True
- ○ False

CHALLENGE ACTIVITY | 11.3.1: Document types.

544874.3500394.qx3zqy7

Start

Select a data category for each scenario.

Pick (a) A relational database storing population data for each European rows.

Pick (b) `{ first_name: "Sue", last_name: "Sato", age: 25` `"Sue.Sato@email.com" }`

Pick (c) Images of places uploaded to a search engine to identify the plac

1 2 3 4 5

Check Next

Exploring further:

- MySQL JSON type
- MySQL JSON functions
- Oracle Database XML type
- Oracle Database JSON type
- PostgreSQL XML type
- PostgreSQL JSON type

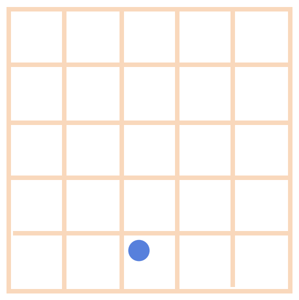
11.4 Spatial types

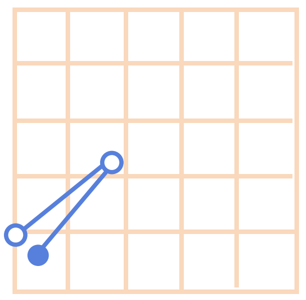
Spatial data

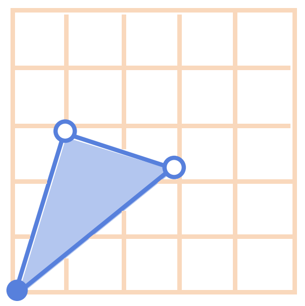
Spatial data is a geometric object, such as a point, line, polygon, or sphere, specified as coordinates in an N-dimensional space. Two-dimensional data is used in mapping applications, such as Google Maps. Three-dimensional data is used in computer-aided design (CAD) systems to engineer airplanes, buildings, 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.

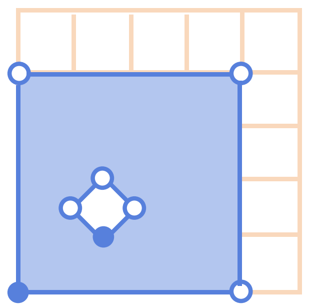
Spatial data is commonly written in a format called ***Well-Known Text*** (***WKT***). WKT format specifies a shape name followed by vertex coordinates.

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))

Animation content:

Static figure:
Four diagrams represent two-dimensional graphs with background grids.

The first diagram has caption POINT(22, 8) and contains a single point.

The second diagram has caption LINESTRING(5 8, 18 21, 0 10). The diagram has two connected line segments with a small circle at each endpoint. One circle, at point (5, 8) on the graph, is solid. The other two circles are empty.

The third diagram has caption POLYGON((0 0, 29 21, 10 29, 0 0)). The diagram has a triangle with a circle at each vertex. One circle, at point (0, 0) on the graph, is solid. The other circles are empty.

The fourth diagram has caption POLYGON((0 0, 40 0, 40 40, 0 40, 0 0), (15 10, 20 15, 15 20, 10 15, 15 10)). The diagram has two rectangles, one inside the other. Both rectangles have a small circle at each vertex. In the outer rectangle, one circle at point (0, 0) on the graph is solid. In the inner rectangle, one circle at point (15, 10) on the graph is solid. All other circles are empty.

Step 1: A point in two dimensions has X and Y coordinates. POINT(22 8) is WKT format. The diagram with caption POINT appears.

Step 2: A WKT linestring is a series of connected line segments. Points are separated by commas. The diagram with caption LINESTRING appears.

Step 3: A WKT polygon is a series of points, separated by commas and enclosed in an extra set of parentheses. The first POLYGON diagram appears.

Step 4: WKT polygons can have holes. The first polygon is the outer boundary and the second is a hole. The second POLYGON diagram appears.

Animation captions:

1. A point in two dimensions has X and Y coordinates. POINT(22 8) is WKT format.
2. A WKT linestring is a series of connected line segments. Points are separated by commas.
3. A WKT polygon is a series of points, separated by commas and enclosed in an extra set of parentheses.
4. WKT polygons can have holes. The first polygon is the outer boundary and the second is a hole.

PARTICIPATION ACTIVITY 11.4.2: Spatial data.

1) Which option is correct WKT format?

- ○ `LINESTRING(10 10, 15 23, 43 30, 40 29)`
- ○ `LINESTRING( (10, 10), (15, 23), (43, 30), (40, 29) )`
- ○ `LINE(10 10, 15 23, 43 30, 40 29)`

2) Which option is correct WKT format?

- ○ `POLYGON(-10 20, 30 40, 20  40, -10 20)`
- ○ `POLYGON( (-10 20, 30 40, 20  40, -10 20) )`
- ○ `POLYGON( (10 20) (30 40) (20  40) (10 20) )`

3) What does the WKT below represent?

```
POLYGON( (0 0, 30 0, 20 20, 0
0),
         (5 5, 10 5, 10 10, 5
10, 5 5) )
```

- ○ Two separate polygons, a triangle, and a square
- ○ A triangle with a square hole

○ A square with a triangular hole

Geometric and geographic data

Spatial data is either geometric or geographic:

- ***Geometric data*** is embedded in a flat plane or 'square' three-dimensional space, called a ***Cartesian coordinate system***.
- ***Geographic data*** is defined with reference to the surface of the earth. Ex: A geographic point can be described as latitude, longitude, and elevation above sea level.

Since the surface of the earth is curved, distance and area computations are different for geometric and geographic data. The surface of the earth is a complex shape and can be described with many alternative coordinate systems, called ***spatial reference systems***. Spatial reference systems are standardized and identified with ***spatial reference system identifiers*** (***SRID***).

Most databases implement a spatial data standard from the Open Geospatial Consortium (OGC). The OGC standard specifies spatial types and functions for SQL. The OGC standard also includes a table of standard spatial reference systems and SRIDs. For more information, see OGC standard.

Most databases support spatial data natively or through a separate, optional component. Ex: SQL Server supports spatial data natively. Oracle Database supports spatial data in a separate component, called Oracle Spatial and Graph. PostgreSQL supports only geometric data, but a separate product, PostGIS, adds support for geographic data.

