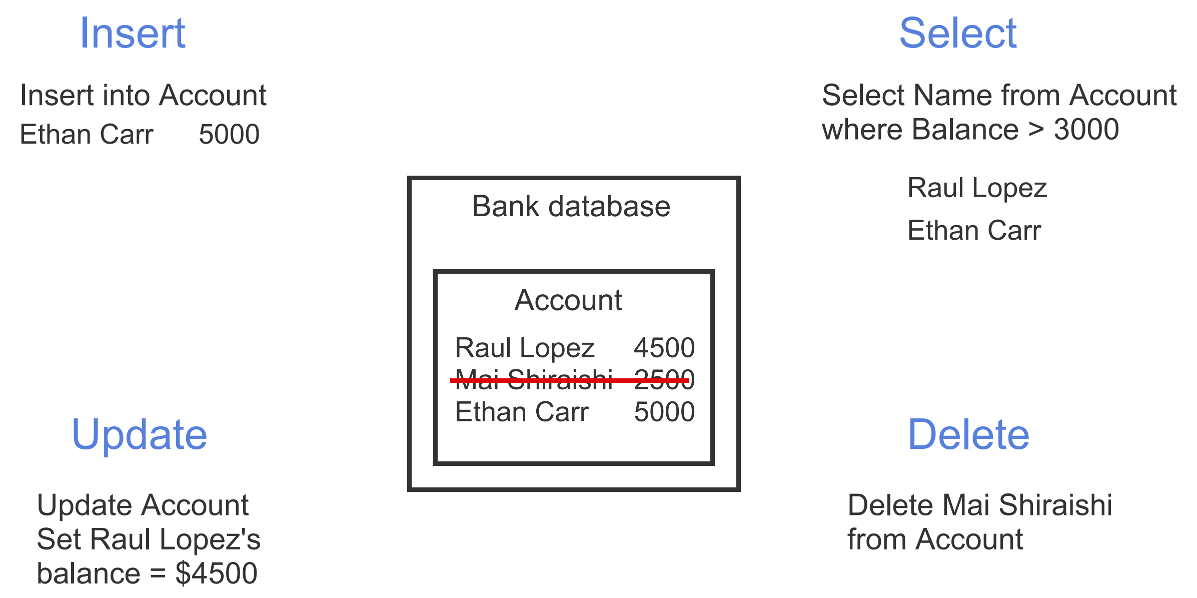
8.1 Query languages

Common queries

A database system responds to queries written in a query language. A ***query*** is a command for a database that typically inserts new data, retrieves data, updates data, or deletes data from a database. A ***query language*** is a computer programming language for writing database queries.

PARTICIPATION ACTIVITY 8.1.1: Insert, select, update, and delete database queries.



Animation content:

Step 1: A bank database stores the names and balances for two accounts: Raul and Mai. There is a box named Bank database. In this box is another box named Account. Account has the names and balances for two people.

Step 2: An insert query inserts new data into the database. Ethan's new account is inserted into the database. Text appears and states Insert into account Ethan Carr 5000. The text Ethan Carr 5000 appears in the box Account.

Step 3: A select query retrieves information from the database. The query retrieves the names of individuals that have a balance more than $3000. New text appears and states Select name from Account where Balance is greater than 3000. The names Raul Lopez and Ethan Carr are highlighted with balances 3300 and 5000 respectively. The names appear below the text.

Step 4: An update query changes existing data in the database. Raul's balance is changed from 3300 to 4500. Text appears and states Update Account Set Raul Lopez's balance equals 4500 dollars. The number for Raul Lopez in the account box is changed from 300 to 4500.

Step 5: A delete query removes data from the database. Mai's account is removed. Text appears and states Delete Mai Shiraishi from Account. Mia Shirashi 2500 is crossed out in box Account.

Animation captions:

1. A bank database stores the names and balances for two accounts: Raul and Mai.
2. An insert query inserts new data into the database. Ethan's new account is inserted into the database.
3. A select query retrieves information from the database. The query retrieves the names of individuals that have a balance more than $3000.
4. An update query changes existing data in the database. Raul's balance is changed from 3300 to 4500.
5. A delete query removes data from the database. Mai's account is removed.

Terminology

*The four common queries are sometimes referred to as **CRUD** operations, an acronym for Create, Read, Update, and Delete data.*

PARTICIPATION ACTIVITY 8.1.2: Queries.

Refer to the animation above.

1) Only one of the queries does not change the database contents.

○ True

○ False

2) Given the data in the Bank database,

a select query for accounts with negative balances would return nothing.

- ○ True
- ○ False

3) The insert, select, update, and delete queries are the only types of commands necessary to interact with a database system.

- ○ True
- ○ False

4) An update query cannot update data that isn't in the database.

- ○ True
- ○ False

Writing queries with SQL

Structured Query Language, or ***SQL***, is the standard query language of relational database systems. The SQL standard is sponsored by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). SQL is pronounced either 'S-Q-L' or 'seekwəl', but the preferred pronunciation is 'S-Q-L'.

SQL was first developed at IBM in the 1970s as an experimental query language for a prototype relational database. At the time, IBM was the dominant computer company, so SQL became the dominant relational query language. Today, all relational database systems support SQL.

Terminology

*The term **NoSQL** refers to a new generation of non-relational databases. NoSQL originally meant 'does not support SQL'. However, many NoSQL databases have added support for SQL, and 'NoSQL' has come to mean 'not only SQL'.*

An SQL ***statement*** is a database command, such as a query that inserts, selects, updates, or deletes data:

- ***INSERT*** inserts rows into a table.

- ***SELECT*** retrieves data from a table.
- ***UPDATE*** modifies data in a table.
- ***DELETE*** deletes rows from a table.

The SQL language contains many other statements for creating and deleting databases, creating and deleting tables, assigning user permissions, and so on.

PARTICIPATION ACTIVITY | 8.1.3: SQL statements: INSERT, SELECT, UPDATE, and DELETE.

Animation captions:

PARTICIPATION ACTIVITY | 8.1.4: SQL statements.

Refer to the Account table below.

Account

ID	Name	Balance
831	Raul Lopez	3300
572	Mai Shiraishi	2500
290	Ethan Carr	5000

1) What is Braden Smith's balance in the following INSERT statement?

```
INSERT INTO Account
VALUES (800, 'Braden Smith',
200);
```

- ○ 800
- ○ 200
- ○ Unknown

2) Which name is retrieved by the following SELECT statement?

```
SELECT Name
FROM Account
WHERE Balance < 3000;
```

- ○ Raul Lopez
- ○ Mai Shiraishi
- ○ Ethan Carr

3) Whose balance does the following UPDATE statement change?

```
UPDATE Account
SET Balance = 850
WHERE ID = 290;
```

- ○ Raul Lopez
- ○ Mai Shiraishi
- ○ Ethan Carr

4) Who is deleted by the following DELETE statement?

```
DELETE Account
WHERE ID = 999;
```

- ○ Mai Shiraishi
- ○ No one
- ○ Everyone

CHALLENGE ACTIVITY | 8.1.1: Query languages.

544874.3500394.qx3zqy7

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

Creating tables with SQL

The SQL ***CREATE TABLE*** statement creates a new table by specifying the table and column names. Each column is assigned a ***data type*** that indicates the format of column values. Data types can be numeric, textual, or complex. Ex:

- INT stores integer values.
- DECIMAL stores fractional numeric values.
- VARCHAR stores textual values.
- DATE stores year, month, and day.

Some data types are followed by one or two numbers in parentheses, indicating the size of the data type. Ex: VARCHAR(10) indicates ten characters. DECIMAL(10, 3) indicates ten significant digits, including three after the decimal point.

PARTICIPATION ACTIVITY 8.1.5: Creating an Employee table.

Animation captions:

PARTICIPATION ACTIVITY | 8.1.6: Creating tables with SQL.

Refer to the animation above.

1) The Employee table is created with 4 different data types.

○ True

○ False

2) Only the ID column stores numbers.

○ True

○ False

3) The BirthDate column stores only a date and no time.

○ True

○ False

4) The Employee table is an empty table once created.

○ True

○ False

PARTICIPATION ACTIVITY | 8.1.7: Query the Movie table.

The SQL statements below create a Movie table and insert some movies. The SELECT statement selects all movies.

Press the Run button to produce a result table. Verify the result table displays five movies.

Modify the statement `SELECT * FROM Movie;` to only select movies released after October 31, 2015, as follows:

```
SELECT *
FROM Movie
WHERE ReleaseDate > '2015-10-31';
```

Then run the SQL again and verify the new query returns only three movies, all with release dates after October 31, 2015.

Hint: Move the semicolon that follows `FROM Movie` to the end of the statement.

```sql
CREATE TABLE Movie (
  ID INT,
  Title VARCHAR(100),
  Rating VARCHAR(5),
  ReleaseDate DATE
);

INSERT INTO Movie VALUES
  (1, 'Rogue One: A Star Wars Story', 'PG-13', '2016-12-10'),
  (2, 'Hidden Figures', 'PG', '2017-01-06'),
  (3, 'Toy Story', 'G', '1995-11-22'),
  (4, 'Avengers: Endgame', 'PG-13', '2019-04-26'),
  (5, 'The Godfather', 'R', '1972-03-14');

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

```

Run   Reset code

▶ View solution

8.2 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 8.2.1: 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 8.2.1: 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 8.2.2: 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 | 8.2.2: 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 8.2.3: Using the LIKE operator.

Animation captions:

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 8.2.4: 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'))
  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 | 8.2.5: 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 BINARY '%E%'   LIKE '%cha'   LIKE '%m_o%'   LIKE '%r%n'

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 8.2.3: 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 8.2.6: 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
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 DISTINCT District
FROM City;
```

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

```
SELECT 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 8.2.4: 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 | 8.2.7: 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 | 8.2.1: Special operators and clauses.

544874.3500394.qx3zqy7

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

Exploring further:

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

8.3 Structured Query Language

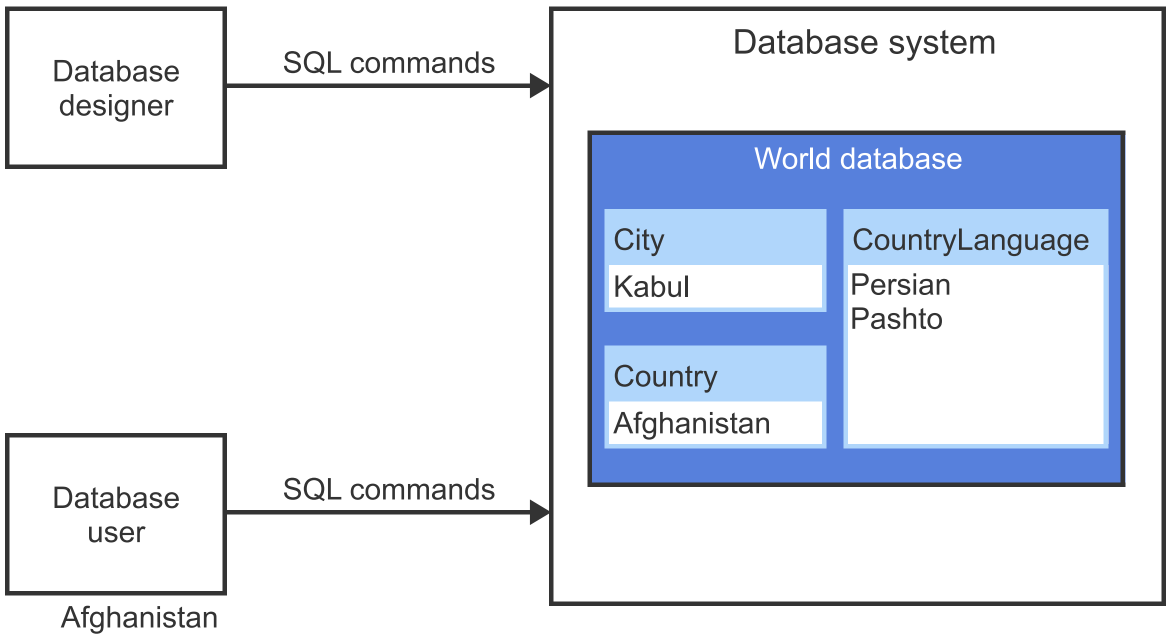
Structured Query Language

Structured Query Language (***SQL***) is a high-level computer language for storing, manipulating, and retrieving data. SQL is the standard language for relational databases, and is commonly supported in non-relational databases. SQL is pronounced either 'S-Q-L' or 'seekwəl', but the preferred pronunciation is 'S-Q-L'.

