

INFS3202/7202 – Web Information Systems

Lecture Week 4: MVC – Databases and CI Models

Dr Aneesha Bakharia (Senior Lecturer, EECS)
Email: a.bakharia1@uq.edu.au

Contents

-
- 01 What is a Relational Database Server?
 - 02 Structured Query Language (SQL)
 - 03 Adding Database Config in CodeIgniter 4
 - 04 CodeIgniter Migrations (Versioning for creating and updating tables)
 - 05 CodeIgniter Models
 - 06 Using the CodeIgniter Query Builder Class
 - 07 Code Walkthrough: Search and Results Page
(Using Models)
 - 08 Designing a Database
-

Course Updates

Issue/Feedback	Change
Sharing Frequent Questions in Labs (Request via Forum Post)	<ul style="list-style-type: none">• Great idea – thank you for posting• Demonstrators will add to an FAQ on the Blackboard site (just below the link to the Lab html file)
Questions regarding Database Diagram style	<ul style="list-style-type: none">• Again thanks for asking via the forum• Will discuss the ERD today

What is a Relational Database Server

What is a Relational Database Server?

- **Stores Data Efficiently:**

A database organizes and stores data in a manner that makes it easily accessible and manageable. Indexes data so that results are returned fast.

- **Multiple Related Tables:**

It can consist of several related tables, allowing for the organization of different types of information in a structured way.

- **Scales with Needs:**

Databases are designed to scale, meaning they can handle an increasing amount of work or data as the need grows.

- **Search, Add, Update via SQL:**

Utilizes Structured Query Language (SQL) for searching, adding, and updating data, providing a powerful tool for managing the database.



What Database Servers are Available?

- **SQLite**: A lightweight database filesystem database. Ideal for development, small applications, and situations where simplicity and minimal setup are advantageous.
- **MySQL**: A popular open-source relational database management system known for its reliability and ease of use. It's widely used for web applications and supports a variety of programming languages.
- **PostgreSQL**: An advanced open-source relational database system, offering strong compliance to SQL standards, and known for its robustness, scalability, and support for advanced data types and functionalities.
- **MS SQL Server**: A comprehensive, enterprise-level database management system developed by Microsoft. Known for its high performance, robust security features, data integration, and analysis tools.
- **Oracle Database**: A comprehensive, enterprise-grade database solution that provides robust support for large-scale databases, complex applications, and data warehousing needs. Known for its performance, security, and scalability.
- **Local and UQCloud Options**: SQLite can be used locally, providing an easy-to-manage database environment without the need for a server. Both MySQL and PostgreSQL are available on UQCloud.



ORACLE®

Why are Databases Essential for Web Development?

- **Data Management:**

Databases provide a structured and efficient way to store, manage, and retrieve data, which is crucial for dynamic websites and applications.

- **User Interaction:**

They enable personalized user experiences by storing user preferences, orders, and account information, enhancing engagement and satisfaction.

- **Scalability:**

Databases can handle increasing amounts of data and user requests, allowing websites to grow and accommodate more traffic seamlessly.

- **Data Integrity:**

Ensure consistency and accuracy of the data through constraints, transactions, and relational database design, minimizing errors and data duplication.

- **Security:**

Databases offer robust security features to protect sensitive information, such as user data and financial transactions, from unauthorized access and breaches.

Web Servers vs Database Servers

- On your UQCloud Zone the Web Server and Database are on the same server
- For Scalability we usually separate the Web Server and the Database
- This just means that the database server software is installed on a separate server

