# Table of Contents

## Module 4: Advanced DDL and DML

- String Patterns, Ranges, and Sets
- Sorting Result Sets
- Grouping Result Sets

## Module 5: Working with multiple tables

- Join Overview
- Inner Join
- Left Outer Join
- Right Outer Join
- Full Join

# Module 1: SQL and Relational Databases 101

# Topic: Introduction to SQL and Relational Databases

## What is SQL?

- **SQL, which stands for "Structured Query Language," is a language used for relational databases to query or retrieve data from a database.**
- **It is sometimes referred to as "sequel."**
- **It was originally developed by IBM for querying, altering, and defining relational databases.**

## What SQL Can Do?

**SQL can execute queries, retrieve data, insert records, update records, delete records, create new databases, create new tables, create stored procedures, create views, and set permissions.**

## SQL Commands

- **Create a table**
- **Insert data to populate the table**
- **Select data from the table**
- **Update data in the table**
- **Delete data from the table**

## Relational Databases

### *Data Organization*

- **Data can be organized in tabular form, similar to spreadsheets with columns and rows.**
- **This structure is known as a relational database.**
- **Tables contain properties or attributes about items, such as Last Name, First Name, email, address, and City.**

### *Relationships in Relational Databases*

- **In a relational database, you can establish relationships between tables, allowing for more complex data modeling.**

### *Examples of RDBMS*

- **MySQL**
- **Oracle Database**
- **DB2 Express-C**
- **SQL Basics**

## Database Management System (DBMS)

- **A set of software tools for managing data in a database, also known as a database server or database system.**
- **For relational databases, it's called a Relational Database Management System (RDBMS).**

## Variety of Database Systems

**Many different database systems (DBMS) exist, including:**

**Microsoft SQL Server**
**Oracle**
**MySQL**
**Microsoft Access**
**IBM DB2**
**Sybase**
**And many others**

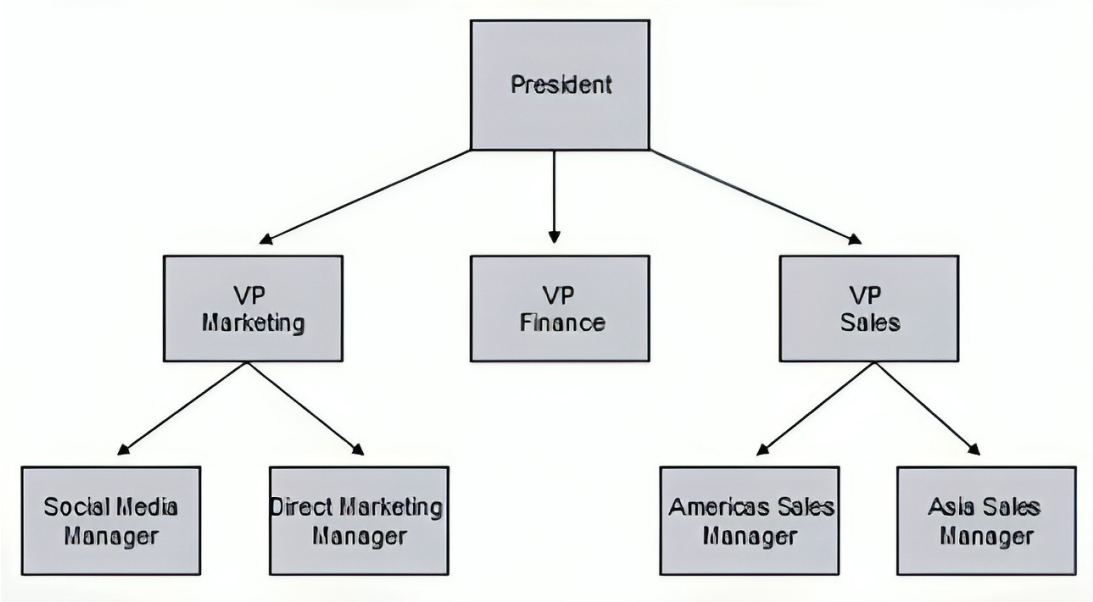
# Topic: Information Models vs. Data Models

- ## Information Model:

    - **An abstract, formal representation of entities, including their properties, relationships, and operations.**
    - **Typically at the conceptual level.**
    - **Defines relationships between objects.**
    - **Represents real-world entities.**

### *Hierarchical Information Model*

- **Organizes data using a tree structure.**
- **Root node is the parent, followed by child nodes.**
- **A child node has only one parent, but a parent can have multiple child nodes.**
- **Developed by IBM in 1968 for the Apollo space program**

