

Need for a database:

In the realm of modern computing, databases play a pivotal role in managing and organizing large volumes of data efficiently. The primary need for a database arises from the necessity to store, retrieve, and manipulate data systematically. Databases ensure data integrity and security, allowing organizations to maintain consistent and accurate data over time. By centralizing data storage, databases eliminate redundancy and minimize the risk of data inconsistencies, which can occur in file-based systems. Moreover, databases support complex querying capabilities, enabling users to extract meaningful insights and generate reports critical for decision-making processes. In an increasingly data-driven world, the reliability and performance offered by databases are essential for businesses, academic institutions, and governments to function effectively.

Furthermore, databases facilitate concurrent access by multiple users, which is crucial for collaborative environments. Advanced features such as transaction management and concurrency control ensure that simultaneous operations do not lead to data corruption or loss. Databases also provide mechanisms for data backup and recovery, safeguarding against accidental deletions, system failures, or disasters. This robustness is particularly vital for mission-critical applications where data availability and integrity are non-negotiable. As data volumes grow exponentially, the scalability of databases allows organizations to expand their storage capabilities without compromising performance, thereby supporting long-term growth and adaptability.

Difference Between SQL and NoSQL Databases

SQL (Structured Query Language) Databases: SQL databases are relational database management systems (RDBMS) that use structured query language (SQL) for defining and manipulating data. These databases are built on a table-based schema, where data is organized into rows and columns. SQL databases excel in handling structured data and support ACID (Atomicity, Consistency, Isolation, Durability) transactions, which ensure reliable processing of transactions even in the event of errors, power failures, or other issues. Examples of SQL databases include MySQL, PostgreSQL, Oracle, and Microsoft SQL Server. These databases are ideal for applications requiring complex queries, relationships between data, and transactional consistency, such as financial systems, inventory management, and customer relationship management (CRM) systems.

NoSQL (Not Only SQL) Databases: NoSQL databases encompass a variety of database technologies designed to handle unstructured, semi-structured, and structured data. Unlike SQL databases, NoSQL databases do not rely on a fixed schema and are designed for distributed data stores. They can be categorized into four main types: document stores (e.g., MongoDB, CouchDB), key-value stores (e.g., Redis, DynamoDB), column-family stores (e.g., Cassandra, HBase), and graph databases (e.g., Neo4j). NoSQL databases are highly scalable and can handle massive volumes of data with low latency, making them suitable for big data applications, real-time web applications, and Internet of Things (IoT) devices. They often

provide eventual consistency rather than strict ACID compliance, prioritizing availability and partition tolerance as per the CAP theorem.

In summary, the choice between SQL and NoSQL databases depends on the specific needs of the application. SQL databases are preferred for structured data and complex transactional operations, while NoSQL databases are favored for their scalability, flexibility, and ability to handle diverse data types in distributed environments.

Sample Basic Queries and execution:

Create Database: This step creates a new database where all tables and data will reside.

```
1 • CREATE DATABASE LibraryDB1;
2 • USE LibraryDB1;
3
```

Output			
Action Output			
#	Time	Action	Message
1	17:13:03	CREATE DATABASE LibraryDB1	1 row(s) affected
2	17:13:03	USE LibraryDB1	0 row(s) affected

Create Tables with Constraints

Explanation: This step creates tables with various constraints to ensure data integrity. Constraints include PRIMARY KEY, UNIQUE, NOT NULL, FOREIGN KEY, and CHECK.

```
1 -- Books table
2 • CREATE TABLE Books (
3     BookID INT AUTO_INCREMENT PRIMARY KEY,
4     Title VARCHAR(100) NOT NULL,
5     Author VARCHAR(100),
6     PublishedYear INT,
7     Genre VARCHAR(50),
8     Price DECIMAL(10, 2) DEFAULT 0.00,
9     ISBN VARCHAR(13) UNIQUE
10 );
11
12 -- Members table
13 • CREATE TABLE Members (
14     MemberID INT AUTO_INCREMENT PRIMARY KEY,
15     FirstName VARCHAR(50) NOT NULL,
16     LastName VARCHAR(50) NOT NULL,
```

```

13 • CREATE TABLE Members (
14     MemberID INT AUTO_INCREMENT PRIMARY KEY,
15     FirstName VARCHAR(50) NOT NULL,
16     LastName VARCHAR(50) NOT NULL,
17     BirthDate DATE,
18     Email VARCHAR(100) UNIQUE,
19     PhoneNumber VARCHAR(15),
20     MembershipStartDate DATE DEFAULT (CURRENT_DATE)
21 );
22
23 -- Loans table
24 • CREATE TABLE Loans (
25     LoanID INT AUTO_INCREMENT PRIMARY KEY,
26     BookID INT,
27     MemberID INT,
28     LoanDate DATE DEFAULT (CURRENT_DATE),
29     ReturnDate DATE,
30     FOREIGN KEY (BookID) REFERENCES Books(BookID),
31     FOREIGN KEY (MemberID) REFERENCES Members(MemberID),
32     CHECK (ReturnDate >= LoanDate)
33 );

```

```

13 • CREATE TABLE Members (
14     MemberID INT AUTO_INCREMENT PRIMARY KEY,
15     FirstName VARCHAR(50) NOT NULL,
16     LastName VARCHAR(50) NOT NULL,
17     BirthDate DATE,
18     Email VARCHAR(100) UNIQUE,
19     PhoneNumber VARCHAR(15),
20     MembershipStartDate DATE DEFAULT (CURRENT_DATE)
21 );
22
23
24
25
26
27
28
29
30
31
32
33

```

Output

Action Output	#	Time	Action	Message
	1	17:13:03	CREATE DATABASE LibraryDB1	1 row(s) affected
	2	17:13:03	USE LibraryDB1	0 row(s) affected
	3	17:14:02	CREATE TABLE Books (BookID INT AUTO_INCREMENT PRIMARY KEY, Title...)	0 row(s) affected
	4	17:14:02	CREATE TABLE Members (MemberID INT AUTO_INCREMENT PRIMARY KEY, ...)	0 row(s) affected
	5	17:14:02	CREATE TABLE Loans (LoanID INT AUTO_INCREMENT PRIMARY KEY, Boo...)	0 row(s) affected