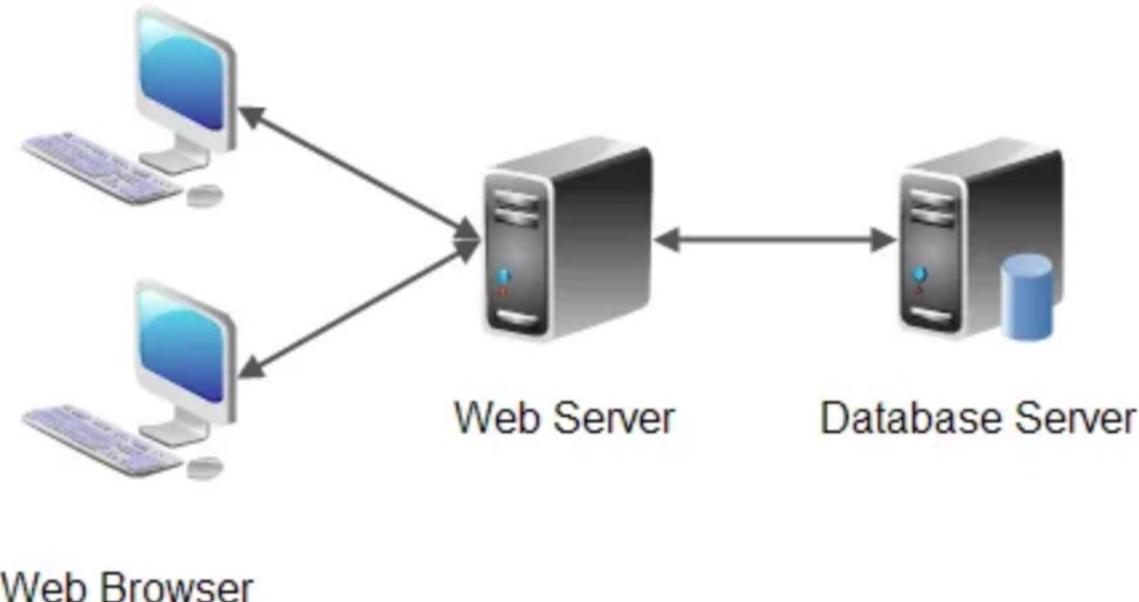


Image from: <https://medium.com/@skariel3/your-db-and-web-server-belong-physically-together-b267747f4230>

A Simple View of a Database

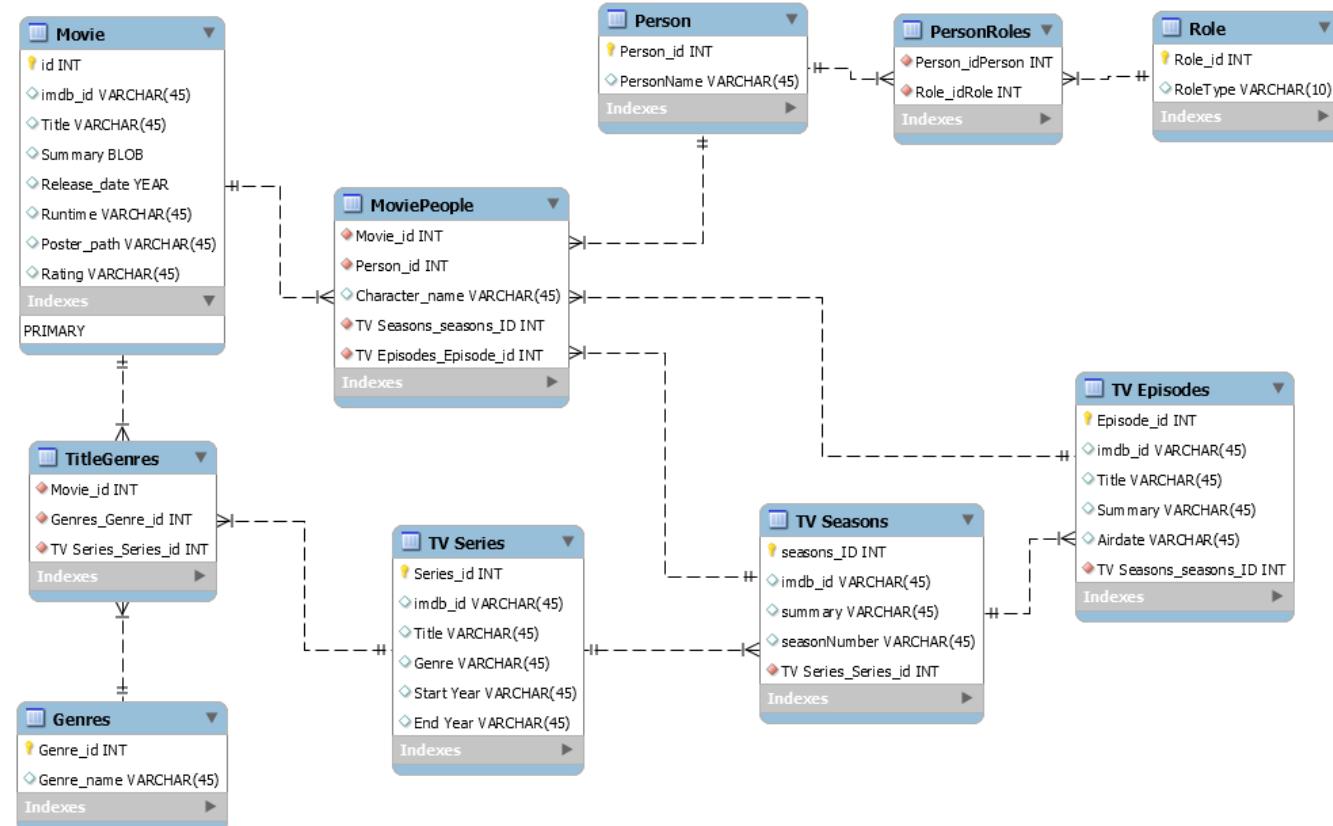
So... What exactly is inside a Relational Database Management System?

