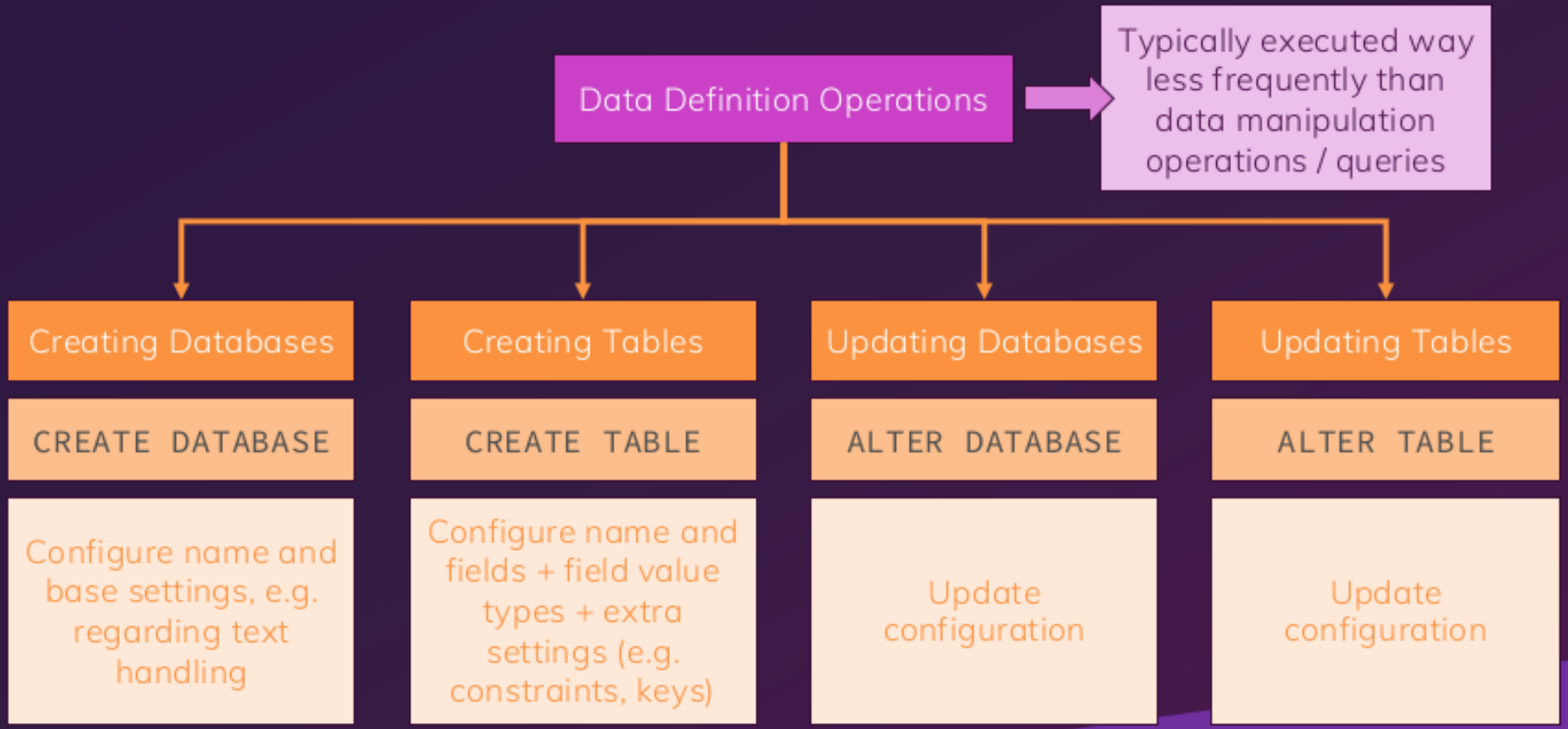
Data Base

پریسا حامد روح بخش

موسسه ی پارس پژوهان

Managing Databases & Tables



Key Data Types / Value Types

Text	Numeric	Date	Other
`CHAR(X)`	`INT, SMALLINT, …`	`DATE`	`BOOLEAN`
Store text up to X characters; shorter text will be space padded	Integer numbers (between min and max boundaries) are allowed	A value like 1986-10-20 (i.e. no hours or minutes)	True or false (0 or 1)
`VARCHAR(X)`	`DECIMAL, NUMERIC`	`DATETIME, TIMESTAMP`	`JSON`
Store text up to X characters; shorter strings will not be changed	Decimal numbers with a fixed precision (exact values)	A value like 1986-10-20 14:39:05 (i.e. with hours, minutes etc.)	JSON-formatted text data
`TEXT, LONGTEXT, …`	`FLOAT, REAL`		`SERIAL`
Text of any size can be stored without specifiy a max size first	Decimal numbers with floating points (approximated values)		An auto-incrementing integer number
`ENUM`			
Only values from a predefined set of allowed values are accepted			

Not all types are part of the official standard – and not all database systems support all types

Integer Values

5 10 -20 ...

Number Values With Decimal Places

3.14

5.58

-10.999

...

CHAR vs VARCHAR vs TEXT (vs LONGTEXT …)

Pre-defined maximum length

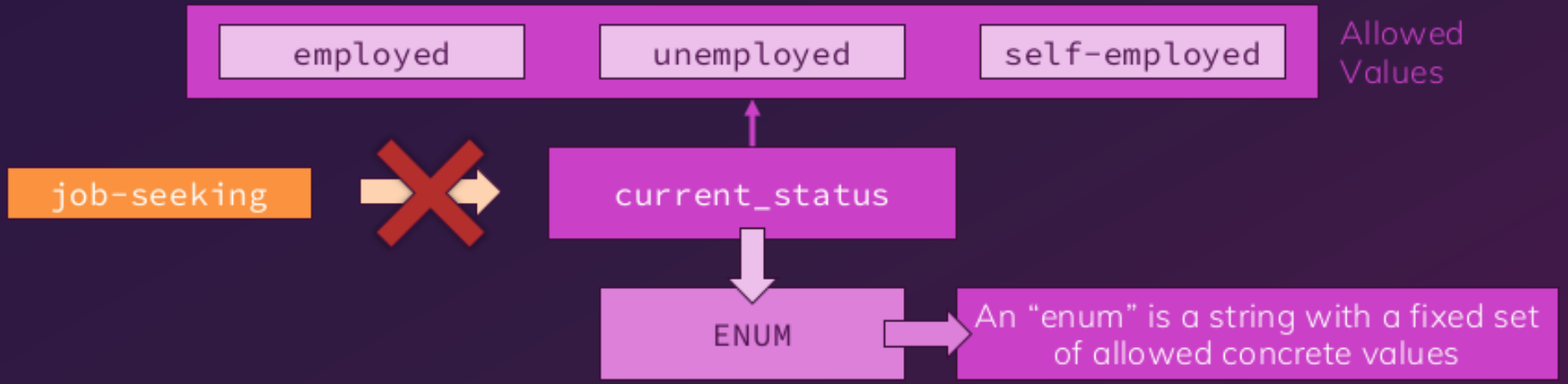