Insert Data

Explanation: This step adds records to the tables.

```
Query 1 ×
| ⚡ | 🔍 | 🗃 | 📁 | ⌛ | ✅ | ✖ | ⚙ | Limit to 1000 rows | ⭐ | 🐧 | 🔎 | 📄 | + |
1 • INSERT INTO Books (Title, Author, PublishedYear, Genre, Price, ISBN) VALUES
2   ('To Kill a Mockingbird', 'Harper Lee', 1960, 'Fiction', 10.99, '9780061120084'),
3   ('1984', 'George Orwell', 1949, 'Dystopian', 8.99, '9780451524935'),
4   ('Moby Dick', 'Herman Melville', 1851, 'Adventure', 12.99, '9781503280786'),
5   ('The Great Gatsby', 'F. Scott Fitzgerald', 1925, 'Classic', 9.99, '9780743273565'),
6   ('Harry Potter and the Philosopher's Stone', 'J.K. Rowling', 1997, 'Fantasy', 11.99, '9780590353427'),
7   ('The Catcher in the Rye', 'J.D. Salinger', 1951, 'Literary Fiction', 7.99, '9780316769488'),
8   ('The Hobbit', 'J.R.R. Tolkien', 1937, 'Fantasy', 10.49, '9780618260300'),
9   ('Brave New World', 'Aldous Huxley', 1932, 'Dystopian', 8.49, '9780060850524'),
10  ('Pride and Prejudice', 'Jane Austen', 1813, 'Romance', 9.99, '9780141439518'),
11  ('Jane Eyre', 'Charlotte Bronte', 1847, 'Gothic Fiction', 7.49, '9780141441146');
12
13 -- Insert data into Members
14 • INSERT INTO Members (FirstName, LastName, BirthDate, Email, PhoneNumber) VALUES
15   ('John', 'Doe', '1985-05-15', 'john.doe@example.com', '555-1234'),
16   ('Jane', 'Smith', '1990-07-25', 'jane.smith@example.com', '555-5678'),
17   ('Michael', 'Johnson', '1982-09-30', 'michael.johnson@example.com', '555-9876'),
18   ('Emily', 'Brown', '1995-03-20', 'emily.brown@example.com', '555-4321'),
```

```
Query 1 ×
| ⌂ | ⚡ | ⚡ | 🔎 | ⌚ | ⏺ | ⏻ | Limit to 1000 rows | ⭐ | 📁 | 🔍 | 🗃 | 🖑 | 🗃

12
13    -- Insert data into Members
14 • INSERT INTO Members (FirstName, LastName, BirthDate, Email, PhoneNumber) VALUES
15     ('John', 'Doe', '1985-05-15', 'john.doe@example.com', '555-1234'),
16     ('Jane', 'Smith', '1990-07-25', 'jane.smith@example.com', '555-5678'),
17     ('Michael', 'Johnson', '1982-09-30', 'michael.johnson@example.com', '555-9876'),
18     ('Emily', 'Brown', '1995-03-20', 'emily.brown@example.com', '555-4321'),
19     ('Robert', 'Williams', '1978-12-10', 'robert.williams@example.com', '555-8765');
20
21    -- Insert data into Loans
22 • INSERT INTO Loans (BookID, MemberID, LoanDate, ReturnDate) VALUES
23     (1, 1, '2023-01-01', '2023-01-15'),
24     (2, 2, '2023-01-10', '2023-01-20'),
25     (3, 3, '2023-01-05', '2023-01-25'),
26     (4, 4, '2023-02-01', NULL),
27     (5, 1, '2023-02-10', NULL),
28     (6, 2, '2023-02-15', NULL),
29     (7, 3, '2023-02-20', NULL);
```

Query 1

12
13 -- Insert data into Members
14 • INSERT INTO Members (FirstName, LastName, BirthDate, Email, PhoneNumber) VALUES
15 ('John', 'Doe', '1985-05-15', 'john.doe@example.com', '555-1234'),
16 ('Jane', 'Smith', '1990-07-25', 'jane.smith@example.com', '555-5678'),
17 ('Michael', 'Johnson', '1982-09-30', 'michael.johnson@example.com', '555-9876'),
18 ('Emily', 'Brown', '1995-03-20', 'emily.brown@example.com', '555-4321'),
19 ('Robert', 'Williams', '1978-12-10', 'robert.williams@example.com', '555-8765');
20
21 -- Insert data into Loans

Output

Action Output

#	Time	Action	Message
1	17:13:03	CREATE DATABASE LibraryDB1	1 row(s) affected
2	17:13:03	USE LibraryDB1	0 row(s) affected
3	17:14:02	CREATE TABLE Books (BookID INT AUTO_INCREMENT PRIMARY KEY, Title...)	0 row(s) affected
4	17:14:02	CREATE TABLE Members (MemberID INT AUTO_INCREMENT PRIMARY KEY, ...)	0 row(s) affected
5	17:14:02	CREATE TABLE Loans (LoanID INT AUTO_INCREMENT PRIMARY KEY, Boo...)	0 row(s) affected
6	17:16:28	INSERT INTO Books (Title, Author, PublishedYear, Genre, Price, ISBN) VALUES (To ...)	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0
7	17:16:28	INSERT INTO Members (FirstName, LastName, BirthDate, Email, PhoneNumber) VAL...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0
8	17:16:28	INSERT INTO Loans (BookID, MemberID, LoanDate, ReturnDate) VALUES (1, 1, '20...	7 row(s) affected Records: 7 Duplicates: 0 Warnings: 0

Basic SQL Queries

Explanation: These are simple queries to retrieve, insert, update, and delete data from the tables.

SELECT * FROM Books: This query selects all columns (*) from the Books table.

Query 1 ×

1 • SELECT * FROM Books

Result Grid							Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:
	BookID	Title	Author	PublishedYear	Genre	Price	ISBN			
▶	1	To Kill a Mockingbird	Harper Lee	1960	Fiction	10.99	9780061120084			
	2	1984	George Orwell	1949	Dystopian	8.99	9780451524935			
	3	Moby Dick	Herman Melville	1851	Adventure	12.99	9781503280786			
	4	The Great Gatsby	F. Scott Fitzgerald	1925	Classic	9.99	9780743273565			
	5	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy	11.99	9780590353427			
	6	The Catcher in the Rye	J.D. Salinger	1951	Literary Fiction	7.99	9780316769488			
	7	The Hobbit	J.R.R. Tolkien	1937	Fantasy	10.49	9780618260300			
	8	Brave New World	Aldous Huxley	1932	Dystopian	8.49	9780060850524			
	9	Pride and Prejudice	Jane Austen	1813	Romance	9.99	9780141439518			
	10	Jane Eyre	Charlotte Bronte	1847	Gothic Fiction	7.49	9780141441146			
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL			

Books 1 ×

SELECT DISTINCT: This query selects unique values from the Genre column in the Books table.

The screenshot shows a 'Query 1' window with the following content:

```
1   SELECT DISTINCT Genre FROM Books
```

Result Grid:

Genre
Fiction
Dystopian
Adventure
Classic
Fantasy
Literary Fiction
Romance
Gothic Fiction

WHERE: Filters rows where the Genre column equals 'Fantasy'.

The screenshot shows a 'Query 1' window with the following content:

```
1   SELECT * FROM Books WHERE Genre = 'Fantasy';
```

Result Grid:

BookID	Title	Author	PublishedYear	Genre	Price	ISBN
5	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy	11.99	9780590353427
7	The Hobbit	J.R.R. Tolkien	1937	Fantasy	10.49	9780618260300
*	NULL	NULL	NULL	NULL	NULL	NULL

AND & OR: Filters rows where the Genre is 'Fantasy' and the Price is less than 12.

Query 1

```
1 SELECT * FROM Books WHERE Genre = 'Fantasy' AND Price < 12;
```

Result Grid

BookID	Title	Author	PublishedYear	Genre	Price	ISBN
5	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy	11.99	9780590353427
7	The Hobbit	J.R.R. Tolkien	1937	Fantasy	10.49	9780618260300
*	NULL	NULL	NULL	NULL	NULL	NULL

Filters rows where the Genre is 'Classic' or Genre is 'Literary Fiction'

Query 1

```
1 • SELECT * FROM Books WHERE Genre = 'Classic' OR Genre = 'Literary Fiction';
```

Result Grid

BookID	Title	Author	PublishedYear	Genre	Price	ISBN
4	The Great Gatsby	F. Scott Fitzgerald	1925	Classic	9.99	9780743273565
6	The Catcher in the Rye	J.D. Salinger	1951	Literary Fiction	7.99	9780316769488
*	NULL	NULL	NULL	NULL	NULL	NULL

ORDER BY : Orders the result set by the PublishedYear column in descending order (DESC).