This section describes MySQL spatial types and functions, supported by the InnoDB, MyISAM, NDB, and ARCHIVE storage engines. MySQL types and functions conform to the OGC standard and are similar to types and functions in other databases.

Table 11.4.1: Spatial type support.

	Native support	Optional component
MySQL	✔	-
Oracle Database	-	Oracle Spatial and Graph
PostgreSQL	✔	PostGIS
SQL Server	✔	-
DB2	✔	

DB2	✓	-
Informix	-	Informix Spatial DataBlade

PARTICIPATION ACTIVITY | 11.4.3: Spatial data.

1) What is the distance between `POINT(0 0)` and `POINT(3 4)`?

- Exactly 5 units
- Exactly 7 units
- Depends on the spatial reference system

2) All spatial data is either two- or three-dimensional.

- True
- False

3) Spatial data types in MySQL, SQL Server, and PostGIS are:

- Completely different
- Similar
- Identical

Spatial types

MySQL types are restricted to two-dimensional coordinate systems. MySQL supports four basic spatial types:

- ***POINT*** describes a specific location, such as an address.
- ***LINESTRING*** consists of one or more line segments and represents objects like rivers and streets. A linestring can be closed, like a rectangle, but is one-dimensional and does not have an area.
- ***POLYGON*** describes two-dimensional surfaces such as regions or postal code areas. Polygons are closed and have an area. Polygons may have holes, represented as inner polygons within an outer polygon.

- ***GEOMETRY*** values can be either a POINT, LINESTRING, or POLYGON.

Each value of these basic types is a single geometric element. In addition, MySQL supports four types that contain multiple geometric elements in each value: ***MULTIPOINT***, ***MULTILINESTRING***, ***MULTIPOLYGON***, and ***GEOMETRYCOLLECTION***. Ex: A GEOMETRYCOLLECTION value can include multiple POINT, LINESTRING, or POLYGON elements.

MySQL stores spatial values in an internal format:

- Four bytes for the SRID
- Four bytes for the spatial type
- Eight bytes for each coordinate

Ex: A polygon with five vertices requires 88 bytes (4 byte SRID + 4 byte type + 5 vertices × 2 coordinates per vertex × 8 bytes per coordinate).

SQL statements must explicitly convert between WKT and internal format. In INSERT and UPDATE statements, WKT is transformed to internal format with functions like ***ST_POINTFROMTEXT()***. In SELECT statements, internal format is transformed to WKT with ***ST_ASTEXT()***.

In CREATE TABLE statements, an optional SRID specifies the spatial reference system for each spatial column. Spatial reference systems are defined in a MySQL catalog table called ***ST_SPATIAL_REFERENCE_SYSTEMS***. Ex: SRID 4326 is a common system for specifying geographic coordinates. If an SRID is not specified, the value defaults to zero, which refers to a Cartesian reference system.

PARTICIPATION ACTIVITY 11.4.4: MySQL spatial types.

Animation captions:

PARTICIPATION ACTIVITY 11.4.5: Spatial types.

1) The Location column is created with the POINT type and no SRID. What is the distance between Location values `POINT(0 0)` and `POINT(3 4)`?

- ○ Exactly 5 units
- ○ Exactly 7 units
- ○ Depends on the spatial reference system

2) In internal MySQL format, how many bytes does `LINESTRING( 0 0, 45 90, -82 .5 )` occupy?

- ○ 40
- ○ 48
- ○ 56

3) Which expression has correct syntax?

- ○ `ST_GEOMETRYFROMTEXT(GEOMETRY( 77 34.5 ), 0)`
- ○ `ST_GEOMETRYFROMTEXT('GEOMETRY( 77 34.5 )')`
- ○ `ST_GEOMETRYFROMTEXT('GEOMETRY( 77 34.5 )', 0)`

4) The District column contains polygonal values. Some districts have 'holes', and others consist of several

disjoint areas. What is the type of District?

- ○ POLYGON
- ○ MULTIPOLYGON
- ○ GEOMETRY

Spatial functions

MySQL supports approximately 90 functions that manipulate spatial data. Since distance and area calculations depend on the spatial reference system, many functions utilize the SRID stored with each spatial value. Most function names have an *ST_* prefix that stands for 'spatial type'. Ex:

- ***ST_AREA()*** returns the area of a polygon or multipolygon.
- ***ST_DISTANCE()*** determines the distance between two spatial values.
- ***ST_OVERLAPS()*** determines if two spatial values overlap.
- ***ST_UNION()*** merges two spatial values into one.
- ***ST_X()*** and ***ST_Y()*** return the X- and Y-coordinate of a point.

A ***minimum bounding rectangle***, or ***MBR***, is the smallest rectangle that contains a spatial value. MBRs are aligned with the X- and Y-axes. Ex: The MBR of `POLYGON( (20 25, 30 30, 34 10, 20 25) )` extends from 20 to 34 on the X-axis and 10 to 30 on the Y-axis. Many spatial operations are optimized using minimum bounding rectangles . Ex: The containment operation finds polygons that contain one or more points. The proximity operation finds the closest spatial value to a point, linestring, or polygon. By comparing MBRs, containment and proximity operations rapidly eliminate many spatial values from consideration.

MySQL supports functions that manipulate MBRs, such as:

- ***ST_MBR()*** returns the smallest rectangle that contains a spatial value.
- ***MBRCONTAINS()*** determines if the MBR of one spatial value contains the MBR of another.
- ***MBROVERLAPS()*** determines if the MBRs of two spatial values overlap.

MySQL operators do not work with spatial values. Instead, operators must use numeric values returned by spatial functions.

PARTICIPATION ACTIVITY | 11.4.6: MySQL spatial functions.

Capital

StateCode	StateName	CapitalName	Location
CA	California	Sacramento	POINT(38.5 -121.5)
IA	Iowa	Des Moines	POINT(41.6 -93.6)
NY	New York	Albany	POINT(42.7 -73.8)

```
SELECT CapitalName,
    ST_DISTANCE(Location,ST_POINTFROMTEXT('POINT(40 -100)',4326),'metre') AS Dist
FROM Capital
WHERE ST_X(Location) < 42;
```

Result

CapitalName	Distance
Sacramento	1858816.95
Des Moines	568397.51

Animation content:

Static figure:
The Capital table has columns StateCode, StateName, CapitalName, and Location. Capital has three rows:
CA, California, Sacramento, POINT(38.5 -121.5)
IA, Iowa, Des Moines, POINT(41.6 -93.6)
NY, New York, Albany, POINT(42.7 -73.8)

An SQL statement selects rows of the Capital table using a spatial function.
Begin SQL code:
SELECT CapitalName, ST_DISTANCE(Location,ST_POINTFROMTEXT('POINT(40 -100)',4326),'metre') AS Distance
FROM Capital
WHERE ST_X(Location) < 42;
End SQL code.

