Databases

SQL Basics

8 February 2024

克明峻德，格物致知

Quick Recap: Relational Databases

Overview of Relational Model:

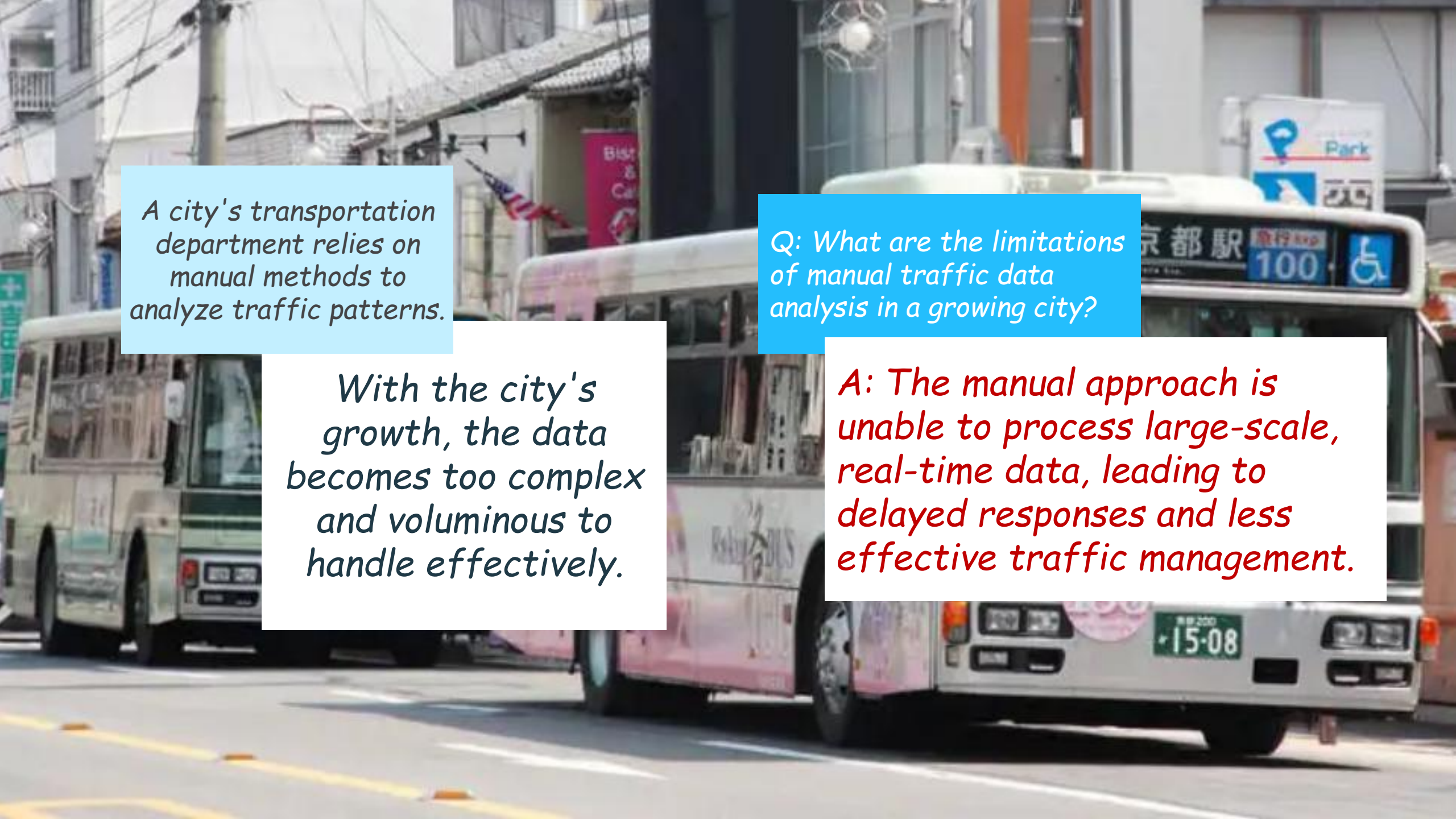
- Data organized in structured tables consisting of rows (tuples) and columns (attributes).
- Emphasis on data uniqueness (no duplicate rows) and maintaining data integrity through typed and static attributes.

SQL Query Example:

```sql
SELECT Name, Email
FROM Members
WHERE JoinYear = 2020;
```

- **Purpose:** Retrieve names and emails of 2020 members
- **Output Table:**

```
| Name           | Email                  |
|----------------|------------------------|
| Bethany Cane   | bethany.c@example.com  |
| Evan Lee       | evan.lee@example.com   |
```



A city's transportation department relies on manual methods to analyze traffic patterns.