The SQL standard is published jointly by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). The standard was first published in 1986 and has since evolved through many versions. For details on the standard, see the link in Exploring Further, below.

Relational databases generally support most important elements of the SQL standard. However, most relational databases do not support the entire standard and many support custom extensions. This material describes standard SQL in most cases. Where MySQL differs from the standard, this material describes MySQL syntax and calls out the differences.

PARTICIPATION ACTIVITY | 8.3.1: SQL interaction with a database system.



Animation content:

Step 1: A database designer uses SQL to create a database and the database tables. A small box named Database designer appears on the left and a large box named Database system appears on the right. An arrow named SQL commands appears going from the Database designer box to the Database system box. The words CREATE DATABASE appear next to the Database designer box and move along the SQL commands arrow to the Database system box. A new box named World database appears inside the Database system box. The words CREATE TABLE appear next to the Database designer box and move along the SQL commands arrow to the Database system box. A new table named City appears inside the World database box. The words CREATE TABLE appear next to the Database designer box and move along the SQL commands arrow to the Database system box. A new table named Country appears inside the World database box. The words CREATE TABLE appear next to the Database designer box and move along the SQL commands arrow to the Database system box. A new table named CountryLanguage appears inside the World database box.

Step 2: A database user uses SQL to insert, retrieve, update, and delete data from the tables. A small box named Database user appears to the left of the Database system box. An arrow named SQL commands appears from the Database user box to the Database system box. The words INSERT Kabul appear next to the Database user box and move along the SQL commands arrow to the Database system box. Kabul appears in row one of the City table. The words SELECT Afghanistan appear next to the Database user box and move along the SQL commands arrow to the Database system box. Afghanistan then appears in row one of the Country table and then moves underneath the Database user box.

Animation captions:

1. A database designer uses SQL to create a database and the database tables.
2. A database user uses SQL to insert, retrieve, update, and delete data from the tables.

PARTICIPATION ACTIVITY | 8.3.2: SQL.

1) SQL commands can create databases and tables.

○ True

○ False

2) A database designer and database user both use SQL.

○ True

○ False

3) All database systems use identical SQL statements.

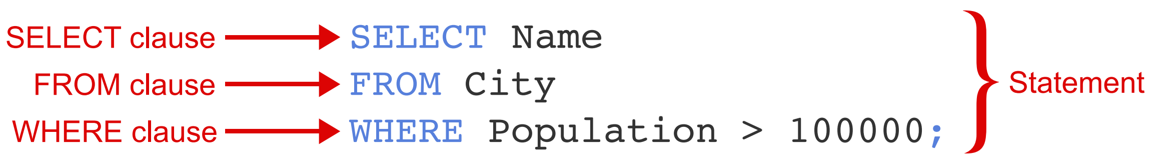
○ True

○ False

SQL syntax

An SQL ***statement*** is a complete command composed of one or more clauses. A ***clause*** groups SQL keywords like SELECT, FROM, and WHERE with table names like City, column names like Name, and conditions like Population > 100000. *An SQL statement may be written on a single line, but good practice is to write each clause on a separate line.*

PARTICIPATION ACTIVITY | 8.3.3: Three clauses in a SELECT statement.



```
SELECT Name
FROM City
WHERE Population > 100000;
```

Animation content:

Step 1: The SELECT clause starts the statement. Name is a column name. The code SELECT Name appears. Text appears to the left of the code and states SELECT clause. An arrow appears from this text to the code.

Step 2: The FROM clause must follow the SELECT clause. City is a table name. The code FROM City appears below the previous code. Text appears below the previous text and states FROM clause. An arrow appears from this text to the second line of code.

Step 3: The WHERE clause is optional. When included, the WHERE clause must follow the FROM clause. Population > 100000 is a condition. The code WHERE Population is greater than 100000 appears below the previous code. Text appears below the previous text and states WHERE clause. An arrow appears from this text to the third line of code.

Step 4: The three clauses ending in a semicolon comprise a statement. A semicolon is added to the end of the third line of code and a right bracket encompassing all three lines of code appears with the text Statement appearing to the right of the bracket.

Animation captions:

1. The SELECT clause starts the statement. Name is a column name.
2. The FROM clause must follow the SELECT clause. City is a table name.
3. The WHERE clause is optional. When included, the WHERE clause must follow the FROM clause. Population > 100000 is a condition.
4. The three clauses ending in a semicolon comprise a statement.

In MySQL, all SQL statements end with a semicolon. SQL keywords like SELECT, FROM, WHERE, etc. are not case sensitive. Ex: SELECT and select are equivalent. However, identifiers like column names and table names are case sensitive in many database systems. This material uses capital letters for SQL keywords so the keywords stand out from other syntactic parts. The table below summarizes various syntactic features of SQL.

Table 8.3.1: Summary of SQL syntax features.

Type	Description	Examples

Literals	Explicit values that are string, numeric, or binary.
	Strings must be surrounded by single quotes or double quotes.
	Binary values are represented with `x'0'` where the 0 is any hex value.
		`'String'`
		`"String"`
		`123`
		`x'0fa2'`

Keywords	Words with special meaning.	SELECT, FROM, WHERE
Identifiers	Objects from the database like tables, columns, etc.	City, Name, Population

Comments	Statement intended only for humans and ignored by the database when parsing an SQL statement.	`-- single line comment`
		`/* multi-line Comment */`

All database systems recognize single quotes for string literals, but some also recognize double quotes. This material uses single quotes to ensure the SQL statements are compatible with all database systems.

PARTICIPATION ACTIVITY | 8.3.4: SQL syntax.

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

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

- ○ 1
- ○ 2
- ○ 3

2) The SQL statement below is used to select students with the last name "Smith". What is wrong with the statement?

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

- ○ The literal "Smith" must be surrounded by single quotes or double quotes.
- ○ The WHERE clause should be removed.
- ○ The last name "Smith" may not exist in the database.

3) What is wrong with the SQL statement below?

```
SELECT FirstName
from Student
```

- ○ The keyword from must be written FROM.

- ○ The WHERE clause is missing.
- ○ A terminating semicolon is missing.

4) What is wrong with the SQL statement below?

```
SELECT Gpa
--FROM Student;
```

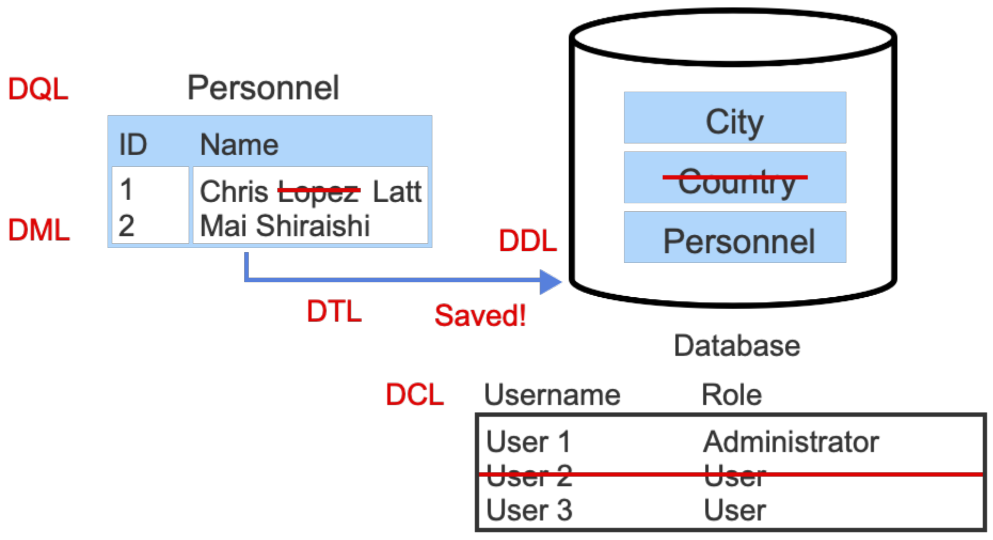
- ○ The FROM clause is a comment.
- ○ The WHERE clause is missing.
- ○ Gpa should be a table name.

SQL sublanguages

The SQL language is divided into five sublanguages:

- ***Data Definition Language*** (DDL) defines the structure of the database.
- ***Data Query Language*** (DQL) retrieves data from the database.
- ***Data Manipulation Language*** (DML) manipulates data stored in a database.
- ***Data Control Language*** (DCL) controls database user access.
- ***Data Transaction Language*** (DTL) manages database transactions.

PARTICIPATION ACTIVITY 8.3.5: SQL sublanguages.



Animation content:

Step 1: DDL creates, alters, and drops tables. A cylinder named Database appears containing the words City and Country. The caption DDL appears. A line strikes through Country and a new word Personnel appears below Country.

Step 2: DQL selects data from a table. The caption DQL appears. The Personnel table appears next to the caption with one row.

Step 3: DML inserts, updates, and deletes data in a table. The caption DML appears. The values of the row in Personnel change and a new row is inserted.

Step 4: DCL grants and revokes permissions to and from users. The caption DCL appears next to a table with columns Username and Role. The table has three rows. A line strikes through the second row.

Step 5: DTL commits data to a database, rolls back data from a database, and creates savepoints. The caption DTL appears. An arrow appears from the Personnel table to the Database cylinder. The arrow is labeled saved.

Animation captions:

1. DDL creates, alters, and drops tables.
2. DQL selects data from a table.
3. DML inserts, updates, and deletes data in a table.
4. DCL grants and revokes permissions to and from users.
5. DTL commits data to a database, rolls back data from a database, and creates savepoints.

PARTICIPATION ACTIVITY 8.3.6: SQL sublanguages.

Match the SQL sublanguage to the statement behavior.

If unable to drag and drop, refresh the page.

DML **DQL** **DCL** **DTL** **DDL**

	Insert a data row into table product.
	Rollback database changes.
	Select all rows from table Product.
	Grant all permissions to user 'tester'.
	Create table Product.

Reset

Exploring further:

- SQL standard (Wikipedia)

8.4 Managing databases

CREATE DATABASE and DROP DATABASE statements

A ***database system instance*** is a single executing copy of a database system. Personal computers usually run just one instance of a database system. Shared computers, such as computers used for cloud services, usually run multiple instances of a database system. Each instance usually contains multiple system and user databases.

Several SQL statements help database administrators, designers, and users manage the databases on an instance. ***CREATE DATABASE DatabaseName*** creates a new database. ***DROP DATABASE DatabaseName*** deletes a database, including all tables in the database.

PARTICIPATION ACTIVITY 8.4.1: CREATE DATABASE and DROP DATABASE statements.

Animation captions:

PARTICIPATION ACTIVITY | 8.4.2: CREATE DATABASE and DROP DATABASE statements.

Assume the database system currently contains only one user database, called 'university', in addition to system databases such as 'mySQL'.

1) The statement `CREATE DATABASE auto;` creates a database called 'auto'.
   - ○ True
   - ○ False

2) The statement `CREATE DATABASE university;` creates a second university database.
   - ○ True
   - ○ False

3) The statement `DROP DATABASE university;` deletes the university database and all associated tables.
   - ○ True
   - ○ False

4) The statement `DROP DATABASE nonprofit;` deletes the nonprofit database.

- ○ True
- ○ False

USE and SHOW statements

USE DatabaseName selects a default database for use in subsequent SQL statements.

Several SHOW statements provide information about databases, tables, and columns:

- ***SHOW DATABASES*** lists all databases in the database system instance.
- ***SHOW TABLES*** lists all tables in the default database.
- ***SHOW COLUMNS FROM TableName*** lists all columns in the TableName table of the default database.
- ***SHOW CREATE TABLE TableName*** shows the CREATE TABLE statement for the TableName table of the default database.

Additional SHOW statements generate information about system errors, configuration, privileges, logs, and so on. For details, see the link in Exploring Further, below.

PARTICIPATION ACTIVITY | 8.4.3: USE and SHOW statements.

Animation captions:

The world database is from MySQL.com

PARTICIPATION ACTIVITY 8.4.4: USE and SHOW statements.

Refer to the animation above.

1) Which statement shows all databases in a database system?