Query 1

```
1 SELECT * FROM Books ORDER BY PublishedYear DESC;
```

Result Grid

BookID	Title	Author	PublishedYear	Genre	Price	ISBN
5	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy	11.99	9780590353427
1	To Kill a Mockingbird	Harper Lee	1960	Fiction	10.99	9780061120084
6	The Catcher in the Rye	J.D. Salinger	1951	Literary Fiction	7.99	9780316769488
2	1984	George Orwell	1949	Dystopian	8.99	9780451524935
7	The Hobbit	J.R.R. Tolkien	1937	Fantasy	10.49	9780618260300
8	Brave New World	Aldous Huxley	1932	Dystopian	8.49	9780060850524
4	The Great Gatsby	F. Scott Fitzgerald	1925	Classic	9.99	9780743273565
3	Moby Dick	Herman Melville	1851	Adventure	12.99	9781503280786
10	Jane Eyre	Charlotte Bronte	1847	Gothic Fiction	7.49	9780141441146
9	Pride and Prejudice	Jane Austen	1813	Romance	9.99	9780141439518

INSERT INTO: Inserts a new row into the Books table with specified values for columns (Title, Author, PublishedYear, Genre, Price, ISBN).

The screenshot shows the MySQL Workbench interface with a query editor titled "Query 1". The query is:

```
1 •  SELECT * FROM Books WHERE ISBN = '9780141439570';  
2
```

The result grid displays the following data:

	BookID	Title	Author	PublishedYear	Genre	Price	ISBN
▶	11	The Picture of Dorian Gray	Oscar Wilde	1890	Gothic Fiction	8.99	9780141439570
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

UPDATE : Updates the Price column to 11.99 for the row where BookID is 1 in the Books table.

The screenshot shows the MySQL Workbench interface with a query editor titled "Query 1" and an "Output" pane. The query is:

```
1 •  UPDATE Books SET Price = 11.99 WHERE BookID = 1;
```

The output pane displays the execution history:

#	Action	Message
3	CREATE TABLE Books (BookID INT AUTO_INCREMENT PRIMARY KEY, ...)	0 row(s) affected
4	CREATE TABLE Members (MemberID INT AUTO_INCREMENT PRIMARY KEY, ...)	0 row(s) affected
5	CREATE TABLE Loans (LoanID INT AUTO_INCREMENT PRIMARY KEY, ...)	0 row(s) affected
6	INSERT INTO Books (Title, Author, PublishedYear, Genre, Price, ISBN) VALUES ('...', ...)	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0
7	INSERT INTO Members (FirstName, LastName, BirthDate, Email, PhoneNumber) V...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0
8	INSERT INTO Loans (BookID, MemberID, LoanDate, ReturnDate) VALUES (1, 1, ...)	7 row(s) affected Records: 7 Duplicates: 0 Warnings: 0
9	SELECT * FROM Books LIMIT 0, 1000	10 row(s) returned
10	SELECT DISTINCT Genre FROM Books LIMIT 0, 1000	8 row(s) returned
11	SELECT * FROM Books WHERE Genre = 'Fantasy' LIMIT 0, 1000	2 row(s) returned
12	SELECT * FROM Books WHERE Genre = 'Fantasy' AND Price < 12 LIMIT 0, 1000	2 row(s) returned
13	SELECT * FROM Books WHERE Genre = 'Classic' OR Genre = 'Literary Fiction' Li...	2 row(s) returned
14	SELECT * FROM Books WHERE Genre = 'Fantasy' AND Price < 12 LIMIT 0, 1000	2 row(s) returned
15	SELECT * FROM Books WHERE Genre = 'Classic' OR Genre = 'Literary Fiction' Li...	2 row(s) returned
16	SELECT * FROM Books ORDER BY PublishedYear DESC LIMIT 0, 1000	10 row(s) returned
17	INSERT INTO Books (Title, Author, PublishedYear, Genre, Price, ISBN) VALUES ('...', ...)	1 row(s) affected
18	SELECT * FROM Books WHERE ISBN = '9780141439570' LIMIT 0, 1000	1 row(s) returned
19	UPDATE Books SET Price = 11.99 WHERE BookID = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

Query 1

1 SELECT * FROM Books

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

	BookID	Title	Author	PublishedYear	Genre	Price	ISBN
▶	1	To Kill a Mockingbird	Harper Lee	1960	Fiction	11.99	9780061120084
	2	1984	George Orwell	1949	Dystopian	8.99	9780451524935
	3	Moby Dick	Herman Melville	1851	Adventure	12.99	9781503280786
	4	The Great Gatsby	F. Scott Fitzgerald	1925	Classic	9.99	9780743273565
	5	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy	11.99	9780590353427
	6	The Catcher in the Rye	J.D. Salinger	1951	Literary Fiction	7.99	9780316769488
	7	The Hobbit	J.R.R. Tolkien	1937	Fantasy	10.49	9780618260300
	8	Brave New World	Aldous Huxley	1932	Dystopian	8.49	9780060850524
	9	Pride and Prejudice	Jane Austen	1813	Romance	9.99	9780141439518
	10	Jane Eyre	Charlotte Bronte	1847	Gothic Fiction	7.49	9780141441146
	11	The Picture of Dorian Gray	Oscar Wilde	1890	Gothic Fiction	8.99	9780141439570
*	HULL	HULL	HULL	HULL	HULL	HULL	HULL

DELETE: Deletes the row from the Books table where BookID is 10.

Query 1

1 DELETE FROM Books WHERE BookID = 10;

Query 1

```
1  SELECT * FROM Books;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Contents: |

	BookID	Title	Author	PublishedYear	Genre	Price	ISBN
▶	1	To Kill a Mockingbird	Harper Lee	1960	Fiction	11.99	9780061120084
	2	1984	George Orwell	1949	Dystopian	8.99	9780451524935
	3	Moby Dick	Herman Melville	1851	Adventure	12.99	9781503280786
	4	The Great Gatsby	F. Scott Fitzgerald	1925	Classic	9.99	9780743273565
	5	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy	11.99	9780590353427
	6	The Catcher in the Rye	J.D. Salinger	1951	Literary Fiction	7.99	9780316769488
	7	The Hobbit	J.R.R. Tolkien	1937	Fantasy	10.49	9780618260300
	8	Brave New World	Aldous Huxley	1932	Dystopian	8.49	9780060850524
	9	Pride and Prejudice	Jane Austen	1813	Romance	9.99	9780141439518
	11	The Picture of Dorian Gray	Oscar Wilde	1890	Gothic Fiction	8.99	9780141439570
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

SQL Injection Prevention with Prepared Statements

Explanation: Demonstrates how to prevent SQL injection attacks by using prepared statements, which separate the query structure from the data. Safely executes a query to select books by genre without exposing the database to SQL injection risks.

Query 1

```
1  PREPARE stmt FROM 'SELECT * FROM Books WHERE Genre = ?';
2  •  SET @genre = 'Dystopian';
3  •  EXECUTE stmt USING @genre;
4  •  DEALLOCATE PREPARE stmt;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Contents: |

	BookID	Title	Author	PublishedYear	Genre	Price	ISBN
▶	2	1984	George Orwell	1949	Dystopian	8.99	9780451524935
	8	Brave New World	Aldous Huxley	1932	Dystopian	8.49	9780060850524

SELECT TOP:

Explanation: Retrieves a specified number of top records from the table. Selects the top 3 most expensive books.

The screenshot shows the MySQL Workbench interface with a query editor titled "Query 1". The query is:

```
1 •  SELECT * FROM Books ORDER BY Price DESC LIMIT 3;
```

The results grid displays the following data:

	BookID	Title	Author	PublishedYear	Genre	Price	ISBN
▶	3	Moby Dick	Herman Melville	1851	Adventure	12.99	9781503280786
	1	To Kill a Mockingbird	Harper Lee	1960	Fiction	11.99	9780061120084
*	5	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy	11.99	9780590353427
*	NUL	NUL	NUL	NUL	NUL	NUL	NUL

LIKE and Wildcards

Explanation: Searches for records that match a pattern.