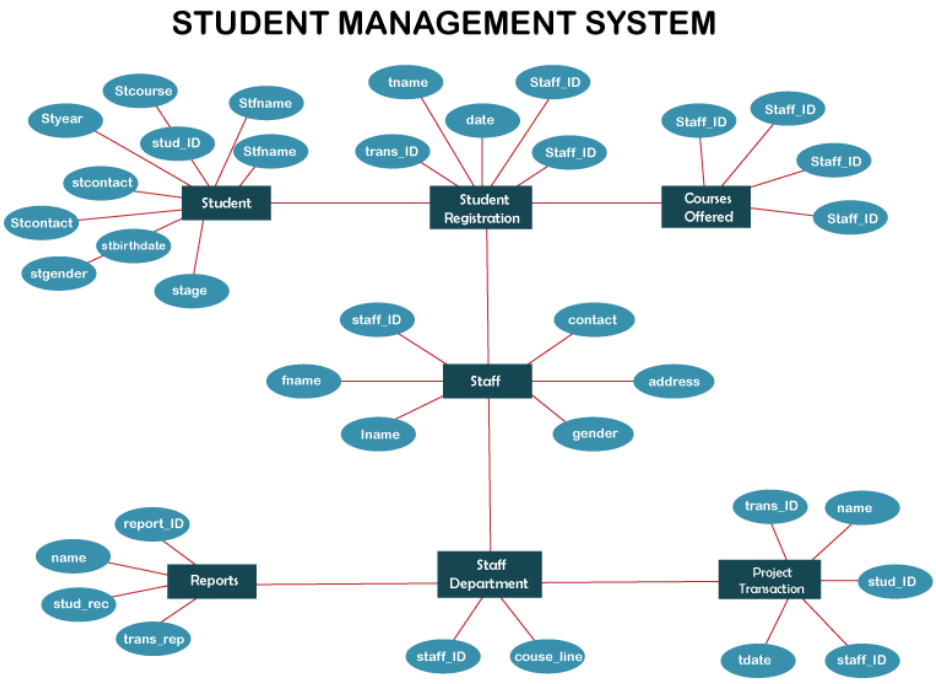



- **Data Model:**
    - **More concrete and specific than information models.**
    - **Serves as the blueprint for a database system.**
    - **Includes details about data storage and structure.**
    - **Enables data independence and storage flexibility.**

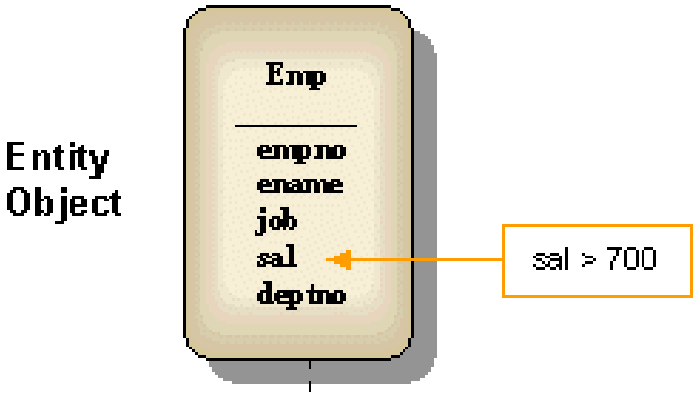
.

## *Entity-Relationship Data Model (ER Model)*

- **Represents entities and their relationships.**
- **Used as a tool to design relational databases.**
- **Entities exist independently of each other.**
- **Entities have attributes that characterize them.**
- **Attributes are connected to exactly one entity.**




- **Entities are objects that exist independently and are represented as rectangles in ER diagrams.**
- **Attributes are characteristics or properties of entities and are represented as ovals in ER diagrams.**




**Emp Table**

| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|---|---|---|---|---|---|---|---|
| 7369 | SMITH | CLERK | 7902 | 17-DEC-80 | 800 | | 20 |
| 7499 | ALLEN | SALESMAN | 7698 | 20-FEB-81 | 1600 | 300 | 30 |
| 7521 | WARD | SALESMAN | 7698 | 22-FEB-81 | 1250 | 500 | 30 |
| 7566 | JONES | MANAGER | 7839 | 02-APR-81 | 2975 | | 20 |
| 7654 | MARTIN | SALESMAN | 7698 | 28-SEP-81 | 1250 | 1400 | 30 |

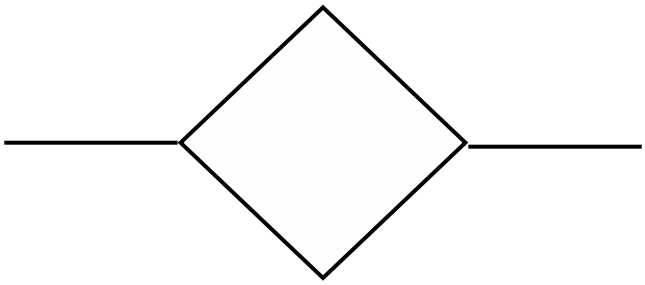
**In the ER Model, entities become tables in the database, and attributes become columns in tables.**

# Topic: Types of Relationships

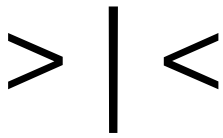
## Building Blocks of a Relationship:

- **Relationships in databases consist of three main components: entities, relationship sets, and notations.**
- **Entities are represented by rectangles.**

- **Relationship sets are represented by diamonds and are connected by lines between associated entities.**



- **Crow foot notations**



**There are three main types of relationships in a database model: one-to-one, one-to-many, and many-to-many.**

- **In a one-to-one relationship, each record in one table is associated with one and only one record in another table**


**For example, a person has only one passport and a passport is given to one person.**

- **A one-to-many relationship is represented using crow's foot notation with a less-than symbol (<).**


**For example – a customer can place many orders but a order cannot be placed by many customers.**

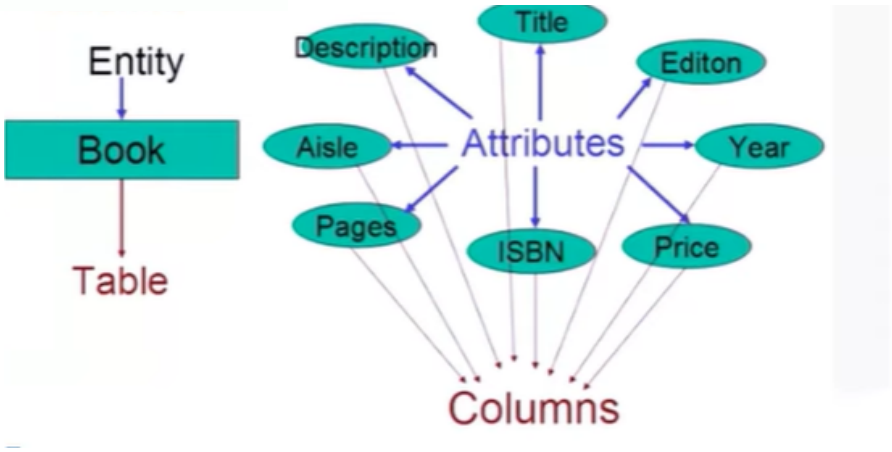
- **A many-to-many relationship is represented using both greater-than (>) and less-than (<) symbols on either side of the relationship set.**