- A collection of tables or entities
- Each table is assigned a unique name
- Each table consists of a set of attributes or fields or columns
- Each attributes or field or column must have a specific data type
- Each table can store many records or rows of data
- Tables can be related via a common id

Course_id	Student_id	attributes		
Course_id	Student_id	First name	Last name	
3202	4132335			
3204	4052922			
Course_list		4044151	Jack	Smith
		4052922	Zoe	Lee
		4132335	John	Hunter
Student_list				

Database Schema and Entity Relationship Diagrams (ERD)

- The database schema of a database system is its **structure** described in a formal language supported by the database management system (DBMS).
- The term "schema" refers to the organization of data as a **blueprint** of how the database is constructed (divided into database tables in the case of relational databases).
- An ERD is used to visually display tables and their relationships



<https://stackoverflow.com/questions/45135485/creating-a-database-schema-from-the-movie-database>

You are required to create an ERD for the Design Document Assessment

Structured Query Language (SQL)

What is SQL?

- **Structured Query Language:**

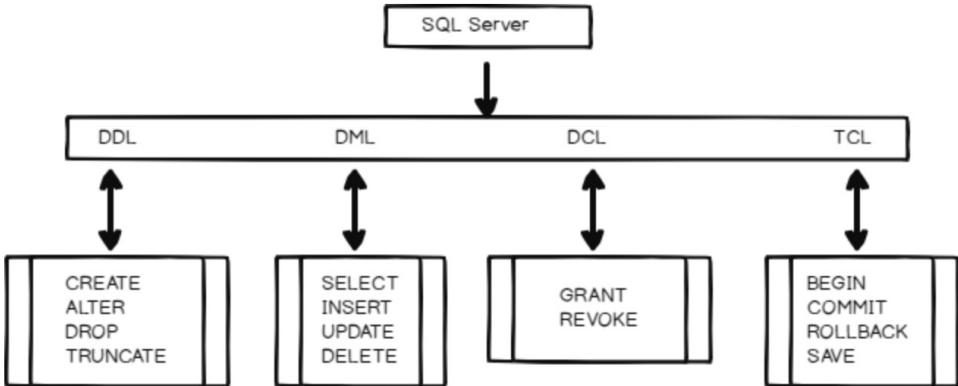
SQL stands for Structured Query Language, a standardized programming language specifically designed for managing and manipulating relational databases.

- **Data Operations:**

It is used to perform various data operations such as querying, inserting, updating, and deleting data within a database.

- **Database Schema Creation and Modification:**

SQL is also used for creating and modifying the structure of database systems, including tables, schemas, and indexes.



