NATIONAL UNIVERSITY
COLLEGE OF COMPUTING AND
INFORMATION TECHNOLOGIES

VER 1.0

Course MANUAL

DATABASE SYSTEMS

This course provides students with the theoretical knowledge and practical skills in the utilization of databases and database management systems in the ICT applications. The logical design, physical design and implementation of relational databases are all covered in this course.

INFORMATION TECHNOLOGY

JULIE ANNE ANGELES-CRYSTAL

ADVANCE DATABASE MANAGEMENT SYSTEMS

Table of Contents

MODULE

#1

NATIONAL UNIVERSITY

FLEX Course Material

Introduction to Advance Database Concept

ADVANCE DATABASE SYSTEMS

MODULE GOALS

- ☑ Introduction to Advance Database Concept
- ☑ Review of Relational Databases and SQL
- ☑ Functions and Operators
- ☑ Joins and Aggregate Functions

Education that works.

LESSON #

#1

Introduction to Advanced Database Concepts

In the evolving world of technology, databases have become the backbone of modern software systems. While basic database concepts like tables, keys, and SQL queries provide a solid foundation, **advanced database concepts** delve deeper into optimizing, securing, and scaling databases for high-performance applications.

Why study advanced database concepts?

•To handle **large-scale data** efficiently.
•To ensure **data integrity, security, and consistency**.
•To implement **complex querying and indexing techniques**.
•To support **distributed databases** and **cloud-based data storage**.

LESSON #

#1

Introduction to Advanced Database Concepts

Importance of Advanced Database Systems

1.Handling Big Data:

1. Advanced databases manage massive volumes of data generated by social media, IoT, and e-commerce platforms.
2. Example: Shopee uses NoSQL databases to quickly process user reviews and product data.

2.Real-time Data Processing:

1. Distributed and time-series databases allow businesses to make instant decisions based on live data streams.
2. Example: Stock trading apps use time-series databases to monitor price changes.

3.Enhanced Scalability:

1. NoSQL and cloud databases let companies add servers as their data grows, unlike traditional systems limited by hardware.
2. Example: Netflix uses distributed databases to handle user preferences and viewing history.

4.Support for Unstructured Data:

1. Ideal for managing text, images, and videos — critical for apps like YouTube or Instagram.
2. Example: MongoDB stores flexible JSON-like documents for user profiles.

MODULE

#1

NATIONAL UNIVERSITY

MODULE GOALS

- ☑ Introduction to Advance Database Concept
- ☑ Review of Relational Databases and SQL
- ☑ Functions and Operators
- ☑ Joins and Aggregate Functions

FLEX Course Material

Review of Relational Databases and SQL

ADVANCE DATABASE SYSTEMS

Education that works.

LESSON #

#1

Review of Relational Database and SQL

Introduction to Relational Databases

A **Relational Database** is a type of database that stores and organizes data in tables, where relationships between the data are defined.

It follows a structured format, using rows and columns, making it easy to retrieve, manage, and manipulate information through SQL (Structured Query Language).

Relational databases are widely used because of their flexibility, accuracy, and ability to handle large amounts of interconnected data.

A **Relational Database** is a collection of data organized into **tables** (also known as relations).

Each table contains **rows** (records) and **columns** (fields), with every row representing a unique data entry and each column representing a data attribute.

The power of relational databases lies in their ability to establish relationships between tables using keys, ensuring data integrity and reducing redundancy.

•**Key Concepts**: Tables, Rows, Columns

LESSON #

#1

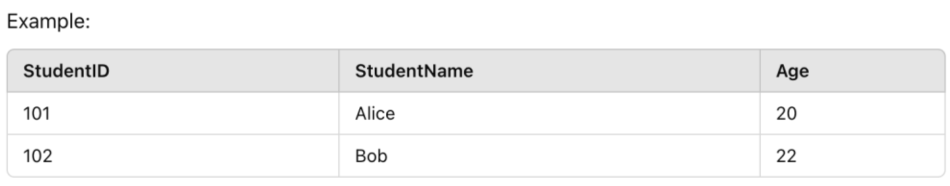
Review of Relational Database and SQL

•**Key Concepts**: Tables, Rows, Columns

Tables: Structured formats where data is stored, similar to spreadsheets. Each table focuses on a specific entity — for example, a "Students" table or a "Products" table.

