36.1 Special operators and clauses

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 36.1.1: 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 36.1.1: IN operator.

Refer to the table below.

Country

Code	Name	Continent
ABW	Aruba	South 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', 'North 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

BETWEEN operator

The ***BETWEEN*** operator provides an alternative way to determine if a value is between two other values. The operator is written `value BETWEEN minValue AND maxValue` and is equivalent to `value >= minValue AND value <= maxValue`.

Figure 36.1.2: Equivalent queries.

```
SELECT Name
FROM Employee
WHERE HireDate >= '2000-01-01' AND HireDate <= '2020-01-
01';
```

```
SELECT Name
FROM Employee
WHERE HireDate BETWEEN '2000-01-01' AND '2020-01-01';
```

PARTICIPATION ACTIVITY 36.1.2: BETWEEN operator.

1) The expression `Age BETWEEN 13 AND 19` is true when Age is 19.

- True
- False

2) If the Name column has data type VARCHAR(20), then the expression `Name BETWEEN 'Anele' AND 'Jose'` is valid.

- True
- False

3) A query containing BETWEEN expression runs faster than a query containing an equivalent expression written with comparison operators.

- True
- False

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 36.1.3: 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:

Step 1: Using A underscore W with the LIKE operator matches 'A', any character, then 'W'. 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.

Step 2: Using A percent n with the LIKE operator matches 'A', any number of characters, then 'n'. 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.

Step 3: Using A percent with the LIKE operator matches 'A', then any number of characters. 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 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 satisfies 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 A_W with the LIKE operator matches 'A', any character, then 'W'.
2. Using A%n with the LIKE operator matches 'A', any number of characters, then 'n'.
3. Using A% with the LIKE operator matches 'A', then any number of characters.

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 36.1.4: 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*.

```
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
  ('Rogue One: A Star Wars Story', 'PG-13', '2016-12-16'),
  ('Star Trek', 'PG-13', '2009-05-08'),
  ('The Dark Knight', 'PG-13', '2008-07-18'),
  ('Stargate', 'PG-13', '1994-10-28'),
  ('Avengers: Endgame', 'PG-13', '2019-04-26');

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

```

Run Reset code

▶ View solution

PARTICIPATION ACTIVITY 36.1.5: 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 '_cha' | LIKE '%m_o%' | LIKE BINARY '%E%' | LIKE '%r%n' | LIKE '%cha'

	Arawakan and Caribbean
	Chibcha
	No matches
	Maori
	English

Reset

DISTINCT clause

The ***DISTINCT*** clause is used with a SELECT statement to return only unique or 'distinct' 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 36.1.3: SELECT DISTINCT example.

CountryLanguage

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 36.1.6: 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

ID	Name	CountryCode	District	Population
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 CountryCode, District
FROM City;
```

```
SELECT District
FROM City;
```



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

```
SELECT DISTINCT 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 36.1.4: 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 (ascending)
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 36.1.7: 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

CHALLENGE ACTIVITY 36.1.1: Special operators and clauses.

539740.3879454.qx3zqy7

Start

Country

● ISOCode2	Name	Capital	Continent
LB	Lebanon	Beirut	Asia
GN	Guinea	Conakry	Africa
SY	Syria	Damascus	Asia


```
SELECT Name
FROM Country
WHERE Continent IN ('Oceania', 'Asia', 'NAmerica');
```

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

- Lebanon

Guinea

Syria

1 2 3 4

Check Next

Exploring further:

- ORDER BY clause from MySQL.com
- String Comparison Functions and Operators from MySQL.com
- Regular Expressions from MySQL.com

36.2 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 36.2.1: 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);` returns 5

PARTICIPATION ACTIVITY 36.2.1: 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 36.2.2: 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 36.2.2: String functions.

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

Avatar

ID	Name	BestMove

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 36.2.3: 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) ***MONTH(d)*** ***YEAR(d)***	Returns the day, month, or year from date *d*	`SELECT DAY('2016-10-25');` returns 25 `SELECT MONTH('2016-10-25');` 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 36.2.3: 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.

```
1 CREATE TABLE Movie (
2   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
  ('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 | 36.2.4: 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.

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

```
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 36.2.1: Simple functions.

539740.3879454.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

36.3 Aggregate functions

Aggregate functions

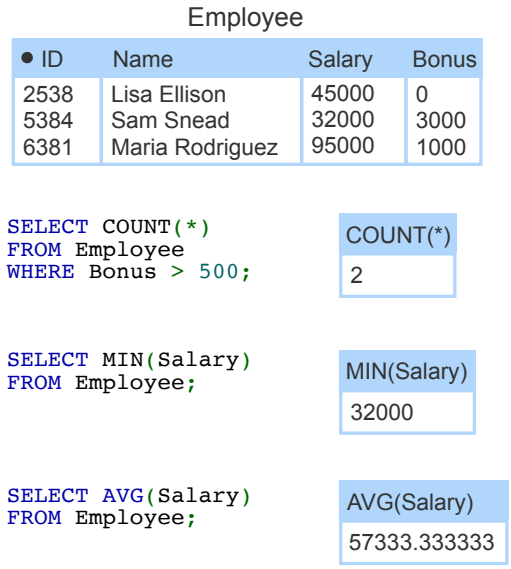
An ***aggregate function*** processes values from a set of rows and returns a summary value. Common aggregate functions are:

- ***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 36.3.1: Using aggregate functions in a SELECT statement.



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.
3. AVG() finds the average of the Salary column, which is 57333.333333.

PARTICIPATION ACTIVITY | 36.3.2: 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;
```

©zyBooks 01/31/24 18:21 1939727
Rob Daglio
MDCCOP2335Spring2024

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. Aside from the aggregate function, the SELECT clause may contain only column(s) that appear in the GROUP BY clause.

PARTICIPATION ACTIVITY 36.3.3: 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.
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 | 36.3.4: 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);
```


©zyBooks 01/31/24 18:21 1939727
Rob Daglio
MDCCOP2335Spring2024

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 36.3.5: 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 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 36.3.6: 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.

```
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', 19
  (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 36.3.7: 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

ID	Make	Model	Type	Year	Price
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 36.3.8: Aggregate functions ignore NULL values.

Compensation

ID	Name	Salary	Bonus
2538	Lisa Ellison	45000	NULL
5384	Sam Snead	32000	1000

172000 4000

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

6381	Maria Rodriguez	95000	3000

Result

SUM(Salary) + SUM(Bonus)
176000

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 36.3.9: SUM(Salary) + SUM(Bonus) is not the same as SUM(Salary + Bonus).

Compensation

ID	Name	Salary	Bonus	
2538	Lisa Ellison	45000	NULL	NULL
5384	Sam Snead	32000	1000	33000
6381	Maria Rodriguez	95000	3000	98000

131000

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

Result

SUM(Salary + Bonus)
131000

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:
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 36.3.10: 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 36.3.1: Aggregate functions.

539740.3879454.qx3zqy7

Start

Country

ID	Name	Capital	Over65PopulationRate
478	Mauritania	Nouakchott	3
266	Gabon	Libreville	5
566	Nigeria	Abuja	3
800	Uganda	Kampala	2

Complete the SELECT statement that finds the largest Over65PopulationRate.

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

(A) /* Type your code here */

(B)

1 2 3 4 5

Check Next

36.4 Join queries

Joins

In relational databases, reports are commonly 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 any columns with comparable data types.

PARTICIPATION ACTIVITY 36.4.1: 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:

Step 1: The join query displays department names and managers. There are three tables named Department Employee and Result. Department has three columns named Code Department Name and Manager. Employee has three columns named ID Employee Name and Salary. Manager is a foreign key of primary key ID. Three lines of code appear on the screen. Line one contains the code SELECT DepartmentName comma EmployeeName. Line two contains the code FROM Department comma Employee. Line three contains the code WHERE Manager equals ID semicolon. Line one is highlighted.

Step 2: Department is the left table. Employee is the right table. Line one highlight is removed. Line two, FROM Department, Employee, is highlighted.

Step 3: The query selects rows for which the foreign key Manager equals the primary key ID. Line two highlight is removed. Line three, WHERE Manager = ID, is highlighted. The Manager column of the Department table is highlighted. The ID column of the Employee column is highlighted.

Step 4: Lisa Ellison manages the Engineering department. Maria Rodriguez manages the Sales and Marketing departments. A result table appears. Result has two columns named Department Name and Employee Name. Row one of columns ID EmployeeName and Salary of the Employee table is highlighted and contains the values 2538 Lisa Ellison and 45000 respectively. Row one of columns Code Department Name and Manager of the Department table contains the values 44 Engineering and 2538 respectively. Engineering appears in row one of column Department Name of the Result table and Lisa Ellington appears in row one of column Employee Name of the Result table. Row three

of columns ID Employee Name and Salary of Employee is highlighted and contains the values 6381 Maria Rodriguez and 92300 respectively. Row two of Code Department Name and Manager of Department is highlighted and contains the values 82 Sales and 6381 respectively. Sales appears in row two of column Department Name of the Result table and Maria Rodriguez appears in row two of column Employee Name of the Result table. Row three of Code Department Name and Manager of Department is highlighted and contains the values 12 Marketing and 6381 respectively. Marketing appears in row three of column Department Name of the Result table and Maria Rodriguez appears in row three of column Employee Name of the Result table.

Step 5: Sam Snead's ID does not appear in the Manager column, so Sam Snead does not appear in the result. Row two of columns ID EmployeeName and Salary of Employee is highlighted and contains the values 5384 Sam Snead and 30500 respectively.

Step 6: Technical support has a NULL manager, so Technical support does not appear in the result. Row 4 of Code DepartmentName and Manager is highlighted and contains the values 99 Technical Support and NULL respectively.

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 | 36.4.2: 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) Refer to the tables in the above figure. What is missing from the join statement?

```
SELECT Employee.Name, 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.

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

Figure 36.4.1: 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
Engineering	Lisa Ellison
Sales	Maria Rodriguez
Marketing	Maria Rodriguez

PARTICIPATION ACTIVITY 36.4.3: Prefixes and aliases.

Refer to the tables below.

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

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

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*** selects only matching left and right table rows.
- ***FULL JOIN*** 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.

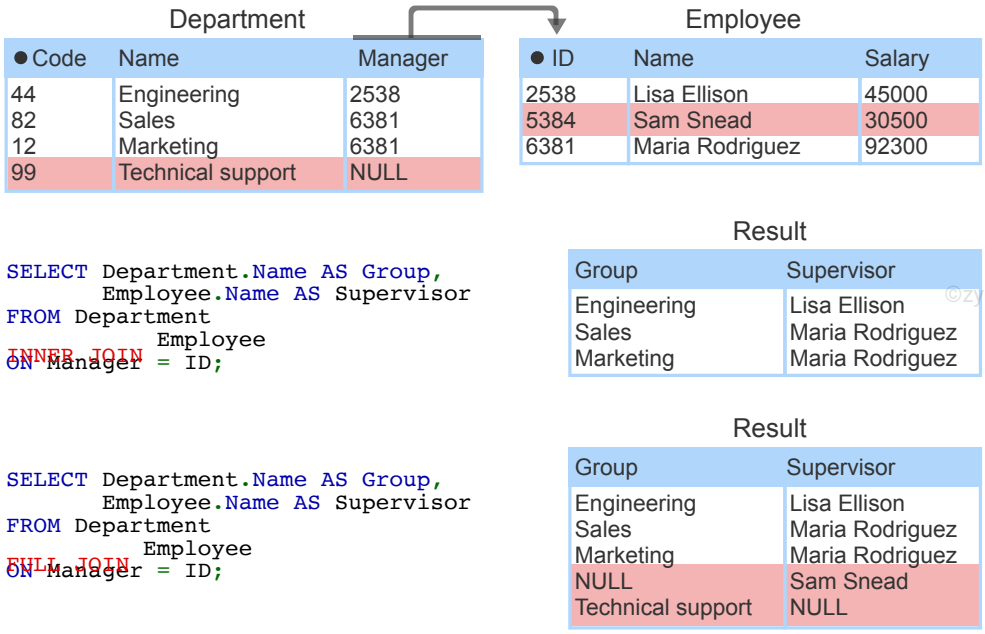
The join clause appears between a FROM clause and an ON clause:

- 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.

