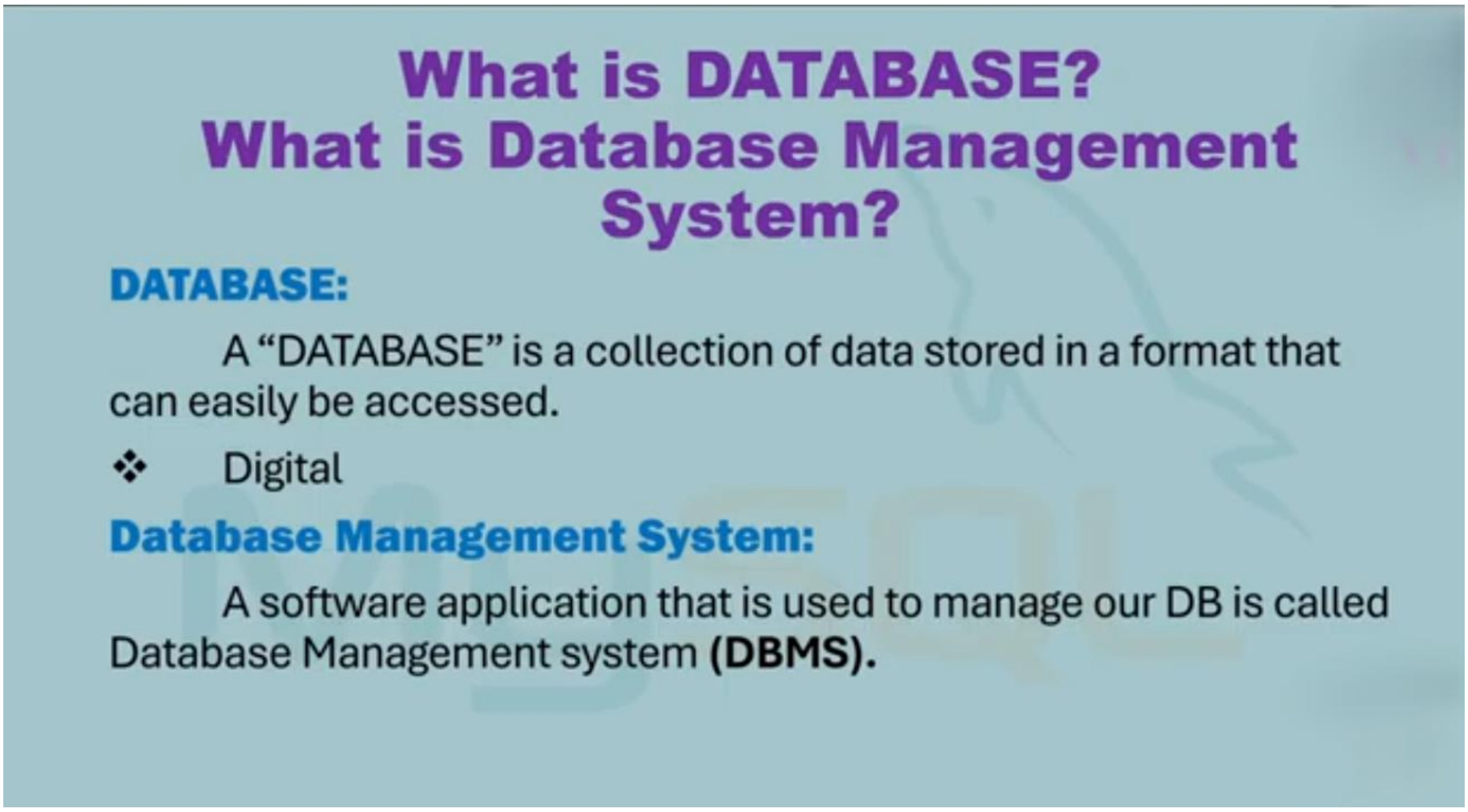


What is DATABASE? What is Database Management System?

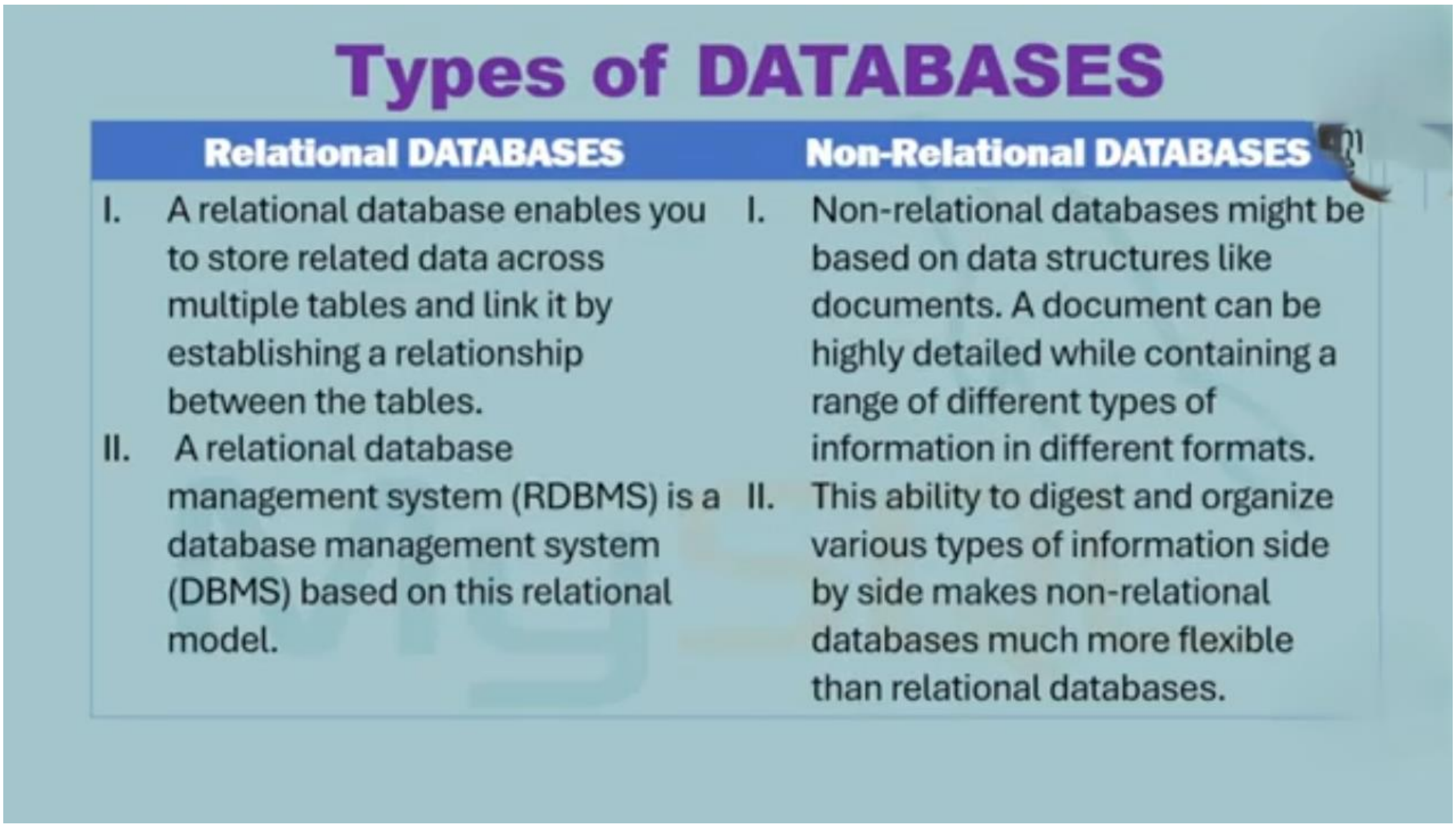
DATABASE:

A "DATABASE" is a collection of data stored in a format that can easily be accessed.

- ❖ Digital

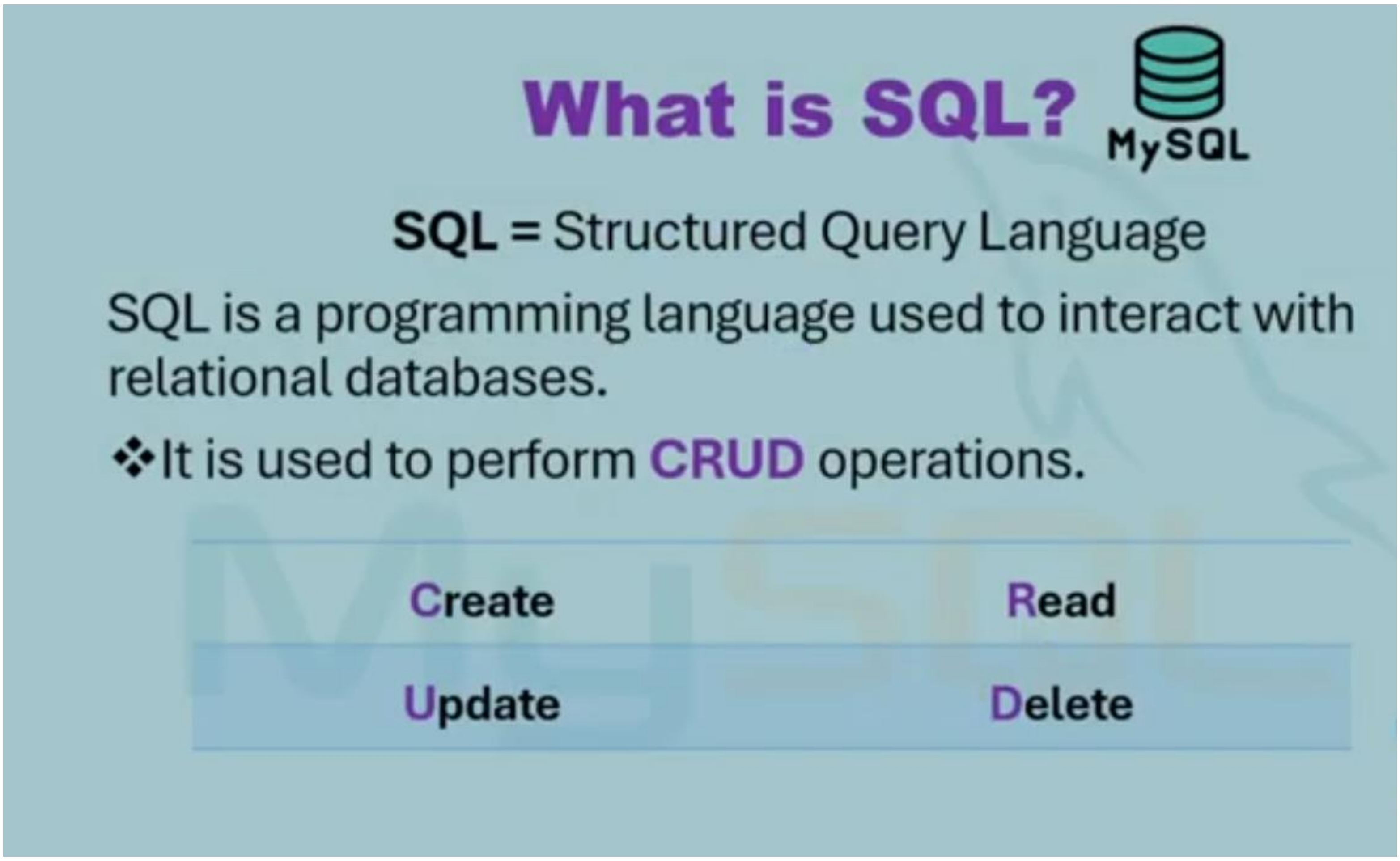
Database Management System:

A software application that is used to manage our DB is called Database Management system **(DBMS).**



Types of DATABASES

Relational DATABASES	Non-Relational DATABASES
I. A relational database enables you to store related data across multiple tables and link it by establishing a relationship between the tables.	I. Non-relational databases might be based on data structures like documents. A document can be highly detailed while containing a range of different types of information in different formats.
II. A relational database management system (RDBMS) is a database management system (DBMS) based on this relational model.	II. This ability to digest and organize various types of information side by side makes non-relational databases much more flexible than relational databases.



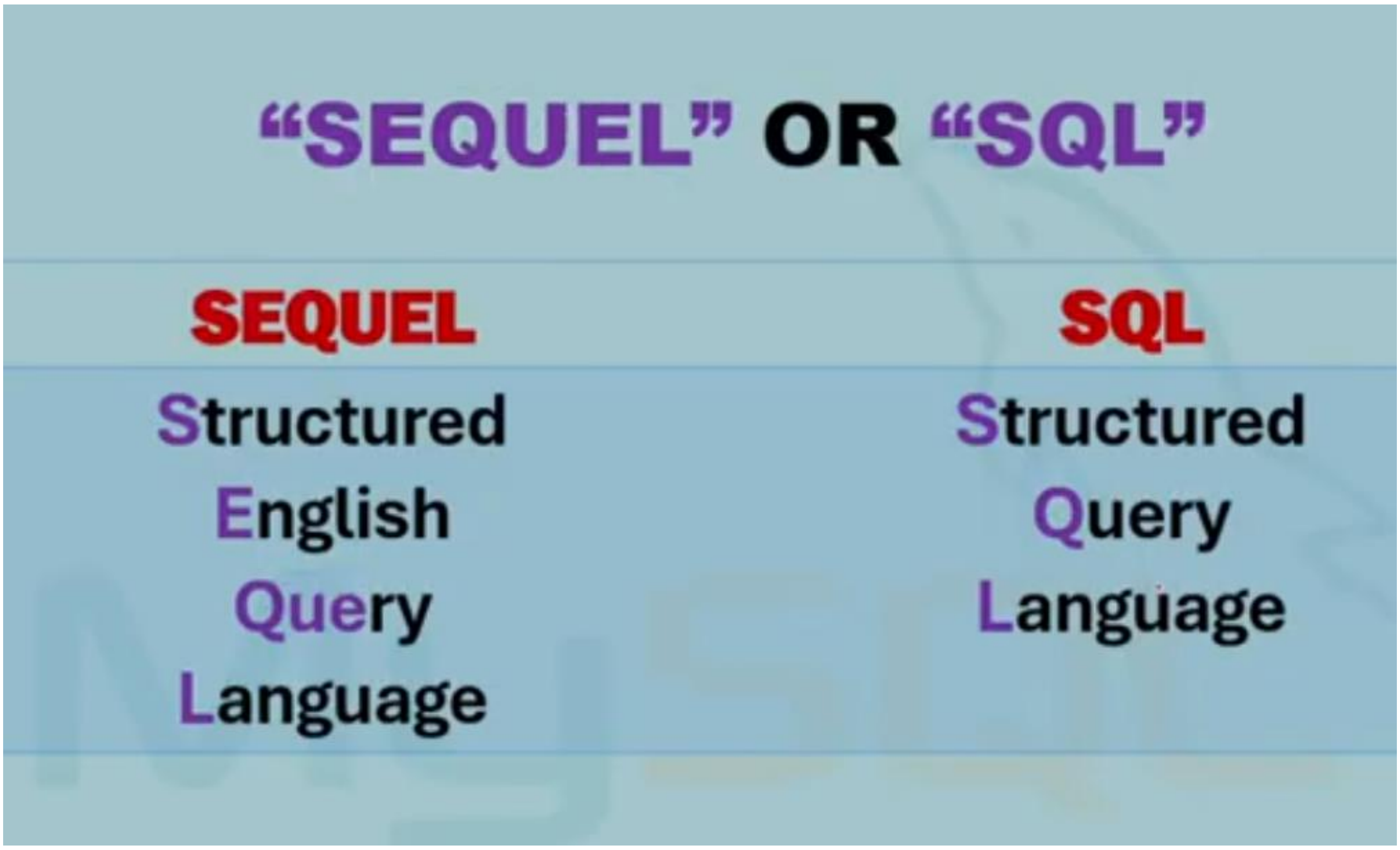
What is SQL?

SQL = Structured Query Language

SQL is a programming language used to interact with relational databases.

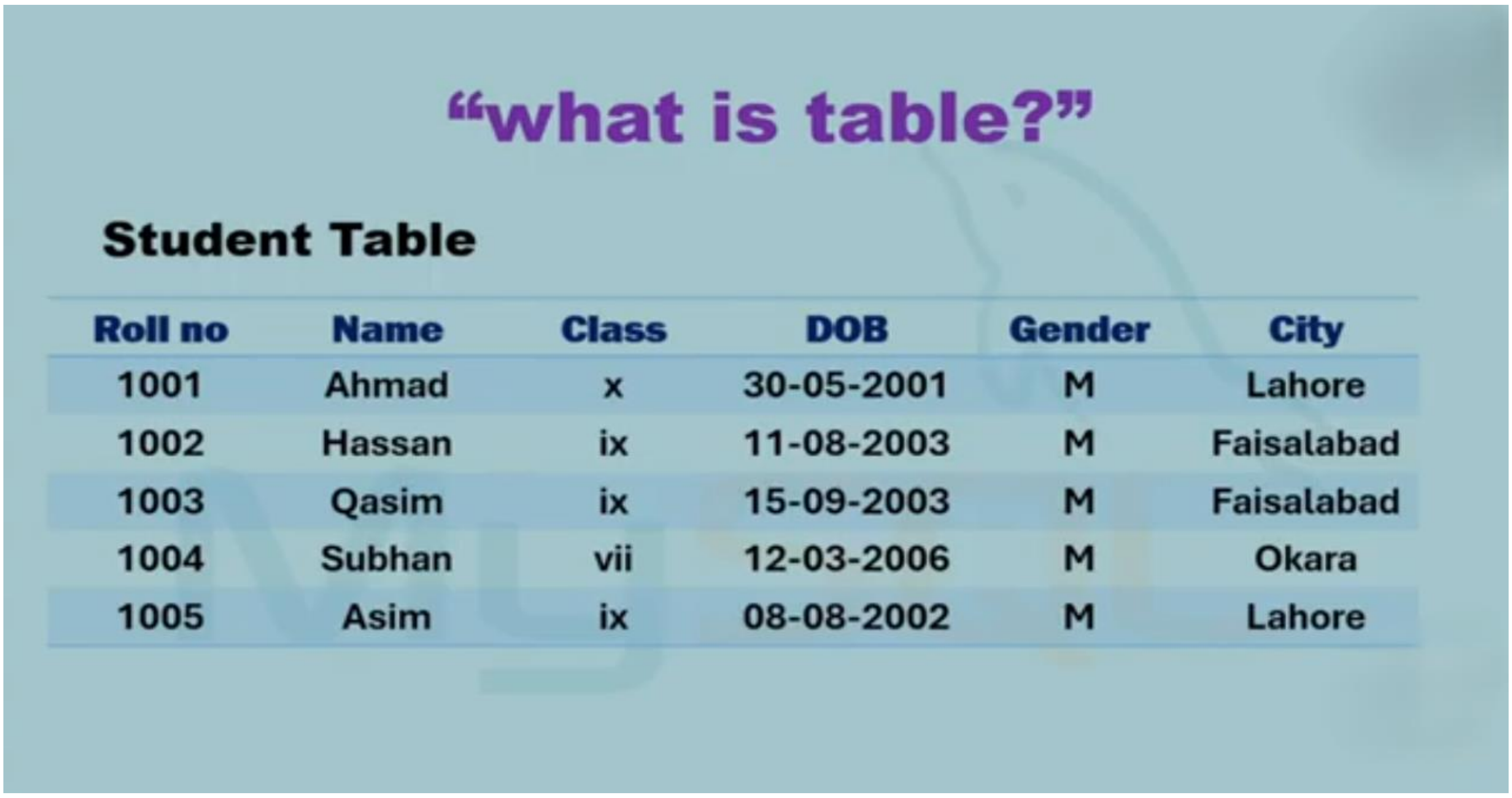
❖It is used to perform **CRUD** operations.