With the city's growth, the data becomes too complex and voluminous to handle effectively.

Q: What are the limitations of manual traffic data analysis in a growing city?

A: The manual approach is unable to process large-scale, real-time data, leading to delayed responses and less effective traffic management.

Structured Query Language (SQL)

Nature of SQL:

- Declarative, focusing on specifying what the result should be, unlike procedural languages that describe how to perform tasks.

Declarative vs Procedural:

- SQL Retrieval Query:

```sql
SELECT Name, Email
FROM Members
WHERE JoinYear = 2020;
```

- Python Query Example:

```python
for member in Members if member['JoinYear'] == 2020 :
   print(member['Name'], member['Email'])
```

Key Functions of SQL:

- **Data Definition:** `CREATE`, `ALTER`, `DROP` for structuring database schemas.
- **Data Manipulation:** `INSERT`, `UPDATE`, `DELETE`, `SELECT` for handling data within tables.
- **Data Control:** `GRANT`, `REVOKE` for managing data access and permissions.

Basic SQL Query

Understanding `SELECT` Statements:

- `SELECT` statements are fundamental in SQL for retrieving specific data from a database.
- They allow users to specify exactly which columns to display and set conditions for selecting rows.

Example: Retrieve All Records

- SQL Query:

```sql
SELECT *
FROM Books;
```

- **Purpose:** Retrieves all columns for every row in the 'Books' table.
- Output Table:

```
| BookID | Title                   | Author       | Genre       | PublishedYear |
|--------|-------------------------|--------------|-------------|---------------|
| 101    | Journey Through SQL     | A. Coder     | Technology  | 2015          |
| 102    | The History of Databases| D. Base      | Education   | 2018          |
| ...    | ...                     | ...          | ...         | ...           |
```

Retrieving Specific Data with SQL

Selecting Specific Columns and Applying Conditions:

- SQL's `SELECT` command can be tailored to extract only certain columns from a database table. Adding a `WHERE` clause allows for condition-based filtering of data.

Example: Filter by Genre

- **Source Table:** 'Books'

BookID	Title	Author	Genre	PublishedYear
101	Journey Through SQL	A. Coder	Technology	2015
102	The History of Databases	D. Base	Education	2018
103	Adventures in Coding	P. Programmer	Fiction	2020
...	...	...	...	...

- **SQL Query:**

```sql
SELECT Title, Author
FROM Books
WHERE Genre = 'Technology';
```

- **Purpose:** To display the titles and authors of books that fall under the 'Technology' genre.

Retrieving Specific Data with SQL

Selecting Specific Columns and Applying Conditions:

- SQL's `SELECT` command can be tailored to extract only certain columns from a database table. Adding a `WHERE` clause allows for condition-based filtering of data.

Example: Filter by Genre

- **Source Table:** 'Books'

BookID	Title	Author	Genre	PublishedYear
101	Journey Through SQL	A. Coder	Technology	2015
102	The History of Databases	D. Base	Education	2018
103	Adventures in Coding	P. Programmer	Fiction	2020
...	...	...	...	...

- **SQL Query:**

```sql
SELECT Title, Author
FROM Books
WHERE Genre = 'Technology';
```

- **Purpose:** To display the titles and authors of books that fall under the 'Technology' genre.

Retrieving Specific Data with SQL

Selecting Specific Columns and Applying Conditions:

- SQL's `SELECT` command can be tailored to extract only certain columns from a database table. Adding a `WHERE` clause allows for condition-based filtering of data.

Example: Filter by Genre

- **Source Table:** 'Books'

BookID	Title	Author	Genre	PublishedYear
101	Journey Through SQL	A. Coder	Technology	2015
102	The History of Databases	D. Base	Education	2018
103	Adventures in Coding	P. Programmer	Fiction	2020
...	...	...	...	...

- **SQL Query:**

```sql
SELECT Title, Author
FROM Books
WHERE Genre = 'Technology';
```

- **Output Table:**

Title	Author
Journey Through SQL	A. Coder

- **Purpose:** To display the titles and authors of books that fall under the 'Technology' genre.

Database Internals

SQL Query Processing Overview

- **Parsing:** Analyze SQL syntax and structure.
- **Conversion:** Translate SQL to relational algebra, a set-based query language.
- **Execution:** Database engine executes relational algebra expressions.
- **Optimization:** Database engine optimizes operations for performance.