**For example – many students can study in a single college but a student cannot study in many colleges at the same time.**

## Topic: Mapping Entities to Tables

- **An ERD is a graphical representation that captures the structure of the data model.**

**In this example, we have the entity "Book," which has several attributes such as title, edition, year, and price, among others.**




- **The entity "Book" becomes a table with the same name, "Book."**
- **All the attributes from the entity "Book" are translated into columns in the "Book" table.**

Table: Book

| Title | Edition | Year | Price | ISBN | Pages | Aisle | Description |
|---|---|---|---|---|---|---|---|
| Database Fundamentals | 1 | 2010 | 24.99 | 978-0-98662 83-1-1 | 300 | DB-A02 | Teaches you the fundamentals of databases |
| Getting started with DB2 Express-C | 1 | 2010 | 24.99 | 978-0-9866 283-5-1 | 280 | DB-A01 | Teaches you the essentials of DB2 using DB2 Express-C, the free version of DB2 |

- **To complete the table, we add data values to each of the columns.**
- **These data values represent the actual information stored in the database.**

# Topic: Relational Model Concepts

- **The relational model was first proposed in 1970 and is based on mathematical principles.**
- **It forms the foundation for relational databases used in modern database management systems.**

## Key Relational Terms:

- **Relation: In the relational model, a relation is a mathematical concept based on the idea of sets. It is also the mathematical term for a table in a database.**

- **Set: A set is an unordered collection of distinct elements, where each element is of the same type, and there are no duplicates.**

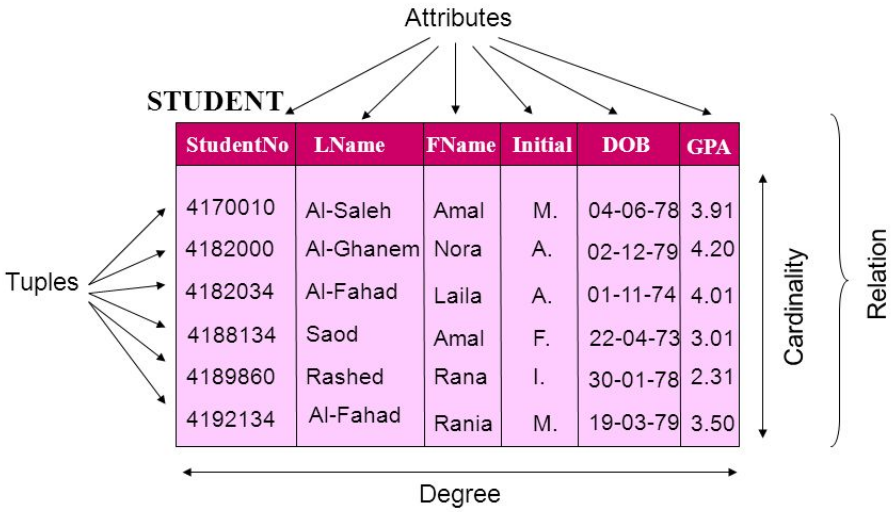
## Components of a Relation:

- **A relation consists of two main parts:**
- **Relational Schema: Specifies the name of the relation (table) and the attributes (columns) it contains. It also defines the data types for each attribute.**
- **Relational Instance: This is the actual table with rows and columns.**

**The columns represent the attributes, and the rows are called tuples.**

# Degree and Cardinality:

- **Degree: Refers to the number of attributes (columns) in a relation. It quantifies the width of the table.**
- **Cardinality: Refers to the number of tuples (rows) in a relation. It quantifies the height or size of the t**




**In this example, cardinality is 6 and  degree is also 6.**

# Module 2: Relational Model Constraints and Data Objects

## Topic: Relational Model Constraints Introduction

### Referencing Between Entities:

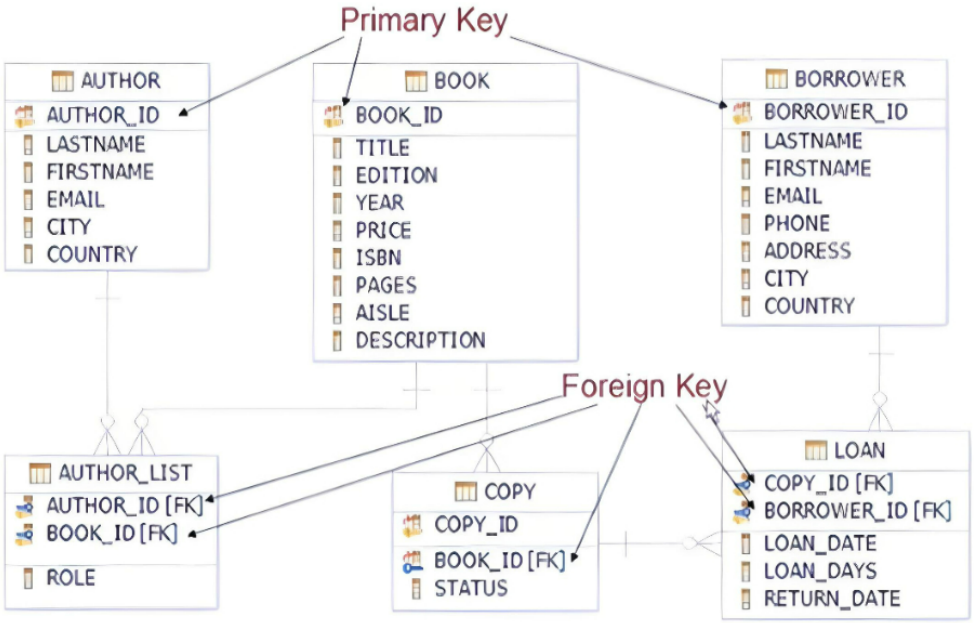
- **Within a business or database, data must adhere to certain restrictions or rules.**
- **In a relational data model, referencing is the practice of establishing connections or references between entities to ensure data integrity.**

