9.1 Simple functions

Numeric functions

A ***function*** operates on an expression enclosed in parentheses, called an ***argument***, and returns a value. Usually, the argument is a simple expression, such as a column name or fixed value. Some functions have several arguments, separated by commas, and a few have no arguments at all.

Each 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 the square root of positive numbers only, so `SQRT(-1)` returns NULL.

Numeric functions operate on, and evaluate to, integer and decimal data types.

Table 9.1.1: Common numeric functions.

Function	Description	Example
ABS(n)	Returns the absolute value of *n*	`SELECT ABS(-5);` returns 5
LOG(n)	Returns the natural logarithm of *n*	`SELECT LOG(10);` returns 2.302585092994046
POW(x, y)	Returns *x* to the power of *y*	`SELECT POW(2, 3);` returns 8
RAND()	Returns a random number between 0 (inclusive) and 1 (exclusive)	`SELECT RAND();` returns 0.11831825703225868
ROUND(n, d)	Returns *n* rounded to *d* decimal places	`SELECT ROUND(16.25, 1);` returns 16.3
SQRT(n)	Returns the square root of *n*	`SELECT SQRT(25);`

SQRT(*n*)	Returns the square root of *n*	returns 5

PARTICIPATION ACTIVITY 9.1.1: Numeric functions.

Referring to the Problem table below, choose the results from each SELECT statement.

Problem

ID	X	Y
1	5	2
2	9	1
3	3	10

1)
```
SELECT ABS(X - Y)
FROM Problem;
```

- ○ `3`
- ○
```
3
8
7
```
- ○
```
-3
-8
7
```

2)
```
SELECT ROUND(X / Y, 0)
FROM Problem;
```

- ○
```
2.5
9.0
0.3
```
- ○
```
3
9
0
```
- ○
```
2.5000
9.0000
0.3000
```

3)
```
SELECT ROUND(SQRT(X), 1)
FROM Problem;
```

- ○
```
2
3
2
```
- ○
```
2.23606797749979
3
```

```
1.7320508075688772
```

- ```
  2.2
  3.0
  1.7
  ```

4)
```
SELECT X, Y, POW(X, Y)
FROM Problem;
```

- ```
  5   2     25
  9   1      9
  3  10  59049
  ```
- ```
  25
  9
  59049
  ```
- ```
  5   2   10
  9   1    9
  3  10   30
  ```

String functions

String functions manipulate string values. SQL string functions are similar to string functions in programming languages like Java and Python.

Table 9.1.2: Common string functions.

Function	Description	Example
CONCAT(s1, s2, ...)	Returns the string that results from concatenating the string arguments	`SELECT CONCAT('Dis', 'en', 'gage');` returns 'Disengage'
LOWER(s)	Returns the lowercase *s*	`SELECT LOWER('MySQL');` returns 'mysql'
REPLACE(s, from, to)	Returns the string *s* with all occurrences of *from* replaced with *to*	`SELECT REPLACE('This and that', 'and', 'or');` returns 'This or that'
SUBSTRING(s, pos, len)	Returns the substring from *s* that starts at position *pos* and has length *len*	`SELECT SUBSTRING('Boomerang', 1, 4);` returns 'Boom'

TRIM(s)	Returns the string *s* without leading and trailing spaces	`SELECT TRIM('  test ');` returns 'test'
UPPER(s)	Returns the uppercase *s*	`SELECT UPPER('mysql');` returns 'MYSQL'

PARTICIPATION ACTIVITY 9.1.2: String functions.

Refer to the Avatar table and type the string that results from each SELECT statement.

Avatar

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

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

Date and time functions

Date and time functions operate on DATE, TIME, and DATETIME data types.

Table 9.1.3: Common date and time functions.

Function	Description	Example
CURDATE() ***CURTIME()*** ***NOW()***	Returns the current date, time, or date and time in `'YYYY-MM-DD'`, `'HH:MM:SS'`, or `'YYYY-MM-DD HH:MM:SS'` format	`SELECT CURDATE();` returns '2019-01-25' `SELECT CURTIME();` returns '21:05:44' `SELECT NOW();` returns '2019-01-25 21:05:44'
DATE(expr) ***TIME(expr)***	Extracts the date or time from a date or datetime expression *expr*	`SELECT DATE('2013-03-25 22:11:45');` returns '2013-03-25' `SELECT TIME('2013-03-25 22:11:45');` returns '22:11:45'
DAY(d)	Returns the day, month, or year from	`SELECT DAY('2016-10-25');` returns 25 `SELECT MONTH('2016-10-25');`

MONTH(d) ***YEAR(d)***	date *d*	returns 10 `SELECT YEAR('2016-10-25');` returns 2016

HOUR(t)
 MINUTE(t)
 SECOND(t)
	Returns the hour, minute, or second from time *t*	`SELECT HOUR('22:11:45');`
		 returns 22
		 `SELECT MINUTE('22:11:45');`
		 returns 11
		 `SELECT SECOND('22:11:45');`
		 returns 45

DATEDIFF(expr1, expr2)
 TIMEDIFF(expr1, expr2)
	Returns *expr1* - *expr2* in number of days or time
	 values, given *expr1* and *expr2* are date, time, or datetime values
		`SELECT DATEDIFF('2013-03-10', '2013-03-04');`
		 returns 6
		 `SELECT TIMEDIFF('10:00:00', '09:45:30');`
		 returns 00:14:30

PARTICIPATION ACTIVITY 9.1.3: Select movies with date functions.

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

Using the YEAR() and MONTH() functions, modify the SELECT statement to select movies that are released after 2017 or in November.

Run your solution and verify the result table shows just the movies with IDs 5, 6, 7, and 9.

```
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)
);
```

```sql
INSERT INTO Movie (Title, Rating, ReleaseDate) VALUES
  ('Rogue One: A Star Wars Story', 'PG-13', '2016-12-16'),
  ('Casablanca', 'PG', '1943-01-23'),
  ('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'),
  ('Bridget Jones\'s Diary', 'PG-13', '2001-04-13'),
  ('Avengers: Endgame', 'PG-13', '2019-04-26');

-- Modify the SELECT statement:
SELECT *
FROM Movie;
```

Run Reset code

▶ View solution

PARTICIPATION ACTIVITY 9.1.4: Date and time functions.

The Assignment table below stores the assigned and due dates/times for various homework assignments. Match the value to the SELECT statement that produces the value.

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

```sql
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

CHALLENGE ACTIVITY 9.1.1: Simple functions.

544874.3500394.qx3zqy7

Start

The Book table has the following columns:

ID - INT
X - INT
Y - INT

Complete the SELECT statement to compute (column X plus a random number), rounded to 2 decimal places, for all rows. The random number should be between 0 (inclusive) and 1 (exclusive).

```
SELECT /* Type your code here */
FROM Book;
```

1 2 3

Check Next

Exploring further:

- String functions from MySQL.com
- Numeric functions from MySQL.com
- Date and time functions from MySQL.com

9.2 Aggregate functions

Aggregate functions

An ***aggregate function*** processes values from a set of rows and returns a summary value. Common aggregate functions are:

- ***COUNT()*** counts the number of rows in the set.
- ***MIN()*** finds the minimum value in the set.
- ***MAX()*** finds the maximum value in the set.
- ***SUM()*** sums all the values in the set.
- ***AVG()*** computes the arithmetic mean of all the values in the set.

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.

PARTICIPATION ACTIVITY 9.2.1: Using aggregate functions in a SELECT statement.

Employee

ID	Name	Salary	Bonus
2538	Lisa Ellison	45000	0
5384	Sam Snead	32000	3000
6381	Maria Rodriguez	95000	1000