The screenshot shows the MySQL Workbench interface with a query editor titled "Query 1". The query is:

```
1      SELECT * FROM Books WHERE Title LIKE '%the%';
```

The results grid displays the following data:

	BookID	Title	Author	PublishedYear	Genre	Price	ISBN
▶	4	The Great Gatsby	F. Scott Fitzgerald	1925	Classic	9.99	9780743273565
	5	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy	11.99	9780590353427
	6	The Catcher in the Rye	J.D. Salinger	1951	Literary Fiction	7.99	9780316769488
	7	The Hobbit	J.R.R. Tolkien	1937	Fantasy	10.49	9780618260300
*	11	The Picture of Dorian Gray	Oscar Wilde	1890	Gothic Fiction	8.99	9780141439570
*	NUL	NUL	NUL	NUL	NUL	NUL	NUL

IN and BETWEEN

Explanation: The IN operator checks if a value matches any value in a list, and BETWEEN checks if a value is within a range. The below example demonstrates to find books in specific genres or within a certain publication year range.

Aliases

Explanation: Assigns temporary names to columns or tables for readability.

Query 1

BookTitle	BookAuthor
To Kill a Mockingbird	Harper Lee
1984	George Orwell
Moby Dick	Herman Melville
The Great Gatsby	F. Scott Fitzgerald
Harry Potter and the Philosopher's Stone	J.K. Rowling
The Catcher in the Rye	J.D. Salinger
The Hobbit	J.R.R. Tolkien
Brave New World	Aldous Huxley
Pride and Prejudice	Jane Austen
The Picture of Dorian Gray	Oscar Wilde

Joins

Explanation: Combines records from two or more tables based on related columns.

Inner Join:

- This query retrieves the first name and last name of members along with the titles of the books they have borrowed, by matching records from the **Loans**, **Members**, and **Books** tables.

Query 1

```
1  SELECT Members.FirstName, Members.LastName, Books.Title
2  FROM Loans
3  INNER JOIN Members ON Loans.MemberID = Members.MemberID
4  INNER JOIN Books ON Loans.BookID = Books.BookID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	FirstName	LastName	Title
▶	John	Doe	To Kill a Mockingbird
	John	Doe	Harry Potter and the Philosopher's Stone
	Jane	Smith	1984
	Jane	Smith	The Catcher in the Rye
	Michael	Johnson	Moby Dick
	Michael	Johnson	The Hobbit
	Emily	Brown	The Great Gatsby

Left Join: This query retrieves all members' first and last names along with the titles of books they have borrowed (if any), showing NULL for books if no loans exist.

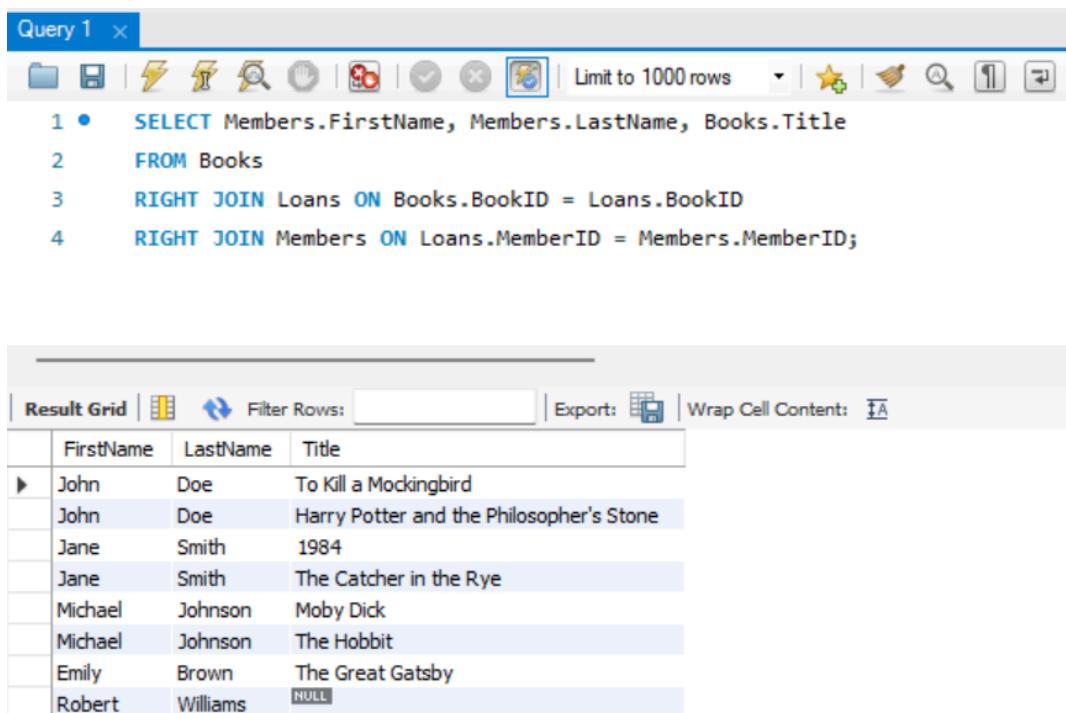
Query 1

```
1 •  SELECT Members.FirstName, Members.LastName, Books.Title
2  FROM Members
3  LEFT JOIN Loans ON Members.MemberID = Loans.MemberID
4  LEFT JOIN Books ON Loans.BookID = Books.BookID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	FirstName	LastName	Title
▶	John	Doe	To Kill a Mockingbird
	John	Doe	Harry Potter and the Philosopher's Stone
	Jane	Smith	1984
	Jane	Smith	The Catcher in the Rye
	Michael	Johnson	Moby Dick
	Michael	Johnson	The Hobbit
	Emily	Brown	The Great Gatsby
	Robert	Williams	NULL

Right Join: This query retrieves the first name and last name of members along with the titles of the books they have borrowed, including all loans even if there are no corresponding book or member records.



The screenshot shows a database query interface with a toolbar at the top containing various icons for file operations, search, and export. The query window displays the following SQL code:

```

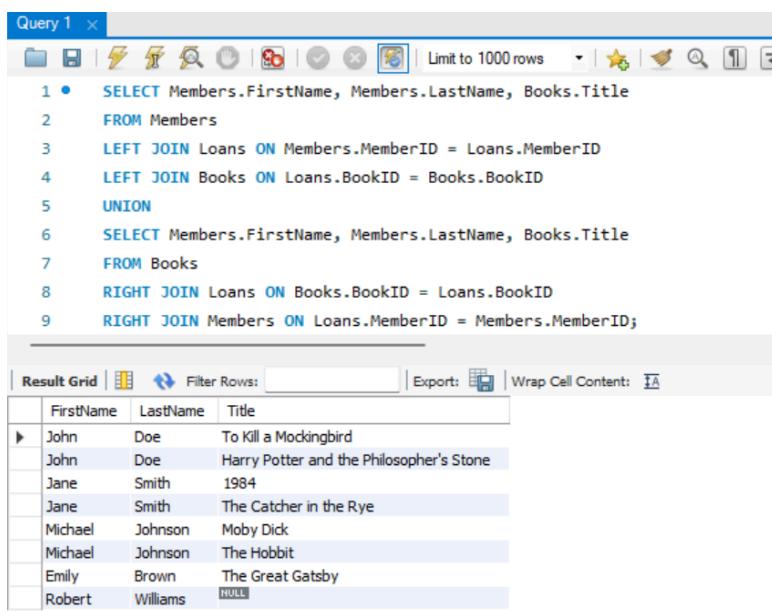
1 •   SELECT Members.FirstName, Members.LastName, Books.Title
2     FROM Books
3     RIGHT JOIN Loans ON Books.BookID = Loans.BookID
4     RIGHT JOIN Members ON Loans.MemberID = Members.MemberID;

```

The result grid shows the following data:

	FirstName	LastName	Title
▶	John	Doe	To Kill a Mockingbird
	John	Doe	Harry Potter and the Philosopher's Stone
	Jane	Smith	1984
	Jane	Smith	The Catcher in the Rye
	Michael	Johnson	Moby Dick
	Michael	Johnson	The Hobbit
	Emily	Brown	The Great Gatsby
	Robert	Williams	NULL

Full Join: This query retrieves all members and the books they have borrowed, including members without loans and books not currently on loan, by combining results from left and right joins using UNION.



