4.1 Relational databases

Learning goals

- Define relational databases.
- Differentiate files, relational databases, and NoSQL databases.
- Define database tables, null values, and keys.
- Define the structure of a SQL statement.

Relational databases

Many tasks in data science involve extracting and manipulating data from a database. Although files and spreadsheets are commonly used when working with small and static data sets, large data sets are handled by multiple users from different geographic regions, which presents many challenges. Complex data sets are usually managed in database systems, for several reasons:

- *Performance* - Databases optimize data organization on storage media for fast query processing.
- *Security* - Databases prevent unauthorized data access and encrypt data in case of a security breach.
- *Concurrency* - Databases manage conflicts that occur when concurrent users update the same data.
- *Recovery* - Databases restore the database to a consistent state after system failures.

Most leading database systems are relational. A ***relational database*** organizes data in rows and columns of tables. Tables are related with special columns, called keys, and queried with Structured Query Language (SQL). Ex: The figure below illustrates two tables in a relational database.

Figure 4.1.1: Relational database example.

Country

CountryCode	CountryName	SurfaceArea	ContinentName
JPN	Japan	377829	Asia
ITA	Italy	301316	Europe
ECU	Ecuador	283561	South America

City

CityName	CountryCode	Population
Osaka	JPN	2595674
Kyoto	JPN	1461974
Rome	ITA	2643581

CityName	CountryCode	Population
Verona	ITA	255268
Naples	ITA	1002619
Quito	ECU	1573458

PARTICIPATION ACTIVITY 4.1.1: Relational databases.

1) Concurrency involves _____.

- the speed in which queries are processed
- handling conflicts when multiple users update data at the same time
- the ability to restore data after a system failure

2) A relational database consists of _____.

- data types
- keys
- tables

©zyBooks 03/21/24 21:41 2087217
Biniam abebe
UNTADTA5340OrhanSpring8wk22024

3) The example below is a relational database.

```
{ "Countries":
   [  { "CountryCode": "JPN",
        "CountryName": "Japan",
        "SurfaceArea": 377829,
        "ContinentName": "Asia",
        "Cities":
           [  { "CityName":
"Osaka", "Population":2595674 }
              { "CityName":
"Kyoto", "Population":1461974 }
           ]
      }
      { "CountryCode": "ITA",
        "CountryName": "Italy",
        "SurfaceArea": 301316,
        "ContinentName":
"Europe",
        "Cities":
              [  { "CityName":
"Rome", "Population":2643581 }
                 { "CityName":
"Verona", "Population":255268 }
                 { "CityName":
"Naples", "Population":1002619 }
              ]
      }
      { "CountryCode": "ECU",
        "CountryName": "Ecuador",
        "SurfaceArea": 283561,
        "ContinentName": "South
America",
        "Cities":
                 { "CityName":
"Quito", "Population":1573458 }
      }
   ]
}
```

- True
- False

Tables

Relational data is structured in tables:

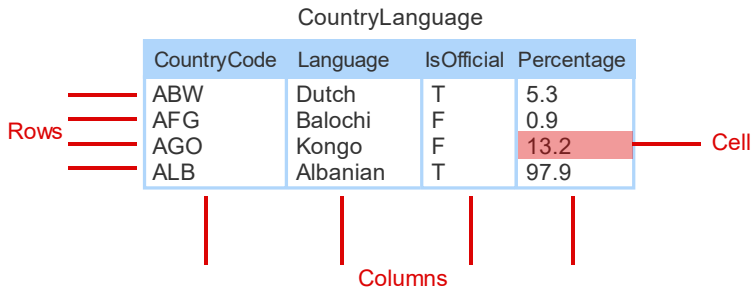
- A ***table*** has a name, a fixed sequence of columns, and a varying set of rows.
- A ***column*** has a name and a data type.
- A ***row*** is an unnamed sequence of values. Each value corresponds to a column and belongs to the column's data type.
- A ***cell*** is a single column of a single row.

A table has at least one column and any number of rows. Tables obey the following rules:

- *One value per cell*. A cell cannot contain multiple values.
- *No duplicate column names*. Duplicate column names are allowed in different tables, but not in the same table.
- *No duplicate rows*. No two rows have identical values in all columns.
- *No row order*. The organization of rows on storage media, such as a disk drive, never affects query results.

The last rule is called *data independence*. Data independence allows tuning physical data organization to improve performance without changing any queries.

PARTICIPATION ACTIVITY 4.1.2: Tables.



Animation content:

Step 1: A table appears with four rows and four columns. Rows are highlighted with arrows. Step 2-3: Columns are highlighted. Step 4: A cell in the table is a single entry at the intersection of one row and one column. The cell is highlighted.

Animation captions:

1. The CountryLanguage table has four rows. All rows are different.
2. CountryLanguage has four columns, named CountryCode, Language, IsOfficial, and Percentage.
3. All column names are different.
4. Each cell contains one value.

PARTICIPATION ACTIVITY 4.1.3: Table rules.

Each example violates a table rule. Map the rule violation to the invalid table.

If unable to drag and drop, refresh the page.

Several values per cell **Duplicate rows** **Duplicate column names**

Country

CountryCode	CountryName	SurfaceArea	ContinentName
JPN	Japan	377829	Asia
ITA	Italy Italia	301316	Europe
ECU	Ecuador	283561	South America

Country

CountryCode	CountryName	SurfaceArea	ContinentName
JPN	Japan	377829	Asia
ITA	Italy	301316	Europe
ECU	Ecuador	283561	South America
ITA	Italy	301316	Europe

Country

Code	Name	SurfaceArea	Name
JPN	Japan	377829	Asia
ITA	Italy	301316	Europe
ECU	Ecuador	283561	South America

Reset

Null values

Occasionally, the data in a cell is unknown or inapplicable. Relational databases represent unknown and inapplicable data as a special ***NULL*** value. Ex: In the table below, the surface area of Italy is unknown. The capital of Antarctica is inapplicable, since Antarctica has no capital city.

Figure 4.1.2: Unknown and inapplicable null values.

Country

CountryCode	CountryName	SurfaceArea	CapitalName
JPN	Japan	377829	Tokyo
ITA	Italy	NULL	Rome
ECU	Ecuador	283561	Quito
ATA	Antarctica	13120000	NULL

NULL is not the same as zero or an empty string. Ex: A zero balance in a bank account is a known, applicable amount. A blank comment in a survey may represent a known, applicable response.

When NULL appears in an arithmetic or comparison operation, the result is always NULL. Ex: The following expressions evaluate to NULL:

- `4 + NULL`
- `11.3 <= NULL`
- `NULL = NULL`

When NULL appears with AND and OR, the result is sometimes NULL, as described in the truth table below. Ex: `TRUE AND NULL` evaluates to NULL. `TRUE OR NULL` evaluates to TRUE.

The result of `NOT NULL` is NULL.

Figure 4.1.3: Null value truth table.

x	y	x AND y	x OR y
TRUE	NULL	NULL	TRUE
NULL	TRUE		
FALSE	NULL	FALSE	NULL
NULL	FALSE		
NULL	NULL	NULL	NULL

PARTICIPATION ACTIVITY 4.1.4: Null values.

1) What is the result of NULL = NULL?
   - TRUE
   - FALSE
   - NULL

2) What is the result of 5 > NULL?
   - TRUE
   - FALSE
   - NULL

3) What is the result of TRUE OR NULL?
   - TRUE
   - FALSE
   - NULL

4) What is the result of TRUE OR (5 > NULL)?
   - TRUE
   - FALSE
   - NULL

5) What is the result of FALSE OR (5 > NULL)?
   - TRUE
   - FALSE
   - NULL

Keys

A ***unique column*** never contains duplicate values in different rows. Each value appears in one row only, regardless of inserts and updates to the table.

Every table has a primary key. A ***primary key*** is a column, or group of columns, that identifies individual rows. To ensure that each primary key value corresponds to exactly one row, primary keys must be unique and not NULL.

A ***foreign key*** is a column, or group of columns, that refer to a primary key. Foreign key values must either be NULL or mat some value of the referenced primary key. This rule prevents references to non-existent rows. Unlike primary keys, foreign keys are not necessarily unique.

The data types of the foreign and primary keys must be the same, but the names may be different. A foreign key is usually in a different table than the referenced primary key, but may be in the same table.

In table diagrams, the primary key is marked by a solid circle (●) and appears as the leftmost column of the table . A foreign key is marked by an empty circle (○) and an arrow leading to the referenced primary key.

PARTICIPATION ACTIVITY 4.1.5: Primary and foreign keys.

City

● CityNumber	CityName	○ CountryCode	Population
1534	Osaka	JPN	2595674
1537	Kyoto	JPN	1461974
1464	Rome	ITA	2643581
1476	Verona	ITA	255268
1466	Naples	ITA	1002619
594	Quito	ECU	1573458

Country

● CountryCode	CountryName	SurfaceArea	Continer
JPN	Japan	377829	Asia
ITA	Italy	301316	Europe
ECU	Ecuador	283561	South Ar

Animation content:

There are two tables named Department and Employee. Department has three columns named Code Name and Manager. Employee has three columns named ID Name and Salary. Manager is a foreign key of primary key ID. Slide 2. Rows one of column Manager of table Department and one column ID of table Employee are highlighted and both contain the value 2538. Slide 3. Rows two and three of column Manager of table Department and row three of column ID of table Employee are highlighted and all contain the value 6381. Slide 4. Row four of column Manager of table Department is highlighted and contains the value NULL.

Animation captions:

1. CityNumber is the primary key of City. CountryCode is the primary key of Country.
2. Primary key values are unique and not null.
3. CountryCode in the City table is a foreign key and refers to the Country table.
4. Foreign key values either match the referenced primary key or are NULL.

PARTICIPATION ACTIVITY 4.1.6: Foreign keys.

In the tables below, the Manager column is a foreign key that refers to the Employee table.

Department

Employee

● Code	Name	○ Manager
44	Engineering	2538
82	Sales	6381
12	Marketing	6381
99	Technical support	NULL

● ID	Name	Salary
2538	Lisa Ellison	45000
5384	Sam Snead	30500
6381	Maria Rodriguez	92300

1) The data type of Manager and ID must be the same.
   - ○ True
   - ○ False

2) NULL in the Manager column refers to an Employee row with a NULL ID.
   - ○ True
   - ○ False

3) Values in a foreign key must be unique.
   - ○ True
   - ○ False

4) Sam Snead does not manage a department.
   - ○ True
   - ○ False

5) Replacing NULL in the Manager column with 5384 assigns Sam Snead as the manager to the technical support department.
   - ○ True
   - ○ False

6) The NULL in the Manager column may be replaced with 9876.
   - ○ True
   - ○ False

Structured Query Language

Structured Query Language (***SQL***) is a computer language for manipulating and retrieving data. SQL is the standard language for relational databases and is also supported by many non-relational databases. SQL is pronounced either "S-Q-L " or "seekwəl ".

SQL consists of language elements:

- A ***keyword*** is a reserved word with special meaning.
- A ***literal*** is a fixed numeric or text value. Text literals are enclosed in single or double quotes.
- An ***identifier*** is a user-defined name for a table, column, database, and so on.
- A ***clause*** groups a keyword with identifiers and expressions.

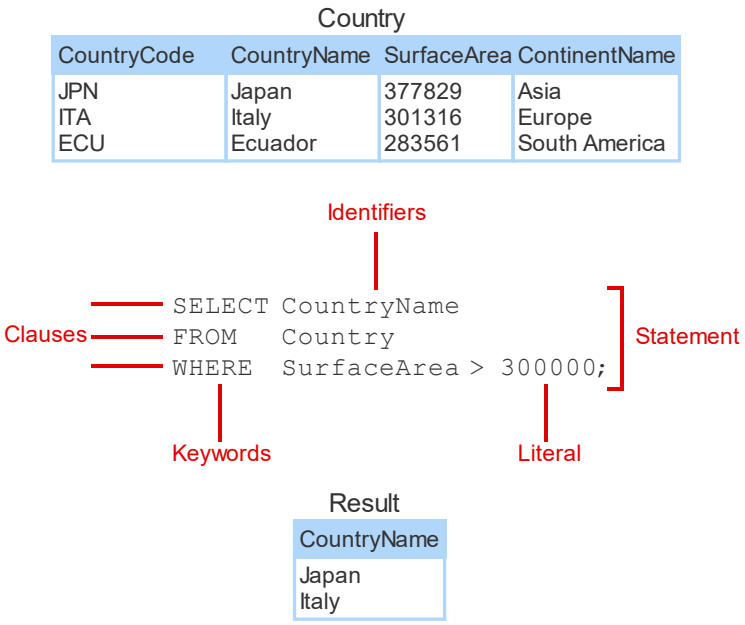
- A ***statement*** is a complete database action, consisting of one or more clauses and ending with a semicolon.

In most database systems, keywords and identifiers are not case-sensitive. Ex: *SELECT* and *select* are equivalent.

SQL ignores white space, such as line breaks. Although a statement can be formatted on one line, each clause usually appears on a separate line for clarity.

Some statements create and drop tables and indexes, while others retrieve, insert, update, and delete data. This material emphasizes statements that retrieve data, called queries.

PARTICIPATION ACTIVITY 4.1.7: SQL language elements.



Animation content:

There are two tables named Department and Employee. Department has three columns named Code Name and Manager. Employee has three columns named ID Name and Salary. Manager is a foreign key of primary key ID. Slide 2. Rows one of column Manager of table Department and one column ID of table Employee are highlighted and both contain the value 2538. Slide 3. Rows two and three of column Manager of table Department and row three of column ID of table Employee are highlighted and all contain the value 6381. Slide 4. Row four of column Manager of table Department is highlighted and contains the value NULL.

Animation captions:

1. SELECT, FROM, and WHERE are keywords.
2. 30000 is a numeric literal.
3. CountryName, Country, and SurfaceArea are identifiers.
4. Each clause appears on a separate line.
5. The three lines end with a semicolon and form a statement.
6. The statement selects the names of countries with area greater than 300,000.

PARTICIPATION ACTIVITY 4.1.8: SQL language elements.

1) The INSERT statement adds a student to the Student table. How many clauses are in the statement?

```
INSERT INTO Student
VALUES (888, 'Smith', 'Dana',
3.0);
```


- 1
- 2
- 3

2) The statement below selects students named Boscano. What is wrong with the statement?

```
SELECT FirstName
FROM Student
WHERE LastName = Boscano;
```

- *Boscano* must be enclosed in single or double quotes.
- The WHERE clause should be removed.
- The last name *Boscano* may not exist in the database.

3) What is wrong with the statement below?

```
SELECT FirstName
from Student
```

- The keyword from must be written FROM.
- The WHERE clause is missing.
- A terminating semicolon is missing.

4) What is wrong with the statement below?

```
SELECT EmailAddress
-- FROM Student;
```


- The FROM clause is a comment.
- The WHERE clause is missing.
- EmailAddress should be a table name.

CHALLENGE ACTIVITY 4.1.1: Nulls and keys.

537150.4174434.qx3zqy7

Start

What is the result of the following expression?

```
( NOT (93 >= NULL) ) AND (19 = 34)
```

Pick

1 2 3

Check Next

4.2 Simple queries

Learning goals

- Define and compare arithmetic, comparison, and logical operators.
- Use operator precedence rules to evaluate expressions.
- Construct simple queries with SELECT, FROM, and WHERE clauses.

Operators

An ***operator*** is a symbol that computes a value from one or more other values, called ***operands***:

- Arithmetic operators compute numeric values from numeric operands.
- Comparison operators compute logical values TRUE or FALSE. Operands may be numeric, character, and other data types.
- Logical operators compute logical values from logical operands.

A ***unary*** operator has one operand. A ***binary*** operator has two operands. Most operators are binary. The logical operator NOT is unary. The arithmetic operator – is either unary or binary.

Arithmetic and comparison operators return NULL when either operand is NULL. Logical operators with NULL operands are discussed elsewhere in this material.

Table 4.2.1: Common operators.