The result of the SELECT statement has columns CapitalName and Distance. The result has two rows:
Sacramento, 1858816.95
Des Moines, 568397.51

Animation captions:

1. The Capital table contains state capitals. Location values are latitude and longitude in spatial reference system 4326.
2. ST_DISTANCE() computes each capital's distance in meters to 40° latitude and -100° longitude. MySQL uses the British spelling 'metre' rather than the American spelling 'meter'.
3. The WHERE clauses uses ST_X() to get the Location's X-coordinate. Spatial data must be converted to numeric data to use a comparison operator.
4. Sacramento and Des Moines both have X-coordinates < 42. Distance is displayed in meters.

PARTICIPATION ACTIVITY 11.4.7: Spatial functions.

Refer to the table in the above animation. The location of Denver is approximately 39.7 latitude and -105.0 longitude. The query below returns all cities further than 1000 kilometers from Denver. 'Metre' is the British spelling of the American 'meter'.

```
SELECT CapitalName
FROM Capital
WHERE ___A___(Location, ST_POINTFROMTEXT(___B___, 4326), 'metre') >
___C___;
```

1) What is A?

Check Show answer

2) What is B?

Check Show answer

3) What is C?

Check Show answer

Spatial indexes

This discussion assumes familiarity with multi-level indexes, described elsewhere in this material

Indexes on spatial columns pose special problems:

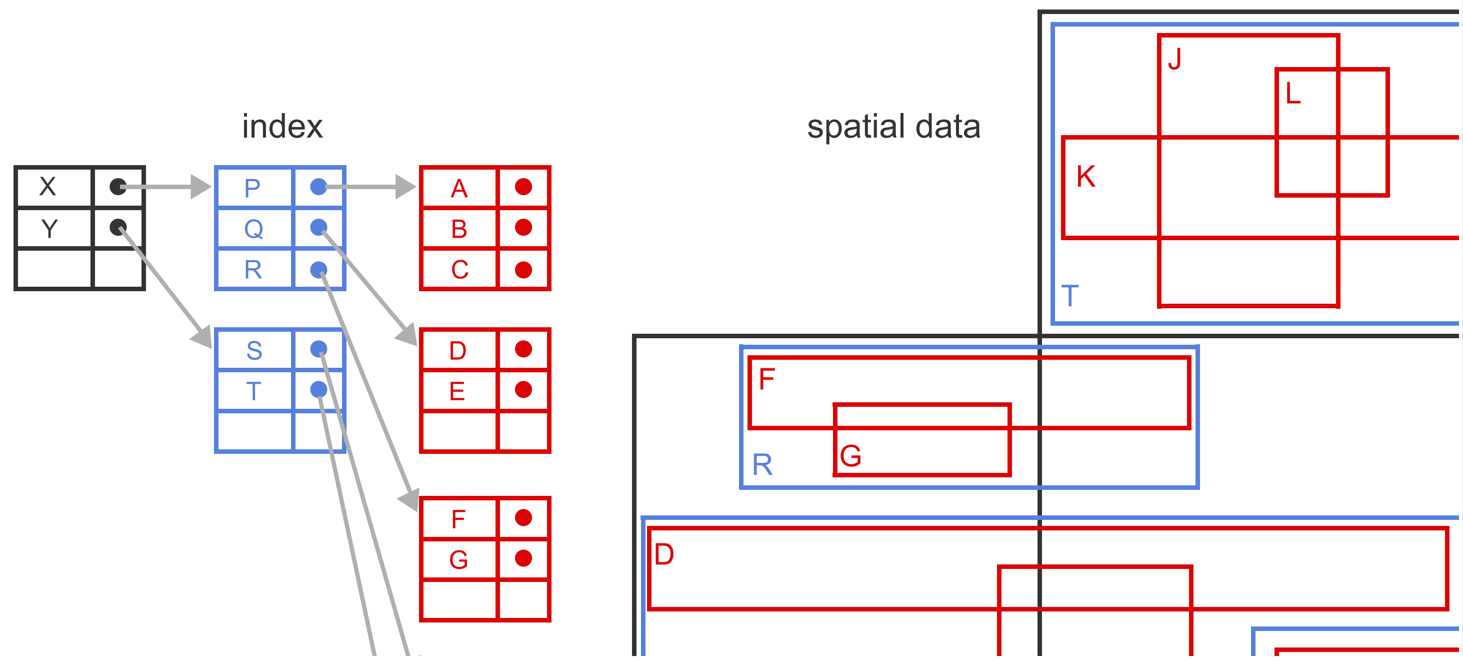
- *Index entries must be sorted on column values*. Two- and three-dimensional spatial values do not have an obvious sort order.
- *Index entries must be small*. A linestring or polygon value may require hundreds of bytes, resulting in large indexes and slow searches.
- *Index entry comparisons must be fast*. Spatial searches involve relatively slow comparisons, such as containment and proximity, that require numerous arithmetic computations.