Rows: Also known as **records** or **tuples** — each row contains data entries, such as one student’s full record in the "Students" table.

Columns: Also called **fields** or **attributes** — each column represents a particular property of the entity, such as "StudentName" or "Age."



Example:

StudentID	StudentName	Age
101	Alice	20
102	Bob	22

•**Primary Keys and Foreign Keys**

•**Relationships**: One-to-One, One-to-Many, Many-to-Many

LESSON #

#1

Review of Relational Database and SQL

•**Primary Keys and Foreign Keys**

Primary Key: A unique identifier for each record in a table. It ensures that each row can be distinctly identified. Example: "StudentID" in the "Students" table.

Foreign Key: A field in one table that refers to the **Primary Key** in another table. It establishes a link between two tables and ensures referential integrity. Example: "CourseID" in the "Enrollments" table linking to the "Courses" table.

•**Relationships**: One-to-One, One-to-Many, Many-to-Many

•**Importance of Relational Databases** in modern applications

Review of Relational Database and SQL

•**Relationships**: One-to-One, One-to-Many, Many-to-Many

One-to-One (1:1): Each record in Table A corresponds to a single record in Table B. Example: A "Person" table and a "Passport" table — each person has one passport.

One-to-Many (1:N): A record in Table A can be associated with many records in Table B, but a record in Table B relates to only one in Table A. Example: A "Department" table and an "Employee" table — one department can have many employees.

Many-to-Many (M:N): Multiple records in Table A relate to multiple records in Table B, usually requiring a **junction table**. Example: A "Students" table and a "Courses" table — students can enroll in many courses, and each course can have many students.

•**Importance of Relational Databases** in modern applications

LESSON #

#1

Review of Relational Database and SQL

•**Importance of Relational Databases** in modern applications

Relational databases play a critical role in modern applications due to their robust structure and flexibility. Here's why they are essential:

Data Integrity and Accuracy

Using **primary and foreign keys** ensures that relationships between data remain consistent, minimizing errors and redundancy. This is vital for applications handling sensitive data, like banking systems or healthcare platforms.

Scalability and Flexibility

Relational databases can grow with an application. They allow adding new tables and fields without disrupting existing relationships crucial for e-commerce platforms like Shopee, which constantly update product catalogs and user data.

LESSON #

#1

Review of Relational Database and SQL

•**Importance of Relational Databases** in modern applications

Complex Query Handling

SQL queries allow for complex data retrieval, helping businesses generate detailed reports, find trends, and make data-driven decisions. For example, an online store can easily pull up sales by product category or region.

Security

Most relational database management systems (RDBMS) offer user access control, ensuring only authorized users can view or modify data — critical for applications dealing with financial records or user credentials.

Integration with Modern Technologies

Relational databases integrate seamlessly with modern web and mobile applications. They support APIs, allowing platforms to sync data across multiple services — like a ride-hailing app updating driver and customer data in real time.

MODULE

#1

NATIONAL UNIVERSITY

FLEX Course Material

Functions and Operators

MODULE GOALS

- ☑ Introduction to Advance Database Concept
- ☑ Review of Relational Databases and SQL
- ☑ Functions and Operators
- ☑ Joins and Aggregate Functions

ADVANCE DATABASE SYSTEMS

Education that works.

LESSON #

#1

Functions and Operators

Functions and Operators in Databases

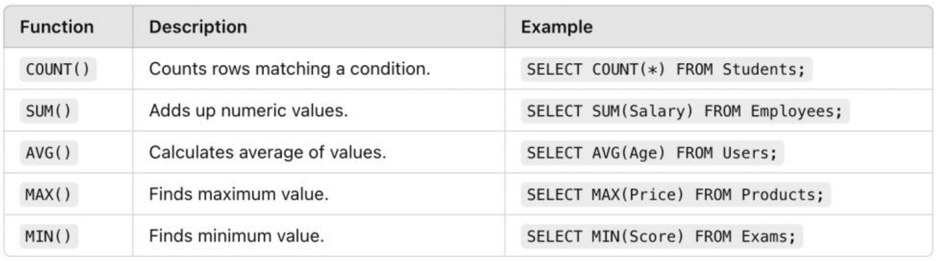
In databases, **functions** and **operators** are essential tools for manipulating and analyzing data. They help you perform calculations, filter results, and transform data directly within SQL queries.