Type	Operator	Description	Example	Value
Arithmetic	+	Adds two numeric values	`4 + 3`	7
	– (unary)	Reverses the sign of one numeric value	`-(-2)`	2
	– (binary)	Subtracts one numeric value from another	`11 - 5`	6
	*	Multiplies two numeric values	`3 * 5`	15
	/	Divides one numeric value by another	`4 / 2`	2
	% (modulo)	Divides one numeric value by another and returns the integer remainder	`5 % 2`	1
	^	Raises one numeric value to the power of another	`5^2`	25
Comparison	=	Compares two values for equality	`1 = 2`	FALSE

	!=	Compares two values for inequality	1 != 2	TRUE
	<	Compares two values with <	2 < 2	FALSE
	<=	Compares two values with ≤	2 <= 2	TRUE
	>	Compares two values with >	'2019-08-13' > '2021-08-13'	FALSE
	>=	Compares two values with ≥	'apple' >= 'banana'	FALSE
Logical	AND	Returns TRUE only when both values are TRUE	TRUE AND FALSE	FALSE
	OR	Returns FALSE only when both values are FALSE	TRUE OR FALSE	TRUE
	NOT	Reverses a logical value	NOT FALSE	TRUE

PARTICIPATION ACTIVITY 4.2.1: Operators.

Match the operator with the description.

If unable to drag and drop, refresh the page.

AND **!=** **-** **NOT** **%**

	Binary arithmetic operator
	Either unary or binary arithmetic operator
	Binary comparison operator
	Unary logical operator
	Binary logical operator

Reset

Expressions

An ***expression*** is a string of operators, operands, and parentheses that evaluates to a single value. Operands may be colu names or fixed values. The value of an expression may be any data type. Ex:
`Salary > 34000 AND Department = 'Marketing'` is an expression with a logical value.

A simple expression may consist of a single column name or a fixed value. Ex: The column `EmployeeName` and the fixed value `'Maria'` are expressions with a character data type.

When an expression is evaluated, column names are replaced with column values for a specific row. Consequently, an expression containing column names may have different values for different rows.

The order of operator evaluation may affect the value of an expression. Operators in an expression are evaluated in the order of ***operator precedence***, shown in the table below. Operators of the same precedence are evaluated from left to right. Regardless of operator precedence, expressions enclosed in parentheses are evaluated before any operators outside the parentheses are applied.

Table 4.2.2: Operator precedence.

Precedence	Operators
1	– (unary)
2	^
3	* / %
4	+ – (binary)
5	= != < > <= >=
6	NOT
7	AND
8	OR

PARTICIPATION ACTIVITY 4.2.2: Evaluating expressions.

Expression: Status = 'Platinum' AND (Quantity * UnitPrice + ShippingPrice) > 100.00

'Platinum' = 'Platinum' AND (4 * 15.00 + 7.50) > 100.00

'Platinum' = 'Platinum' AND (60.00 + 7.50) > 100.00

'Platinum' = 'Platinum' AND 67.50 > 100.00

TRUE AND 67.50 > 100.00

	TRUE	AND	FALSE
Value:		FALSE	

Animation content:

Animation shows the process of evaluating an expression.

Step 1: An expression appears.
Status = 'Platinum' AND (Quantity * UnitPrice + ShippingPrice) > 100

Step 2: Status, Quantity, UnitPrice, and ShippingPrice are replaced with values from a row in the dataset.
'Platinum' = 'Platinum' AND (4*15 + 7.50) > 100

Step 3: The multiplication operator inside the parentheses is evaluated first.
'Platinum' = 'Platinum' AND (60 + 7.50) > 100

Step 4: The addition operator is evaluated.
'Platinum' = 'Platinum' AND (67.50) > 100

Step 5: The equality operator is evaluated. 'Platinum' = 'Platinum' evaluates to TRUE.
TRUE AND (67.50) > 100

Step 6: The greater than operator is evaluated. 67.50 > 100 evaluates to FALSE.
TRUE AND FALSE

Step 7: The expression evaluates to FALSE.

Animation captions:

1. The expression includes column names Status, Quantity, UnitPrice, and ShippingPrice.
2. The expression is evaluated for a specific row. Column names are replaced with column values for the row.
3. Operators within parentheses have the highest precedence. * has higher precedence than + and is evaluated first.
4. + is the only remaining operator inside the parentheses and is evaluated next.
5. = has higher precedence than AND. = has the same precedence as > but appears to the left. So = is evaluated next.
6. > has higher precedence than AND and is evaluated next.
7. The expression evaluates to FALSE for the row.

PARTICIPATION ACTIVITY 4.2.3: Expressions.

What is the value of the following expressions?

1) `7 + 3 * 2`

- 13
- 20

2) `(7 + 3) * 2`

- 13
- 20

3) `(8 % 3 + 10 > 15) AND TRUE`

- TRUE
- FALSE

4) `(Age >= 13 AND Age <= 18) OR Military = 'Army'`
where Age = 8 and Miltary = 'Army'

- TRUE
- FALSE

SELECT statement

The SELECT statement selects rows from a table. The statement has a ***SELECT*** clause and a ***FROM*** clause. The FROM clause specifies the table from which rows are selected. The SELECT clause specifies one or more expressions, separated by commas, that determine what values are returned for each row.

Figure 4.2.1: SELECT with expressions.

```
SELECT Expression1, Expression2,
...
FROM TableName;
```

In many queries, each expression in the SELECT clause is a simple column name.

Figure 4.2.2: SELECT with columns.

```
SELECT Column1, Column2,
...
FROM TableName;
```


To select all columns, the expressions can be replaced with a single asterisk.

Figure 4.2.3: SELECT with asterisk.

```
SELECT *
FROM
TableName;
```

The SELECT statement returns a set of rows, called the ***result table***.

PARTICIPATION ACTIVITY 4.2.4: Selecting from table CountryLanguage.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
ALB	Albanian	T	97.9

```
SELECT *
FROM CountryLanguage;
```

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
ALB	Albanian	T	97.9

```
SELECT CountryCode, Language
FROM CountryLanguage;
```

CountryCode	Language
ABW	Dutch
AFG	Balochi
AGO	Kongo
ALB	Albanian

Animation content:

Slide 1. There is a table named Country Language and has four columns named Country Code Language Is Official and Percentage. Slide 2. Two lines of code appear below. Line one of code states SELECT asterisk. Line two of code states FROM Country Language semicolon. An unnamed table appears with the same columns as table Country Language. Then all the data inside the table Country Language is duplicated and moved into the unnamed table. Slide 3. Two new lines of code appears to the right. Line one of code states SELECT CountryCode comma Language. Line two of code states FROM Country Language semicolon. An unnamed table appears and has two columns named Country Code and Language. The data from columns Country Code and Language in table Country Language are duplicated and moved into the new unnamed table.

Animation captions:

1. The CountryLanguage table contains four rows with columns CountryCode, Language, IsOfficial, and Percentage.
2. The SELECT statement selects all columns using *. All rows and columns appear in the result table.
3. The SELECT statement selects only columns CountryCode and Language, so only two columns appear in the result table.

PARTICIPATION ACTIVITY 4.2.5: SELECT with columns.

Refer to the following table.

City

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabul	1780000
2	Kandahar	AFG	Kandahar	237500
3	Amsterdam	NLD	Noord-Holland	731200
4	Rotterdam	NLD	Zuid-Holland	593321
5	Sydney	AUS	New South Wales	3276207

1) Select all rows and only the Name and District columns.

```
SELECT
FROM City;
```

Check Show answer

2) Select all rows and columns.

```
SELECT
FROM City;
```

Check Show answer

3) Select all rows and columns except ID in order of columns shown.

```
SELECT
FROM City;
```

Check Show answer

PARTICIPATION ACTIVITY 4.2.6: SELECT with expressions.

Refer to the following table.

Customer

ID	Name	Balance	Payment
193	Chen	2100	300
584	Ravindran	5000	250
231	Bolt	300	10
608	Gomez	950	125

1) What values are returned?

```
SELECT Balance + Payment
FROM Customer;
```

- 2400, 5250, 310, 1075
- 2400
- 2100, 5000, 300, 950

2) What values are returned?

```
SELECT 2 * (Balance - Payment)
FROM Customer;
```

- 3900, 9750, 590, 1775
- 3600
- 3600, 9500, 580, 1650

WHERE clause

An expression may return a value of any data type. A ***condition*** is an expression that evaluates to a logical value.

A SELECT statement has an optional ***WHERE*** clause that specifies a condition for selecting rows. A row is selected when the condition is TRUE for the row values. A row is omitted when the condition is either FALSE or NULL.

The WHERE clause follows the FROM clause. When a SELECT statement has no WHERE clause, all rows are selected.

Figure 4.2.4: WHERE clause.

```
SELECT Expression1, Expression2,
...
FROM TableName
WHERE Condition;
```

PARTICIPATION ACTIVITY 4.2.7: WHERE clause.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
ALB	Albanian	T	97.9

```
SELECT CountryCode, Language
FROM CountryLanguage
WHERE Percentage > 0.0
 AND Percentage < 10.0;
```

CountryCode	Language
ABW	Dutch
AFG	Balochi

```
SELECT CountryCode, Language
FROM CountryLanguage
WHERE Percentage < 5.0
 OR Percentage > 90.0;
```

CountryCode	Language
AFG	Balochi
ALB	Albanian

Animation content:

There is a table named Country Language with four columns named Country Code Language Is Official and Percentage. Slide 2. Four lines of code appear below the table. Line one states SELECT

Country Code comma Language. Line two states FROM Country Language. Line three states WHERE Percentage is greater than zero point zero. Line four states AND Percentage is less than ten point zero. Percentage is greater than zero point zero and percentage is less than ten point zero are both compared with every value in column Percentage in table Country Language. The end result is true true false false. A new table named Results appears below the code with two columns named Country Code and Language and the values are taken from the true rows. Slide 3. Four lines of code appear below the table. Line one states SELECT Country Code comma Language. Line two states FROM Country Language. Line three states WHERE Percentage is less than five point zero. Line four states OR Percentage is greater than ninety point zero.Percentage is less than five point zero and percentage is greater than ninety point zero is compared with the Percentage column in table Country Language and results in False True False True. A new table named Result appears below the code with two columns named Country Code and Language and the values and the values are taken from the true rows.

Animation captions:

1. The statement selects rows with percentage between 0.0 and 10.0. Two rows are returned.
2. The statement selects rows with percentage < 5.0 or percentage > 90.0. Two rows are returned.

PARTICIPATION ACTIVITY 4.2.8: WHERE clause.

Refer to the City table.

City

ID	Name	CountryCode	District	Population
207	Rio de Janeiro	BRA	Rio de Janeiro	5598953
462	Manchester	GBR	England	430000
554	Santiago de Chile	CHL	Santiago	4703954
654	Barcelona	ESP	Catalonia	1503451
826	Valencia	PHL	Northern Mindanao	147924
1025	Delhi	IND	Delhi	7206704

1) What cities are selected?

```
SELECT *
FROM City
WHERE Population < 150000;
```

- All cities
- Valencia and Manchester
- Valencia

2) What districts are selected?

```
SELECT District
FROM City
WHERE Name = 'Rio de Janeiro';
```


- No districts
- Rio de Janeiro
- Rio de Janeiro and Catalonia

3) What cities are selected?

```
SELECT Name
FROM City
WHERE CountryCode != 'CHL';
```

- No cities
- All cities
- All cities except Santiago de Chile

4) What names are selected?

```
SELECT Name
FROM City
WHERE NOT ID < 2000;
```

- No names
- All names
- Valencia

5) What cities are selected?

```
SELECT *
FROM City
WHERE ID <= 1000
  AND ID >= 600;
```

- All cities
- Barcelona and Valencia
- Delhi

6) What cities are selected?

```
SELECT *
FROM city
WHERE ID < 300
   OR Population > 7500000;
```

- All cities
- Delhi and Rio de Janeiro
- Rio de Janeiro

7) What cities are selected?

```
SELECT *
FROM City
WHERE (Population >= 7000000 AND
Population <= 8000000)
   OR ID < 500;
```


- All cities
- Rio de Janeiro and Manchester
- Rio de Janeiro, Manchester, and Delhi

PARTICIPATION ACTIVITY 4.2.9: Select movies with logical operators.

The given SQL creates a Movie table and inserts some movies. The SELECT statement selects all movies released before January 1, 2000.

Modify the SELECT statement to select the title and release date of PG-13 movies that are released after January 1, 2008.

Run your solution and verify the result table shows just the titles and release dates for *The Dark Knight* and *Crazy Rich Asians*.

```
CREATE TABLE Movie (
  ID INT AUTO_INCREMENT,
  Title VARCHAR(100),
  Rating CHAR(5) CHECK (Rating IN ('G', 'PG', 'PG-13', 'R')),
  ReleaseDate DATE,
  PRIMARY KEY (ID)
);

INSERT INTO Movie (Title, Rating, ReleaseDate) VALUES
  ('Casablanca', 'PG', '1943-01-23'),
  ('Bridget Jones\'s Diary', 'PG-13', '2001-04-13'),
  ('The Dark Knight', 'PG-13', '2008-07-18'),
  ('Hidden Figures', 'PG', '2017-01-06'),
  ('Toy Story', 'G', '1995-11-22'),
  ('Rocky', 'PG', '1976-11-21'),
  ('Crazy Rich Asians', 'PG-13', '2018-08-15');

-- Modify the SELECT statement:
SELECT *
FROM Movie
WHERE ReleaseDate < '2000-01-01';
```


Run Reset code

▶ View solution

CHALLENGE ACTIVITY 4.2.1: Selecting rows.

537150.4174434.qx3zqy7

Start

Country

IndepYear	Continent	ISOCode2
1811	SAmerica	VE
1919	Asia	KR
1975	Africa	AO

What columns are returned by the given statements?

```
SELECT Continent
FROM Country;
```

- ☐ IndepYear
- ☐ Continent
- ☐ ISOCode2

```
SELECT *
FROM Country;
```

- ☐ IndepYear
- ☐ Continent
- ☐ ISOCode2

1 2 3 4 5

Check Next

4.3 Special operators and clauses

Learning goals

- Use IN and LIKE operators in queries.
- Use DISTINCT and ORDER BY clauses in queries.
- Interpret limiting clauses in MySQL, SQL Server, and Oracle Database.

IN operator

The ***IN*** operator is used in a WHERE clause to determine if a value matches one of several values. The SELECT statement in the figure below uses the IN operator to select only rows where the Language column has a Dutch, Kongo, or Albanian value.

Figure 4.3.1: Using the IN operator.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
ALB	Albanian	T	97.9
AND	Catalan	T	32.3

```
SELECT *
FROM CountryLanguage
WHERE Language IN ('Dutch', 'Kongo', 'Albanian');
```

```
ABW   Dutch      T   5.3
AGO   Kongo      F   13.2
ALB   Albanian   T   97.9
```

PARTICIPATION ACTIVITY 4.3.1: IN operator.

Refer to the Country table below.

Country

Code	Name	Continent
ABW	Aruba	North America
AFG	Afghanistan	Asia
AGO	Angola	Africa
ALB	Albania	Europe
AND	Andorra	Europe

1) What is returned by the query?

```
SELECT Name
FROM Country
WHERE Continent IN ('Asia',
'Europe', 'South America');
```

- No results
- Afghanistan, Albania, Aruba
- Afghanistan, Albania, Andorra

2) What is returned by the query?

```
SELECT Name
FROM Country
WHERE Code IN ('AGO', 'Aruba',
'Europe', NULL);
```

- No results
- Angola
- Aruba

3) What is returned by the query?

```
SELECT Name
FROM Country
WHERE Continent NOT IN ('Asia',
'Antarctica', 'Europe');
```

- No results
- Afghanistan, Albania, Andorra
- Aruba, Angola

LIKE operator

The ***LIKE*** operator, when used in a WHERE clause, matches text against a pattern using the two wildcard characters `%` and _.

- `%` matches any number of characters. Ex: `LIKE 'L%t'` matches "Lt", "Lot", "Lift", and "Lol cat".
- _ matches exactly one character. Ex: `LIKE 'L_t'` matches "Lot" and "Lit" but not "Lt" and "Loot".

The LIKE operator performs case-insensitive pattern matching by default or case-sensitive pattern matching if followed by the ***BINARY*** keyword. Ex: `LIKE BINARY 'L%t'` matches 'Left' but not 'left'.

To search for the wildcard characters % or _, a backslash (\) must precede % or _. Ex: `LIKE 'a\%'` matches "a%".