- ○ SHOW TABLES;
- ○ SHOW DATABASES;
- ○ SHOW COLUMNS;

2) Which statement shows all tables in the database petStore?

- ○ SHOW TABLES;
- ○ SHOW TABLES FROM petStore;
- ○ SHOW DATABASES;

3) Which statement must precede a SHOW TABLES statement to see the tables from the bikeStore database?

- ○ SHOW DATABASES;
- ○ USE bikeStore;
- ○ SHOW COLUMNS FROM bikeStore;

4) Which statement shows all the columns in the Country table?

- ○ SHOW COLUMNS;
- ○ SHOW COLUMNS Country;
- ○ SHOW COLUMNS FROM Country;

Country,

Exploring further:

- Creating and using MySQL databases
- MySQL SHOW statement

8.5 Tables

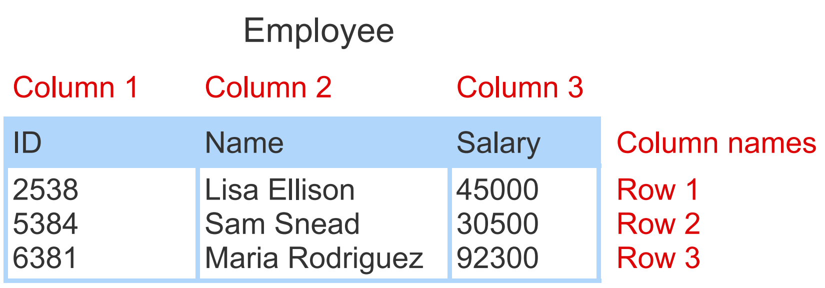
Tables, columns, and rows

All data in a relational database is structured in tables:

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

A table must have at least one column but any number of rows. A table without rows is called an ***empty table***.

PARTICIPATION ACTIVITY 8.5.1: Tables, columns, and rows.



Employee

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

Column 1
Column 2
Column 3
Column names
Row 1
Row 2
Row 3

Animation content:

Step 1: The Employee table has 3 columns with the names ID, Name, and Salary. The table has 3 rows. The Employee table appears with values:
2538, Lisa Ellison, 45000
5384, Sam Snead, 30500
6381, Maria Rodriguez, 92300

Step 2: Each value appears in a single cell. In the first row of the table, "2538" is the ID column's value. The first row of column ID is highlighted.

Step 3: In the first row of the table, "Lisa Ellison" is the Name column's value. The first row of column Name is highlighted.

Step 4: In the first row of the table, "45000" is the Salary column's value. The first row of column Salary is highlighted.

Animation captions:

1. The Employee table has 3 columns with the names ID, Name, and Salary. The table has 3 rows.
2. Each value appears in a single cell. In the first row of the table, "2538" is the ID column's value.
3. In the first row of the table, "Lisa Ellison" is the Name column's value.
4. In the first row of the table, "45000" is the Salary column's value.

PARTICIPATION ACTIVITY 8.5.2: Tables, columns, and rows.

Refer to the Employee table in the above animation.

1) Which choice is a column name?

- ○ 6381
- ○ Sam Snead
- ○ Salary

2) Which choice is a cell value?

- ○ ID
- ○ Elsa Brinks
- ○ 30500

3) What are the components of a column?

- ○ Name only
- ○ Data type only
- ○ Name and data type

4) Must a table have at least one row?

- ○ Yes
- ○ No

5) How does a database administrator specify column names and data types?

- ○ In a special file managed by the database administrator
- ○ With the Python language
- ○ With the SQL language

Rules governing tables

Tables must obey relational rules, including:

1. *Exactly one value per cell*. A cell may not contain multiple values. Unknown data is represented with a special NULL value.
2. *No duplicate column names*. Duplicate column names are allowed in different tables, but not in the same table.
3. *No duplicate rows*. No two rows may have identical values in all columns.
4. *No row order*. Rows are not ordered. The organization of rows on a storage device, such as a disk drive, never affects query results.

Rules 3 and 4 follow directly from the definition of a table. A table is a set of rows. Since a set's elements may not repeat and are not ordered, the same is true of a table's rows.

Rule 4 is called ***data independence***. Data independence allows database administrators to improve query performance by changing the organization of data on storage devices, without affecting query results.

PARTICIPATION

ACTIVITY | 8.5.3: Rules governing tables.

Employee

ID	Name	Salary
2538	Lisa Ellison	45000
5384	Sam Snead	30500
6381	Maria Rodriguez	92300, ~~70000~~

Employee

ID	Name	Salary	~~Salary~~
2538	Lisa Ellison	45000	~~35000~~
5384	Sam Snead	30500	~~25600~~
6381	Maria Rodriguez	92300	~~88100~~

Employee

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

Animation content:

Step 1. A table cannot have multiple values in the same cell. The Employee table appears, with three columns named ID, Name, and Salary. Two values appear within the cell for the Salary column of row three:
92300, 70000
The second value, 70000, is struck out with a red line.

Step 2. A table cannot have multiple columns with the same name. The Employee table appears, with four columns named ID, Name, Salary, and Salary. The column name and all values in the second Salary column are struck out with red lines.

Step 3. A table cannot have multiple rows with identical data. The Employee table appears, with three columns named ID, Name, and Salary. The table has three rows. Rows two and three have identical values. Row three is struck out with a red line.

Animation captions:

1. A table cannot have multiple values in the same cell.

2. A table cannot have multiple columns with the same name.
3. A table cannot have multiple rows with identical data.

Duplicate rows

Most databases allow tables with duplicate rows, in violation of rule 3. However, these tables are usually temporary. Ex: External data with duplicate rows might be loaded to a temporary table. The duplicate rows are normally eliminated as data is moved to a permanent table.

PARTICIPATION ACTIVITY | 8.5.4: Rules governing tables.

1) _____ may appear in each cell.

- ○ Any number of values
- ○ Exactly one value
- ○ No values or one value

2) Where are duplicate column names allowed?

- ○ Within a single table.
- ○ In different tables.
- ○ Never.

3) What does the principle of data independence state?

- ○ The performance of a query is not related to the physical organization of data.
- ○ Data in each row is independent of data in all other rows.
- ○ The result of a database query is not affected by the physical organization of data on storage devices

4) How can driver's license information be added to the Employee table?

○ Driver's licenses cannot be added, since the Employee table already has an ID column.

○ By creating another column named ID.

○ By creating another column with a new name, such as ID2 or DriverLicense.

CREATE TABLE and DROP TABLE statements

The ***CREATE TABLE*** statement creates a new table by specifying the table name, column names, and column data types. Example data types are:

- *INT or INTEGER* — integer values
- *VARCHAR(N)* — values with 0 to N characters
- *DATE* — date values
- *DECIMAL(M, D)* — numeric values with M digits, of which D digits follow the decimal point

The ***DROP TABLE*** statement deletes a table, along with all the table's rows, from a database.

Figure 8.5.1: CREATE TABLE and DROP TABLE statements.

```
CREATE TABLE TableName
(
  Column1 DATA_TYPE,
  Column2 DATA_TYPE,
  ...
  ColumnN DATA_TYPE
);


DROP TABLE TableName;
```

PARTICIPATION ACTIVITY 8.5.5: Creating an Employee table.

Animation captions:

PARTICIPATION ACTIVITY | 8.5.6: Creating and deleting tables.

Refer to the animation above.

1) The Employee table is created with 4 columns.
   - ○ True
   - ○ False

2) The Name column contains character string values.
   - ○ True
   - ○ False

3) The statement `DROP Employee;` deletes the Employee table.
   - ○ True
   - ○ False

4) The DROP TABLE statement does not delete the table unless the table is empty.
   - ○ True
   - ○ False

PARTICIPATION ACTIVITY | 8.5.7: Create a Product table with common data types.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

ALTER TABLE statement

The ***ALTER TABLE*** statement adds, deletes, or modifies columns on an existing table. The ALTER TABLE statement specifies the table name followed by a clause that indicates what should be altered. The table below summarizes the three ALTER TABLE clauses.

Table 8.5.1: ALTER TABLE statement.

ALTER TABLE clause	Description	Syntax

ADD	Adds a column	`ALTER TABLE TableName`
		`ADD ColumnName DataType;`

CHANGE	Modifies a column	`ALTER TABLE TableName`
		`CHANGE CurrentColumnName NewColumnName`
		`NewDataType;`

DROP	Deletes a column	`ALTER TABLE TableName`
		`DROP ColumnName;`

PARTICIPATION ACTIVITY | 8.5.8: Adding, modifying, and deleting columns.

Employee

ID	Name	BirthDate
2538	Lisa Ellison	1993-10-02
5384	Sam Snead	1995-03-15
6381	Maria Rodriguez	2001-12-21
7343	Gary Smith	1984-09-22

Animation captions:

PARTICIPATION ACTIVITY | 8.5.9: ALTER TABLE statement.

1) Add a column called Description to the Department table.

```
ALTER TABLE Department
______
VARCHAR(50);
```

Check   Show answer

2) Rename column Description to ShortDesc.

```
ALTER TABLE Department
______
VARCHAR(50);
```

Check   Show answer

3) Change column ShortDesc to accept up to 80 characters.

```
ALTER TABLE Department
CHANGE
```

;

Check Show answer

4) Delete the column ShortDesc.

ALTER

Check Show answer

CHALLENGE ACTIVITY | 8.5.1: Creating, dropping, and altering tables.

544874.3500394.qx3zqy7

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

Exploring further:

- MySQL CREATE TABLE syntax
- MySQL DROP TABLE syntax
- MySQL ALTER TABLE syntax

8.6 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 8.6.1: CREATE VIEW statement.

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

PARTICIPATION ACTIVITY 8.6.1: Creating a view table.

Animation captions:

PARTICIPATION ACTIVITY | 8.6.2: 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

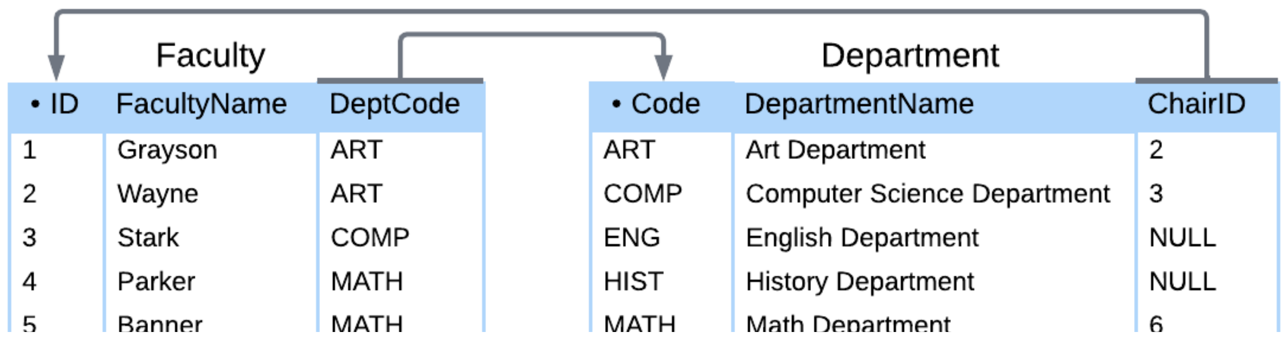
A view can be defined on other view tables when the view query FROM clause includes additional view tables. In this case, the additional view tables are not base tables. ***Base tables*** *are always source tables, created as tables rather than as views.*

PARTICIPATION ACTIVITY | 8.6.3: Merging the view query and user query.