Create	**Read**
Update	**Delete**



“SEQUEL” OR “SQL”

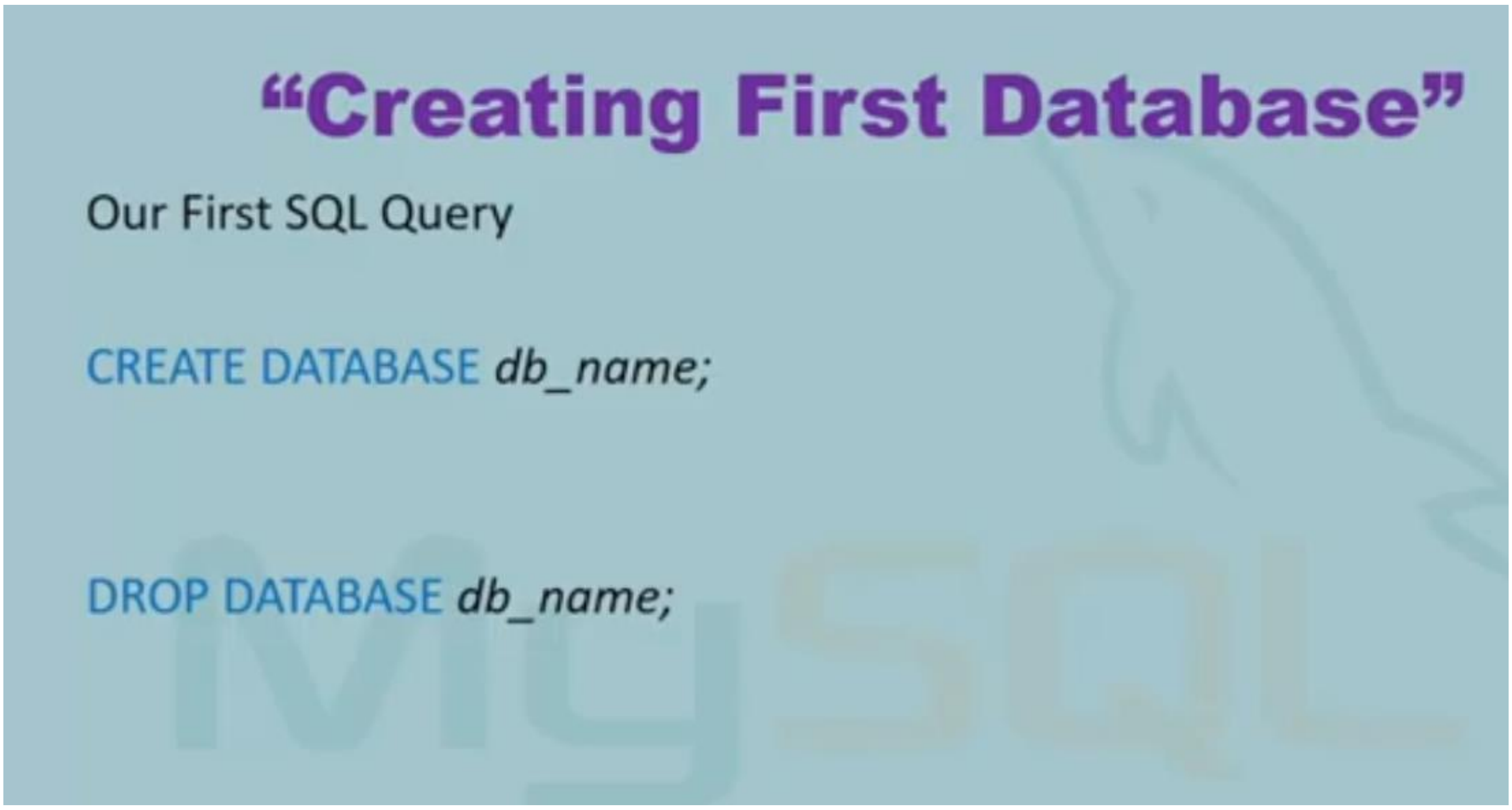
SEQUEL	**SQL**
Structured	**Structured**
English	**Query**
Query	**Language**
Language	



"what is table?"

Student Table

Roll no	Name	Class	DOB	Gender	City
1001	Ahmad	x	30-05-2001	M	Lahore
1002	Hassan	ix	11-08-2003	M	Faisalabad
1003	Qasim	ix	15-09-2003	M	Faisalabad
1004	Subhan	vii	12-03-2006	M	Okara
1005	Asim	ix	08-08-2002	M	Lahore

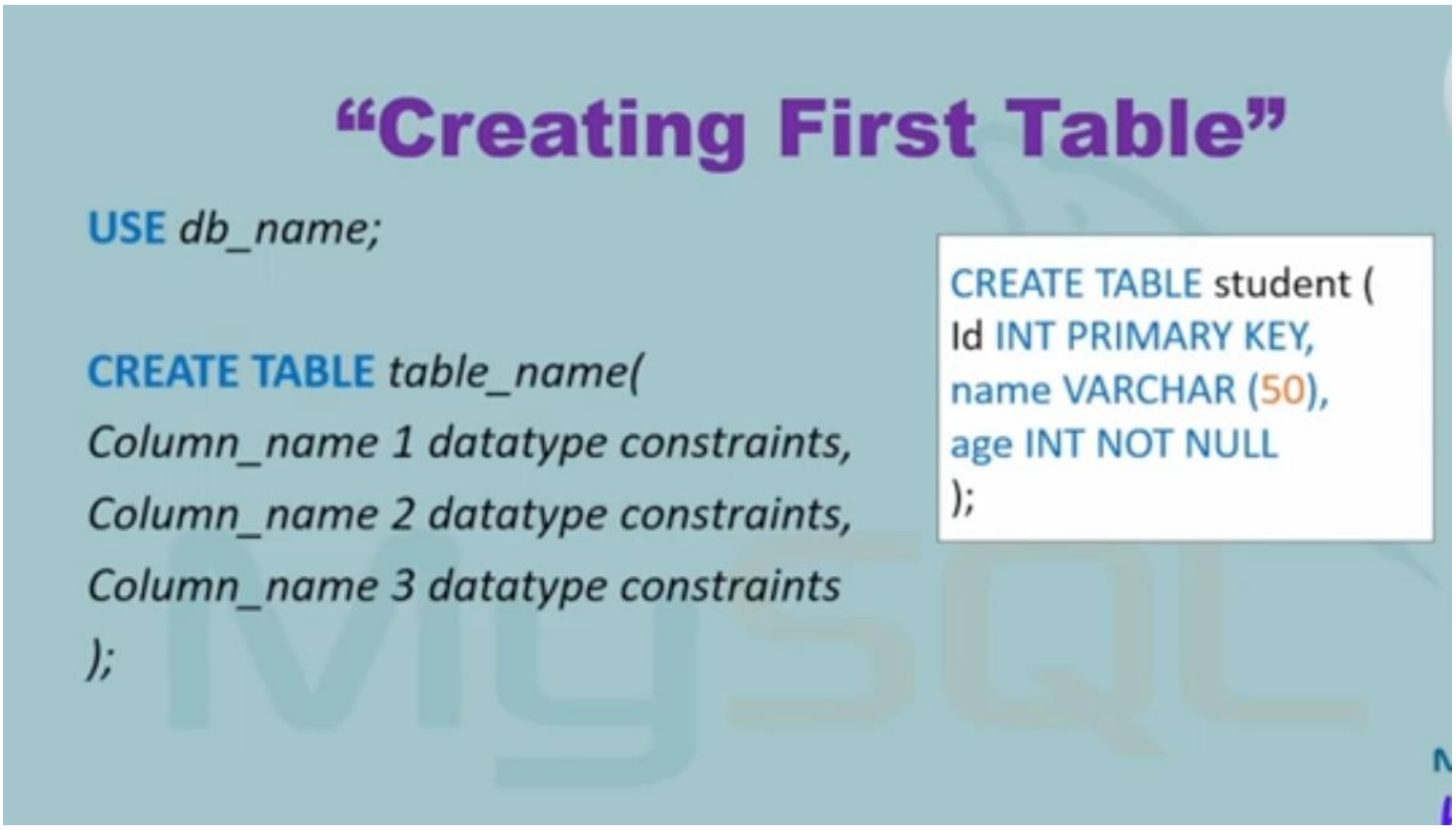


"Creating First Database"

Our First SQL Query

```
CREATE DATABASE db_name;
```

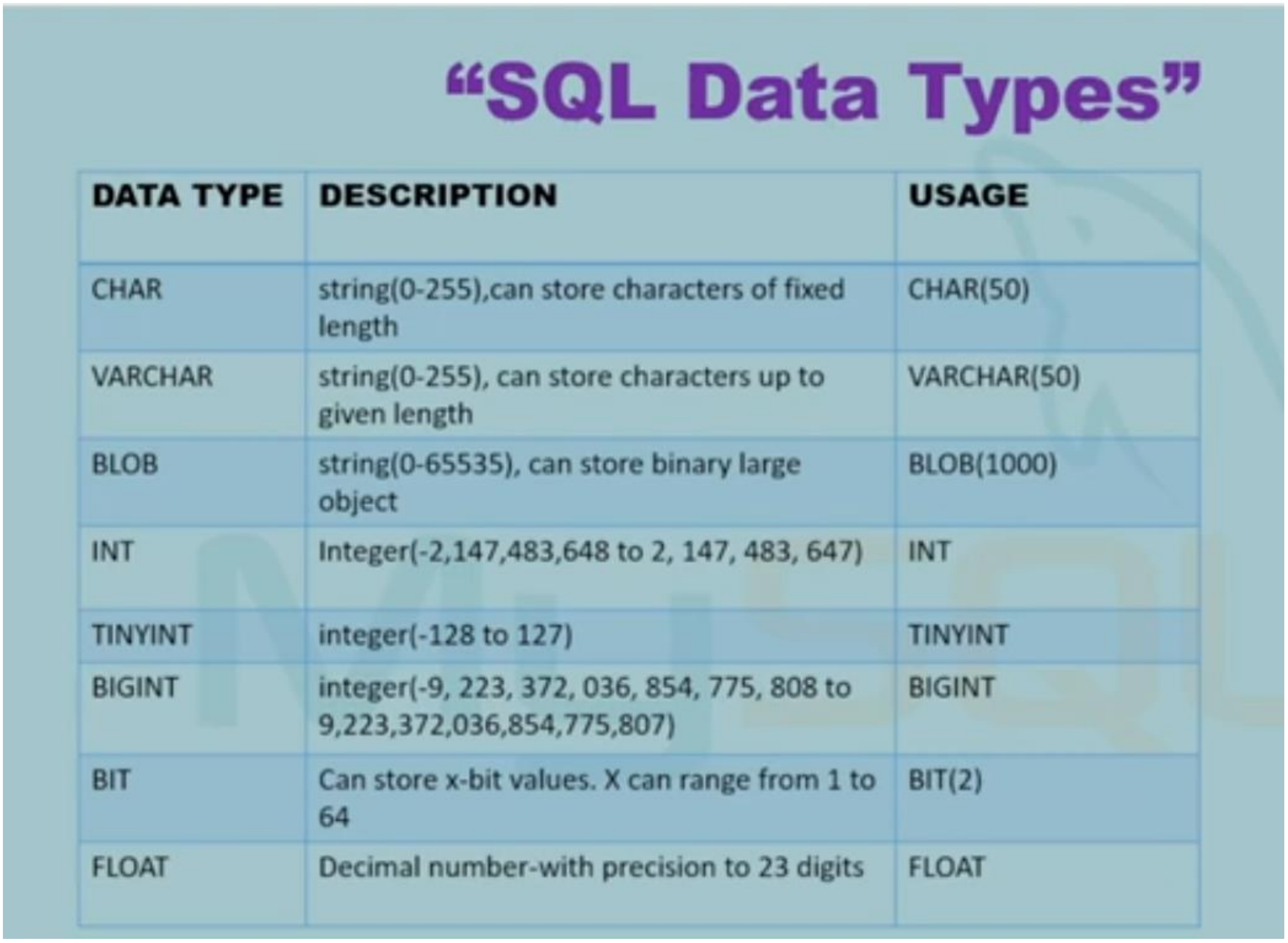
```
DROP DATABASE db_name;
```



“Creating First Table”

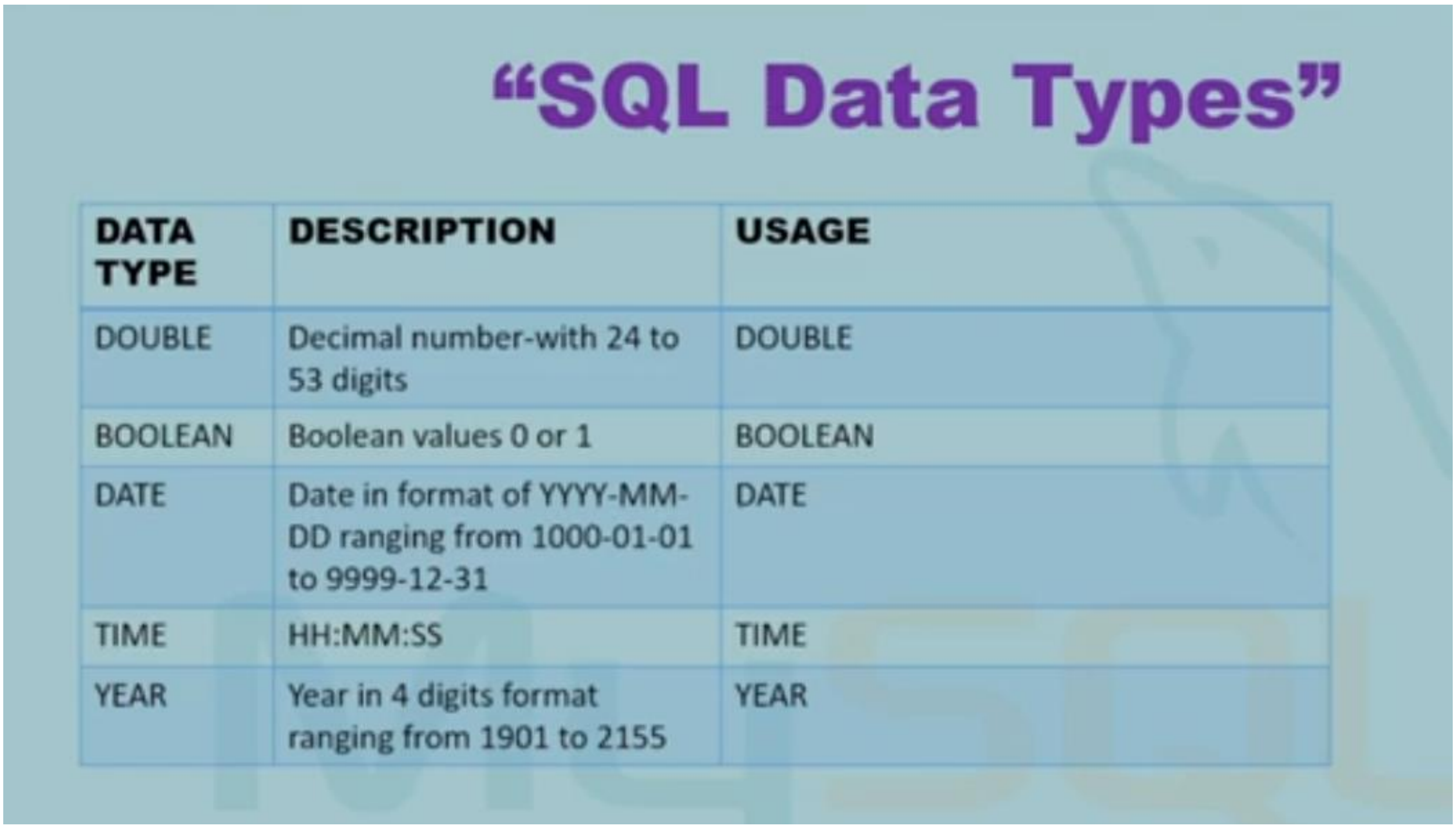
```
USE db_name;

CREATE TABLE table_name(
Column_name 1 datatype constraints,
Column_name 2 datatype constraints,
Column_name 3 datatype constraints
);
```

```
CREATE TABLE student (
Id INT PRIMARY KEY,
name VARCHAR (50),
age INT NOT NULL
);
```