<https://www.sqlshack.com/what-is-sql-in-sql-server/>

The Structured Query Language (SQL)

- **Data Definition Language (DDL)** - Defines and modifies a schema e.g. **CREATE / DROP / ALTER table**; does not manipulate data
- **Data Manipulation Language (DML)** - Language used to retrieve (**SELECT**), add (**INSERT**), modify (**UPDATE**) and **DELETE data**
- **Data Control Language (DCL)** statements. Used for providing (**GRANT**) / withdrawing (**REVOKE**) **access privileges**
- **Transaction Control Language (TCL)** statements are used to manage the changes made by DML statements. It allows statements to be grouped together into **logical transactions**. Example: **COMMIT, ROLLBACK**, etc.
-

Running SQL

- Multiple ways to run SQL
 - Use a GUI tool
e.g. PHPMyAdmin
 - Use a command line tool
e.g. MySQL command line
 - Use a CodeIgniter Migration
(In this course we'll mainly focus on using Migrations but it is good to see how the create statement works)

Using PHPMyAdmin on UQCloud

- Main steps
 - Setup the mysql service
 - sudo webprojctl enable mysql
 - Go to <https://youruqzoneurl/phpmyadmin>

The screenshot shows the phpMyAdmin interface for the 'resume_builder' database. The left sidebar lists databases: New, information_schema, mysql, performance_schema, phpmyadmin, resume_builder (selected), sys, and sys. The resume_builder database has tables: New, Address, Education, migrations, User, and Work. The main area displays the 'Structure' tab for the 'Address' table. The table structure includes columns: id (Primary Key, Auto Increment), address_line1, address_line2, city, state, zip_code, and country. The table contains 3 rows, is InnoDB, and uses utf8mb3_general_ci collation. The total size is 32.0 KiB.

Table	Action	Rows	Type	Collation	Size	Overhead
Address	Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb3_general_ci	32.0 KiB	
Education	Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb3_general_ci	32.0 KiB	
migrations	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb3_general_ci	16.0 KiB	
User	Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb3_general_ci	16.0 KiB	
Work	Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb3_general_ci	32.0 KiB	

- Provides a good first place to:
 - Create a database
 - Create tables
 - Run SQL select statements

Using PHPMyAdmin on UQCloud

- We will now setup a database with 2 related tables and run some queries using PHPMyAdmin

Table 1: [customers](#)

Field Name	Data Type
id	INT
name	VARCHAR(255)
email	VARCHAR(255)

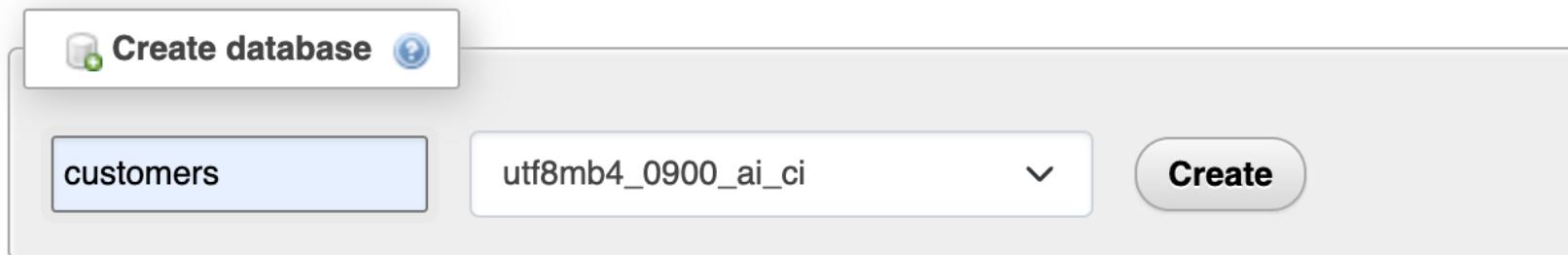
Table 2: [orders](#)

Field Name	Data Type
id	INT
customer_id	INT
order_date	DATE
total_amount	DECIMAL(10,2)

Creating the Database

- You can create a database via the GUI
- Enter the name of the database and choose the charset

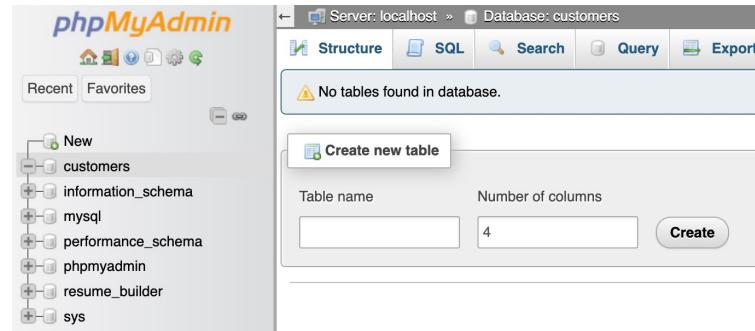
Databases



- In Lab 3, you'll learn how to use the command line to create a new database.

