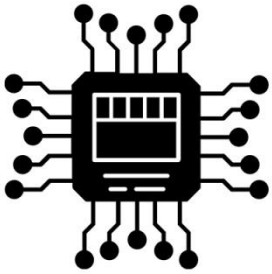

قسم علوم الذكاء الاصطناعي
Department of Artificial Intelligence

Subject:

Database

Class:

SECOND

Lecturer:

M.Sc. Ali Haider Alazam

Noor Hassan Aldulimi

Laboratory : (3)

SELECT

1. Database Basics and SQL Syntax Rules

Before writing queries, it is important to understand the environment you are working in. A database typically contains one or more tables. Each table has a unique name (such as "Customers" or "Orders") and is made up of columns (attributes) and rows (records containing the actual data).

Most of the actions you perform on a database use **SQL (Structured Query Language)** statements, which are built using straightforward, English-like keywords.

Important Rules to Remember:

• **Case Insensitivity:** SQL keywords are NOT case sensitive. Writing select is exactly the same as writing SELECT.

• **Semicolons:** A semicolon (;) is the standard way to separate SQL statements. Some database systems strictly require a semicolon at the end of every statement.

While there are many SQL commands (like UPDATE to modify data, DELETE to remove data, and INSERT INTO to add data), the most common command you will use is SELECT.

2. The SELECT Statement: Extracting Columns

The SELECT statement is used to extract specific data from your database. Think of it as choosing the *vertical* slices (columns) of your table that you want to see.

Basic Syntax:

```
SELECT column1, column2 FROM table_name;
```

Here, column1 and column2 are the specific names of the columns you want to see, and table_name is the table where the data lives.

Example 1: Fetching Specific Columns If you have an employees table and only need a list of names, you specify just those columns separated by a comma. Notice there is no comma

after the final column name before the FROM keyword.

```
SELECT first_name, last_name FROM employees;
```

Example 2: Fetching All Columns To select every column in a table without typing each name out, use the asterisk (*) wildcard. While this is great for exploring a new table, it is best practice to name your columns explicitly in professional applications to save memory.

```
SELECT * FROM employees;
```

Example 3: Renaming Columns in the Output (AS) If database column names are unclear, you can temporarily rename them in your results using the AS keyword.

```
SELECT first_name AS "First Name", salary AS "Annual Compensation" FROM employees;
```

Example 4:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

The SELECT DISTINCT Statement: Removing Duplicates

In a real-world database, a column might contain many duplicate values. For example, in an Employees table, the Department column might list "Sales" dozens of times. If you only want a list of the unique departments in your company, you use the DISTINCT keyword.

Basic Syntax:

```
SELECT DISTINCT column1, column2 FROM table_name;
```

Example: Finding Unique Values If you want to see all the different countries your customers come from without seeing the same country repeated for every individual customer:

```
SELECT DISTINCT Country FROM Customers;
```

Important Rules for DISTINCT:

- **The Position:** The DISTINCT keyword must come **immediately after** SELECT.
- **Multi-Column Distinct:** If you list more than one column (e.g., SELECT DISTINCT City, Country), SQL looks for unique *combinations*. It will only hide a row if both the City and the Country are identical to another row in the results.
- **Performance Note:** Using DISTINCT requires the database to sort and compare your data to find duplicates, which can be slower on very large tables. Only use it when you actually need unique values.