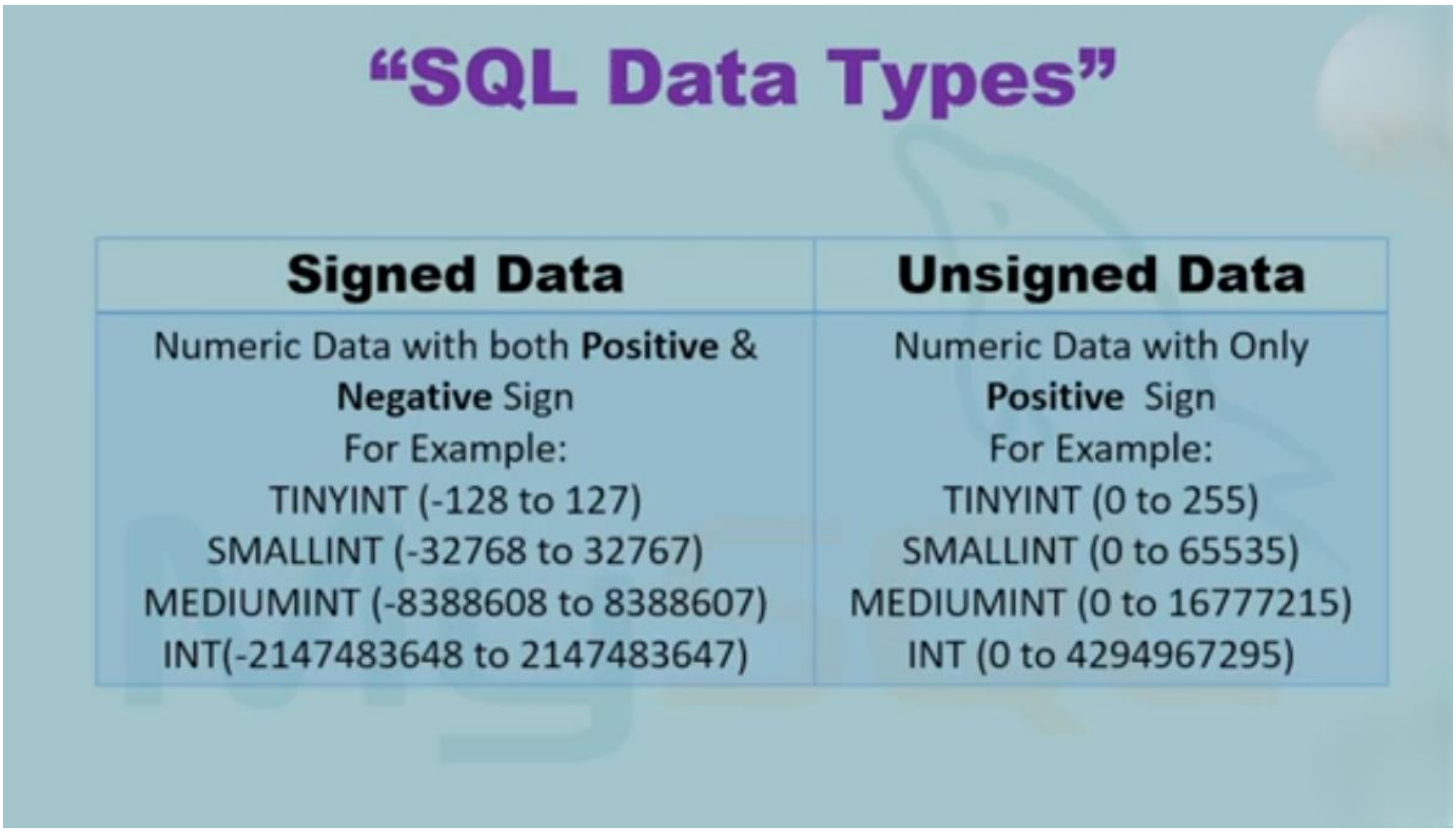
“SQL Data Types”

DATA TYPE	DESCRIPTION	USAGE
CHAR	string(0-255),can store characters of fixed length	CHAR(50)
VARCHAR	string(0-255), can store characters up to given length	VARCHAR(50)
BLOB	string(0-65535), can store binary large object	BLOB(1000)
INT	Integer(-2,147,483,648 to 2, 147, 483, 647)	INT
TINYINT	integer(-128 to 127)	TINYINT
BIGINT	integer(-9, 223, 372, 036, 854, 775, 808 to 9,223,372,036,854,775,807)	BIGINT
BIT	Can store x-bit values. X can range from 1 to 64	BIT(2)
FLOAT	Decimal number-with precision to 23 digits	FLOAT



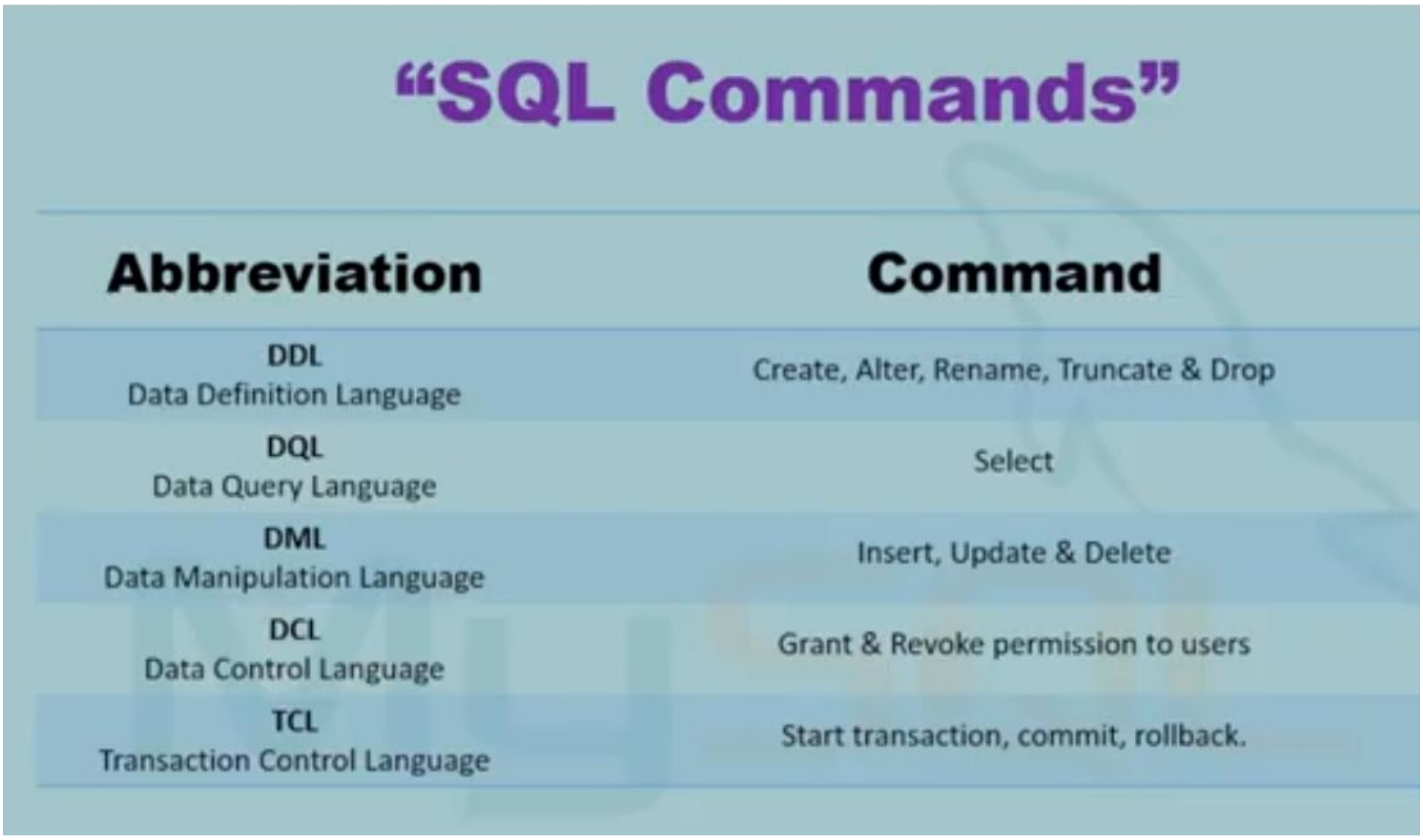
"SQL Data Types"

DATA TYPE	DESCRIPTION	USAGE
DOUBLE	Decimal number-with 24 to 53 digits	DOUBLE
BOOLEAN	Boolean values 0 or 1	BOOLEAN
DATE	Date in format of YYYY-MM-DD ranging from 1000-01-01 to 9999-12-31	DATE
TIME	HH:MM:SS	TIME
YEAR	Year in 4 digits format ranging from 1901 to 2155	YEAR



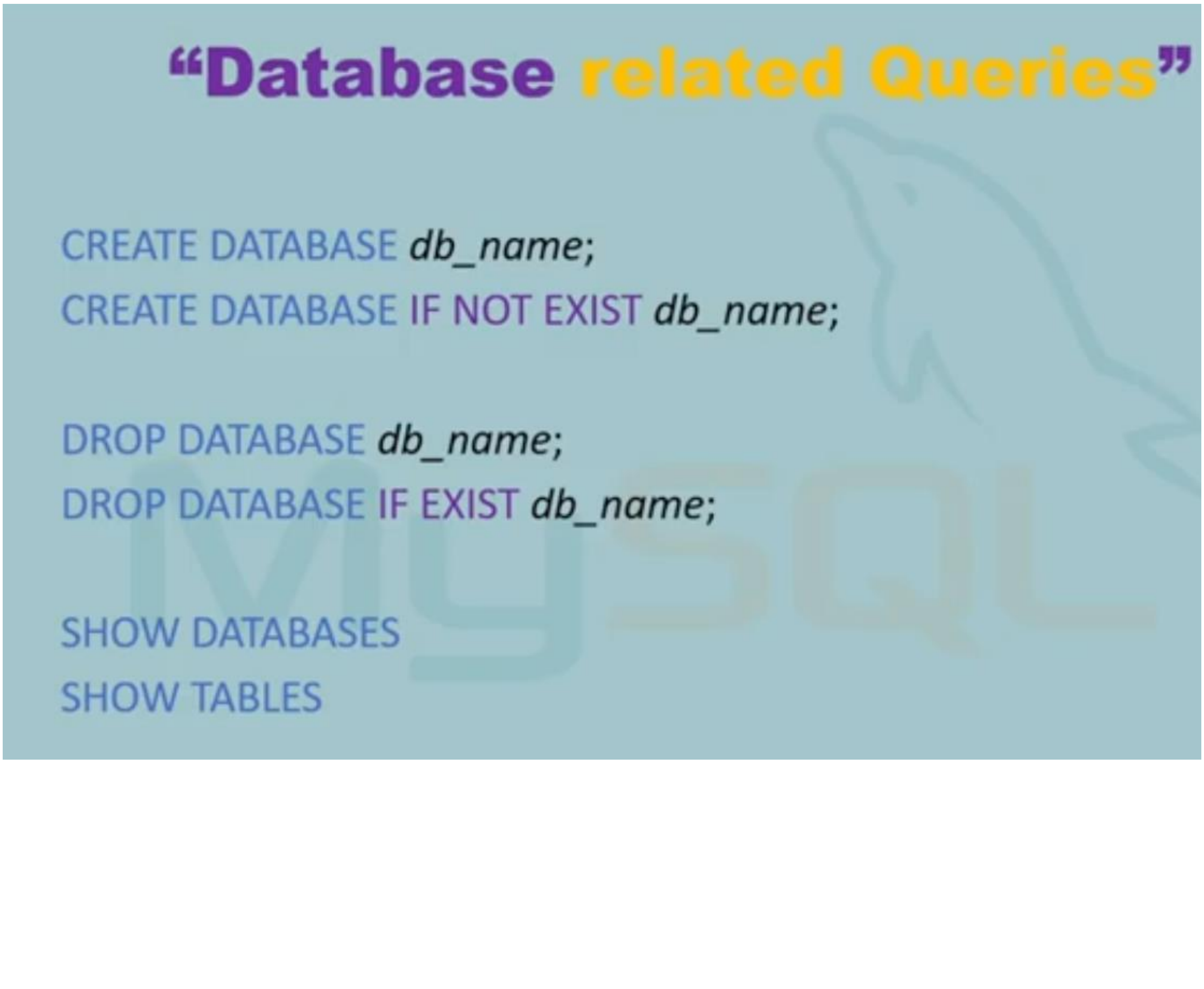
"SQL Data Types"

Signed Data	Unsigned Data
Numeric Data with both **Positive** & **Negative** Sign For Example: TINYINT (-128 to 127) SMALLINT (-32768 to 32767) MEDIUMINT (-8388608 to 8388607) INT(-2147483648 to 2147483647)	Numeric Data with Only **Positive** Sign For Example: TINYINT (0 to 255) SMALLINT (0 to 65535) MEDIUMINT (0 to 16777215) INT (0 to 4294967295)



“SQL Commands”

Abbreviation	Command
DDL Data Definition Language	Create, Alter, Rename, Truncate & Drop
DQL Data Query Language	Select
DML Data Manipulation Language	Insert, Update & Delete
DCL Data Control Language	Grant & Revoke permission to users
TCL Transaction Control Language	Start transaction, commit, rollback.

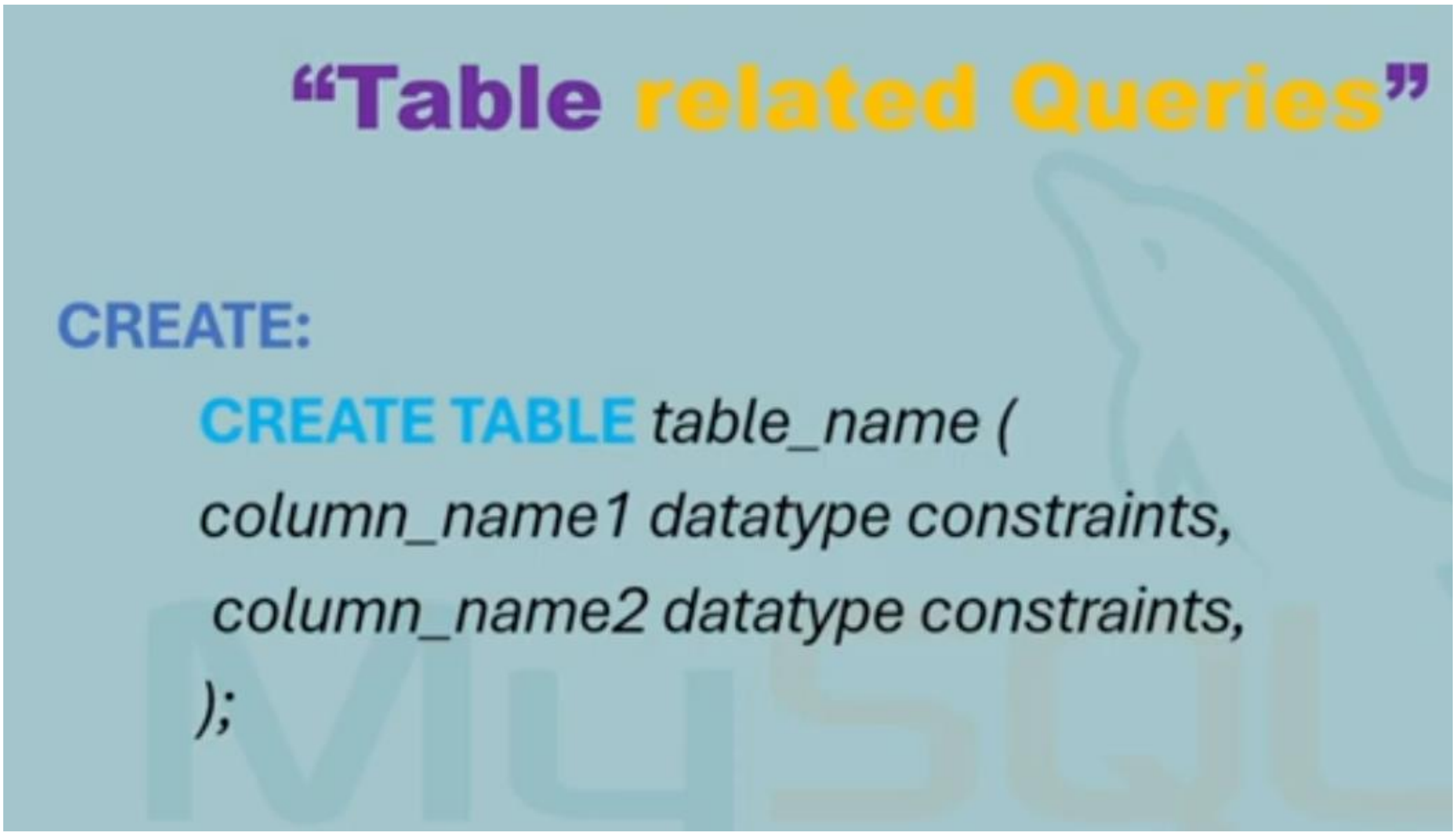


“Database related Queries”

CREATE DATABASE *db_name*;
CREATE DATABASE IF NOT EXIST *db_name*;

DROP DATABASE *db_name*;
DROP DATABASE IF EXIST *db_name*;