Animation captions:

PARTICIPATION ACTIVITY | 8.6.4: 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

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

6	Quinn	MATH
7	Grey	NULL

```
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 | 8.6.5: Advantages of views.

Animation captions:

PARTICIPATION ACTIVITY | 8.6.6: 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 | 8.6.7: Inserting, updating, and deleting views.

Animation captions:

PARTICIPATION ACTIVITY | 8.6.8: 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 8.6.2: WITH CHECK OPTION clause.

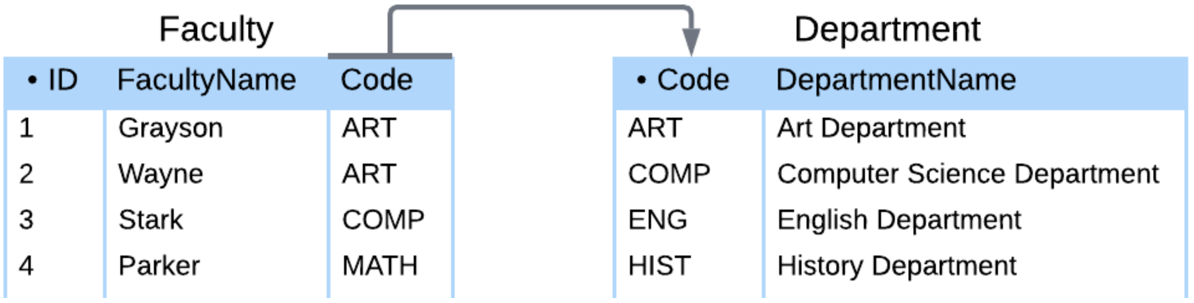
```
CREATE VIEW ViewName [ ( Column1, Column2, ...
) ]
AS SelectStatement
[ WITH CHECK OPTION ];
```

PARTICIPATION ACTIVITY | 8.6.9: WITH CHECK OPTION clause.

Animation captions:

PARTICIPATION ACTIVITY | 8.6.10: 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

Department

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

5	Banner	MATH
6	Quinn	MATH
7	Grey	NULL

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 | 8.6.1: View tables.

544874.3500394.qx3zqy7

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

Exploring further:

- MySQL CREATE VIEW statement
- Using MySQL views

8.7 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 8.7.1: 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 8.7.1: Relational algebra operations.

Match the symbol to the operation name.

If unable to drag and drop, refresh the page.

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

	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

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

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

PARTICIPATION ACTIVITY | 8.7.2: 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

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

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 8.7.3: Select and project operations.

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

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

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

- True
- False

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

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

- True
- False

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

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

- 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:

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

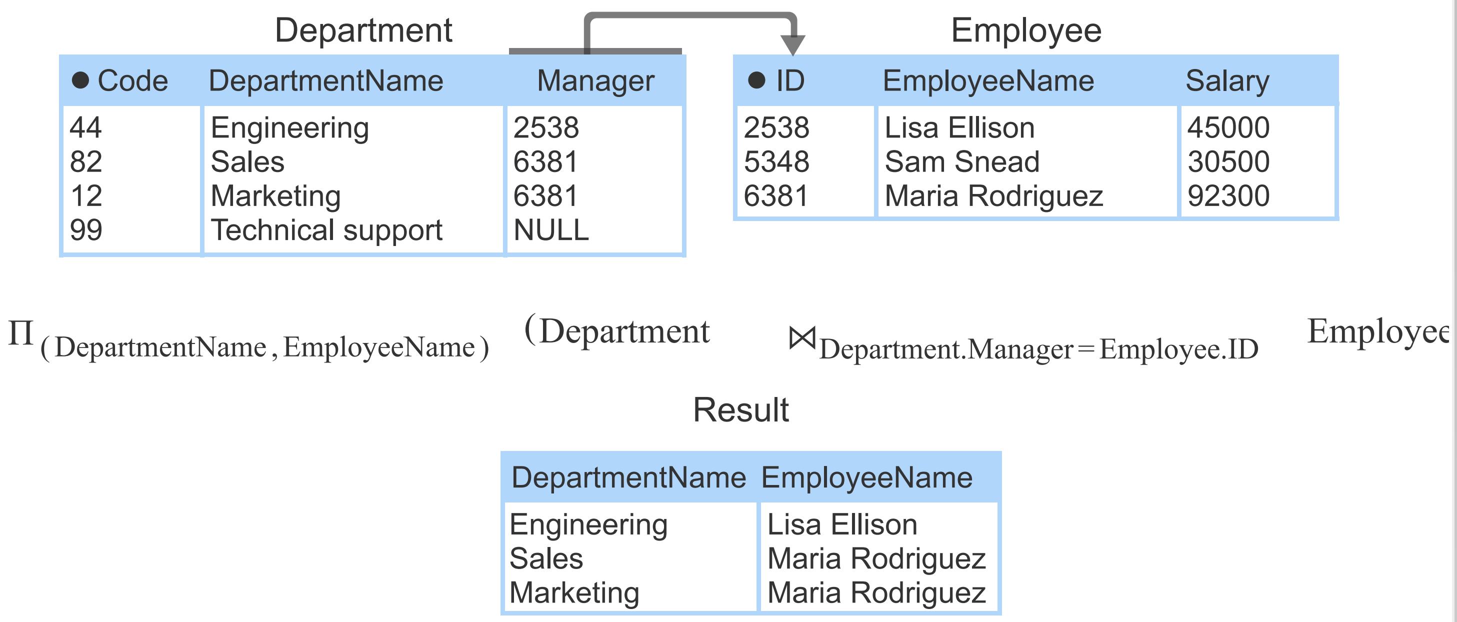
The join operation is equivalent to `SELECT * FROM Table1 INNER JOIN Table2 ON expression`. The join expression is any expression on columns of Table1 and Table2, and is often denoted with the Greek letter theta:

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

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 | 8.7.4: 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 8.7.5: 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?

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

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`.

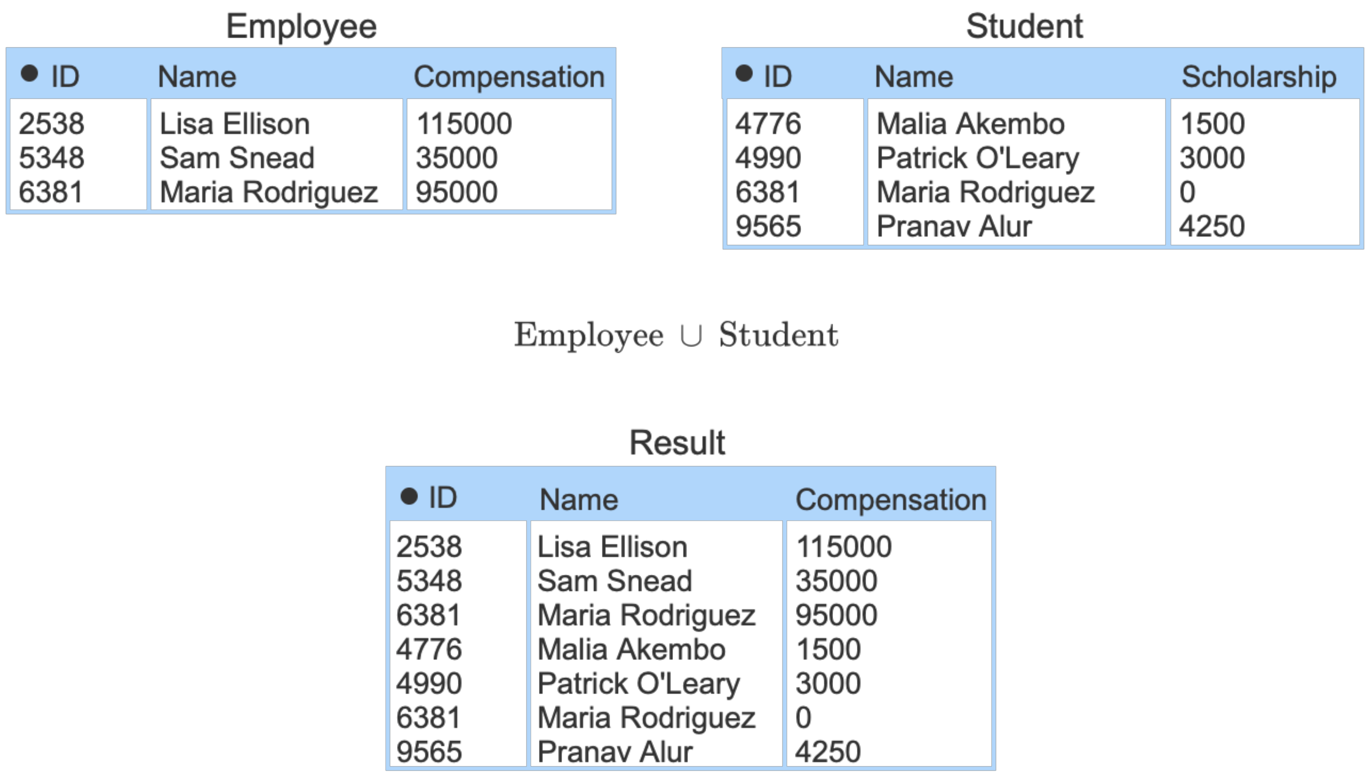
The ***difference*** operation removes from a table all rows that appear in a second compatible table. The difference operation is written as

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

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 8.7.6: 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 8.7.7: 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?

$\mathbf{Student} \cup \mathbf{Student}$

○ 0

- ○ 4
- ○ 8

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

$$\textbf{Employee} \cap \textbf{Student}$$

- ○ 0
- ○ 1
- ○ 7

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

$$\textbf{Employee} - \textbf{Student}$$

- ○ 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?

$$\textbf{Employee} - \textbf{Student}$$

- ○ 0
- ○ 2
- ○ 3

Set operations in MySQL

The EXCEPT keyword is equivalent to the MINUS keyword. Support for these keywords varies by database.

MySQL supports UNION but not INTERSECT, MINUS, or EXCEPT. 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;
```

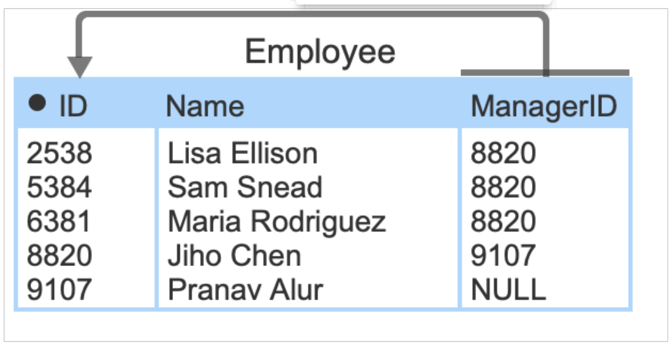
Rename and aggregate

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

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

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

PARTICIPATION ACTIVITY 8.7.8: Rename operation.



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

Animation captions:

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 | 8.7.9: Aggregate operation.

Animation captions:

PARTICIPATION ACTIVITY | 8.7.10: 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 ($\mathscr{F}$), or the script A ($\mathscr{A}$).

Query optimization

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

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

is equivalent to

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

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 8.7.11: 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 8.7.12: 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 8.7.1: Relational algebra.

544874.3500394.qx3zqy7

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

Exploring further:

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

8.8 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 | 8.8.1: Example join.

Animation captions:

PARTICIPATION ACTIVITY | 8.8.2: 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 animation above. This join is intended to list departments with managers that earn more than $50,000:

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

What is missing from the statement?

- ○ Left and right tables are not specified.
- ○ The WHERE clause does not compare Employee and Department columns.
- ○ No Employee columns appear in the SELECT clause.

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 8.8.1: 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 8.8.3: 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.

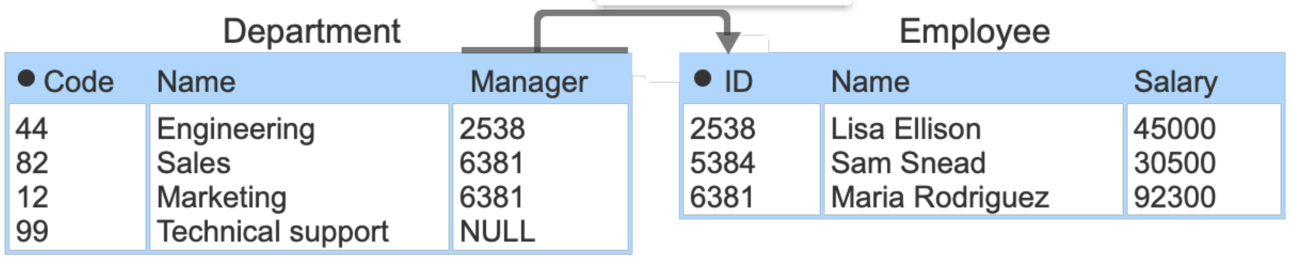
The join clause appears between a FROM clause and an ON clause:

- 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 | 8.8.4: Inner and full joins.