Join clauses are standard SQL syntax and supported by most relational databases. MySQL supports INNER JOIN but not FULL JOIN. For details of MySQL join syntax, see the link in 'Exploring further' below.

PARTICIPATION ACTIVITY 36.4.4: Inner and full joins.



Animation content:

Step 1: Inner joins are written with the keywords INNER JOIN. Duplicate columns Name are replaced with aliases Group and Supervisor. 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.

Step 2: No unmatched rows appear in an inner join result. 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.

Step 3: Full joins are written with the FULL JOIN keywords. Unmatched rows from both tables appear in the result. 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

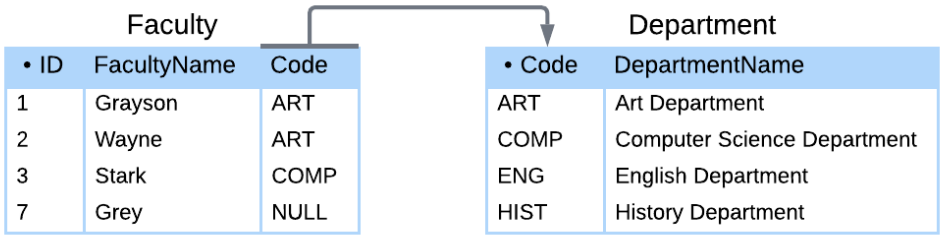
.

Animation captions:

1. Inner joins are written with the keywords INNER JOIN. Duplicate columns Name are replaced with 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 36.4.5: 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, including left, right, and full joins.

MySQL supports both LEFT JOIN and RIGHT JOIN.

PARTICIPATION ACTIVITY 36.4.6: 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
6381	Maria Rodriguez	92300

12	Marketing	6381
99	Technical support	NULL

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

Result

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

Animation content:

Step 1: Left joins are written with the LEFT JOIN keywords. Unmatched rows from Department, the left table, appear in the result. 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.

Step 2: Right joins are written with the RIGHT JOIN keywords. Unmatched rows from Employee, the right table, appear in the result. 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 appears with two columns named Department dot Name and Employee dot Name. Row two of columns ID Name and Salary of Employee is 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 36.4.7: Inner, left, right, and full joins.

Refer to the tables below.

Faculty

• ID	FacultyName	Code
1	Grayson	ART
2	Parker	MATH
3	Banner	MATH
4	Grey	NULL

Department

• Code	DepartmentName
ART	Art Department
ENG	English Department
HIST	History Department
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.

INNER RIGHT FULL LEFT

FacultyName	DepartmentName
Banner	Math Department
Grayson	Art Department
Parker	Math Department

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

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

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

Reset

Alternative join queries

Inner joins can be written without the JOIN keyword. Outer joins can be written with a UNION keyword instead of a JOIN keyword. ***UNION*** combines the results of two SELECT clauses into one result table:

- For a left join, one SELECT returns matching rows and another returns unmatched left table rows.
- For a right join, one SELECT returns matching rows and another returns unmatched right table rows.
- For a full join, three SELECT clauses are necessary. One SELECT returns matching rows, another returns unmatched left table rows, and a third returns unmatched right table rows. The three results are merged with two UNION keywords.

Use of the JOIN keyword is good practice. Join queries written with UNION are complex and difficult to understand. LEFT JOIN, RIGHT JOIN, and FULL JOIN clarify join behavior and simplify queries.

PARTICIPATION ACTIVITY 36.4.8: Equivalent join queries.

Left join with JOIN

```
SELECT Department.Name, Employee.Name
FROM Department
LEFT JOIN Employee
ON Manager = ID;
```

Left join with UNION

```
SELECT Department.Name, Employee.Name
FROM Department, Employee
WHERE Manager = ID

UNION
SELECT Department.Name, NULL
FROM Department
WHERE Manager IS NULL

   OR Manager NOT IN
     (SELECT ID
      FROM Employee
      WHERE ID IS NOT NULL);
```

Animation content:

Static figure:
Two SQL statements appear. The first statement has caption left join with JOIN:
Begin SQL code:
SELECT Department.Name, Employee.Name
FROM Department
LEFT JOIN Employee
ON Manager = ID;
End SQL code.

The second statement has caption left join with UNION:
Begin SQL code:
SELECT Department.Name, Employee.Name
FROM Department, Employee
WHERE Manager = ID
UNION
SELECT Department.Name, NULL
FROM Department
WHERE Manager IS NULL
  OR Manager NOT IN
   (SELECT ID
    FROM Employee
    WHERE ID IS NOT NULL);
End SQL code.

Step 1: The left join displays matching employees and departments, as well as unmatched departments. The caption left join with JOIN appears. The first statement appears.

Step 2: An equivalent left join can be written with two SELECT clauses and a UNION keyword. The caption left join with UNION appears. Three keywords appear under the caption:
SELECT
UNION
SELECT

Step 3: The first SELECT returns matching employees and departments. The first SELECT clause appears above the UNION keyword.

Step 4: The second SELECT returns departments with a NULL manager. The first three lines of the second SELECT clause appear below the UNION keyword:
Begin SQL code:
```
SELECT Department.Name, NULL
FROM Department
WHERE Manager IS NULL
```
End SQL code.

Step 5: "OR Manager NOT IN . . ." also selects non-NULL managers that do not match any employee. The last four lines of the second SELECT clause appear:
Begin SQL code:
```
  OR Manager NOT IN
    (SELECT ID
     FROM Employee
     WHERE ID IS NOT NULL);
```
End SQL code.

Step 6: "Manager IS NULL" is unnecessary if the Manager column is NOT NULL. "Manager IS NULL OR" is struck out.

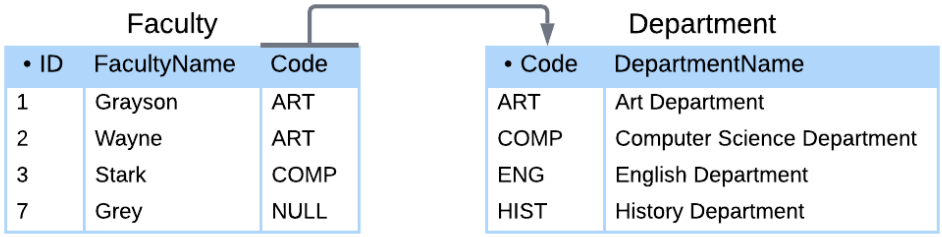
Step 7: "Manager NOT IN . . ." is unnecessary if Manager is a foreign key that references ID. The last four lines of the second SELECT clause, beginning with OR, are struck out.

Animation captions:

1. The left join displays matching employees and departments, as well as unmatched departments.
2. An equivalent left join can be written with two SELECT clauses and a UNION keyword.
3. The first SELECT returns matching employees and departments.
4. The second SELECT returns departments with a NULL manager.
5. "OR Manager NOT IN . . ." also selects non-NULL managers that do not match any employee.
6. "Manager IS NULL" is unnecessary if the Manager column is NOT NULL.
7. "Manager NOT IN . . ." is unnecessary if Manager is a foreign key that references ID.

PARTICIPATION ACTIVITY 36.4.9: Alternative join queries.

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

Match the join type to the example query.

If unable to drag and drop, refresh the page.

inner join | left join | full join | right join

```
SELECT FacultyName,
DepartmentName
FROM Faculty, Department
WHERE Faculty.Code =
Department.Code
UNION
SELECT NULL, DepartmentName
FROM Department
WHERE Department.Code NOT IN
   (SELECT Code FROM Faculty
WHERE Code IS NOT NULL);
```

```
SELECT FacultyName,
DepartmentName
FROM Faculty, Department
WHERE Faculty.Code =
Department.Code;
```

```
SELECT FacultyName,
DepartmentName
FROM Faculty, Department
WHERE Faculty.Code =
Department.Code
UNION
SELECT FacultyName, NULL
FROM Faculty
WHERE Faculty.Code IS NULL;
```

```
SELECT FacultyName,
DepartmentName
FROM Faculty, Department
WHERE Faculty.Code =
Department.Code
UNION
SELECT FacultyName, NULL
FROM Faculty
WHERE  Faculty.Code IS NULL
UNION
SELECT NULL, DepartmentName
FROM Department
WHERE Department.Code NOT IN
   (SELECT Code FROM Faculty
WHERE CODE IS NOT NULL);
```

©zyBooks 01/31/24 18:21 1939727
Rob Daglio
MDCCOP2335Spring2024

Reset

PARTICIPATION ACTIVITY 36.4.10: Join coding exercise.

The SQL below creates Genre and Song tables, inserts some genres and songs, and performs an inner join.

Run the SQL. Verify the result table does not include songs with NULL genre or genres that are not associated with songs.

Make the following changes:

1. In the CREATE TABLE statement for Song, rename GenreCode to Code. Modify the SELECT statement to work with the new name. Run the SQL and verify the result table is unchanged.
2. Modify the SELECT statement to perform a left join. Run the SQL and verify the result table includes songs with NULL genre.
3. Modify the SELECT statement to perform a right join. Run the SQL and verify the result table includes genres that are not associated with any songs.
4. Combine the left and right joins into one statement that performs a full join. Run the SQL and verify the result table includes all songs and genres.

Hints: In steps 2 and 3, use keywords LEFT and RIGHT. In step 4, use keyword UNION, since MySQL does not support FULL JOIN.


```
CREATE TABLE Genre (
  Code CHAR(3),
  Name VARCHAR(20),
  Description VARCHAR(200),
  PRIMARY KEY(Code)
);

CREATE TABLE Song (
  ID INT,
  Title VARCHAR(60),
  Artist VARCHAR(60),
  GenreCode CHAR(3),
  PRIMARY KEY (ID),
  FOREIGN KEY (GenreCode) REFERENCES Genre(Code)
);

INSERT INTO Genre VALUES
  ('CLA', 'Classical', 'Orchestral music composed and performed by profes
  ('COU', 'Country', 'Developed mostly in southern USA, with roots in tr
  ('DRO', 'Drone', 'Minimalist music that emphasizes sustained or repeat
  ('GRU', 'Grunge', 'Alternative rock inspired by hardcore punk, heavy me
  ('PRC', 'Pop Rock', 'Rock music with less attitude'),
  ('RAB', 'R&B', 'Modern version of soul and funk African-American pop mu
  ('TEC', 'Techno', 'Electronic music');

INSERT INTO Song VALUES
  (100, 'Hey Jude', 'Beatles', 'PRC'),
  (200, 'You Belong With Me', 'Taylor Swift', NULL),
  (300, 'Need You Now', 'Lady Antebellum', 'COU'),
  (400, 'Old Town Road', 'Lil Nas X', NULL),
  (500, 'That\'s The Way Love Goes', 'Janet Jackson', 'RAB'),
  (600, 'Even Flow', 'Pearl Jam', 'GRU');


SELECT *
```

Run Reset code

▶ View solution

CHALLENGE ACTIVITY | 36.4.1: Inner and outer joins.

539740.3879454.qx3zqy7

Start

Complete the SQL statement to generate the Result table. The column names in the SELECT c the Result table exactly.