```
SELECT COUNT(*)
FROM Employee
WHERE Bonus > 500;
```

COUNT(*)
2

```
SELECT MIN(Salary)
FROM Employee;
```

MIN(Salary)
32000

```
SELECT AVG(Salary)
FROM Employee;
```

AVG(Salary)
57333.333333

Animation content:

Step 1: COUNT() counts how many rows are selected. Two employees have Bonus > 500. 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.

Step 2: MIN() finds the smallest value in the Salary column, which is 32000. 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.

Step 3: AVG() finds the average of the Salary column, which is 57333.333333. Two new lines of code appear. SELECT AVG left parenthesis Salary right parenthesis. The second line of code states FROM Employee semicolon. All three values in column Salary are highlighted and the value 57333.333333 appears below the table. A new table appears and has one column named AVG left parenthesis Salary right parenthesis. The value 57333.333333 is added to this column.

Animation captions:

1. COUNT() counts how many rows are selected. Two employees have Bonus > 500.
2. MIN() finds the smallest value in the Salary column, which is 32000.

2. MIN() finds the smallest value in the Salary column, which is 52000.
3. AVG() finds the average of the Salary column, which is 57333.333333.

PARTICIPATION ACTIVITY 9.2.2: 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 consists of the GROUP BY keyword and one or more columns. Each simple or composite value of the column(s) becomes a group. The query computes the aggregate function separately, and returns one row, for each group.

The GROUP BY clause appears between the WHERE clause, if any, and the ORDER BY clause.

In general, the SELECT clause may contain only the aggregate function and column(s) that appear in the GROUP BY clause. However, MySQL supports limited exceptions to this rule. Refer to MySQL Handling of GROUP BY for details.

PARTICIPATION ACTIVITY 9.2.3: 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:

Step 1: The SUM() function sums the Population values in each group. There is a table named City with five columns. ID Name CountryCode District and Population. The SQL code states. Select countrycode comma sum of population. From city. Group by countrycode semicolon. The sum of population is boxed and the population column in table city is highlighted.

Step 2: The GROUP BY clause forms groups based on the CountryCode column. The group by statement is boxed and the countrycode column in table city is highlighted.

Step 3: CountryCode contains two unique values: ZMB and ZWE. So two rows are returned with the total population of each CountryCode value. CountryCode contains two unique values: ZMB and ZWE. So two rows are returned with the total population of each CountryCode value. The first five rows of column countrycode are highlighted and contain the value ZMB. The values in column population of the five highlighted rows are summed together to 2231500. The remaining three rows of column countrycode are highlighted and contain the value ZWE. The values in column population of the three highlighted rows are summed together to 2306654. The return values are column countrycode ZMB and ZWE and sum of population 2231500 and 2306654.

Step 4: The COUNT() function counts how many rows exist in each group. New SQL code appears and states. Select countrycode comma distract comma count of all. From city. Group by countrycode comma district semicolon. Count of all is boxed.

Step 5: The groups are formed by the CountryCode and District columns. The group by statement is boxed and columns countrycode and district are highlighted.

Step 6: Each unique CountryCode and District combination is counted. The first row of columns countrycode and district is highlighted and contains the values ZMB and 1. The count is 1. The second third and fifth rows of columns countrycode and district are highlighted and contain the values ZMB and 2. The count is 3. The fourth row of columns countrycode and district is highlighted and contains the values ZMB and 3. The count is 1. The sixth and eighth rows of columns countrycode and district are highlighted and contain the values ZWE and 1. The count is 2. The seventh row of columns countrycode and district are highlighted and contains the values ZWE and 1. The count is 1. The return values are columns countrycode 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

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 9.2.4: 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);
```

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.

PARTICIPATION ACTIVITY | 9.2.5: 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:

Step 1: The HAVING clause follows the GROUP BY clause. There is a table named city with five columns. ID Name CountrCode District and Population. The SQL code states. Select countrycode comma sum of population. From city. Group by countrycode. Having sum of population greater than 2300000 semicolon. The group by and having statements are boxed.

Step 2: Although the GROUP BY clause creates two groups based on CountryCode, the HAVING clause selects only the group with a population sum greater than 2,300,000. The first five rows in

clause selects only the group with a population sum greater than 2,300,000. The first five rows in column countrycode are highlighted and all contain the value ZMB. The values in column population of the five highlighted rows are summed together to the value 2231500. The remaining three rows in column countrycode are highlighted and all contain the value ZWE. The values in column population of the three highlighted rows are summed together to the value 2306654. The returned values are countrycode ZWE and sum of population 2306654.

Step 3: The HAVING clause selects only groups that have a row count greater than or equal to 2. Only the ZMB, 2 and ZWE, 1 groups have a least 2 rows. New SQL code appears and states. Select countrycode comma district comma count of all. From city. Group by countrycode 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 columns countrycode and district are highlighted and contain the values ZMB and 1. The count equals 1. The second third and fifth rows of columns countrycode and district are highlighted and contain the values ZMB and 2. The count equals 3. The fourth row of column countrycode 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 countrycode 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 9.2.6: Find most recent release year for each genre.

The given SQL creates a Song table and inserts some 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 country pop, R&B, and grunge genres, row counts, and most recent release years appear in the result table.

```
1 CREATE TABLE Song (
2   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'
  (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

PARTICIPATION ACTIVITY 9.2.7: 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;
  ```

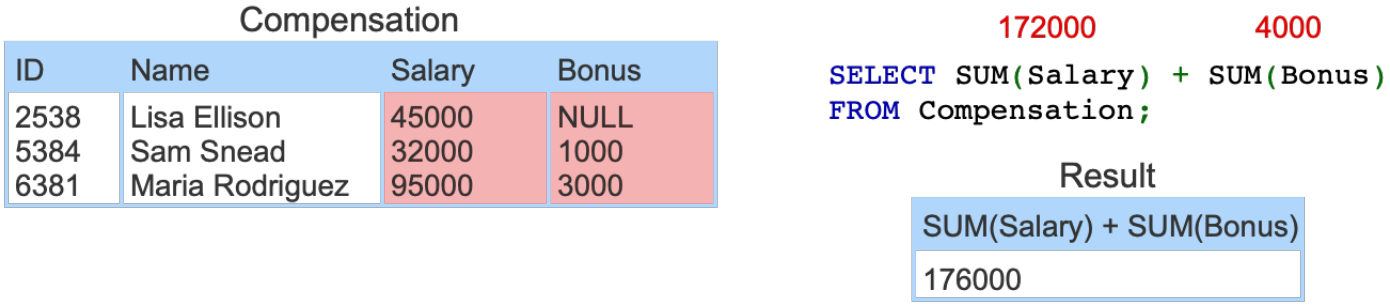
- ```
  SELECT Year, Type,
  MAX(Price)
  FROM Auto
  GROUP BY Year, Type
  HAVING MAX(Price) >
  15000
  ORDER BY Year,
  MAX(Price);
  ```

Aggregate functions and NULL values

Aggregate functions ignore NULL values. Ex: `SUM(Salary)` adds all non-NULL salaries and ignores rows containing a NULL salary.

Aggregate functions and arithmetic operators handle NULL differently. Arithmetic operators return NULL when either operand is NULL. As a result, aggregate functions may generate surprising results when NULL is present. Ex: In the animations below, `SUM(Salary) + SUM(Bonus)` is not equal to `SUM(Salary + Bonus)`.

PARTICIPATION ACTIVITY | 9.2.8: Aggregate functions ignore NULL values.



Animation content:

Static figure:
The Compensation table has columns ID, Name, Salary, and Bonus. Compensation has three rows:
2538, Lisa Ellison, 45000, NULL
5384, Sam Snead, 32000, 1000
6381, Maria Rodriguez, 95000, 3000

An SQL statement appears:
Begin SQL code:
SELECT SUM(Salary) + SUM(Bonus)
FROM Compensation;
End SQL code.
The caption 172000 appears above SUM(Salary). The caption 4000 appears above SUM(Bonus).

The Result table has one column named SUM(Salary) + SUM(Bonus). Result has one row:
176000

Step 1: The SELECT statement has no WHERE clause, so all rows are selected. The Compensation table and the statement appear. All rows of Compensation are highlighted.