Creating the Tables

- You can create a table and add columns via the GUI



- OR
- Use the SQL Create Statement

```
CREATE TABLE customers (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL
);

CREATE TABLE orders (
    id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT NOT NULL,
    order_date DATE NOT NULL,
    total_amount DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (customer_id) REFERENCES customers(id)
);
```

Code in week4_code/sql/create_table.sql

The PHPMyAdmin - Designer

- You can also use the Designer to create tables and relationships
- An ERD is automatically created

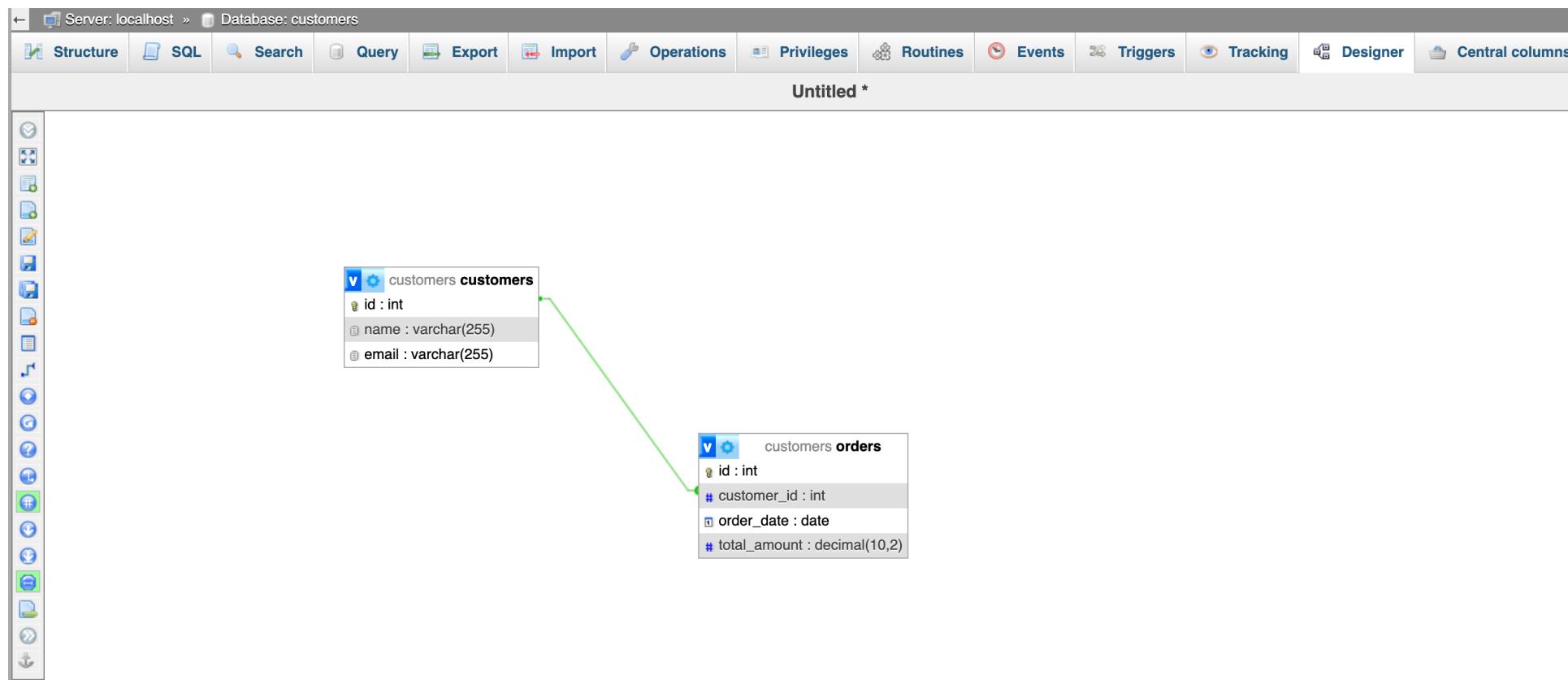


Table Datatypes in MySQL

CHAR (n) – holds a fixed length string

- the length must be n

VARCHAR (n) – holds a variable length string

- the maximum size is n

BLOB – binary large object

- Images, audio or other multimedia objects

DATE

- Format: YYYY-MM-DD

TIME

- Format: HH:MM:SS

TIMESTAMP

- Format: YYYY-MM-DD HH:MM:SS
- The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
- Automatically set itself to the current date and time, in an INSERT or UPDATE query

Insert Records into a Database

Syntax

Insert into *TableName* **values** (*value1*, *value2*, ...)