Relational Algebra in SQL

- **Key Operations:**
  - *Selection (σ):* Filters rows based on criteria.
  - *Projection (π):* Retrieves specific columns.
  - *Join (⋈):* Combines rows from different tables.

```
[Members Table]
       |
       | (σ JoinYear=2020)
       V
[Filtered Rows - 2020]
       |
       | (π Name)
       V
[Result - Names of Members Joined in 2020]
```

Example Translation

- SQL:

```sql
SELECT Name FROM Members WHERE JoinYear = 2020;
```

- Relational Algebra:

```
π Name (σ JoinYear=2020 (Members))
```

For-Each Semantics

Understanding Through Examples:

- Source Table: 'Books':

BookID	Title	Author	Genre	PublishedYear
101	Journey Through SQL	A. Coder	Technology	2015
102	The History of Databases	D. Base	Education	2018
103	Adventures in Coding	P. Programmer	Fiction	2020
...	...	...	...	...

- SQL Query:

```sql
SELECT Title, Author
FROM Books
WHERE Genre = 'Technology';
```

- Pseudocode Equivalent:

```
for each row in Books:
    if (row.Genre == 'Technology'):
        output row.Title
```

- Retrieves titles from the 'Books' table where the genre is 'Technology'.

For-Each Semantics

Understanding Through Examples:

- Source Table: 'Books'

BookID	Title	Author	Genre	PublishedYear
101	Journey Through SQL	A. Coder	Technology	2015
102	The History of Databases	D. Base	Education	2018
103	Adventures in Coding	P. Programmer	Fiction	2020
...	...	...	...	...

- SQL Query:

```sql
SELECT Title, Author
FROM Books
WHERE Genre = 'Education';
```

- Pseudocode Equivalent:

```
for each row in Books:
    if (row.Genre == 'Education'):
        output row.Title
```

- Retrieves titles from the 'Books' table where the genre is 'Technology'.
- Output:

Title	Author

For-Each Semantics

Understanding Through Examples:

- Source Table: 'Books':

BookID	Title	Author	Genre	PublishedYear
101	Journey Through SQL	A. Coder	Technology	2015
102	The History of Databases	D. Base	Education	2018
103	Adventures in Coding	P. Programmer	Fiction	2020
...	...	...	...	...

- SQL Query:

```sql
SELECT Title, Author
FROM Books
WHERE Genre = 'Education';
```

- Pseudocode Equivalent:

```
for each row in Books:
    if (row.Genre == 'Education'):
        output row.Title
```

- Retrieves titles from the 'Books' table where the genre is 'Technology'.
- Output:

Title	Author
The History of Databases	D. Base

For-Each Semantics

Understanding Through Examples:

- Source Table: 'Books':

BookID	Title	Author	Genre	PublishedYear
101	Journey Through SQL	A. Coder	Technology	2015
102	The History of Databases	D. Base	Education	2018
103	Adventures in Coding	P. Programmer	Fiction	2020
...	...	...	...	...

- SQL Query:

```sql
SELECT Title, Author
FROM Books
WHERE Genre = 'Education';
```

- Pseudocode Equivalent:

```
for each row in Books:
    if (row.Genre == 'Eduction'):
        output row.Title
```

- Retrieves titles from the 'Books' table where the genre is 'Technology'.
- Output:

Title	Author
The History of Databases	D. Base

For-Each Semantics

Understanding Through Examples:

- Source Table: 'Books':

BookID	Title	Author	Genre	PublishedYear
101	Journey Through SQL	A. Coder	Technology	2015
102	The History of Databases	D. Base	Education	2018
103	Adventures in Coding	P. Programmer	Fiction	2020
...	...	...	...	...

- SQL Query:

```sql
SELECT Title, Author
FROM Books
WHERE Genre = 'Education';
```

- Pseudocode Equivalent:

```
for each row in Books:
    if (row.Genre == 'Eduction'):
        output row.Title
```

- Retrieves titles from the 'Books' table where the genre is 'Technology'.
- Output:

Title	Author
The History of Databases	D. Base
...	...

SQL Functions: ORDER BY

- **Purpose:** Organizes query results by sorting data in either ascending or descending order.
- **Key Usage:** Essential for presenting data in a readable and meaningful order, especially in reports and analytics.

Example and Explanation

- SQL Query:

```
SELECT Title, PublishedYear
FROM Books
ORDER BY PublishedYear DESC;
```

- **Purpose:** To sort books by their publication year, starting with the most recent.