### Primary Key and Foreign Key:

- **A Primary Key is a set of columns in a relational table that uniquely identifies each row in that table.**
- **A Foreign Key is a set of columns in one table that refers to the primary key of another table.**
- **Foreign keys establish relationships between entities.**

### Parent Table and Dependent Table:

- **A Parent Table is a table containing a primary key that is related to at least one foreign key in another table.**
- **A Dependent Table (or child table) is a table that contains one or more foreign keys referring to primary keys in other tables.**




- **Attributes with a special icon, such as Author_ID, Book_ID, and Borrower_ID, represent primary keys**
- **Entities in the lower half of the screen have attributes with "FK" (Foreign Key) next to them, such as "Copy" with the attribute "Book_ID" being a foreign key.**
- **"Author" and "Book" entities are identified as parent tables.**
- **"Author_List" entity is a dependent table with foreign keys referring to "Author" and "Book."**

# Topic: Relational Model Constraints Advanced

## Entity Integrity Constraint:

As we already know that a primary key should we unique and shouldn't contain null or duplicate values. It should represent a set or records or tuples to be identified uniquely. These rules that are related to primary key comes t

AUTHOR

| Author_ID [PK] | Lastname | Firstname | Email | City | Country |
|---|---|---|---|---|---|
| NULL | Chong | Raul | rfc@ibm.com | Toronto | CA |
| A2 | Ahuja | Rav | ra@ibm.com | Toronto | CA |
| A3 | Hakes | Ian | ih@ibm.com | Toronto | CA |
| NULL | Sharma | Neeraj | ns@ibm.com | Chennai | IN |
| A5 | Perniu | Liviu | lp@univ.com | Transilvania | RO |

In the snapshot of the table above, Author_ID is a primary of the author table. As you can notice that there are a couple of NULL values in the first attribute of the author entity. As we know that null values violate the rules of the Entity Integrity Constraint, so it must be fixed. One way of easily implementing the constraint is through indexing.

## Referential Integrity Constraint:

Referential integrity constraint defines relationships between tables and ensures that these relationships remain valid. The validity of the data is enforced using a combination of primary keys and foreign keys.

## Semantic Integrity Constraint:

**It relates to correctness of the data with the type of data needed in the corresponding tuple or a column. For example, In the City column, invalid data such as %FYG#@^ will be considered as dirty and invalid because there is no city with such name.**

## Domain Constraint:

**A domain constraint is related to specifying the permissible values for each attribute in an entity. For example, in the relation author, the attribute country must contain a two-letter country code such as CA for Canada or IN for India. If a number value of 34 is entered for the country attribute instead of a two-letter country code, the value 34 does not have any meaning. As we saw previously, the entity integrity constraint states that no attribute participating in the primary key is allowed to accept null values. The null constraint specifies that attribute values cannot be null.**

## NULL Constraint:

**As the title of the constraint suggest, this rule is related to null values not being permissible in a tuple.**

## Check Constraint:

**This constraint is related to enforcing the correctness of data in a column. For example: In the book table below, Check constraint is enforced in the Year column to evaluate that the year when the book was published.**

## More on Primary Keys

**If a relation schema has more than one key, then each key is called a candidate key.One of the candidate keys is designated as the primary key, and the others are called secondary keys.**

**In a practical relational database, each relation schema must have a primary key.**

***Rules for primary keys*:**

- **The value of the Primary Key must be unique for each instance of the entity.**
- **There can be no missing values( ie. Not Null) for Primary Keys. If the Primary Key is composed of multiple attributes, each of those attributes must have a value for each instance.**
- **The Primary Key is immutable.i.e., once created the value of the Primary Key cannot be changed.**
- **If the Primary Key consists of multiple attributes, none of these values can be updated.**

# Module 3: Data Definition Language (DDL) and Data Manipulation Language (DML)

## Topic: Types of SQL Statements (DDL vs. DML)

**SQL (Structured Query Language) is a powerful tool for interacting with relational databases. SQL statements are categorized into two main types: Data Definition Language (DDL) and Data Manipulation Language (DML).**

### Data Definition Language (DDL) Statements

***Definition*: DDL statements are used to define, modify, or delete database objects, primarily tables, and their structural elements.**

**Common DDL Statement Types**

1. **CREATE**
   - ***Purpose*: Creating Tables and Defining Columns**
   - ***Description*: Used to create new tables in the database and specify the attributes (columns) they will contain.**
2. **ALTER**
   - ***Purpose*: Altering Tables**
   - ***Description*: Used for making changes to existing tables, such as adding or dropping columns and modifying their data types.**
3. **TRUNCATE**
   - ***Purpose*: Deleting Data (Not the Table)**
   - ***Description*: Deletes all data within a table while retaining the table's structure.**

4. **DROP**

***Purpose*: Deleting Tables**

***Description*: Used to completely remove tables from the database, including all associated data.**

## Data Manipulation Language (DML) Statements

***Definition*: DML statements are designed for reading and modifying data within database tables. They correspond to CRUD operations: Create, Read, Update, and Delete.**