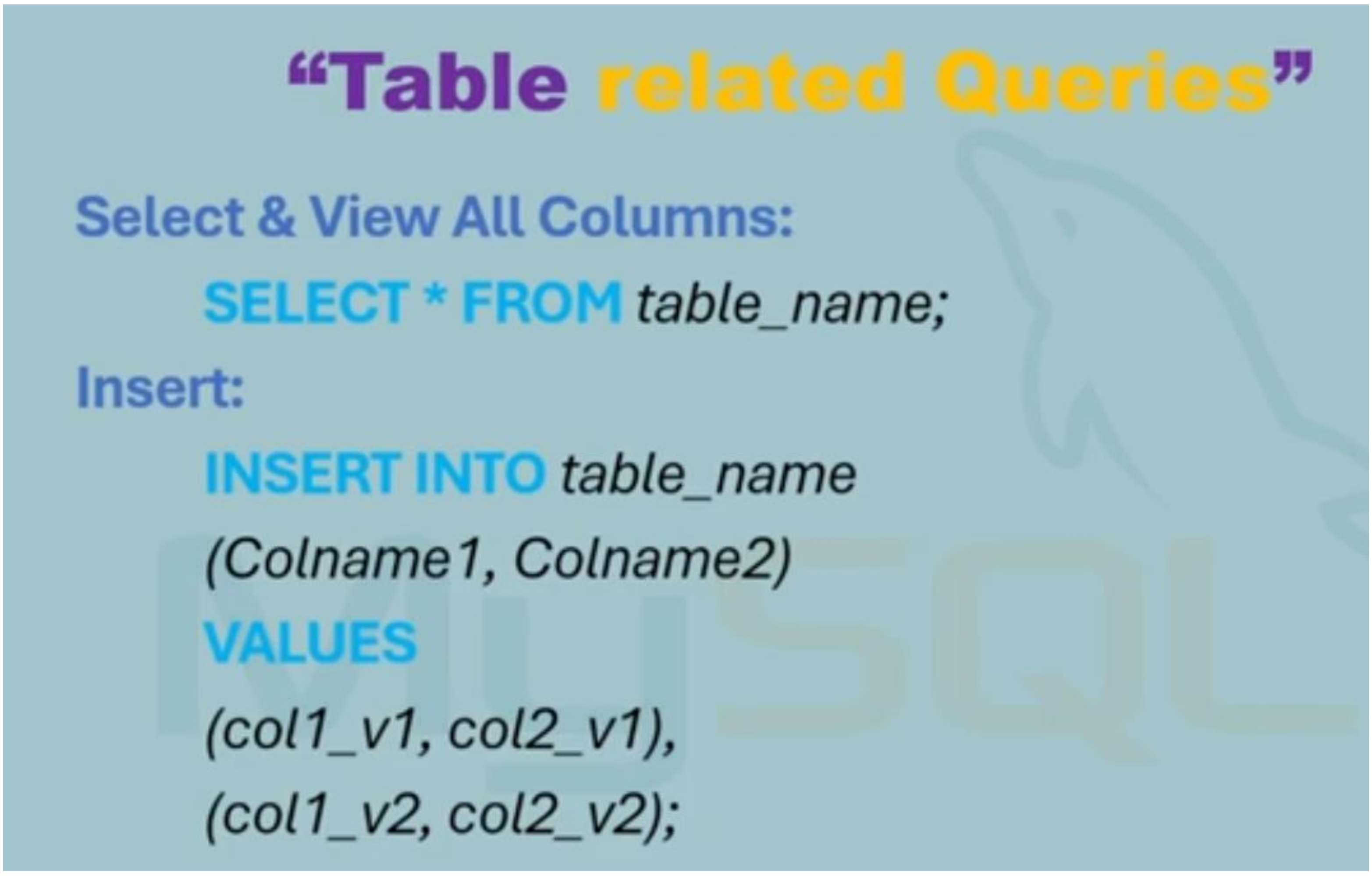
SHOW DATABASES
SHOW TABLES



“Table related Queries”

CREATE:

```
CREATE TABLE table_name (
column_name1 datatype constraints,
 column_name2 datatype constraints,
);
```



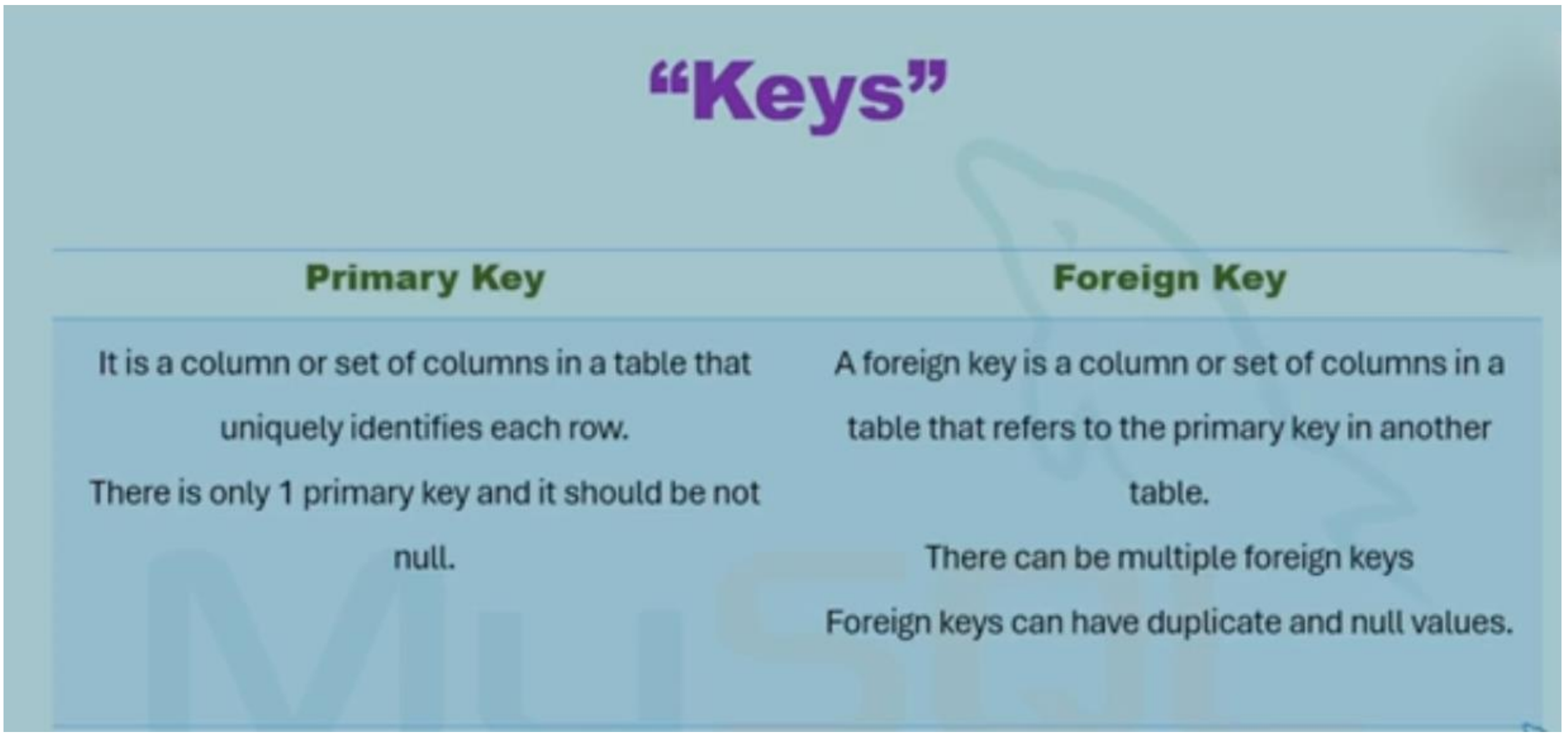
“Table related Queries”

Select & View All Columns:

```
SELECT * FROM table_name;
```

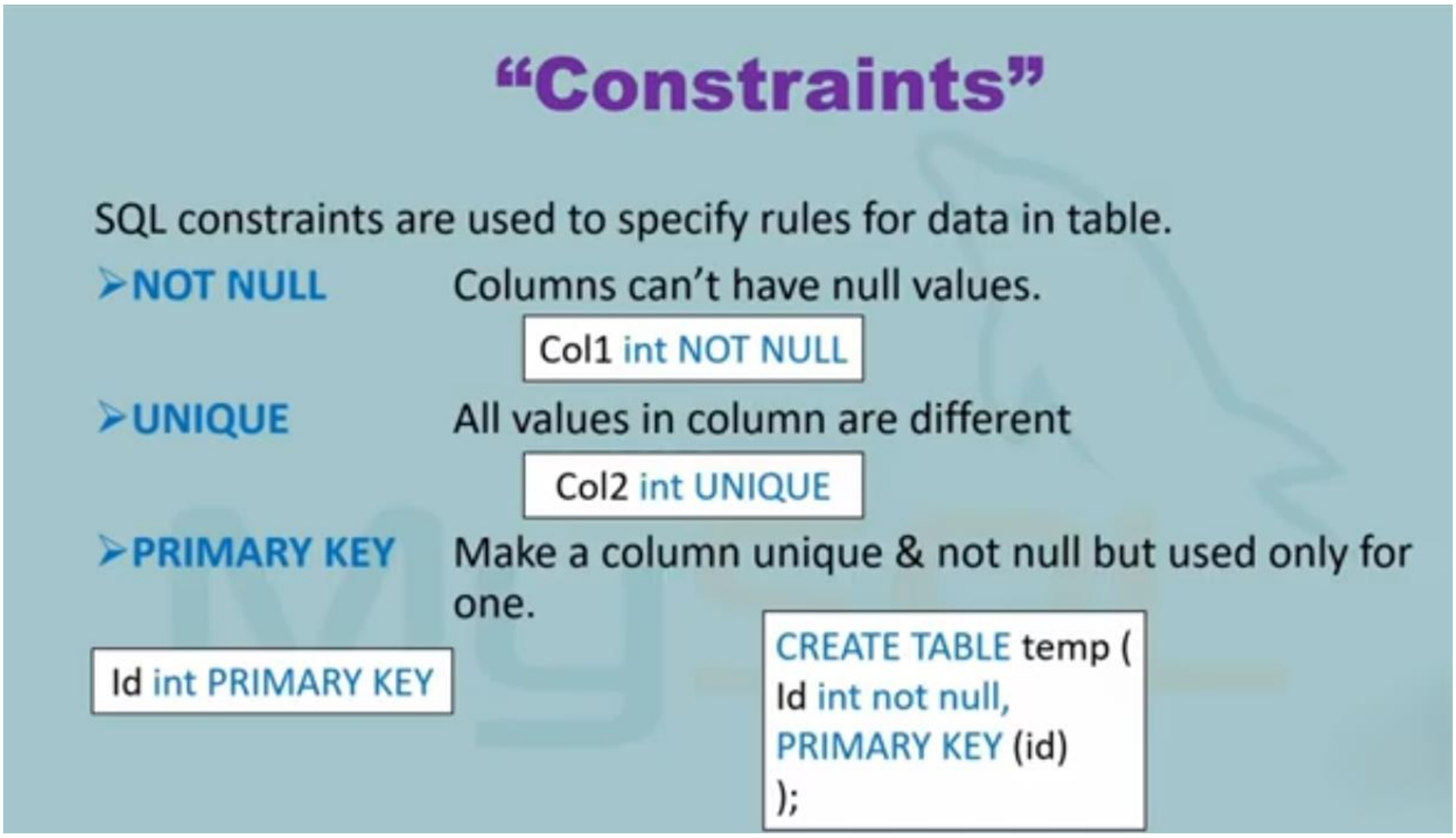
Insert:

```
INSERT INTO table_name
(Colname1, Colname2)
VALUES
(col1_v1, col2_v1),
(col1_v2, col2_v2);
```



"Keys"

Primary Key	Foreign Key
It is a column or set of columns in a table that uniquely identifies each row. There is only 1 primary key and it should be not null.	A foreign key is a column or set of columns in a table that refers to the primary key in another table. There can be multiple foreign keys Foreign keys can have duplicate and null values.



"Constraints"

SQL constraints are used to specify rules for data in table.

- **NOT NULL** Columns can't have null values.

```
Col1 int NOT NULL
```

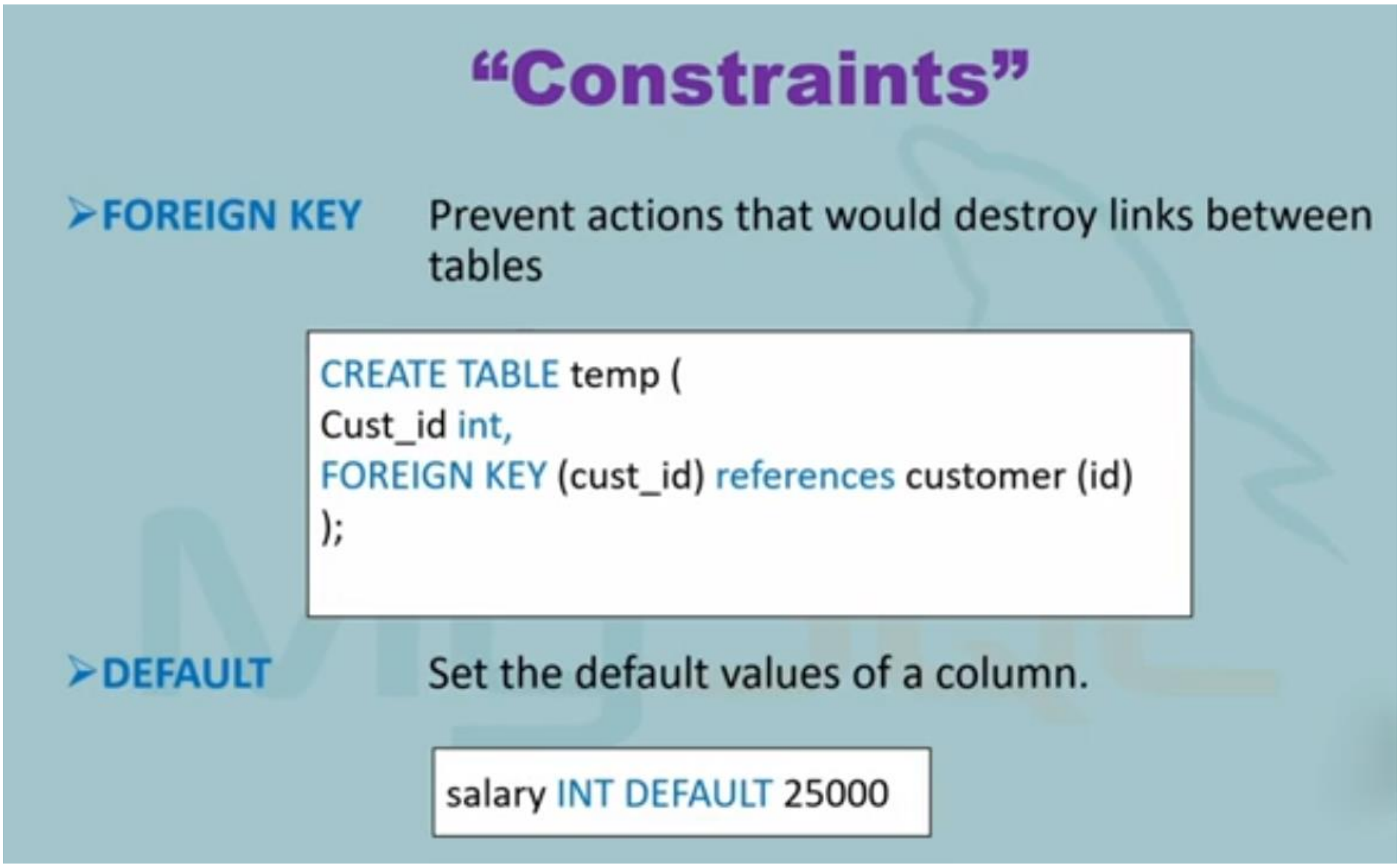
- **UNIQUE** All values in column are different

```
Col2 int UNIQUE
```

- **PRIMARY KEY** Make a column unique & not null but used only for one.

```
Id int PRIMARY KEY
```

```
CREATE TABLE temp (
Id int not null,
PRIMARY KEY (id)
);
```



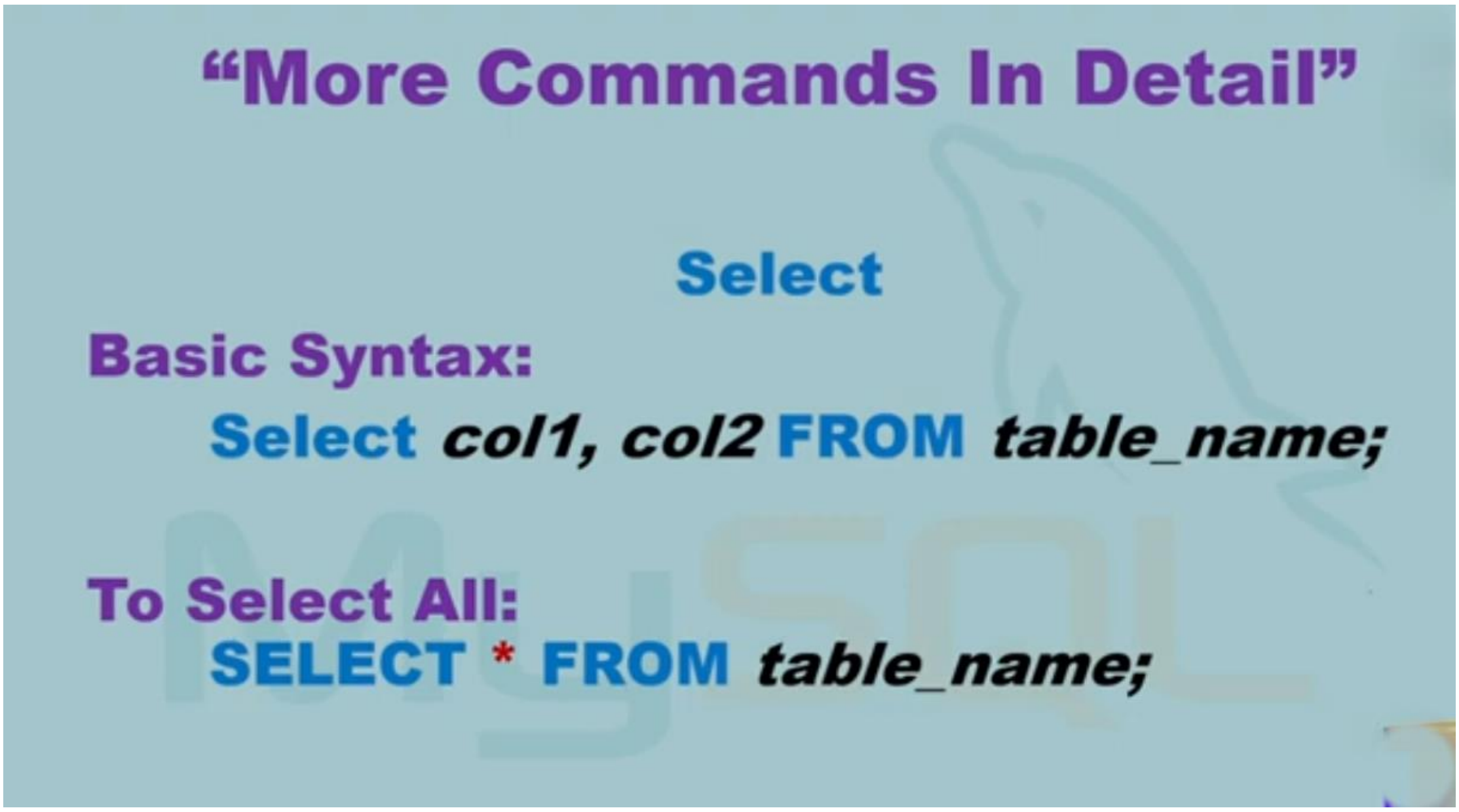
"Constraints"