The screenshot shows a database query interface with a toolbar at the top containing various icons for file operations, search, and export. The query window displays the following SQL code:

```

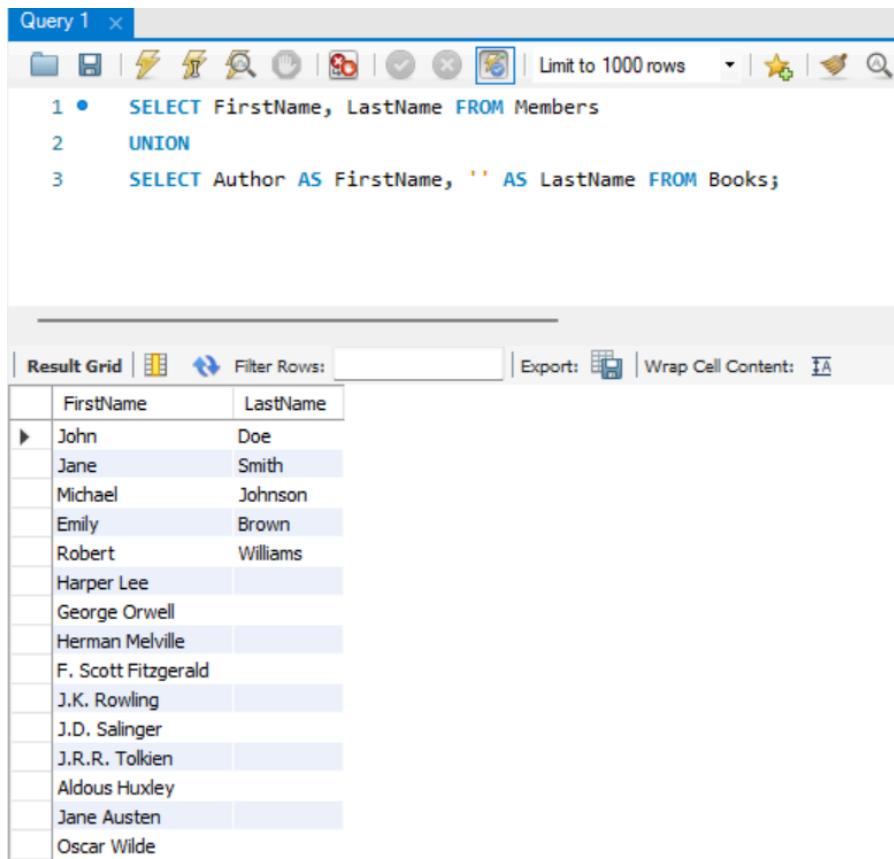
1 •   SELECT Members.FirstName, Members.LastName, Books.Title
2     FROM Members
3     LEFT JOIN Loans ON Members.MemberID = Loans.MemberID
4     LEFT JOIN Books ON Loans.BookID = Books.BookID
5     UNION
6     SELECT Members.FirstName, Members.LastName, Books.Title
7     FROM Books
8     RIGHT JOIN Loans ON Books.BookID = Loans.BookID
9     RIGHT JOIN Members ON Loans.MemberID = Members.MemberID;

```

The result grid shows the same data as the previous screenshot:

	FirstName	LastName	Title
▶	John	Doe	To Kill a Mockingbird
	John	Doe	Harry Potter and the Philosopher's Stone
	Jane	Smith	1984
	Jane	Smith	The Catcher in the Rye
	Michael	Johnson	Moby Dick
	Michael	Johnson	The Hobbit
	Emily	Brown	The Great Gatsby
	Robert	Williams	NULL

Union: This query combines a list of members' first and last names with a list of book authors (using the author's name as the first name and leaving the last name empty), removing any duplicates.



The screenshot shows the MySQL Workbench interface with a query editor titled "Query 1". The query is:

```

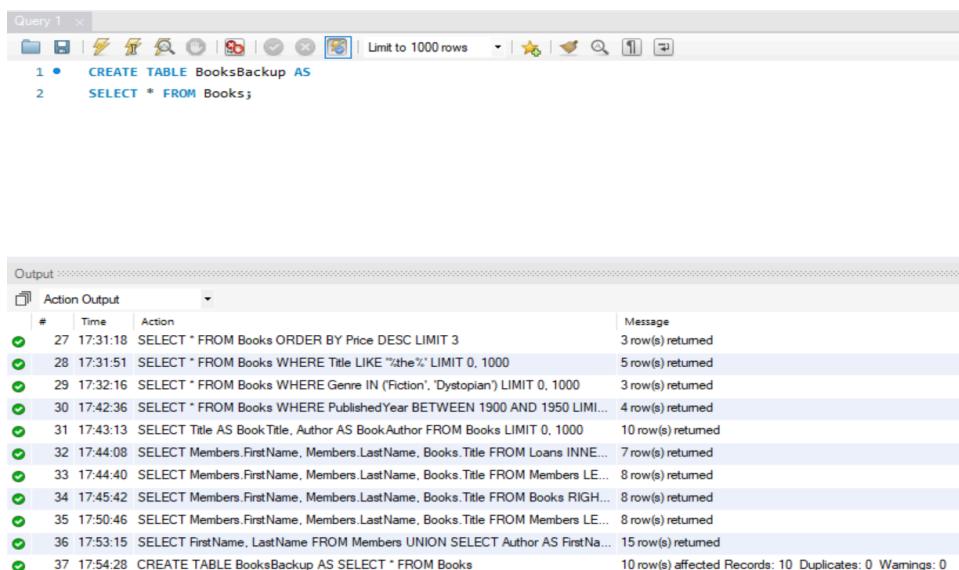
1 •  SELECT FirstName, LastName FROM Members
2 UNION
3      SELECT Author AS FirstName, '' AS LastName FROM Books;

```

The results are displayed in a "Result Grid" table:

FirstName	LastName
John	Doe
Jane	Smith
Michael	Johnson
Emily	Brown
Robert	Williams
Harper Lee	
George Orwell	
Herman Melville	
F. Scott Fitzgerald	
J.K. Rowling	
J.D. Salinger	
J.R.R. Tolkien	
Aldous Huxley	
Jane Austen	
Oscar Wilde	

Select Into: This command creates a new table named BooksBackup and populates it with all the data from the existing Books table.



The screenshot shows the MySQL Workbench interface with a query editor titled "Query 1". The query is:

```

1 •  CREATE TABLE BooksBackup AS
2      SELECT * FROM Books;

```

The results are displayed in an "Output" window under "Action Output".