Step 2: SUM(Salary) returns 172000. SUM(Salary) is highlighted. The values in Salary are highlighted. The caption 172000 appears above SUM(Salary).

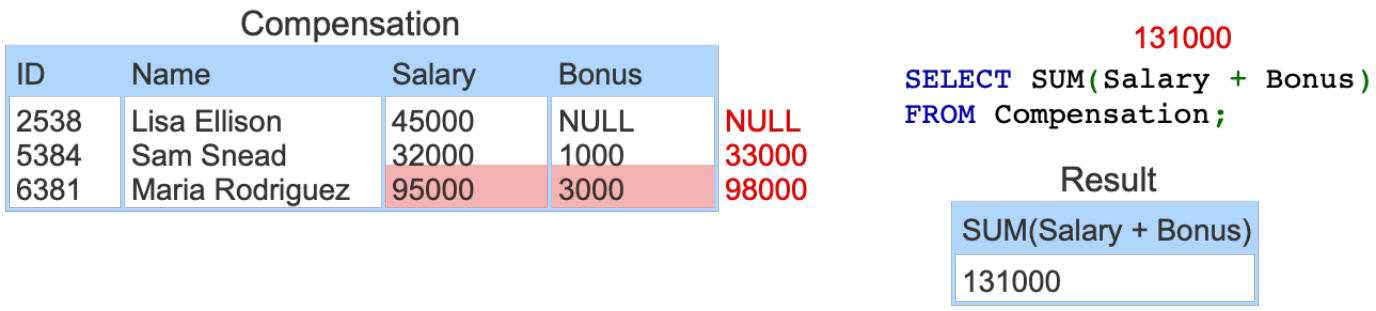
Step 3: SUM(Bonus) ignores NULL and returns 4000. The SELECT statement returns 172000 + 4000. SUM(Bonus) is highlighted. The values in Bonus are highlighted. The caption 4000 appears above SUM(Bonus).

Step 4: The SELECT statement returns 172000 + 4000. The Result table appears.

Animation captions:

1. The SELECT statement has no WHERE clause, so all rows are selected.
2. SUM(Salary) returns 172000.
3. SUM(Bonus) ignores NULL and returns 4000.
4. The SELECT statement returns 172000 + 4000.

PARTICIPATION ACTIVITY | 9.2.9: SUM(Salary) + SUM(Bonus) is not the same as SUM(Salary + Bonus).



Animation content:

Static figure:
The Compensation table has columns ID, Name, Salary, and Bonus. Compensation has three rows:
2538, Lisa Ellison, 45000, NULL
5384, Sam Snead, 32000, 1000
6381, Maria Rodriguez, 95000, 3000
The caption NULL appears next to row one. The caption 33000 appears next to row two. The caption 98000 appears next to row three.

An SQL statement appears:

An SQL statement appears:
Begin SQL code:
SELECT SUM(Salary + Bonus)
FROM Compensation;
End SQL code.
The caption 131000 appears above SUM(Salary + Bonus).

The Result table has one column named SUM(Salary + Bonus). Result has one row:
131000

Step 1: 45000 + NULL is NULL. The values in Salary and Bonus of row one are highlighted. NULL appears next to row one.

Step 2: Salary + Bonus is computed for the remaining rows. The values in Salary and Bonus of rows two and three are highlighted. 33000 appears next to row two. 98000 appears next to row three.

Step 3: SUM() ignores NULL and returns 33000 + 98000. 131000 appears above SUM(Salary + Bonus). The Result table appears.

Animation captions:

1. 45000 + NULL is NULL.
2. Salary + Bonus is computed for the remaining rows.
3. SUM() ignores NULL and returns 33000 + 98000.

PARTICIPATION ACTIVITY 9.2.10: Aggregate functions and NULL values.

Refer to the Compensation table below.

Compensation

ID	Name	Salary	Bonus
2538	Lisa Ellison	115000	NULL
5348	Sam Snead	35000	55000
6381	Maria Rodriguez	95000	3000
8820	Jiho Chen	NULL	48000

What is the result of the following statements?

1)
```
SELECT MAX(Bonus)
```

```
FROM Compensation;
```

Check Show answer

2)

```
SELECT MAX(Salary +
Bonus)
FROM Compensation;
```

Check Show answer

CHALLENGE ACTIVITY 9.2.1: Aggregate functions.

544874.3500394.qx3zqy7

Start

Country

ID	Name	Capital	PopulationMillions
430	Liberia	Monrovia	4.82
800	Uganda	Kampala	42.72
364	Iran	Tehran	81.8
192	Cuba	Havana	11.34

Complete the SELECT statement that finds the total PopulationMillions.

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

(A) /* Type your code here */

(B)

1 2 3 4 5

Check Next

9.3 Subqueries

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 9.3.1: 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');
```

subc

CountryCode	Language
ALB	Albanian
AND	Catalan

Animation content:

Static figure:
The CountryLanguage table has columns CountryCode, Language, IsOfficial, and Percentage. CountryLanguage has five rows:
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

An SQL statement appears below the table:
Begin SQL code:

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

End SQL code.
The code within parentheses is labeled "subquery".

A result table appears below the statement, with columns Language and Percentage. The result has three rows:
Kongo, 13.2
Albanian, 97.9
Catalan, 32.3

The Country table appears to the right of CountryLanguage, with columns Code, Name, and Continent. Country has five rows:
ABW, Aruba, North America
AFG, Afghanistan, Asia
AGO, Angola, Africa
ALB, Albania, Europe
AND, Andorra, Europe

An SQL statement appears below Country:
Begin SQL code:

```
SELECT CountryCode, Language
FROM CountryLanguage
WHERE CountryCode IN
```

WHERE CountryCode IN
(SELECT Code
FROM Country
WHERE Continent = 'Europe');
End SQL code.
The code within parentheses is labeled "subquery".

A result table appears below the second SQL statement, with columns CountryCode and Language. The result has two rows:
ALB, Albanian
AND, Catalan

Step 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. Both tables and the first statement appear.

Step 2: The subquery executes first to find the official language Percentage for ABW, which is 5.3. The subquery of the first statement is highlighted. The first row of Country is highlighted. The value 5.3 appears next to the clause WHERE Percentage greater than.

Step 3: The outer query executes using the value 5.3 returned by the subquery. Three languages have Percentage > 5.3. The outer query is highlighted. Rows three through five of CountryLanguage are highlighted. The values of Language and Percentage for these rows appear in the result table.

Step 4: The SELECT statement uses the IN operator with a subquery to determine which Languages are used in Europe. The second statement appears.

Step 5: The subquery first finds all Codes from Europe: ALB and AND. The subquery of the second statement is highlighted. The last two rows of Country are highlighted. (ALB, AND) appears next to the clause WHERE CountryCode IN.

Step 6: The outer query then selects the CountryCode and Language for the CountryCodes ALB and AND. The outer query of the second statement is highlighted. The last two rows of Country are highlighted. The values of CountryCode and Language for these rows appear in the result 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

5. 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 9.3.2: 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'
  (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;
```

```
ORDER BY ReleaseYear;

SELECT ReleaseYear
FROM Song
WHERE ID = 800;

-- Write your SELECT statement here:

```

Run Reset code

▶ View solution

PARTICIPATION ACTIVITY 9.3.3: Subqueries.

Refer to the CountryLanguage and Country tables below.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
ABW	Papeiamento	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');
```

- ○ Papeiamento
- ○ Mbundu
- ○ Dutch, Balochi, Kongo

2) What does the query return?