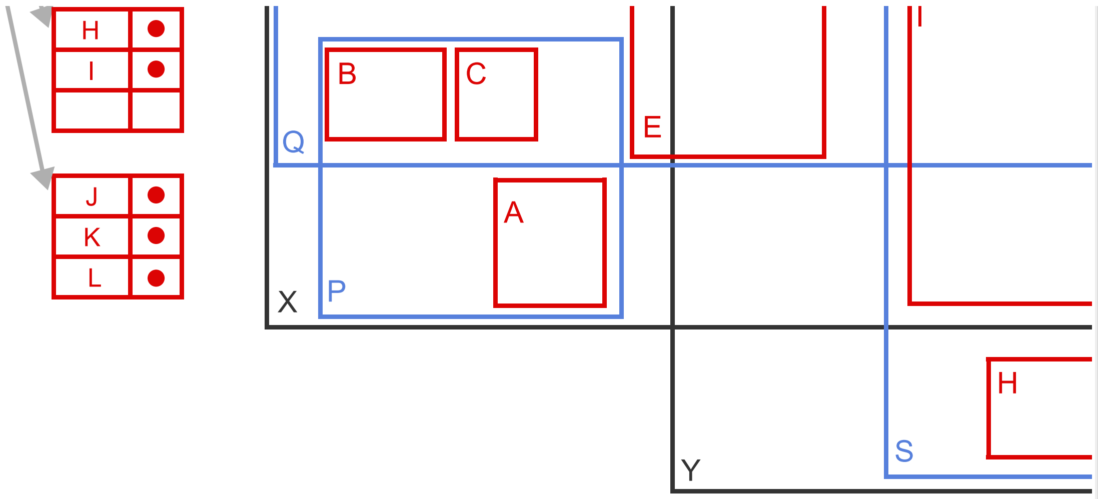
For the above reasons, most databases use a special structure for spatial indexes, called an R-tree. An ***R-tree*** is a B+tree in which index entries contain MBRs rather than column values. Like a B+tree, an R-tree has multiple index levels:

- *The bottom level* has one index entry for each spatial value in a column. Each entry has the MBR for the value and a pointer to the block containing the value.
- *Higher levels* consist of index entries pointing to lower levels. Each entry has a pointer to one lower-level block, along with an MBR containing all MBRs in the block.

As spatial values are inserted and deleted in a column, index entries are added and removed in the bottom level. Index blocks eventually fill or empty and, as with a B+tree, split or merge to maintain a balanced tree. When an insertion causes a block split, index entries in one block are divided across two blocks using the ***quadratic split*** algorithm. The algorithm attempts to minimize the size of MBRs in index entries pointing to the two new blocks.

PARTICIPATION ACTIVITY | 11.4.8: R-tree index.





Animation content:

Step 1: A table contains a spatial column with lines, linestrings, and polygons. Twelve objects are arranged randomly on the right of the screen. The objects are lines, linestrings, and polygons.

Step 2: To create an R-tree index on the spatial column, the database determines an MBR for each spatial value. Each object is surrounded by a red rectangle. The rectangles are labeled A through L.

Step 3: The quadratic split algorithm groups bottom-level MBRs into middle-level MBRs. The red rectangles are grouped by proximity. Each of five groups is surrounded by a blue rectangle, labeled P through T.

Step 4: The quadratic split algorithm groups middle-level MBRs into top-level MBRs. The blue rectangles are grouped by proximity. Each of two groups is surrounded by a black rectangle, labeled X and Y.

Step 5: The bottom level of the index contains bottom-level MBRs and pointers to table blocks containing spatial values. The bottom level of an index appears to the left of the rectangles. Each entry contains a letter, A through L, with an arrow pointing to the corresponding red rectangle.

Step 6: The middle level of the index contains middle-level MBRs and pointers to bottom-level index blocks. The middle level of the index appears to the left of the bottom level. Each entry contains a letter, P through T, with an arrow pointing to a bottom level index block.

Step 7: The top level of the index contains top-level MBRs and pointers to middle-level index blocks. The top level of the index appears to the left of the middle level. Each entry contains a letter, X or Y, with an arrow pointing to a middle-level index block.

Animation captions:

1. A table contains a spatial column with lines, linestrings, and polygons.
2. To create an R-tree index on the spatial column, the database determines an MBR for each spatial value.
3. The quadratic split algorithm groups bottom-level MBRs into middle-level MBRs.
4. The quadratic split algorithm groups middle-level MBRs into top-level MBRs.
5. The bottom level of the index contains bottom-level MBRs and pointers to table blocks containing spatial values.
6. The middle level of the index contains middle-level MBRs and pointers to bottom-level index blocks.
7. The top level of the index contains top-level MBRs and pointers to middle-level index blocks.

In MySQL, the InnoDB and MyISAM storage engines automatically create R-tree indexes on spatial columns. NDB and ARCHIVE use B+tree indexes on spatial columns, resulting in relatively slow performance for many spatial functions. MySQL imposes some restrictions on R-tree indexes — for details, see 'Exploring Further', below.

PARTICIPATION ACTIVITY | 11.4.9: Spatial indexes.

1) The bottom level of an R-tree index is sparse.

- ○ True
- ○ False

2) If all blocks of an R-tree index are full, an insert to the spatial column always causes a block split.

- ○ True
- ○ False

3) Referring to the animation above, adding another MBR to Q will always increase X's area.

- ○ True
- ○ False

4) When a value is deleted from a column with an R-tree index, MBRs in

smaller.

○ True

○ False

CHALLENGE ACTIVITY 11.4.1: Spatial types.

544874.3500394.qx3zqy7

Start

Complete the rectangle:

```
   POLYGON( (0 0, 20 0, 20 (A), (B) 90, 0 0) )
```

(A) Ex: 50

(B)

1 2 3

Check Next

Exploring further:

- WKT format
- Spatial reference systems
- OGC standard
- MySQL spatial types
- MySQL spatial functions
- MySQL spatial indexes

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.

Animation captions:

PARTICIPATION ACTIVITY | 11.5.2: Object-orientation.

Refer to the animation above. Match the object-oriented capability with the Python example.

If unable to drag and drop, refresh the page.

Composite types **Methods** **Overriding** **Subtypes**

	GlobalAddress is derived from Address.
	Address has properties street, city, stateCode, and postalCode.
	printAddress() is associated with

	the Address class.
	printAddress() has the same name but different behavior in the Address and GlobalAddress classes.

Reset

Object-relational databases

As the industry struggled to merge objects and tables, three technologies emerged: object databases, object-relational mappings, and object-relational databases.

An ***object database*** adds database capabilities, such as persistence and transaction management, to an object-oriented language. The object database approach promised full database support for objects and seamless integration with object-oriented languages.

Approximately a dozen object databases are available. Leading products include ObjectStore and Action NoSQL Database (formerly Versant Object Database). Object databases are not widely used, however, for two reasons:

- Relational databases were mature and firmly entrenched when object databases arrived in the 1990s. Object databases and query languages could not displace relational databases and SQL.
- Database management and application programming have different technical requirements. Adding database capabilities to object-oriented languages is challenging and complex.