PARTICIPATION ACTIVITY 4.3.2: Using the LIKE operator.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
ALB	Albanian	T	97.9
ARW	Arawak	F	4.9


```
SELECT *
FROM CountryLanguage
WHERE CountryCode LIKE 'A_W';
```

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
ARW	Arawak	F	4.9

```
SELECT *
FROM CountryLanguage
WHERE Language LIKE 'A%n';
```

CountryCode	Language	IsOfficial	Percentage
ALB	Albanian	T	97.9

```
SELECT *
FROM CountryLanguage
WHERE Language LIKE 'A%';
```

CountryCode	Language	IsOfficial	Percentage
ALB	Albanian	T	97.9
ARW	Arawak	F	4.9

Animation content:

Slide 1. There is a table named Country Language with four columns named Country Code Language Is Official and Percentage. Three lines of code appear below the table. The first line of code states SELECT asterisk. The second line of code states FROM Country Language. The third line of code states WHERE Country Code LIKE apostrophe A underscore W apostrophe semicolon. The condition Country Code LIKE apostrophe A underscore W apostrophe is compared to each row of column Country Code of table Country Language. The first row has the value ABW and satisfies the condition so row one is true. The second row has the value AFG and does not satisfy the condition so row two is false. The third row has the value AGO and does not satisfy the condition so row three is false. The fourth row has the value ALB and does not satisfy the condition so row four is false. The fifth row has the value ARW and satisfies the condition so row five is true. The two true rows are added to an unnamed table with four columns named Country Code Language Is Official and Percentage. Slide 2. Three lines of code appear below the previous code. The first line of code states SELECT asterisk. The second line of code states FROM Country Language. The third line of code states WHERE Language LIKE apostrophe A percent a apostrophe semicolon. The condition WHERE Language LIKE apostrophe A percent n apostrophe semicolon is compared to each row of column Language of table Country Language. The first row has the value Dutch and does not satisfy the condition so row one is false. The second row has the value Balochi and does not satisfy the condition so row two is false. The third row has the value Kongo and does not satisfy the condition so row three is false. The fourth row has the value Albanian and satisfies the condition so row four is true. The fifth row has the value Arawak and does not satisfy the condition so row five is false. The true row is added to an unnamed table with four columns named Country Code Language Is Official and Percentage. Slide 3. Three lines of code appear below the previous code. The first line of code states SELECT asterisk. The second line of code states FROM Country Language. The third line of code states WHERE Language LIKE apostrophe K percent apostrophe semicolon. WHERE Language LIKE apostrophe A percent apostrophe semicolon is compared to each row in column Language of table Country Language. The first row has the value Dutch and does not satisfy the condition

Language LIKE apostrophe A percent apostrophe semicolon, so row one is false. The second row has the value Balochi and does not satisfy the condition Language LIKE apostrophe A percent apostrophe semicolon, so row two is false. The third row has the value Kongo and does not satisfy the condition Language LIKE apostrophe A percent apostrophe semicolon, so row three is false. The fourth row has the value Albania and does satisfy the condition Language LIKE apostrophe A percent apostrophe semicolon, so row four is true. The fifth row has the value Arawak and satisfies the condition Language LIKE apostrophe A percent apostrophe semicolon, so row five is true.

Animation captions:

1. Using the _ wildcard character with the LIKE operator matches exactly one character.
2. Using the % wildcard character with the LIKE operator matches any number of characters in the middle of a string.
3. Using the % wildcard character with the LIKE operator matches any number of characters at the beginning or end of a string.

Regular expressions

Most relational databases provide other mechanisms to perform more advanced pattern matching with regular expressions. Ex: MySQL uses REGEXP to match text against a regular expression, and PostgreSQL uses SIMILAR TO.

PARTICIPATION ACTIVITY 4.3.3: LIKE operator.

Match the LIKE statement with the matching Language.

```
SELECT Language
FROM CountryLanguage
WHERE Language _______;
```

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
COK	English	F	0.0
COK	Maori	T	0.0
COL	Arawakan	F	0.1
COL	Caribbean	F	0.1
COL	Chibcha	F	0.4

If unable to drag and drop, refresh the page.

LIKE '%m_o%' | LIKE '_cha' | LIKE BINARY '%E%' | LIKE '%r%n' | LIKE '%cha'

Arawakan and Caribbean

	Chibcha
	No matches
	Maori
	English

Reset

PARTICIPATION ACTIVITY 4.3.4: Select movie titles with LIKE.

The given SQL creates a Movie table and inserts some movies. The SELECT statement selects all movies.

Modify the SELECT statement to select movies with the word "star" somewhere in the title.

Run your solution and verify the result table shows just the movies *Rogue One: A Star Wars Story*, *Star Trek* and *Stargate*.

```
1  CREATE TABLE Movie (
2    ID INT AUTO_INCREMENT,
3    Title VARCHAR(100),
4    Rating CHAR(5) CHECK (Rating IN ('G', 'PG', 'PG-13', 'R')),
5    ReleaseDate DATE,
6    PRIMARY KEY (ID)
7  );
8
9  INSERT INTO Movie (Title, Rating, ReleaseDate) VALUES
10   ('Rogue One: A Star Wars Story', 'PG-13', '2016-12-16'),
11   ('Star Trek', 'PG-13', '2009-05-08'),
12   ('The Dark Knight', 'PG-13', '2008-07-18'),
13   ('Stargate', 'PG-13', '1994-10-28'),
14   ('Avengers: Endgame', 'PG-13', '2019-04-26');
15
16 -- Modify the SELECT statement:
17 SELECT *
18 FROM Movie;
19
```

Run Reset code

▶ View solution

DISTINCT clause

The ***DISTINCT*** clause is used with a SELECT statement to return only unique values. Ex: The first SELECT statement in the figure below results in two 'Spanish' rows, but the second SELECT statement returns only unique languages, resulting in only one 'Spanish' row.

Figure 4.3.2: SELECT DISTINCT example.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage

CountryCode	Language	IsOfficial	Percentage
ABW	Spanish	F	7.4
AFG	Balochi	F	0.9
ARG	Spanish	T	96.8
BLZ	Spanish	F	31.6
BRA	Portuguese	T	97.5

```
SELECT Language
FROM CountryLanguage
WHERE IsOfficial = 'F';
```

```
Spanish
Balochi
Spanish
```

```
SELECT DISTINCT Language
FROM CountryLanguage
WHERE IsOfficial = 'F';
```

```
Spanish
Balochi
```


PARTICIPATION ACTIVITY 4.3.5: DISTINCT clause.

Match the query with the result set.

City

ID	Name	CountryCode	District	Population
69	Buenos Aires	ARG	Distrito Federal	2982146
310	Taboão da Serra	BRA	São Paulo	197550
1888	Nyeri	KEN	Central	91258
2732	Lalitapur	NPL	Central	145847
2733	Birgunj	NPL	Central	90639
3539	Caracas	VEN	Distrito Federal	1975294

If unable to drag and drop, refresh the page.

```
SELECT District
FROM City;
```

```
SELECT DISTINCT CountryCode, District
FROM City;
```

```
SELECT DISTINCT District
FROM City;
```

```
SELECT CountryCode, District
FROM City;
```

```
Distrito Federal
São Paulo
Central
Central
Central
Distrito Federal
```

```
Distrito Federal
São Paulo
Central
```

```
ARG  Distrito Federal
BRA  São Paulo
KEN  Central
NPL  Central
NPL  Central
VEN  Distrito Federal
```

```
ARG  Distrito Federal
BRA  São Paulo
KEN  Central
NPL  Central
VEN  Distrito Federal
```


Reset

ORDER BY clause

A SELECT statement selects rows from a table with no guarantee the data will come back in a certain order. The ***ORDER BY*** clause orders selected rows by one or more columns in ascending (alphabetic or increasing) order. The ***DESC*** keyword with the ORDER BY clause orders rows in descending order.

Figure 4.3.3: Ordering by columns.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
FSM	Woleai	F	3.7
FSM	Yap	F	5.8
GAB	Fang	F	35.8
GAB	Mbete	F	13.8

```
-- Order by Language (ascending)
SELECT *
FROM CountryLanguage
ORDER BY Language;
```

```
GAB   Fang    F   35.8
GAB   Mbete   F   13.8
FSM   Woleai  F    3.7
FSM   Yap     F    5.8
```

```
-- Order by Language (descending)
SELECT *
FROM CountryLanguage
ORDER BY Language DESC;
```

```
FSM   Yap     F    5.8
FSM   Woleai  F    3.7
GAB   Mbete   F   13.8
GAB   Fang    F   35.8
```

```
-- Order by CountryCode, then Language
SELECT *
FROM CountryLanguage
ORDER BY CountryCode, Language;
```

```
FSM   Woleai  F    3.7
FSM   Yap     F    5.8
GAB   Fang    F   35.8
GAB   Mbete   F   13.8
```


PARTICIPATION ACTIVITY 4.3.6: ORDER BY clause.

Choose the correct ORDER BY clause to produce the results in each question.

```
SELECT Name, District, Population
FROM City
_______;
```

City

ID	Name	CountryCode	District	Population
301	Embu	BRA	São Paulo	222223
302	Mossoró	BRA	Rio Grande do Norte	214901
303	Várzea Grande	BRA	Mato Grosso	214435
304	Petrolina	BRA	Pernambuco	210540
305	Barueri	BRA	São Paulo	208426

1)

```
Barueri         São Paulo
208426
Embu            São Paulo
222223
Mossoró         Rio Grande do Norte
214901
Petrolina       Pernambuco
210540
Várzea Grande   Mato Grosso
214435
```

- ORDER BY Name
- ORDER BY CountryCode
- ORDER BY District

2)

```
Barueri         São Paulo
208426
Petrolina       Pernambuco
210540
Várzea Grande   Mato Grosso
214435
Mossoró         Rio Grande do Norte
214901
Embu            São Paulo
222223
```

- ORDER BY Name
- ORDER BY Population
- ORDER BY District

3)

```
Embu            São Paulo
222223
Barueri         São Paulo
208426
Mossoró         Rio Grande do Norte
214901
Petrolina       Pernambuco
210540
Várzea Grande   Mato Grosso
214435
```

- ORDER BY Name DESC
- ORDER BY District
- ORDER BY District DESC

4)

```
Várzea Grande   Mato Grosso
214435
Petrolina       Pernambuco
210540
Mossoró         Rio Grande do Norte
214901
Barueri         São Paulo
208426
Embu            São Paulo
222223
```

- ORDER BY Name, Population
- ORDER BY Population, District
- ORDER BY District, Population

Limiting clauses

Queries against very large tables might return thousands or millions of rows. Databases support special clauses that limit the number of rows returned by a query. The syntax of limiting clauses is not standardized.

MySQL supports a ***LIMIT*** clause, consisting of the LIMIT keyword and number of rows. The LIMIT clause appears at the end of the SELECT statement.

Figure 4.3.4: MySQL example.

```
SELECT *
FROM
CountryLanguage
LIMIT 20;
```

SQL Server supports a ***TOP*** clause, consisting of the TOP keyword and number of rows. The TOP clause follows the SELECT keyword.

Figure 4.3.5: SQL Server example.

```
SELECT TOP 20 *
FROM
CountryLanguage;
```

Oracle Database supports a ***FETCH*** clause, with syntax `FETCH FIRST number ROWS ONLY`. The FETCH clause appears at the end of the SELECT statement.

Figure 4.3.6: Oracle Database example.

```
SELECT *
FROM CountryLanguage
FETCH FIRST 20 ROWS
ONLY;
```

When a limit clause is used without an ORDER BY clause, the rows that appear are unpredictable. When a limit clause is used with ORDER BY, the rows that appear are based on row order and predictable. For this reason, LIMIT, TOP, and FETCH clauses are usually used with an ORDER BY clause.

PARTICIPATION ACTIVITY 4.3.7: Limiting clauses.

The City table has 200,000 rows with columns ID, Name, CountryCode, and Population.

1) The following MySQL query returns the top 10 cities by population. What replaces XXX?

```
SELECT *
FROM City
ORDER BY Population
XXX;
```

Check Show answer

2) The following SQL Server query returns the first 100 city names in alphabetic order. What replaces XXX?

```
SELECT XXX Name
FROM City
ORDER BY Name;
```

Check Show answer

3) The following Oracle Database query returns 35 cities with district and population. What replaces XXX?

```
SELECT Name, District,
Population
FROM City
XXX ROWS ONLY;
```

Check Show answer

CHALLENGE ACTIVITY 4.3.1: Special operators and clauses.

537150.4174434.qx3zqy7

Start

Country

ISOCode2	Name	Capital	Continent
LA	Laos	Vientiane	Asia
ER	Eritrea	Asmara	Africa
PA	Panama	Panama City	NAmerica


```
SELECT Name
FROM Country
WHERE Continent IN ('Africa', NULL, 'Oceania');
```

Select the Name values in the row(s) returned by the above statement.

- [ ] Laos
- [ ] Eritrea
- [ ] Panama

1 2 3 4

Check Next

4.4 Aggregate functions

Learning goals

- List common arithmetic, string, date, and time functions.
- Differentiate simple and aggregate functions.
- Use GROUP BY and HAVING clauses in queries.

Simple functions

A ***simple function*** operates on an expression enclosed in parentheses, called an ***argument***, and returns a value. The expression is usually a column name or fixed value, but may include operators and other functions. Some functions have several arguments, separated by commas, and a few have no arguments.

Each simple function operates on, and evaluates to, specific data types. Ex: The LOG function operates on any numeric data type and returns a DOUBLE value. If the argument is invalid, the function returns NULL. Ex: The SQRT function computes square root of positive numbers only, so SQRT(-1) returns NULL.

Most simple functions operate on numeric, string, or date and time values. Common simple functions in SQL appear below.

Table 4.4.1: Simple functions.

Type	Function	Description	Example
Numeric	`ABS(n)`	Absolute value of *n*	`SELECT ABS(-5);`
	`LOG(n)`	Natural logarithm of *n*	`SELECT LOG(10);`
	`POW(x, y)`	*x* to the power of *y*	`SELECT POW(2, 3);`
	`RAND()`	Random number between 0 (inclusive) and 1 (exclusive)	`SELECT RAND();`
	`ROUND(n, d)`	*n* rounded to *d* decimal places	`SELECT ROUND(16.251);`
	`SQRT(n)`	Square root of *n*	`SELECT SQRT(25);`
String	`CONCAT(s1, s2, ...)`	Concatenation of the strings *s1*, *s2*, ...	`SELECT CONCAT('Dis', 'en' 'gage');`
	`LOWER(s)`	*s* converted to lower case	`SELECT LOWER('MySQL');`
	`UPPER(s)`	*s* converted to upper case	`SELECT UPPER('mysql');`
	`REPLACE(s, from, to)`	*s* with all occurrences of *from* replaced by *to*	`SELECT REPLACE('Orange', 'O', 'St');`
	`SUBSTRING(s, pos, len)`	Substring of *s* that starts at position *pos* with length *len*	`SELECT SUBSTRING('Boomerang', 1, 4);`

Date Time	`CURDATE()`
	`CURTIME()`
	`NOW()`
		Current date, time, or date and time in `'YYYY-MM-DD'`, `'HH:MM:SS'`, or `'YYYY-MM-DD HH:MM:SS'` format	`SELECT CURDATE();`

	`DAY(d)`
	`MONTH(d)`
	`YEAR(d)`
		Day, month, or year of *d*	`SELECT MONTH('2016-10-25');`

	`HOUR(t)`
	`MINUTE(t)`
	`SECOND(t)`
		Hour, minute, or second of *t*	`SELECT MINUTE('22:11:45')`

	`DATEDIFF(dt1, dt2)` `TIMEDIFF(dt1, dt2)`	Difference of *dt1 − dt2*, in number of days or amount of time	`SELECT DATEDIFF('2013-03-10', '2013-03-04')`

PARTICIPATION ACTIVITY 4.4.1: String functions.

Refer to the table below.

Avatar

ID	Name	BestMove
1	Link	Triforce Slash
2	Meta Knight	Galaxia Darkness
3	Mewtwo	Psystrike
4	Mario	Mario Finale

Enter the result of the following queries.

1)
```
SELECT CONCAT('Super ', Name)
FROM Avatar
WHERE ID = 1;
```

Check Show answer

2)
```
SELECT LOWER(BestMove)
FROM Avatar
WHERE ID = 3;
```

Check Show answer

3)
```
SELECT SUBSTRING(BestMove, 7, 6)
FROM Avatar
WHERE ID = 4;
```

Check Show answer

4)
```
SELECT REPLACE(Name, 'Kn',
'Fr')
FROM Avatar
WHERE ID = 2;
```

Check Show answer

PARTICIPATION ACTIVITY 4.4.2: Date and time functions.

The table below stores assigned and due datetimes for homework. Match the datetime value to the query.