College

• CollegeName	City	Enrollment
UCLA	Los Angeles	7425
Pomona	Claremont	1663
MIT	Cambridge	11376
Duke	Durham	15192

Location

• City	StateCode
Los Angeles	CA
Claremont	CA
Cambridge	MA
Durham	NC

```
SELECT /* Type your code here */
FROM College, Location
WHERE /* Type your code here */ ;
```

Result

CollegeName	StateCode
UCLA	CA
Pomona	CA
MIT	MA
Duke	NC

1 2 3

Check Next

Exploring further:

- MySQL join syntax

36.5 Equijoins, self-joins, and cross-joins

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 36.5.1: 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 36.5.1: 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

Self-joins

A ***self-join*** joins a table to itself. A self-join can compare any columns of a table, as long as the columns have comparable data types. If a foreign key and the referenced primary key are in the same table, a self-join commonly compares those key columns. In a self-join, aliases are necessary to distinguish left and right tables.

In the figure below, A is the left table's alias, and B is the right table's alias. `A.Name` is the Name column of the left table, representing the employee. `B.Name` is the Name column of the right table, representing the employee's manager. The result shows employees along with each employee's manager.

Figure 36.5.2: Self-join example.

Employee

• ID	Name	Manager
2538	Lisa Ellison	8820
5384	Sam Snead	8820
6381	Maria Rodriguez	8820
8820	Jiho Chen	NULL

```
SELECT A.Name, B.Name
FROM Employee A
INNER JOIN Employee B
ON B.ID = A.Manager;
```

Result

A.Name	B.Name
Lisa Ellison	Jiho Chen
Sam Snead	Jiho Chen
Maria Rodriguez	Jiho Chen

PARTICIPATION ACTIVITY 36.5.2: Self-joins.

Refer to the following table.

```
CREATE TABLE Person (
  ID INT,
  FullName VARCHAR(20) NOT NULL,
  FirstParentID INT,
  SecondParentID INT,
  PRIMARY KEY (ID),
  FOREIGN KEY (FirstParentID) REFERENCES Person(ID),
  FOREIGN KEY (SecondParentID) REFERENCES Person(ID)
);
```


The following self-join lists the name of each person with the person's parent (or legal guardian). If a person has two parents, each parent appears in a separate row.

```
SELECT ___A___, Parent.FullName
FROM Person AS Child
INNER JOIN ___B___
ON Child.FirstParentID = Parent.ID OR ___C___;
```

1) What is A?

Check Show answer

2) What is B?

Check Show answer

3) What is C?

Check Show answer

Cross-joins

A ***cross-join*** combines two tables without comparing columns. A cross-join uses a ***CROSS JOIN*** clause without an ON clause. As a result, all possible combinations of rows from both tables appear in the result.

In the figure below, all configurations of iPhone models and storage appear, along with total price.

Figure 36.5.3: Cross-join example.

IPhone

Model	Price
X	1100
XR	800

Storage

Gigabytes	Price
64	0
128	100
256	200

```
SELECT Model, Gigabytes, IPhone.Price +
Storage.Price
FROM IPhone
CROSS JOIN Storage;
```

Result

Model	Gigabytes	IPhone.Price + Storage.Price
X	64	1100
XR	64	800
X	128	1200
XR	128	900
X	256	1300
XR	256	1000

PARTICIPATION ACTIVITY 36.5.3: Special cases.

Match the special join with the definition.

If unable to drag and drop, refresh the page.

Cross-join | Equijoin | Self-join | Non-equijoin

	Combines two tables without comparing columns.
	Compares columns with an operator other than =, such as < and >.
	Joins a table to itself.
	Compares columns of two tables with the = operator.

Reset

CHALLENGE ACTIVITY 36.5.1: Equijoins, self-joins, and cross-joins.

539740.3879454.qx3zqy7

Start

What type of join is the query? Select all that apply.

```
SELECT Person1.Name as Spouse1, Person2.Name as Spouse2
FROM Person AS Person1, Person AS Person2
WHERE Person1.SpouseID = Person2.ID
```

- [ ] Equijoin
- [ ] Non-equijoin
- [ ] Self-join
- [ ] Cross-join

1

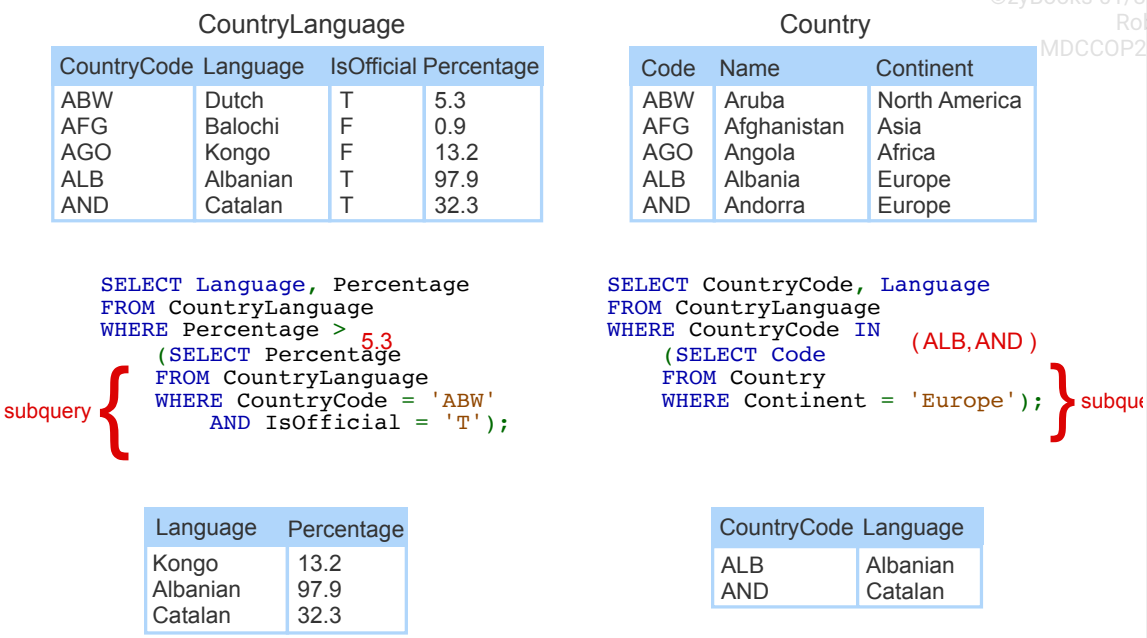
Check Try again

36.6 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 | 36.6.1: Subquery examples.



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
 (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 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 36.6.2: 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', 19
  (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

PARTICIPATION ACTIVITY 36.6.3: 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
    (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 36.6.4: 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:

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.

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 36.6.5: 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 36.6.6: Correlated subquery using EXISTS.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ARG	Italian	F	1.7
ARG	Spanish	T	96.8
BRA			

City

Id	Name	CountryCode	Population
69	Buenos Aires	ARG	2982146
70	La Matanza	ARG	1266461

BRA	German	F	0.5	206	São Paulo	BRA	9968485
	Portuguese	T	97.5	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 36.6.7: 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 36.6.8: 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 | 36.6.9: 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:

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
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 36.6.10: 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 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

©zyBooks 01/31/24 18:21 1939727
Rob Daglio
MDCCOP2335Spring2024

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 | 36.6.1: Subqueries.

539740.3879454.qx3zqy7

Start

Course

• CourseId	CourseCode	CourseName	Capacity	Instructor
5570	HIST78	American History	200	3
2035	BUS200	Financial Management	100	1
7485	PHIL30	Formal Logic	125	2
2607	BUS376	Personal Finance	150	1
1382	HIST49	European History	175	3

Instructor

• InstructorId	InstructorName	Rank	Department
1	Ani Khan	Professor	Business
2	Taj Hill	Professor	Philosophy
3	Ben Leon	Associate 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
WHERE InstructorId =
   (SELECT InstructorId
   FROM Instructor
   WHERE InstructorName = 'Taj Hill');
```

- PHIL30
- HIST78
- Formal Logic

1 2 3 4 5

Check Next

Exploring further:

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

36.7 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?"

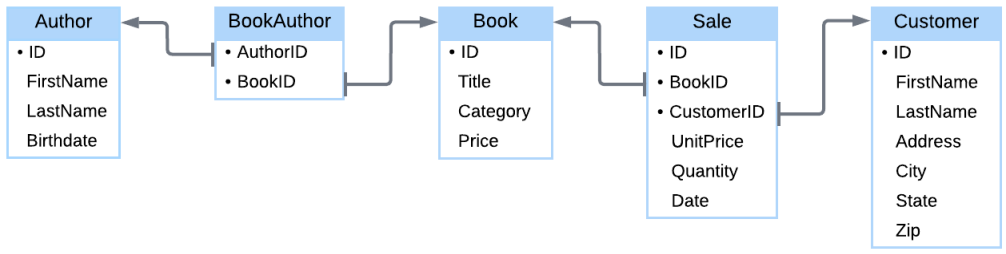
To create a complex query, a database user can employ the following strategy:

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 36.7.1: Table diagram for Zion Bookstore database.



PARTICIPATION ACTIVITY 36.7.1: 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 36.7.2: 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','fantasy'
  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 36.7.3: Joining tables.

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

- CO
- NM
- OK

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 36.7.4: 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','fantasy'
  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 36.7.5: 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 36.7.6: 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;
```

```
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','fantasy'
  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 36.7.7: 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 36.7.8: 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','fantasy'
  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 36.7.9: 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.

CHALLENGE ACTIVITY 36.7.1: Complex queries.

539740.3879454.qx3zqy7

Start

A university wants to know the names of courses with more than 3 credits, taken by the oldest email address.

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

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

1 2 3 4

Check Next

36.8 View tables

Creating views

Table design is optimized for a variety of reasons, such as minimal storage, fast query execution, and support for relational and business rules. Occasionally, the design is not ideal for database users and programmers. Ex: The Employee table may contain personal information, such as name, marital status, and birth date. A separate Address table may contain employee addresses. This design allows several employees to share the same address. Human resources staff, however, may prefer to see personal and address information in one table rather than two.

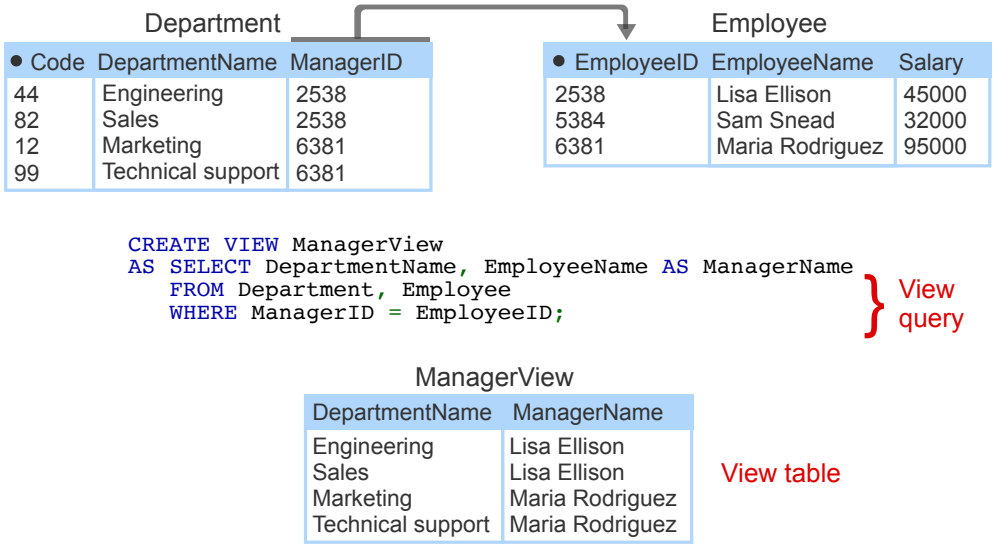
View tables solve this problem. Views restructure table columns and data types without changes to the underlying database design.

A ***view table*** is a table name associated with a SELECT statement, called the ***view query***. The ***CREATE VIEW*** statement creates a view table and specifies the view name, query, and, optionally, column names. If column names are not specified, column names are the same as in the view query result table.

Figure 36.8.1: CREATE VIEW statement.