```
SELECT Language
FROM CountryLanguage
WHERE Percentage <
```

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

- Papeiamento
- 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, Papeiamento, 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'
```

Correlated subqueries

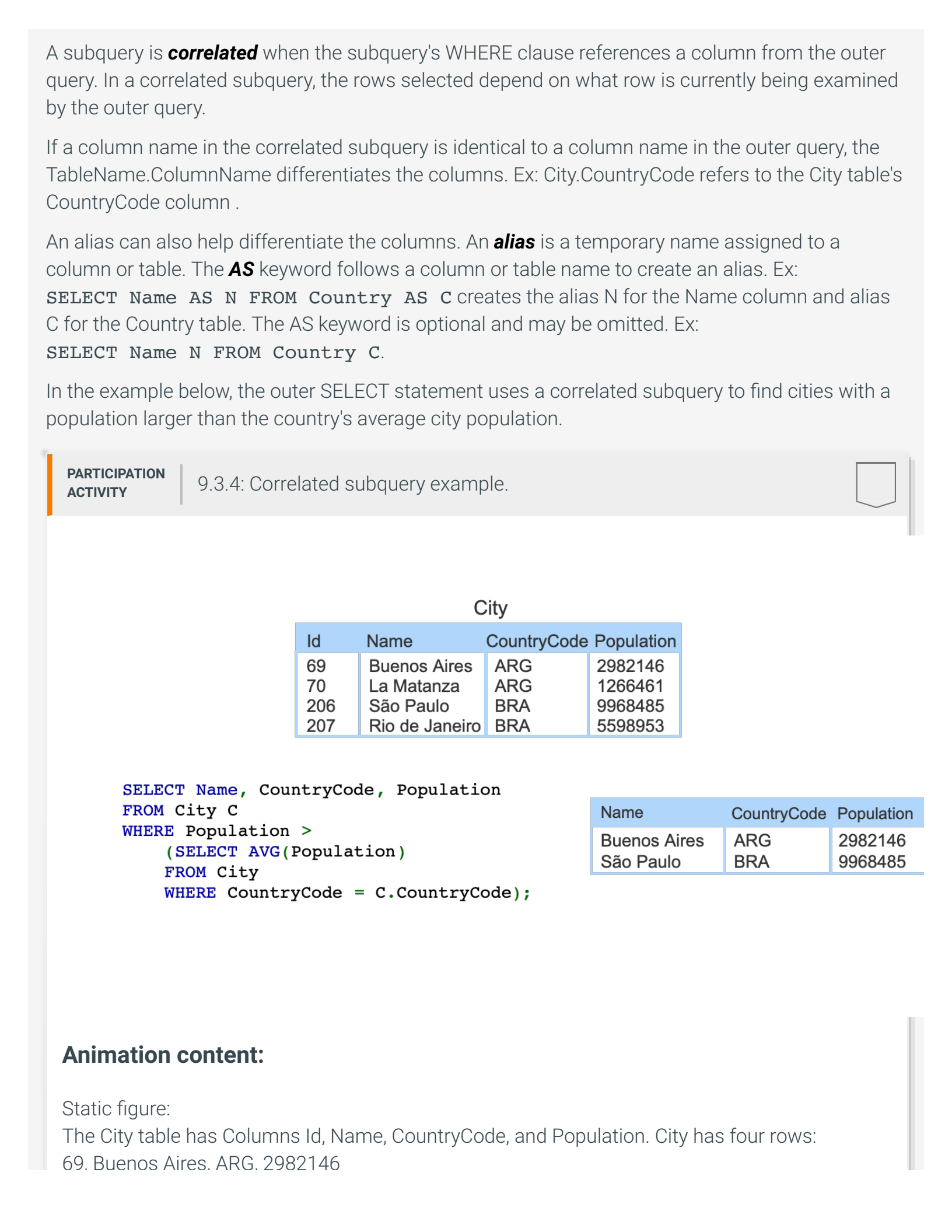
A subquery is ***correlated*** when the subquery's WHERE clause references a column from the outer query. In a correlated subquery, the rows selected depend on what row is currently being examined by the outer query.

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*** keyword follows a column or table name to create an alias. Ex: `SELECT Name AS N FROM Country AS C` creates the alias N for the Name column and alias C for the Country table. The AS keyword is optional and may be 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 9.3.4: 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:

Static figure:
The City table has Columns Id, Name, CountryCode, and Population. City has four rows:
69. Buenos Aires. ARG. 2982146

70, Latanza, ARG, 1266461
206, Sao Paulo, BRA, 9968485
207, Rio de Janeiro, BRA, 5598953

An SQL statement appears below the table:
Begin SQL code:

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

End SQL code.

The result table has columns Name, CountryCode, and Population. The result has two rows:
Buenos Aires, ARG, 2982146
Sao Paulo, BRA, 9968485

Step 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. In the first FROM clause, City C is highlighted. In the second WHERE clause, C.CountryCode is highlighted.

Step 2: The outer query selects rows from the City table. As each City row is selected, the subquery finds the average population for the city's country. The outer query and first row of City are highlighted. The subquery is highlighted and ARG appears below C.CountryCode. The first two rows are highlighted. Avg(Population) = 2124303.5 appears next to the first two rows. 2124303.5 appears next to WHERE Population >.

Step 3: Then the outer query executes using the average population returned from the subquery. Buenos Aires has a population 2982146 greater than 2124303.5. The outer query is highlighted. The first row of City is highlighted and appears in the result table.

Step 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 greater than 2124303.5. The subquery and second row of City are highlighted. The animation of steps 2 and 3 repeat. The outer query retrieves the same row as in step 3, so the result table does not change.

Step 5: The outer query finds São Paulo also has a population greater than BRA's average population. The animation repeats for the third row of City. Avg(Population) = 7783719 appears next to the third and fourth row of City. 7783719 appears next to WHERE Population greater than. The third row is highlighted and added to the result table.

The third row is highlighted and added to the result table.

Step 6: Rio de Janeiro is not selected because Rio de Janeiro's population 5598953 is not greater than 7783719. The fourth row is highlighted but not added to the result 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 selects rows from the City table. As each City row is selected, 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 9.3.5: Correlated subqueries.

Refer to the CountryLanguage and City tables below.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
ABW	Papeiamento	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	Qandahar	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 9.3.6: 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	Populatior
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:

Static figure:
The CountryLanguage table has columns CountryCode, Language, IsOfficial, and Percentage. CountryLanguage has four rows:
ARG, Italian, F, 1.7
ARG, Spanish, T, 96.8
BRA, German, F, 0.5
BRA, Portuguese, T, 97.5

The City table has columns Id, Name, CountryCode, and Population. City has four rows:
69, Buenos Aires, ARG, 2982146
70, Latanza, ARG, 1266461
206, Sao Paulo, BRA, 9968485
207, Rio de Janeiro, BRA, 5598953

Begin SQL code:

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

End SQL code.

A result table has columns Name and CountryCode. The result table has two rows:
Sao Paulo, BRA
Rio de Janeiro, BRA

Step 1: The query selects cities in countries where at least one language is spoken by more than 97 percent of the population. The EXISTS keyword is highlighted. The subquery is highlighted.

Step 2: The subquery selects no rows because no ARG percentage is greater than 97. So the EXISTS clause is false and the outer query does not select Buenos Aires. The first row of City is highlighted. The first two rows of CountryLanguage, with CountryCode of ARG, are highlighted. FALSE appears next to EXISTS.

Step 3: Since the EXISTS clause is false for ARG, no cities in Argentina are selected. The second row of City is highlighted. The first two rows of CountryLanguage, with CountryCode of ARG, remain highlighted. FALSE remains next to EXISTS.

Step 4: The subquery selects one row because one BRA percentage is greater than 97. So the EXISTS clause is true and the outer query selects Sao Paulo. The third row of City is highlighted. The last two rows of CountryLanguage, with CountryCode of BRA, are highlighted. The value 97.5 in the last row of CountryLanguage is highlighted. TRUE appears next to EXISTS. The result table appears with one row:
Sao Paolo, BRA

Step 5: Since the EXISTS clause is true for BRA, all cities in Brazil are selected. The fourth row of City is highlighted. The last two rows of CountryLanguage, with CountryCode of BRA, remain highlighted. TRUE remains next to EXISTS. A second row is added to the result table:
Rio de Janeiro, BRA

Animation captions:

1. The query selects cities in countries where at least one language is spoken by more than 97% of the population.
2. The subquery selects no rows because no ARG percentage is > 97. So the EXISTS clause is false and the outer query does not select Buenos Aires.
3. Since the EXISTS clause is false for ARG, no cities in Argentina are selected.
4. The subquery selects one row because one BRA percentage is > 97. So the EXISTS clause is true and the outer query selects Sao Paulo.
5. Since the EXISTS clause is true for BRA, all cities in Brazil are selected.