Assignment

ID	Assigned	Due
1	2019-11-01 08:00:00	2019-11-02 08:00:00
2	2019-11-02 12:30:00	2019-11-02 23:59:00
3	2019-11-05 10:15:00	2019-11-05 11:15:00
4	2019-11-07 08:00:00	2019-11-14 08:00:00

If unable to drag and drop, refresh the page.

14 01:00:00 1 23:59:00 42

```
SELECT TIME(Due)
FROM Assignment
WHERE ID = 2;
```

```
SELECT DAY(Due)
FROM Assignment
WHERE ID = 4;
```

```
SELECT HOUR(Assigned) +
MINUTE(Assigned)
FROM Assignment
WHERE ID = 2;
```

```
SELECT DATEDIFF(Due, Assigned)
FROM Assignment WHERE
ID = 1;
```

```
SELECT TIMEDIFF(Due, Assigned)
FROM Assignment
WHERE ID = 3;
```


Reset

Aggregate functions

An ***aggregate function*** processes values from a set of rows and returns a summary value. Aggregate functions appear in a SELECT clause and process all rows that satisfy the WHERE clause condition. If a SELECT statement has no WHERE clause,

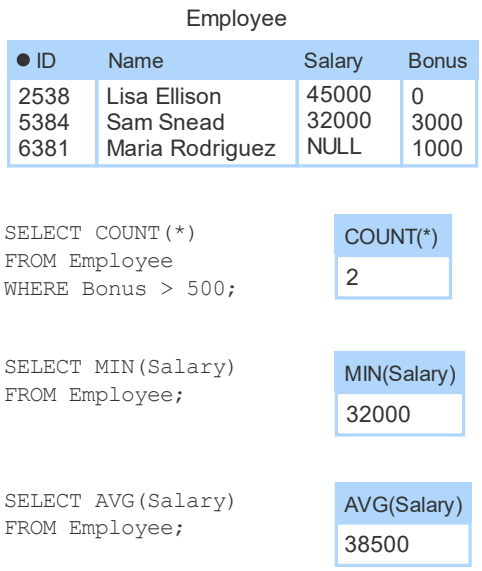
the aggregate function processes all rows.

Common aggregate functions are:

- ***COUNT()*** counts the number of selected values.
- ***MIN()*** finds the minimum of selected values.
- ***MAX()*** finds the maximum of selected values.
- ***SUM()*** sums selected values.
- ***AVG()*** computes the arithmetic mean of selected values.
- ***VARIANCE()*** computes the standard variance of selected values.

Aggregate functions have a single argument. The argument may be any expression but is usually a column name. Aggregate functions ignore rows for which the expression evaluates to NULL.

PARTICIPATION ACTIVITY 4.4.3: Using aggregate functions in a SELECT statement.



Animation content:

Slide 1. There is a table named Employee with four columns named ID Name Salary and Bonus. Three lines of code appear. The first line of code state SELECT COUNT left parenthesis asterisk right parenthesis. The second line of code states FROM Employee. The third line of code states WHERE Bonus is greater than 500 semicolon. COUNT left parenthesis asterisk right parenthesis is boxed and rows two and three of table Employee are highlighted. The values of rows two and three of column Bonus in table Employee are 3000 and 1000 respectively. A new table appears and has one column named COUNT left parenthesis asterisk right parenthesis. The value 2 is added to this column. Slide 2. Two new lines of code appear. The first line of code states SELECT MIN left parenthesis Salary right parenthesis. The second line of code states FROM Employee semicolon. Row two of column Salary in table Employee is highlighted and contains the value 32000. A new table appears and has one column anmed MIN left parenthesis Salary right parenthesis. The value 32000 is added to this column. Slide 3. Two new lines of code appear. SELECT AVG left parenthesis Salary right parenthesis. The second line of code states FROM Employee semicolon. Two non-null values in column Salary are highlighted and the value 38500 appears below the table. A new table appears and has one column named AVG left parenthesis Salary right parenthesis. The value 38500 is added to this column.

Animation captions:

1. COUNT(*) counts the number of selected values. Two employees have Bonus > 500.
2. MIN(Salary) finds the smallest salary. The NULL is ignored.
3. AVG(Salary) finds the average of non-NULL salaries, or (45000 + 32000)/2 = 38500.

PARTICIPATION ACTIVITY 4.4.4: Aggregate functions.

Choose the correct SELECT statement that returns the given results from the Auto table below.

Auto

●ID	Make	Model	Type	Year	Price
1	Toyota	Camry	sedan	2015	9800
2	Ford	Escape	crossover	2015	15900
3	Honda	Civic	sedan	2016	10200
4	Volkswagen	Golf	compact	2014	8800
5	Toyota	RAV4	crossover	2016	12800
6	Toyota	4Runner	SUV	2015	16900
7	Honda	CR-V	crossover	2016	17900

1) `2014`

- ```
  SELECT MAX(Year)
  FROM Auto;
  ```
- ```
  SELECT MIN(Price)
  FROM Auto;
  ```
- ```
  SELECT MIN(Year)
  FROM Auto;
  ```

2) `92300`

- ```
  SELECT SUM(Price)
  FROM Auto;
  ```
- ```
  SELECT AVG(Price)
  FROM Auto;
  ```
- ```
  SELECT MAX(Price)
  FROM Auto;
  ```

3) `2`

- ```
  SELECT COUNT(*)
  FROM Auto;
  ```
- ```
  SELECT COUNT(*)
  FROM Auto
  WHERE Price > 10000;
  ```
- ```
  SELECT COUNT(*)
  FROM Auto
  WHERE Price < 10000;
  ```


GROUP BY clause
  ```

Aggregate functions are commonly used with the GROUP BY clause. The ***GROUP BY*** clause groups rows with identical values into a set of summary rows. Some important points about the GROUP BY clause:

- One or more columns are listed after GROUP BY, separated by commas.
- GROUP BY clause returns one row for each group.
- Each group may be ordered with the ORDER BY clause.
- GROUP BY clause must appear before the ORDER BY clause and after the WHERE clause (if present).

PARTICIPATION ACTIVITY 4.4.5: GROUP BY clause.

City

ID	Name	CountryCode	District	Population
3162	Lusaka	ZMB	1	1317000
3163	Ndola	ZMB	2	329200
3164	Kitwe	ZMB	2	288600
3165	Kabwe	ZMB	3	154300
3166	Chingola	ZMB	2	142400
4068	Harare	ZWE	1	1410000
4069	Bulawayo	ZWE	2	621742
4070	Chitungwiza	ZWE	1	274912

```
SELECT CountryCode, SUM(Population)
FROM City
GROUP BY CountryCode;
```

CountryCode	SUM(Population)
ZMB	2231500
ZWE	2306654

```
SELECT CountryCode, District, COUNT(*)
FROM City
GROUP BY CountryCode, District;
```

CountryCode	District	COUNT(*)
ZMB	1	1
ZMB	2	3
ZMB	3	1
ZWE	1	2
ZWE	2	1

Animation content:

Slide 1. There is a table named City with five features. ID Name Country Code District and Population. The S Q L code states. Select country code comma sum of population. From city. Group by country code semicolon. The sum of population is boxed and the population feature in table city is highlighted. Slide 2. The group by statement is boxed and the country code feature in table city is highlighted. Slide 3. The first five rows of features country code are highlighted and contain the value ZMB. The values in feature population of the five highlighted rows are summed together to 2231500. The remaining three rows of feature country code are highlighted and contain the value ZWE. The values in feature population of the three highlighted rows are summed together to 2306654. The return values are feature country code ZMB and ZWE and sum of population 2231500 and 2306654. Slide 4. New S Q L code appears and states. Select country code comma distract comma count of all. From city. Group by country code comma district semicolon. Count of all is boxed. Slide 5. The group by statement is boxed and features country code and district are highlighted. Slide 6. The first row of features country code and district is highlighted and contains the values ZMB and 1. The count is 1. The second third and fifth rows of features country code and district are highlighted and contain the values ZMB and 2. The count is 3. The fourth row of features country codes and district is highlighted and contains the values ZMB and 3. The count is 1. The sixth and eighth rows of features country code and district are highlighted and contain the values ZWE and 1. The count is 2. The seventh row of features country code and district are highlighted

and contains the values ZWE and 1. The count is 1. The return values are features country code AMB ZMB ZMB ZWE and ZWE district 1 2 3 1 and 2 and count of all 1 3 1 2 and 1.

Animation captions:

1. The SUM() function sums the Population values in each group.
2. The GROUP BY clause forms groups based on the CountryCode column.
3. CountryCode contains two unique values: ZMB and ZWE. So two rows are returned with the total population of each CountryCode value.
4. The COUNT() function counts how many rows exist in each group.
5. The groups are formed by the CountryCode and District columns.
6. Each unique CountryCode and District combination is counted.

PARTICIPATION ACTIVITY 4.4.6: GROUP BY clause.

Choose the SELECT statement that returns the given results from the Auto table below.

Auto

ID	Make	Model	Type	Year	Price
1	Toyota	Camry	sedan	2016	9800
2	Ford	Escape	crossover	2015	15900
3	Honda	Civic	sedan	2016	10200
4	Volkswagen	Golf	compact	2014	8800
5	Toyota	RAV4	crossover	2016	12800
6	Toyota	4Runner	SUV	2015	16900
7	Honda	CR-V	crossover	2016	17900

1)

```
Ford         1
Honda        2
Toyota       3
Volkswagen   1
```

```
SELECT Make, COUNT(*)
FROM Auto
ORDER BY Make;
```

```
SELECT Make, COUNT(*)
FROM Auto
GROUP BY Make
ORDER BY Make;
```

```
SELECT COUNT(*)
FROM Auto
GROUP BY Make
ORDER BY Make;
```

2)

```
2014   8800
2015   16400
2016   12675
```

-
```
SELECT Year, AVG(Price)
FROM Auto
GROUP BY Year
ORDER BY Year;
```

-
```
SELECT Year, AVG(Year)
FROM Auto
GROUP BY Year
ORDER BY Year;
```

-
```
SELECT Price, AVG(Price)
FROM Auto
GROUP BY Year
ORDER BY Year;
```

3)

```
compact   8800
sedan     10200
SUV       16900
crossover 17900
```

-
```
SELECT Type, MAX(Price)
FROM Auto
GROUP BY Type;
```

-
```
SELECT Type, MAX(Price)
FROM Auto
GROUP BY Type
ORDER BY Price;
```

-
```
SELECT Type, MAX(Price)
FROM Auto
GROUP BY Type
ORDER BY MAX(Price);
```

4)

```
2014  compact    8800
2015  crossover  15900
2015  SUV        16900
2016  sedan      10200
2016  crossover  17900
```

-
```
SELECT Year, MAX(Price)
FROM Auto
GROUP BY Year, Type
ORDER BY Year, MAX(Price);
```

-
```
SELECT Year, Type,
MAX(Price)
FROM Auto
GROUP BY Year
ORDER BY Year, MAX(Price);
```

-
```
SELECT Year, Type,
MAX(Price)
FROM Auto
GROUP BY Year, Type
ORDER BY Year, MAX(Price);
```


©zyBooks 03/21/24 21:41 2087217
Biniam abebe
UNTADTA5340OrhanSpring8wk22024

HAVING clause

The ***HAVING*** clause is used with the GROUP BY clause to filter group results. The optional HAVING clause follows the GROUP BY clause and precedes the optional ORDER BY clause.

The HAVING clause must include the same aggregate function that appears in the SELECT clause.

PARTICIPATION ACTIVITY 4.4.7: HAVING clause.

City

ID	Name	CountryCode	District	Population
3162	Lusaka	ZMB	1	1317000
3163	Ndola	ZMB	2	329200
3164	Kitwe	ZMB	2	288600
3165	Kabwe	ZMB	3	154300
3166	Chingola	ZMB	2	142400
4068	Harare	ZWE	1	1410000
4069	Bulawayo	ZWE	2	621742
4070	Chitungwiza	ZWE	1	274912


```
SELECT CountryCode, SUM(Population)
FROM City
GROUP BY CountryCode
HAVING SUM(Population) > 2300000;
```

CountryCode	SUM(Population)
ZWE	2306654

```
SELECT CountryCode, District, COUNT(*)
FROM City
GROUP BY CountryCode, District
HAVING COUNT(*) >= 2;
```

CountryCode	District	COUNT(*)
ZMB	2	3
ZWE	1	2

Animation content:

Slide 1. There is a table named city with five features. ID Name Country Code District and Population. The S Q L code states. Select country code comma sum of population. From city. Group by country code. Having sum of population greater than 2300000 semicolon. The group by and having statements are boxed. Slide 2. The first five rows in feature country code are highlighted and all contain the value ZMB. The values in feature population of the five highlighted rows are summed together to the value 2231500. The remaining three rows in feature country code are highlighted and all contain the value ZWE. The values in feature population of the three highlighted rows are summed together to the value 2306654. The returned values are country code ZWE and sum of population 2306654. Slide 3. New S Q L code appears and states. Select country code comma district comma count of all. From city. Group by country code comma district. Having count of all greater than or equal to two semicolon. The group by and having statements are boxed. The first row of features country code and district are highlighted and contain the values ZMB and 1. The count equals 1. The second third and fifth rows of features country code and district are highlighted and contain the values ZMB and 2. The count equals 3. The fourth row of feature country code and district are highlighted and contain the values ZMB and 3. The count equals 1. The sixth and eighth rows are highlighted and contain the values ZWE and 1. The count is 2. The seventh row is highlighted and contains the values ZWE and 2. The count is 1. The returned values are country code ZMB and ZWE district 2 and 1 and count of all 3 and 2.

Animation captions:

1. The HAVING clause follows the GROUP BY clause.
2. Although the GROUP BY clause creates two groups based on CountryCode, the HAVING clause selects only the group with a population sum > 2,300,000.
3. The HAVING clause selects only groups that have a row count >= 2. Only the ZMB, 2 and ZWE, 1 groups have at least 2 rows.

PARTICIPATION ACTIVITY 4.4.8: HAVING clause.

Choose the SELECT statement that returns the given results from the Auto table below.

Auto

ID	Make	Model	Type	Year	Price
1	Toyota	Camry	sedan	2016	9800
2	Ford	Escape	crossover	2015	15900
3	Honda	Civic	sedan	2016	10200
4	Volkswagen	Golf	compact	2014	8800
5	Toyota	RAV4	crossover	2016	12800
6	Toyota	4Runner	SUV	2015	16900
7	Honda	CR-V	crossover	2016	17900

1)

```
Honda   2
Toyota  3
```

```
SELECT Make, COUNT(Make)
FROM Auto
GROUP BY Make;
```

```
SELECT Make, COUNT(Make)
FROM Auto
GROUP BY Make
HAVING COUNT(Make) > 1;
```

```
SELECT Make, COUNT(Make)
FROM Auto
GROUP BY Make
HAVING COUNT > 1;;
```

2)

```
2015  crossover  15900
2015  SUV        16900
2016  crossover  17900
```

```
SELECT Year, Type,
MAX(Price)
FROM Auto
GROUP BY Year, Type
ORDER BY Year, MAX(Price);
```

```
SELECT Year, Type,
MAX(Price)
FROM Auto
GROUP BY Year, Type
ORDER BY Year, MAX(Price)
HAVING MAX(Price) > 15000;
```

```
SELECT Year, Type,
MAX(Price)
FROM Auto
GROUP BY Year, Type
HAVING MAX(Price) > 15000
ORDER BY Year, MAX(Price);
```


PARTICIPATION ACTIVITY 4.4.9: Find the most recent release year for each genre.

The given SQL creates a Song table and inserts songs. The SELECT statement selects the genre and row count for each genre group.

Add a new column to the SELECT statement that uses MAX() to find the most recent release year for each genre. Then add a HAVING clause that selects only genre groups that have more than one row count.

Run your solution and verify that country pop, R&B, and grunge genres, row counts, and the most recent release years appear in the result table.