Department

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

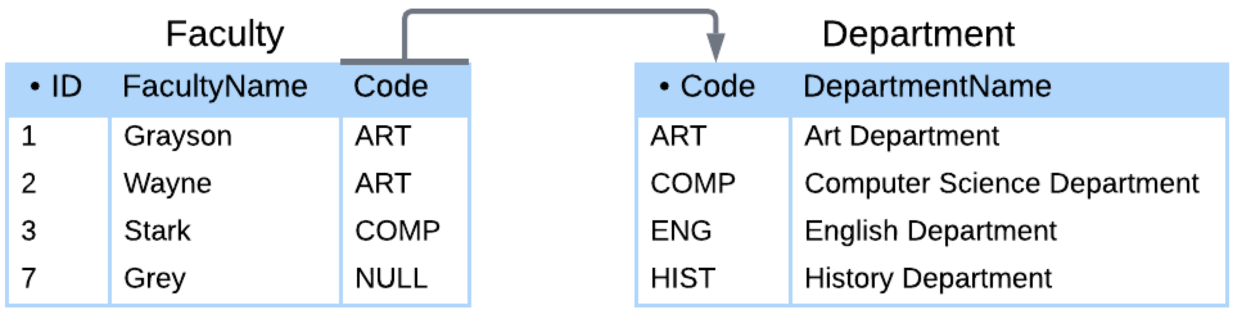
Employee

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

Animation captions:

PARTICIPATION ACTIVITY 8.8.5: 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:

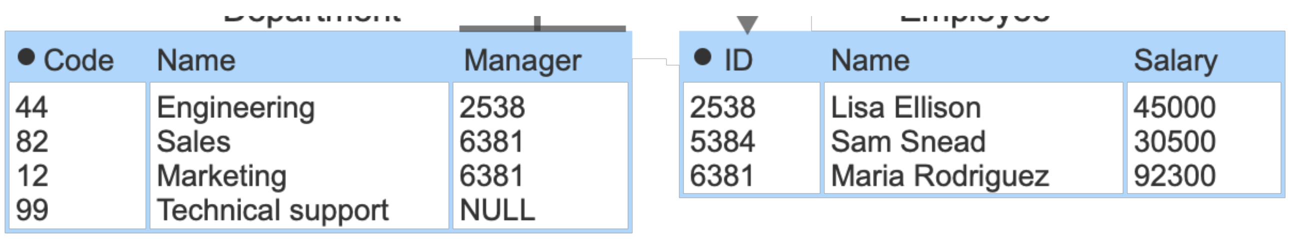
- ***LEFT JOIN*** selects all left table rows, but only matching right table rows.
- ***RIGHT JOIN*** selects all right table rows, but only matching left table rows.

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 | 8.8.6: Left and right joins.

Department Employee



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

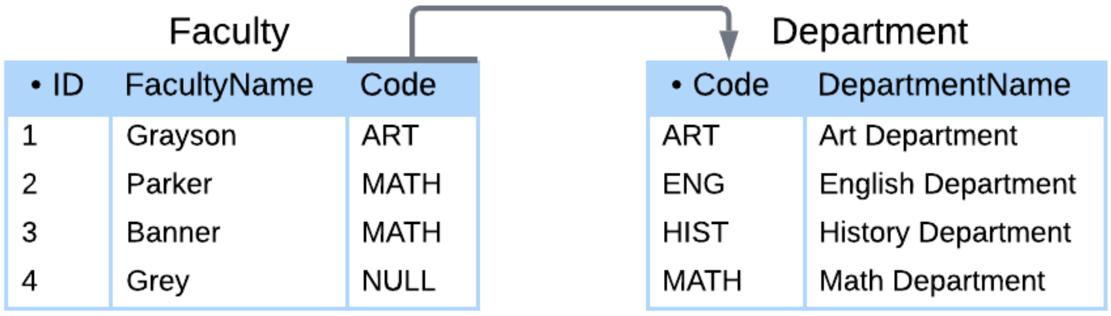
● ID	Name	Salary
2538	Lisa Ellison	45000
5384	Sam Snead	30500
6381	Maria Rodriguez	92300

Animation captions:

PARCIPATION ACTIVITY

8.8.7: 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.

FULL LEFT RIGHT INNER

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 | 8.8.8: Equivalent join queries.

Left join with JOIN

Animation captions:

PARTICIPATION ACTIVITY | 8.8.9: Alternative join queries.

Refer to the tables below.

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

• 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.

full join   inner join   right join   left 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);
```

Reset

PARTICIPATION ACTIVITY 8.8.10: Join coding exercise.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

CHALLENGE ACTIVITY 8.8.1: Inner and outer joins.

544874.3500394.qx3zqy7

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

Exploring further:

- MySQL join syntax

8.9 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 8.9.1: 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 8.9.1: Equijoins and non-equijoins.

Refer to the following tables

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?

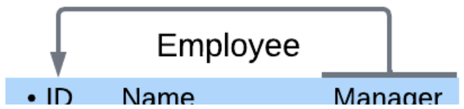
- ○ INNER JOIN and FULL JOIN only
- ○ LEFT JOIN and RIGHT JOIN only
- ○ All JOIN clauses

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 8.9.2: Self-join example.



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 | 8.9.2: 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?

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 8.9.3: 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 | 8.9.3: Special cases.

Match the special join with the definition.

If unable to drag and drop, refresh the page.

Cross-join Equijoin Non-equijoin Self-join

	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 | 8.9.1: Equijoins, self-joins, and cross-joins.

544874.3500394.qx3zqy7

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

8.10 Data types

Data type categories

A ***data type*** is a named set of values from which column values are drawn. In relational databases, most data types fall into one of the following categories:

- ***Integer*** data types represent positive and negative integers. Several integer data types exist, varying by the number of bytes allocated for each value. Common integer data types include INT, implemented as 4 bytes of storage, and SMALLINT, implemented as 2 bytes.
- ***Decimal*** data types represent numbers with fractional values. Decimal data types vary by number of digits after the decimal point and maximum size. Common decimal data types include FLOAT and DECIMAL.
- ***Character*** data types represent textual characters. Common character data types include CHAR, a fixed string of characters, and VARCHAR, a string of variable length up to a specified maximum size.
- ***Date and time*** data types represent date, time, or both. Some date and time data types include a time zone or specify a time interval. Some date and time data types represent an interval rather than a point in time. Common date and time data types include DATE, TIME, DATETIME, and TIMESTAMP.
- ***Binary*** data types store data exactly as the data appears in memory or computer files, bit for bit. The database manages binary data as a series of zeros and ones. Common binary data types include BLOB, BINARY, VARBINARY, and IMAGE.
- ***Spatial*** data types store geometric information, such as lines, polygons, and map coordinates. Examples include POLYGON, POINT, and GEOMETRY. Spatial data types are relatively new and consequently vary greatly across database systems.
- ***Document*** data types contain textual data in a structured format such as XML or JSON.

Databases support additional data types for specialized uses. Ex: MONEY for currency values, BOOLEAN for true-false values, BIT for zeros and ones, and ENUM for a small, fixed set of alternative values.

Table 8.10.1: Example data types.

Category	Data type	Value
Integer	INT	-9281344

Decimal	FLOAT	3.1415
Character	VARCHAR	Chicago
Date and time	DATETIME	12/25/2020 10:35:00
Binary	BLOB	1001011101 . . .
Spatial	POINT	(2.5, 33.44)

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

PARTICIPATION ACTIVITY 8.10.1: Data types.

Match the value with the data type.

If unable to drag and drop, refresh the page.

3.14 **16** **Nadia** **00011011001** **09/12/1986**

	VARCHAR
	INT
	DATE
	BLOB

	FLOAT

Reset

PARTICIPATION ACTIVITY | 8.10.2: Data types.

1) The CHAR data type represents a variable string of characters.
   - ○ True
   - ○ False

2) A binary data type stores data as an exact copy of computer memory.
   - ○ True
   - ○ False

3) Some date and time data types include time zone.
   - ○ True
   - ○ False

MySQL data types

All relational databases support integer, decimal, date and time, and character data types. Most databases allow integer and decimal numbers to be signed or unsigned. A ***signed*** number may be negative. An ***unsigned*** number cannot be negative.

Data types vary in storage requirements. Ex:

- Character data types use one or two bytes per character.
- Integer data types use a fixed number of bytes per number.
- Unsigned data types can store larger numbers than the signed version of the same data type.

To minimize table size, the data type with the smallest storage requirements should be used. Ex: Any integer data type can store integers that range from -100 to 100. However, TINYINT requires only one byte per number and is the best choice for this range.

Common MySQL data types appear in the table below. For a complete list of MySQL data types, see the link in Exploring Further, below.

See the link in Exploring Further, below.

Table 8.10.2: Common MySQL data types.

Category	Example	Data type	Storage	Notes
Integer	34 and -739448	TINYINT	1 byte	Signed range: -128 to 127 Unsigned range: 0 to 255
		SMALLINT	2 bytes	Signed range: -32,768 to 32,767 Unsigned range: 0 to 65,535
		MEDIUMINT	3 bytes	Signed range: -8,388,608 to 8,388,607 Unsigned range: 0 to 16,777,215
		INTEGER or INT	4 bytes	Signed range: -2,147,483,648 to 2,147,483,647 Unsigned range: 0 to 4,294,967,295
		BIGINT	8 bytes	Signed range: -2^{63} to 2^{63} -1 Unsigned range: 0 to 2^{64} -1
Decimal	123.4 and -54.29685	DECIMAL(M,D)	Varies depending on *M* and *D*	Exact decimal number where *M* = number of significant digits, *D* = number of digits after decimal point
		FLOAT	4 bytes	Approximate decimal numbers with range: -3.4E+38 to 3.4E+38
		DOUBLE	8 bytes	Approximate decimal numbers with range: -1.8E+308 to 1.8E+308
Date and time	'1776-07-04 13:45:22'	DATE	3 bytes	Format: YYYY-MM-DD. Range: '1000-01-01' to '9999-12-31'
		TIME	3 bytes	Format: hh:mm:ss
		DATETIME	5 bytes	Format: YYYY-MM-DD hh:mm:ss. Range: '1000-01-01 00:00:00' to '9999-12-31

				23:59:59'.
Character	'string'	CHAR(N)	*N* bytes	Fixed-length string of length *N*; 0 ≤ *N* ≤ 255
		VARCHAR(N)	Length of characters + 1 bytes	Variable-length string with maximum *N* characters; 0 ≤ *N* ≤ 65,535
		TEXT	Length of characters + 2 bytes	Variable-length string with maximum 65,535 characters

PARTICIPATION ACTIVITY 8.10.3: Common MySQL data types.

Choose the best data type for each scenario with the minimum storage requirements.

1) Storing a city's population, which ranges from a dozen to 24 million people.
   - ○ unsigned MEDIUMINT
   - ○ unsigned INTEGER
   - ○ unsigned BIGINT

2) Storing the annual gain or loss in a city's population, which ranges from -1 million to 1 million.
   - ○ signed SMALLINT
   - ○ unsigned MEDIUMINT
   - ○ signed MEDIUMINT

3) Storing the price of an item that ranges from a few dollars to a few hundred dollars.
   - ○ FLOAT
   - ○ DECIMAL(2,5)
   - ○ DECIMAL(5,2)

4) Storing the date and time an item is

purchased.

- ○ DATE
- ○ TIME
- ○ DATETIME

5) Storing a student's assigned letter grade, like A or D.

- ○ TINYINT
- ○ CHAR(1)
- ○ VARCHAR(1)

6) Storing a student's email address.

- ○ VARCHAR(100)
- ○ VARCHAR(10)
- ○ CHAR(100)

PARTICIPATION ACTIVITY 8.10.4: Create a Product table with MySQL data types.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

Exploring further:

- MySQL data types

8.11 Selecting rows

Operators

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

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

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

Operators may return NULL when either operand is NULL. NULL-valued operations are discussed elsewhere in this material.

Table 8.11.1: Common operators.

Type	Operator	Description	Example	Value
Arithmetic	`+`	Adds two numeric values	`4 + 3`	`7`
	`-` (unary)	Reverses the sign of one numeric value	`-(-2)`	`2`
	`-` (binary)	Subtracts one numeric value from another	`11 - 5`	`6`
	`*`	Multiplies two numeric values	`3 * 5`	`15`
	`/`	Divides one numeric value by another	`4 / 2`	`2`

	`%` (modulo)	Divides one numeric value by another and returns the integer remainder	`5 % 2`	`1`
	`^`	Raises one numeric value to the power of another	`5^2`	`25`
Comparison	`=`	Compares two values for equality	`1 = 2`	`FALSE`

	`!=`
	`<>`
		Compares two values for inequality	`1 != 2`
			`1 <> 2`
				`TRUE`

	`<`	Compares two values with <	`2 < 2`	`FALSE`
	`<=`	Compares two values with ≤	`2 <= 2`	`TRUE`
	`>`	Compares two values with >	`'2019-08-13' > '2021-08-13'`	`FALSE`
	`>=`	Compares two values with ≥	`'apple' >= 'banana'`	`FALSE`
	`AND`	Returns TRUE only when both values are TRUE	`TRUE AND FALSE`	`FALSE`

Logical	`OR`	Returns FALSE only when both values are FALSE	`TRUE OR FALSE`	`TRUE`
	`NOT`	Reverses a logical value	`NOT FALSE`	`TRUE`

PARTICIPATION ACTIVITY 8.11.1: Operators.

Match the operator with the description.

If unable to drag and drop, refresh the page.

% AND != - NOT

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

Reset

Expressions

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

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

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

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

Expressions may evaluate to NULL. NULL-valued expressions are discussed elsewhere in this material.

Table 8.11.2: Operator precedence.

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

PARTICIPATION ACTIVITY 8.11.2: Evaluating expressions.

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

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

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

'Platinum' = 'Platinum' AND 67.50 > 100.00

TRUE AND 67.50 > 100.00

TRUE AND FALSE

Value: FALSE

Animation content:

Static figure:
An expression appears with evaluation steps on subsequent lines:
Status = 'Platinum' AND (Quantity * UnitPrice + ShippingPrice) > 100.00
'Platinum' = 'Platinum' AND (4 * 15.00 + 7.50) > 100.00
'Platinum' = 'Platinum' AND (60.00 + 7.50) > 100.00
'Platinum' = 'Platinum' AND 67.50 > 100.00
TRUE AND 67.50 > 100.00
TRUE AND FALSE
FALSE

Step 1: The expression includes column names Status, Quantity, UnitPrice, and ShippingPrice. The initial expression appears:
Status = 'Platinum' AND (Quantity * UnitPrice + ShippingPrice) greater than 100.00

Step 2: The expression is evaluated for a specific row. Column names are replaced with column values for the row. The first evaluation step appears below the initial expression:
'Platinum' = 'Platinum' AND (4 * 15.00 + 7.50) greater than 100.00

Step 3:Operators within parentheses have the highest precedence. * has higher precedence than + and is evaluated first. The next evaluation step appears below the prior step:
'Platinum' = 'Platinum' AND (60.00 + 7.50) greater than 100.00

Step 4: + is the only remaining operator inside the parentheses, and is evaluated next. The next evaluation step appears below the prior step:
'Platinum' = 'Platinum' AND 67.50 greater than 100.00

Step 5: = has higher precedence than AND. = has the same precedence as > but appears to the left. So = is evaluated next. The next evaluation step appears below the prior step:
TRUE AND 67.50 greater than 100.00

Step 6: "greater than" has higher precedence than AND, and is evaluated next. The next evaluation step appears below the prior step:
TRUE AND FALSE

Step 7: The expression evaluates to FALSE for the row. FALSE appears under the prior step.

Animation captions:

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

PARTICIPATION ACTIVITY 8.11.3: Expressions.

What is the value of the following expressions?

1) `7 + 3 * 2`

- ○ 13
- ○ 20

2) `(7 + 3) * 2`

- ○ 13
- ○ 20

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

- ○ TRUE
- ○ FALSE

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

◯ TRUE

◯ FALSE

SELECT statement

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

In many queries, each expression in the SELECT clause is a simple column name. To select all columns, the expressions can be replaced with a single asterisk.

Figure 8.11.1: SELECT statement.