PARTICIPATION ACTIVITY | 9.3.7: Select albums with EXISTS.

The given SQL creates an Album and Song tables and inserts some 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*.

```
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),
  (400, 'Rolling in the Deep', 'Adele', 4),
  (401, 'Someone Like You', 'Adele', 4),
  (402, 'Set Fire to the Rain', 'Adele', 4),
  (403, 'Rumor Has It', 'Adele', 4);

SELECT *
```

Run Reset code

▶ View solution

PARTICIPATION ACTIVITY 9.3.8: EXISTS operator in subqueries.

Refer to the Employee and Family tables below.

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) 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'
```

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.

After this first pass, test the flattened query and adjust to achieve the correct result. Verify that the original and flattened queries are equivalent against a variety of data.

PARTICIPATION ACTIVITY 9.3.9: 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:

Static figure:
The Country table has columns Code, Name, and Continent, with four rows:
AUS, Australia, Australia
SAF, South Africa, Africa
SPA, Spain, Europe
USA, United States, North America

The City has columns Id, Name, CountryCode, and Population, with six rows:
144, Salzburg, AUS, 152367
284, Cape Town, SAF, 4681000
471, Durban, SAF, 3442361
650, Barcelona, SPA, 1620000
938, Madrid, SPA, 3233000
942, Denver, USA, 705576

An SQL statement appears:
Begin SQL code:
SELECT Name
FROM Country
WHERE Code IN
  (SELECT CountryCode
  FROM City
  WHERE Population > 1000000);
End SQL code.

A result table appears below the statement, with column Name and two rows:
South Africa
Spain

A second SQL statement appears:
Begin SQL code:
SELECT DISTINCT Name
FROM Country
INNER JOIN City ON Code = CountryCode
WHERE Population > 1000000;
End SQL code.

A second result table appears below the second statement, with column Name and two rows:
South Africa
Spain

Step 1: The subquery selects country codes for cities with population greater than 1,000,000. The subquery in the first statement is highlighted. Rows two through five of City, with values SAF or SPA in CountryCode, are highlighted. (SAF, SPA) appears next to the IN keyword.

Step 2: The outer query selects the country names. The outer query is highlighted. Rows two and three of Country, with values SAF and SPA in Code, are highlighted. The result table appears with rows:
South Africa

South Africa
Spain

Step 3: To flatten the query, the subquery is replaced with an INNER JOIN clause. The second statement appears without the DISTINCT keyword.

Step 4: The join query selects the one country name for each city with population greater than 1,000,000. Rows two and three of City, with value SAF in CountryCode, are highlighted. Row two of Country, with value SAF in Code, is highlighted. Two rows are added to the second result table:
South Africa
South Africa
Rows three and four of City, with value SPA in CountryCode, are highlighted. Row three of Country, with value SPA in Code, is highlighted. Two more rows are added to the second result table:
Spain
Spain

Step 5: The DISTINCT clause eliminates duplicate rows. The subquery and join query are equivalent. DISTINCT is inserted in the SELECT clause of the second statement. Duplicate rows are removed from the second result table.

Animation captions:

1. The subquery selects country codes for cities with population > 1,000,000.
2. The outer query selects the country names.
3. To flatten the query, the subquery is replaced with an INNER JOIN clause.
4. The join query selects the one country name for each city with population > 1,000,000.
5. The DISTINCT clause eliminates duplicate rows. The subquery and join query are equivalent.

PARTICIPATION ACTIVITY 9.3.10: 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 E.Name
FROM Employee E
WHERE EXISTS
  (SELECT *
   FROM Family F
   WHERE F.Id = E.Id AND
Relationship = 'Spouse');

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

- ○ Same result
- ○ Different result

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

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

- ○ Same result
- ○ Different result

3)
```
SELECT E.Name
FROM Employee E
WHERE EXISTS
  (SELECT *
   FROM Family F
   WHERE F.Id = E.Id AND
Relationship = 'Daughter');

SELECT DISTINCT E.Name
FROM Employee E
INNER JOIN Family F ON F.Id =
E.Id
WHERE Relationship =
'Daughter';
```

- ○ Same result

Different result

4)

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

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

Same result

Different result

CHALLENGE ACTIVITY 9.3.1: Subqueries.

544874.3500394.qx3zqy7

Start

Course

CourseId	CourseCode	CourseName	Capacity	Instru
6842	HIST64	American History	25	3
7558	PHIL64	American Philosophy	150	2
7767	BIOL848	Cell Biology	100	1
2946	BIOL477	Genetics	200	1
8183	HIST397	World History	75	3

Instructor

InstructorId	InstructorName	Rank	Department
1	Aya Chen	Associate Professor	Biology
2	Ken Sanz	Assistant Professor	Philosophy
3	Ben Cruz	Professor	History

Note: Both tables may not be necessary to complete this level.

Select the values returned by the query below.

```
SELECT CourseCode
FROM Course
```

PHIL64

```
WHERE Capacity <=
   (SELECT Capacity
   FROM Course
   WHERE CourseName = 'Cell Biology');
```

- [ ] BIOL477
- [ ] HIST64

1 2 3 4 5

Check Next

Exploring further:

- Subqueries from MySQL.com
- Flattening queries in the Apache Derby database

9.4 Complex query example

Writing a complex query

Database users frequently create complex SQL queries that join data from multiple tables to answer business questions. Ex: A bookstore might ask, "Which books are selling best in summer?" and "What types of books do customers from the West Coast purchase?"

To create a complex query, a database user can employ the following strategy:

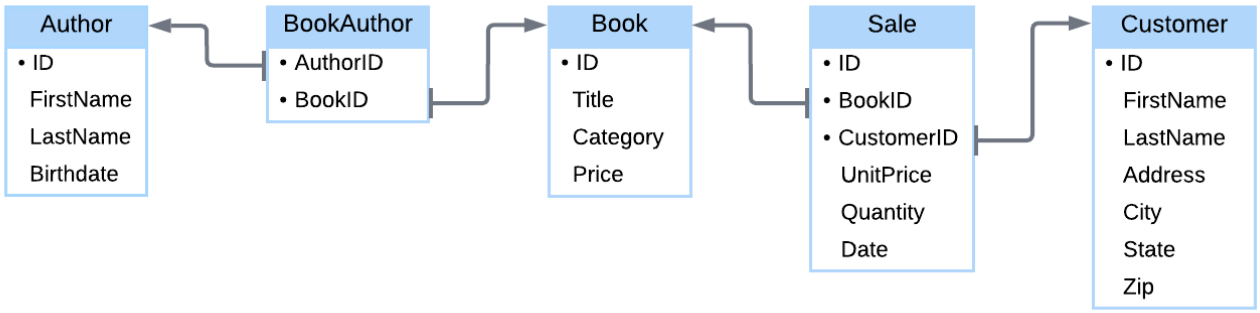
1. Examine a table diagram or other database summary to understand the tables and relationships.
2. Identify the tables containing the necessary data to answer the question.
3. Determine which columns should appear in the result table.
4. Write a query that joins the tables using the table's primary and foreign keys.
5. Break the problem into simple queries, writing one part of the query at a time.

The Zion Bookstore wants to know which books, written by a single author, generated the most sales to customers from Colorado or Oklahoma in February 2020. The information required to

answer this question is spread across several tables, requiring a complex query to answer the question.

The table diagram in the figure below describes the Zion Bookstore database, which tracks books, customers, and sales.

Figure 9.4.1: Table diagram for Zion Bookstore database.



PARTICIPATION ACTIVITY 9.4.1: Tables containing data.

Indicate if the given table contains data relevant to Zion Bookstore's question:

Which books, written by a single author, generated the most sales to customers from Colorado or Oklahoma in February 2020?

1) Author

- True
- False

2) BookAuthor

- True
- False

3) Book

- True
- False

4) Sale

- True

- False

5) Customer

- True
- False

Joining tables

To answer the Zion Bookstore question, the result table must contain columns from the previously identified tables or columns that can be computed from the tables. The result table should contain the following: Customer state (Customer.State), Book ID (Sale.BookID), book title (Book.Title), number of books purchased (Sale.Quantity), and total price (Sale.Quantity × Sale.UnitPrice).

PARTICIPATION ACTIVITY 9.4.2: Join the tables.

Tables for the Zion Bookstore database are created and populated below. Run a query that joins the Sale, Customer, and Book tables:

```
SELECT S.CustID, C.State, S.BookID, B.Title, S.Quantity, S.UnitPrice *
S.Quantity
FROM Sale S
INNER JOIN Customer C ON C.ID = S.CustID
INNER JOIN Book AS B ON B.ID = S.BookID;
```

```
CREATE TABLE Author (
  ID INT NOT NULL,
  FirstName VARCHAR(45) DEFAULT NULL,
  LastName VARCHAR(45) DEFAULT NULL,
  BirthDate DATE DEFAULT NULL,
  PRIMARY KEY (ID)
);

INSERT INTO Author VALUES
(1,'Jennifer','McCoy', '1980-05-01'),
(2,'Yuto','Takahashi', '1973-12-04'),
(3,'Jose','Martinez', NULL),
(4,'Jasmine','Baxter', NULL),
(5,'Xiu','Tao', '1992-11-13'),
(6,'Ethan','Lonestar', '1965-02-15'),
(7,'Amar','Agarwal', NULL),
(8,'Emilia','Russo', '1999-08-03');

CREATE TABLE Book (
```

```
  ID INT NOT NULL,
  Title VARCHAR(200) DEFAULT NULL,
  Publisher VARCHAR(45) DEFAULT NULL,
  Category VARCHAR(20) CHECK (Category IN ('adventure','drama','fanta
  Price DECIMAL(6,2) DEFAULT NULL,
  PRIMARY KEY (ID)
);

INSERT INTO Book VALUES
(100,'The Black Box','Wright Pub','adventure',22.50),
(101,'Lost Time','Caster','scifi',19.99),
(102,'Which Way Home?','Light House','humor',8.99),
(103,'Grant Me Three Wishes','Caster','romance',10.75),
(104,'The Last Attempt','Longshot','scifi',15.99),
(105,'My Crazy Life','Light House','humor',9.67);

```

Run Reset code

PARTICIPATION ACTIVITY 9.4.3: Joining tables.

1) A customer from which state purchased the most copies of *Grant Me Three Wishes*?

- CO
- NM
- OK

2) Does *Lost Time* always sell for the same price?

- Yes
- No

Grouping by state and book

The result table needs to show the total sales per book and per state.

PARTICIPATION ACTIVITY 9.4.4: Group by state and book.

Use a modified query that sums the quantity and sales price for each book using the

SUM() function. The GROUP BY clause groups the sums together for each book ID and state. The ORDER BY clause sorts the total sales in descending order.

```
SELECT C.State, S.BookID, B.Title, SUM(S.Quantity) AS Quantity,
SUM(S.UnitPrice * S.Quantity) AS TotalSales
FROM Sale S
INNER JOIN Customer C ON C.ID = S.CustID
INNER JOIN Book AS B ON B.ID = S.BookID
GROUP BY C.State, S.BookID
ORDER BY TotalSales DESC;
```

```
CREATE TABLE Author (
  ID INT NOT NULL,
  FirstName VARCHAR(45) DEFAULT NULL,
  LastName VARCHAR(45) DEFAULT NULL,
  BirthDate DATE DEFAULT NULL,
  PRIMARY KEY (ID)
);

INSERT INTO Author VALUES
(1,'Jennifer','McCoy', '1980-05-01'),
(2,'Yuto','Takahashi', '1973-12-04'),
(3,'Jose','Martinez', NULL),
(4,'Jasmine','Baxter', NULL),
(5,'Xiu','Tao', '1992-11-13'),
(6,'Ethan','Lonestar', '1965-02-15'),
(7,'Amar','Agarwal', NULL),
(8,'Emilia','Russo', '1999-08-03');

CREATE TABLE Book (
  ID INT NOT NULL,
  Title VARCHAR(200) DEFAULT NULL,
  Publisher VARCHAR(45) DEFAULT NULL,
  Category VARCHAR(20) CHECK (Category IN ('adventure','drama','fanta
  Price DECIMAL(6,2) DEFAULT NULL,
  PRIMARY KEY (ID)
);

INSERT INTO Book VALUES
(100,'The Black Box','Wright Pub','adventure',22.50),
(101,'Lost Time','Caster','scifi',19.99),
(102,'Which Way Home?','Light House','humor',8.99),
(103,'Grant Me Three Wishes','Caster','romance',10.75),
(104,'The Last Attempt','Longshot','scifi',15.99),
(105,'My Crazy Life','Light House','humor',9.67);

```

Run Reset code

PARTICIPATION ACTIVITY 9.4.5: Grouping by state and book.

1) Which book has sold the most copies in a single state?
   - 100 - *The Black Box*
   - 103 - *Grant Me Three Wishes*
   - 104 - *The Last Attempt*

2) Which alteration of the GROUP BY clause merges all states' quantities together to reveal the total number of copies each book has sold?
   - GROUP BY Quantity
   - GROUP BY C.State
   - GROUP BY S.BookID

Filtering states and dates

The result table should only show results from Colorado and Oklahoma. Only purchases made during February 2020 should be considered. All the filtering criteria must be specified in the query's WHERE clause.

PARTICIPATION ACTIVITY 9.4.6: Filtering states and dates.

Add a WHERE clause to restrict the result table to sales from Colorado and Oklahoma only. Use the MONTH() and YEAR() functions to select only sales in month 2 and year 2020.

```
SELECT C.State, S.BookID, B.Title, SUM(S.Quantity) AS Quantity,
SUM(S.UnitPrice * S.Quantity) AS TotalSales
FROM Sale S
INNER JOIN Customer C ON C.ID = S.CustID
INNER JOIN Book AS B ON B.ID = S.BookID
WHERE (C.State = 'CO' OR C.State = 'OK') AND MONTH(S.Date) = 2 AND
YEAR(S.Date) = 2020
GROUP BY C.State, S.BookID
ORDER BY TotalSales DESC;
```

```
1 CREATE TABLE Author (
2   ID INT NOT NULL,
```

```
  FirstName VARCHAR(45) DEFAULT NULL,
  LastName VARCHAR(45) DEFAULT NULL,
  BirthDate DATE DEFAULT NULL,
  PRIMARY KEY (ID)
);

INSERT INTO Author VALUES
(1,'Jennifer','McCoy', '1980-05-01'),
(2,'Yuto','Takahashi', '1973-12-04'),
(3,'Jose','Martinez', NULL),
(4,'Jasmine','Baxter', NULL),
(5,'Xiu','Tao', '1992-11-13'),
(6,'Ethan','Lonestar', '1965-02-15'),
(7,'Amar','Agarwal', NULL),
(8,'Emilia','Russo', '1999-08-03');

CREATE TABLE Book (
  ID INT NOT NULL,
  Title VARCHAR(200) DEFAULT NULL,
  Publisher VARCHAR(45) DEFAULT NULL,
  Category VARCHAR(20) CHECK (Category IN ('adventure','drama','fanta
  Price DECIMAL(6,2) DEFAULT NULL,
  PRIMARY KEY (ID)
);