```
CREATE TABLE Song (
  ID INT,
  Title VARCHAR(60),
  Artist VARCHAR(60),
  ReleaseYear INT,
  Genre VARCHAR(20),
  PRIMARY KEY (ID)
);

INSERT INTO Song VALUES
  (100, 'Hey Jude', 'Beatles', 1968, 'pop rock'),
  (200, 'You Belong With Me', 'Taylor Swift', 2008, 'country pop'),
  (300, 'You\'re Still the One', 'Shania Twain', 1998, 'country pop'),
  (400, 'Need You Now', 'Lady Antebellum', 2009, 'country pop'),
  (500, 'You\'ve Lost That Lovin\' Feeling', 'The Righteous Brothers', 1964,
  (600, 'That\'s The Way Love Goes', 'Janet Jackson', 1993, 'R&B'),
  (700, 'Smells Like Teen Spirit', 'Nirvana', 1991, 'grunge'),
  (800, 'Even Flow', 'Pearl Jam', 1992, 'grunge'),
  (900, 'Black Hole Sun', 'Soundgarden', 1994, 'grunge');

-- Modify the SELECT statement
SELECT Genre, COUNT(*)
FROM Song
GROUP BY Genre;
```

Run Reset code

▶ View solution

CHALLENGE ACTIVITY 4.4.1: Aggregate functions.

537150.4174434.qx3zqy7

Start

Country

ID	Name	Capital	PopulationMillions
380	Italy	Rome	60.42
132	Cape Verde	Praia	0.54
438	Liechtenstein	Vaduz	0.04
44	Bahamas	Nassau	0.39

Complete the SELECT statement that finds the average PopulationMillions.

```
SELECT ____(A)____(____(B)____)
FROM Country;
```

(A) /* Type your code here */

(B)

1 2 3 4 5

Check Next

4.5 Join queries

Learning goals

- Explain join queries on left and right tables.
- Use aliases and prefixes.
- Construct inner, full, left, and right joins.
- Construct equijoins and non-equijoins.

Joins

In relational databases, reports are generated from data in multiple tables. Multi-table reports are written with join statements.

A ***join*** is a SELECT statement that combines data from two tables, known as the ***left table*** and ***right table***, into a single result. The tables are combined by comparing columns from the left and right tables, usually with the = operator. The columns must have comparable data types.

Usually, a join compares a foreign key of one table to the primary key of another. However, a join can compare columns with comparable data types.

PARTICIPATION ACTIVITY 4.5.1: Example join.

Department

● Code	DepartmentName	○ Manager
44	Engineering	2538
82	Sales	6381
12	Marketing	6381
99	Technical support	NULL

Employee

● ID	EmployeeName	Salary
2538	Lisa Ellison	45000
5384	Sam Snead	30500
6381	Maria Rodriguez	92300

```
SELECT DepartmentName, EmployeeName
FROM Department, Employee
WHERE Manager = ID;
```

Result

DepartmentName	EmployeeName
Engineering	Lisa Ellison
Sales	Maria Rodriguez
Marketing	Maria Rodriguez

Animation content:

Static figure: An example of a join. The following tables Department and Employee are joined to create the Result table.

Department

Code	Department Name	Manager
44	Engineering	2538
82	Sales	6381
12	Marketing	6381
99	Technical support	NULL

Employee

ID	Employee Name	Salary
2538	Lisa Ellison	4500
5384	Sam Snead	30500
6381	Maria Rodriguez	92300

Result

Department Name	Employee Name
Engineering	Lisa Ellison
Sales	Maria Rodriguez
Marketing	Maria Rodriguez

The query is 'SELECT DepartmentName, EmployeeName
FROM Department, Employee
WHERE Manager = ID;'

Step 1: The join query displays department names and managers.
The Department table and the Employee table appear. The first line of the query, 'SELECT DepartmentName, EmployeeName', is highlighted.
Step 2: Department is the left table. Employee is the right table.

The second line of the query, 'FROM Department, Employee', is highlighted.
Step 3: The query selects rows for which the foreign key Manager equals the primary key ID.
The third line of the query, 'WHERE Manager = ID;' is highlighted. An arrow points from the Manager column in the Manager table to the ID column in the Employee table.
Step 4: Lisa Ellison manages the Engineering department. Maria Rodriguez manages the Sales and Marketing departments.
The Result table appears with only the headers filled in. The first row of the Employee table where ID is 2538 and Employee Name is Lisa Ellison is highlighted. The Department table is scanned and the row where Manager is 2538 and Department Name is Engineering is also highlighted. The first row of the Result table is filled in with Engineering for the Department Name and Lisa Ellison for the Employee Name. The third row of the Employee table where ID is 6381 and Employee Name is Maria Rodriguez is highlighted. The Department table is scanned and Manager is equal to the id 6381 in two rows. Two more rows in the Result table are filled in with the corresponding Department Names: Sales and Marketing. The Employee Name in both rows is Maria Rodriguez.
Step 5: Sam Snead's ID does not appear in the Manager column, so Sam Snead does not appear in the result.
The third row of the Employee table where ID is 5384 and Employee Name is Sam Snead is highlighted. 5384 does not appear in the Manager column of the Department table. No rows are added to the Result table.
Step 6: Technical support has a NULL manager, so Technical support does not appear in the result.
The last row of the Department table where Manager is NULL and the Department Name is Technical support is highlighted. No row is added to the Result table.

Animation captions:

1. The join query displays department names and managers.
2. Department is the left table. Employee is the right table.
3. The query selects rows for which the foreign key Manager equals the primary key ID.
4. Lisa Ellison manages the Engineering department. Maria Rodriguez manages the Sales and Marketing departments.
5. Sam Snead's ID does not appear in the Manager column, so Sam Snead does not appear in the result.
6. Technical support has a NULL manager, so Technical support does not appear in the result.

PARTICIPATION ACTIVITY 4.5.2: Joins.

1) Which columns can be compared in a join?

- Only primary and foreign key columns.
- Only columns with comparable data types.
- Any columns.

2) In a join, what are the first and second tables in the FROM clause called?

- Left and right tables, respectively.
- Right and left tables, respectively.
- Table one and table two, respectively.

3) What is missing from the join statement?

```
SELECT EmployeeName, Salary
FROM Employee, Department
WHERE Salary > 50000;
```

- Left and right tables are not specified.
- No columns from Department appear in the SELECT clause.
- A column from Employee is not compared to a column from Department.

Prefixes and aliases

Occasionally, join tables contain columns with the same name. When duplicate column names appear in a query, the names must be distinguished with a prefix. The prefix is the table name followed by a period.

Use of a prefix makes column names more complex. To simplify queries or result tables, a column name can be replaced with an alias. The alias follows the column name, separated by an optional ***AS*** keyword.

Ex: In the figure below, the Name column appears in both tables and thus must have a prefix in the join query. Department.Name and Employee.Name are assigned aliases Group and Supervisor, which simplify the result column names.

Figure 4.5.1: Prefixes and aliases.

Department

● Code	Name	○ Manager
44	Engineering	2538
82	Sales	6381
12	Marketing	6381
99	Technical support	NULL

Employee

● ID	Name	Salary
2538	Lisa Ellison	45000
5284	Sam Snead	30500
6381	Maria Rodriguez	92300


```
SELECT Department.Name AS Group,
       Employee.Name AS Supervisor
FROM Department, Employee
WHERE Manager = ID;
```

Result

Group	Supervisor

Group	Supervisor
Engineering	Lisa Ellison
Sales	Maria Rodriguez
Marketing	Maria Rodriguez

PARTICIPATION ACTIVITY 4.5.3: Prefixes and aliases.

Refer to the Country and City tables below.

Country

•Code	Name	Language	IsOfficial	Percentage
ABW	Aruba	Dutch	T	5.3
ABW	Aruba	Papeiamento	F	76.7
AFG	Afghanistan	Balochi	F	0.9
AGO	Angola	Kongo	F	13.2
AGO	Angola	Mbundu	F	21.6

City

•ID	Name	○Code	Population
1	Kabul	AFG	1780000
2	Qandahar	AFG	237500
56	Luanda	AGO	2022000
57	Huambo	AGO	163100
129	Oranjestad	ABW	29034

1) Complete the following query to join Country to City on the Code columns.

```
SELECT Language, Population
FROM Country, City
WHERE                    ;
```

Check Show answer

2) Complete the following query to generate a result with columns Town and Language. Town is the name of the city where the language is spoken.

```
SELECT
, Language
FROM Country, City
WHERE Country.Code = City.Code;
```

Check Show answer

Inner and full joins

A ***join clause*** determines how a join query handles unmatched rows. Two common join clauses are:

- ***INNER JOIN***, which selects only matching left and right table rows.
- ***FULL JOIN***, which selects all left and right table rows, regardless of match.

In a FULL JOIN result table, unmatched left table rows appear with NULL values in right table columns and vice versa.

The join clause appears between a FROM clause and an ON clause. Specifically:

- The FROM clause specifies the left table.
- The INNER JOIN or FULL JOIN clause specifies the right table.
- The ***ON*** clause specifies the join columns.

An optional WHERE clause follows the ON clause.

PARTICIPATION ACTIVITY 4.5.4: Inner and full joins.

Department

Code	Name	Manager
44	Engineering	2538
82	Sales	6381
12	Marketing	6381
99	Technical support	NULL

Employee

ID	Name	Salary
2538	Lisa Ellison	45000
5384	Sam Snead	30500
6381	Maria Rodriguez	92300

```
SELECT Department.Name AS Group,
      Employee.Name AS Supervisor
FROM Department
INNER JOINEmployee
ON Manager = ID;
```

Result

Group	Supervisor
Engineering	Lisa Ellison
Sales	Maria Rodriguez
Marketing	Maria Rodriguez

```
SELECT Department.Name AS Group,
      Employee.Name AS Supervisor
FROM Department
FULL JOINEmployee
ON Manager = ID;
```

Result

Group	Supervisor
Engineering	Lisa Ellison
Sales	Maria Rodriguez
Marketing	Maria Rodriguez
NULL	Sam Snead
Technical support	NULL

Animation content:

Slide 1. There are two tables named Department and Employee. Department has three columns named Code Name and Manager. Employee has three columns named ID Name and Salary. Manager is a foreign key of primary key ID. Four lines of code appear. Line 1 contains the code SELECT Department dot Name AS Group comma Employee dot Name AS Supervisor. Line 2 contains the code FROM Department. Line 3 contains the code INNER JOIN Employee. Line 4 contains the code ON Manager equals ID. Rows one and three of columns ID Name and Salary of table Employee are highlighted. Row one contains the values 2538 Lisa Ellison and 45000 respectively. Row three contains the values 6381 Maria Rodriguez and 92300 respectively. A new table Result appears with two columns labeled Group and Supervisor. Row one contains the values 44 Engineering and 2538 respectively. Row two contains the values 82 Sales and 6381 respectively. Row three contains the values 12 Marketing and 6381 respectively. These three lines are highlighted with the previous two rows. A new table named Result appears and has two columns named Group and Manager. Slide 2. Four new lines of code appear. Line one contains the code SELECT Department dot Name AS Group comma Employee dot Name AS Supervisor. Line 2 contains the code FROM Department. Line 3 contains the code FULL JOIN Employee. Line four contains the code ON Manager equals ID. A new table named Result appears with two columns labeled Group and Supervisor. Row four of Columns Code Name and Manager of Department is highlighted and contains the values 99 Technical Support and NULL. Row thee of Columns ID Name and Salary of Employee is highlighted and contains the values 6381 Maria Rodriguez and 92300 respectively. Rows four and five of the second Results table are highlighted. Row four contains the values NULL and Sam Snead respectively. Row five contains the values Technical Support and NULL respectively.

Animation captions:

1. Inner joins are written with the keywords INNER JOIN. Duplicate columns Name are replaced with the aliases Group and Supervisor.
2. No unmatched rows appear in an inner join result.
3. Full joins are written with the FULL JOIN keywords. Unmatched rows from both tables appear in the result.

PARTICIPATION ACTIVITY 4.5.5: Inner and full joins.

Refer to the tables below.

Faculty

● ID	FacultyName	○ Code
1	Grayson	ART
2	Wayne	ART
3	Stark	COMP
7	Grey	NULL

Department

● Code	DepartmentName
ART	Art Department
COMP	Computer Science Department
ENG	English Department
HIST	History Department

1) What is the result of the following query?

```
SELECT FacultyName,
DepartmentName
FROM Faculty
INNER JOIN Department
ON Faculty.Code =
Department.Code;
```

○

FacultyName	DepartmentName
Grayson	Art Department
Wayne	Art Department
Stark	Computer Science Department

○

FacultyName	DepartmentName
Grayson	Art Department
Wayne	Art Department
Stark	Computer Science Department
Grey	NULL

○

FacultyName	DepartmentName
Grayson	Art Department
Wayne	Art Department
Stark	Computer Science Department
NULL	English Department
NULL	History Department

2) What is the result of the following query?

```
SELECT FacultyName,
DepartmentName
FROM Faculty
FULL JOIN Department
ON Faculty.Code =
Department.Code;
```


FacultyName	DepartmentName
Grayson	Art Department
Wayne	Art Department
Stark	Computer Science Department
Grey	NULL

FacultyName	DepartmentName
Grayson	Art Department
Wayne	Art Department
Stark	Computer Science Department
NULL	English Department
NULL	History Department

FacultyName	DepartmentName
Grayson	Art Department
Wayne	Art Department
Stark	Computer Science Department
Grey	NULL
NULL	English Department
NULL	History Department

Left and right joins

In some cases, the database user wants to see unmatched rows from either the left or right table, but not both. To enable these cases, relational databases support left and right joins:

- ***LEFT JOIN*** selects all left table rows, but only matching right table rows.
- ***RIGHT JOIN*** selects all right table rows, but only matching left table rows.

An ***outer join*** is any join that selects unmatched rows. Left, right, and full joins are all outer joins.

Join clauses are standard SQL syntax and supported by most relational databases. Ex: MySQL supports INNER, LEFT, and RIGHT JOIN but not FULL JOIN.

PARTICIPATION ACTIVITY 4.5.6: Left and right joins.

Department

● Code	Name	○ Manager
44	Engineering	2538
82	Sales	6381

Employee

● ID	Name	Salary
2538	Lisa Ellison	45000
5384	Sam Snead	30500

12	Marketing	6381
99	Technical support	NULL

6381	Maria Rodriguez	92300

```
SELECT Department.Name AS Group,
      Employee.Name AS Supervisor
FROM Department
LEFT JOINEmployee
ON Manager = ID;
```

Result

Group	Supervisor
Engineering	Lisa Ellison
Sales	Maria Rodriguez
Marketing	Maria Rodriguez
Technical support	NULL

```
SELECT Department.Name AS Group,
      Employee.Name AS Supervisor
FROM Department
RIGHT JOINEmployee
ON Manager = ID;
```

Result

Group	Supervisor
Engineering	Lisa Ellison
Sales	Maria Rodriguez
Marketing	Maria Rodriguez
NULL	Sam Snead

Animation content:

Slide 1. There are two tables named Department and Employee. Department has three columns named Code Name and Manager. Employee has three columns named ID Name and Salary. Manager is a foreign key of primary key ID. Four lines of code appear. Line 1 contains the code SELECT Department dot Name comma Employee dot Name. Line 2 contains the code FROM Department. Line 3 contains the code LEFT JOIN Employee. Line 4 contains the code ON Department dot Manager equals Employee dot ID. A new table named Result appears and has two columns named Department dot Name and Employee dot Name. Row four of columns Code Name and Manager of the Department table is highlighted and contains the values 99 Technical Support and NULL respectively. Row four of Department dot Name and Employee dot Name of Result is highlighted and contains the values Technical Support and NULL respectively. Four new lines of code appear. Line one contains the code SELECT Department dot Name comma Employee dot Name. Line 2 contains the code FROM Department. Line 3 contains the code Right JOIN Employee. Line four contains the code ON Department dot Manager equals Employee dot ID. A new table named Result appear with two columns named Department dot Name and Employee dot Name. Row two of columns ID Name and Salary of Employee are highlighted and contains the values 5384 Sam Snead and 30500 respectively. Row four of columns Department dot Name and Employee dot Name is highlighted and contains the values NULL and Sam Snead.

Animation captions:

1. Left joins are written with the LEFT JOIN keywords. Unmatched rows from Department (the left table) appear in the result.
2. Right joins are written with the RIGHT JOIN keywords. Unmatched rows from Employee (the right table) appear in the result.

PARTICIPATION ACTIVITY 4.5.7: Inner, left, right, and full joins.

Refer to the tables below.

Faculty

● ID	FacultyName	○ Code
1	Grayson	ART
2	Wayne	ART
3	Parker	MATH

Department

● Code	DepartmentName
ART	Art Department
ENG	English Department
HIST	History Department

● ID	FacultyName	○ Code
4	Banner	MATH
5	Grey	NULL

● Code	DepartmentName
MATH	Math Department

Insert each keyword into the statement below, then match the keyword with the result.