```
-- SELECT with expressions
SELECT Expression1, Expression2,
...
FROM TableName;

-- SELECT with column names
SELECT Column1, Column2, ...
FROM TableName;

-- SELECT with asterisk
SELECT *
FROM TableName;
```

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

PARTICIPATION ACTIVITY | 8.11.4: Selecting from table CountryLanguage.

Animation captions:

PARTICIPATION ACTIVITY | 8.11.5: SELECT with columns.

Refer to the following table.

City

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

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

```
SELECT

FROM City;
```

Check   Show answer

2) Select all rows and columns.

```
SELECT

FROM City;
```

Check   Show answer

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

```
SELECT

FROM City;
```

Check Show answer

PARTICIPATION ACTIVITY | 8.11.6: SELECT with expressions.

Refer to the following table.

Customer

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

1) What values are returned?

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

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

2) What values are returned?

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

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

Long tables

*Some tables may contain thousands or millions of rows, and selecting all rows can take a long time. MySQL has a **LIMIT** clause that limits the number of rows returned by a SELECT statement. Ex: The SELECT statement below returns only the first 100 rows from the City table.*

```
SELECT *
FROM City
LIMIT 100;
```

WHERE clause

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

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

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

Figure 8.11.2: WHERE clause.

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

PARTICIPATION ACTIVITY 8.11.7: WHERE clause.

Animation captions:

PARTICIPATION ACTIVITY 8.11.8: WHERE clause.

Refer to the City table.

City

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

1) What cities are selected?

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

- ○ All cities
- ○ Valencia and Manchester
- ○ Valencia

2) What districts are selected?

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

No districts

Rio de Janeiro

Rio de Janeiro and Katalonia

3) What cities are selected?

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

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

4) What names are selected?

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

- No names
- All names
- Valencia

5) What cities are selected?

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

- All cities
- Barcelona and Valencia
- Delhi

6) What cities are selected?

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

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

7) What cities are selected?

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

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

PARTICIPATION ACTIVITY 8.11.9: Select movies with logical operators.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

CHALLENGE ACTIVITY 8.11.1: Selecting rows.

544874.3500394.qx3zqy7

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

Exploring further:

- MySQL SELECT syntax
- MySQL operators

8.12 Constraints

Column and table constraints

A ***constraint*** is a rule that governs allowable values in a database. Constraints are based on relational and business rules, and implemented with special keywords in a CREATE TABLE statement. The database automatically rejects insert, update, and delete statements that violate a constraint.

The following constraints are described elsewhere in this material:

- NOT NULL
- DEFAULT
- PRIMARY KEY
- FOREIGN KEY

A ***column constraint*** appears after the column name and data type in a CREATE TABLE statement. Column constraints govern values in a single column. Ex: NOT NULL is a column constraint.

A ***table constraint*** appears in a separate clause of a CREATE TABLE statement and governs values in one or more columns. Ex: FOREIGN KEY is a table constraint.

Some constraint types can be defined as either column or table constraints. Ex: A PRIMARY KEY constraint on a single column can appear either in the column declaration or a separate CREATE TABLE clause. A PRIMARY KEY constraint on a composite column must be defined as a table constraint.

Figure 8.12.1: Example constraints.

```
CREATE TABLE Employee (
   ID             INT,
   Name           VARCHAR(20) NOT NULL,
   DepartmentCode INT DEFAULT 999,
   PRIMARY KEY (ID),
   FOREIGN KEY (DepartmentCode) REFERENCES Department
(Code)
);
```

DEFAULT constraint

The DEFAULT constraint does not actually limit allowable values in a column. Instead, DEFAULT specifies a value that is inserted when a column is omitted from an INSERT statement. For this reason, DEFAULT is not always considered a constraint.

PARTICIPATION ACTIVITY | 8.12.1: Column and table constraints.

Match the constraint with the type.

If unable to drag and drop, refresh the page.

NOT NULL **FOREIGN KEY** **PRIMARY KEY**

	Table constraint only
	Column constraint only
	Either column or table constraint

Reset

UNIQUE constraint

The ***UNIQUE*** constraint ensures that values in a column, or group of columns, are unique. When applied to a single column, UNIQUE may appear either in the column declaration or a separate clause. When applied to a group of columns, UNIQUE is a table constraint and must appear in a separate clause.

The UNIQUE constraint can be applied to primary key columns but is unnecessary, since primary keys are automatically unique.

MySQL creates an index for each UNIQUE constraint. The index stores the values of the unique column, or group of columns, in sorted order. When new values are inserted or updated, MySQL searches the index to quickly determine if the new value is unique.

PARTICIPATION ACTIVITY | 8.12.2: UNIQUE constraints on the Employee table.

Animation captions:

PARTICIPATION ACTIVITY | 8.12.3: UNIQUE constraint.

Indicate whether the rows violate the UNIQUE constraints on Department.

```
CREATE TABLE Department (
   Code        TINYINT UNSIGNED,
   Name        VARCHAR(20) UNIQUE,
   ManagerID   SMALLINT,
   Appointment DATE,
   PRIMARY KEY (Code),
   UNIQUE (ManagerID, Appointment)
);
```

1) `(44, 'Engineering', 2538, '2020-01-15')`
`(82, 'Sales', 6381, '2019-08-01')`

- OK
- Violates

2) `(12, 'Marketing', 2538, '2019-11-22')`
`(99, 'Marketing', NULL, NULL)`

```
(99, 'Marketing', NULL, NULL)
```

○ OK

○ Violates

3)
```
(44, 'Engineering', 2538,
'2020-01-15')
(82, 'Sales', 2538, '2020-01-
15')
```

○ OK

○ Violates

4)
```
(12, 'Marketing', 5384, '2019-
11-22')
(99, 'Technical Support',
5384, NULL)
```

○ OK

○ Violates

CHECK constraint

The ***CHECK*** constraint specifies an expression on one or more columns of a table. The constraint is violated when the expression is FALSE and satisfied when the expression is either TRUE or NULL.

When the expression contains one column, CHECK may appear either in the column declaration or a separate clause. When the expression contains multiple columns, CHECK is a table constraint and must be a separate clause.

PARTICIPATION ACTIVITY | 8.12.4: CHECK constraint.

Animation captions:

PARTICIPATION ACTIVITY 8.12.5: CHECK constraint.

Indicate whether each row violates the CHECK constraints on Department. The Size column constraint limits values to 'small', 'medium', or 'large'.

```
CREATE TABLE Department (
   Code           TINYINT UNSIGNED,
   Name           VARCHAR(20),
   ManagerID      SMALLINT,
   AdminAssistID SMALLINT,
   Size           VARCHAR(6) CHECK (Size IN ('small', 'medium', 'large')),
   PRIMARY KEY (Code),
   CHECK (ManagerID >= 1000 AND ManagerID <> AdminAssistID)
);
```

1) `(44, 'Engineering', 2538, 1024, 'tiny')`

- ○ OK
- ○ Violates

2) `(82, 'Sales', 999, 1024, 'medium')`

- ○ OK
- ○ Violates

3) `(99, 'Technical Support', 5384, 5384, 'large')`

- ○ OK
- ○ Violates

4) `(99, 'Technical Support',`

```
5384, 2399, 'large' )
```

○ OK

○ Violates

5) `(50, 'Maintenance', NULL, 4380, NULL)`

○ OK

○ Violates

Constraint names

Table constraints may be named using the optional ***CONSTRAINT*** keyword, followed by the constraint name and declaration. If no name is provided, the database generates a default name. Constraint names appear in error messages when constraints are violated.

Most column constraints cannot be named. However, the CHECK column constraint is an exception and can be named with a CONSTRAINT clause in the column declaration.

PARTICIPATION ACTIVITY 8.12.6: Constraint names.

Animation captions:

MySQL constraint names

The MySQL statement ***`SHOW CREATE TABLE TableName`*** *returns the CREATE TABLE statement for TableName. The statement shows all CONSTRAINT ConstraintName clauses but does not show default constraint names*

The following query displays all names of constraints on TableName, including default names:

```
SELECT Column_Name, Constraint_Name
FROM Information_Schema.Key_Column_Usage
WHERE Table_Name = 'TableName';
```

Key_Column_Usage is a table from the Information_Schema system database. Other Key_Column_Usage columns contain additional constraint details. Ex: Referenced_Table_Name and Referenced_Column_Name provide information about foreign key constraints.

PARTICIPATION ACTIVITY | 8.12.7: Constraint names.

1) Name the primary key table constraint 'DepartmentKey'.