```
CREATE VIEW ViewName [ ( Column1, Column2, ...
) ]
AS SelectStatement;
```


PARTICIPATION ACTIVITY 36.8.1: Creating a view table.



Animation content:

Static figure:
The Department table has columns Code, DepartmentName, and ManagerID. Code is the primary key. Department has four rows:
44, Engineering, 2538
82, Sales, 2538
12, Marketing, 6381
99, Technical Support, 6381

The Employee table has columns EmployeeID, EmployeeName, Salary. EmployeeID is the primary key. Employee has three rows:
2538, Lisa Ellison, 45000
5284, Sam Snead, 32000
6381, Maria Rodriguez, 95000

An arrow points from Manager to ID.

An SQL statement appears:
Begin SQL code:
CREATE VIEW ManagerView
AS SELECT DepartmentName, EmployeeName AS ManagerName
FROM Department, Employee

WHERE ManagerID = EmployeeID;
End SQL code.
The last three lines of the statement are labeled View query.

The ManagerView table appears below the statement, with columns DepartmentName and ManagerName. ManagerView has four rows:
Engineering, Lisa Ellison
Sales, Lisa Ellison
Marketing, Maria Rodriguez
Technical Support, Maria Rodriguez

The label View table appears next to ManagerView.

Step 1: Department contains a foreign key that identifies the manager in Employee. Pairs of Department and Employee rows with the same ManagerID and EmployeeID are highlighted.

Step 2: CREATE VIEW creates a view table called ManagerView. The view query is a SELECT statement. The statement appears. The last three lines are labeled View query.

Step 3: The view query joins Department to Employee and renames EmployeeName as ManagerName. EmployeeName as ManagerName is highlighted. The ManagerView table appears. The label View table appears next to ManagerView.

Animation captions:

1. Department contains a foreign key that identifies the manager in Employee.
2. CREATE VIEW creates a view table called ManagerView. The view query is a SELECT statement.
3. The view query joins Department to Employee and renames EmployeeName as ManagerName.

PARTICIPATION ACTIVITY 36.8.2: Creating view tables.

Refer to the following CREATE VIEW statement and view table.

Faculty

• ID	FacultyName	Code
1	Grayson	ART
2	Wayne	ART
3	Stark	COMP
4	Parker	MATH
5	Banner	MATH
6	Quinn	MATH
7	Grey	NULL

Department

• Code	DepartmentName
ART	Art Department
COMP	Computer Science Department
ENG	English Department
HIST	History Department
MATH	Math Department


```
CREATE VIEW ___A___
AS SELECT FacultyName AS ___B___, ___C___ AS Assignment
   FROM Faculty, Department
   WHERE Faculty.___D___ = Department.Code
         AND Department.Code = '___E___';
```

MathFacultyView

Professor	Assignment
Parker	Math Department
Banner	Math Department
Quinn	Math Department

1) What is table name A?

Check Show answer

2) What is column name B?

Check Show answer

3) What is column name C?

Check Show answer

4) What is view column name D?

Check Show answer

5) What is value E?

Check Show answer

Querying views

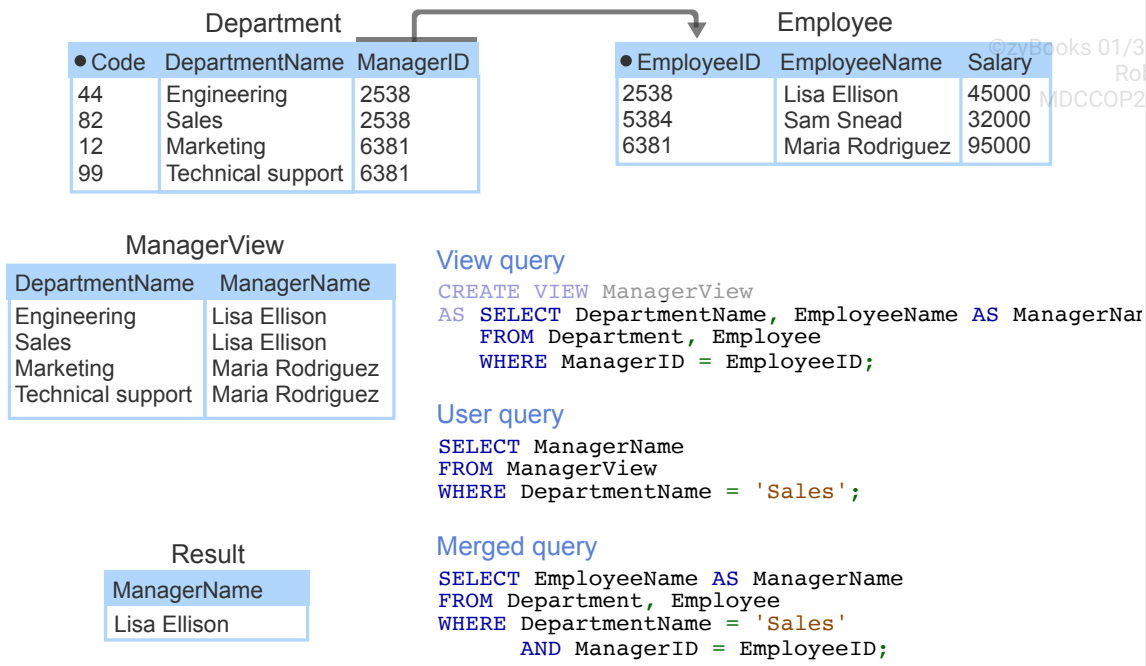
A table specified in the view query's FROM clause is called a ***base table***. Unlike base table data, view table data is not normally stored. Instead, when a view table appears in an SQL statement, the view query is merged with the SQL query. The database executes the merged query against base tables.

In some databases, view data can be stored. A ***materialized view*** is a view for which data is stored at all times. Whenever a base table changes, the corresponding view tables can also change, so materialized views must be refreshed. To avoid the overhead of refreshing views, MySQL and many other databases do not support materialized views.

Terminology

A view can be defined on other view tables when the view query FROM clause includes additional view tables. In this case, the additional view tables are not base tables. ***Base tables*** *are always source tables, created as tables rather than as views.*

PARTICIPATION ACTIVITY 36.8.3: Merging the view query and user query.



Animation content:

Static figure:
The Department table has columns Code, DepartmentName, and ManagerID. Code is the primary key. Department has four rows:
44, Engineering, 2538
82, Sales, 2538
12, Marketing, 6381
99, Technical Support, 6381

The Employee table has columns EmployeeID, EmployeeName, and Salary. ID is the primary key. Employee has three rows:
2538, Lisa Ellison, 45000
5284, Sam Snead, 32000
6381, Maria Rodriguez, 95000

An arrow points from ManagerID to EmployeeID.

An SQL statement appears, labeled View query:
Begin SQL code:
CREATE VIEW ManagerView
AS SELECT DepartmentName, EmployeeName AS ManagerName
  FROM Department, Employee
  WHERE ManagerID = EmployeeID;
End SQL code.
The last three lines of the statement are labeled View query.

The ManagerView table appears next to the statement, with columns DepartmentName and ManagerName. ManagerView has four rows:
Engineering, Lisa Ellison
Sales, Lisa Ellison
Marketing, Maria Rodriguez
Technical Support, Maria Rodriguez

A second SQL statement appears, labeled User query.
Begin SQL code:
SELECT ManagerName
FROM MangerView
WHERE DepartmentName = 'Sales';
End SQL code.

A third SQL statement appears, labeled Merged query.
Begin SQL code:
SELECT EmployeeName AS ManagerName
FROM Department, Employee
WHERE DepartmentName = 'Sales'
    AND ManagerID = EmployeeID;
End SQL code.

A result table appears with column ManagerName and one row:
Lisa Ellison

Step 1: A view query joins the base tables Department to Employee to create the ManagerView view table. The Department and Employee tables appear. The first statement appears. The ManagerView table appears.

Step 2: The user submits a query against the ManagerView table. The user query appears.

Step 3: The database merges the user query with the view query. The user query is duplicated. ManagerName is replaced with EmployeeName AS ManagerName. FROM ManagerView is replaced with FROM Department, Employee. AND ManagerID = EmployeeID is added to the WHERE clause. The modified query is labeled Merged query.

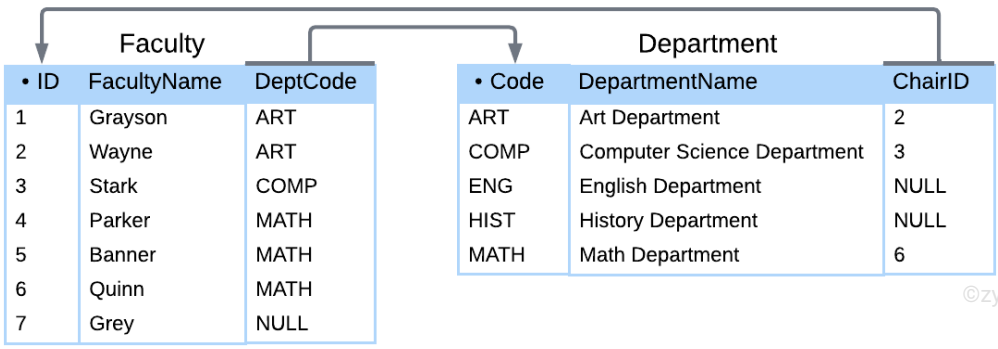
Step 4: The merged query is executed against base tables. The result shows the manager of Sales. The result table appears.

Animation captions:

1. A view query joins the base tables Department to Employee to create the ManagerView view table.
2. The user submits a query against the ManagerView table.
3. The database merges the user query with the view query.
4. The merged query is executed against base tables. The result shows the manager of Sales.

PARTICIPATION ACTIVITY 36.8.4: Querying views.

Refer to the following CREATE VIEW statement and view table.



Faculty

• ID	FacultyName	DeptCode
1	Grayson	ART
2	Wayne	ART
3	Stark	COMP
4	Parker	MATH
5	Banner	MATH
6	Quinn	MATH
7	Grey	NULL

Department

• Code	DepartmentName	ChairID
ART	Art Department	2
COMP	Computer Science Department	3
ENG	English Department	NULL
HIST	History Department	NULL
MATH	Math Department	6


```
CREATE VIEW DepartmentView
AS SELECT Department.Name AS DepartmentName, Faculty.Name AS Chair
   FROM Department, Faculty
   WHERE ChairID = Faculty.ID;
```

DepartmentView

DepartmentName	Chair
Art Department	Wayne
Computer Science Department	Stark
Math Department	Quinn

1) What is missing to get all Chair names?

```
SELECT Chair
FROM ______;
```

- Faculty
- Department
- DepartmentView

2) What are the view query's base tables?

- Faculty only
- Faculty and Department
- DepartmentView

3) If the view table is not stored, against what tables is the query executed?

```
SELECT *
FROM DepartmentView;
```

- Department only
- Faculty and Department
- DepartmentView

4) If DepartmentView is a materialized view, against what tables is the query executed?