- **FOREIGN KEY** Prevent actions that would destroy links between tables

```
CREATE TABLE temp (
Cust_id int,
FOREIGN KEY (cust_id) references customer (id)
);
```

- **DEFAULT** Set the default values of a column.

```
salary INT DEFAULT 25000
```



"More Commands In Detail"

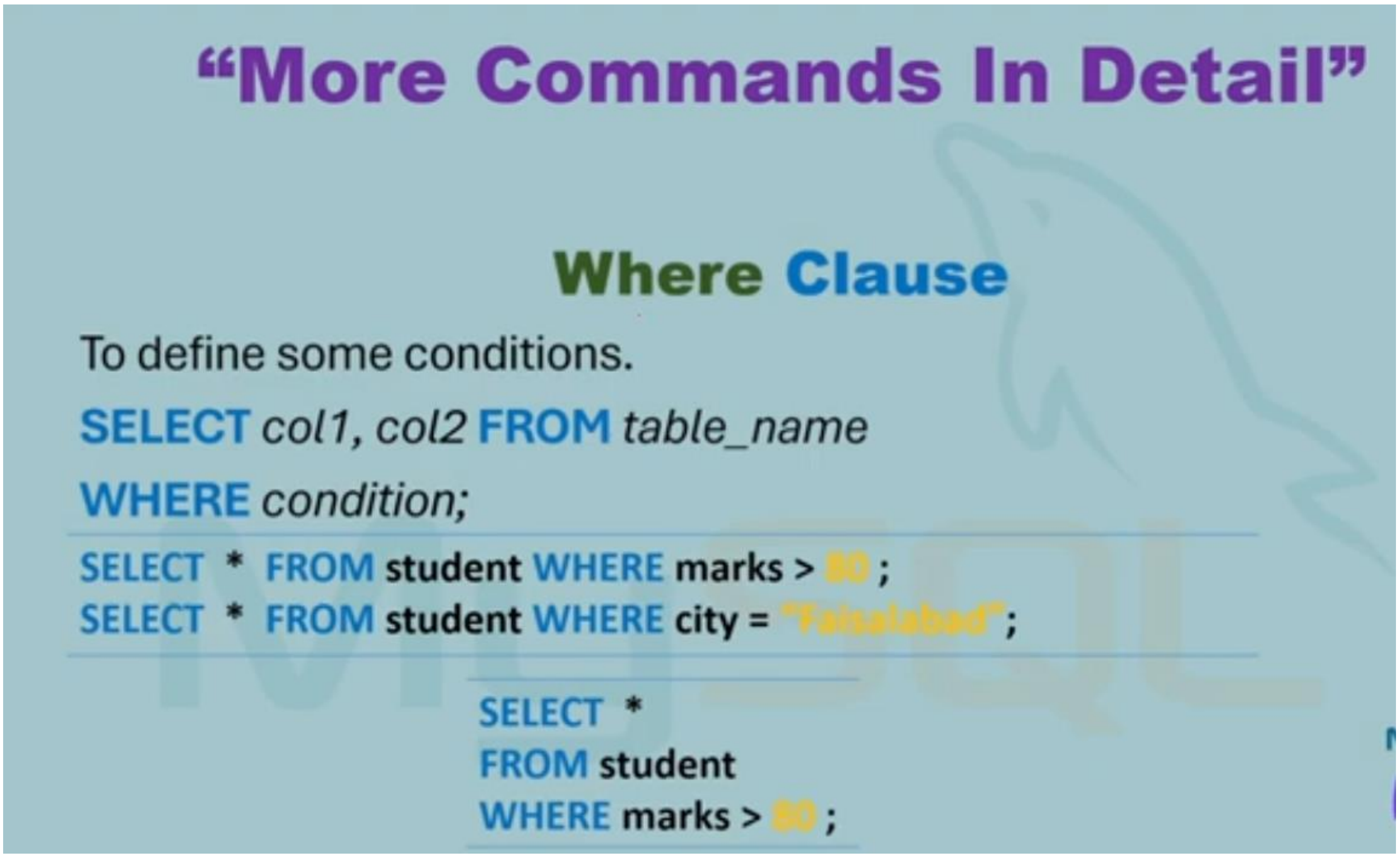
Select

Basic Syntax:

Select ***col1, col2*** **FROM** ***table_name;***

To Select All:

SELECT * FROM ***table_name;***



“More Commands In Detail”

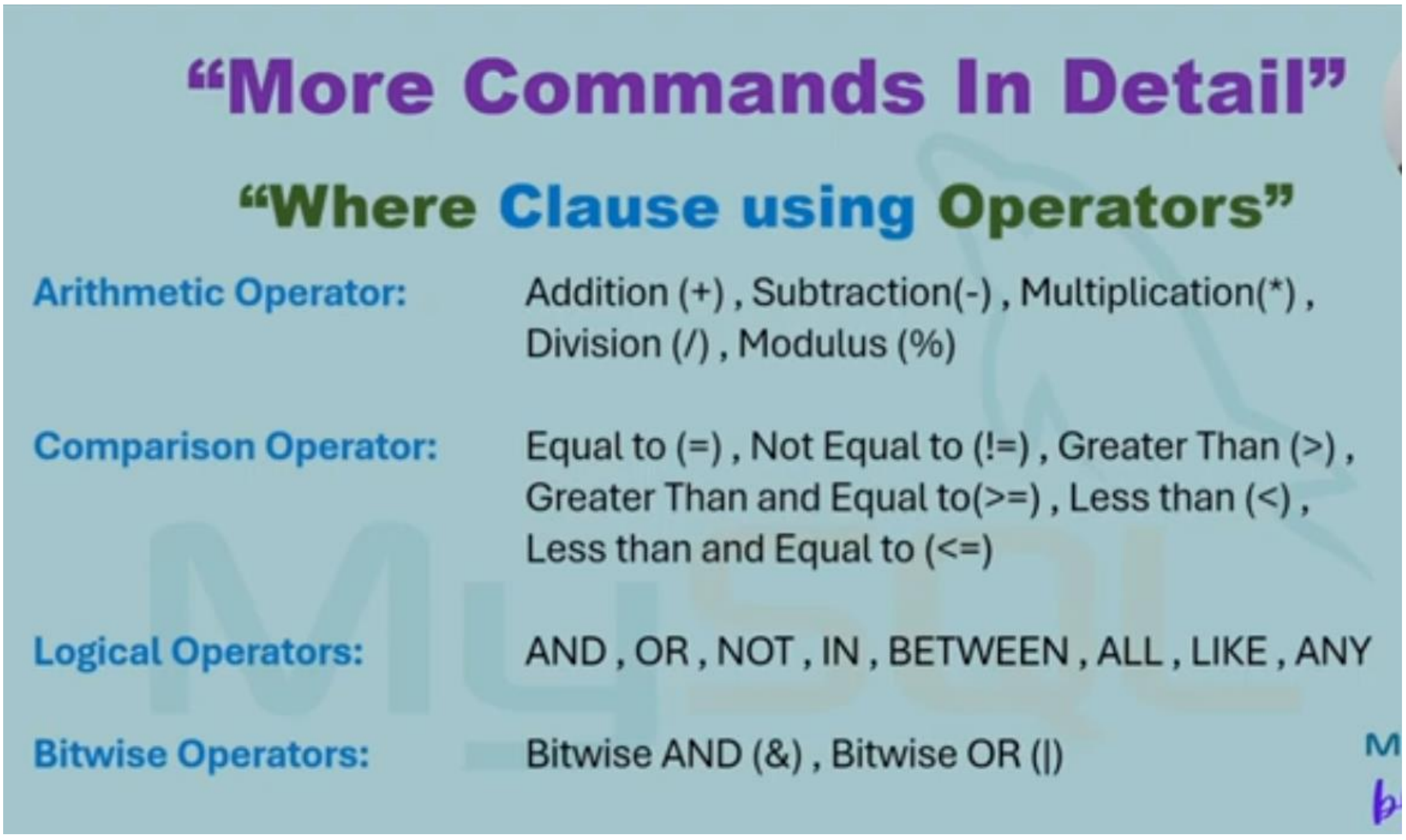
Where Clause

To define some conditions.

```
SELECT col1, col2 FROM table_name
WHERE condition;
```

```
SELECT * FROM student WHERE marks > 80 ;
SELECT * FROM student WHERE city = "Faisalabad";
```

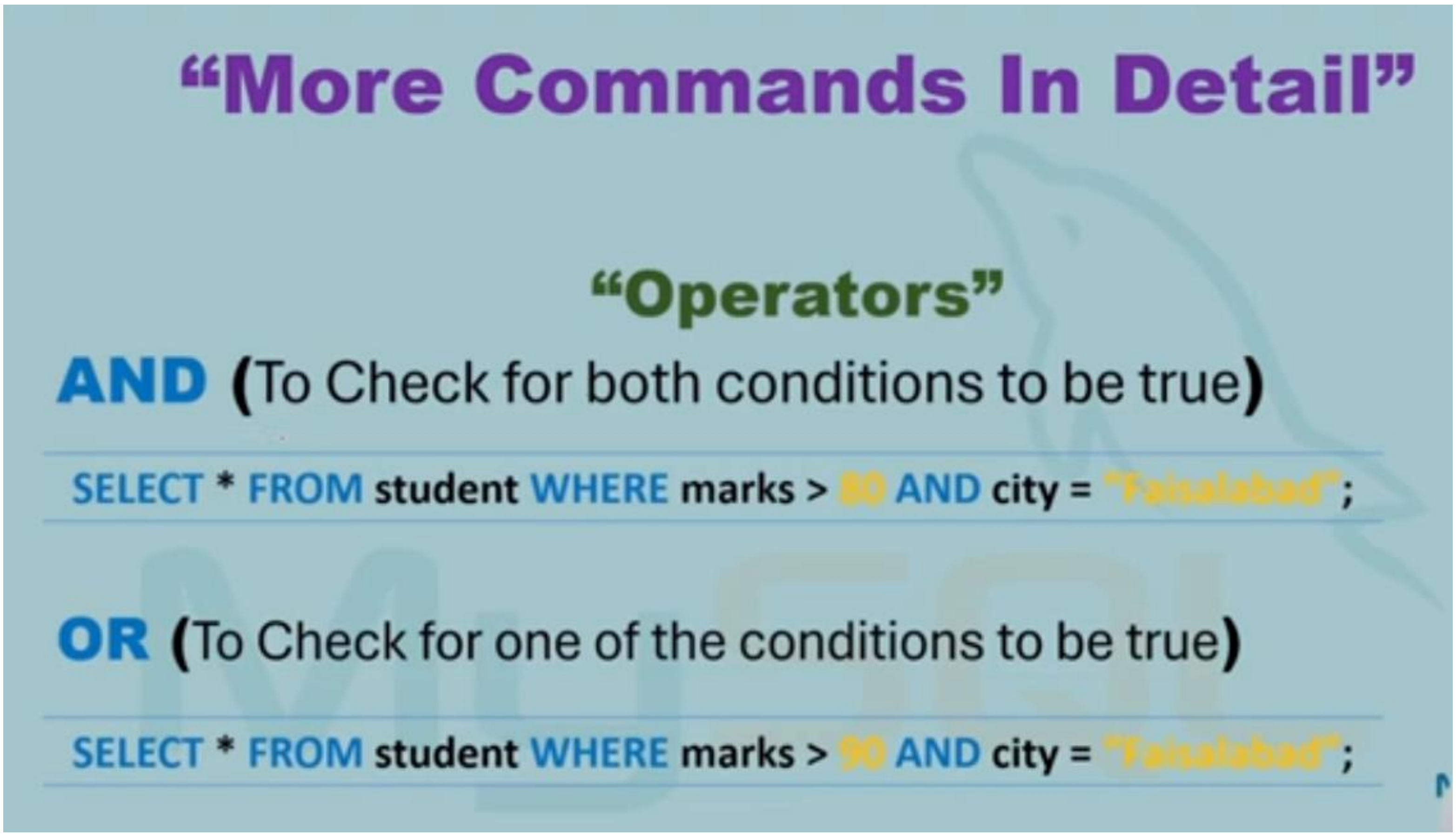
```
SELECT *
FROM student
WHERE marks > 80 ;
```



“More Commands In Detail”

“Where Clause using Operators”

Arithmetic Operator:	Addition (+) , Subtraction(-) , Multiplication(*) , Division (/) , Modulus (%)
Comparison Operator:	Equal to (=) , Not Equal to (!=) , Greater Than (>) , Greater Than and Equal to(>=) , Less than (<) , Less than and Equal to (<=)
Logical Operators:	AND , OR , NOT , IN , BETWEEN , ALL , LIKE , ANY
Bitwise Operators:	Bitwise AND (&) , Bitwise OR (|)



“More Commands In Detail”

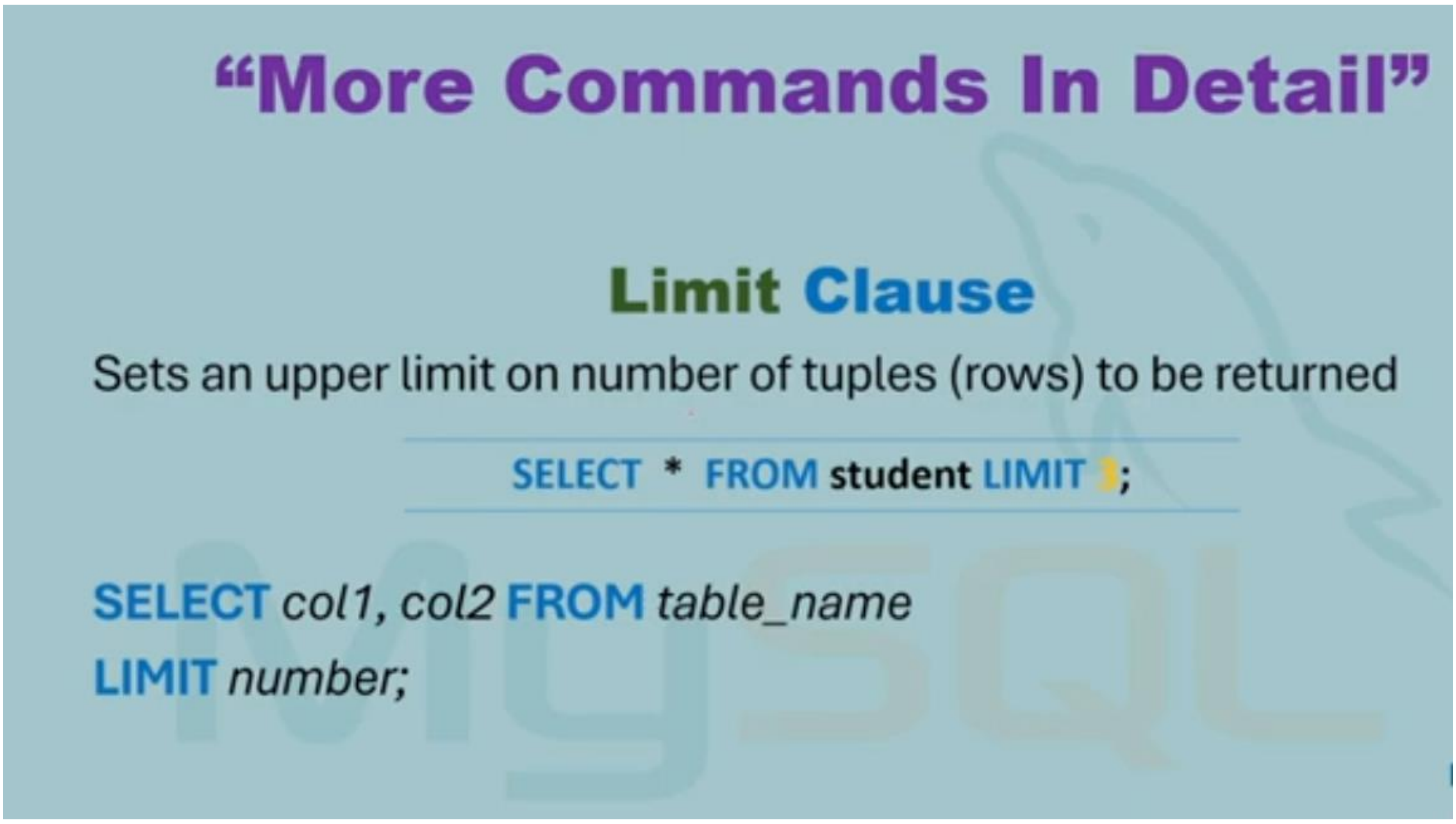
“Operators”

AND (To Check for both conditions to be true)

```
SELECT * FROM student WHERE marks > 80 AND city = "Faisalabad";
```

OR (To Check for one of the conditions to be true)

```
SELECT * FROM student WHERE marks > 90 AND city = "Faisalabad";
```