What are Functions?

Functions are pre-defined operations that take inputs (arguments), process them, and return results.

They are commonly divided into:

•**Aggregate Functions** — operate on multiple rows to produce a single result.



Function	Description	Example
`COUNT()`	Counts rows matching a condition.	`SELECT COUNT(*) FROM Students;`
`SUM()`	Adds up numeric values.	`SELECT SUM(Salary) FROM Employees;`
`AVG()`	Calculates average of values.	`SELECT AVG(Age) FROM Users;`
`MAX()`	Finds maximum value.	`SELECT MAX(Price) FROM Products;`
`MIN()`	Finds minimum value.	`SELECT MIN(Score) FROM Exams;`

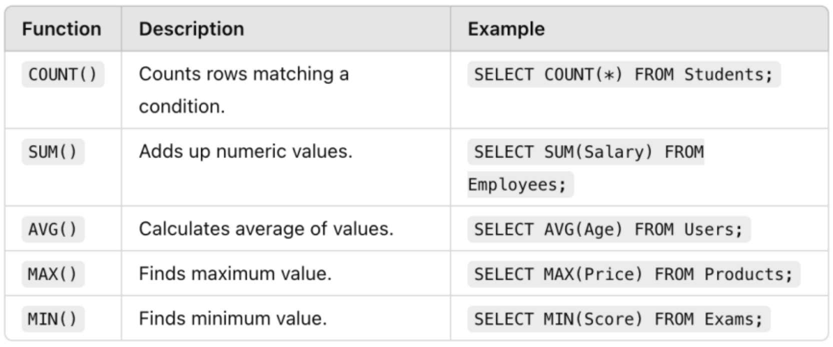
LESSON #

#1

Functions and Operators

Functions and Operators in Databases

Scalar Functions — operate on individual rows and return one result per row



Function	Description	Example
`COUNT()`	Counts rows matching a condition.	`SELECT COUNT(*) FROM Students;`
`SUM()`	Adds up numeric values.	`SELECT SUM(Salary) FROM Employees;`
`AVG()`	Calculates average of values.	`SELECT AVG(Age) FROM Users;`
`MAX()`	Finds maximum value.	`SELECT MAX(Price) FROM Products;`
`MIN()`	Finds minimum value.	`SELECT MIN(Score) FROM Exams;`

EXAMPLE:

Find the total number of students enrolled in a course:

```
SELECT CourseID, COUNT(StudentID)
FROM Enrollments
GROUP BY CourseID;
```

Format product prices to two decimal places:

```
SELECT ProductName, ROUND(Price, 2)
FROM Products;
```

LESSON #

#1

Functions and Operators

Functions and Operators in Databases

Operators are symbols used to perform operations on data. They're crucial for filtering data and forming conditions in queries.

Arithmetic Operators:

Operator	Description	Example
`+`	Addition	`SELECT Price + 100 FROM Products;`
`-`	Subtraction	`SELECT Stock - 5 FROM Items;`
`*`	Multiplication	`SELECT Salary * 2 FROM Employees;`
`/`	Division	`SELECT Price / 2 FROM Products;`
`%`	Modulus	`SELECT Quantity % 2 FROM Orders;`

Comparison Operators:

Operator	Description	Example
`=`	Equal to	`SELECT * FROM Users WHERE Age = 25;`
`!=` or `<>`	Not equal to	`SELECT * FROM Products WHERE Price <> 100;`
`>`	Greater than	`SELECT * FROM Orders WHERE Amount > 500;`
`<`	Less than	`SELECT * FROM Students WHERE Age < 20;`
`>=`	Greater than or equal to	`SELECT * FROM Scores WHERE Score >= 90;`
`<=`	Less than or equal to	`SELECT * FROM Sales WHERE Discount <= 10;`