CHAR(X)	VARCHAR(X)
	Typically used!
Text with max. length of X bytes	
One byte can be one character ➔ Depends on encoding	
Shorter text is space-padded	Shorter text is not changed
CHAR(4)	VARCHAR(4)
Inserted 'hi'	Inserted 'hi'
Stored ' hi'	Stored 'hi'

No maximum length
(database system limits apply)

TEXT	LONGTEXT, …
Typically used!	
Text with no user-defined max. length (max. length depends on data type)	
One byte can be one character ➔ Depends on encoding	
Max. size is 1GB in Postgres, 65,535 characters in MySQL	Not supported in Postgres, different types with different sizes in MySQL

Not part of the SQL standard but supported by many database systems

Enums



Not standard SQL but supported by many database systems – but concrete implementations differ

MySQL

```
current_status ENUM(‘v1’, ‘v2’, …)
```

Postgresql

```
CREATE TYPE my_status AS ENUM(‘v1’, …)

current_status my_status
```

The Problem With "No Data"

first_name	last_name	salary	status
Max	Schwarz	15000	self-employed
Julie	Barnes	19000	employed
Michael	Smith	0	unemployed

Might distort analyses

Average: 11,333.33

The NULL Value

first_name	last_name	salary	status
Max	Schwarz	15000	self-employed
Julie	Barnes	19000	employed
Michael	Smith	[NULL]	unemployed

Shouldn't be null!