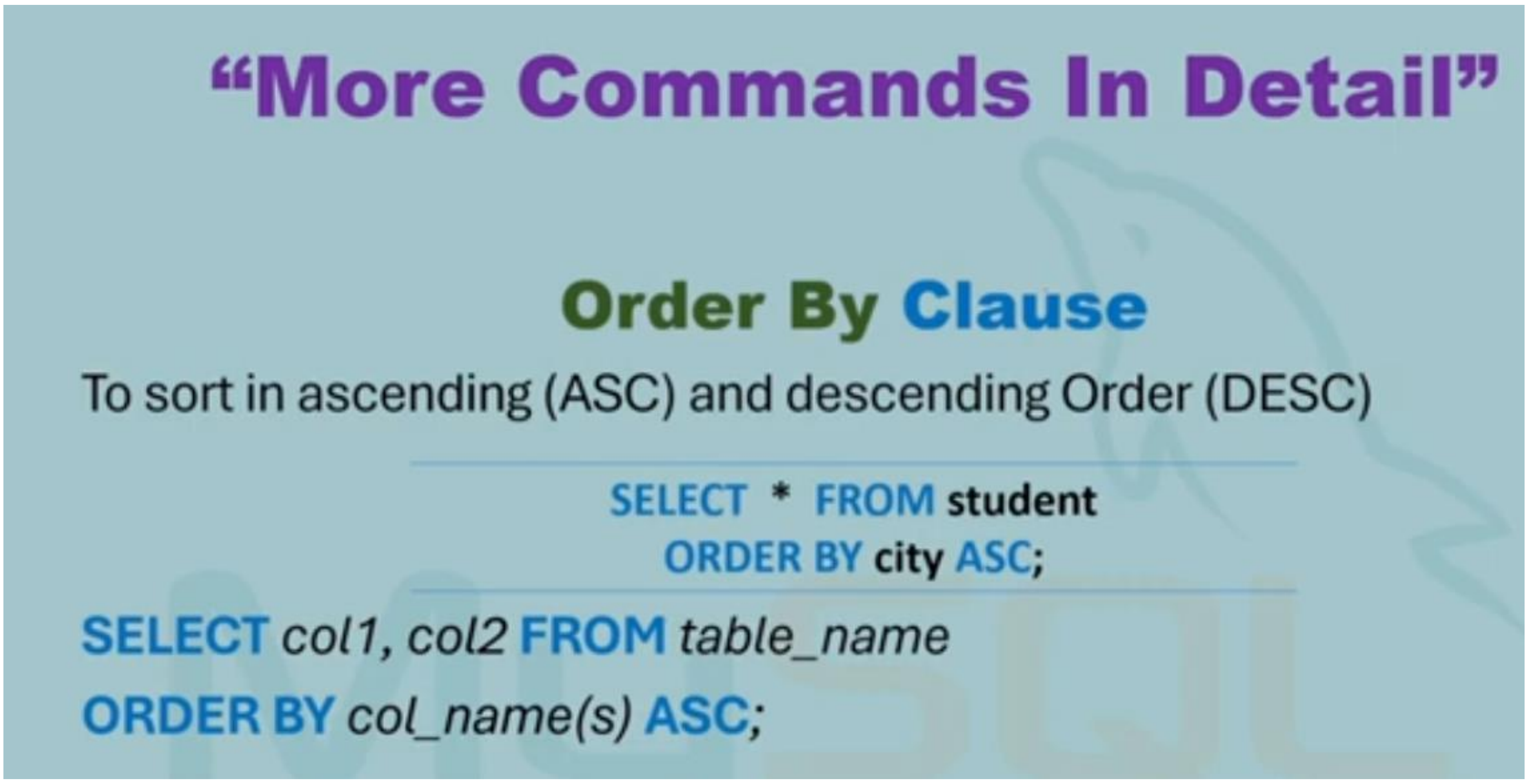
“More Commands In Detail”

Limit Clause

Sets an upper limit on number of tuples (rows) to be returned

```
SELECT * FROM student LIMIT 3;
```

```
SELECT col1, col2 FROM table_name
LIMIT number;
```



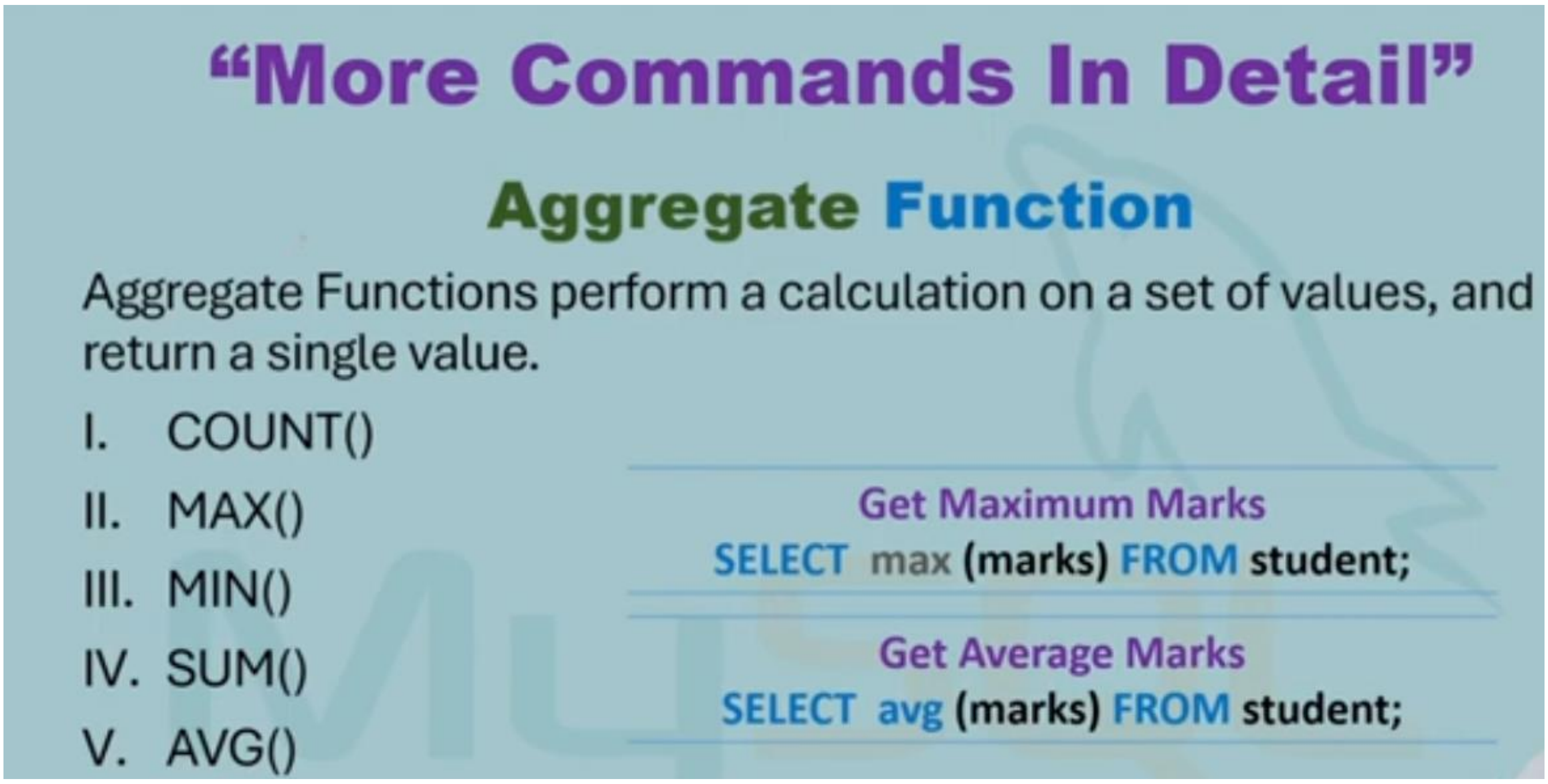
"More Commands In Detail"

Order By Clause

To sort in ascending (ASC) and descending Order (DESC)

```
SELECT * FROM student
ORDER BY city ASC;
```

```
SELECT col1, col2 FROM table_name
ORDER BY col_name(s) ASC;
```



"More Commands In Detail"

Aggregate Function

Aggregate Functions perform a calculation on a set of values, and return a single value.

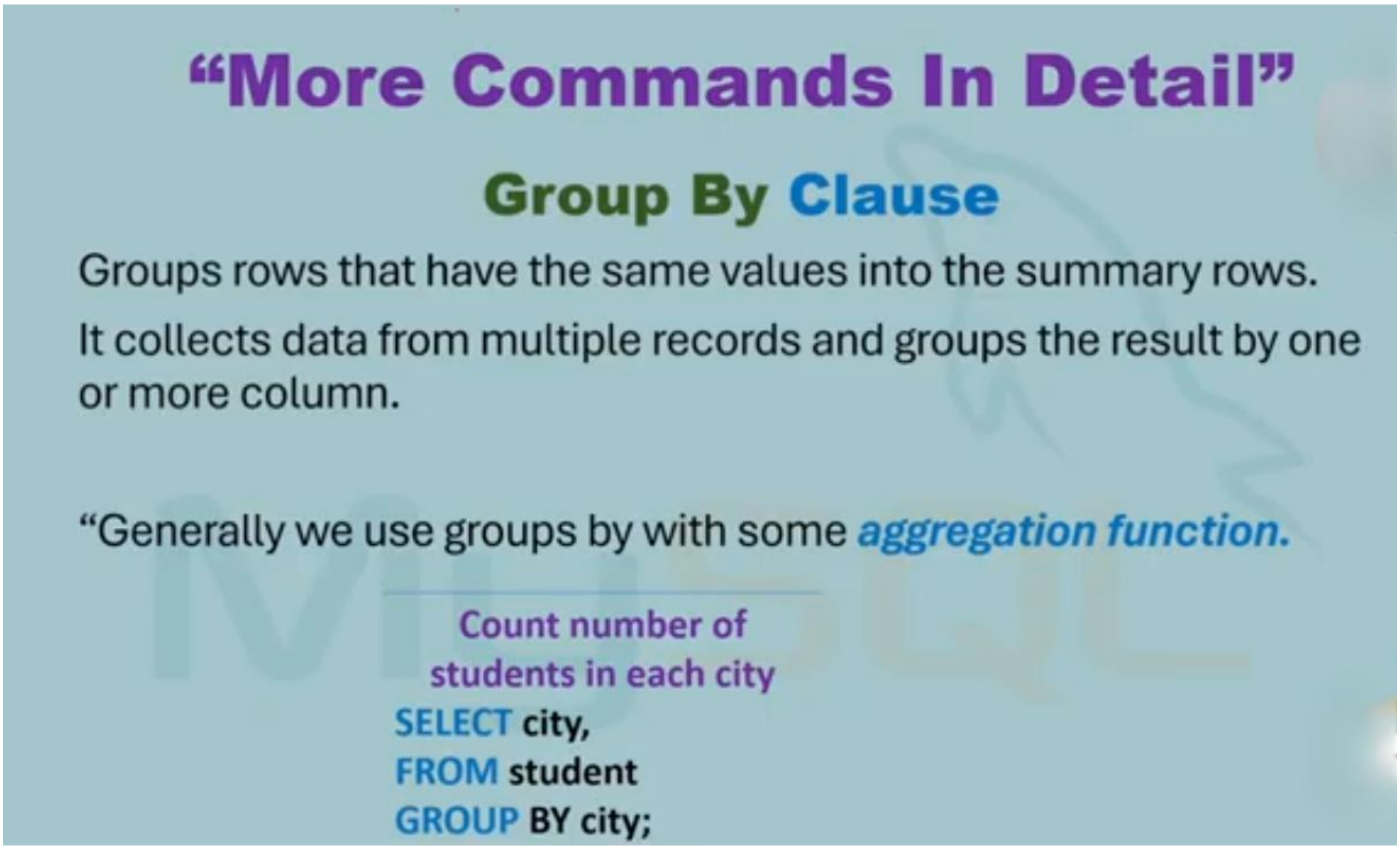
I. COUNT()
II. MAX()
III. MIN()
IV. SUM()
V. AVG()

Get Maximum Marks

```
SELECT max (marks) FROM student;
```

Get Average Marks

```
SELECT avg (marks) FROM student;
```



“More Commands In Detail”

Group By Clause

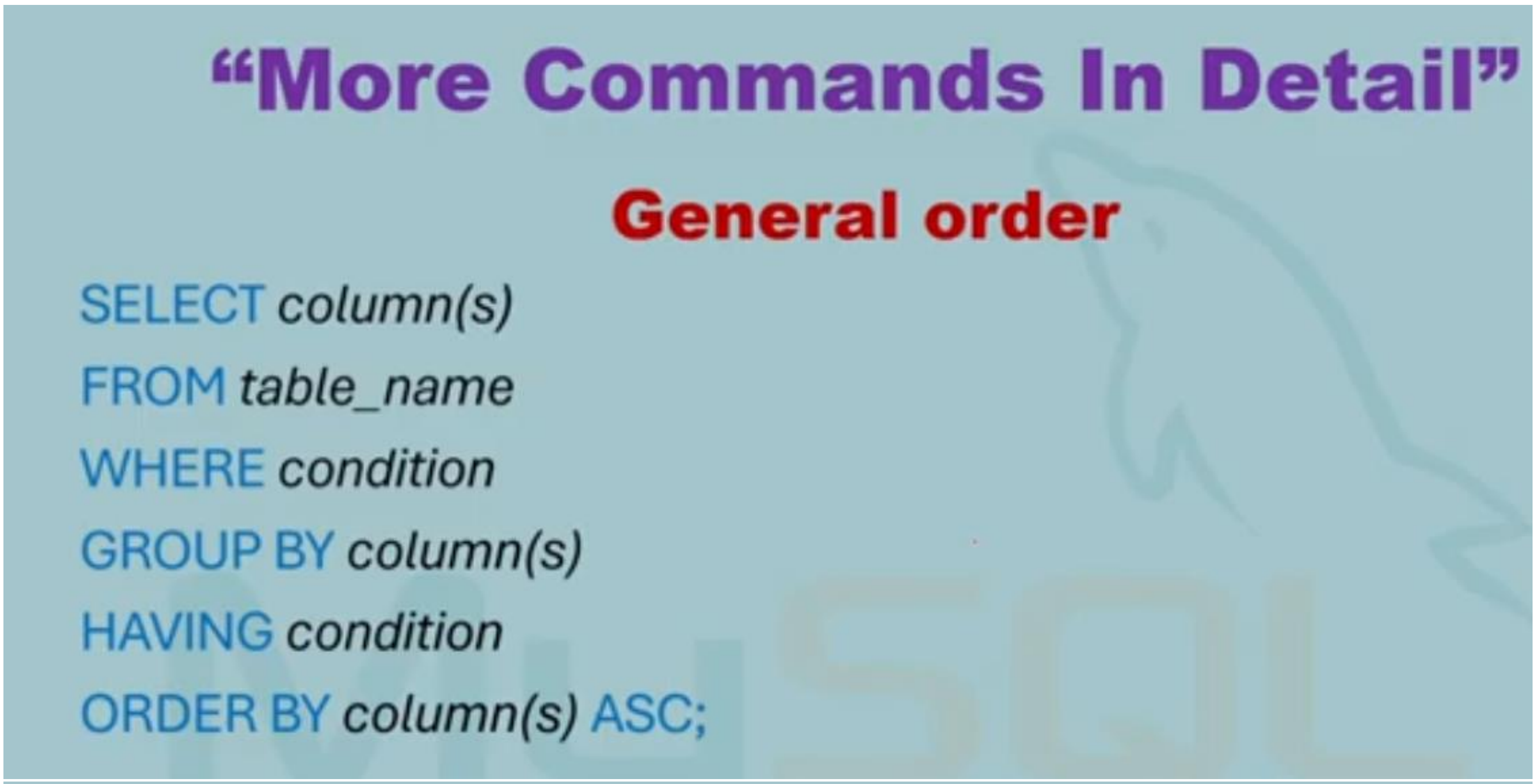
Groups rows that have the same values into the summary rows.

It collects data from multiple records and groups the result by one or more column.

“Generally we use groups by with some *aggregation function.*

Count number of students in each city

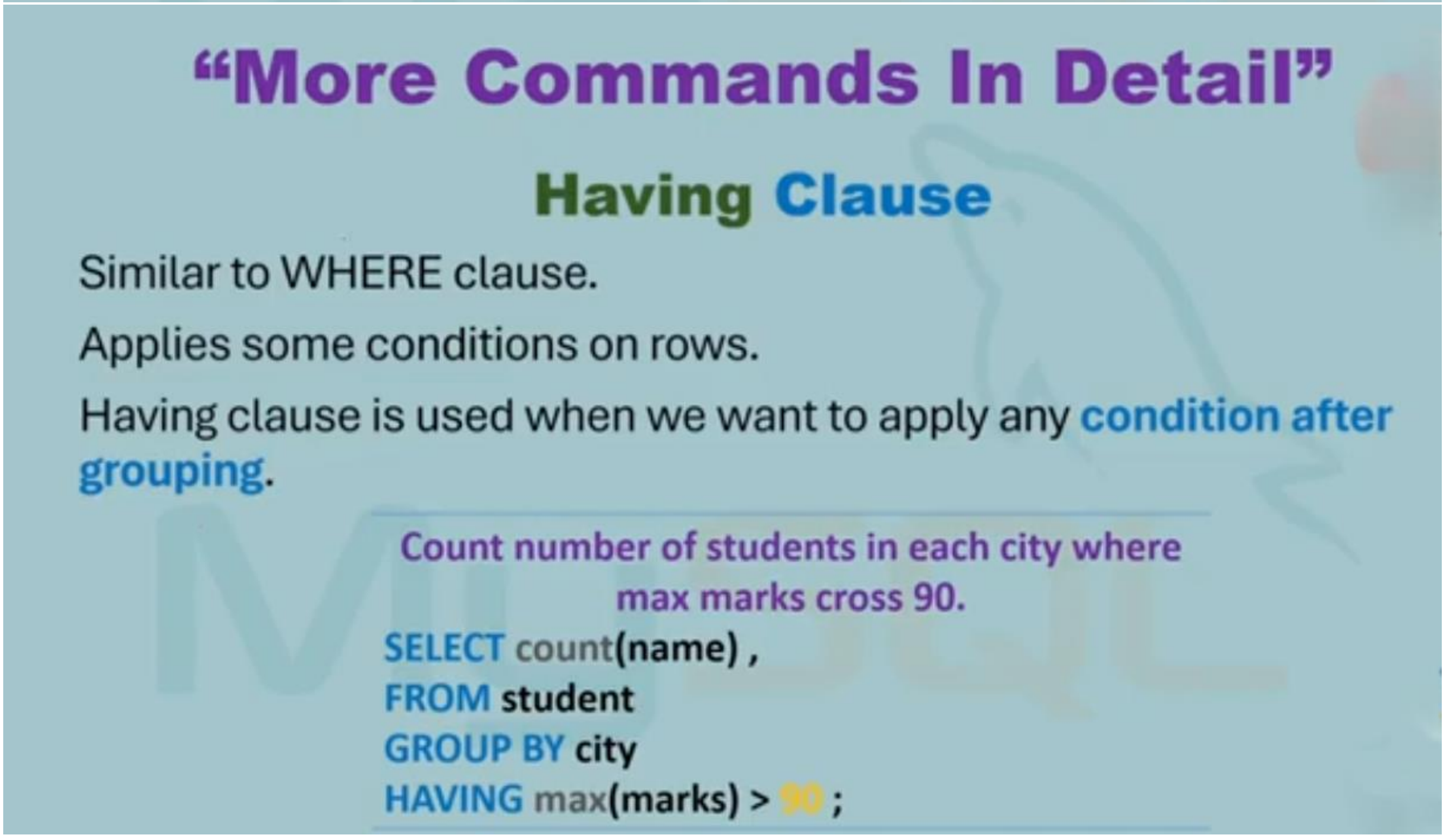
```
SELECT city,
FROM student
GROUP BY city;
```



