# Database Design

DB Initialization Scripts
To Create Database & Objects

# OVERVIEW OF THE FIVE DATABASE SESSIONS

- Session 1:  The Transactional Relational Database
  - Work product: Conceptual Model
- Session 2:  Normalizing the Transactional Relational Database
  - Work product: Logical Model
- Session 3: Defining Data Structures Specific To A Database Platform (MariaDB)
  - Work product: Physical Model
- **Session 4: Database Initialization Scripts To Create Databases & Objects**
  - **Work product: SQL Scripts To Create Database Objects**
- Session 5: SQL Essentials To Query Databases
  - Work product: SQL Commands To Query The Database

# First, Let's Review

- Which model must be understood by everyone?
  - **The Conceptual Model**

- Which model shows the relationship between the entities using primary and foreign keys?
  - **The Logical Model**

- Which model is dependent on which database is being used?
  - **The Physical Model**

# Session 4 Objectives

- Create table statements
- Alter tables
- Database constraints
- Update, Delete, and Insert data
- Database Security
  - Access Control
    - Roles
    - Users
  - SQL Injection
- Add Create Scripts to docker container

# Create Table Statements

- Must Contain:
  - Table name
  - Column names
  - Data types
- “Best Practice” To Also Contain:
  - Nullable options
  - Auto Increment
  - Primary Keys (these should not be null)
  - Default Values
  - Comments (for table and/or columns)
  - Use descriptive and consistent names

# Alter Table – Add/Modify/Drop Columns (use DBeaver)

- Add New Column

```
ALTER TABLE technician
    ADD COLUMN status VARCHAR(10) NOT NULL
    COMMENT 'Status'
    AFTER technician_last_name;
```

- Rename / Modify Column:

```
ALTER TABLE technician RENAME COLUMN status TO active;
ALTER TABLE technician MODIFY active VARCHAR(10) NOT NULL COMMENT 'Active';
```

- Drop Column

```
ALTER TABLE technician DROP COLUMN active;
```

# Database Constraints

- They enforce data **Integrity**, **Accuracy**, and **Reliability** in the database.
- Referential Integrity ensures that the values of one column in a table are valid based on the values in a column in another table
  - How is Referential Integrity (RI) implemented?
    - By using Primary Keys and Foreign Keys.

# Database Constraints – “Best Practices”

- Use descriptive and consistent names

- Advantages of using descriptive constraint names:
  - Easily identify the table in the database.
  - Quickly identify and fix any errors.
  - Confidently modify or drop the correct constraints.

# Types Of Database Constraints

- Foreign Keys –
  - used to create relationships and enforce data integrity
- Unique Keys –
  - data values are unique for each row
- Non-unique key –
  - data values may be shared among several rows
- Check Constraints –
  - used to limit what values can be placed in a column

# Update Data – "Best Practices" (use DBeaver)

1. Select the data that needs updated – Verify the WHERE clause

```
SELECT *
FROM work_order_pro.state
WHERE state_code = 'OH';
```

2. Update the data – Use the same WHERE clause

```
UPDATE work_order_pro.state
SET state_name = 'Wrong State'
WHERE state_code = 'OH';
```

3. Verify the correct data was updated – Use the same WHERE clause

```
SELECT *
FROM work_order_pro.state
WHERE state_code = 'OH';
```

# Delete Data – "Best Practices" (use DBeaver)

1. Select the data that needs deleted – Verify the WHERE clause

```
SELECT *
FROM work_order_pro.state
WHERE state_code = 'OH';
```

2. Delete the data – Use the same WHERE clause

```
DELETE FROM work_order_pro.state
WHERE state_code = 'OH';
```

3. Verify the correct data was deleted – Use the same WHERE clause

```
SELECT *
FROM work_order_pro.state
WHERE state_code = 'OH';
```

# Insert Data – "Best Practices" (use DBeaver)

1. Verify the data does not already exist

```
SELECT *
FROM work_order_pro.state
WHERE state_code = 'OH';
```

2. Insert the data

```
INSERT INTO work_order_pro.state (state_code, state_name) VALUES ('OH', 'OHIO');
```

3. Verify the data was inserted

```
SELECT *
FROM work_order_pro.state
WHERE state_code = 'OH';
```

# Insert Data – 3 variations of syntax

```
INSERT INTO work_order_pro.state (state_code, state_name) VALUES ('AL', 'Alabama');
INSERT INTO work_order_pro.state (state_code, state_name) VALUES ('AK', 'Alaska’);

INSERT INTO work_order_pro.state VALUES ('AZ', 'Arizona’);
INSERT INTO work_order_pro.state VALUES ('OH', 'Ohio’);

INSERT INTO work_order_pro.state
VALUES
      ('AL', 'Alabama’)
     ,('AZ', 'Arizona’)
     ,('OH', 'Ohio’)
     ,('AK', 'Alaska’);
```

# Database Security — **Access Control**

- Database Security is defining access rights in the database.
- Users and Applications access the database thru database permissions.
- Database Permissions:
  - Control what a user and application can or cannot do in the database
  - May be granted to an individual or to a role
  - Can be granted or revoked at any time

# Database Security — **Roles**

- A database role groups a set of database permissions that let allow users and applications to perform specific tasks