LESSON #

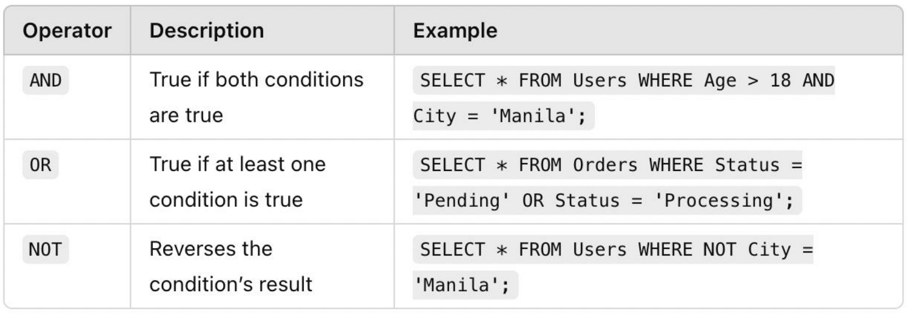
#1

Functions and Operators

Functions and Operators in Databases

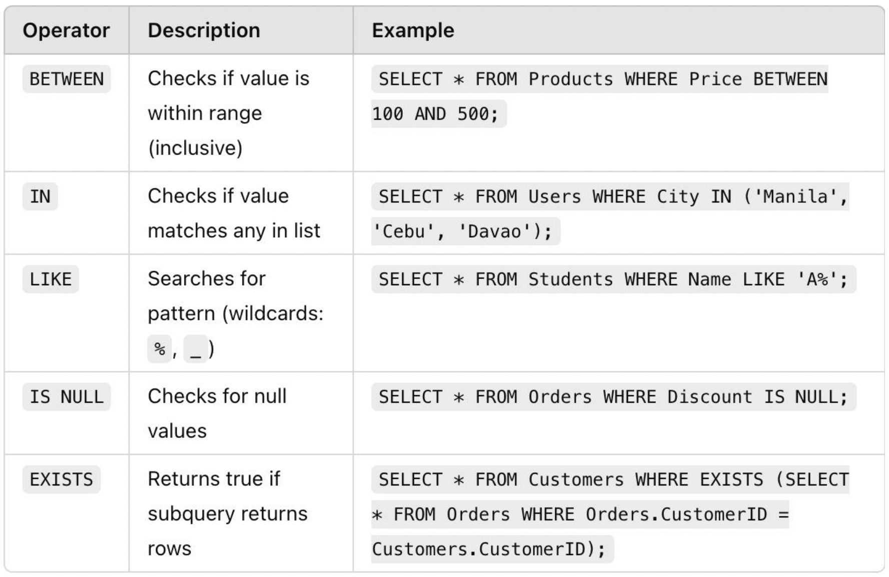
Operators are symbols used to perform operations on data. They're crucial for filtering data and forming conditions in queries.

Logical Operators:



Operator	Description	Example
`AND`	True if both conditions are true	`SELECT * FROM Users WHERE Age > 18 AND City = 'Manila';`
`OR`	True if at least one condition is true	`SELECT * FROM Orders WHERE Status = 'Pending' OR Status = 'Processing';`
`NOT`	Reverses the condition's result	`SELECT * FROM Users WHERE NOT City = 'Manila';`

Special Operators:



Operator	Description	Example
`BETWEEN`	Checks if value is within range (inclusive)	`SELECT * FROM Products WHERE Price BETWEEN 100 AND 500;`
`IN`	Checks if value matches any in list	`SELECT * FROM Users WHERE City IN ('Manila', 'Cebu', 'Davao');`
`LIKE`	Searches for pattern (wildcards: `%`, `_`)	`SELECT * FROM Students WHERE Name LIKE 'A%';`
`IS NULL`	Checks for null values	`SELECT * FROM Orders WHERE Discount IS NULL;`
`EXISTS`	Returns true if subquery returns rows	`SELECT * FROM Customers WHERE EXISTS (SELECT * FROM Orders WHERE Orders.CustomerID = Customers.CustomerID);`

MODULE

#1

MODULE GOALS

- ☑ Introduction to Advance Database Concept
- ☑ Review of Relational Databases and SQL
- ☑ Functions and Operators
- ☑ Joins and Aggregate Functions

NATIONAL UNIVERSITY

FLEX Course Material

Joins and Aggregate Functions

ADVANCE DATABASE SYSTEMS

Education that works.