**Common DML Statement Types**

1. **INSERT**

***Purpose*: Adding Rows of Data**

***Description*: Used to insert one or multiple rows of data into a table.**

2. **SELECT**

***Purpose*: Retrieving Data**

***Description*: Reads and selects rows from one or more tables, allowing for data retrieval and filtering.**

3. **UPDATE**

***Purpose*: Modifying Data**

***Description*: Edits existing rows in a table, making changes to specific data values.**

4. **DELETE**

***Purpose*: Removing Data Rows**

***Description*: Removes one or more rows of data from a table, effectively deleting them.**

# TOPIC: CREATE TABLE Statement

***Syntax*:**

```
CREATE TABLE table_name
(
column_name_1 datatype optional_parameters,
column_name_ 2 datatype,
...
column_name_n datatype
)
```

# TOPIC: INSERT Statement

***Syntax*:**

```
INSERT INTO [TableName]
<{{ColumnName)...)>
VALUES ([Value]...)
```

***Example*:**

```
INSERT INTO AUTHOR
 (AUTHOR_ID, LASTNAME, FIRSTNAME, EMAIL, CITY,
COUNTRY)
VALUES ("A1", "Chong", "Raul, rfc@ibm.com", "Toronto',
'CA")
```

# TOPIC: SELECT Statement

```
Select statement:   Query
Result from the query:  Result set/table
```

**Syntax:**

**Select * from <tablename>**
***//retrieving a subset of the columns//***
**Select <column1>, <column2>, <column3> from <tablename>**

## The WHERE Clause: Comparison and Logical Operators

**Syntax:**

**Select <column1>, <column2> from <tablename> *where* <columnname>= ‘ ’**

***Example*:**

**Select book_id, title from book *where* book_id = ‘B1’**

# SQL Comparison Operators

| Comparison Operator | Definition |
|---|---|
| = | Equal |
| !=, <> | Not equal |
| >, >= | Greater than, greater than or equal to |
| <, <= | Less than, less than or equal to |
| BETWEEN ... AND ... | Inclusive of two values |
| LIKE | Pattern matching with wildcard characters % and _ |
| IN (...) | List of values |
| IS NULL | Test for null values |

**Example:**

```
SELECT description, cost, prerequisite   FROM course
 WHERE cost = 1195
   AND prerequisite = 20
    OR prerequisite = 25
```

# TOPIC: UPDATE & DELETE STATEMENT

## UPDATE Statement

***Syntax*:**

**UPDATE [TableName]**
**SET [[ColumnName]=[Value]]**
**<WHERE [Condition]>**

***Example*:**

| Author_Id | LastName | FirstName | Email | City | Country |
|---|---|---|---|---|---|
| A1 | CHONG | RAUL | rfc@ibm.com | Toronto | CA |
| A2 | AHUJA | RAV | ra@ibm.com | Toronto | CA |
| A3 | HAKES | IAN | ih@ibm.com | Toronto | CA |

```
UPDATE AUTHOR
SET LASTNAME='KATTA'
    FIRSTNAME='LAKSHMI'
    WHERE AUTHOR_ID='A2'
```

| Author_Id | LastName | FirstName | Email | City | Country |
|---|---|---|---|---|---|
| A1 | CHONG | RAUL | rfc@ibm.com | Toronto | CA |
| A2 | KATTA | LAKSHMI | ra@ibm.com | Toronto | CA |
| A3 | HAKES | IAN | ih@ibm.com | Toronto | CA |

# DELETE Statement

## *Syntax*:

**DELETE from [TableName]**
**<WHERE [Condition]>**

## *Example*:

| Author_Id | LastName | FirstName | Email | City | Country |
|---|---|---|---|---|---|
| A1 | Chong | Raul | rfc@ibm.com | Toronto | CA |
| A2 | Ahuja | Rav | ra@ibm.com | Toronto | CA |
| A3 | Hakes | Ian | ih@ibm.com | Toronto | CA |
| A4 | Sharma | Neeraj | ns@ibm.com | Chennai | IN |
| A5 | Perniu | Liviu | lp@ibm.com | Transylvania | RO |

```
DELETE FROM AUTHOR
   WHERE AUTHOR_ID IN ('A2', 'A3')
```

| Author_Id | LastName | FirstName | Email | City | Country |
|---|---|---|---|---|---|
| A1 | Chong | Raul | rfc@ibm.com | Toronto | CA |
| A4 | Sharma | Neeraj | ns@ibm.com | Chennai | IN |
| A5 | Perniu | Liviu | lp@ibm.com | Transylvania | RO |

# Module 4: Advanced DML & DDL

# Topic: String Patterns, Ranges, and Sets

## Using String Patterns

**Use string patterns to search for matching rows.**

***LIKE Predicate*:**
**The LIKE predicate is employed in the WHERE clause to search for patterns in a column. The percent sign (%) is used as a *wildcard character* to represent missing letters. It can be placed before the pattern, after it, or both.**

**Example:**

```
db2 => select firstname from author
where firstname like 'R%'

FIRSTNAME
---------------
RAUL
RAV

  2 record(s) selected.
```

## Using Ranges

**Instead of using separate greater-than-or-equal-to and less-than-or-equal-to operators, you can simplify the query.**

**BETWEEN AND Operator: This operator compares two values and specifies an inclusive range of values.**

**Example:**

```
db2 => select title, pages from book
where pages between 290 and 300

TITLE                                PAGES
-------------------------------- --------
Database Fundamentals                 300
Getting started with DB2 App Dev      298

  2 record(s) selected.
```

## Using Sets of Values

**When dealing with multiple values, listing them individually in the WHERE clause can be cumbersome.**

**IN Operator: The IN operator allows you to specify a set of values in the WHERE clause, making it more concise.**

**Example:**

```
db2 => select firstname, lastname,
  country from author where country
  IN ('AU','BR')

FIRSTNAME       LASTNAME        COUNTRY
--------------- --------------- -------
Xiqiang         Ji              AU
Juliano         Martins         BR

  2 record(s) selected.
```