```
SELECT FacultyName, DepartmentName
FROM Faculty
_____ JOIN Department
ON Faculty.Code = Department.Code
```


If unable to drag and drop, refresh the page.

LEFT RIGHT FULL INNER

FacultyName	DepartmentName
Banner	Math Department
Grayson	Art Department
Parker	Math Department
Wayne	Art Department

FacultyName	DepartmentName
Banner	Math Department
Grayson	Art Department
Grey	NULL
Parker	Math Department
Wayne	Art Department

FacultyName	DepartmentName
NULL	English Department
NULL	History Department
Banner	Math Department
Grayson	Art Department
Parker	Math Department
Wayne	Art Department

FacultyName	DepartmentName
NULL	English Department
NULL	History Department
Banner	Math Department
Grayson	Art Department
Grey	NULL
Parker	Math Department
Wayne	Art Department

Reset

Equijoins

An ***equijoin*** compares columns of two tables with the = operator. Most joins are equijoins. A ***non-equijoin*** compares columns with an operator other than =, such as <, >, ≤, and ≥.

In the figure below, a non-equijoin selects all buyers along with properties priced below the buyer's maximum price.

Figure 4.5.2: Non-equijoin example.

Buyer

Name	MaxPrice
Lisa Ellison	600000
Sam Snead	900000
Jiho Chen	500000
Maria Rodriguez	800000

Property

Address	Price
23 Maple Street	700000
4 Oak Street	850000
59 Alvarado Avenue	1299000
800 Richards Road	1000000

```
SELECT Name, Address
FROM Buyer
LEFT JOIN Property
ON Price < MaxPrice;
```

Result

Name	Address
Lisa Ellison	NULL
Sam Snead	23 Maple Street
Sam Snead	4 Oak Street
Jiho Chen	NULL
Maria Rodriguez	23 Maple Street

PARTICIPATION ACTIVITY 4.5.8: Equijoins and non-equijoins.

Refer to the following tables.

Class

● Code	Name	AverageGrade
MATH23	Calculus 1	3.1
BIO101	Cell Biology	3.6
CHEM89	Organic Chemistry	2.2
MATH130	Probability and Statistics	NULL

Student

● ID	○ Code	Name	StudentGrade
2943	MATH23	Alberto Rimas	3.3
4408	BIO101	Francoise Verone	3.5
5993	BIO101	Duyen Hue	2.9
2866	CHEM89	Dmitri Putin	3.0

1) Which equijoin generates the following result?

Class.Name	Student.Name
Calculus 1	Alberto Rimas
Cell Biology	Francoise Verone
Organic Chemistry	Dmitri Putin
Probability and Statistics	NULL

-
```
SELECT Class.Name,
Student.Name
FROM Class
RIGHT JOIN Student
ON Student.Code =
Class.Code
WHERE StudentGrade >= 3.0;
```

-
```
SELECT Class.Name,
Student.Name
FROM Class
LEFT JOIN Student
ON Student.Code =
Class.Code
WHERE StudentGrade >= 3.0;
```

-
```
SELECT Class.Name,
Student.Name
FROM Class
LEFT JOIN Student
ON Student.Code =
Class.Code;
```

2) Which non-equijoin returns students who scored higher than the class average?

-
```
SELECT Student.Name
FROM Class
INNER JOIN Student
ON StudentGrade <
AverageGrade;
```

-
```
SELECT Student.Name
FROM Class
INNER JOIN Student
ON StudentGrade >
AverageGrade;
```

-
```
SELECT Student.Name
FROM Class
INNER JOIN Student
ON StudentGrade >
AverageGrade
AND Student.Code =
Class.Code;
```


3) Which join clauses can be used in a non-equijoin query?

- INNER JOIN and FULL JOIN only
- LEFT JOIN and RIGHT JOIN only
- All JOIN clauses

CHALLENGE ACTIVITY 4.5.1: Inner and outer joins.

537150.4174434.qx3zqy7

Start

Complete the SQL statement to generate the Result table.

Order

●OrderCode	○PartNumber
A36	662
R58	492
Z23	492

Part

●PartNumber	PartName
662	Wingding
827	Left wizard
492	Buzzer

```
SELECT */Type your code here */
FROM Order, Part
WHERE */Type your code here */ ;
```

Result

Order.PartNumber	PartName
662	Wingding
492	Buzzer
492	Buzzer

1 2 3

Check Next

4.6 Subqueries

Learning goals

- Define and construct subqueries.
- Define and construct correlated subqueries.
- Define the EXISTS operator.

- Compare subqueries and flattened queries.

Subqueries

A ***subquery***, sometimes called a ***nested query*** or ***inner query***, is a query within another SQL query. The subquery is typically used in a SELECT statement's WHERE clause to return data to the outer query and restrict the selected results. The subquery is placed inside parentheses `()`.

PARTICIPATION ACTIVITY 4.6.1: Subquery examples.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
ALB	Albanian	T	97.9
AND	Catalan	T	32.3

Country

Code	Name	Continent
ABW	Aruba	North America
AFG	Afghanistan	Asia
AGO	Angola	Africa
ALB	Albania	Europe
AND	Andorra	Europe

```
SELECT Language, Percentage
FROM CountryLanguage
WHERE Percentage >  5.3
    (SELECT Percentage
    FROM CountryLanguage
    WHERE CountryCode = 'ABW'
        AND IsOfficial = 'T');
```

Subquery

Language	Percentage
Kongo	13.2
Albanian	97.9
Catalan	32.3

```
SELECT CountryCode, Language
FROM CountryLanguage
WHERE CountryCode IN    ( ALB, AND )
    (SELECT Code
    FROM Country
    WHERE Continent = 'Europe');
```

Subqu

CountryCode	Language
ALB	Albanian
AND	Catalan

Animation content:

Slide 1. There is a table named Country Language and has four columns named Country Code Language Is Official and Percentage. Slide 2. Two lines of code appear below. Line one of code states SELECT asterisk. Line two of code states FROM Country Language semicolon. An unnamed table appears with the same columns as table Country Language. Then all the data inside the table Country Language is duplicated and moved into the unnamed table. Slide 3. Two new lines of code appears to the right. Line one of code states SELECT CountryCode comma Language. Line two of code states FROM Country Language semicolon. An unnamed table appears and has two columns named Country Code and Language. The data from columns Country Code and Language in table Country Language are duplicated and moved into the new unnamed table.

Animation captions:

1. The outer SELECT statement uses a subquery to determine which languages are used by a larger percentage of a country's population than Aruba's official language.
2. The subquery executes first to find the official language Percentage for ABW, which is 5.3.
3. The outer query executes using the value 5.3 returned by the subquery. Three languages have Percentage > 5.3
4. The SELECT statement uses the IN operator with a subquery to determine which Languages are used in Europe.
5. The subquery first finds all Codes from Europe: ALB and AND.
6. The outer query then selects the CountryCode and Language for the CountryCodes ALB and AND.

PARTICIPATION ACTIVITY 4.6.2: Subqueries.

Refer to the CountryLanguage and Country tables below.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
ABW	Papiamento	F	76.7
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
AGO	Mbundu	F	21.6

Country

Code	Name	Continent
ABW	Aruba	North America
AFG	Afghanistan	Asia
AGO	Angola	Africa

1) What does the query return?

```
SELECT Language
FROM CountryLanguage
WHERE Percentage <
    (SELECT Percentage
    FROM CountryLanguage
    WHERE Language = 'Mbundu');
```

- Papiamento
- Mbundu
- Dutch, Balochi, Kongo

2) What does the query return?

```
SELECT Language
FROM CountryLanguage
WHERE Percentage <
    (SELECT Percentage
    FROM CountryLanguage
    WHERE IsOfficial = 'F');
```

- Papiamento
- Balochi
- The query produces an error.

3) Which subquery makes the outer query return Kongo and Mbundu?

```
SELECT Language
FROM CountryLanguage
WHERE CountryCode = (___);
```

- `SELECT Language FROM Country WHERE Name = 'Angola'`
- `SELECT Code FROM Country WHERE Name = 'Angola'`
- `SELECT Name FROM Country WHERE Code = 'AGO'`

4) Which subquery makes the outer query return Dutch, Papiamento, and Balochi?

```
SELECT Language
FROM CountryLanguage
WHERE CountryCode IN (___);
```

- `SELECT CountryCode FROM Country WHERE Continent != 'Africa'`
- `SELECT Code FROM Country WHERE Continent != 'Africa'`
- `SELECT Code FROM Country WHERE Continent = 'Africa'`

PARTICIPATION ACTIVITY 4.6.3: Select songs with subquery.

The given SQL creates a Song table and inserts some songs. The first SELECT statement selects songs released after 1992. The second SELECT statement selects the release year for song with ID 800.

Create a third query that combines the two existing queries. The first SELECT should be the outer query, and the second SELECT should be the subquery. The ORDER BY clause should appear after the subquery.

Run your solution and verify the new query returns a result table with five rows, all with release years after 1992.

```
CREATE TABLE Song (
  ID INT,
  Title VARCHAR(60),
  Artist VARCHAR(60),
  ReleaseYear INT,
  Genre VARCHAR(20),
  PRIMARY KEY (ID)
);

INSERT INTO Song VALUES
  (100, 'Hey Jude', 'Beatles', 1968, 'pop rock'),
  (200, 'You Belong With Me', 'Taylor Swift', 2008, 'country pop'),
  (300, 'You\'re Still the One', 'Shania Twain', 1998, 'country pop'),
  (400, 'Need You Now', 'Lady Antebellum', 2011, 'country pop'),
  (500, 'You\'ve Lost That Lovin\' Feeling', 'The Righteous Brothers', 1964,
```

```
  (600, 'That\'s The Way Love Goes', 'Janet Jackson', 1993, 'R&B'),
  (700, 'Smells Like Teen Spirit', 'Nirvana', 1991, 'grunge'),
  (800, 'Even Flow', 'Pearl Jam', 1992, 'grunge'),
  (900, 'Black Hole Sun', 'Soundgarden', 1994, 'grunge');

SELECT *
FROM Song
WHERE ReleaseYear > 1992
ORDER BY ReleaseYear;

SELECT ReleaseYear
FROM Song
WHERE ID = 800;

-- Write your SELECT statement here:

```


Run Reset code

▶ View solution

Correlated subqueries

A subquery is ***correlated*** when the subquery's WHERE clause references a column from the outer query. In a correlated subquery, the rows selected depend on which row is currently being examined by the outer query.

If a column name in the correlated subquery is identical to a column name in the outer query, the TableName.ColumnName differentiates the columns. Ex: City.CountryCode refers to the City table's CountryCode column .

An alias can also help differentiate the columns. An ***alias*** is a temporary name assigned to a column or table. The ***AS*** ke try AS C creates the alias N fo omitted. Ex: SELECT Name N FROM Country C.

In the example below, the outer SELECT statement uses a correlated subquery to find cities with a population larger than the country's average city population.

PARTICIPATION ACTIVITY 4.6.4: Correlated subquery example.

City

Id	Name	CountryCode	Population
69	Buenos Aires	ARG	2982146
70	La Matanza	ARG	1266461
206	São Paulo	BRA	9968485
207	Rio de Janeiro	BRA	5598953

```
SELECT Name, CountryCode, Population
FROM City C
WHERE Population >
    (SELECT AVG(Population)
    FROM City
    WHERE CountryCode = C.CountryCode);
```

Name	CountryCode	Population
Buenos Aires	ARG	2982146
São Paulo	BRA	9968485

Animation content:

Slide 1. There is a table named Country Language and has four columns named Country Code Language Is Official and Percentage. Slide 2. Two lines of code appear below. Line one of code states SELECT asterisk. Line two of code states FROM Country Language semicolon. An unnamed table appears with the same columns as table Country Language. Then all the data inside the table Country Language is duplicated and moved into the unnamed table. Slide 3. Two new lines of code appears to the right. Line one of code states SELECT CountryCode comma Language. Line two of code states FROM Country Language semicolon. An unnamed table appears and has two columns named Country Code and Language. The data from columns Country Code and Language in table Country Language are duplicated and moved into the new unnamed table.

Animation captions:

1. The outer query and correlated subquery both select from the City table. The outer query uses an alias C for the City table, so C.CountryCode refers to the outer query's CountryCode column.
2. The outer query executes first to process rows in the City table. As each City row is processed, the subquery finds the average population for the city's country.
3. Then the outer query executes using the average population returned from the subquery. Buenos Aires has a population 2982146 > 2124303.5.
4. The outer query processes the next row, and the average population for ARG is calculated again. La Matanza is not selected because La Matanza's population is not > 2124303.5.
5. The outer query finds São Paulo also has a population > BRA's average population.
6. Rio de Janeiro is not selected because Rio de Janeiro's population 5598953 is not > 7783719.

PARTICIPATION ACTIVITY 4.6.5: Correlated subqueries.

Refer to the CountryLanguage and City tables below.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
ABW	Papiamento	F	76.7
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
AGO	Mbundu	F	21.6

City

Id	Name	CountryCode	Population
1	Kabul	AFG	1780000
2	Kandahar	AFG	237500
56	Luanda	AGO	2022000
57	Huambo	AGO	163100
129	Oranjestad	ABW	29034

1) What is missing to compare the subquery's CountryCode with the outer query's CountryCode?

```
SELECT Name, CountryCode
FROM City
WHERE 2 <=
    (SELECT COUNT(*)
    FROM CountryLanguage
    WHERE CountryCode =
);
```


Check Show answer

2) What is missing to compare the subquery's CountryCode with the outer query's CountryCode?

```
SELECT Name, CountryCode
FROM City T
WHERE 2 <=
    (SELECT COUNT(*)
    FROM CountryLanguage
    WHERE CountryCode =
);
```

Check Show answer

3) How many times does the subquery execute?

```
SELECT Name, CountryCode
FROM City C
WHERE 2 <=
    (SELECT COUNT(*)
    FROM CountryLanguage
    WHERE CountryCode =
C.CountryCode);
```

Check Show answer

4) How many cities does the query return?

```
SELECT Name, CountryCode
FROM City C
WHERE 2 <=
    (SELECT COUNT(*)
    FROM CountryLanguage
    WHERE CountryCode =
C.CountryCode);
```

Check Show answer

5) What is missing to select the languages used most in each country?

```
SELECT Language
FROM CountryLanguage C
WHERE
=
    (SELECT MAX(Percentage)
    FROM CountryLanguage
    WHERE CountryCode =
C.CountryCode);
```

Check Show answer

EXISTS operator

Correlated subqueries commonly use the ***EXISTS*** operator, which returns TRUE if a subquery selects at least one row and FALSE if no rows are selected. The ***NOT EXISTS*** operator returns TRUE if a subquery selects no rows and FALSE if at least one row is selected.

PARTICIPATION ACTIVITY 4.6.6: Correlated subquery using EXISTS.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ARG	Italian	F	1.7
ARG	Spanish	T	96.8
BRA	German	F	0.5
BRA	Portuguese	T	97.5

City

Id	Name	CountryCode	Population
69	Buenos Aires	ARG	2982146
70	La Matanza	ARG	1266461
206	São Paulo	BRA	9968485
207	Rio de Janeiro	BRA	5598953

```
SELECT Name, CountryCode
FROM City C
WHERE EXISTS
    (SELECT *
    FROM CountryLanguage
    WHERE CountryCode = C.CountryCode
        AND Percentage > 97);
```

Name	CountryCode
São Paulo	BRA
Rio de Janeiro	BRA

Animation content:

Slide 1. There is a table named Country Language and has four columns named Country Code Language Is Official and Percentage. Slide 2. Two lines of code appear below. Line one of code states SELECT asterisk. Line two of code states FROM Country Language semicolon. An unnamed table appears with the same columns as table Country Language. Then all the data inside the table Country Language is duplicated and moved into the unnamed table. Slide 3. Two new lines of code appears to the right. Line one of code states SELECT CountryCode comma Language. Line two of code states FROM Country Language semicolon. An unnamed table appears and has two columns named Country Code and Language. The data from columns Country Code and Language in table Country Language are duplicated and moved into the new unnamed table.

Animation captions:

1. The outer query uses EXISTS with a correlated subquery to select only cities in countries where at least one language is spoken by more than 97% of the population.
2. The correlated subquery selects no rows for ARG because no Percentage value is > 97. Since no rows are selected, EXISTS returns FALSE and no ARG cities are selected.
3. The correlated subquery selects one row for BRA because Portuguese's percentage 97.5 > 97. EXISTS returns TRUE when at least one row is selected, so all BRA cities are selected.
4. Rio de Janeiro is also selected because at least one BRA row has Percentage > 97.

PARTICIPATION ACTIVITY 4.6.7: EXISTS operator in subqueries.

Refer to the Employee and Family tables below.

Employee Family

Id	Name	Salary
2538	Lisa Ellison	45000
5384	Sam Snead	30400
6381	Maria Rodriguez	92300

Id	Number	Relationship	Name
2538	1	Spouse	Henry Ellison
2538	2	Son	Edward Ellison
5384	1	Son	Braden Snead
6381	1	Spouse	Jose Rodriguez
6381	2	Daughter	Gina Rodriguez
6381	3	Daughter	Clara Rodriguez

1) What does the query return?