```
SELECT *
FROM DepartmentView;
```

- Department only
- Faculty and Department
- DepartmentView

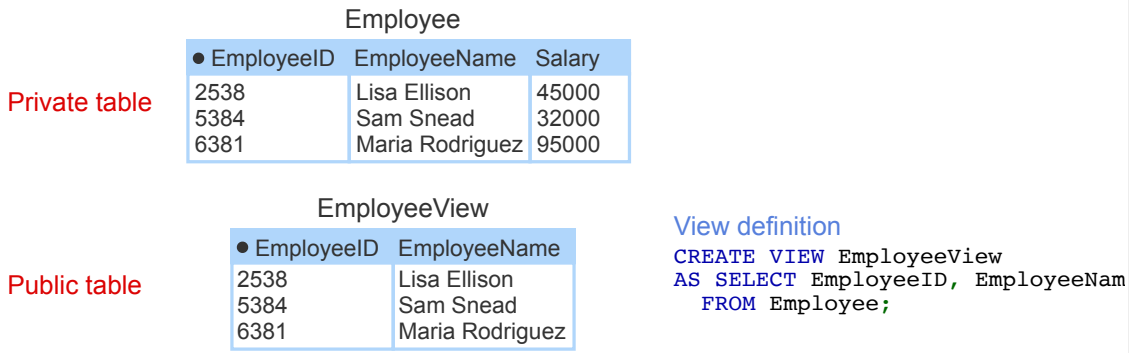
Advantages of views

View tables have several advantages:

- *Protect sensitive data*. A table may contain sensitive data. Ex: The Employee table contains compensation columns such as Salary and Bonus. A view can exclude sensitive columns but include all other columns. Authorizing users and programmers to access the view but not the underlying table protects the sensitive data.
- *Save complex queries*. Complex SELECT statements can be saved as a view. Database users can reference the view without writing the SELECT statement.
- *Save optimized queries*. Often, the same result table can be generated with equivalent SELECT statements. Although the results of equivalent statements are the same, performance may vary. To ensure fast execution, the optimal statement can be saved as a view and distributed to database users.

For the above reasons, views are supported in all relational databases and are frequently created by database administrators. Database users need not be aware of the difference between view and base tables.

PARTICIPATION ACTIVITY | 36.8.5: Advantages of views.



Animation content:

Static figure:
The Employee table has columns EmployeeID, EmployeeName, and Salary. EmployeeID is the primary key. Employee has three rows:
2538, Lisa Ellison, 45000
5284, Sam Snead, 32000
6381, Maria Rodriguez, 95000
The Employee table is labeled Private table.

An SQL statement appears, labeled View definition:
Begin SQL code:
CREATE VIEW EmployeeView
AS SELECT EmployeeID, EmployeeName
FROM Employee;
End SQL code.

The EmployeeView table has columns EmployeeID and EmployeeName. EmployeeID is the primary key. EmployeeView has three rows:
2538, Lisa Ellison

5284, Sam Snead
6381, Maria Rodriguez
The EmployeeView table is labeled Public table.

Step 1: The Employee table contains sensitive salary data. The Salary column is highlighted.

Step 2: The EmployeeView query hides salary data. The view definition appears. EmployeeView appears.

Step 3: Only Human Resources staff can access Employee. Anyone can access EmployeeView. Employee is labeled Private table. EmployeeView is labeled Public table.

Animation captions:

1. The Employee table contains sensitive salary data.
2. The EmployeeView view table hides salary data.
3. Only Human Resources staff can access Employee. Anyone can access EmployeeView.

PARTICIPATION ACTIVITY 36.8.6: Advantages of views.

1) Using materialized views always improves database performance.
   - ○ True
   - ○ False

2) The performance of a user query on a view is identical to the performance of the corresponding merged query on base tables.
   - ○ True
   - ○ False

3) A view query can reference another view table.
   - ○ True
   - ○ False

4) In MySQL, two different queries that generate the same result table always have the same execution time.
   - ○ True
   - ○ False

5) Views can be used to hide rows as well as columns from database users.
   - ○ True
   - ○ False

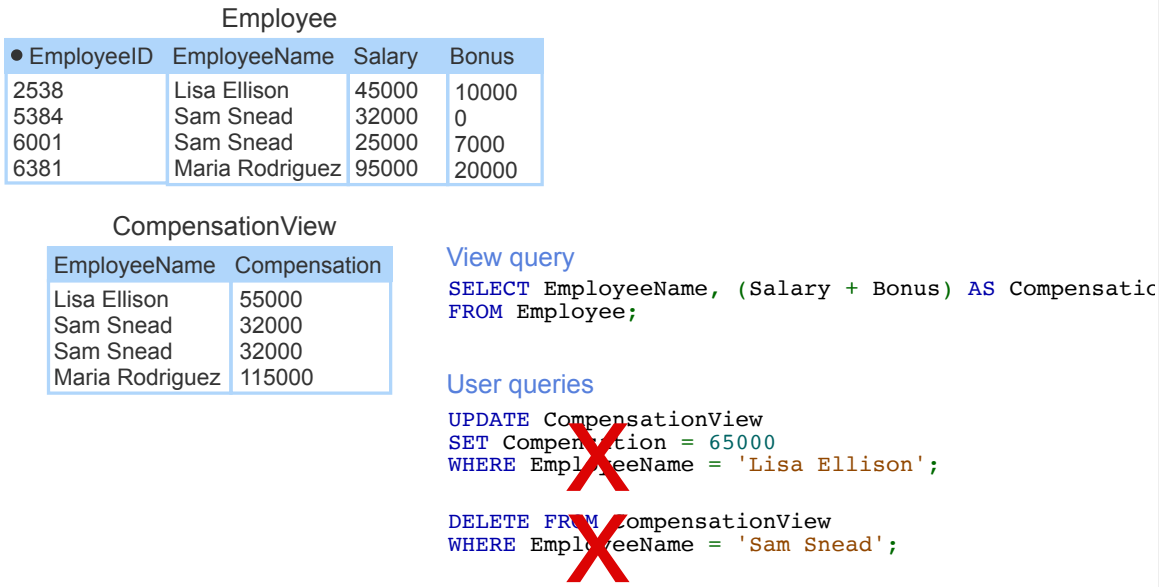
Inserting, updating, and deleting views

View tables are commonly used in SELECT statements. Using views in INSERT, UPDATE, and DELETE statements is problematic:

- *Primary keys*. If a base table primary key does not appear in a view, an insert to the view generates a NULL primary key value. Since primary keys may not be NULL, the insert is not allowed.
- *Aggregate values*. A view query may contain aggregate functions such as AVG() or SUM(). One aggregate value corresponds to many base table values. An update or insert to the view may create a new aggregate value, which must be converted to many base table values. The conversion is undefined, so the insert or update is not allowed.
- *Join views*. In a join view, foreign keys of one base table may match primary keys of another. A delete from a view might delete foreign key rows only, or primary key rows only, or both the primary and foreign key rows. The effect of the join view delete is undefined and therefore not allowed.

The above examples illustrate just a few of many potential problems of changing data in view tables. As a result, relational databases either disallow or severely limit view table inserts, updates, and deletes. Regardless of specific database limitations, inserts, updates, and deletes to views should be avoided. Views are best for reading data.

PARTICIPATION ACTIVITY 36.8.7: Inserting, updating, and deleting views.



Animation content:

Static figure:
The Employee table has columns EmployeeID, EmployeeName, Salary, and Bonus. EmployeeID is the primary key. Employee has four rows:
2538, Lisa Ellison, 45000, 10000
5284, Sam Snead, 32000, 0
6001, Sam Snead, 25000, 7000
6381, Maria Rodriguez, 95000, 20000

An SQL statement appears, labeled View query:
Begin SQL code:
SELECT EmployeeName, (Salary + Bonus) AS Compensation
FROM Employee;
End SQL code.

The CompensationView table has columns EmployeeName and Compensation. CompensationView has four rows:
Lisa Ellison, 55000
Sam Snead, 32000
Sam Snead, 32000
Maria Rodriguez, 115000

Two more SQL statements appear, labeled User queries:
Begin SQL code:
UPDATE CompensationView
SET Compensation = 6500
WHERE EmployeeName = 'Lisa Ellison';

DELETE FROM CompensationView
WHERE EmployeeName = 'Sam Snead';
End SQL code.
A bold X appears above both statements.

Step 1: The Employee table contains salary and bonus information. The Employee table appears. Salary and Bonus columns are highlighted,

Step 2: Two different employees have the same name but different EmployeeIDs. Rows two and three of Employee, named Sam Snead, are highlighted.

Step 3: The CompensationView table contains total compensation. The view query appears. (Salary + Bonus) AS Compensation is highlighted. The CompensationView table appears. The Compensation column is highlighted.

Step 4: The user query updates Lisa Ellison's compensation. Lisa Ellison's new salary and bonus are undefined, so the query is rejected. The first user query, the UPDATE statement, appears. ? appears in the Salary and Bonus columns of Employee for the Lisa Ellison row. An X appears above the UPDATE statement.

Step 5: The user query deletes Sam Snead from CompensationView. Which Sam Snead is deleted from the base table? Query is rejected. The second user query, the DELETE statement, appears. ? appears next to the two Sam Snead rows in Employee. An X appears above the DELETE statement.

Animation captions:

1. The Employee table contains salary and bonus information.
2. Two different employees have the same name but different EmployeeIDs.
3. The CompensationView table contains total compensation.
4. The user query updates Lisa Ellison's compensation. Lisa Ellison's new salary and bonus are undefined, so the query is rejected.
5. The user query deletes Sam Snead from CompensationView. Which Sam Snead is deleted from the base table? Query is rejected.

PARTICIPATION ACTIVITY 36.8.8: Inserting, updating, and deleting views.

Refer to the following base table, view definition, and view table.

Department

• Code	DepartmentName	Budget	Actual
ART	Art Department	85000	80000
COMP	Computer Science Department	72000	79000
ENG	English Department	35000	35000
HIST	History Department	102000	98000
MATH	Math Department	60000	70000


```
CREATE VIEW AccountView
AS SELECT Code, DepartmentName, (Budget - Actual) AS Difference
   FROM Department;
```

AccountView

Code	DepartmentName	Difference
ART	Art Department	5000
COMP	Computer Science Department	-7000
ENG	English Department	0
HIST	History Department	4000
MATH	Math Department	-10000

Indicate whether the following queries are valid.

1)
```
INSERT INTO AccountView
(DepartmentName)
VALUES ('Music Department');
```

- ○ Valid
- ○ Invalid

2)
```
UPDATE AccountView
SET Difference = 4400
WHERE Code = 'COMP';
```

- ○ Valid
- ○ Invalid

3)
```
UPDATE AccountView
SET DepartmentName =
'Information Technology
Department'
WHERE Code = 'COMP';
```

- ○ Valid
- ○ Invalid

4)
```
DELETE FROM AccountView
WHERE DepartmentName = 'History
Department';
```

- ○ Valid
- ○ Invalid

WITH CHECK OPTION clause

Databases that allow view updates face one particularly bothersome behavior. A view insert or update may create a row that does not satisfy the view query WHERE clause. In this case, the inserted or updated row does not appear in the view table.

From the perspective of the database user, the insert or update appears to fail even though the base tables have changed.

To prevent inserts or updates that appear to fail, databases that support view updates have an optional WITH CHECK OPTION clause. When ***WITH CHECK OPTION*** is specified, the database rejects inserts and updates that do not satisfy the view query WHERE clause. Instead, the database generates an error message that explains the violation.

Figure 36.8.2: WITH CHECK OPTION clause.