An ***object-relational mapping*** (***ORM***) is a software layer between the programming language and a relational database. The layer converts object structures and queries to relational structures and queries. ORMs simplify object-oriented programming while retaining a relational database. The primary disadvantage is that, in some cases, complex ORM queries are inefficient and must be written in SQL.

ORM technology is more successful than object databases. Leading ORM products include Django ORM, which maps Python language to relational databases, and Hibernate for Java.

An ***object-relational database*** extends SQL with an object type. The object type was incorporated in the SQL standard in 1999 with support for composite types, methods, and subtypes. Adding composite types to relational databases is straightforward. Methods and subtypes, however, are difficult to implement, administer, and query. As a result, most relational databases do not support an object type.

Some relational databases support subtables instead of subtypes. A ***subtable*** inherits columns and constraints from another table, called a ***supertable***. Subtables are limited compared to subtypes, since inheritance applies to entire tables rather than individual columns.

Figure 11.5.1: Object type support.

	Composite types	Methods	Subtypes
MySQL	-	-	-
Oracle Database	✔	✔	✔
PostgreSQL	✔	-	subtables
SQL Server	✔	-	-
DB2	✔	-	-
Informix	✔	-	subtables

Terminology

*In some databases, the term **object type** refers to catalog information and is unrelated to the standard SQL object type. Ex: In PostgreSQL, the object identifier type is the type of primary key columns of catalog tables. In SQL Server, the object type is the type of a database object, such as a view or foreign key, in the catalog.*

PARTICIPATION ACTIVITY 11.5.3: Object-relational database.

Match the technology with the disadvantage.

If unable to drag and drop, refresh the page.

Object-relational database **Object-relational mapping** **Object database**

Relational database

	Does not run on a relational database.
	Some complex queries are inefficient and must be rewritten in SQL.
	Inheritance is difficult to implement and administer.
	Difficult to query from an object-oriented programming language.

Reset

Object type

Oracle Database is a leading object-relational database. This section describes the Oracle Database object type.

CREATE TYPE AS OBJECT specifies a type name and associated properties, functions, and procedures.

Figure 11.5.2: CREATE TYPE AS OBJECT statement.

```
CREATE TYPE ObjectTypeName AS OBJECT (
   PropertyName Type,
   PropertyName Type,
   ...
   MEMBER FUNCTION FunctionName( <parameter>, <parameter> ... ) RETURN
ReturnType,
   MEMBER FUNCTION FunctionName( <parameter>, <parameter> ... ) RETURN
ReturnType,
   ...
   MEMBER PROCEDURE ProcedureName( <parameter>, <parameter> ... ),
   MEMBER PROCEDURE ProcedureName( <parameter>, <parameter> ... ),
   ...
);

<parameter>:
[ IN | OUT | INOUT ] ParameterName Type
```

CREATE TYPE BODY AS specifies the code, or ***body***, of each function and procedure associated with the object type. The code is written in PL/SQL, the Oracle procedural SQL language. Procedural SQL is described elsewhere in this material.

Figure 11.5.3: CREATE TYPE BODY AS statement.

```
CREATE TYPE BODY ObjectTypeName AS
   MEMBER FUNCTION FunctionName( <parameter>, <parameter> ... ) RETURN
ReturnType IS
   Body;
END;

CREATE TYPE BODY ObjectTypeName AS
   MEMBER PROCEDURE ProcedureName( <parameter>, <parameter> ... ) IS
   Body;
END;
```

An object type can define an individual column, much like a simple type. Alternatively, an object type can define an entire table. The ***CREATE TABLE OF*** statement creates a table from an object type. Each row of the table contains one value of the object type.

Figure 11.5.4: CREATE TABLE OF statement.

```
CREATE TABLE TableName OF
ObjectTypeName;
```

PARTICIPATION ACTIVITY 11.5.4: Creating an AddressType object with a function.

Animation captions:

PARTICIPATION ACTIVITY | 11.5.5: Object type.

1) A database method is:

- A function only
- A procedure only
- Either a function or procedure

2) How many methods can an object type have?

- At least one
- At most one
- Zero, one, or many

3) The Part column has type PartType, below. What is the correct format for inserting a value into the Part column?

```
CREATE TYPE PartType AS OBJECT
(
   Code CHAR(6),
   ListPrice NUMBER(10,2)
```

```
      ListPrice NUMBER(10,2),
      Description (VARCHAR(50)
   );
```

- ○ `Part('AD7Z82', 188.95, 'Left-handed hammer drive')`
- ○ `PartType('AD7Z82', 188.95, 'Left-handed hammer drive')`
- ○ `PartType('AD7Z82', '188.95', 'Left-handed hammer drive')`

Subtypes

Oracle Database subtypes are created with the ***CREATE TYPE UNDER*** statement. The subtype automatically inherits all supertype properties, functions, and procedures. Additional subtype properties, functions, and procedures can be specified in the CREATE TYPE UNDER statement.

To allow subtypes, the CREATE TYPE AS OBJECT statement for the supertype must specify ***NOT FINAL***.

Figure 11.5.5: CREATE TYPE UNDER statement.

```
CREATE TYPE SupertypeName AS OBJECT (
   ...
)
NOT FINAL;

CREATE TYPE SubtypeName UNDER SupertypeName (
   PropertyName Type,
   PropertyName Type,
   ...
   MEMBER FUNCTION FunctionName( <parameter>, <parameter> ... ) RETURN
ReturnType,
   MEMBER FUNCTION FunctionName( <parameter>, <parameter> ... ) RETURN
ReturnType,
   ...
   MEMBER PROCEDURE ProcedureName ( <parameter>, <parameter> ... ),
   MEMBER PROCEDURE ProcedureName ( <parameter>, <parameter> ... ),
   ...
);
```

A subtype can override inherited functions and procedures. Overriding replaces supertype code with new subtype code for a procedure or function. The procedure or function name must be the same in the supertype and subtype. The ***OVERRIDING*** keyword must appear in the CREATE TYPE

UNDER and CREATE TYPE BODY statements for the subtype.