```
SELECT Name
FROM Employee E
WHERE EXISTS
   (SELECT *
   FROM Family
   WHERE Id = E.Id
      AND Relationship =
'Spouse');
```

- Sam Snead
- Lisa Ellison and Maria Rodriguez
- All names

2) What does the query return?

```
SELECT Name
FROM Employee E
WHERE NOT EXISTS
   (SELECT *
   FROM Family
   WHERE ID = E.ID
      AND Relationship =
'Spouse');
```

- Sam Snead
- Lisa Ellison and Maria Rodriguez
- All names

3) Which subquery makes the outer query return only Jose, Gina, and Clara Rodriguez?

```
SELECT Name
FROM Family F
WHERE EXISTS (___);
```

-
```
SELECT *
   FROM Employee
   WHERE Salary > 50000
```

-
```
SELECT *
   FROM Employee
   WHERE ID = F.ID
      AND Salary > 50000
```

-
```
SELECT *
   FROM Employee
   WHERE ID = F.ID
      AND Salary > 35000
```

4) Which subquery makes the outer query return two rows?

```
SELECT *
FROM Employee E
WHERE EXISTS (___);
```

- ○
```
SELECT *
   FROM Family
   WHERE ID = E.ID
      AND Relationship =
'Son'
```

- ○
```
SELECT *
   FROM Family
   WHERE ID = E.ID
      AND Relationship IN
('Son', 'Daughter')
```

- ○
```
SELECT *
   FROM Family
   WHERE ID = E.ID
      AND Relationship !=
'Spouse'
```


PARTICIPATION ACTIVITY 4.6.8: Select albums with EXISTS.

The given SQL creates Album and Song tables and inserts albums and songs. Each song is associated with an album.

1. The SELECT statement selects all albums with three or more songs. Run the query and verify the result table shows just the albums *Saturday Night Fever* and *21*.
2. Modify the GROUP BY clause to select albums with three or more songs **by the same artist**. Run the query and verify the result table shows just the album *21*.
3. Try to write a query that selects all albums with three or more songs without using a subquery. What happens?

```
CREATE TABLE Album (
  ID INT,
  Title VARCHAR(60),
  ReleaseYear INT,
  PRIMARY KEY (ID)
);

INSERT INTO Album VALUES
  (1, 'Saturday Night Fever', 1977),
  (2, 'Born in the U.S.A.', 1984),
  (3, 'Supernatural', 1999),
  (4, '21', 2011);

CREATE TABLE Song (
  ID INT,
  Title VARCHAR(60),
  Artist VARCHAR(60),
  AlbumID INT,
  PRIMARY KEY (ID),
  FOREIGN KEY (AlbumID) REFERENCES Album(ID)
);

INSERT INTO Song VALUES
  (100, 'Stayin\' Alive', 'Bee Gees', 1),
  (101, 'More Than a Woman', 'Bee Gees', 1),
  (102, 'If I Can\'t Have You', 'Yvonne Elliman', 1),
  (200, 'Dancing in the Dark', 'Bruce Springsteen', 2),
  (201, 'Glory Days', 'Bruce Springsteen', 2),
  (300, 'Smooth', 'Santana', 3),
```

```
30  (400, 'Rolling in the Deep', 'Adele', 4),
31  (401, 'Someone Like You', 'Adele', 4),
32  (402, 'Set Fire to the Rain', 'Adele', 4),
33  (403, 'Rumor Has It', 'Adele', 4);
34
35 SELECT *
```

Run Reset code

▶ View solution

Flattening subqueries

Many subqueries can be rewritten as a join. Most databases optimize a subquery and outer query separately, whereas joins are optimized in one pass. So joins are usually faster and preferred when performance is a concern.

Replacing a subquery with an equivalent join is called ***flattening*** a query. The criteria for flattening subqueries are complex and depend on the SQL implementation in each database system. Most subqueries that follow IN or EXISTS, or return a single value, can be flattened. Most subqueries that follow NOT EXISTS or contain a GROUP BY clause cannot be flattened.

The following steps are a first pass at flattening a query:

1. Retain the outer query SELECT, FROM, GROUP BY, HAVING, and ORDER BY clauses.
2. Add INNER JOIN clauses for each subquery table,
3. Move comparisons between subquery and outer query columns to ON clauses.
4. Add a WHERE clause with the remaining expressions in the subquery and outer query WHERE clauses.
5. If necessary, remove duplicate rows with SELECT DISTINCT.

Af e original and flattened qu

PARTICIPATION ACTIVITY 4.6.9: Flattening subqueries.

Country

Code	Name	Continent
AUS	Australia	Australia
SAF	South Africa	Africa
SPA	Spain	Europe
USA	United States	North America

City

ID	Name	CountryCode	Population
144	Salzburg	AUS	152367
384	Cape Town	SAF	4618000
471	Durban	SAF	3442361
650	Barcelona	SPA	1620000
938	Madrid	SPA	3233000
942	Denver	USA	705576


```
SELECT Name
FROM Country
WHERE Code IN
   (SELECT CountryCode
    FROM City
    WHERE Population > 1000000);
```

Name
South Africa
Spain

```
SELECT DISTINCT Name
FROM Country
INNER JOIN City ON Code = CountryCode
WHERE Population > 1000000;
```

Name
South Africa
Spain

Animation content:

Slide 1. There is a table named Country that has three columns named Code, Name, and Continent. There is a second table named City that has four columns named ID, Name, CountryCode, and Population. Six lines of code appear below the Country table. Line one of code states SELECT name. Line two of code states FROM Country. Line three states WHERE Code IN. Line four states left-parenthesis SELECT CountryCode. Line five states FROM City. Line six states WHERE Population greater than 1000000 right-parenthesis semicolon. Lines three through five are labeled subquery. A red box appears around the subquery. Slide 2. The red box moves to surround lines one and two. An unnamed table appears and has one column named Name. South Africa and Spain are duplicated and moved from the Country table to the unnamed table. Slide 3. The six lines of code are duplicated and moved below the City table. Lines three through six fade out and are replaced by two lines of code. The first new line states INNER JOIN City ON Code equals CountryCode. The second new line states WHERE Population greater than 1000000 semicolon. Slide 4. A new unnamed table appears and has one column named Name. Four rows are added to the new unnamed table: South Africa, South Africa, Spain, Spain. Slide 5. In the first line of code, Name is moved to the right and DISTINCT is added. The first line now reads SELECT DISTINCT Name. In the second unnamed table, two rows are removed: South Africa and Spain.

Animation captions:

1. The subquery selects country codes for cities with population > 1000000.
2. The outer query selects the country names.
3. To flatten the query, replace the subquery with an INNER JOIN clause.
4. The join query selects the one country name for each city with population > 1000000.
5. The DISTINCT clause eliminates duplicate rows. The subquery and join query are equivalent.

PARTICIPATION ACTIVITY 4.6.10: Flattening correlated subqueries.

Given the data in the tables below, which query pairs return the same result table?

Employee

ID	Name	Salary
2538	Lisa Ellison	45000
5384	Sam Snead	30400
6381	Maria Rodriguez	92300

Family

ID	Number	Relationship	Name
2538	1	Spouse	Henry Ellison
2538	2	Son	Edward Ellison
5384	1	Son	Braden Snead
6381	1	Spouse	Jose Rodriguez
6381	2	Daughter	Gina Rodriguez
6381	3	Daughter	Clara Rodriguez

1)
```
SELECT Name
FROM Employee
WHERE EXISTS
   (SELECT *
    FROM Family
    WHERE Family.ID = Employee.ID
       AND Relationship =
'Spouse');

SELECT Name
FROM Employee
INNER JOIN Family ON Family.ID =
Employee.ID
WHERE Relationship = 'Spouse';
```

- Same result
- Different result

2)
```
SELECT Name
FROM Employee
WHERE EXISTS
   (SELECT *
    FROM Family
    WHERE Family.ID = Employee.ID
       AND Relationship =
'Daughter');

SELECT Name
FROM Employee
INNER JOIN Family ON Family.ID =
Employee.ID
WHERE Relationship = 'Daughter';
```

- Same result
- Different result

3) In question 2, change the second query to SELECT DISTINCT.

- Same result
- Different result

4)
```
SELECT Name
FROM Employee
WHERE NOT EXISTS
   (SELECT *
    FROM Family
    WHERE Family.ID = Employee.ID
       AND Relationship =
'Spouse');

SELECT Name
FROM Employee
INNER JOIN Family ON Family.ID =
Employee.ID
WHERE Relationship != 'Spouse';
```

- Same result
- Different result

5) In question 4, change the second query to SELECT DISTINCT.

- Same result
- Different result

CHALLENGE ACTIVITY 4.6.1: Subqueries.

537150.4174434.qx3zqy7

Start

Course

CourseId	CourseCode	CourseName	Capacity	InstructorId
8338	MATH667	Algebra	125	1
9165	PHIL37	Critical Thinking	25	2
8318	MATH61	Linear Algebra	100	1
9552	BUS449	Financial Accounting	175	3
6260	BUS409	Financial Management	75	3

I

InstructorId	InstructorName
1	Ari Reed
2	Mia Sanz
3	Noa Ward

Note: Both tables may not be n

Select the values returned by the query below.

```
SELECT CourseId
FROM Course
WHERE InstructorId =
   (SELECT InstructorId
   FROM Instructor
   WHERE InstructorName = 'Noa Ward');
```

- [ ] 8318
- [ ] 3
- [ ] 6260

1 2 3 4 5

Check Next

4.7 Queries in Python

Learning goals

- Use Python to connect to a MySQL database.
- Explain Python connection and cursor objects.
- Construct query parameters and fetch query results.
- Define SQL injection attacks and explain how to prevent an attack.

Connections

This section assumes familiarity with the Python language.

This section describes database programming in Python and MySQL using Connector/Python. ***Connector/Python*** is a MySQL component based on the DB-API standard from the Python Software Foundation. A Python program uses Connector/Python by importing the `mysql.connector` module. A ***module*** is a file containing Python classes, functions, or variables.

Python programs must connect to a database prior to executing queries. A connection is an object of the ***MySQLConnection*** class that is created with the ***mysql.connector.connect()*** function, specifying the database server address, database name, login username, and password.

Creating a connection fails if the database is not found or login credentials are invalid. Therefore, connections are usually created within the `try` block of a `try-except` statement. If the connection fails, the `except` block executes and typically prints an error message.

The Python program releases the connection when no longer needed with the ***connection.close()*** method.

PARTICIPATION ACTIVITY 4.7.1: Creating a connection.

```
import mysql.connector
from mysql.connector import errorcode

try:
   reservationConnection = mysql.connector.connect(
      user='samsnead',
      password='*jksi72$',
      host='127.0.0.1',
      database='Reservation')

except mysql.connector.Error as err:
   if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
      print('Invalid credentials')
   elif err.errno == errorcode.ER_BAD_DB_ERROR:
      print('Database not found')
   else:
      print('Cannot connect to database:', err)

else:
   # Execute database operations...
   reservationConnection.close()
```

Animation content:

Animation displays and highlights parts of a Python program.

import mysql.connector
from mysql.connector import errorcode

try:
  reservationConnection = mysql.connector.connect(
    user='samsnead',
    password='*jksi72$',
    host='127.0.0.1',
    database='Reservation')

except mysql.connector.Error as err:

```
  if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
    print('Invalid credentials')
  elif err.errno == errorcode.ER_BAD_DB_ERROR:
    print('Database not found')
  else:
    print('Cannot connect to database:', err)

else:
  # Execute database operations...
  reservationConnection.close()
```


Animation captions:

1. The import statements make Connector/Python code and errorcode module available to the program.
2. The mysql.connector.connect() function creates a MySQLConnection object named reservationConnection.
3. If the connection fails, the except block executes and prints an error message.
4. If the connection succeeds, the else block executes. Subsequent queries use the reservationConnection object.
5. When database processing is finished, close() releases the connection.

PARTICIPATION ACTIVITY 4.7.2: Python connections.

1) A connection must always be created within the `try` clause of a `try-except-else` statement.
   - True
   - False

2) In the animation above, MySQL server is running on a machine with address 127.0.0.1.
   - True
   - False

3) In the animation above, the `close()` method is called if the password is incorrect.
   - True
   - False

4) A connection must be created prior to executing any database operation.
   - True
   - False

Cursors

A cursor is an object of the ***MySQLCursor*** class that executes SQL statements and stores the results. Cursors are created with the ***connection.cursor()*** method and are always associated with a specific database.

SQL statements are compiled (translated to low-level database operations) and executed with the ***cursor.execute()*** method. The `cursor.execute()` method has one required parameter: a string containing the SQL statement.

A cursor is always associated with one connection but may be reused for multiple SQL statements. Cursors must be released with the ***cursor.close()*** method prior to closing the associated connection.

When using Connector/Python with MySQL and the InnoDB storage engine, the results of INSERT, UPDATE, and DELETE statements are not automatically saved in the database. The ***connection.commit()*** method saves all changes. The ***connection.rollback()*** method discards all changes. The `commit()` and `rollback()` methods must appear prior to releasing the cursor used to execute the changes.

MySQLCursor contains additional methods and properties, such as:

- The ***cursor.rowcount*** property is the number of rows returned or altered by a query.
- The ***cursor.column_names*** property is a list of column names in a query result.
- The ***cursor.fetchwarnings()*** method returns a list of warnings generated by a query.

PARTICIPATION ACTIVITY 4.7.3: Using cursors.

```
flightCursor = reservationConnection.cursor()

flightCreate = ('CREATE TABLE Flight ('
                'FlightNumber INT PRIMARY KEY NOT NULL,'
                'AirlineName VARCHAR(20),'
                'AirportCode CHAR(3) )')

flightCursor.execute(flightCreate)

flightQuery = ("INSERT INTO Flight "
               "(FlightNumber, AirlineName, AirportCode) "
               "VALUES (280, 'China Airlines', 'PEK')")

flightCursor.execute(flightQuery)

reservationConnection.commit()
flightCursor.close()
```

Animation content:

Animation displays and highlights parts of a Python program.

flightCursor = reservationConnection.cursor()

flightCreate = ('CREATE TABLE Flight ('
'FlightNumber INT PRIMARY KEY NOT NULL,'
'AirlineName VARCHAR(20),'
'AirportCode CHAR(3))')

flightCursor.execute(flightCreate)

flightQuery = ("INSERT INTO Flight "
"(FlightNumber, AirlineName, AirportCode) "
"VALUES (280, 'China Airlines', 'PEK')")

```
flightCursor.execute(flightQuery)

reservationConnection.commit()
flightCursor.close()
```

Animation captions:

1. cursor() creates a cursor called flightCursor, associated with the Reservation database.
2. execute() creates the Flight table.
3. The next execute() inserts a row into the Flight table.
4. commit() saves the insert in the database before the cursor is released.

PARTICIPATION ACTIVITY 4.7.4: Cursors.

1) Which class contains the `cursor()` method?
   - MySQLConnection
   - MySQLCursor
   - `reservationConnection`

2) The `execute()` method:
   - Executes an SQL statement that must be previously compiled.
   - Compiles but does not execute an SQL statement
   - Both compiles and executes an SQL statement.

3) Which MySQLCursor method should always be called prior to closing the cursor's connection?
   - `close()`
   - `commit()`
   - `rollback()`
   - Either `commit()` or `rollback()`

4) If the INSERT in the animation above successfully inserts a new flight, what is `flightCursor.rowcount` after the INSERT executes?
   - 0
   - 1
   - -1

Query parameters

The `cursor.execute()` method can substitute Python values into the given SQL statement using query parameters:

- The SQL statement uses the placeholder characters `%s` to represent the query values.
- A tuple containing values to be assigned to the placeholders is passed as the second argument to `cursor.execute()`.

The tuple values are mapped to the `%s` placeholders, in sequence, when the SQL statement executes. Python data types are automatically converted to MySQL data types.

A database programmer must be cautious when inserting input data into an SQL statement. An ***SQL injection attack*** is when a user intentionally enters values that alter the intent of an SQL statement. The `cursor.execute()` method prevents SQL injection when assigning values to placeholders.

PARTICIPATION ACTIVITY 4.7.5: Using query parameters to insert a new flight.