```
CREATE VIEW ViewName [ ( Column1, Column2, ...
) ]
AS SelectStatement
[ WITH CHECK OPTION ];
```


PARTICIPATION ACTIVITY 36.8.9: WITH CHECK OPTION clause.

Employee

EmployeeID	EmployeeName	DepartmentCode
2538	Lisa Ellison	23
5384	Sam Snead	51
6381	Maria Rodriguez	51

SalesEmployee

ID	Name	Department
5384	Sam Snead	51
6381	Maria Rodriguez	51

View definition

```
CREATE VIEW SalesEmployee (ID, Name, Department
AS SELECT *
    FROM Employee
    WHERE DepartmentCode = 51

WITH CHECK OPTION;
```

User Query

```
INSERT INTO SalesEmployee
VALUES (852[illegible] 'Jiho Chen', 80);
```

X

Animation content:

Static figure:
The Employee table has columns EmployeeID, EmployeeName, and DepartmentCode. EmployeeID is the primary key. Employee has three rows:
2538, Lisa Ellison, 23
5284, Sam Snead, 51
6381, Maria Rodriguez, 51

An SQL statement appears, labeled View definition:
Begin SQL code:
CREATE VIEW SalesEmployee (ID, Name, Department)
AS SELECT *
 FROM Employee
 WHERE DepartmentCode = 51
WITH CHECK OPTION;
End SQL code.
The clause WITH CHECK OPTION is highlighted.

The SalesEmployee table has columns ID, Name, and Department. SalesEmployee has two rows:
5384, Sam Snead, 51
6381, Maria Rodriguez, 51

Another SQL statement appears, labeled User query:
Begin SQL code:
INSERT INTO SalesEmployee
VALUES (8520, 'Jiho Chen', 80);
End SQL code.
An X appears above the statement.

Step 1: Two employees are in department 51, Sales. The Employee table appears. Rows two and three are highlighted.

Step 2: CREATE VIEW specifies new names for SalesEmployee columns. The view definition appears without the clause WITH CHECK OPTION. In the view definition, (ID, Name, Department) is highlighted. The SalesEmployee column names are highlighted.

Step 3: SalesEmployee view only includes employees in department 51. The clause WHERE DepartementCode = 51 is highlighted. The two values 51 in SalesEmployee are highlighted.

Step 4: The user query inserts a new employee in department 80 into the view table. The user query appears.

Step 5: The new employee is inserted into Employee but does not appear in the view table. A new row is added to the Employee table:
8520, Jiho Chen, 80

Step 6: WITH CHECK OPTION prevents inserts and updates that do not satisfy the view query WHERE clause. The WITH CHECK OPTION clause is added to the view definition. An X appears above the user query. The new row added in step 5 is removed from the Employee table.

Animation captions:

1. Two employees are in department 51, Sales.
2. CREATE VIEW specifies new names for SalesEmployee columns.
3. SalesEmployee view only includes employees in department 51.
4. The user query inserts a new employee in department 80 into the view table.
5. The new employee is inserted into Employee but does not appear in the view table.
6. WITH CHECK OPTION prevents inserts and updates that do not satisfy the view query WHERE clause.

PARTICIPATION ACTIVITY 36.8.10: WITH CHECK OPTION clause.

Refer to the following base tables, statement, and view table.

Faculty

• ID	FacultyName	Code
1	Grayson	ART
2	Wayne	ART
3	Stark	COMP
4	Parker	MATH
5	Banner	MATH
6	Quinn	MATH
7	Grey	NULL

Department

• Code	DepartmentName
ART	Art Department
COMP	Computer Science Department
ENG	English Department
HIST	History Department
MATH	Math Department


```
CREATE VIEW ___A___ (EmployeeNumber, ___B___, Dept)
AS SELECT ___C___, FacultyName, DepartmentName
   FROM Faculty, Department
   WHERE Faculty.Code = Department.Code
WITH ___D___;
```

FacultyDirectory

EmployeeNumber	Name	Dept
1	Grayson	Art Department
2	Wayne	Art Department
3	Stark	Computer Science Department
4	Parker	Math Department
5	Banner	Math Department
6	Quinn	Math Department

1) What is table name A?

Check Show answer

2) What is column name B?

Check Show answer

3) What is column name C?

Check Show answer

4) What are keywords D?

Check Show answer

CHALLENGE ACTIVITY 36.8.1: View tables.

539740.3879454.qx3zqy7

Exploring further:

- MySQL CREATE VIEW statement
- Using MySQL views

36.9 Relational algebra

Operations and expressions

In his original paper on the relational model, E. F. Codd introduced formal operations for manipulating tables. Codd's operations, called ***relational algebra***, have since been refined and are the theoretical foundation of SQL. Relational algebra is primarily of theoretical interest but does have practical application in compiling SQL queries.

Relational algebra has nine operations. Each operation is denoted with a special symbol, often a letter of the Greek alphabet. Operation symbols can be combined with tables in expressions, just as + - × / can be combined with numbers in arithmetic expressions. Each relational algebra expression is equivalent to an SQL query and defines a single result table.

Table 36.9.1: Relational algebra operations.

Operation	Symbol	Greek letter	Derivation
Select	σ	sigma	corresponds to Latin letter s, for Select
Project	Π	Pi	corresponds to Latin letter P, for Project
Product	$\times$		multiplication symbol
Join	$\bowtie$		multiplication symbol with vertical bars
Union	$\cup$		set theory
Intersect	$\cap$		set theory
Difference	–		set theory
Rename	ρ	rho	corresponds to Latin letter r, for Rename
Aggregate	γ	gamma	corresponds to Latin letter g, for group

PARTICIPATION ACTIVITY 36.9.1: Relational algebra operations.

Match the symbol to the operation name.

If unable to drag and drop, refresh the page.

σ $\bowtie$ $-$ ρ $\times$ Π $\cap$ $\cup$ γ

	Select
	Project
	Product
	Join
	Union
	Intersect
	Difference
	Rename
	Aggregate

Reset

Additional operations

*Relational algebra is not standardized. Some authors include additional operations, such as **assign** and **divide**. Variations of the join operation, such as inner join, full join, and equijoin, are sometimes considered distinct operations.*

Select and project

The ***select operation*** selects table rows based on a logical expression. The select operation is written as

$$\sigma_{\text{expression}}(\text{Table})$$

and is equivalent to `SELECT * FROM Table WHERE expression`.

The ***project operation*** selects table columns. The project operation is written as

$$\Pi_{(\text{Column1}, \text{Column2}, \ldots)}(\text{Table})$$

and is equivalent to `SELECT Column1, Column2, ... FROM Table`.

PARTICIPATION ACTIVITY 36.9.2: Select and project operations.

Employee

ID	Name	Salary	Bonus
2538	Lisa Ellison	115000	0
5384	Sam Snead	32000	55000
6381	Maria Rodriguez	95000	3000
8820	Jiho Chen	48000	8000

$$\Pi_{(\text{Name},\ \text{Salary})}(\ \sigma_{(\text{Salary}>50000)}(\text{Employee})\quad)$$

Result

Name	Salary
Lisa Ellison	115000
Maria Rodriguez	95000

Animation content:

Static figure:
The Employee table has columns ID, Name, Salary, and Bonus. ID is the primary key. Employee has four rows:
2538, Lisa Ellison, 115000, 0
5384, Sam Snead, 32000, 55000
6381, Maria Rodriguez, 95000, 3000
8820, Jiho Chen, 48000, 8000

Below the Employee table is a relational algebra expression. The expression is Pi, then a subscript (Name, Salary), then a large open parentheses. Inside the large open parentheses is sigma, then a subscript (Salary greater than 50000), then Employee, then a large close parenthesis.

The Result table appears below the expression, with columns Name and Salary. Result has 2 rows:
Lisa Ellison, 115000
Maria Rodriguez, 95000

Step 1: Operations can be combined in expressions. Employee table appears. The relational algebra expression appears below the Employee table.

Step 2: The select operation selects employees who earn more than $50,000 in salary. Pi subscript (Name, Salary) fades out. Sigma subcript (Salary greater than 50000) parenthesis Employee is highlighted. Result table appears below the expression.

Step 3: The project operation selects the Name and salary Columns from the result. Pi subscript (Name, Salary) fades in and is highlighted. Sigma subscript (Salary greater than 50000) parenthesis Employee is unhighlighted.

Animation captions:

1. Operations can be combined in expressions.
2. The select operation selects employees who earn more than $50,000 in salary.
3. The project operation selects the Name and Salary columns from the result.

PARTICIPATION ACTIVITY 36.9.3: Select and project operations.

Refer to the Employee table in the above animation. Indicate whether the following expression pairs define the same table.

1) $\Pi_{(\text{Name},\ \text{Salary})}(\sigma_{(\text{Salary} > 50000)}(\text{Employee}))$

$\sigma_{(\text{Salary} > 50000)}(\Pi_{(\text{Name},\ \text{Salary})}(\text{Employee}))$

- True
- False

2) $\sigma_{(\text{Bonus} > 10000)}(\sigma_{(\text{Salary} > 50000)}(\text{Employee}))$

$\sigma_{(\text{Bonus} > 10000\ \text{AND}\ \text{Salary} > 50000)}(\text{Employee})$

- True
- False

3) $\Pi_{(\text{Name},\ \text{Salary})}(\sigma_{(\text{Salary} > 50000)}(\text{Employee}))$

$\Pi_{(\text{Name})}(\sigma_{(\text{Salary} > 50000)}(\Pi_{(\text{Salary})}(\text{Employee})))$

- True
- False

Product and join

The ***product operation*** combines two tables into one result. The result includes all columns and all combinations of rows from both tables. The product operation is written as

$$\text{Table1} \times \text{Table2}$$

and is equivalent to `SELECT * FROM Table1 CROSS JOIN Table2`. If Table1 has n_1 rows and Table2 has n_2 rows, then the product has $n_1 \times n_2$ rows.

The ***join*** operation, denoted with a "bowtie" symbol, is written as

$$\text{Table1} \bowtie_{\text{expression}} \text{Table2}$$

and is identical to a product followed by a select:

$$\sigma_{\text{expression}}(\text{Table1} \times \text{Table2})$$

The join operation is equivalent to `SELECT * FROM Table1 INNER JOIN Table2 ON expression`. The join expression is any expression on columns of Table1 and Table2, and is often denoted with the Greek letter theta:

$$\text{Table1} \bowtie_{\theta} \text{Table2}$$

Because of theta notation, the join operation is sometimes called a ***theta join***.

The bowtie symbol ⋈ represents an inner join operation. Left, right, and full joins are represented by symbols ⟕ , ⟖ , and ⟗ , respectively.

PARTICIPATION ACTIVITY | 36.9.4: Product and join operations.

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
5348	Sam Snead	30500
6381	Maria Rodriguez	92300

$$\Pi_{(\text{DepartmentName}, \text{EmployeeName})} (\text{Department} \bowtie_{\text{Department.Manager} = \text{Employee.ID}} \text{Employee})$$

Result

DepartmentName	EmployeeName
Engineering	Lisa Ellison
Sales	Maria Rodriguez
Marketing	Maria Rodriguez

Animation content:

Step 1: The product operation generates the combination of all rows from both tables. There are two tables named Department and Employee. Department has three columns named Code, Department Name, and Manager, with Code as the primary key, and Manager as a foreign key. Employee has three columns named ID, Employee Name, and Salary, with ID as the primary key. A line with "Department x Employee" (the product operation) appears. A result table starts appearing. The result table has 6 columns: the left 3 columns are the same as the Department table, and the right 3 columns are the same as the Employee table. The four rows of the Department table are repeated 3 times under the left 3 columns of the result table. (The rows of the Department table are: row 1: 44, Engineering, 2538, row 2: 82, Sales, 6381, row 3: 12, Marketing, 6381, row 4: 99, Technical support, NULL). The right 3 columns of the result table are as follows: the first row of the Employee table (2538, Lisa Ellison, 45000) is copied to the result table, and repeated 4 times. Then below those four lines, the second row of the Employee table (5348, Sam Snead, 30500) is copied to result table and repeated 4 times. Then below those lines, the third row of the Employee table (6381, Maria Rodriguez, 92300) is copied to result table and repeated 4 times.

Step 2: The select operation selects rows with Manager = ID. The product operation changes to the following expression: SELECT Department.Manager=Employee.ID(Department x Employee). Then, 9 rows of the Result table disappear, leaving 3 rows. These 3 rows are the rows where the value in the Manager column is equal to the value in the ID column. The three rows are: (row 1: 44, Engineering, 2538, 2538, Lisa Ellison, 45000, row 2: 82, Sales, 6381, 6381, Maria Rodriguez, 92300, row 3: 12, Marketing, 6381, 6381, Maria Rodriguez, 92300).

Step 3: A product followed by a select is equivalent to a join operation. The expression changes to Department (join Department.Manager = Employee.ID) Employee. The 'join' symbol in the expression looks like a bowtie.

Step 4: Commonly, a join operation is followed by a project. The expression changes to Pi (DepartmentName,EmployeeName)(Department (join Department.Manager = Employee.ID) Employee). The uppercase Pi symbol is the project symbol. Then, four columns of the result table disappear. The two columns remaining are the DepartmentName and EmployeeName columns. The 3 rows of the result table are: (row 1: Engineering, Lisa Ellison, row 2: Sales, Maria Rodriguez, row 3: Marketing, Maria Rodriguez).

Animation captions:

1. The product operation generates the combination of all rows from both tables.
2. The select operation selects rows with Manager = ID
3. A product followed by a select is equivalent to a join operation.
4. Commonly, a join operation is followed by a project.

PARTICIPATION ACTIVITY 36.9.5: Product and join operations.

Refer to the tables below.

Department

• Code	DepartmentName
ART	Art and Architecture
COMP	ComputerScience
ENG	English

Course

• CourseNumber	Code	CourseName
101	ART	Art History
341	ART	Oil Painting
200	COMP	Data Structures
201	COMP	Computer Architecture

1) How many rows are in the table defined by the following expression?

$$\text{Department} \times \text{Course}$$

- 4
- 7
- 12

2) What is the result of the following expression?

$$\text{Department} \bowtie_{\text{Department.Code}=\text{Course.Code}} \text{Course}$$

Department.Code	DepartmentName	CourseNumber	CourseName
ART	Art and Architecture	101	Art History
ART	Art and Architecture	341	Oil Painting
COMP	Computer Science	200	Data Structures
COMP	Computer Science	201	Computer Architecture

Department.Code	DepartmentName	CourseNumber	Course.Code	CourseName
ART	Art and Architecture	101	ART	Art History
ART	Art and Architecture	341	ART	Oil Painting
COMP	Computer Science	200	COMP	Data Structures
COMP	Computer Science	201	COMP	Computer Architecture

Department.Code	DepartmentName	CourseNumber	Course.Code	CourseName
ART	Art and Architecture	101	ART	Art History
ART	Art and Architecture	341	ART	Oil Painting
COMP	Computer Science	200	COMP	Data Structures
COMP	Computer Science	201	COMP	Computer Architecture
ENG	English	NULL	NULL	NULL

3) How many course names are selected by the following expression?

$$\Pi_{(\text{CourseName})}(\sigma_{(\text{DepartmentName}=\text{``Art and Architecture''})}(\text{Department} \bowtie_{\text{Department.Code}=\text{Course.Code}} \text{Course}))$$

- 1
- 2
- 4

Union, intersect, and difference

Compatible tables have the same number of columns with the same data types. Column names may be different. Union, intersect, and difference operate on compatible tables and, collectively, are called ***set operations***.

The ***union*** operation combines all rows of two compatible tables into a single table. Duplicate rows are excluded from the result table. The union operation is written as

$$\text{Table1} \cup \text{Table2}$$

and is equivalent to `SELECT * FROM Table1 UNION SELECT * FROM Table2`.

Intersect operates on two compatible tables and returns only rows that appear in both tables. The intersect operation is written as

$$\text{Table1} \cap \text{Table2}$$

and is equivalent to `SELECT * FROM Table1 INTERSECT SELECT * FROM Table2`.

The ***difference*** operation removes from a table all rows that appear in a second compatible table. The difference operation is written as

$$\text{Table1} - \text{Table2}$$

and is equivalent to `SELECT * FROM Table1 MINUS SELECT * FROM Table2`.

For all three set operations, the result column names are the Table1 column names.

PARTICIPATION ACTIVITY 36.9.6: Union operation.

Employee

ID	Name	Compensation
2538	Lisa Ellison	115000
5348	Sam Snead	35000
6381	Maria Rodriguez	95000

Student

ID	Name	Scholarship
4776	Malia Akembo	1500
4990	Patrick O'Leary	3000
6381	Maria Rodriguez	0
9565	Pranav Alur	4250

$$\text{Employee} \cup \text{Student}$$

Result

ID	Name	Compensation
2538	Lisa Ellison	115000
5348	Sam Snead	35000
6381	Maria Rodriguez	95000
4776	Malia Akembo	1500
4990	Patrick O'Leary	3000
6381	Maria Rodriguez	0
9565	Pranav Alur	4250

Animation content:

Step 1: A university has employee and student tables. The tables are compatible even though the third column names are different. There are two tables named Employee and Student. Employee has 3 columns: ID, Name, and Compensation, with ID as the primary key. Compensation is highlighted in red. Employee has 3 rows. Student has 3 columns: ID, Name, and Scholarship, with ID as the primary key. Scholarship is highlighted in red. Student has 4 rows.

Step 2: The compatible tables can be combined in a union operation. An expression appears: Employee U Student. The U is the union operation.

Step 3: The result column names are taken from the first table in the union. A table called Result appears. Result has 3 columns: ID, Name, and Compensation, with ID as the primary key. The Result table in this step is empty.

Step 4: Union combines rows of the two tables in the result. The 3 rows of the Employee table are

copied to the top of the Result table. Then the 4 rows of the Student table are copied to the Result table below the first 3 rows.

Step 5: Union eliminates duplicate rows. But both Maria Rodriguez rows remain, since compensation values are different. The two rows containing the name Maria Rodriguez are highlighted. One Maria Rodriguez row is from the Employee table and one is from the Student table. Maria Rodriguez has the same ID in both tables, but has a compensation of 95000 in the Employee table and 0 in the Student table.

Animation captions:

1. A university has employee and student tables. The tables are compatible even though the third column names are different.
2. The compatible tables can be combined in a union operation.
3. The result column names are taken from the first table in the union.
4. Union combines rows of the two tables in the result.
5. Union eliminates duplicate rows. But both Maria Rodriguez rows remain, since compensation values are different.

PARTICIPATION ACTIVITY 36.9.7: Union, intersect, and difference operations.

Refer to the tables in the above animation.

1) How many rows are in the table defined by the following expression?

$$\text{Student} \cup \text{Student}$$

- 0
- 4
- 8

2) How many rows are in the table defined by the following expression?

$$\text{Employee} \cap \text{Student}$$

- 0
- 1
- 7

3) How many rows are in the table defined by the following expression?

$$\text{Employee} - \text{Student}$$

- 0
- 2
- 3

4) Maria Rodriguez takes an unpaid leave. She continues as an employee, but her salary is now 0. How many rows are in the table defined by the following expression?

$$\text{Employee} - \text{Student}$$

- 0
- 2
- 3

Set operations in MySQL

The UNION, INTERSECT, and MINUS keywords are part of the SQL standard. MySQL supports UNION but not INTERSECT and MINUS. In MySQL, the intersect and difference operations can be implemented as joins. Ex: Table A has columns A1 and A2. Table B has columns B1 and B2.

$A \cap B$ is implemented as