"More Commands In Detail"

General order

```
SELECT column(s)
FROM table_name
WHERE condition
GROUP BY column(s)
HAVING condition
ORDER BY column(s) ASC;
```



"More Commands In Detail"

Having Clause

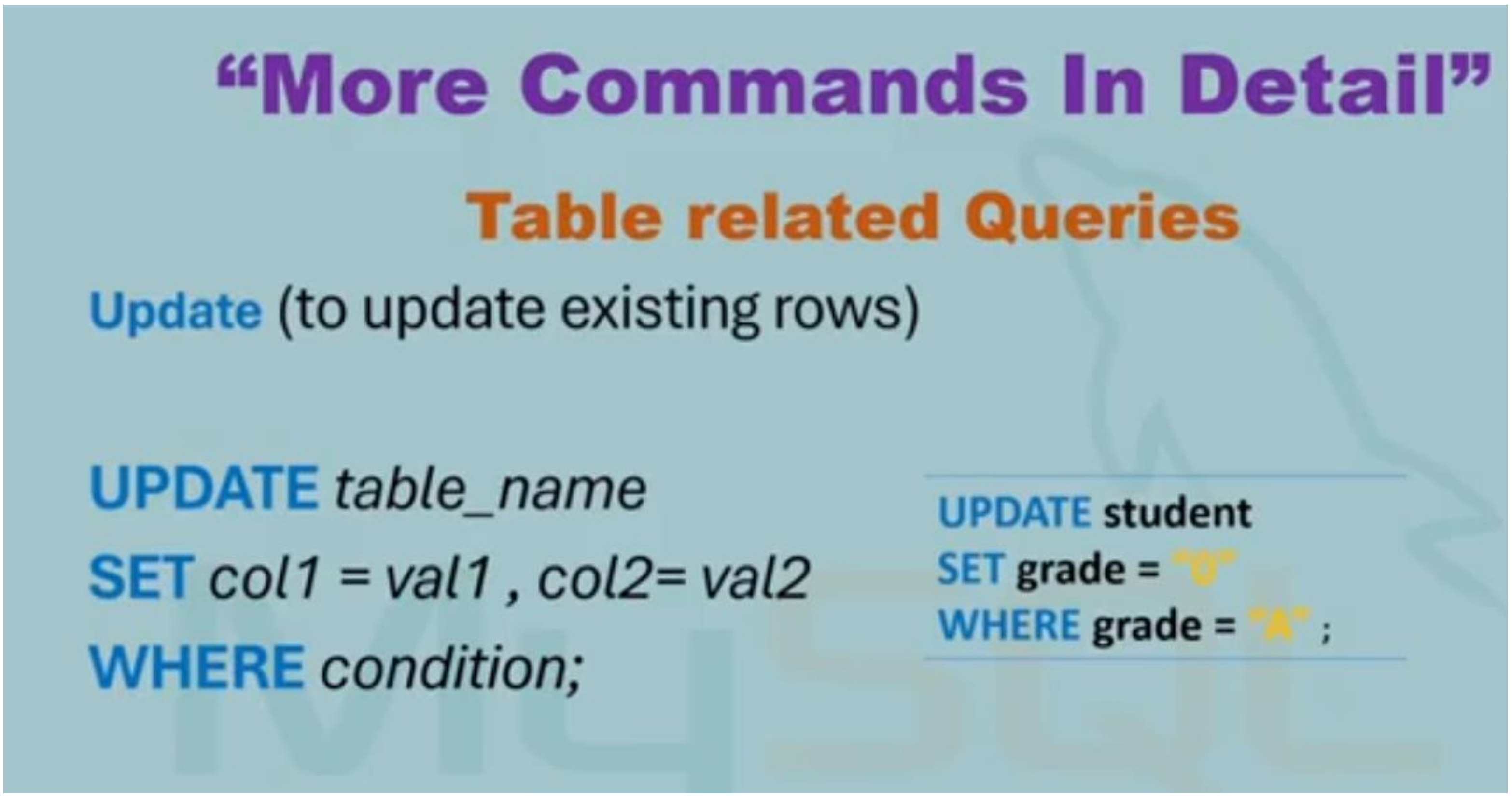
Similar to WHERE clause.

Applies some conditions on rows.

Having clause is used when we want to apply any **condition after grouping**.

Count number of students in each city where max marks cross 90.

```
SELECT count(name) ,
FROM student
GROUP BY city
HAVING max(marks) > 90 ;
```



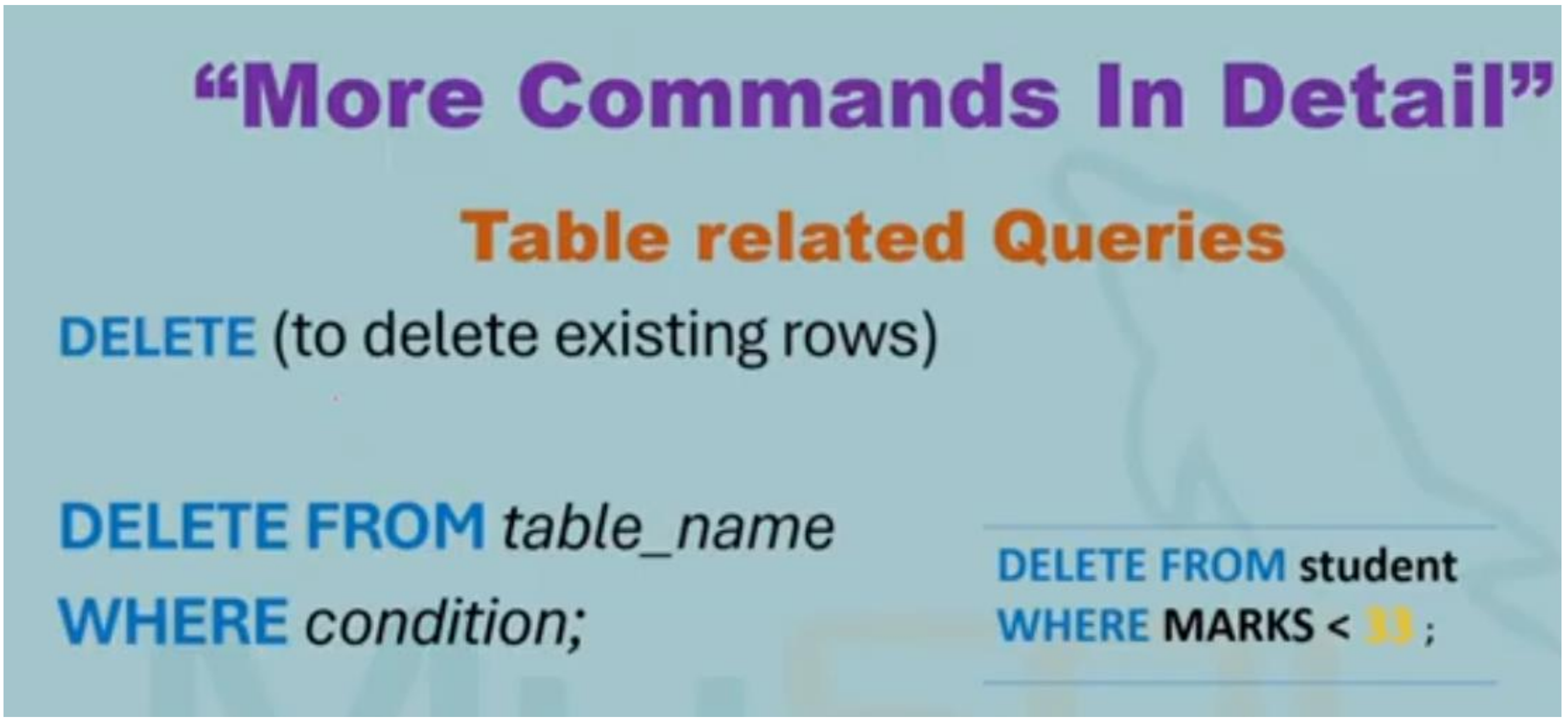
“More Commands In Detail”

Table related Queries

Update (to update existing rows)

```
UPDATE table_name
SET col1 = val1 , col2= val2
WHERE condition;
```

```
UPDATE student
SET grade = "O"
WHERE grade = "A" ;
```



“More Commands In Detail”

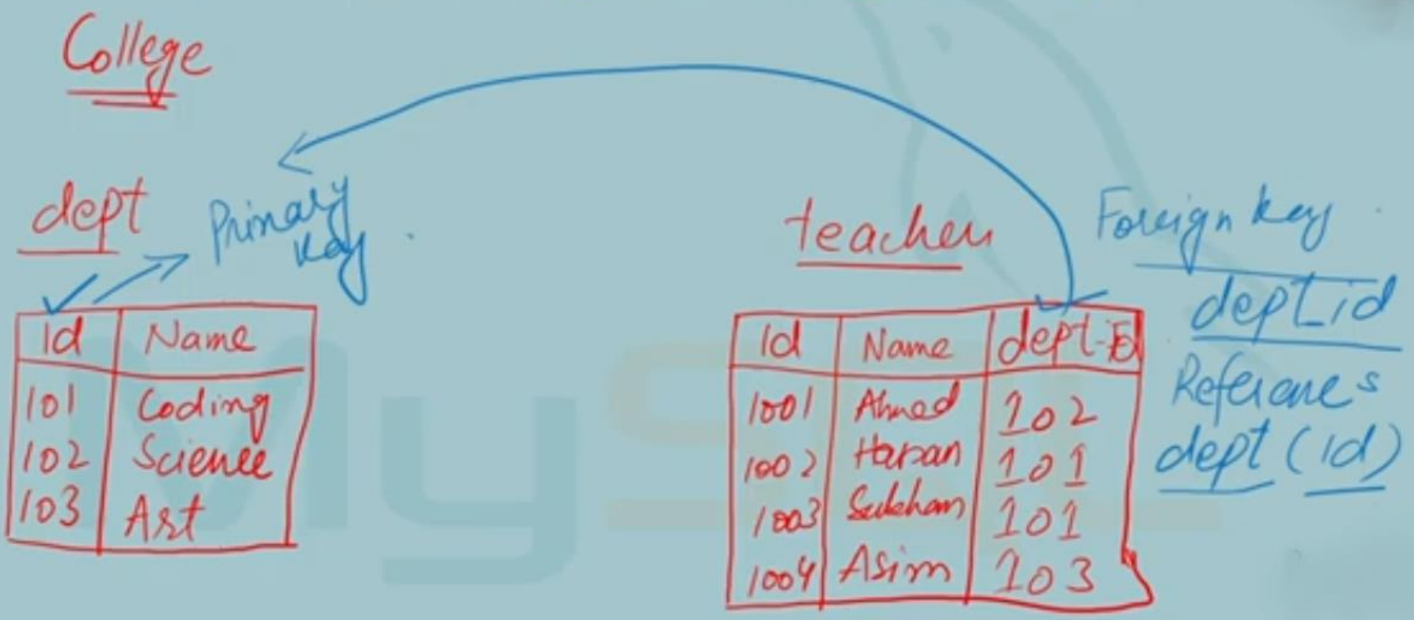
Table related Queries

DELETE (to delete existing rows)

```
DELETE FROM table_name
WHERE condition;
```

```
DELETE FROM student
WHERE MARKS < 33 ;
```

“Revisiting Foreign keys”



Id	Name
101	Coding
102	Science
103	Art

Id	Name	dept-Id
1001	Ahmed	102
1002	Hassan	101
1003	Subhan	101
1004	Asim	103

“More Commands In Detail”

Table Related Queries

Cascading for FK(Foreign Key)

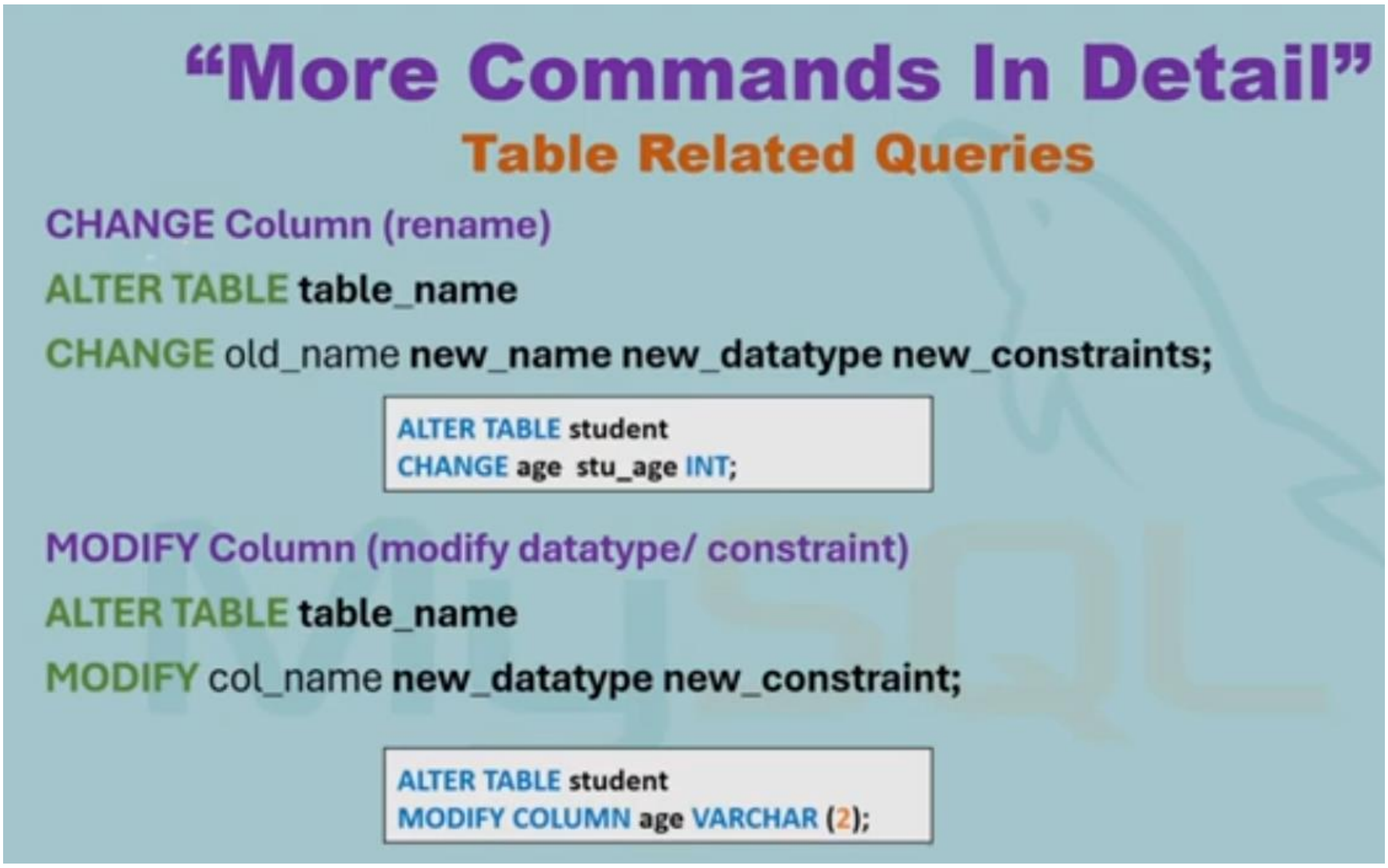
On Delete Cascade:

When we create a FK using this option , it deletes the referencing row in the child table when the refenced row is deleted in the parent table which has a primary key.