```
print('Flight number? ', end = '')
flightNum = input()
print('Airline? ', end = '')
airline = input()
print('Airport code? ', end = '')
airportCode = input()

flightCursor = reservationConnection.cursor()
flightQuery = ('INSERT INTO Flight '
               '(FlightNumber, AirlineName, AirportCode) '
               'VALUES (%s, %s, %s)')
flightData = (flightNum, airline, airportCode)
flightCursor.execute(flightQuery, flightData)

reservationConnection.commit()
flightCursor.close()
```

Flight number? 105
Airline? Air India
Airport code? DEL

Animation content:

Animation displays and highlights parts of a Python program.

print('Flight number? ', end = '')
flightNum = input()
print('Airline? ', end = '')
airline = input()
print('Airport code? ', end = '')
airportCode = input()

flightCursor = reservationConnection.cursor()
flightQuery = ('INSERT INTO Flight '
'(FlightNumber, AirlineName, AirportCode) '
'VALUES (%s, %s, %s)')

flightData = (flightNum, airline, airportCode)
flightCursor.execute(flightQuery, flightData)

```
reservationConnection.commit()
flightCursor.close()
```

Animation captions:

1. The user enters flight data.
2. flightQuery contains three %s placeholders. Values are assigned when the cursor executes.
3. flightData tuple contains three values.
4. When the cursor executes, flightData values are assigned to placeholders. Python data types are automatically converted to MySQL data types.
5. commit() saves the insert in the database, and close() releases the cursor resources.

PARTICIPATION ACTIVITY 4.7.6: Query parameters.

1) Which line assigns the correct tuple for the UPDATE statement?

```
flightQuery = ('UPDATE Flight '
               'SET AirlineName =
%s '
               'WHERE
FlightNumber = %s')
flightCursor.execute(flightQuery,
flightData)
```

- `flightData = ('United Airlines', 'PEK')`
- `flightData = ('United Airlines', 'PEK', 105)`
- `flightData = ('United Airlines', 105)`

2) What does the code output?

```
try:
   flightQuery = ('UPDATE Flight
'
                  'SET
AirlineName = %s '
                  'WHERE
FlightNumber = %s')
   flightData = (301, 'United
Airlines')

flightCursor.execute(flightQuery,
flightData)
   print('Flight updated.')
except
mysql.connector.errors.DataError:
   print('Unable to update
flight.')
```

- Flight updated.
- Unable to update flight.
- Nothing is output.

3) Assume the `airline` and `flightNum` values are obtained from user input. What is wrong with the following code?

```
flightQuery =
   ('UPDATE Flight SET
AirlineName = "' + airline +
    '" WHERE FlightNumber = ' +
flightNum)
flightCursor.execute(flightQuery)
```

- `execute()` is missing a second argument.
- The code is susceptible to an SQL injection attack.
- Nothing is wrong.

Fetching values

When `cursor.execute()` executes a SELECT statement, the query results are accessed with fetch methods:

- ***cursor.fetchone()*** returns a tuple containing a single result row or the value `None` if no rows are selected. If a query returns multiple rows, `cursor.fetchone()` may be executed repeatedly until it returns `None`.
- ***cursor.fetchall()*** returns a list of tuples containing all result rows. The tuple list can be processed in a loop. Ex: `for rowTuple in cursor.fetchall()` assigns each row to `rowTuple` and terminates when all rows are processed.

PARTICIPATION ACTIVITY 4.7.7: Fetching flight results.

```
flightCursor = reservationConnection.cursor()
flightQuery = ('SELECT FlightNumber, DepartureTime FROM Flight '
               'WHERE AirportCode = %s AND AirlineName = %s')
flightData = ('PEK', 'China Airlines')
flightCursor.execute(flightQuery, flightData)

for row in flightCursor.fetchall():
   print('Flight', row[0], 'departs at', row[1])

flightCursor.close()
```

```
Flight 107 departs at 13:30:00
Flight 350 departs at 8:15:00
Flight 482 departs at 10:07:00
```


Animation content:

Animation displays and highlights parts of a Python program.

flightCursor = reservationConnection.cursor()
flightQuery = ('SELECT FlightNumber, DepartureTime FROM Flight '
'WHERE AirportCode = %s AND AirlineName = %s')

```
flightData = ('PEK', 'China Airlines')
flightCursor.execute(flightQuery, flightData)

for row in flightCursor.fetchall():
  print('Flight', row[0], 'departs at', row[1])

flightCursor.close()
```

Animation captions:

1. flightCursor executes a SELECT query.
2. fetchall() returns a set of tuples. Each tuple is a row of the result table.
3. The for statement processes each tuple in the set. row[0] contains data from the first column, and row[1] from the second column.

PARTICIPATION ACTIVITY 4.7.8: Fetching values.

Refer to the code below.

```
flightQuery = ('SELECT AirlineName, DepartureTime, AirportCode FROM Flight '
 'WHERE FlightNumber = %s')
flightData = (140,) # Comma required for single value tuple
flightCursor.execute(flightQuery, flightData)

row = flightCursor.___A___()
if row is ___B___:
 print("Flight not found")
else:
 print('Flight', row[0], 'departs from', ___C___)

flightCursor.close()
```

1) What is identifier A?

Check Show answer

2) What is identifier B?

Check Show answer

3) What is identifier C?

Check Show answer

CHALLENGE ACTIVITY 4.7.1: Database programming with Python.

537150.4174434.qx3zqy7

Start

The Python code below prints out the names and capacities of all the rooms in the database. Fill in the missing keywords.

```
from mysql import connector  # load connector functions

connection = connector._____(A)_____(
    user='Blaise',
    password='Pascal',
    host='localhost',
    database='college'
)

rowcursor = connection._____(B)_____()
query = 'SELECT * FROM Room'
_____(C)_____.execute(_____(D)_____)
for row in rowcursor._____(E)_____(): print(row)
rowcursor._____(F)_____()
_____(G)_____.close()
```

(A) Ex: keyword (D) (G)

(B) (E)

(C) (F)

1

Check Try again

4.8 Case study: Queries in SQL and pandas

Learning goals

- Compare use cases of SQL and Python.
- Translate SQL queries to `pandas` queries.
- Write `pandas` queries in Jupyter.

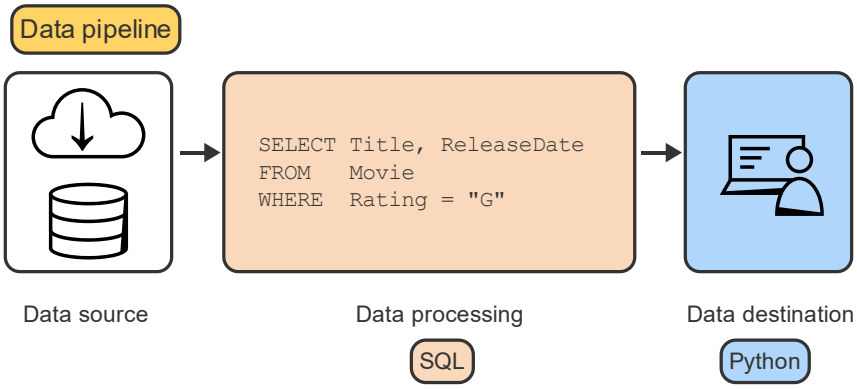
Which language: Python or SQL?

Data scientists beginning their careers often ask which language is best: Python or SQL? The answer: Neither! Both languages have advantages and disadvantages, and data scientists often use both in their daily work.

SQL is a query and retrieval language. SQL is intended to search a database and extract instances that meet a particular condition. SQL does not contain functions for analyzing or modeling a dataset. Python is useful for processing data, summarizing data, manipulating data, and modeling data.

SQL is typically faster at running queries than Python, but querying data from SQL requires an additional step and programming language. Unlike SQL, Python has a package manager which allows data scientists to import new Python functions vs. writing packages from scratch. SQL can be used to calculate basic descriptive statistics on a dataset, but advanced data visualization and modeling is not possible.

PARTICIPATION ACTIVITY 4.8.1: SQL and Python in a data pipeline.



Animation content:

Static image: Diagram of data pipelines. Data source points to data processing, which points to data destination.
Step 1: A data pipeline is a series of data processing steps. Data pipelines use code from SQL and Python to automate data gathering and analysis.
Step 2: Data pipelines contain three elements: the data source, the data processing, and the data destination.
Step 3: The data source is the data's original location. The data source is usually a cloud-based data warehouse or database. A box appears containing icons representing cloud-based storage and databases.
Step 4: During data processing, a query requests the desired features and instances for the analysis. Data may be combined from multiple sources during processing. A box appears containing an SQL query from a movie-rating database.
Step 5: The data destination is a place where the data scientist can access and interact with the data. A box appears containing an icon of a person with a laptop computer.
Step 6: SQL is the most commonly used language at the data processing step. Once data has reached its destination, an analysis language such as Python is used. A text box containing "SQL" appears under the data processing step. A text box containing "Python" appears under the data destination step.

Animation captions:

1. A data pipeline is a series of data processing steps. Data pipelines use code from SQL and Python to automate data gathering and analysis.
2. Data pipelines contain three elements: the data source, the data processing, and the data destination.

3. The data source is the data's original location. The data source is usually a cloud-based data warehouse or database.
4. During data processing, a query requests the desired features and instances for the analysis. Data may be combined from multiple sources during processing.
5. The data destination is a place where the data scientist can access and interact with the data.
6. SQL is the most commonly used language at the data processing step. Once data has reached its destination, an analysis language such as Python is used.

PARTICIPATION ACTIVITY 4.8.2: SQL or Python?

1) SQL is used during data processing because _____.

- SQL queries are easier to write than Python queries
- SQL queries tend to execute faster than Python queries
- Python cannot be used to write queries

2) Python is used at the data destination because data scientists _____.

- need a package manager to import libraries for data analysis
- are not comfortable writing SQL code
- create models and visualizations from the resulting dataset

Translating queries from SQL to pandas

`pandas` contains a method, `DataFrame.query()`, which allows users to write SQL-like queries. Queries in `pandas` must contain Boolean expressions, which evaluate to `True/False`. All instances in the dataframe for which the expression is `True` are returned by the query.

Queries in `pandas` refer to features as text within a string. Ex: `DataFrame.query("Feature == 0")` will return all instances in the dataset for which `Feature` is exactly equal to 0. `pandas` comparison operators can be used within `.query()`, as can logical operators. Unlike other Python methods, `.query()` also recognizes English language logical operators "and", "or", and "not".

Table 4.8.1: Comparison operators.

Comparison operator	Description
`==`	Outputs `True` if the two operands are equal.
`!=`	Outputs `True` if the two operands are not equal.
`>`	Outputs `True` if the left operand is greater than the right operand.

>=	Outputs `True` if the left operand is greater than or equal to the right operand.
<	Outputs `True` if the left operand is less than the right operand.
<=	Outputs `True` if the left operand is less than or equal to the right operand.

Table 4.8.2: Logical operators.

Logical operator	English operator	Description
&	and	Outputs `True` if the two operands are both `True`.
|	or	Outputs `True` if at least one of the operands is `True`.
~	not	Outputs the opposite truth value of the expression.

PARTICIPATION ACTIVITY 4.8.3: Converting SQL queries to pandas.

Country

CountryCode	CountryName	SurfaceArea	ContinentName
JPN	Japan	377829	Asia
ITA	Italy	301316	Europe
ECU	Ecuador	283561	South America

SQL

```
SELECT CountryName
FROM   Country
WHERE  SurfaceArea > 30000;
```

pandas

```
Country.query("SurfaceArea > 30000")['CountryName']
```

Result

CountryName
Japan
Italy

Animation content:

Static image: A dataset labeled Country appears with four features (Country Code, CountryName, SurfaceArea, and ContinentName) and three countries (Japan, Italy, and Ecuador). A SQL query and the corresponding pandas query appear, with lines linking parts of the queries together.

CountryCode	CountryName	SurfaceArea	ContinentName
JPN	Japan	377829	Asia
ITA	Italy	301316	Europe
ECU	Ecuador	283561	South America

Step 1: In SQL, SELECT, FROM, and WHERE are keywords for defining a query. But pandas structures queries using Python syntax and not SQL keywords.
SQL query appears.
SELECT CountryName
FROM Country
WHERE SurfaceArea > 30000;
Step 2: In pandas, the dataframe is called first instead of the features. Adding .query() defines the query.
Step 3: The WHERE statement becomes a string passed to the .query() method. The entire query should be within a single string.
Step 4: The SELECT clause is replaced by a subset using square brackets.
pandas query is completed.
Country.query("SurfaceArea > 30000")['CountryName']
Step 5: The query selects the names of countries with an area greater than 300,000. The SQL and pandas queries return the same result.

Animation captions:

1. In SQL, SELECT, FROM, and WHERE are keywords for defining a query. But pandas structures queries using Python syntax and not SQL keywords.
2. In pandas, the dataframe is called first instead of the features. Adding .query defines the query.
3. The WHERE statement becomes a string passed to the .query() method. The entire query should be within a single string.
4. The SELECT clause is replaced by a subset using square brackets.
5. The query selects the names of countries with an area greater than 300,000. The SQL and pandas queries return the same result.

PARTICIPATION ACTIVITY 4.8.4: Converting a SQL query to pandas.

Consider the SQL query below.

```
SELECT CountryCode
FROM Country
WHERE ContinentName = "Asia"
```

Fill in blanks A, B, and C in the corresponding `pandas` query.

```
__A__.query("__B__")['__C__']
```

1) `__A__`

Check Show answer

2) `__B__`

Check Show answer

3) `__C__`

Check Show answer

4) Suppose the query is applied to the `Country` dataset from the previous animation. What is the result?

Check Show answer

Queries on the movies dataset

The movies dataset contains data from a set of 7,614 movies listed on the Internet Movie Database (IMDb). The dataset was collected using a technique called web scraping. ***Web scraping*** is the use of code to automate extracting information from a series of websites. Web scraping is often used to extract information from online databases or social media sites.

The movies dataset contains features including the title, rating, genre, year, release date, IMDb average score, number of user votes, budget, gross revenue, and run time. Additional features list the stars, writers, country, and production company for each movie.

Converting SQL queries to pandas.

Full screen

The Python code below imports the movies dataset using `pandas`

- Click the double right arrow icon to restart the kernel and run all cells.
- For each cell that contains an SQL query, convert the query to `pandas` and re-run the cells.

In [1]:
```
import pandas as pd

Movie = pd.read_csv('movies.csv')
```

In [2]:
```
Movie['ReleaseDate'] = pd.to_datetime(Movie['ReleaseDate'], format='%
Movie
```


Out[2]:

	Title	Rating	Genre	Year	ReleaseDate	Score	Votes	Director
0	Superbabies: Baby Geniuses 2	PG	Comedy	2004	2004-08-27	1.9	30000.0	Bob Clark
1	The Hottie & the Nottie	PG-13	Comedy	2008	2008-02-21	1.9	36000.0	Tom Putnam
2	Disaster Movie	PG-13	Comedy	2008	2008-08-29	1.9	88000.0	Jason Friedberg
3	From Justin to Kelly	PG	Comedy	2003	2003-06-20	2.1	25000.0	Robert Iscove
4	House of the Dead	R	Action	2003	2003-10-10	2.1	36000.0	Uwe Boll

In [3]:
```
# SELECT Title
# FROM Movie
```

Data source: Daniel Grijalva. 2020. "Movie industry" https://www.kaggle.com/datasets/danielgrijalvas/movies

PARTICIPATION ACTIVITY 4.8.5: Querying the movie dataset.

How many movies are returned using the following queries?

1)
```
SELECT Title
FROM Movie
WHERE Rating="PG-13" and
ReleaseDate > "2008-01-01"
```

Check Show answer

2)
```
SELECT Title
FROM Movie
WHERE Rating In ("G", "PG")
```

Check Show answer

3)
```
SELECT Title, Year, Budget
FROM Movie
WHERE Budget > 300000000
```

Check Show answer

4)
```
SELECT Title
FROM Movie
WHERE Budget > Gross
```

Check Show answer

©zyBooks 03/21/24 21:41 2087217
Biniam abebe
UNTADTA5340OrhanSpring8wk22024