INSERT INTO Book VALUES
(100,'The Black Box','Wright Pub','adventure',22.50),
(101,'Lost Time','Caster','scifi',19.99),
(102,'Which Way Home?','Light House','humor',8.99),
(103,'Grant Me Three Wishes','Caster','romance',10.75),
(104,'The Last Attempt','Longshot','scifi',15.99),
(105,'My Crazy Life','Light House','humor',9.67);

```

Run Reset code

PARTICIPATION ACTIVITY 9.4.7: Filtering states and dates.

1) Removing `AND MONTH(S.Date) = 2 AND YEAR(S.Date) = 2020` from the query increases the number of rows in the result table.

- ○ True
- ○ False

2) Removing the parentheses around `C.State = 'CO' OR C.State = 'OK'` changes nothing in the result table.

- True
- False

Books with single author

The result table should contain only books with a single author. A row in BookAuthor assigns one author to one book. Ex: A row with AuthorID = 5 and BookID = 103 means the author 5 wrote book 103. Books with multiple authors have multiple rows, so a subquery that examines the number of rows in BookAuthor for a given book ID can limit the results to just single-author books.

PARTICIPATION ACTIVITY 9.4.8: Books with single author.

Modify the WHERE clause to use a subquery. The subquery uses a HAVING clause with COUNT() to select only book IDs that appear in one row of BookAuthor.

```
SELECT C.State, S.BookID, B.Title, SUM(S.Quantity) AS Quantity,
SUM(S.UnitPrice * S.Quantity) AS TotalSales
FROM Sale S
INNER JOIN Customer C ON C.ID = S.CustID
INNER JOIN Book AS B ON B.ID = S.BookID
WHERE (C.State = 'CO' OR C.State = 'OK') AND MONTH(S.Date) = 2 AND
YEAR(S.Date) = 2020 AND B.ID IN
   (SELECT BookID
   FROM BookAuthor
   GROUP BY BookID
   HAVING COUNT(*) = 1)
GROUP BY C.State, S.BookID
ORDER BY TotalSales DESC;
```

```
CREATE TABLE Author (
  ID INT NOT NULL,
  FirstName VARCHAR(45) DEFAULT NULL,
  LastName VARCHAR(45) DEFAULT NULL,
  BirthDate DATE DEFAULT NULL,
  PRIMARY KEY (ID)
);

INSERT INTO Author VALUES
(1,'Jennifer','McCoy', '1980-05-01'),
(2,'Yuto','Takahashi', '1973-12-04'),
(3,'Jose','Martinez', NULL),
```

```
(4,'Jasmine','Baxter', NULL),
(5,'Xiu','Tao', '1992-11-13'),
(6,'Ethan','Lonestar', '1965-02-15'),
(7,'Amar','Agarwal', NULL),
(8,'Emilia','Russo', '1999-08-03');

CREATE TABLE Book (
  ID INT NOT NULL,
  Title VARCHAR(200) DEFAULT NULL,
  Publisher VARCHAR(45) DEFAULT NULL,
  Category VARCHAR(20) CHECK (Category IN ('adventure','drama','fanta
  Price DECIMAL(6,2) DEFAULT NULL,
  PRIMARY KEY (ID)
);

INSERT INTO Book VALUES
(100,'The Black Box','Wright Pub','adventure',22.50),
(101,'Lost Time','Caster','scifi',19.99),
(102,'Which Way Home?','Light House','humor',8.99),
(103,'Grant Me Three Wishes','Caster','romance',10.75),
(104,'The Last Attempt','Longshot','scifi',15.99),
(105,'My Crazy Life','Light House','humor',9.67);

```

Run Reset code

PARTICIPATION ACTIVITY 9.4.9: Books with single author.

1) Which book with a single author has the largest total sales?

- 101 - *Lost Time*
- 103 - *Grant Me Three Wishes*
- 104 - *The Last Attempt*

2) What effect does changing the subquery to `HAVING COUNT(*) = 2` have on the results table?

- No rows are selected.
- Only books with two authors are selected.
- Only books that sell two copies are selected.

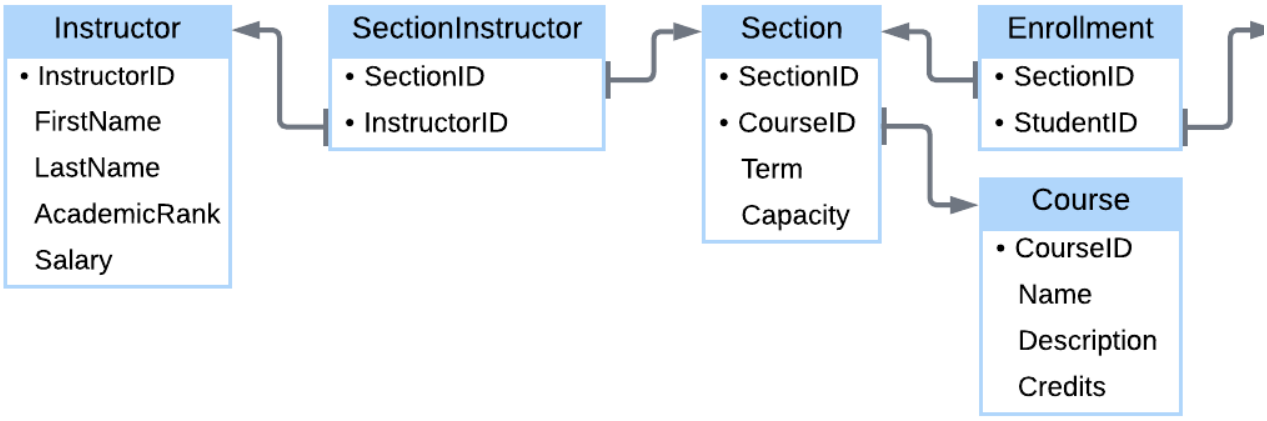
are selected.

CHALLENGE ACTIVITY | 9.4.1: Complex queries.

544874.3500394.qx3zqy7

Start

A university wants to know the names of students who have enrolled in less than 4 sections where all courses taken have fewer credits than the course with the most credits.



What tables contain data relevant to the university's question?

- [ ] Instructor
- [ ] SectionInstructor
- [ ] Section
- [ ] Course
- [ ] Enrollment
- [ ] Student

1 2 3 4

Check Next

9.5 LAB - Select number of movies grouped by

9.5 LAB - Select number of movies grouped by year

The **Movie** table has the following columns:

- **ID** - integer, primary key
- **Title** - variable-length string
- **Genre** - variable-length string
- **RatingCode** - variable-length string
- **Year** - integer

Write a SELECT statement to select the year and the total number of movies for that year.

Hint: Use the COUNT() function and GROUP BY clause.

LAB ACTIVITY | 9.5.1: LAB - Select number of movies grouped by year | 10 / 10

Main.sql — Load default template...

```sql
-- Your SELECT statement goes here
SELECT Year, COUNT(*) AS TotalMovies
FROM Movie
GROUP BY Year;

```


Develop mode | Submit mode

Explore the database and run your program as often as you'd like, before submitting for grading. Click **Run program** and observe the program's output in the second box.

Run program

Main.sql → Output (shown below)

(Your program)

Program output displayed here

Coding trail of your work [What is this?](#)

```
2/27 T10 min:3
```

9.6 LAB - Select lesson schedule with inner join

The database has three tables for tracking horse-riding lessons:

1. **Horse** with columns:
    - ID - primary key
    - RegisteredName
    - Breed
    - Height
    - BirthDate
2. **Student** with columns:
    - ID - primary key
    - FirstName
    - LastName
    - Street
    - City
    - State
    - Zip
    - Phone
    - EmailAddress
3. **LessonSchedule** with columns:
    - HorseID - partial primary key, foreign key references Horse(ID)
    - StudentID - foreign key references Student(ID)
    - LessonDateTime - partial primary key

Write a SELECT statement to create a lesson schedule with the lesson date/time, horse ID, and the

student's first and last names. Order the results in ascending order by lesson date/time, then by horse ID. Unassigned lesson times (student ID is NULL) should not appear in the schedule.

Hint: Perform a join on the Student and LessonSchedule tables, matching the student IDs.

544874.3500394.qx3zqy7

LAB ACTIVITY | 9.6.1: LAB - Select lesson schedule with inner join | 10 / 10

Main.sql — Load default template...