- Output Table:

```
| Title                     | PublishedYear |
| ------------------------- | ------------- |
| Adventures in Coding      | 2020          |
| Mysteries of the Universe | 2019          |
| The History of Databases  | 2018          |
| ...                       | ...           |
```

SQL Functions: DISTINCT

- **Purpose:** Eliminates duplicate rows from query results, providing a set of unique records.
- **Key Usage:** Useful in queries where identifying unique values is crucial, such as summarizing data or ensuring data quality.

Example and Explanation

- **SQL Query:**

```
SELECT DISTINCT Genre
FROM Books;
```

- **Purpose:** To list all unique genres available in the 'Books' table.
- **Output Table:**

```
| Genre      |
| ---------- |
| Technology |
| Education  |
| Fiction    |
| ...        |
```

SQL Functions: COUNT

- Purpose: The `COUNT` function in SQL is used to count the number of rows in a table or rows that match a certain condition.
- Common Uses: `COUNT` is essential for data analysis, particularly in scenarios requiring the quantification of data, such as calculating totals, averages, or identifying data density.

Example and Explanation

- **SQL Query:**

```
SELECT COUNT(*) AS TotalBooks
FROM Books;
```

- **Purpose:** Determine the total number of books in the 'Books' table and label the output as 'TotalBooks'.
- **Output Table:**

```
| TotalBooks |
| ---------- |
| 5          |
```

Creating and Managing Tables

Table Creation in SQL:

- Table Structures in Relational Databases: Tables are the primary structure for storing data in SQL databases, composed of rows and columns.
- Using `CREATE TABLE` Statement: The statement is crucial for defining new tables within a database, specifying column names, data types, and constraints.

SQL Query:

- **Example: Creating the 'Books' Table:**

```sql
CREATE TABLE Books (
  BookID INT PRIMARY KEY,
  Title VARCHAR(100),
  Author VARCHAR(100),
  Genre VARCHAR(100),
  PublishedYear INT
);
```

- **`BookID INT PRIMARY KEY`:** Unique identifier for each record.

Data Types

Understanding SQL Data Types

- **Purpose of Data Types:**
  - Data types define the kind of data a column can hold in an SQL table. Choosing the correct data type is crucial for data integrity and query performance.

Common SQL Data Types

- **Integer (INT):** A whole number without a decimal.
- **Character (CHAR):** A fixed-length string.
- **Variable Character (VARCHAR):** A string of text of variable length.
- **Date and Time Types (DATE, TIME, DATETIME):** Represent dates and times.
- **Floating Point (FLOAT, DOUBLE):** Numbers with fractional parts.

CHAR vs VARCHAR

Text Data Types

- CHAR(size): A fixed-length string. Space-efficient for strings that are always the same length.
- VARCHAR(size): A variable-length string. Ideal for strings that vary in length.

Usage Examples and Considerations

- Choosing CHAR:
  - Use CHAR for data that is consistently the same size, such as country codes (CHAR(2) for 'US', 'UK').
  - Example: StateCode CHAR(2)
- Choosing VARCHAR:
  - Use VARCHAR for data that varies in size, such as names or addresses.
  - Example: Title VARCHAR(100)

Performance Implications

- CHAR:
  - Faster access due to fixed length. Can waste storage space if used for variable-length data.
- VARCHAR:
  - More flexible and storage-efficient for variable-length data. Slightly slower access due to variable length management.

Entity Integrity

Understanding Primary Key

- **Definition:** A Primary Key is a unique identifier for each row in a database table.
- **Purpose:** Ensures **entity integity**, i.e. uniqueness and integrity, of the data, preventing duplicate records.

Example:

- Books Table:

```
| BookID (PK) | Title                     | Author          | Genre       | PublishedYear |
|-------------|---------------------------|-----------------|-------------|---------------|
| 101         | Journey Through SQL       | A. Coder        | Technology  | 2015          |
| 102         | The History of Databases  | D. Base         | Education   | 2018          |
| 103         | Adventures in Coding      | P. Programmer   | Fiction     | 2020          |
| ...         | ...                       | ...             | ...         | ...           |
```

- Failed Insert Query:

```
INSERT INTO Books (BookID, Title, Author, Genre, PublisedYear)
VALUES (101, ‘Data Mastery’, ‘L. Data’, 2021);
```

- This query attempts to insert a new book record with an existing `BookID` (101).
- Result:
  - The database system rejects the query and throws an error (e.g., `Duplicate entry '101' for key 'PRIMARY'`).