Figure 11.5.6: Overriding functions.

```
CREATE TYPE SubtypeName UNDER SupertypeName (
  OVERRIDING MEMBER FUNCTION FunctionName( <parameter>, <parameter> ... )
RETURN ReturnType
);

CREATE TYPE BODY SubtypeName AS
  OVERRIDING MEMBER FUNCTION FunctionName( <parameter>, <parameter> ... )
RETURN ReturnType IS
   Body;
END;
```

Subtypes and overriding are difficult to design and administer in a database. Inheritance is commonly used in object-oriented programming languages but not in relational databases.

PARTICIPATION ACTIVITY 11.5.6: Creating a GlobalAddressType subtype.

Animation captions:

PARTICIPATION ACTIVITY | 11.5.7: Subtypes.

Refer to the code below.

```
CREATE TYPE Employee AS ___A___ (
   ID INT,
   Name VARCHAR(30),
   EmailAddress VARCHAR(20),
   SalaryAmount NUMBER(12, 2),
   MEMBER FUNCTION TotalCompensation RETURN NUMBER
)
NOT ___B___;

CREATE TYPE ExecutiveEmployee ___C___ Employee (
   BonusAmount NUMBER(10,2),
   ___D___ MEMBER FUNCTION TotalCompensation RETURN NUMBER
);
```

The TotalCompensation function returns the salary for an Employee and (salary + bonus) for an ExecutiveEmployee. The `CREATE TYPE BODY` statements are not shown.

1) What is keyword A?

Check   Show answer

2) What is keyword B?

Check   Show answer

3) What is keyword C?

Check   Show answer

4) What is keyword D?

Check Show answer

Exploring further:

- Object databases
- ORMs
- ORM products
- Oracle Database object type

11.6 Database programming for the web

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.

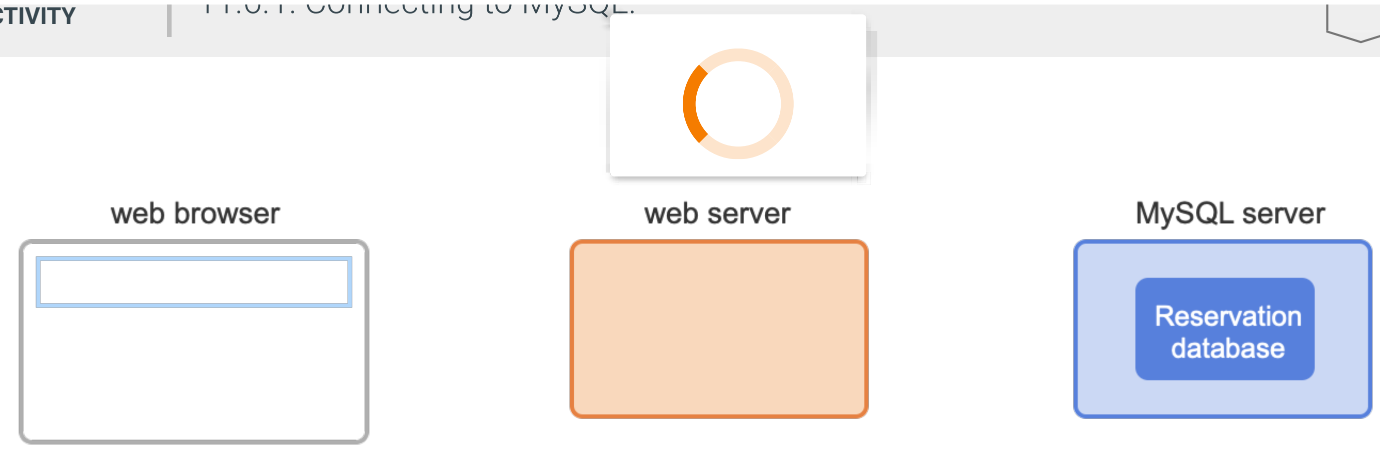
PHP scripts must connect to a database prior to executing queries. A PHP script establishes a connection by creating a ***PDO object***. The PDO constructor specifies a DSN and a MySQL username and password. A ***Data Source Name*** (***DSN***) is a string containing database connection information:

- 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 11.6.1: Connecting to MySQL

ACTIVITY | 11.6.1: Connecting to MySQL.



Animation captions:

PARTICIPATION ACTIVITY | 11.6.2: Connecting to a database.

1) What DSN value is omitted from the animation above?

- DSN prefix
- host
- port
- dbname

2) What values are required to create a PDO object?

- Username and password only

○ Hostname only

○ Hostname, database name, username, and password

3) Referring to the animation above, why might the PDO constructor throw a PDOException?

○ The username and password are correct.

○ The MySQL server is running on localhost.

○ The Reservation database does not exist.

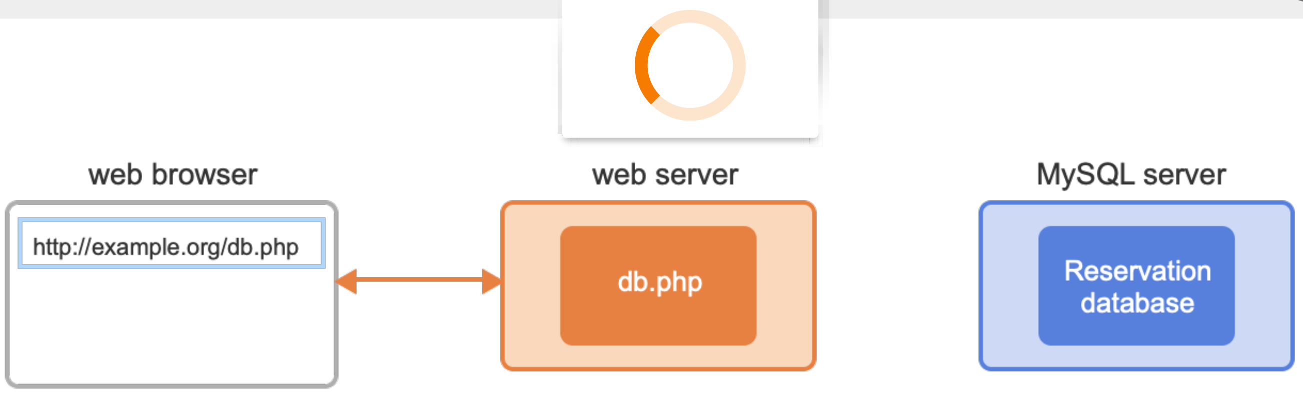
Executing statements

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.