Insert into *TableName* (*attribute1*, *attribute2*, ...) **values** (*value1*, *value2*, ...)

The screenshot shows the MySQL Workbench interface with the SQL tab selected. A query editor window displays two SQL scripts. The first script inserts data into the 'customers' table:

```
1 INSERT INTO customers (name, email) VALUES
2 ('Raj Patel', 'raj@example.com'),
3 ('Wei Chen', 'wei@example.com'),
4 ('Emma Müller', 'emma@example.com'),
5 ('Priya Gupta', 'priya@example.com'),
6 ('Liam Smith', 'liam@example.com');

7
8 INSERT INTO orders (customer_id, order_date, total_amount) VALUES
9 (1, '2023-04-01', 100.00),
10 (2, '2023-04-05', 80.50),
11 (3, '2023-04-10', 120.25),
12 (4, '2023-04-15', 90.75),
13 (1, '2023-04-20', 60.00),
14 (5, '2023-04-25', 110.00),
15 (2, '2023-04-30', 95.50);
```

The second script inserts data into the 'orders' table. Below the code, there are several buttons: SELECT *, SELECT, INSERT, UPDATE, DELETE, Clear, Format, Get auto-saved query, Bind parameters, and a bookmark input field. At the bottom, there are options for Delimiter, Show this query here again, Retain query box, Rollback when finished, Enable foreign key checks, and a Go button.

Code in week4_code/sql/create_table.sql

Check that the Records have been added

Customers Table

			id	name	email
<input type="checkbox"/>				1 Raj Patel	raj@example.com
<input type="checkbox"/>				2 Wei Chen	wei@example.com
<input type="checkbox"/>				3 Emma Müller	emma@example.com
<input type="checkbox"/>				4 Priya Gupta	priya@example.com
<input type="checkbox"/>				5 Liam Smith	liam@example.com

Check all With selected:

Orders Table

			id	customer_id	order_date	total_amount
<input type="checkbox"/>				1	1 2023-04-01	100.00
<input type="checkbox"/>				2	2 2023-04-05	80.50
<input type="checkbox"/>				3	3 2023-04-10	120.25
<input type="checkbox"/>				4	4 2023-04-15	90.75
<input type="checkbox"/>				5	1 2023-04-20	60.00
<input type="checkbox"/>				6	5 2023-04-25	110.00
<input type="checkbox"/>				7	2 2023-04-30	95.50

Check all With selected:

Retrieving Data from a Table

Syntax

Select fields from TableName where condition

Some examples:

```
SELECT * FROM customers;
```

```
SELECT name, email FROM customers;
```

```
SELECT * FROM customers WHERE id = 3;
```

```
SELECT * FROM customers WHERE email LIKE '%Chen%';
```

```
SELECT COUNT(name) FROM customers WHERE name LIKE '%P%';
```

```
SELECT * FROM customers WHERE email LIKE '%example.com';
```

Note:

The **WHERE** clause is used to extract only those records that fulfil a specified criterion.

The **LIKE** pattern matching operator can also be used in the conditional selection of the where clause.