Is ignored

Average: 17,000

Allowing Or Forbidding NULL Values

```
CREATE TABLE users (
    full_name VARCHAR(255) NOT NULL,
    salary INT -- NULL is allowed because it's not forbidden
);
```

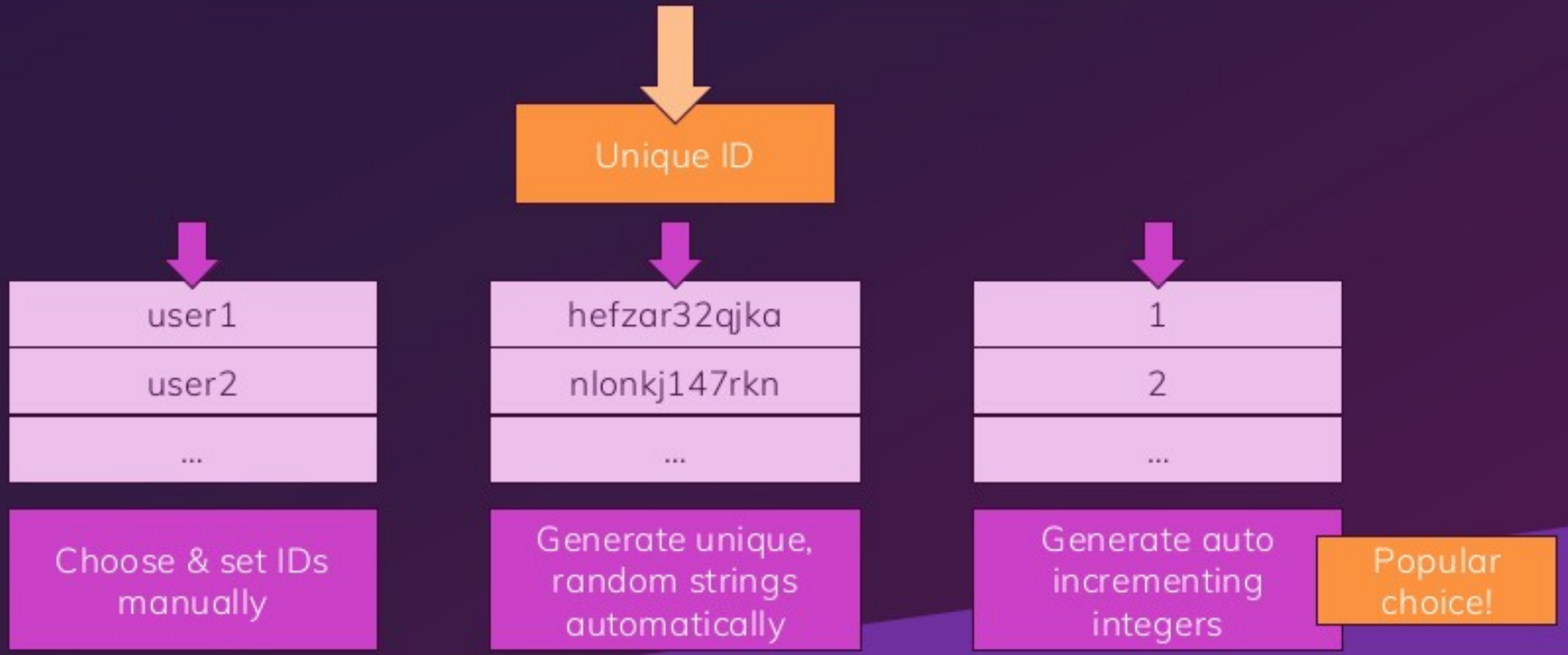
NOT NULL is a "Constraint"

This column must contain a (valid) value – omitting it is not possible

What If Multiple Users Have The Same Name?