```sql
-- Your SELECT statement goes here
SELECT ls.LessonDateTime, ls.HorseID, s.FirstName, s.LastName
FROM LessonSchedule ls
JOIN Student s ON ls.StudentID = s.ID
WHERE ls.StudentID IS NOT NULL
ORDER BY ls.LessonDateTime ASC, ls.HorseID ASC;

```

Develop mode | Submit mode

Explore the database and run your program as often as you'd like, before submitting for grading. Click **Run program** and observe the program's output in the second box.

Run program

Main.sql (Your program) → Output (shown below)

Program output displayed here

Coding trail of your work [What is this?](#)

```
2/27 T10 min:1
```

9.7 LAB - Select employees and managers with inner join

The **Employee** table has the following columns:

- **ID** - integer, primary key
- **FirstName** - variable-length string
- **LastName** - variable-length string
- **ManagerID** - integer

Write a SELECT statement to show a list of all employees' first names and their managers' first names. List only employees that have a manager. Order the results by Employee first name. Use aliases to give the result columns distinctly different names, like "Employee" and "Manager".

Hint: Join the **Employee** table to itself using INNER JOIN.

544874.3500394.qx3zqy7

LAB ACTIVITY | 9.7.1: LAB - Select employees and managers with inner join | 10 / 10

Main.sql | Load default template...

```
1 -- Your SELECT statement goes here
2 SELECT e.FirstName AS Employee, m.FirstName AS Manager
3 FROM Employee e
4 INNER JOIN Employee m ON e.ManagerID = m.ID
5 ORDER BY e.FirstName;
6
```

Develop mode | Submit mode

Explore the database and run your program as often as

you'd like, before submitting for grading. Click **Run program** and observe the program's output in the second box.

Run program

Main.sql
(Your program) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

```
2/27 T10 min:2
```

9.8 LAB - Select lesson schedule with multiple joins

The database has three tables for tracking horse-riding lessons:

1. **Horse** with columns:
    - ID - primary key
    - RegisteredName
    - Breed
    - Height
    - BirthDate
2. **Student** with columns:
    - ID - primary key
    - FirstName
    - LastName
    - Street
    - City
    - State
    - Zip
    - Phone
    - EmailAddress

3. **LessonSchedule** with columns:
    - HorseID - partial primary key, foreign key references Horse(ID)
    - StudentID - foreign key references Student(ID)
    - LessonDateTime - partial primary key

Write a SELECT statement to create a lesson schedule for Feb 1, 2020 with the lesson date/time, student's first and last names, and the horse's registered name. Order the results in ascending order by lesson date/time, then by the horse's registered name. Make sure unassigned lesson times (student ID is NULL) appear in the results.

Hint: Perform a join on the LessonSchedule, Student, and Horse tables, matching the student IDs and horse IDs.

544874.3500394.qx3zqy7

LAB ACTIVITY | 9.8.1: LAB - Select lesson schedule with multiple joins | 10 / 10

Main.sql — Load default template...

```
1 -- Your SELECT statement goes here
2 SELECT ls.LessonDateTime, s.FirstName AS StudentFirstName, s.LastName AS Stu
3 FROM LessonSchedule ls
4 LEFT JOIN Student s ON ls.StudentID = s.ID
5 LEFT JOIN Horse h ON ls.HorseID = h.ID
6 WHERE DATE(ls.LessonDateTime) = '2020-02-01'
7 ORDER BY ls.LessonDateTime ASC, h.RegisteredName ASC;
8
```

Develop mode | **Submit mode**

Explore the database and run your program as often as you'd like, before submitting for grading. Click **Run program** and observe the program's output in the second box.

Run program

Main.sql (Your program) → Output (shown below)

Program output displayed here

Coding trail of your work [What is this?](#)

```
2/27 T10 min:2
```

9.9 LAB - Select tall horses with subquery

The **Horse** table has the following columns:

- **ID** - integer, primary key
- **RegisteredName** - variable-length string
- **Breed** - variable-length string
- **Height** - decimal number
- **BirthDate** - date

Write a SELECT statement to select the registered name and height for only horses that have an above average height. Order the results by height (ascending).

Hint: Use a subquery to find the average height.

544874.3500394.qx3zqy7

LAB ACTIVITY | 9.9.1: LAB - Select tall horses with subquery | 10 / 10

Main.sql — Load default template...

```sql
-- Your SQL statements go here
SELECT RegisteredName, Height
FROM Horse
WHERE Height > (SELECT AVG(Height) FROM Horse)
ORDER BY Height ASC;

```

Develop mode | Submit mode

Explore the database and run your program as often as you'd like, before submitting for grading. Click **Run program** and observe the program's output in the second box.

Run program

Main.sql
(Your program) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

```
2/27 T10 min:1
```

9.10 LAB - Multiple joins with aggregate (Sakila)

Refer to the `film`, `actor`, and `film_actor` tables of the Sakila database. The tables in this lab have the same columns and data types but fewer rows.

Write a query that:

- Computes the average length of all films that each actor appears in.
- Rounds average length to the nearest minute and renames the result column "average".
- Displays last name, first name, and average, in that order, for each actor.
- Sorts the result in **descending** order by average, then **ascending** order by last name.

The query should exclude films with no actors and actors that do not appear in films.

Hint: Use the ROUND() and AVG() functions.

544874.3500394.qx3zqy7

LAB ACTIVITY | 9.10.1: LAB - Multiple joins with aggregate (Sakila) | 10 / 10

Main.sql — Load default template...

```sql
-- Your SELECT statment goes here
-- Your SELECT statment goes here
SELECT
    a.last_name,
    a.first_name,
    ROUND(AVG(f.length)) AS average
FROM
    actor a
JOIN film_actor fa ON a.actor_id = fa.actor_id
JOIN film f ON fa.film_id = f.film_id
GROUP BY
    a.actor_id
ORDER BY
    average DESC,
    a.last_name ASC;
```

Develop mode | Submit mode

Explore the database and run your program as often as you'd like, before submitting for grading. Click **Run program** and observe the program's output in the second box.

Run program

Main.sql (Your program) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

```
2/27 T0,0,0,0 M10,10 min:5
```

9.11 LAB - Nested aggregates (Sakila)

Refer to the `film` and `inventory` tables of the Sakila database. The tables in this lab have the same columns and data types but fewer rows.

Write a query that lists the titles of films with the fewest rows in the inventory table.

This query requires a subquery that computes the minimum of counts by film_id:

```
SELECT MIN(count_film_id)
FROM ( SELECT COUNT(film_id) AS count_film_id
       FROM inventory
       GROUP BY film_id )
AS temp_table;
```

This subquery is provided in the template.

544874.3500394.qx3zqy7

LAB ACTIVITY | 9.11.1: LAB - Nested aggregates (Sakila) | 10 / 10

Main.sql — Load default template...

```
1  -- Your SELECT statement goes here
2
3
4  -- Use the following subquery:
5  SELECT film.title
6  FROM film
7  JOIN (
8      SELECT film_id, COUNT(*) AS count_film_id
9      FROM inventory
10     GROUP BY film_id
11 ) AS temp_table ON film.film_id = temp_table.film_id
12 WHERE temp_table.count_film_id = (
13     SELECT MIN(count_film_id)
14     FROM (
15         SELECT COUNT(film_id) AS count_film_id
```

Develop mode | **Submit mode**

Explore the database and run your program as often as you'd like, before submitting for grading. Click **Run program** and observe the program's output in the second box.

Run program

Main.sql (Your program) → Output (shown below)

Program output displayed here

Coding trail of your work [What is this?]

```
2/27 T10 U10 min:3
```