#	Time	Action	Message
27	17:31:18	SELECT * FROM Books ORDER BY Price DESC LIMIT 3	3 row(s) returned
28	17:31:51	SELECT * FROM Books WHERE Title LIKE '%the%' LIMIT 0, 1000	5 row(s) returned
29	17:32:16	SELECT * FROM Books WHERE Genre IN ('Fiction', 'Dystopian') LIMIT 0, 1000	3 row(s) returned
30	17:42:36	SELECT * FROM Books WHERE PublishedYear BETWEEN 1900 AND 1950 LIMIT 0, 1000	4 row(s) returned
31	17:43:13	SELECT Title AS BookTitle, Author AS BookAuthor FROM Books LIMIT 0, 1000	10 row(s) returned
32	17:44:08	SELECT Members.FirstName, Members.LastName, Books.Title FROM Loans INNER JOIN Books ON Loans.BookID = Books.ID	7 row(s) returned
33	17:44:40	SELECT Members.FirstName, Members.LastName, Books.Title FROM Members LEFT JOIN Books ON Members.MemberID = Books.BookID	8 row(s) returned
34	17:45:42	SELECT Members.FirstName, Members.LastName, Books.Title FROM Books RIGHT JOIN Members ON Books.BookID = Members.MemberID	8 row(s) returned
35	17:50:42	SELECT Members.FirstName, Members.LastName, Books.Title FROM Members LEFT JOIN Books ON Members.MemberID = Books.BookID	8 row(s) returned
36	17:53:15	SELECT FirstName, LastName FROM Members UNION SELECT Author AS FirstName, '' AS LastName FROM Books	15 row(s) returned
37	17:54:28	CREATE TABLE BooksBackup AS SELECT * FROM Books	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0

Insert Into : This SQL statement copies books categorized under 'Fantasy' from the Books table into the BooksBackup table, including all columns (Title, Author, PublishedYear, Genre, Price, ISBN).

INSERT INTO BooksBackup (Title, Author, PublishedYear, Genre, Price, ISBN) SELECT Title, Author, PublishedYear, Genre, Price, ISBN FROM Books WHERE Genre = 'Fantasy';

The screenshot shows a MySQL Workbench interface with a query editor titled 'Query 1'. The query is:

```
1 •  SELECT * FROM BooksBackup
```

The results are displayed in a 'Result Grid' table:

	BookID	Title	Author	PublishedYear	Genre	Price	ISBN
▶	1	To Kill a Mockingbird	Harper Lee	1960	Fiction	11.99	9780061120084
	2	1984	George Orwell	1949	Dystopian	8.99	9780451524935
	3	Moby Dick	Herman Melville	1851	Adventure	12.99	9781503280786
	4	The Great Gatsby	F. Scott Fitzgerald	1925	Classic	9.99	9780743273565
	5	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy	11.99	9780590353427
	6	The Catcher in the Rye	J.D. Salinger	1951	Literary Fiction	7.99	9780316769488
	7	The Hobbit	J.R.R. Tolkien	1937	Fantasy	10.49	9780618260300
	8	Brave New World	Aldous Huxley	1932	Dystopian	8.49	9780060850524
	9	Pride and Prejudice	Jane Austen	1813	Romance	9.99	9780141439518
	11	The Picture of Dorian Gray	Oscar Wilde	1890	Gothic Fiction	8.99	9780141439570
	0	Harry Potter and the Philosopher's Stone	J.K. Rowling	1997	Fantasy	11.99	9780590353427
	0	The Hobbit	J.R.R. Tolkien	1937	Fantasy	10.49	9780618260300

Not Null : This SQL statement modifies the Members table by changing the data type of the Email column to VARCHAR(100) and sets it to be NOT NULL, ensuring all members must have an email address defined.

The screenshot shows a MySQL Workbench interface with a query editor titled 'Query 1'. The query is:

```
1 •  ALTER TABLE Members MODIFY Email VARCHAR(100) NOT NULL;
```

The output window displays the results of previous queries and the execution of the ALTER TABLE command:

Action Output	#	Time	Action	Message
27 17:31:18	SELECT * FROM Books ORDER BY Price DESC LIMIT 3			3 row(s) returned
28 17:31:51	SELECT * FROM Books WHERE Title LIKE "%the%" LIMIT 0, 1000			5 row(s) returned
29 17:32:16	SELECT * FROM Books WHERE Genre IN ('Fiction', 'Dystopian') LIMIT 0, 1000			3 row(s) returned
30 17:42:36	SELECT * FROM Books WHERE PublishedYear BETWEEN 1900 AND 1950 LIMIT 0, 1000			4 row(s) returned
31 17:43:13	SELECT Title AS BookTitle, Author AS BookAuthor FROM Books LIMIT 0, 1000			10 row(s) returned
32 17:44:08	SELECT Members.FirstName, Members.LastName, Books.Title FROM Loans INNER JOIN Members ON Loans.MemberID = Members.MemberID INNER JOIN Books ON Loans.BookID = Books.BookID			7 row(s) returned
33 17:44:40	SELECT Members.FirstName, Members.LastName, Books.Title FROM Members LEFT JOIN Books ON Members.MemberID = Books.BookID			8 row(s) returned
34 17:45:42	SELECT Members.FirstName, Members.LastName, Books.Title FROM Books RIGHT JOIN Members ON Books.BookID = Members.MemberID			8 row(s) returned
35 17:50:46	SELECT Members.FirstName, Members.LastName, Books.Title FROM Members LEFT JOIN Books ON Members.MemberID = Books.BookID			8 row(s) returned
36 17:53:15	SELECT FirstName, LastName FROM Members UNION SELECT Author AS FirstName, BookTitle AS LastName FROM Books			15 row(s) returned
37 17:54:28	CREATE TABLE BooksBackup AS SELECT * FROM Books			10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0
38 17:54:31	CREATE TABLE BooksBackup AS SELECT * FROM Books			Error Code: 1050. Table 'BooksBackup' already exists
39 17:57:40	INSERT INTO BooksBackup (Title, Author, PublishedYear, Genre, Price, ISBN) SELECT * FROM Books LIMIT 0, 1000			2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
40 17:59:02	SELECT * FROM BooksBackup LIMIT 0, 1000			12 row(s) returned
41 18:00:11	ALTER TABLE Members MODIFY Email VARCHAR(100) NOT NULL			0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Query 1

```

1 -- Check the structure of the Members table
2 • DESCRIBE Members;
3

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	Field	Type	Null	Key	Default	Extra
▶	MemberID	int	NO	PRI	NULL	auto_increment
	FirstName	varchar(50)	NO		NULL	
	LastName	varchar(50)	NO		NULL	
	BirthDate	date	YES		NULL	
	Email	varchar(100)	NO	UNI	NULL	
	PhoneNumber	varchar(15)	YES		NULL	
	MembershipStartDate	date	YES		curdate()	DEFAULT_GENERATED

Unique : This SQL command adds a unique constraint (UC_Email) on the Email column in the Members table, ensuring that each email address must be unique among all rows in the table.

Query 1

```

1 • SHOW INDEX FROM Members;
2

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_t
▶	members	0	PRIMARY	1	MemberID	A	5				BTREE
	members	0	Email	1	Email	A	5				BTREE
	members	0	UC_Email	1	Email	A	5				BTREE

Check : This SQL statement adds a constraint to the Loans table ensuring that the ReturnDate must be greater than or equal to the LoanDate for all records.

Query 1

```
1 • ALTER TABLE Loans ADD CONSTRAINT CHK_ReturnDate CHECK (ReturnDate >= LoanDate);
```

Output:

#	Time	Action	Message
47	18:08:54	DROP INDEX UC_Email ON Members	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
48	18:09:06	ALTER TABLE Members ADD CONSTRAINT UC_Email UNIQUE (Email)	0 row(s) affected, 1 warning(s): 1831 Duplicate index 'UC_Email'
49	18:10:14	SHOW INDEX FROM Members	3 row(s) returned
50	18:11:16	ALTER TABLE Members DROP INDEX UC_Email	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
51	18:11:22	SHOW INDEX FROM Members	2 row(s) returned
52	18:12:24	SHOW INDEX FROM Members	2 row(s) returned
53	18:12:24	ALTER TABLE Members ADD CONSTRAINT UC_Email UNIQUE (Email)	0 row(s) affected, 1 warning(s): 1831 Duplicate index 'UC_Email'
54	18:12:48	SHOW WARNINGS	1 row(s) returned
55	18:14:19	SHOW INDEX FROM Members	3 row(s) returned
56	18:16:32	ALTER TABLE Loans ADD CONSTRAINT CHK_ReturnDate CHECK (ReturnDate >= LoanDate)	7 row(s) affected Records: 7 Duplicates: 0 Warnings: 0

Default : This SQL statement modifies the Members table schema to set a default value of the current date (CURRENT_DATE) for the MembershipStartDate column.