The Role & Importance Of Unique IDs

When storing data, each data entry should have at least one unique value (for identifying the record)



Setting Unique IDs & Primary Keys

```
CREATE TABLE users (
    id INT NOT NULL UNIQUE,
    full_name VARCHAR(255) NOT NULL,
    salary INT
);
```

UNIQUE is a "Constraint"

This column must not contain duplicate values

Setting Unique IDs & Primary Keys

```
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    full_name VARCHAR(255) NOT NULL,
    salary INT
);
```

`AUTO_INCREMENT` is an "Attribute"

The db system will automatically insert an incrementing value

`PRIMARY KEY` is a "Constraint"

This column must be unique and `NOT NULL`; Only one `PRIMARY KEY` per table is allowed!

In most SQL databases / environments (e.g. Postgresql), `AUTO_INCREMENT` is not supported.

Use `SERIAL PRIMARY KEY` instead of `INT PRIMARY KEY AUTO_INCREMENT`

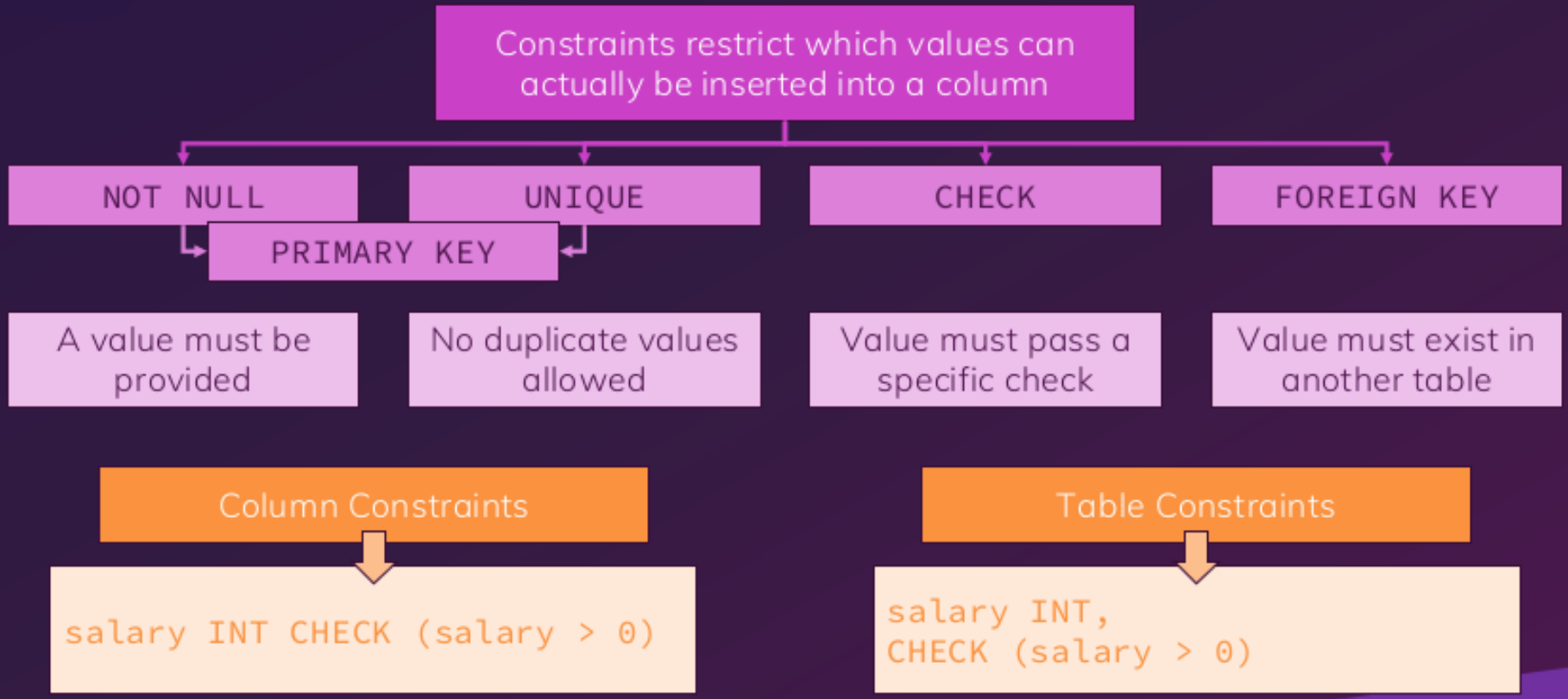
Setting Unique IDs & Primary Keys (Postgres)