# Topic: Sorting Result Sets

**To make the result set more convenient, you can use the ORDER BY clause.**

## ORDER BY Clause:

**The ORDER BY clause is used to sort the result set by a specified column. By default, the result set is sorted in ascending order.**

**Example:**

```
db2 => select title from book
       order by title

TITLE
-----------------------------------
Database Fundamentals
Getting started with DB2 App Dev
Getting started with DB2 Express-C
Getting started with WAS CE

  4 record(s) selected.
```

**Ascending order**

```
db2 => select title from book
       order by title desc

TITLE
-----------------------------------
Getting started with WAS CE
Getting started with DB2 Express-C
Getting started with DB2 App Dev
Database Fundamentals

  4 record(s) selected.
```

**Descending order**

## Indicating the Sort Column

**You can also indicate the column for sorting using either the column name or its sequence number in the SELECT statement.**

**Column Name: Specify the column name for sorting.**

## Column Sequence Number:

**Indicate the column sequence number for sorting. This is useful when you want to sort by a column without specifying its name.**

**Example:**

```
db2 => select title, pages from book
            order by 2

TITLE                                PAGES
-------------------------------      -------
Getting started with WAS CE          278
Getting started with DB2 Express-C   280
Getting started with DB2 App Dev     298
Database Fundamentals                300

  4 record(s) selected.
```

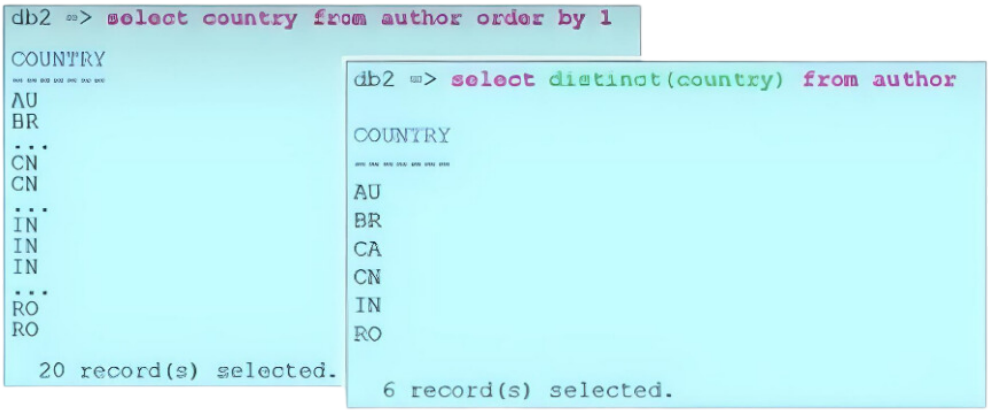
**Ascending order by Column 2 (number of pages)**

# Topic: Grouping Result Sets

## Eliminating Duplicates from Result Sets

**In some cases, a SELECT statement result set may contain duplicate values.**
**To eliminate duplicates and get a list of unique values, we can use the DISTINCT keyword.**

**Example::**



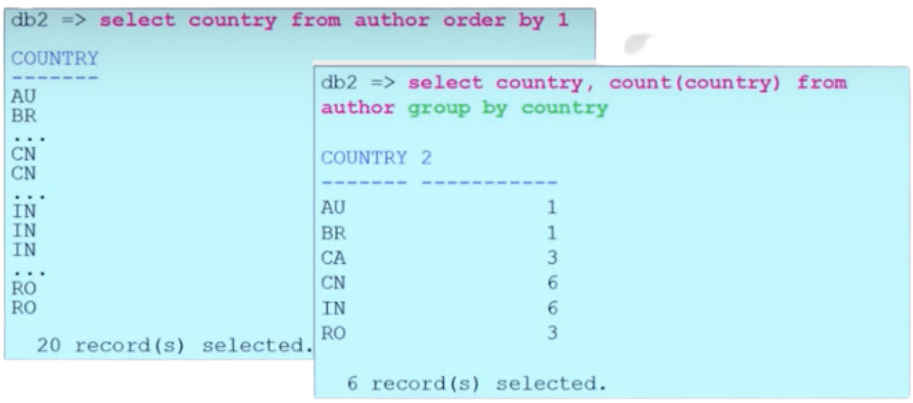

# The GROUP BY clause

**Groups the result into subsets with matching values for one or more columns by using the COUNT function.**

**Example:**

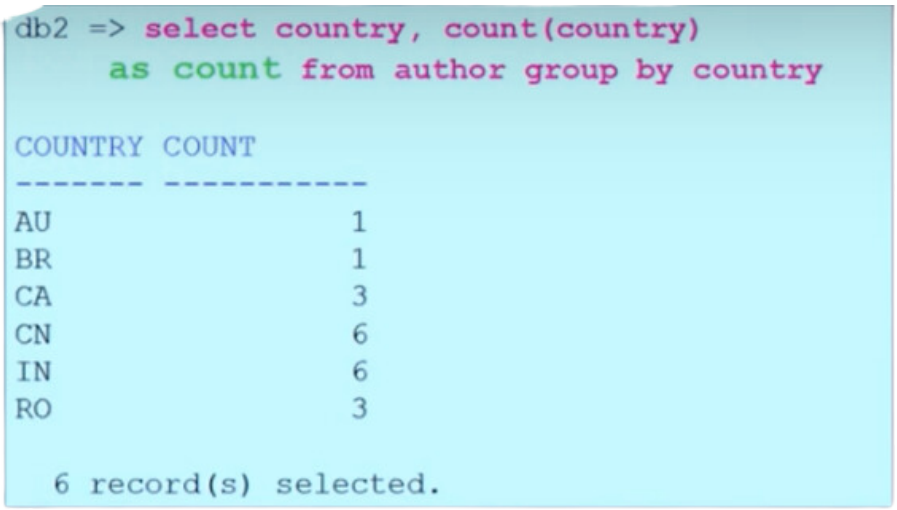
- **Using COUNT with GROUP BY:**

**SELECT Country, COUNT(*) FROM Author GROUP BY Country counts authors from each country and displays the result set with two columns: "Country" and "COUNT(*)" (since the column name isn't directly available in the table).**