Animation captions:

PARTICIPATION ACTIVITY 11.6.4: Executing statements.

Refer to the animation above.

1) Removing the call to setAttribute() stops the query() method from executing the SQL statement.

   ○ True

   ○ False

2) If FlightNumber is the primary key, executing the INSERT statement twice results in a PDOException being thrown.

   ○ True

   ○ False

3) If $sql is changed to insert two flights, the call to rowCount() returns 2.

   ○ True

   ○ False

Prepared statements

Prepared statements

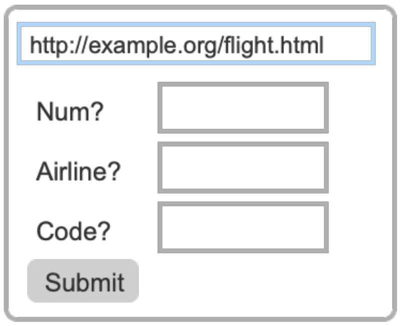
An SQL statement may be created dynamically from user input. A ***prepared statement*** is an SQL statement that can be customized with parameter values retrieved from user input. Prepared statements are immune to most SQL injection attacks.

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.

addflight.php



Animation captions:

PARTICIPATION ACTIVITY 11.6.6: Prepared statements.

1) Which values are missing?

```
$sql = "UPDATE Flight " .
       "SET AirlineName = ? "
.
       "WHERE FlightNumber =
?";

$statement = $pdo-
>prepare($sql);
$statement->bindValue(1,
____);
$statement->bindValue(2,
____);
$statement->execute();
```

- "United Airlines" and "PEK"
- 305 and "United Airlines"
- "United Airlines" and 305

2) Which values are missing?

```
$sql = "UPDATE Flight " .
       "SET AirlineName =
:airlineName " .
       "WHERE FlightNumber =
:flightNum";

$statement = $pdo-
>prepare($sql);
$statement->bindValue(____,
"Southwest Airlines");
$statement->bindValue(____,
445);
$statement->execute();
```

- AirlineName and FlightNumber
- "airlineName" and "flightNum"

○ :airlineName and :flightNum

3) Assume the airline and flightNum values are obtained from a submitted form. What is wrong with the following code?

```
$sql = "UPDATE Flight " .
       "SET AirlineName =
'$_POST[airline]'" .
       "WHERE FlightNumber =
$_POST[flighNum]";

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

○ The SQL statement is missing parameter identifiers.

○ The code is susceptible to an SQL injection attack.

○ Nothing is wrong.

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.

Animation captions:

PARTICIPATION ACTIVITY | 11.6.8: Fetching values.

Refer to the code below.

```
$flightNum = 140;
$sql = "SELECT AirlineName, DepartureTime, AirportCode FROM Flight " .
       "WHERE FlightNumber = ?";
$statement = $pdo->prepare($sql);
$statement->bindValue(___A___, $flightNum);
$statement->execute();

$row = $statement->___B___();
if ($row) {
   echo "$row[AirlineName] $flightNum departs from $row[___C___]";
}
else {
   echo "Flight not found";
```

1) What is identifier A?

Check Show answer

2) What is identifier B?

Check Show answer

3) What is identifier C?

Check Show answer

Stored procedure calls

Stored procedures are part of MySQL procedural SQL. Stored procedures are saved in the database and may be called from a PHP script.

Stored procedures are called with a prepared statement. Values are bound to the stored procedure's IN parameters. Stored procedures with an OUT or INOUT parameter require a call with an SQL user-defined variable. A second query is required to fetch the value of the user-defined variable that is holding the output parameter value.

PARTICIPATION ACTIVITY 11.6.9: Calling a stored procedure to get flight count.

Animation captions:

PARTICIPATION ACTIVITY | 11.6.10: Stored procedure call with PDO.

Refer to the stored procedure that, given an airport and airline, finds the flight number of the last flight. Place the PHP code in order.

```
CREATE PROCEDURE LastFlight(IN airport CHAR(3), IN airline VARCHAR(45),
OUT flightNum INT)
BEGIN
    SELECT FlightNumber
    INTO flightNum
    FROM Flight
    WHERE AirportCode = airport AND AirlineName = airline
    ORDER BY DepartureTime DESC
    LIMIT 1;
END;
```

If unable to drag and drop, refresh the page.

```
$statement = $pdo->query("SELECT @flightNum AS flightNum");
$row = $statement->fetch();
```

```
$statement->execute();
```

```
echo "Flight number = $row[flightNum]";
```

```
$statement->bindValue(1, "DEN");
$statement->bindValue(2, "United Airlines");
```

```
$statement = $pdo->prepare("CALL LastFlight(?, ?, @flightNum)");
```

Step 1

	Step 2
	Step 3
	Step 4
	Step 5

Reset

Exploring further:

- PHP Data Objects documentation from PHP.net

11.7 LAB - Database programming with Python (SQLite)

Complete the Python program to create a Horse table, insert one row, and display the row. The main program calls four functions:

1. `create_connection()` creates a connection to the database.
2. `create_table()` creates the Horse table.
3. `insert_horse()` inserts one row into Horse.
4. `select_all_horses()` outputs all Horse rows.

Complete all four functions. Function parameters are described in the template. Do not modify the main program.

The Horse table should have five columns, with the following names, data types, constraints, and values:

Name	Data type	Constraints	Value
Id	integer	primary key, not null	1
Name	text		'Babe'

Breed	text		'Quarter horse'
Height	double		15.3
BirthDate	text		'2015-02-10'

The program output should be:

```
All horses:
(1, 'Babe', 'Quarter Horse', 15.3, '2015-02-10')
```

This lab uses the SQLite database rather than MySQL. The Python API for SQLite is similar to MySQL Connector/Python. Consequently, the API is as described in the text, with a few exceptions:

- Use the import library provided in the program template.
- Create a connection object with the function `sqlite3.connect(":memory:")`.
- Use the character `?` instead of `%s` as a placeholder for query parameters.
- Use data type `text` instead of `char` and `varchar`.

SQLite reference information can be found at SQLite Python Tutorial, but is not necessary to complete this lab.

544874.3500394.qx3zqy7

LAB ACTIVITY 11.7.1: LAB - Database programming with Python (SQLite) 10 / 10

main.py **Load default template...**

```python
import sqlite3
from sqlite3 import Error

# Creates connection to sqlite in-memory database
def create_connection():
    """
    Create a connection to in-memory database
    :return: Connection object
    """
    try:
        conn = sqlite3.connect(":memory:")
     #  print("Connection to SQLite in-memory database successful")
        return conn
    except Error as e:
        print(e)
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **main.py** (Your program) → Outp

Program output displayed here

Coding trail of your work [What is this?]