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    full_name VARCHAR(255) NOT NULL,
    salary INT
);
```

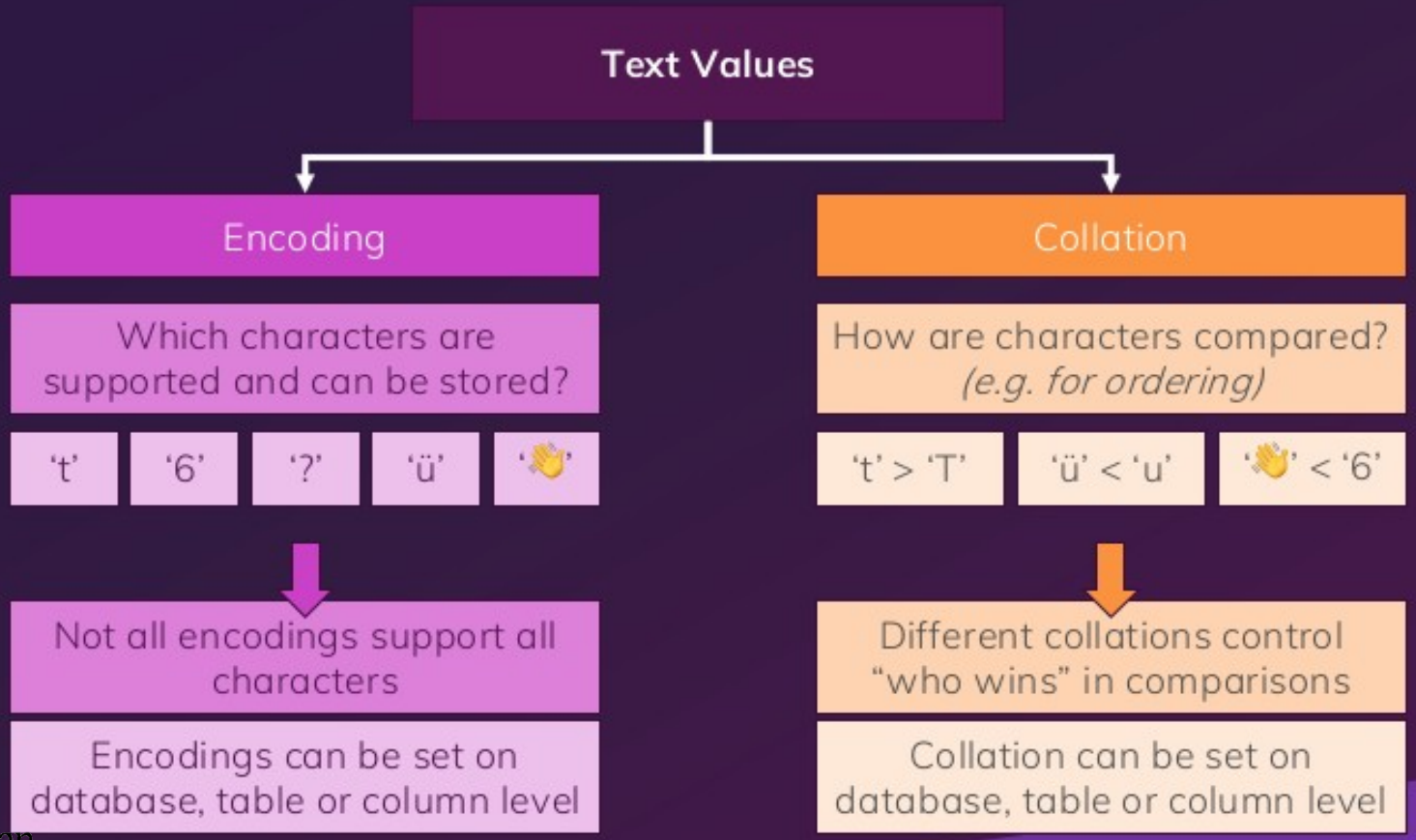
SERIAL is a Special Data Type

Creates an auto-incrementing integer
(not supported in MySQL)

More On Constraints



Text: Encoding & Collation



More Ways Of Creating Tables

Temporary Tables