- **Assigning Column Aliases**

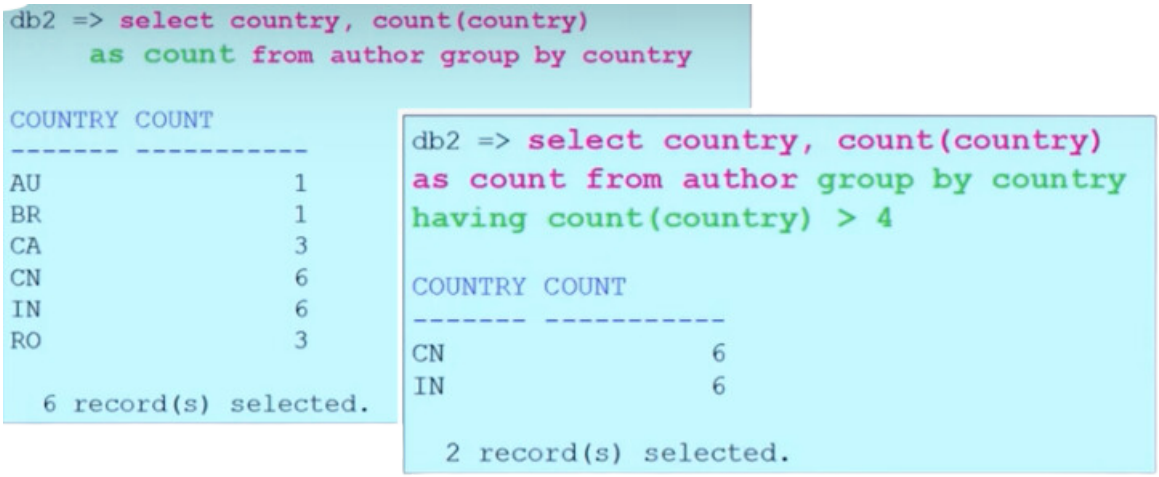
**To make the result set more readable and meaningful, you can assign column aliases using the AS keyword.**



```
db2 => select country, count(country)
     as count from author group by country

COUNTRY COUNT
------- -----------
AU                1
BR                1
CA                3
CN                6
IN                6
RO                3

  6 record(s) selected.
```

- **Further Restricting Result Sets**

**Using HAVING with GROUP BY: To set conditions for a GROUP BY clause, we use the HAVING keyword. The HAVING clause works in conjunction with the GROUP BY clause.**



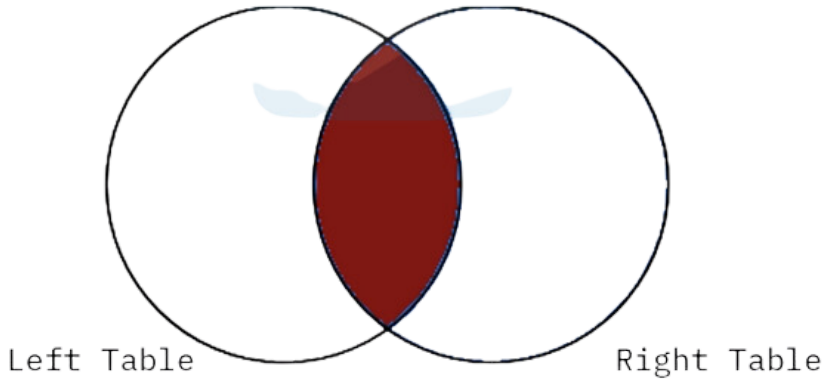
```
db2 => select country, count(country)
     as count from author group by country

COUNTRY COUNT
------- -----------
AU                1
BR                1
CA                3
CN                6
IN                6
RO                3

  6 record(s) selected.
```

```
db2 => select country, count(country)
as count from author group by country
having count(country) > 4

COUNTRY COUNT
------- -----------
CN                6
IN                6

  2 record(s) selected.
```

# Module 5: Working with Multiple Tables

## Topic: Inner Join

- **An inner join matches rows from two or more tables and displays only the result set that matches the criteria specified in the query.**
- **It returns only the rows that have matching values in both tables.**

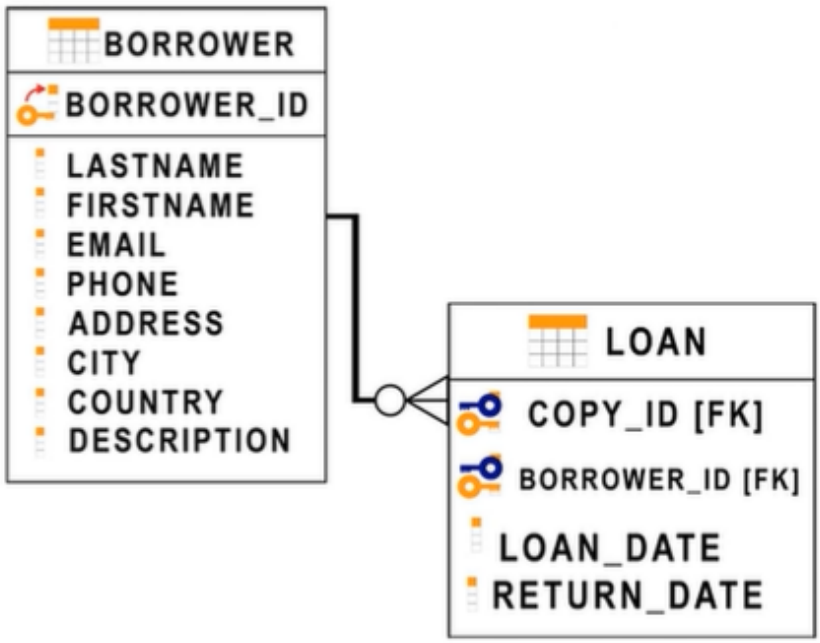



**Imagine you want to find the names of people who have borrowed a book, with this information split between two tables: "Borrower" and "Loan."**