```
SELECT A1, A2
FROM  A
INNER JOIN B
ON (A1 = B1 AND A2 = B2);
```

$A - B$ is implemented as

```
SELECT A1, A2
FROM  A
LEFT JOIN B
ON (A1 = B1 AND A2 = B2)
WHERE B1 IS NULL;
```

Rename and aggregate

The ***rename operation*** specifies new table and column names. The rename operation is written as

$$\rho_{\text{TableName}(\text{ColumnName1}, \text{ColumnName2}, \ldots)}(\text{Table})$$

If TableName is omitted, only the column names are changed. If (ColumnName1, ColumnName2, ...) is omitted, only the table name is changed.

PARTICIPATION ACTIVITY 36.9.8: Rename operation.

Employee

ID	Name	ManagerID
2538	Lisa Ellison	8820
5384	Sam Snead	8820
6381	Maria Rodriguez	8820
8820	Jiho Chen	9107
9107	Pranav Alur	NULL

\rho_{(\text{Name1},\:\text{Name2})}(\Pi_{\text{Employee.Name},\:\text{Manager.Name}}(

\:\text{Employee}\bowtie_{\text{ Employee.ManagerID = Manager.ID }}\rho_{\text{Manager}}(\text{Employee})\)

```
SELECT Employee.Name, Manager.Name AS Supervisor
FROM Employee
INNER JOIN Employee AS Manager
ON Employee.ManagerID = Manager.ID;
```

Result

Name	Supervisor
Lisa Ellison	Jiho Chen
Sam Snead	Jiho Chen
Maria Rodriguez	Jiho Chen
Jiho Chen	Pranav Alur

Animation content:

Static figure:
The Employee table has columns ID, Name, and ManagerID. ID is the primary key. An arrow from ManagerID points to ID. Employee has five rows:
2538, Lisa Ellison, 8820
5384, Sam Snead, 8820
6381, Maria Rodriguez, 8820
8820, Jiho Chen, 9107
9107, Pranav Alur, NULL

A relational algebra expression appears below the Employee table. The expression contains operations rho, large open parenthesis, pi, large open parenthesis, bowtie, large closed parenthesis, large closed parenthesis. The rho operation has a subscript (Name, Supervisor). The pi operation has a subscript (Employee.Name, Manager.Name). The bowtie operation has Employee on the left, a subscript (Employee.ManagerID = Manager.ID), and another rho operation on the right. The second rho operation has a subscript (Manager), followed by Employee.

An SQL statement appears:
Begin SQL code:
SELECT Employee.Name, Manager.Name AS Supervisor
FROM Employee
INNER JOIN Employee AS Manager
ON Employee.ManagerID = Manager.ID;
End SQL code.

The Result table appears below the statement, with columns Name and Supervisor. Result has four rows:
Lisa Ellison, Jiho Chen
Sam Snead, Jiho Chen
Maria Rodriguez, Jiho Chen
Jiho Chen, Pranav Alur

Step 1: The ManagerID column is a foreign key that refers to the ID column. The Employee table appears.

Step 2: To list the names of employees and their managers, Employee is renamed Manager and joined to itself. The relational algebra expression line Employee bowtie subscript Employee.ManagerID=Manager.ID parenthesis rho subscript Manager (Employee) appears. Rho

subscript Manager and outer parentheses are highlighted.

Step 3: The result is projected onto two columns. Manager.Name is renamed Supervisor. The line rho(Name, Supervisor)(Pi subscript (Employee.Name, Manager.Name) appears above Employee in the relational algebra statement. Rho(Name, Supervisor) are highlighted.

Step 4: The expression is equivalent to a self-join query. The SQL statement appears.

Step 5: Since the query is an inner join, employees with a NULL manager do not appear. The Result table appears below the SQL statement. The row 9107, Pranav Alur, NULL is highlighted in the Employee table.

Animation captions:

1. The ManagerID column is a foreign key that refers to the ID column.
2. To list the names of employees and their managers, Employee is renamed Manager and joined to itself.
3. The result is projected onto two columns. Manager.Name is renamed Supervisor.
4. The expression is equivalent to a self-join query.
5. Since the query is an inner join, employees with a NULL manager do not appear.

The ***aggregate operation*** applies aggregate functions like SUM(), AVG(), MIN(), and MAX(). The aggregate operation is written as

$$_{\text{GroupColumn}}\gamma_{\text{Function(Column)}}(\text{Table})$$

and is equivalent to `SELECT GroupColumn, Function(Column) FROM Table GROUP BY GroupColumn`.

If GroupColumn is omitted, the operation is equivalent to `SELECT Function(Column) FROM Table` and computes a single aggregate value for all rows.

PARTICIPATION ACTIVITY | 36.9.9: Aggregate operation.

Employee

ID	Name	Salary	DepartmentCode
2538	Lisa Ellison	115000	ENG
5384	Sam Snead	35000	MKTG
6381	Maria Rodriguez	95000	MKTG
8820	Jiho Chen	48000	ENG
9107	Pranav Alur	67000	ENG

$$\gamma_{\text{DepartmentCode}}\text{SUM(Salary)}(\text{Employee})$$