```
CREATE TEMPORARY TABLE …
```

Tables that are only stored temporarily (in memory of the database server)

Useful for non-permanent data *(e.g. intermediate results)*

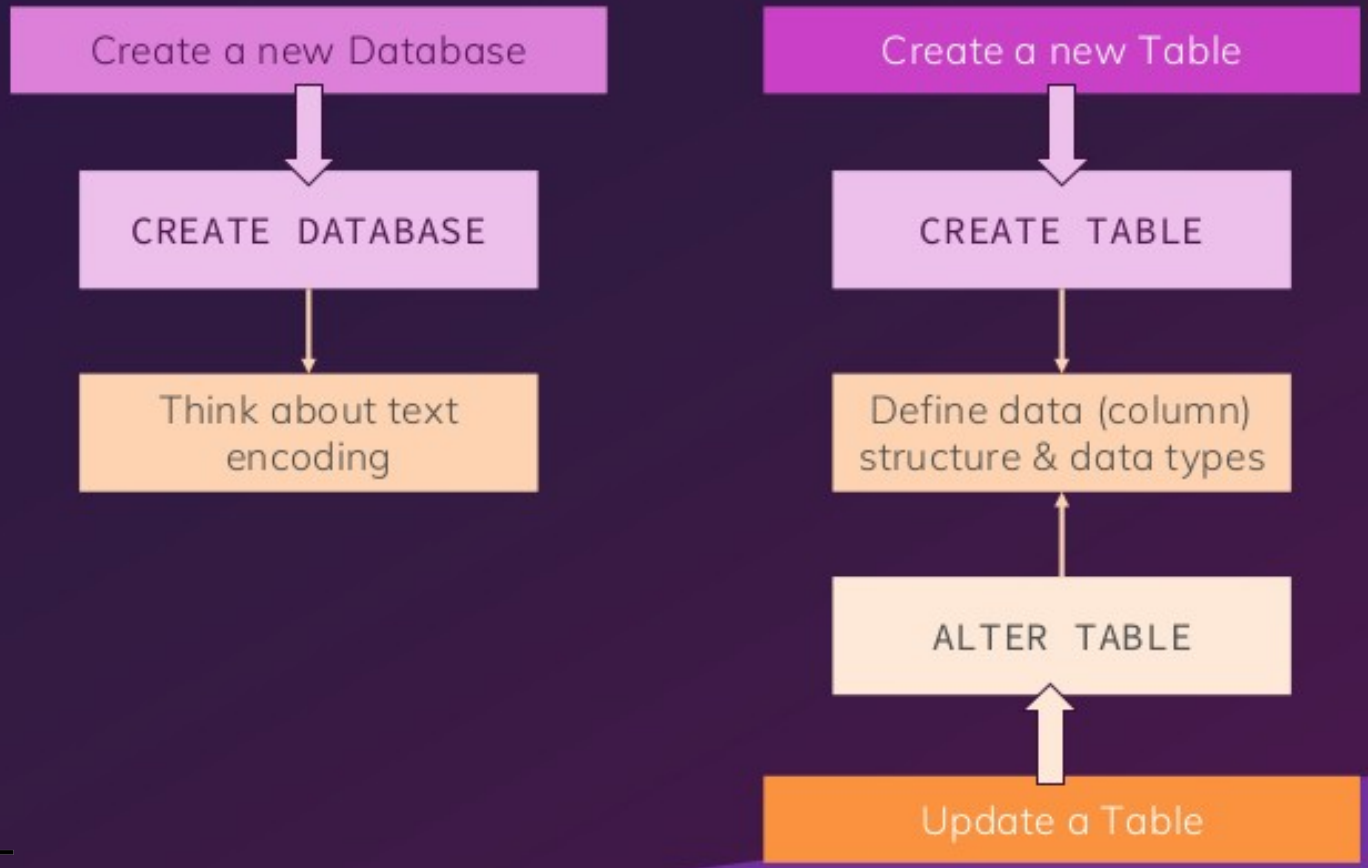
Tables Based On Other Tables / Data