"%" can be used as a wild card to match any possible character

SQL to Join Tables

Syntax

```
SELECT column1, column2, ...
```

```
FROM table1
```

```
JOIN table2 ON table1.column = table2.column;
```

Get all orders for each customer

```
SELECT o.id AS order_id, o.order_date, o.total_amount, c.name, c.email  
FROM orders o JOIN customers c ON o.customer_id = c.id;
```

order_id	order_date	total_amount	name	email
1	2023-04-01	100.00	Raj Patel	raj@example.com
5	2023-04-20	60.00	Raj Patel	raj@example.com
2	2023-04-05	80.50	Wei Chen	wei@example.com
7	2023-04-30	95.50	Wei Chen	wei@example.com
3	2023-04-10	120.25	Emma Müller	emma@example.com
4	2023-04-15	90.75	Priya Gupta	priya@example.com
6	2023-04-25	110.00	Liam Smith	liam@example.com

SQL to Join Tables

Get orders for a specific customer by ID

```
SELECT o.id AS order_id, o.order_date, o.total_amount, c.name, c.email  
FROM orders o  
JOIN customers c ON o.customer_id = c.id  
WHERE c.id = 2;
```

order_id	order_date	total_amount	name	email
2	2023-04-05	80.50	Wei Chen	wei@example.com
7	2023-04-30	95.50	Wei Chen	wei@example.com

Retrieving Data from a Table

A bit more...

ORDER BY: to sort the result-set

```
SELECT column_name(s)  
FROM table_name  
ORDER BY column_name(s) ASC|DESC
```

SQL uses single quotes around text values; Numeric values should not be enclosed in quotes.

```
SELECT * FROM customers WHERE Name='Emma'  
SELECT * FROM customers WHERE Name=Emma
```



AND & OR Operator

```
SELECT * FROM customers  
WHERE Name='Emma' AND id > 2
```

Deleting Data

Syntax

```
DELETE FROM table_name  
WHERE some_column=some_value
```

```
DELETE FROM customers  
WHERE name='Mary'
```

Notes:

Delete all rows in a table without deleting the table

```
DELETE FROM table_name  
or  
DELETE * FROM table_name
```

The table structure, attributes, and indexes will be intact

Update Records in a Table

Syntax

```
UPDATE table_name  
SET column1=value,  
column2=value2,...  
WHERE some_column=some_value
```

```
UPDATE StudentRecord  
SET Name='Jack'  
WHERE studentID = '1234'
```

Warning!

Result?

```
UPDATE customers  
SET Name='Jack'
```

Always include a WHERE clause

Alter Table

Syntax

Add a column in a table

```
ALTER TABLE table_name  
ADD column_name datatype
```

```
ALTER TABLE customers  
ADD Country varchar(20)
```

Change the data type of a column in a table

```
ALTER TABLE table_name  
ALTER COLUMN column_name datatype
```

```
ALTER TABLE customers  
ALTER COLUMN birthday year
```

Delete a column

```
ALTER TABLE table_name  
DROP COLUMN column_name
```

```
ALTER TABLE customers  
DROP COLUMN DriverL
```

Drop Table

Syntax

Drop a table (or Remove)

```
DROP TABLE table_name
```

```
DROP TABLE customers
```

SQL Reference

The screenshot shows a web browser displaying the [w3schools.com/sql/](https://www.w3schools.com/sql/) page. The page title is "SQL Tutorial". The left sidebar has a green header "SQL HOME" containing a list of SQL topics. The main content area features a large heading "SQL Tutorial" with navigation buttons "Home" and "Next". Below the heading is a green box with text about SQL being a standard language for databases and its use in various systems. A button "Start learning SQL now »" is present. The main content section is titled "Examples in Each Chapter" and includes a "Try it Yourself" button and a "Get your own SQL Server" button. The footer features a "Modern Web Development" section with icons for HTML, CSS, and JS.

SQL Tutorial

SQL HOME

- SQL Intro
- SQL Syntax
- SQL Select
- SQL Select Distinct
- SQL Where
- SQL Order By
- SQL And
- SQL Or
- SQL Not
- SQL Insert Into
- SQL Null Values
- SQL Update
- SQL Delete
- SQL Select Top
- SQL Aggregate Functions
- SQL Min and Max
- SQL Count
- SQL Sum
- SQL Avg
- SQL Like
- SQL Wildcards
- SQL In
- SQL Between
- SQL Aliases
- SQL Joins
- SQL Inner Join
- SQL Left Join
- SQL Right Join
- SQL Full Join

SQL is a standard language for storing, manipulating and retrieving data in databases.

Our SQL tutorial will teach you how to use SQL in: MySQL, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres, and other database systems.

Start learning SQL now »

Examples in Each Chapter

With our online SQL editor, you can edit the SQL statements, and click on a button to view the result.