Query 1

```
1     ALTER TABLE Members MODIFY MembershipStartDate DATE DEFAULT (CURRENT_DATE);
```

Output:

#	Time	Action	Message
48	18:09:06	ALTER TABLE Members ADD CONSTRAINT UC_Email UNIQUE (Email)	0 row(s) affected, 1 warning(s): 1831 Duplicate index 'UC_Email'
49	18:10:14	SHOW INDEX FROM Members	3 row(s) returned
50	18:11:16	ALTER TABLE Members DROP INDEX UC_Email	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
51	18:11:22	SHOW INDEX FROM Members	2 row(s) returned
52	18:12:24	SHOW INDEX FROM Members	2 row(s) returned
53	18:12:24	ALTER TABLE Members ADD CONSTRAINT UC_Email UNIQUE (Email)	0 row(s) affected, 1 warning(s): 1831 Duplicate index 'UC_Email'
54	18:12:48	SHOW WARNINGS	1 row(s) returned
55	18:14:19	SHOW INDEX FROM Members	3 row(s) returned
56	18:16:32	ALTER TABLE Loans ADD CONSTRAINT CHK_ReturnDate CHECK (ReturnDate >= LoanDate)	7 row(s) affected Records: 7 Duplicates: 0 Warnings: 0
57	18:19:13	ALTER TABLE Members MODIFY MembershipStartDate DATE DEFAULT (CURRENT_DATE)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Create Index: This SQL statement creates an index named idx_title on the Title column of the Books table, which improves the efficiency of searching and retrieving rows based on the Title column.

```
CREATE INDEX idx_title ON Books (Title);
```

The screenshot shows the MySQL Workbench interface with a query editor titled "Query 1". The query is:

```
1
2 • SHOW INDEXES FROM Books;
```

The results are displayed in a "Result Grid" table:

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
▶	books	0	PRIMARY	1	BookID	A	10	NULL	NULL		BTREE
	books	0	ISBN	1	ISBN	A	10	NULL	NULL	YES	BTREE
	books	1	idx_title	1	Title	A	10	NULL	NULL		BTREE

Alter: This SQL command adds a new column named Publisher to the Books table with a data type of VARCHAR(100).

```
ALTER TABLE Books ADD COLUMN Publisher VARCHAR(100);
```

The screenshot shows the MySQL Workbench interface with a query editor titled "Query 1". The query is:

```
1 DESCRIBE Books;
2
```

The results are displayed in a "Result Grid" table:

	Field	Type	Null	Key	Default	Extra
▶	BookID	int	NO	PRI	NULL	auto_increment
	Title	varchar(100)	NO	MUL	NULL	
	Author	varchar(100)	YES		NULL	
	PublishedYear	int	YES		NULL	
	Genre	varchar(50)	YES		NULL	
	Price	decimal(10,2)	YES		0.00	
	ISBN	varchar(13)	YES	UNI	NULL	
	Publisher	varchar(100)	YES		NULL	

DROP: This command deletes (drops) the table named **BooksBackup** from the database.

The screenshot shows the MySQL Workbench interface. In the top window, titled "Query 1", there is a toolbar with various icons and a status bar indicating "Limit to 1000 rows". Below the toolbar, a single query is listed: "DROP TABLE BooksBackup;". In the bottom window, titled "Output", there is a table titled "Action Output" showing the results of the previous queries. The table has columns for "#", "Time", "Action", and "Message". The entries are:

#	Time	Action	Message
59	18:21:35	CREATE INDEX idx_title ON Books (Title)	Error Code: 1061. Duplicate key name 'idx_title'
60	18:21:42	SHOW INDEXES FROM Books	3 row(s) returned
61	18:22:27	ALTER TABLE Books ADD COLUMN Publisher VARCHAR(100)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
62	18:23:52	DESCRIBE Books	8 row(s) returned
63	18:27:33	DROP TABLE BooksBackup	0 row(s) affected

Auto Increment : The AUTO_INCREMENT keyword in the LoanID INT AUTO_INCREMENT PRIMARY KEY statement ensures that the LoanID column will automatically generate a unique sequential number for each new row inserted into the Loans table, simplifying the management of primary key values.

Example: CREATE TABLE Loans (LoanID INT AUTO_INCREMENT PRIMARY KEY, BookID INT, MemberID INT, LoanDate DATE DEFAULT CURRENT_DATE, ReturnDate DATE, FOREIGN KEY (BookID) REFERENCES Books(BookID), FOREIGN KEY (MemberID) REFERENCES Members(MemberID), CHECK (ReturnDate >= LoanDate));

Views: This view AvailableBooks lists titles, authors, and genres from the Books table excluding those currently on loan (where ReturnDate is NULL in the Loans table).

```
CREATE VIEW AvailableBooks AS SELECT Title, Author, Genre FROM Books WHERE BookID NOT IN (SELECT BookID FROM Loans WHERE ReturnDate IS NULL);
```

```
Query 1 ×
| File | New | Open | Save | Print | Help | Limit to 1000 rows | Favorites | Find | Replace | Close |
1 -- Query the AvailableBooks view
2 • SELECT * FROM AvailableBooks;
3

Result Grid | Filter Rows: _____ | Export: | Wrap Cell Content: □


|   | Title                      | Author          | Genre          |
|---|----------------------------|-----------------|----------------|
| ▶ | To Kill a Mockingbird      | Harper Lee      | Fiction        |
|   | 1984                       | George Orwell   | Dystopian      |
|   | Moby Dick                  | Herman Melville | Adventure      |
|   | Brave New World            | Aldous Huxley   | Dystopian      |
|   | Pride and Prejudice        | Jane Austen     | Romance        |
|   | The Picture of Dorian Gray | Oscar Wilde     | Gothic Fiction |


```

Null Values : This query selects all records from the Books table where the Publisher column has a NULL value.

Group By : This query counts the number of books in each genre from the Books table and displays the genre along with the count of books in that genre.

Query 1

```
1  SELECT Genre, COUNT(*) AS NumberOfBooks
2  FROM Books
3  GROUP BY Genre;
```

Result Grid | Filter Rows: Export: Wrap Cell Content:

Genre	NumberOfBooks
Fiction	1
Dystopian	2
Adventure	1
Classic	1
Fantasy	2
Literary Fiction	1
Romance	1
Gothic Fiction	1

Null Functions : This query selects the Title of books from the Books table, replacing NULL values in the Publisher column with 'Unknown' using the COALESCE function, and aliases the result column as Publisher.

Query 1

```
1 •  SELECT Title, COALESCE(Publisher, 'Unknown') AS Publisher FROM Books;
```

Result Grid | Filter Rows: Export: Wrap Cell Content:

Title	Publisher
To Kill a Mockingbird	Unknown
1984	Unknown
Moby Dick	Unknown
The Great Gatsby	Unknown
Harry Potter and the Philosopher's Stone	Unknown
The Catcher in the Rye	Unknown
The Hobbit	Unknown
Brave New World	Unknown
Pride and Prejudice	Unknown
The Picture of Dorian Gray	Unknown

Stored Procedure : This SQL script defines a stored procedure named `GetBooksByGenre` that accepts a genre name as input and retrieves all columns from the `Books` table where the genre matches the input parameter.