On Update Cascade:

When we create a FK using UPDATE CASCADE the referencing rows are updated in the child table when the referenced row is updated in the parent table which has a primary key.

```
CREATE TABLE student (
id INT PRIMARY KEY
CourseID INT ,
FOREIGN KEY (courseID) REFFERENCES course(id)
ON DELETE CASCADE
ON UPDATE CASCADE
);
```



“More Commands In Detail”

Table Related Queries

CHANGE Column (rename)

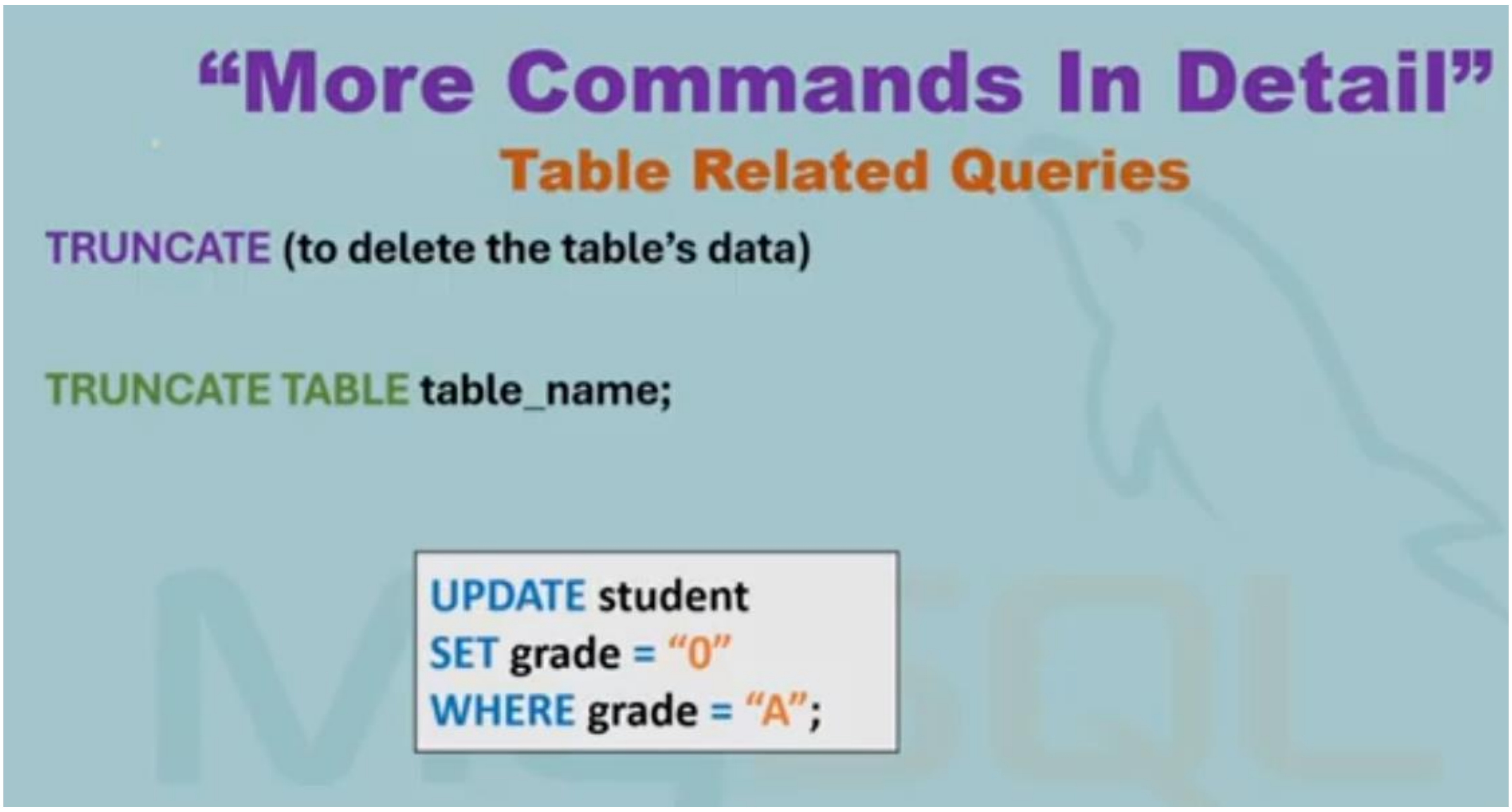
```
ALTER TABLE table_name
CHANGE old_name new_name new_datatype new_constraints;
```

```
ALTER TABLE student
CHANGE age  stu_age INT;
```

MODIFY Column (modify datatype/ constraint)

```
ALTER TABLE table_name
MODIFY col_name new_datatype new_constraint;
```

```
ALTER TABLE student
MODIFY COLUMN age VARCHAR (2);
```



“More Commands In Detail”

Table Related Queries

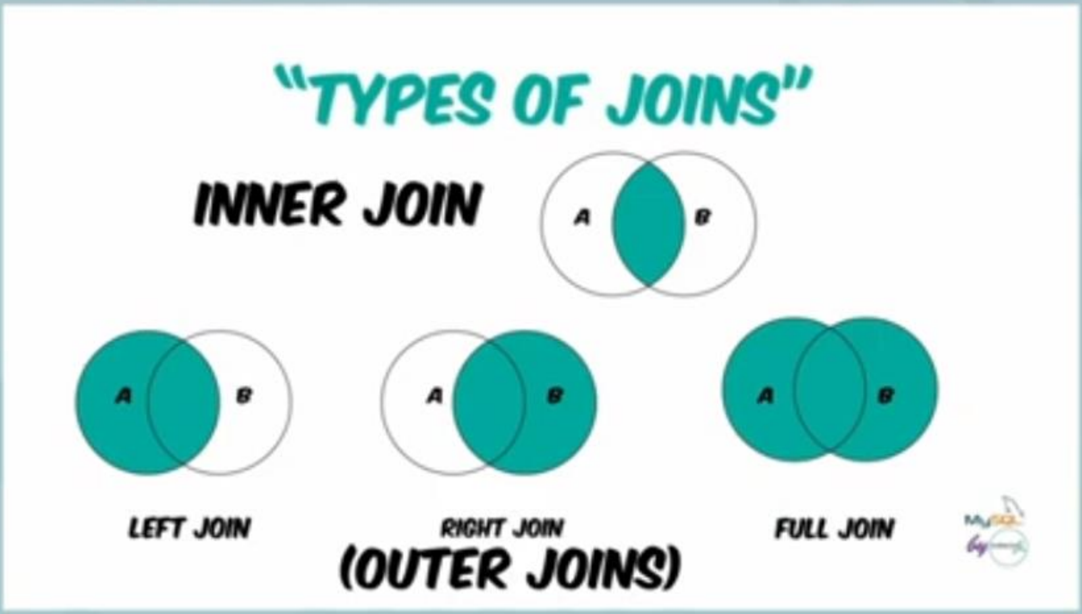
TRUNCATE (to delete the table’s data)

```
TRUNCATE TABLE table_name;
```

```
UPDATE student
SET grade = “0”
WHERE grade = “A”;
```

"Joins in SQL"

Join is used to combine rows from two or more tables, based on a related column between them.

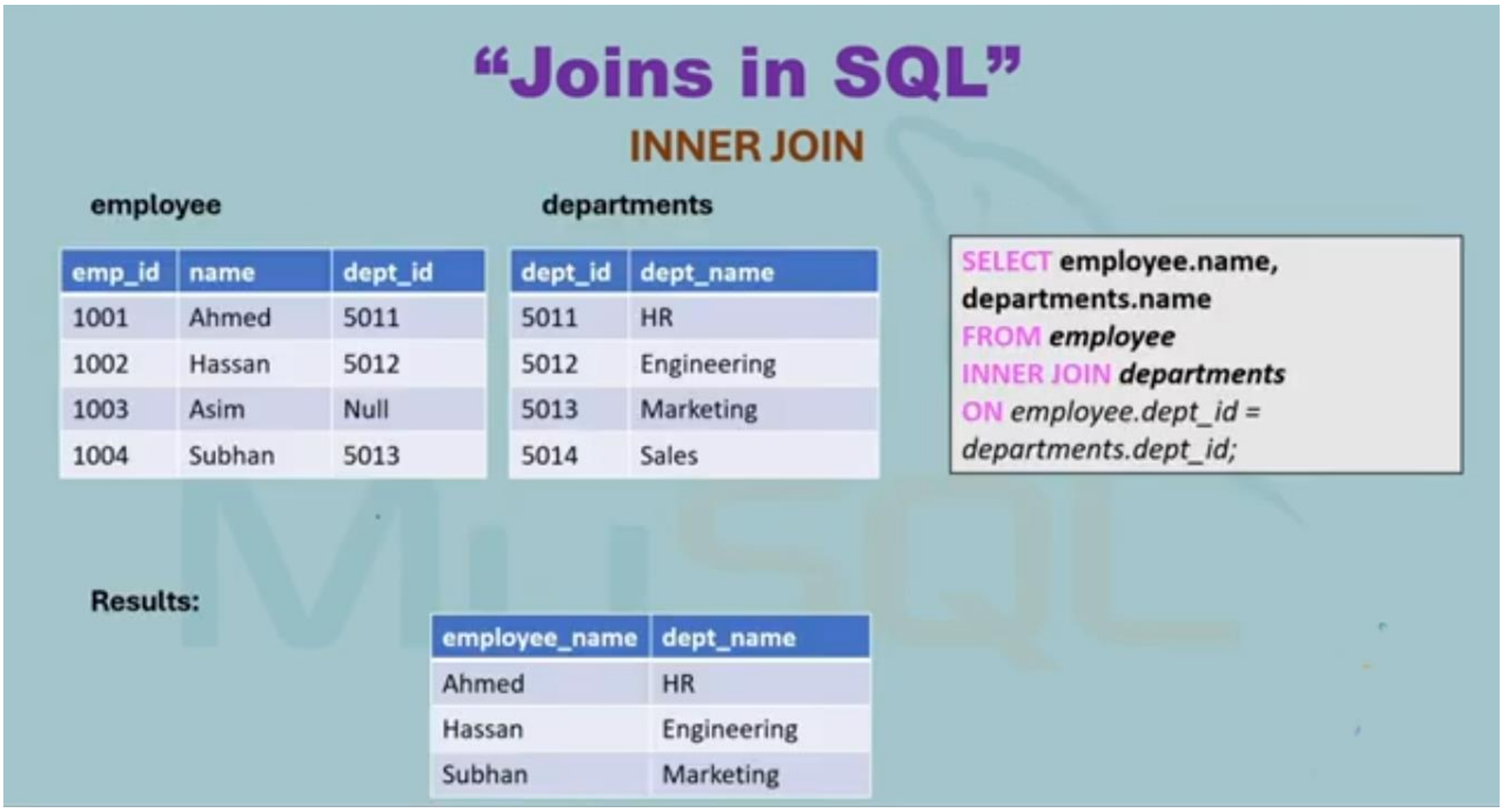


"Joins in SQL"

INNER JOIN

Returns records that have matching values in both tables.

```
SELECT column(s)
FROM tableA
INNER JOIN tableB
ON tableA.col_name = tableB.col_name;
```



"Joins in SQL"

INNER JOIN

employee

emp_id	name	dept_id
1001	Ahmed	5011
1002	Hassan	5012
1003	Asim	Null
1004	Subhan	5013

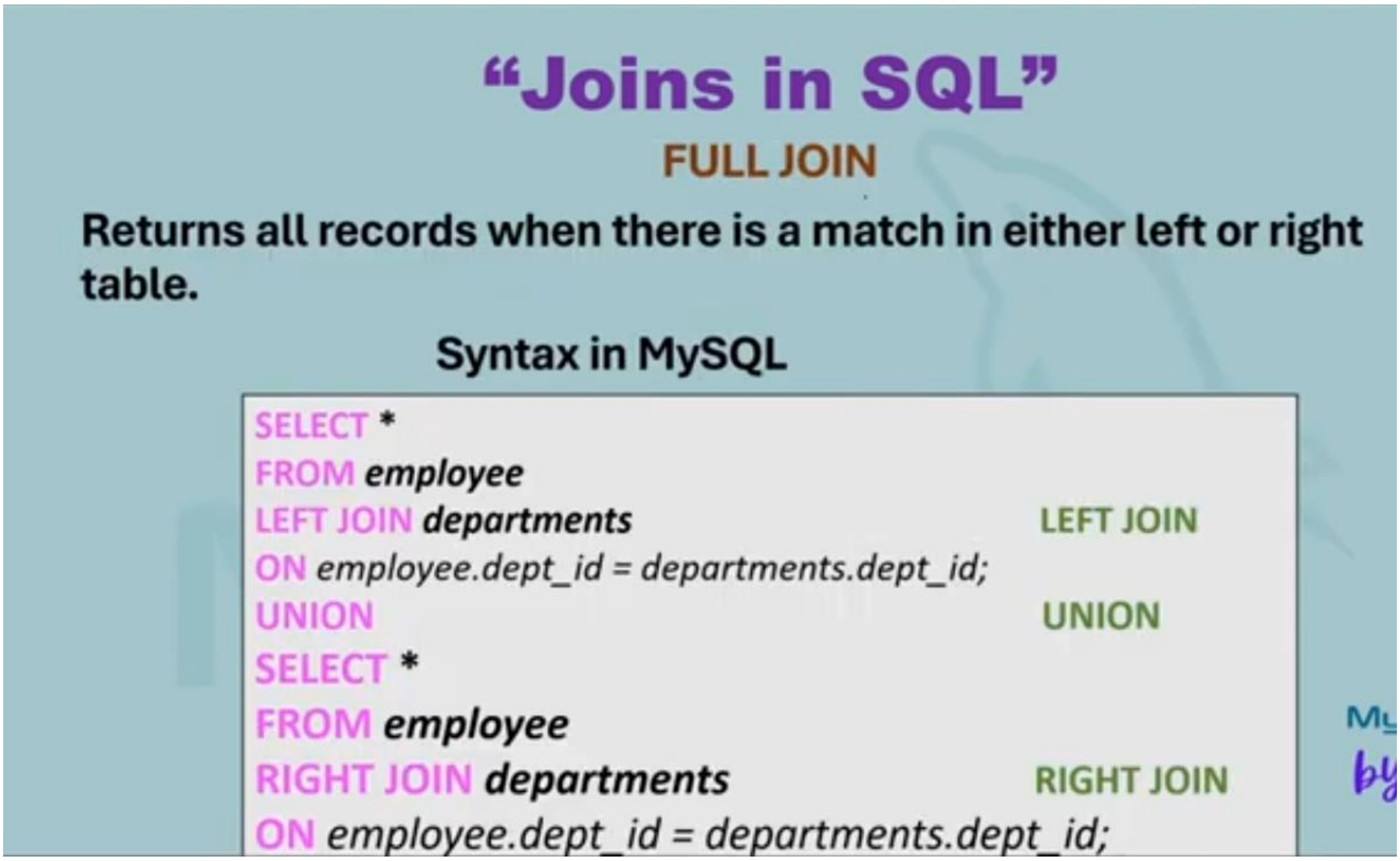
departments

dept_id	dept_name
5011	HR
5012	Engineering
5013	Marketing
5014	Sales

```
SELECT employee.name,
departments.name
FROM employee
INNER JOIN departments
ON employee.dept_id =
departments.dept_id;
```

Results:

employee_name	dept_name
Ahmed	HR
Hassan	Engineering
Subhan	Marketing



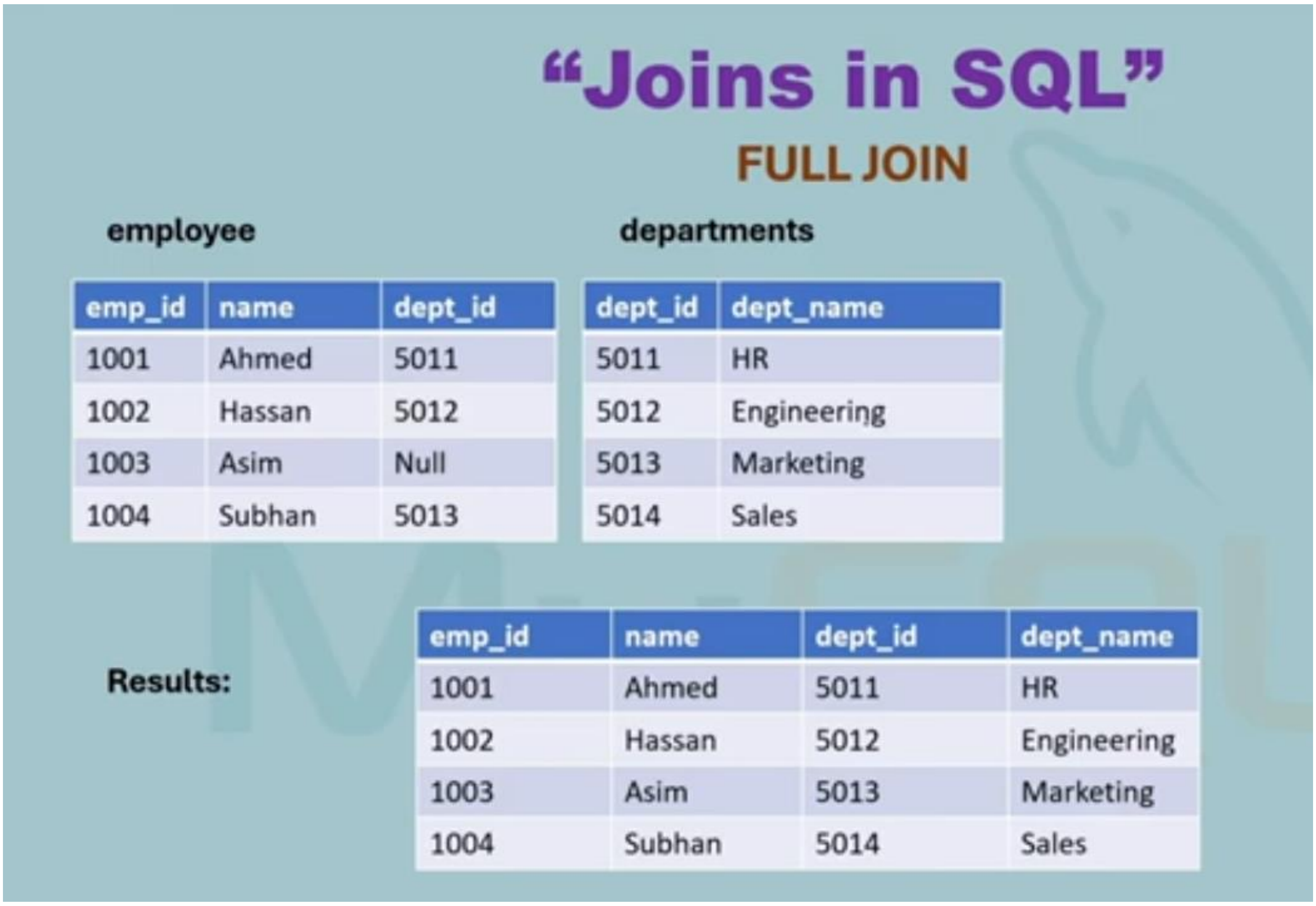
"Joins in SQL"

FULL JOIN

Returns all records when there is a match in either left or right table.

Syntax in MySQL

```
SELECT *
FROM employee
LEFT JOIN departments                              LEFT JOIN
ON employee.dept_id = departments.dept_id;
UNION                                              UNION
SELECT *
FROM employee
RIGHT JOIN departments                             RIGHT JOIN
ON employee.dept_id = departments.dept_id;
```



“Joins in SQL”

FULL JOIN

employee

emp_id	name	dept_id
1001	Ahmed	5011
1002	Hassan	5012
1003	Asim	Null
1004	Subhan	5013

departments

dept_id	dept_name
5011	HR
5012	Engineering
5013	Marketing
5014	Sales

Results:

emp_id	name	dept_id	dept_name
1001	Ahmed	5011	HR
1002	Hassan	5012	Engineering
1003	Asim	5013	Marketing
1004	Subhan	5014	Sales