```
CREATE TABLE Department (
   Code INT,
   Name VARCHAR(20),
   ManagerID INT,
   
PRIMARY KEY (Code)
);
```

Check   Show answer

2) Name the CHECK column constraint 'CheckManager'

```
CREATE TABLE Department (
   Code INT,
   Name VARCHAR(20),
   ManagerID INT

                       CHECK
(ManagerID > 999),
   PRIMARY KEY (Code)
);
```

Check   Show answer

3) Add a UNIQUE constraint called UniqueNameMgr that ensures the Name and ManagerID combination are unique.

```
CREATE TABLE Department (
   Code TINYINT UNSIGNED,
   Name VARCHAR(20),
   ManagerID SMALLINT,
```

```
,
   PRIMARY KEY (Code)
);
```

Check   Show answer

Adding and dropping constraints

Constraints are added and dropped with the `ALTER TABLE TableName` followed by an ADD, DROP, or CHANGE clause.

Unnamed constraints such as NOT NULL and DEFAULT are added or dropped with a CHANGE clause:

- `CHANGE CurrentColumnName NewColumnName NewDataType [ConstraintDeclarat`

Named constraints are added with an ADD clause:

- `ADD [CONSTRAINT ConstraintName] PRIMARY KEY (Column1, Column2 ...)`
- `ADD [CONSTRAINT ConstraintName] FOREIGN KEY (Column1, Column2 ...) RE`
- `ADD [CONSTRAINT ConstraintName] UNIQUE (Column1, Column2 ...)`
- `ADD [CONSTRAINT ConstraintName] CHECK (expression)`

Adding a constraint fails when the table contains data that violates the constraint.

Named constraints are dropped with a DROP clause:

- `DROP PRIMARY KEY`
- `DROP FOREIGN KEY ConstraintName`
- `DROP INDEX ConstraintName` (drops UNIQUE constraints)
- `DROP CHECK ConstraintName`
- `DROP CONSTRAINT ConstraintName` (drops any named constraint)

Dropping a table fails when a foreign key constraint refers to the table's primary key. Before dropping the table, either the foreign key constraint or the foreign key table must be dropped.

PARTICIPATION ACTIVITY | 8.12.8: Adding and dropping constraints.

Animation captions:

PARTICIPATION ACTIVITY | 8.12.9: Adding and dropping constraints.

1) Add a NOT NULL constraint to an existing column Salary in the Department table.

```
ALTER TABLE Department
                      Salary
INT NOT NULL;
```

Check   Show answer

2) Add a constraint called MgrIDCheck to the

existing Department table to ensure ManagerID is 2000 or above.

```
ALTER TABLE Department

(ManagerID >= 2000);
```

Check   Show answer

3) Drop the UNIQUE constraint called UniqueNameMgr from the existing Department table.

```
ALTER TABLE Department

;
```

Check   Show answer

CHALLENGE ACTIVITY 8.12.1: Constraints.

544874.3500394.qx3zqy7

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

Exploring further:

- MySQL constraint handling
- MySQL CHECK constraint
- MySQL ALTER TABLE statement

8.13 Inserting, updating, and deleting rows

INSERT statement

The ***INSERT*** statement adds rows to a table. The INSERT statement has two clauses:

- The ***INSERT INTO*** clause names the table and columns where data is to be added. The keyword INTO is optional.
- The ***VALUES*** clause specifies the column values to be added.

The VALUES clause may list any number of rows in parentheses to insert multiple rows.

Figure 8.13.1: INSERT syntax.

```
INSERT [INTO] TableName (Column1, Column2,
...)
VALUES (Value1, Value2, ...);
```

PARTICIPATION ACTIVITY 8.13.1: Inserting rows into the Employee table.

Employee

ID	Name	Salary

Animation captions:

PARTICIPATION ACTIVITY | 8.13.2: INSERT statement.

Refer to the table definition below.

```
CREATE TABLE Department (
   Code TINYINT UNSIGNED,
   Name VARCHAR(20),
   ManagerID SMALLINT UNSIGNED
);
```

1) Which statement correctly inserts Engineering?

- ○
```
INSERT INTO Department
Code, Name, ManagerID
VALUES (44,
'Engineering', 2538);
```
- ○
```
INSERT INTO (Code, Name,
ManagerID)
VALUES (44,
'Engineering', 2538);
```
- ○
```
INSERT INTO Department
(Code, Name, ManagerID)
VALUES (44,
'Engineering', 2538);
```

2) Which statement correctly inserts Sales?

- ○
```
INSERT Department
VALUES (82, 'Sales',
6381);
```
- ○
```
INSERT INTO Department
VALUES ('Sales', 82,
6381);
```
- ○
```
INSERT INTO Department
(82, 'Sales', 6381);
```

3) Which statement correctly inserts Marketing?

- ○
```
INSERT INTO Department
(Name, ManagerID, Code)
VALUES (12, 'Marketing',
6381);
```

○
```
INSERT INTO Department
(Name, ManagerID, Code)
VALUES ('Marketing',
6381, 12);
```

○
```
INSERT INTO Department
(Name, ManagerID, Code)
VALUES (6381, 12,
'Marketing');
```

4) Which statement correctly inserts Technical Support?

○
```
INSERT INTO Department
(Code, Name)
VALUES (99);
```

○
```
INSERT INTO Department
(Code, Name)
VALUES (99, 'Technical
Support');
```

○
```
INSERT INTO Department
(Code, Name)
VALUES (99, 'Technical
Support', NULL);
```

DEFAULT values

Columns may be omitted from an INSERT statement. When omitted, a column is assigned a NULL value. If the NOT NULL constraint is specified on the column, the insert is rejected.

Alternatively, a default value may be specified for a column. The optional ***DEFAULT*** keyword and default value follow the column name and data type in a CREATE TABLE statement. The column is assigned the default value, rather than NULL, when omitted from an INSERT statement.

PARTICIPATION ACTIVITY | 8.13.3: DEFAULT values.

Animation captions:

PARTICIPATION ACTIVITY 8.13.4: DEFAULT values.

Refer to the statement below.

```
CREATE TABLE Department (
   Code      SMALLINT UNSIGNED DEFAULT 1000,
   Name      VARCHAR(20),
   ManagerID SMALLINT
);
```

1) Which alteration to the CREATE TABLE statement sets the default Name to Accounting?

- `Name VARCHAR(20) DEFAULT Accounting,`
- `Name VARCHAR(20) DEFAULT 'Accounting',`
- `Name VARCHAR(20) 'Accounting',`

2) What columns are NULL after the following statement executes?

```
INSERT INTO Department (Code)
VALUES (924);
```

- Name only
- ManagerID only
- Name and ManagerID

UPDATE statement

The ***UPDATE*** statement modifies existing rows in a table. The UPDATE statement uses the ***SET*** clause to specify the new column values. An optional WHERE clause specifies which rows are updated. Omitting the WHERE clause results in all rows being updated.

Figure 8.13.2: UPDATE syntax.

```
UPDATE TableName
SET Column1 = Value1, Column2 = Value2,
...
WHERE condition;
```

PARTICIPATION ACTIVITY 8.13.5: Updating rows in the Employee table.

Employee

ID	Name	BirthDate	Salary
2538	Lisa Ellison	1993-10-02	45000
5384	Sam Snead	1995-03-15	30500
6381	Maria Rodriguez	2001-12-21	92300

Animation captions:

PARTICIPATION ACTIVITY 8.13.6: Update the Song table.

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists

If an issue persists,

send feedback to zyBooks support

PARTICIPATION ACTIVITY 8.13.7: UPDATE statement.

Refer to the Department table.

Department

Code	Name	ManagerID
44	Engineering	2538
82	Sales	6381
12	Marketing	6381
99	Technical Support	NULL

1) What is missing to change "Sales" to "Custodial"?

```
UPDATE Department
SET ___
WHERE Code = 82;
```

- ○ 'Sales' = 'Custodial'
- ○ Department = 'Custodial'
- ○ Name = 'Custodial'

2) What departments are changed?

```
UPDATE Department
SET Name = 'Administration'
WHERE ManagerID IS NOT NULL;
```

- ○ Administration
- ○ All departments except Technical Support
- ○ All departments

3) What is missing to change Marketing's Code to 55 and Name to Administration?

```
UPDATE Department
SET Code = 55, Name =
'Administration'
WHERE ___;
```

- Code = 55
- ManagerID = 6381
- Code = 12

4) What department managers are changed?

```
UPDATE Department
SET ManagerID = 2538;
```

- All managers
- No managers
- Only the Engineering manager

DELETE statement

The ***DELETE*** statement deletes existing rows in a table. The ***FROM*** keyword is followed by the table name whose rows are to be deleted. An optional WHERE clause specifies which rows should be deleted. Omitting the WHERE clause results in all rows in the table being deleted.

Figure 8.13.3: DELETE syntax.

```
DELETE FROM
TableName
WHERE condition;
```

PARTICIPATION ACTIVITY 8.13.8: Deleting rows from the Employee table.

Employee

ID	Name	BirthDate	Salary
2538	Lisa Ellison	1993-10-02	45000
5384	Sam Snead	1995-03-15	30500
6381	Maria Rodriguez	2001-12-21	72300

Animation captions:

PARTICIPATION ACTIVITY | 8.13.9: DELETE statement.

Refer to the Department table.

Department

Code	Name	ManagerID
44	Engineering	2538
82	Sales	6381
12	Marketing	6381
99	Technical Support	NULL

1) What departments are deleted?

```
DELETE FROM Department
WHERE ManagerID = 6381;
```

- Sales only
- Marketing only
- Sales and Marketing

2) What departments are deleted?

```
DELETE FROM Department;
```

- All departments
- No departments

○ Engineering only

3) What is missing to delete only Sales?

```
DELETE FROM Department
WHERE ___;
```

○ ManagerID = 6381

○ Code = 82

○ Code = 'Sales'

TRUNCATE

*The **TRUNCATE** statement deletes all rows from a table. TRUNCATE is nearly identical to a DELETE statement with no WHERE clause except for minor differences that depend on the database system.*

```
TRUNCATE TABLE TableName;
```

CHALLENGE ACTIVITY 8.13.1: Inserting, updating, and deleting rows.

544874.3500394.qx3zqy7

This activity failed to load. Please try refreshing the page. If that fails, you might also try clearing your browser's cache.

If an issue persists,

send feedback to zyBooks support

Exploring further:

- MySQL INSERT syntax
- MySQL UPDATE syntax
- MySQL DELETE syntax

8.14 LAB - Create Horse table with constraints

Create a **Horse** table with the following columns, data types, and constraints. NULL is allowed unless 'not NULL' is explicitly stated.

- **ID** - integer with range 0 to 65535, auto increment, primary key
- **RegisteredName** - variable-length string with max 15 chars, not NULL
- **Breed** - variable-length string with max 20 chars, must be one of the following: Egyptian Arab, Holsteiner, Quarter Horse, Paint, Saddlebred
- **Height** - number with 3 significant digits and 1 decimal place, must be ≥ 10.0 and ≤ 20.0
- **BirthDate** - date, must be ≥ Jan 1, 2015

Notes: Not all constraints can be tested due to current limitations of MySQL. Your SQL code does not display any results in Develop mode. Use Submit mode to test your code.

544874.3500394.qx3zqy7

LAB ACTIVITY | 8.14.1: LAB - Create Horse table with constraints | 10 / 10 ✓

Main.sql

Load default template...

```
Loading latest submission...
```

Develop mode **Submit mode**

Explore the database and run your program as often as you'd like, before submitting for grading. Click **Run**

program and observe the program's output in the second box.

Run program

Main.sql
(Your program)

→ Output (shown below)

Program output displayed here

Coding trail of your work What is this?