The screenshot shows the MySQL Workbench interface. The top pane is titled "Query 1" and contains the SQL code for creating a stored procedure:

```
1 DELIMITER //
2 • CREATE PROCEDURE GetBooksByGenre(IN genreName VARCHAR(50))
3 BEGIN
4     SELECT * FROM Books WHERE Genre = genreName;
5 END //
6 DELIMITER ;
```

The bottom pane is titled "Output" and shows the execution log with the following entries:

#	Time	Action	Message
58	18:21:05	CREATE INDEX idx_title ON Books (Title)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
59	18:21:35	CREATE INDEX idx_title ON Books (Title)	Error Code: 1061. Duplicate key name 'idx_title'
60	18:21:42	SHOW INDEXES FROM Books	3 row(s) returned
61	18:22:27	ALTER TABLE Books ADD COLUMN Publisher VARCHAR(100)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
62	18:23:52	DESCRIBE Books	8 row(s) returned
63	18:27:33	DROP TABLE BooksBackup	0 row(s) affected
64	18:30:02	CREATE VIEW AvailableBooks AS SELECT Title, Author, Genre FROM Books W...	0 row(s) affected
65	18:31:03	SELECT * FROM AvailableBooks LIMIT 0, 1000	6 row(s) returned
66	18:32:31	SELECT * FROM Books WHERE Publisher IS NULL LIMIT 0, 1000	10 row(s) returned
67	18:33:23	SELECT Genre, COUNT(*) AS NumberOfBooks FROM Books GROUP BY Genre L...	8 row(s) returned
68	18:34:45	SELECT Title, COALESCE(Publisher, 'Unknown') AS Publisher FROM Books LIMIT...	10 row(s) returned
69	18:36:09	CREATE PROCEDURE GetBooksByGenre(IN genreName VARCHAR(50)) BEGIN ...	0 row(s) affected

```
CALL GetBooksByGenre('Fiction');
```

The screenshot shows the MySQL Workbench interface with the results of the stored procedure execution. The top pane is titled "Query 1" and contains the same SQL code as before. The bottom pane is titled "Result Grid" and displays the following data:

BookID	Title	Author	PublishedYear	Genre	Price	ISBN	Publisher
1	To Kill a Mockingbird	Harper Lee	1960	Fiction	11.99	9780061120084	NULL

Primary Key & Foreign Key:

Primary Key: Uniquely identifies each row in a table. Ensures data integrity, prevents duplicates, and allows for efficient indexing.

Foreign Key: Establishes relationships between tables based on the primary key of another table. Ensures referential integrity, maintains data consistency, and supports relational database structure.

The screenshot shows a MySQL Workbench query editor titled "Query 1". The interface includes a toolbar with various icons for database management. The code area contains three CREATE TABLE statements:

```
1 • CREATE TABLE Students (
2     StudentID INT AUTO_INCREMENT,
3     FirstName VARCHAR(50) NOT NULL,
4     LastName VARCHAR(50) NOT NULL,
5     DateOfBirth DATE,
6     PRIMARY KEY (StudentID)
7 );
8
9 -- Courses table
10 • CREATE TABLE Courses (
11     CourseID INT AUTO_INCREMENT,
12     CourseName VARCHAR(100) NOT NULL,
13     CourseCode VARCHAR(20) UNIQUE,
14     PRIMARY KEY (CourseID)
15 );
16
17 -- Enrollments table
18 • CREATE TABLE Enrollments (
19     EnrollmentID INT AUTO_INCREMENT,
20     StudentID INT,
21     CourseID INT,
22     EnrollmentDate DATE,
```

Query 1

```

5     DateOfBirth DATE,
6     PRIMARY KEY (StudentID)
7 );
8
9 -- Courses table
10 • Ⓜ CREATE TABLE Courses (
11     CourseID INT AUTO_INCREMENT,
12     CourseName VARCHAR(100) NOT NULL,
13     CourseCode VARCHAR(20) UNIQUE,
14     PRIMARY KEY (CourseID)
15 );
16
17 -- Enrollments table
18 • Ⓜ CREATE TABLE Enrollments (
19     EnrollmentID INT AUTO_INCREMENT,
20     StudentID INT,
21     CourseID INT,
22     EnrollmentDate DATE,
23     PRIMARY KEY (EnrollmentID),
24     FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
25     FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
26 );

```

Having Function : The HAVING clause in SQL filters grouped data based on aggregate conditions specified after GROUP BY. It's used to select specific groups that meet certain criteria, akin to how WHERE filters individual rows.

Query 1

```

1 Ⓜ CREATE TABLE Students1 (
2     StudentID INT AUTO_INCREMENT,
3     FirstName VARCHAR(50) NOT NULL,
4     LastName VARCHAR(50) NOT NULL,
5     DateOfBirth DATE,
6     PRIMARY KEY (StudentID)
7 );
8
9 • INSERT INTO Students (FirstName, LastName, DateOfBirth)
10    VALUES
11        ('John', 'Doe', '1990-05-15'),
12        ('Jane', 'Smith', '1992-08-20'),
13        ('Michael', 'Johnson', '1991-02-10');
14

```

Query 1 ×

```

1 •  SELECT
2      AVG(YEAR(CURRENT_DATE) - YEAR(DateOfBirth)) AS AverageAge,
3      COUNT(*) AS StudentCount
4  FROM
5      Students
6  GROUP BY
7      StudentID
8  HAVING
9      AverageAge > 30;
10

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	AverageAge	StudentCount
▶	34.0000	1
	32.0000	1
	33.0000	1

Callable Statement

Theory:

- Definition: A CallableStatement in SQL is a feature used in programming languages like Java to execute stored procedures or functions that are stored in a database.
- Usage: It allows applications to call database procedures with input parameters and retrieve output parameters or results.
- Preparedness: Unlike regular SQL statements, CallableStatement is precompiled, which means it can be executed multiple times with different parameters efficiently.
- Benefits: It enhances security by preventing SQL injection and improves performance by reducing compilation overhead.
- Examples: In Java, you prepare a CallableStatement to execute a stored procedure with parameters and retrieve results as needed.

Prepared Statement

Theory:

- Definition: A PreparedStatement in SQL is a feature used to execute SQL queries multiple times with varying parameters.
- Functionality: It precompiles SQL queries and allows parameterized execution, enhancing performance and security.
- Parameterization: SQL parameters are placeholders in the query that are filled dynamically during execution, preventing SQL injection attacks.
- Efficiency: Prepared statements are compiled once by the database server and cached, enabling quick execution with different parameter values.
- Example: In a web application, a PreparedStatement is used to insert user data into a database securely, ensuring data integrity and protection.

Concept of Normalization in SQL

Theory:

- Definition: Normalization in SQL is the process of organizing data in a database to reduce redundancy and dependency by dividing large tables into smaller tables and defining relationships between them.
- Objective: To eliminate data anomalies (like insertion, update, and deletion anomalies) and improve data integrity.
- Levels of Normalization:
 - First Normal Form (1NF): Ensures atomicity of data (no repeating groups).
 - Second Normal Form (2NF): Ensures all non-key attributes are fully functional dependent on the primary key.
 - Third Normal Form (3NF): Ensures no transitive dependency among non-key attributes.
 - Higher Normal Forms (BCNF, 4NF, 5NF): Address more complex data dependencies and ensure further reduction in redundancy.
- Benefits:
 - Reduces storage space by avoiding duplicate data.
 - Improves data consistency and accuracy.
 - Simplifies maintenance and enhances query performance.
- Example:
 - In a library database, separate tables for Books, Authors, and Genres are created. Each table contains specific attributes related to books, authors, and genres respectively, linked through primary and foreign keys.