```
CREATE TABLE … AS <query>
```

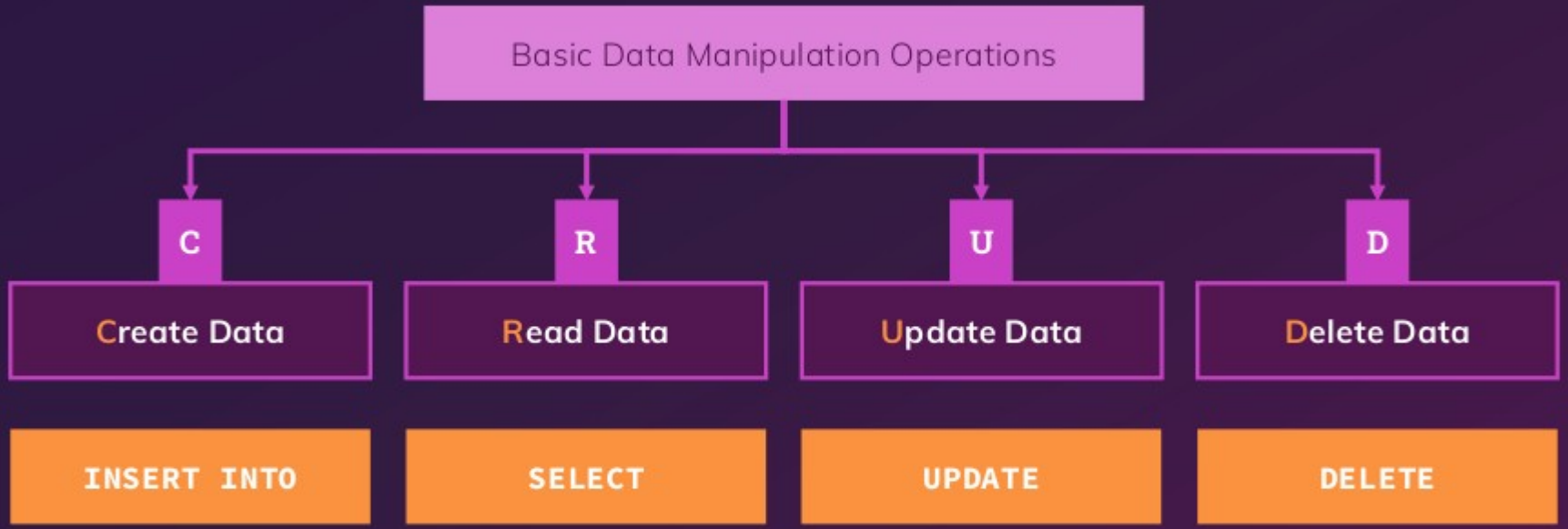
Creates a table and pre-populates it with data from a query result set

Useful if a subset of data from another table should be stored in a separate table

Module Summary



What Are "CRUD" Operations?



Inserting Data

```
INSERT INTO <table name> (<column names>)
VALUES (<column values>);
```

Insert values into columns *(column names can be omitted)*

```
INSERT INTO <table name> (<column names>)
VALUES
  (<column values>), -- row 1
  (<column values>); -- row 2
```

Insert multiple rows in one single command

```
INSERT INTO <table name> (<column names>)
SELECT <query>;
```

Insert values returned by a query

Basic Data Fetching (Selecting Data)

```
SELECT <columns> FROM <table name>;
```

Fetch the values for the specified columns from the specified table

```
SELECT <columns> FROM <table name> WHERE <condition>;
```

Fetch the values for the specified columns from the specified table – but only for records (rows) where the condition is met

Updating Data