- Database roles:
  - Are used to organize database permissions into groups
    - Read Only access (select)
    - Read Write access (select, insert, update, and delete; but not create objects)
  - Each role is assigned specific permissions and privileges, or even another role
  - Reduce access maintenance by allowing permissions to be granted as a group instead of individually

- MariaDB specific — users can only 1 active role at a time

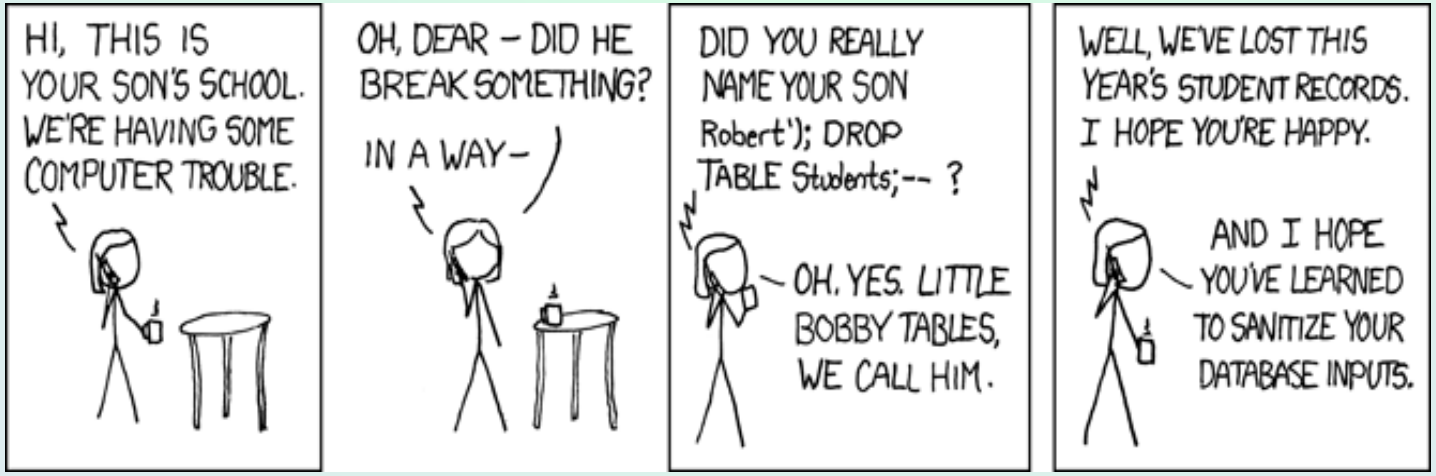
# Database Security — **Users**

- Different type of users
  - Database Administrators
    - DBA privileges in all databases
  - Developers
    - Read Write in Development databases
    - Read Only in QA and Production databases
  - Business Users
    - Read Only in all databases
  - Application Accounts
    - Read Write in all databases


- Database users are authenticated by a username and a password

# Database Security — What is SQL Injection

- SQL injection is one of the most common web hacking techniques.
- It is a cyber attack technique that may destroy your database.
- An attack consists of the *insertion or "injection" of a SQL query* via the input data from the user application.
  - For example, a user login screen



# Database Security — Prevent SQL Injection

- Most SQL injection attacks can be attributed to faulty scripts and programs.
- SQL injection attacks can be prevented:
  - Keep the server software up to date.
  - "Sanitize" database inputs by restricting input text or by filtering out certain characters.
    - For example: replace ; with blank, replace = with blank, replace '' with blank
  - "Parameterize" the SQL queries — use placeholders/variables in the query that refer to the user input instead of inserting it directly into the query.
    - For example: @username = 'value they input'

# Add Create Scripts to docker container

- Create folder like Work_Order_Pro
- Copy docker-compose.yml file & Create init_Project_Name.sql file
  - docker-compose.yml
    - container_name:

      container_name: WSU_2024_improved_w_survey
    - volumes:

      volumes:

        - ./init_Project_Name.sql:/docker-entrypoint-initdb.d/init.sql
  - init_Project_Name.sql – Modify to:
    - CREATE DATABASE Project_Name;
    - USE Project_Name;
    - Add all of the code in your create table scripts
      (**Note: make sure each statement ends with a ; )

# HOMEWORK – WRITE SQL QUERIES

- Session 1:  The Transactional Relational Database
  - Work product: Conceptual Model
- Session 2:  Normalizing the Transactional Relational Database
  - Work product: Logical Model
- Session 3: Defining Data Structures Specific To A Database Platform (MariaDB)
  - Work product: Physical Model
- Session 4: Database Initialization Scripts To Create Databases & Objects
  - Work product: SQL Scripts To Create Database Objects
- **Session 5: SQL Essentials To Query Databases**
  - **Work product: SQL Commands To Query The Database**

# HOMEWORK – WRITE SQL QUERIES (details)

- Some scopes to think of are below: (your group must have 5 purpose-driven scripts)

- Query to view all records
  - Think: "View all customers"
- Track occurrences of something
  - Think: "How often are we doing..."
- Query count of something
  - Think: "How many of these do we need?"
- Query between multiple tables
  - Think: "How does x relate to y?"