Composite Primary Keys

Advanced Key Concepts

- **Definition:** A composite primary key is formed by using two or more columns in combination to create a unique identifier for each row in a table. It is used when no single column can serve as a unique identifier.

Example: Reservation Table with Composite Key

- Creating the BookReservations Table:

```sql
CREATE TABLE Borrow (
    BookID INT,
    MemberID INT,
    BorrowDate DATE,
    ReturnDate DATE,
    PRIMARY KEY (BookID, MemberID, BorrowDate),
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);
```

When to Use

- Applicable in many-to-many relationships or when the uniqueness is only guaranteed by the combination of multiple attributes.

- BookReservations Table:

BookID (PK,FK)	MemberID (PK, FK)	BorrowDate (PK)	ReturnDate
101	501	2022-01-15	2022-01-30
102	502	2022-01-16	2022-01-31
101	501	2022-02-01	2022-02-15

Foreign Keys

Understanding Foreign Keys

- **Definition:** A Foreign Key in one table is a field that references a Primary Key in another table, creating a link between the two.
- **Purpose:** Ensures **referential integrity** and enables **relationships** across tables in a relational database.

SQL Query: Adding a Foreign Key

- Example: Linking 'Borrow' and 'Books' Tables

```
CREATE TABLE Borrow (
    BookID INT,
    MemberID INT,
    BorrowDate DATE,
    ReturnDate DATE,
    PRIMARY KEY (BookID, MemberID, BorrowDate),
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);
```

- Key Component:
  - `FOREIGN KEY (BookID) REFERENCES Books(BookID)`: `BookID` in the 'Borrow' table is the foreign key that references `BookID` in the 'Books' table.

Referencial Integrity

Maintaining Data Consistency

- Foreign Keys maintain consistency by ensuring that the relationship between the data in two tables remains valid and that any changes to the data do not violate the integrity of the database.

Example: Enforcing Referential Integrity

- Books Table:

```
| BookID (PK) | Title           | Author       |
|-------------|-----------------|--------------|
| 101         | SQL Essentials  | A. Coder     |
| 102         | Database Deep   | D. Analyst   |
```

- Borrow Table:

```
| BookID (FK) | MemberID | BorrowDate | ReturnDate  |
|-------------|----------|------------|-------------|
| 101         | 501      | 2022-01-15 | 2022-01-30  |
| 102         | 502      | 2022-01-16 | 2022-01-31  |
| 101         | 501      | 2022-02-01 | 2022-02-15  |
```

- Attempted Deletion from 'Books' Table:
- SQL Query:

```sql
DELETE FROM Books WHERE BookID = 101;
```

- Result: The database system rejects the deletion because 'Borrow' table has a foreign key that depends on the book record being deleted.

```
ERROR: Cannot delete or update a parent row: a foreign key constraint fails (`database`.`borrow`, CONSTRAINT `borrow_bookid_fk`
FOREIGN KEY (`BookID`) REFERENCES `books` (`BookID`))
```

Hand-On SQL Demonstration

- **Focus:** Applying SQL in a realistic library database context.
- **Goal:** Illustrate effective data extraction and manipulation using SQL.

Example Query

- **Identify Fiction Titles:**
  - *SQL Query:*

```
SELECT Title, Author
FROM Books
WHERE Genre = 'Technology';
```

  - *Purpose:* Demonstrate SQL's ability to filter data based on specific criteria.

Try It Yourself

1. **Retrieve Recent Books:**
   - *Task:* Write a query to list all books published after 2015.
2. **Order By Genre:**
   - *Task:* Modify the query to sort the results by genre in alphabetical order.

Recap and Key Takeaways

- SQL is a declarative language basing on set operations. We specify 'What' from the data, but not 'How' to get it. SQL enables data retreival and manipulation to be carried out at scale.
- Entity- and Referential-Integrities make data consistency possible across the business. Primary Key (PK) enforces uniqueness within the table, and Foriegn Key (FK) ensures accurate information referring from the other tables.

Example SQL Recap:

```sql
SELECT Title, PublishedYear
FROM Books
WHERE PublishedYear > 2005
ORDER BY PublishedYear DESC;
```

- **Purpose:** Demonstrated a SQL application in targeted data retrieval and organising the query result.

Preparing for Joining Tables:

- Next Lecture Preview:
  - Dive deeper into SQL **JOIN**s.
  - Explore **Relational Algebra** equivalencies

Why is maintaining data integrity crucial in a database, and how do primary and foreign keys contribute to this process?