LESSON #

#1

Joins & Aggregate Functions

A **join** is used to combine rows from two or more tables based on a related column.

Joins and **Aggregate Functions** allows you to combine data from multiple tables and perform calculations on data sets, giving you powerful ways to analyze and extract insights from relational databases.

INNER JOIN

•Returns rows that have matching values in both tables.

•**Use case:** Retrieve orders along with customer names for orders that have customer details.

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Joins & Aggregate Functions

LEFT JOIN (or LEFT OUTER JOIN)

•Returns all rows from the left table and the matched rows from the right table. Unmatched rows from the right table will be NULL.

•**Use case:** Show all customers, even those who haven't placed any orders.

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

RIGHT JOIN (or RIGHT OUTER JOIN)

•Returns all rows from the right table and matched rows from the left table.

•**Use case:** List all orders, even if some customer information is missing.

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
RIGHT JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Joins & Aggregate Functions

FULL JOIN (or FULL OUTER JOIN)

•Combines all rows from both tables, returning matched rows and NULL for unmatched rows.

•**Use case:** Get a complete list of customers and orders, whether or not they have matches.

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

CROSS JOIN

•Returns the Cartesian product of two tables every combination of rows from both tables.

•**Use case:** Generate all possible combinations of customers and products.

```
SELECT Customers.CustomerName, Products.ProductName
FROM Customers
CROSS JOIN Products;
```

LESSON #

#1

Joins & Aggregate Functions

Using Aggregate Functions with GROUP BY

The GROUP BY clause groups rows sharing a property so you can use aggregate functions on each group.

Example: Get the total amount spent by each customer:

```
SELECT Customers.CustomerName, SUM(Orders.Amount) AS TotalSpent
FROM Customers
JOIN Orders ON Customers.CustomerID = Orders.CustomerID
GROUP BY Customers.CustomerName;
```

•**JOIN** combines the tables.
•**SUM()** calculates the total.
•**GROUP BY** organizes the results by customer.

Example: Find the number of orders and total sales per product:

```
SELECT Products.ProductName, COUNT(Orders.OrderID) AS OrderCount,
SUM(Orders.Amount) AS TotalSales
FROM Products
JOIN Orders ON Products.ProductID = Orders.ProductID
GROUP BY Products.ProductName;
```

•**JOIN** links the Products and Orders tables.
•**COUNT()** counts the number of orders for each product.
•**SUM()** adds up the sales amounts.
•**GROUP BY** organizes the results by product name.

LESSON #

#1

LABORATORY ACTIVITY

Date:	
Name:	
Year & Section:	

ADVANCE DATABASE SYSTEMS

INSTRUCTIONS:
•COLLATE ALL THE SCREENSHOT (ALL SQL COMMANDS & QUERY OUTPUT) OF THE FOLLOWING EXERCISES.
•SAVE FILE AS SURNAME_WEEK1_ACTIVITY.PDF AND TURN IN YOUR WORK.

EXERCISE 1: PERFORM ALL THE QUERIES

•Create the Students Table:
Columns: StudentID, FirstName, LastName, Age, and Grade.
•Insert Data into the Students Table using INSERT statements. (20 record)
• Populate the list of student surnames with only those that start with the same letter as your surname. For example, if your surname is 'Angeles,' include only surnames that begin with the letter 'A.' Ensure that all surnames follow this rule:

Write SQL queries to perform the following tasks:
1. Retrieve all students from the Students table.
2. Retrieve students who are 18 years old or older.
3. Calculate the average grade of all students.
4. Update the grade of a specific student.
5. Delete a student record from the table
6. Calculate the total number of students.
7. Retrieve students who have grade higher than 90.
8. Retrieve students who are either 17 years old or have a grade higher than 90.
9. Calculate the maximum grades among students.
10. Calculate the minimum grades among students.