```
2/27 T-4,8,0,8,10 U10 min:5
```

8.15 LAB - Create Student table with constraints

Create a **Student** table with the following column names, data types, and constraints:

- **ID** - integer with range 0 to 65 thousand, auto increment, primary key
- **FirstName** - variable-length string with max 20 chars, not NULL
- **LastName** - variable-length string with max 30 chars, not NULL
- **Street** - variable-length string with max 50 chars, not NULL
- **City** - variable-length string with max 20 chars, not NULL
- **State** - fixed-length string of 2 chars, not NULL, default "TX"
- **Zip** - integer with range 0 to 16 million, not NULL
- **Phone** - fixed-length string of 10 chars, not NULL
- **Email** - variable-length string with max 30 chars, must be unique

Note: Your SQL code does not display any results in Develop mode. Use Submit mode to test your code.

544874.3500394.qx3zqy7

LAB ACTIVITY | 8.15.1: LAB - Create Student table with constraints 10 / 10 ✓

Main.sql

```
Loading latest submission...
```

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?

Retrieving signature

8.16 LAB - Create LessonSchedule table with FK constraints

The database contains a **Horse** table, with columns:

- **ID** - integer, primary key
- **RegisteredName** - variable-length string

The database contains a **Student** table, with columns:

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

Create a third table, named **LessonSchedule**, with columns:

- **HorseID** - integer with range 0 to 65 thousand, not NULL, foreign key references Horse(ID)
- **StudentID** - integer with range 0 to 65 thousand, foreign key references Student(ID)
- **LessonDateTime** - date/time, not NULL
- Primary key is **(HorseID, LessonDateTime)**

If a row is deleted from Horse, the rows with the same horse ID should be deleted from LessonSchedule automatically.

If a row is deleted from Student, the same student IDs should be set to NULL in LessonSchedule automatically.

Notes: Table and column names are case sensitive in the auto-grader. Your SQL code does not display any results in Develop mode. Use Submit mode to test your code.

544874.3500394.qx3zqy7

LAB ACTIVITY 8.16.1: LAB - Create LessonSchedule table with FK constraints 10 / 10

Main.sql **Load default template...**

```
CREATE TABLE Horse (
   ID                SMALLINT UNSIGNED AUTO_INCREMENT,
   RegisteredName    VARCHAR(15),
   PRIMARY KEY (ID)
);

CREATE TABLE Student (
   ID               SMALLINT UNSIGNED AUTO_INCREMENT,
   FirstName        VARCHAR(20),
   LastName         VARCHAR(30),
   PRIMARY KEY (ID)
);

```

```
-- Your SQL statements go here

```

Develop mode | Submit mode

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

Run program

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

Program output displayed here

Coding trail of your work What is this?

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

8.17 LAB - Insert rows into Horse table

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

- **ID** - integer, auto increment, primary key
- **RegisteredName** - variable-length string
- **Breed** - variable-length string, must be one of the following: Egyptian Arab, Holsteiner, Quarter Horse, Paint, Saddlebred
- **Height** - decimal number, must be between 10.0 and 20.0
- **BirthDate** - date, must be on or after Jan 1, 2015

Insert the following data into the Horse table:

RegisteredName	Breed	Height	BirthDate
Babe	Quarter Horse	15.3	2015-02-10
Independence	Holsteiner	16.0	2017-03-13
Ellie	Saddlebred	15.0	2016-12-22

NULL	Egyptian Arab	14.9	2019-10-12

Notes:

- Your SQL code does not display any results in Develop mode. Use Submit mode to test your code.
- In another lab that creates the Horse table, RegisteredName is NOT NULL. In this lab, NULL is allowed in this column.

544874.3500394.qx3zqy7

LAB ACTIVITY 8.17.1: LAB - Insert rows into Horse table 10 / 10 ✓

Main.sql **Load default template...**

```
Loading latest submission...
```

Develop mode **Submit mode**

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

Run program

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

Program output displayed here

Coding trail of your work   What is this?

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

8.18 LAB - Update rows in Horse table

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

- **ID** - integer, auto increment, primary key
- **RegisteredName** - variable-length string
- **Breed** - variable-length string, must be one of the following: Egyptian Arab, Holsteiner, Quarter Horse, Paint, Saddlebred
- **Height** - decimal number, must be ≥ 10.0 and ≤ 20.0
- **BirthDate** - date, must be ≥ Jan 1, 2015

Make the following updates:

1. Change the height to **15.6** for horse with ID 2.
2. Change the registered name to **Lady Luck** and birth date to **May 1, 2015** for horse with ID 4.
3. Change every horse breed to ***NULL*** for horses born on or after December 22, 2016.

Note: Your SQL code does not display any results in Develop mode. Use Submit mode to test your code.

544874.3500394.qx3zqy7

LAB ACTIVITY | 8.18.1: LAB - Update rows in Horse table   10 / 10 ✓

Main.sql   Load default template...

```sql
-- Your SQL statements goes here
-- Change the height to 15.6 for horse with ID 2
UPDATE Horse SET Height = 15.6 WHERE ID = 2;

-- Change the registered name to 'Lady Luck' and birth date to May 1, 2015
UPDATE Horse SET RegisteredName = 'Lady Luck', BirthDate = '2015-05-01' WHE

-- Change every horse breed to NULL for horses born on or after December 22
-- Change every horse breed to NULL for horses born on or after December 22
UPDATE Horse SET Breed = NULL WHERE BirthDate >= '2016-12-22';

```

Develop mode **Submit mode**

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

Run program

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

Program output displayed here

Coding trail of your work [What is this?]

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

8.19 LAB - Delete rows from Horse table

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

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

Delete the following rows:

1. Horse with ID 5.
2. All horses with breed Holsteiner or Paint.
3. All horses born before March 13, 2013

3. All horses born before March 13, 2013.

Note: Your SQL code does not display any results in Develop mode. Use Submit mode to test your code.

544874.3500394.qx3zqy7

LAB ACTIVITY 8.19.1: LAB - Delete rows from Horse table 10 / 10 ✓

Main.sql Load default template...

```sql
-- Your SQL statements goes here
-- Delete the horse with ID 5
DELETE FROM Horse WHERE ID = 5;

-- Delete all horses with breed Holsteiner or Paint
DELETE FROM Horse WHERE Breed IN ('Holsteiner', 'Paint');

-- Delete all horses born before March 13, 2013
DELETE FROM Horse WHERE BirthDate < '2013-03-13';

```

Develop mode **Submit mode**

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

Run program

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

Program output displayed here

Coding trail of your work What is this?

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

8.20 LAB - Select horses with logical operators

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, height, and birth date for only horses that have a height between 15.0 and 16.0 (inclusive) or have a birth date on or after January 1, 2020.

544874.3500394.qx3zqy7

LAB ACTIVITY | 8.20.1: LAB - Select horses with logical operators | 10 / 10

Main.sql

Load default template...

```
-- Your SELECT statement goes here
SELECT RegisteredName, Height, BirthDate
FROM Horse
WHERE (Height BETWEEN 15.0 AND 16.0)
   OR (BirthDate >= '2020-01-01');

```

Develop mode | **Submit mode**

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

Run program

Main.sql
(Your program)

Output (shown below)

Program output displayed here

Coding trail of your work What is this?

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

8.21 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.

544874.3500394.qx3zqy7

LAB ACTIVITY | 8.21.1: LAB - Select movie ratings with left join | 10 / 10 ✓

Main.sql Load default template...

```
Your SELECT statement goes here
```

```
-- Your SELECT statement goes here
SELECT Movie.Title, Movie.Year, Rating.Description
FROM Movie
LEFT JOIN Rating ON Movie.RatingCode = Rating.Code;

```

Develop mode Submit mode

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

Run program

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

Program output displayed here

Coding trail of your work What is this?

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

8.22 LAB - Create Movie table

Create a **Movie** table with the following columns:

- **ID** - positive integer with maximum value of 50,000
- **Title** - variable-length string with up to 50 characters
- **Rating** - fixed-length string with 4 characters
- **ReleaseDate** - date

- **Budget** - decimal value representing a cost of up to 999,999 dollars, with 2 digits for cents

Note: Your SQL code does not display any results in Develop mode. Use Submit mode to test your code.

544874.3500394.qx3zqy7

LAB ACTIVITY | 8.22.1: LAB - Create Movie table | 10 / 10 ✓

Main.sql | Load default template...

```
Loading latest submission...
```

Develop mode | Submit mode

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

Run program

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

Program output displayed here

Coding trail of your work What is this?

```
2/27 T6,8,10 min:2
```

8.23 LAB - Alter Movie table

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

- **ID** - positive integer
- **Title** - variable-length string
- **Genre** - variable-length string
- **RatingCode** - variable-length string
- **Year** - integer

Write ALTER statements to make the following modifications to the Movie table:

1. Add a **Producer** column with VARCHAR data type (max 50 chars).
2. Remove the **Genre** column.
3. Change the Year column's name to **ReleaseYear**, and change the data type to **SMALLINT**.

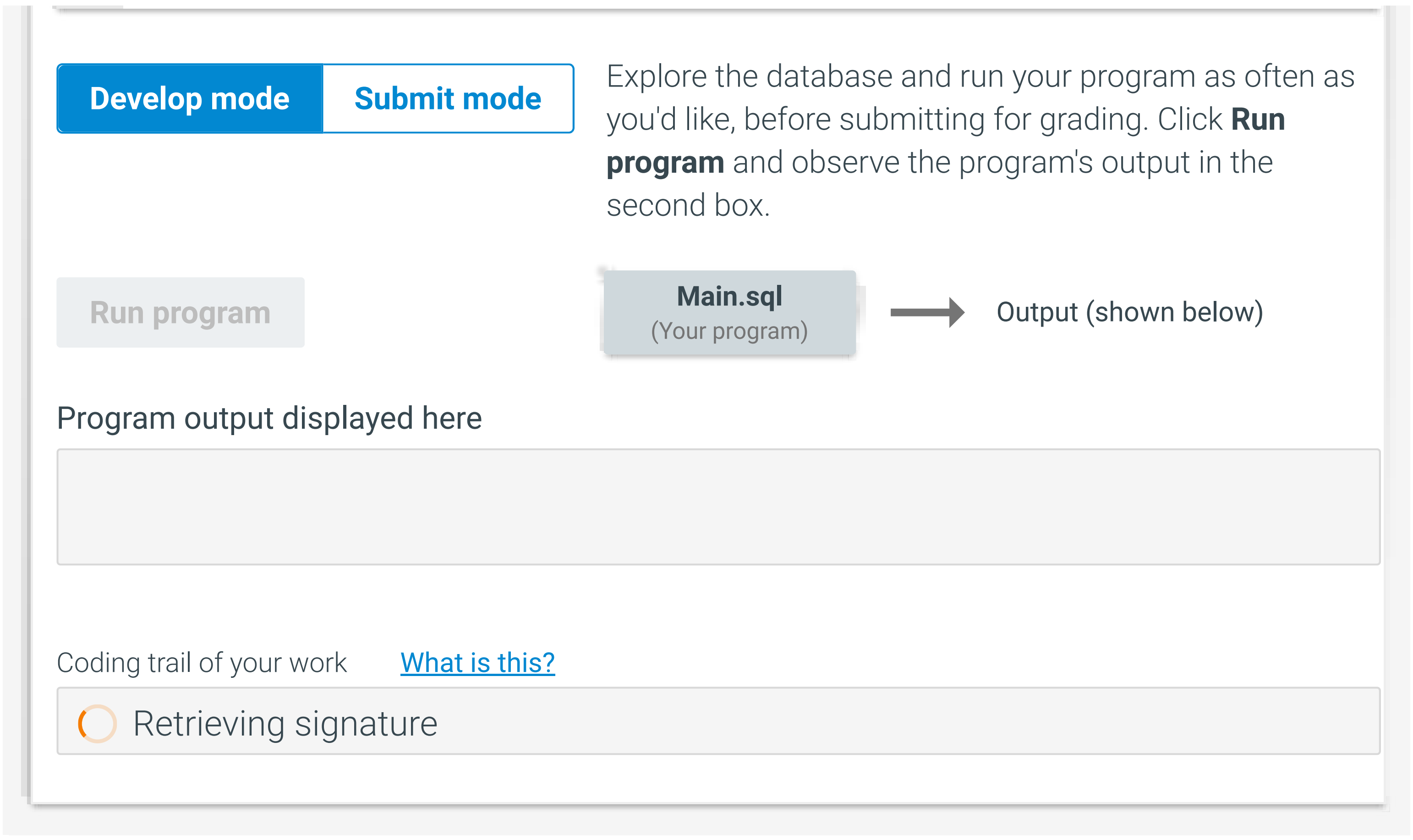
Note: Your SQL code does not display any results in Develop mode. Use Submit mode to test your code.

544874.3500394.qx3zqy7

LAB ACTIVITY 8.23.1: LAB - Alter Movie table 10 / 10 ✓

Main.sql

```
Loading latest submission...
```



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.

Program output displayed here

Coding trail of your work What is this?

Retrieving signature