```
SELECT DepartmentCode, SUM(Salary)
FROM Employee
GROUP BY DepartmentCode;
```

Result

DepartmentCode	SUM(Salary)
ENG	230000
MKTG	130000

Animation content:

Static figure:
An Employee table has columns ID, Name, Salary, and DepartmentCode. A relational algebra expression is below the Employee table. The expression is subscript (DepartmentCode), gamma, subscript (SUM(Salary)), Employee. The equivalent SQL statement is below the expression. The statement is SELECT DepartmentCode, SUM(Salary) FROM Employee GROUP BY DepartmentCode. The Result table is below the statement. The Result table has columns DepartmentCode and SUM(Salary).

Animation captions:

1. The aggregate operation sums all salaries in Employee.
2. The operation is equivalent to a query with the SUM() function.
3. Adding DepartmentCode to the operation sums salaries by department.
4. With DepartmentCode, the operation is equivalent to a GROUP BY query.

PARTICIPATION ACTIVITY 36.9.10: Rename and aggregate operations.

Refer to the Employee table in the above animation.

1) What does the following expression do?

$$\rho_{(\text{ID, Name, Compensation})}(\text{Employee})$$

- Computes total employee compensation.
- Renames the Employee table to Compensation.
- Renames the Salary column to Compensation.

2) What is the result of the following expression?

$$\sigma_{(\text{Salary} > 50000)}(\rho_{(\text{Salary})}(\gamma_{\text{SUM(Salary)}}(\text{Employee})))$$

DepartmentCode	Salary
ENG	230000
MKTG	130000
Salary
277000
Salary
360000

3) What is the result of the following expression?

$$\rho_{(\text{Salary})}(\gamma_{\text{SUM(Salary)}}(\sigma_{(\text{Salary}>50000)}(\text{Employee})))$$

DepartmentCode	Salary
ENG	230000
MKTG	130000
Salary
277000
Salary
360000

4) The expressions in questions 2 and 3 have the same operations in different order:

$$\sigma_{(\text{Salary}>50000)}(\rho_{(\text{Salary})}(\gamma_{\text{SUM(Salary)}}(\text{Employee})))$$

$$\rho_{(\text{Salary})}(\gamma_{\text{SUM(Salary)}}(\sigma_{(\text{Salary}>50000)}(\text{Employee})))$$

Do these expressions define the same table?

- Yes
- No

Notation

Notation for the aggregate operation varies. In this material, the aggregate operation is denoted with the Greek letter gamma (γ). Some publications use the Latin letter G, the script F (ℱ), or the script A (𝒜).

Query optimization

Relational algebra expressions are ***equivalent*** if the expressions operate on the same tables and generate the same result. Ex:

$$\Pi_{(\text{ID},\ \text{Salary})}(\sigma_{(\text{Salary}>50000)}(\text{Employee}))$$

is equivalent to

$$\sigma_{(\text{Salary}>50000)}(\Pi_{(\text{ID},\ \text{Salary})}(\text{Employee}))$$

Although the operation order is different, both expressions operate on Employee and define the same result table.

A ***query optimizer*** converts an SQL query into a sequence of low-level database actions, called the ***query execution plan***. The query execution plan specifies precisely how to process an SQL statement. A query optimizer is similar to a programming language compiler.

Query optimizers use equivalent expressions to optimize query execution. The query optimizer:

1. Converts a query to a relational algebra expression.
2. Generates equivalent expressions.
3. Estimates the 'cost' of each operation of each expression.
4. Determines the optimal expression with the lowest total cost.
5. Converts the optimal expression into a query execution plan.

The ***cost*** of an operation is a numeric estimate of processing time. The cost estimate usually combines both storage media access and computation time in a single measure. The animation below illustrates cost as the number of rows read. In practice, query optimizers use more sophisticated cost measures.

PARTICIPATION ACTIVITY | 36.9.11: Query optimization.

```
SELECT *
FROM Department
INNER JOIN Employee
ON Department.Manager = Employee.ID
AND EmployeeName = "Lisa Ellison";
```

$\sigma_{(\text{EmployeeName}=\text{``Lisa Ellison''})}(\ \sigma_{(\text{Department.Manager}=\text{Employee.ID})}\ (\text{Department} \times \text{Employee})\ )$

22 rows

$\sigma_{(\text{Department.Manager}=\text{Employee.ID})}\ (\text{Department} \times (\ \sigma_{(\text{EmployeeName}=\text{``Lisa Ellison''})}\ \text{Employee})\ )$

12 rows

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
5348	Sam Snead	30500
6381	Maria Rodriguez	92300

Animation content:

Step 1: The query selects departments managed by Lisa Ellison. The following SQL query appears: SELECT * FROM Department INNER JOIN Employee ON Department.Manager = Employee.ID AND EmployeeName = Lisa Ellison;

Step 2: In relational algebra, the query is a joined followed by a select. The following expression appears: select(EmployeeName=Lisa Ellison)(Department join(Department.Manager=Employee.ID) Employee)

Step 3: The join operation is equivalent to a product followed by a select. The expression turns into: select(EmployeeName=Lisa Ellison)(select(Department.Manager=Employee.ID) (Department x Employee))

Step 4: The product reads 4 Department rows and 3 Employee rows. The first select reads 12 rows. The second select reads 3 rows. The Department table appears with 3 columns: Code, DepartmentName, and Manager. The Department table has 4 rows. The Employee table appears with 3 columns: ID, EmployeeName, and Salary. The Employee table has 3 rows. The Department x Employee part of the expression is highlighted. The 4 rows of the Department table and 3 rows of the Employee table are read; then 4 rows + 3 rows appears below Department x Employee. The Result table appears. The Result table is the result of the product operation of Department x Employee. Next, the select(Department.Manager=Employee.ID) part of the expression is highlighted in red. Then the 12 rows of the Result table are read, and 12 rows appears below the select(Department.Manager=Employee.ID). Then the select operation is performed on the Result table. The Result table now has 3 rows. Next, the select(EmployeeName=Lisa Ellison) part of the expression is highlighted. Then, the 3 rows of the Result table are read, and 3 rows appears below select(EmployeeName=Lisa Ellison). Next, this select operation is performed on the Result table. The Result table is now 1 row.

Step 5: An equivalent expression first selects Lisa Ellison from Employee. A new expression appears below the previous expression: select(Department.Manager=Employee.ID)(Department x (select(EmployeeName=Lisa Ellison) Employee)).

Step 6: The first select reads 3 rows. The product reads 4 department rows and 1 Employee row. The second select reads 4 rows. The select(EmployeeName=Lisa Ellison) part of the second expression is highlighted. The same tables as before appear - Department with 4 rows and Employee with 3 rows. The 3 rows of the Employee table are read, and 3 rows appears below select(EmployeeName=Lisa Ellison). The select operation is performed. The Employee table now has 1 row. The Department x Employee part of the second expression is highlighted. The 4 rows of the Department table and the 1 row of the Employee table are read, and 4 rows + 1 row appears below Department x in the expression. The product operation is performed, and the Result table appears, with 4 rows. Next, the select(Department.Manager=Employee.ID) part of the second expression is highlighted. Then, the 4 rows of the Result table are read. Then the select operation is performed, and the Result table now has 1 row.

Step 7: The second expression costs less than the first and is used for the query execution plan. The following equation appears below the first expression: 3 + 12 + 4 + 3 = 22 rows. Then, the following equation appears below the second expression: 4 + 4 + 1 + 3 = 12 rows.

Animation captions:

1. The query selects departments managed by Lisa Ellison.
2. In relational algebra, the query is a join followed by a select.
3. The join operation is equivalent to a product followed by a select.
4. The product reads 4 Department rows and 3 Employee rows. The first select reads 12 rows. The second select reads 3 rows.
5. An equivalent expression first selects Lisa Ellison from Employee.
6. The first select reads 3 rows. The product reads 4 Department rows and 1 Employee row. The second select reads 4 rows.
7. The second expression costs less than the first and is used for the query execution plan.

PARTICIPATION ACTIVITY 36.9.12: Query optimization.

1) Changing the order of operations can alter the result of an expression.

- True
- False

2) Exactly one query execution plan is possible for each SQL query.

- True
- False

3) Query optimizers typically choose an optimal expression based on total number of rows processed.

- True
- False

4) The join operation is a low-level database action.

- True
- False

CHALLENGE ACTIVITY 36.9.1: Relational algebra.

539740.3879454.qx3zqy7

Exploring further:

- Relational algebra (Wikipedia)
- Original paper on the relational model, by E. F. Codd

36.10 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.

539740.3879454.qx3zqy7

LAB ACTIVITY 36.10.1: LAB - Select number of movies grouped by year 0 / 10

Main.sql Load default template...

```
1 -- Your SELECT statement goes here
2
```


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?

```
History of your effort will appear here once you begin working
on this zyLab.
```

36.11 LAB - Select movie ratings with left join

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

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

- **Code** - variable-length string, primary key
- **Description** - variable-length string

Write a SELECT statement to select the Title, Year, and rating Description. Display all movies, whether or not a RatingCode is available.

Hint: Perform a LEFT JOIN on the Movie and Rating tables, matching the RatingCode and Code columns.

539740.3879454.qx3zqy7

LAB ACTIVITY | 36.11.1: LAB - Select movie ratings with left join | 0 / 10

Main.sql — Load default template...

```
1 -- Your SELECT statement goes here
2
```


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?

```
History of your effort will appear here once you begin working
on this zyLab.
```

36.12 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.

539740.3879454.qx3zqy7

LAB ACTIVITY | 36.12.1: LAB - Select employees and managers with inner join | 0 / 10

Main.sql Load default template...

```
1 -- Your SELECT statement goes here
2
```

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?

```
History of your effort will appear here once you begin working
on this zyLab.
```


36.13 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.

539740.3879454.qx3zqy7

LAB ACTIVITY | 36.13.1: LAB - Select lesson schedule with inner join | 0 / 10

Main.sql — Load default template...

```
1 -- Your SELECT statement goes here
2
```


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?

```
History of your effort will appear here once you begin working
on this zyLab.
```


36.14 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.

539740.3879454.qx3zqy7

LAB ACTIVITY | 36.14.1: LAB - Select lesson schedule with multiple joins | 0 / 10

Main.sql Load default template...

```
1 -- Your SELECT statement goes here
2
```

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?

```
History of your effort will appear here once you begin working
on this zyLab.
```

36.15 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.

539740.3879454.qx3zqy7

LAB ACTIVITY | 36.15.1: LAB - Select tall horses with subquery 0 / 10

Main.sql Load default template...

```
1 -- Your SQL statements go here
```

```
2
```


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?

```
History of your effort will appear here once you begin working
on this zyLab.
```

36.16 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.

LAB ACTIVITY | 36.16.1: LAB - Multiple joins with aggregate (Sakila) 0 / 10

Main.sql Load default template...

```
1 -- Your SELECT statment goes here
2
```


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?

```
History of your effort will appear here once you begin working
on this zyLab.
```

36.17 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.

LAB ACTIVITY | 36.17.1: LAB - Nested aggregates (Sakila) | 0 / 10

Main.sql — Load default template...

```
1 -- Your SELECT statement goes here
2
3
4 -- Use the following subquery:
5   ( SELECT MIN(count_film_id) FROM
6       ( SELECT COUNT(film_id) AS count_film_id
7         FROM inventory
8         GROUP BY film_id )
9     AS temp_table );
```


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?

History of your effort will appear here once you begin working on this zyLab.

©zyBooks 01/31/24 18:21 1939727
Rob Daglio
MDCCOP2335Spring2024