**You establish the relationship between these tables using a common column, such as "borrower_id," which exists as a primary key in the "Borrower" table and a foreign key in the "Loan" table.**

**You can use the inner join operator to match occurrences of "borrower_id" in both tables.**

```
SELECT B. BORROWER_ID, B. LASTNAME, B.COUNTRY,
       L. BORROWER_ID, L. LOAN DATE
FROM BORROWER B INNER JOIN LOAN L
     ON B. BORROWER_ID = L.BORROWER_ID
```



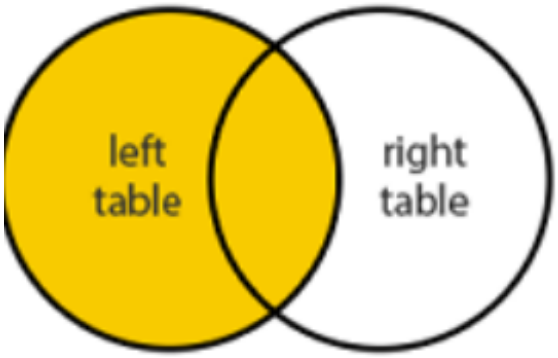

## Topic: Outer Join

- **Outer joins, like inner joins, combine rows from multiple tables based on matching values in specified columns.**

- **However, unlike inner joins, outer joins also include rows that do not have matches in the joined tables.**

**Left Outer Join:**

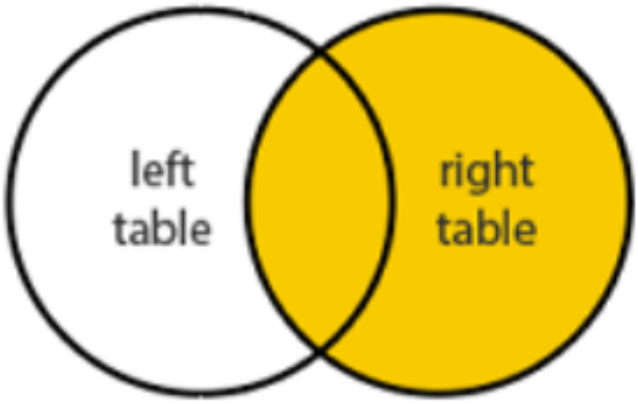
**In a left outer join, all rows from the first table (left side of the join) are included, along with matching rows from the second table (right side of the join).**
**Rows from the first table that have no match in the second table are also included.**




```
SELECT B. BORROWER_ID, B. LASTNAME,
B.COUNTRY,

        L. BORROWER_ID, L. LOAN DATE

FROM BORROWER B LEFT JOIN LOAN L

      ON B. BORROWER_ID = L.BORROWER_ID
```

**Right Outer Join:**

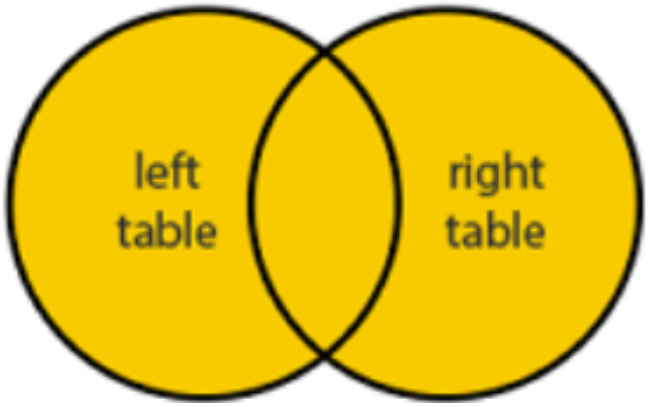
**A right outer join is similar to a left outer join but in reverse. It includes all rows from the second table and matching rows from the first table.**
**Rows from the second table that have no match in the first table are also included.**




```
SELECT B. BORROWER_ID, B. LASTNAME,
B.COUNTRY,

        L. BORROWER_ID, L. LOAN DATE

FROM BORROWER B RIGHT JOIN LOAN L

      ON B. BORROWER_ID = L.BORROWER_ID
```

**Full Outer Join:**

**A full outer join returns all rows from both the left and right tables.**
**It includes matching rows as well as rows from both tables that do not have matches.**



```
SELECT B. BORROWER_ID, B. LASTNAME,
B.COUNTRY,

        L. BORROWER_ID, L. LOAN DATE

FROM BORROWER B FULL JOIN LOAN L

      ON B. BORROWER_ID = L.BORROWER_ID
```

**Q. How does a CROSS JOIN (also known as Cartesian Join) statement syntax look?**

```
SELECT column_name(s)
FROM table1
CROSS JOIN table2;
```

**Q. How does an INNER JOIN statement syntax look?**

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
WHERE condition;
```

**Q. How does a LEFT OUTER JOIN statement syntax look?**

```
SELECT column_name(s)
FROM table1
LEFT OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

**Q. How does a RIGHT OUTER JOIN statement syntax look?**

```
SELECT column_name(s)
FROM table1
RIGHT OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

**Q. How does a FULL OUTER JOIN statement syntax look?**

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

**Q. How does a SELF JOIN statement syntax look?**

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```