```
UPDATE <table name>
SET <column name> = <new value>, …
WHERE <condition>;
```

Update all identified rows in a specified table and set the specified columns to new values

Multiple rows can be updated in one operation *(if the condition is met by multiple rows)*

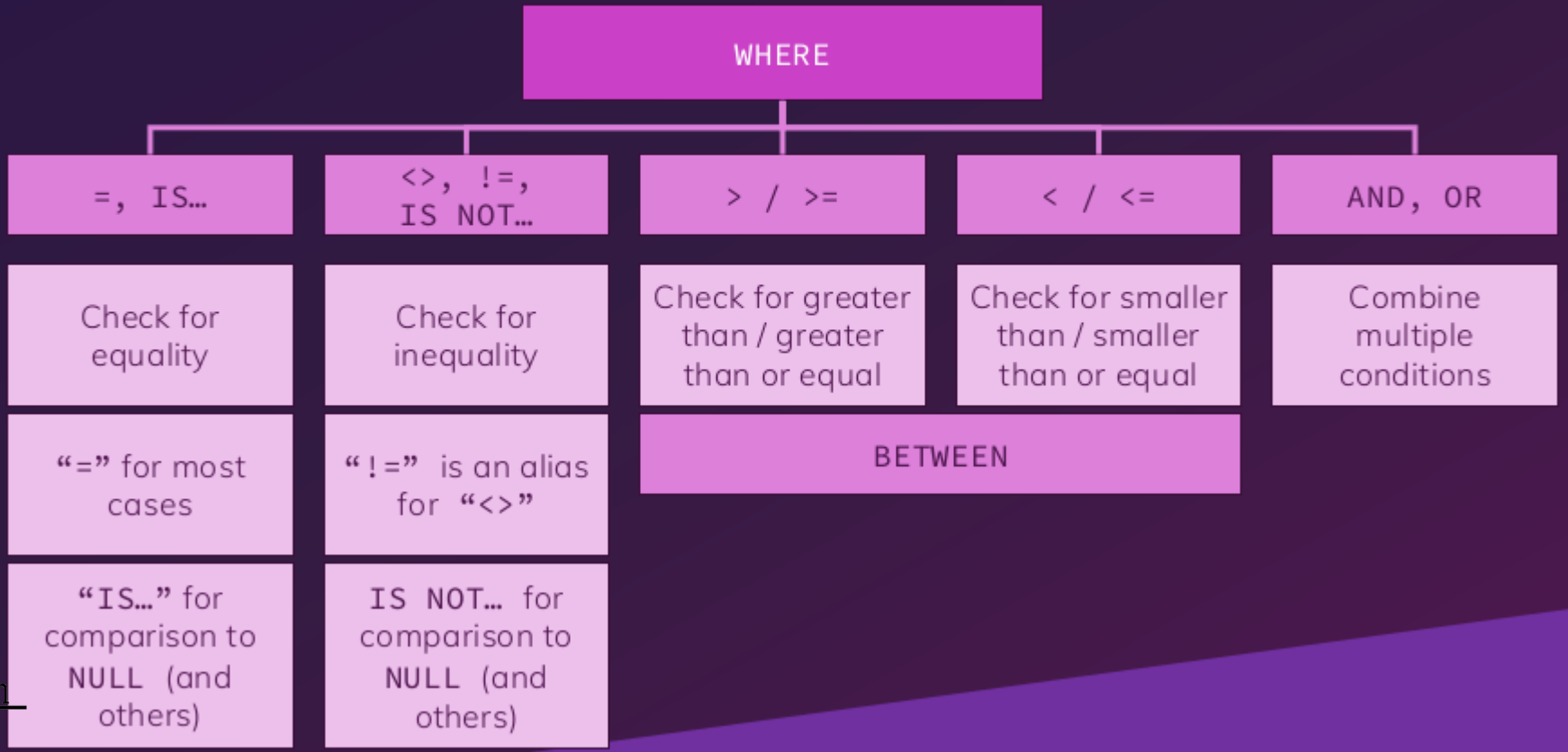
Deleting Data

```
DELETE FROM <table name>
WHERE <condition>;
```

Delete all identified rows in a specified table

Multiple rows can be delete in one operation
(if the condition is met by multiple rows)

A Closer Look At Filtering



Sorting / Ordering Results

```
SELECT * FROM <table>
ORDER BY <column name>;
```

Sort the selected data in ascending order

```
SELECT * FROM <table>
ORDER BY <column name> DESC;
```

Sort the selected data in descending order *(you could specify **ASC** but that would be the default)*

LIMIT & DISTINCT

`SELECT * FROM <table>` `LIMIT <number X>;`	Select only the first X number of rows *(can be combined with sorting)*

`SELECT * FROM <table>`
`LIMIT <number X>`
`OFFSET <offset number Y>;`
	Select X number of rows after skipping Y number of rows

`SELECT DISTINCT * FROM <table>;`	Drop any duplicates in the result set

Question?