3/4 M10 min:1

11.8 LAB - Database programming with Java (SQLite)

Complete the Java program to create a Horse table, insert one row, and display the row. The main program calls four methods:

1. `createConnection()` creates a connection to the database.
2. `createTable()` creates the Horse table.
3. `insertHorse()` inserts one row into Horse.
4. `selectAllHorses()` outputs all Horse rows.

Complete all four methods. Method parameters are described in the template. Do not modify the main program.

The Horse table should have five columns, with the following names, data types, constraints, and values:

Name	Data type	Constraints	Value
Id	integer	primary key, not null	1
Name	text		'Babe'
Breed	text		'Quarter horse'
Height	double		15.3
BirthDate	text		'2015-02-10'

The program output should be:

```
All horses:
(1, 'Babe', 'Quarter Horse', 15.3, '2015-02-10')
```

This lab uses the SQLite database rather than MySQL. Both SQLite and MySQL Connector/J implement the JDBC API. Consequently, the API is as described in the text, with a few exceptions:

- Use the connection string "jdbc:sqlite::in-memory" to connect to an in-memory database.
- Use the `text` data type instead of `char` and `varchar`.

SQLite reference information can be found at SQLite Java Tutorial, but is not necessary to complete this lab.

544874.3500394.qx3zqy7

LAB ACTIVITY 11.8.1: LAB - Database programming with Java (SQLite) 10 / 10

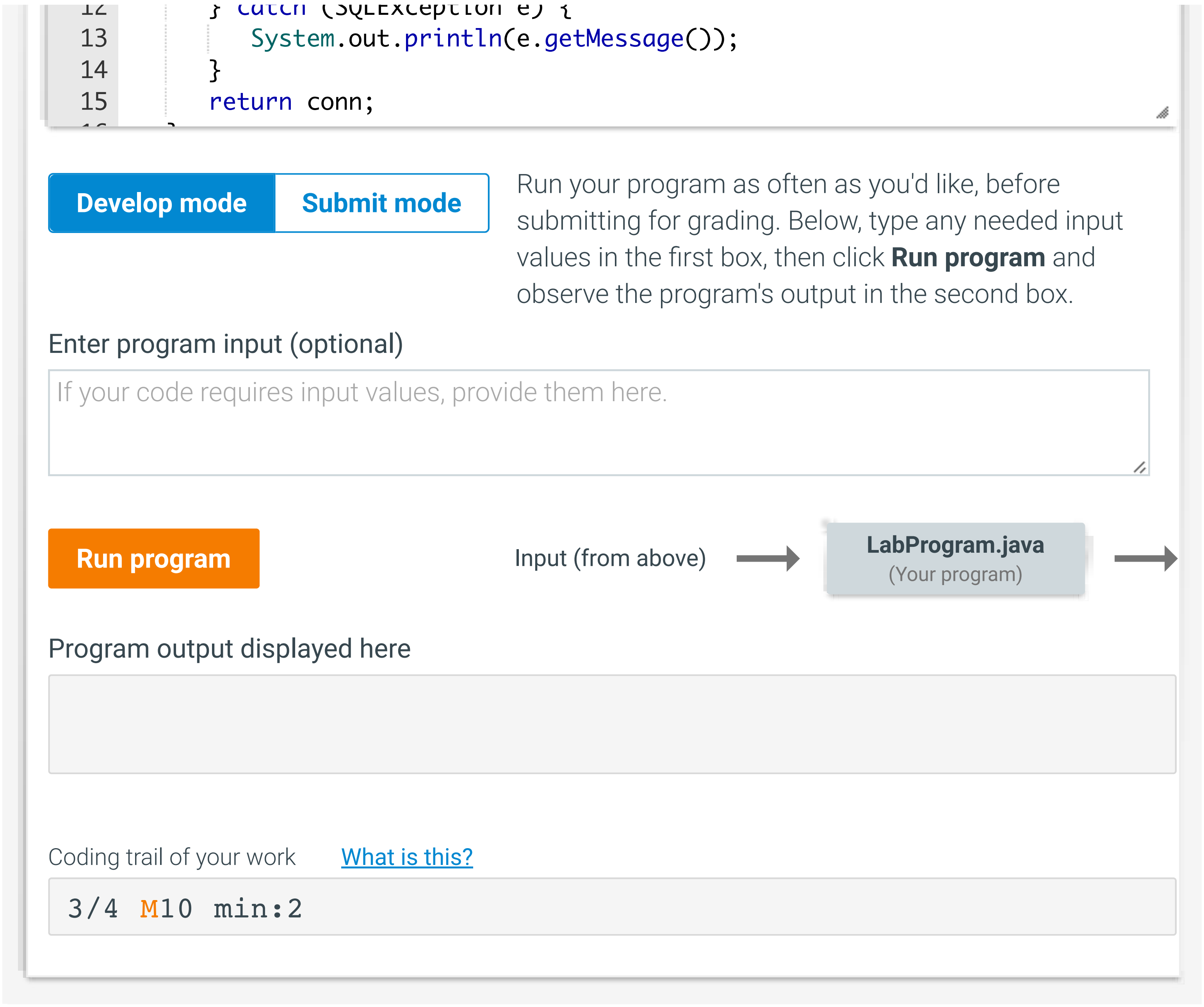
LabProgram.java

Load default template...

```
import java.sql.*;

public class LabProgram {

   // Create a connection to a sqlite in-memory database
   // Returns Connection object
   public static Connection createConnection() {
      Connection conn = null;
      try {
         // Use connection string "jdbc:sqlite::memory:" to connect to an i
         conn = DriverManager.getConnection("jdbc:sqlite::memory:");
      } catch (SQLException e) {
```



```
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        return conn;
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **LabProgram.java** (Your program) →

Program output displayed here

Coding trail of your work [What is this?]

```
3/4 M10 min:2
```