Example

```
SELECT * FROM Customers;
```

Get your own SQL Server

Try it Yourself »

Modern Web Development

HTML CSS JS

<https://www.w3schools.com/sql/>

CodeIgniter – How does it help with Data Access?

- Using a Web Framework should make your life as a developer easier
- A Framework will include a number of abstractions to help you write code to interact with a database
- CodeIgniter 4 has numerous features:
 - Central **config file (.env)** to store database connections
 - **Migrations** to help you create tables and have **database version control**
 - **Models** to simplify **reading, inserting, updating and deleting** content from a database table (just use simple objects instead of using SQL)
 - The **Spark** command line to help you create Migration files and new model files.
 - Very competitive with other frameworks (e.g. Django)

Adding Database Config in CodeIgniter 4

Add database details to the .env file

- Main steps on UQCloud
 - Setup the mysql service
 - sudo webprojctl enable mysql
 - Get MySQL Password
 - sudo mdata-get mysql_pw
 - Setup in CodeIgniter .env file
 - Remember to uncomment by removing the #

```
cidemo > ⚙ .env
 8  # By default, all of the settings are commented out. If you want
 9  # to override the setting, you must un-comment it by removing the '#'
10 # at the beginning of the line.
11 #
12 #
13 #
14 # ENVIRONMENT
15 #
16 #
17 CI_ENVIRONMENT = development
18 #
19 #
20 # APP
21 #
22 #
23 app.baseURL = 'https://infs32022024v2.uqcloud.net/cidemo/'
24 # If you have trouble with `.', you could also use `_'.
25 # app_baseURL = ''
26 # app.forceGlobalSecureRequests = false
27 # app.CSPEnabled = false
28 #
29 #
30 # DATABASE
31 #
32 #
33 database.default.hostname = localhost
34 database.default.database = resume_builder
35 database.default.username = yourusername
36 database.default.password = yourdbserverpassword
37 database.default.DBDriver = MySQLi
38 # database.default.DBPrefix =
39 database.default.port = 3306
40
```

Creating Migrations in CodeIgniter 4

What is a Migration?

- **Versioned Database Schemas:**

Migrations in CodeIgniter 4 provide a method for version controlling your database schema. This allows for easy management of database changes over time.

- **Automated Database Changes:**

They automate the process of applying database changes, such as creating or altering tables, adding indexes, and more, ensuring consistency across different environments.

- **Simplifies Development and Deployment Workflow:**

Migrations simplify the development workflow by allowing team members to share and apply database changes in a controlled and systematic manner. Easy to deploy the schema of a database

- **Programmatic Schema Manipulation:**

Migrations are PHP classes that contain methods to programmatically execute database operations, giving developers a powerful tool to manipulate the schema.

CodeIgniter – What is spark?

- In CodeIgniter 4, “spark” is a command-line tool that facilitates the development and management of CodeIgniter applications.
 - Running database migrations and seeders to manage database schema and initial data setup
 - Starting a development web server
 - Generating boilerplate code (new files) for new controllers, models, and other components.
 - Listing routes, displaying current environment settings, and more, providing insights into the application configuration

```
uqabakh1@uqabakh1-9893 codeigniter test % php spark
CodeIgniter v4.4.4 Command Line Tool - Server Time: 2024-03-02 09:04:45 UTC+00:00

Cache
  cache:clear      Clears the current system caches.
  cache:info       Shows file cache information in the current system.

CodeIgniter
  env             Retrieves the current environment, or set a new one.
  filter:check    Check filters for a route.
  help            Displays basic usage information.
  list             Lists the available commands.
  namespaces      Verifies your namespaces are setup correctly.
  publish          Discovers and executes all predefined Publisher classes.
  routes           Displays all routes.
  serve            Launches the CodeIgniter PHP-Development Server.

Database
  db:create        Create a new database schema.
  db:seed           Runs the specified seeder to populate known data into the database.
  db:table          Retrieves information on the selected table.
  migrate          Locates and runs all new migrations against the database.
  migrate:refresh  Does a rollback followed by a latest to refresh the current state of the database.
  migrate:rollback Runs the "down" method for all migrations in the last batch.
  migrate:status   Displays a list of all migrations and whether they've been run or not.

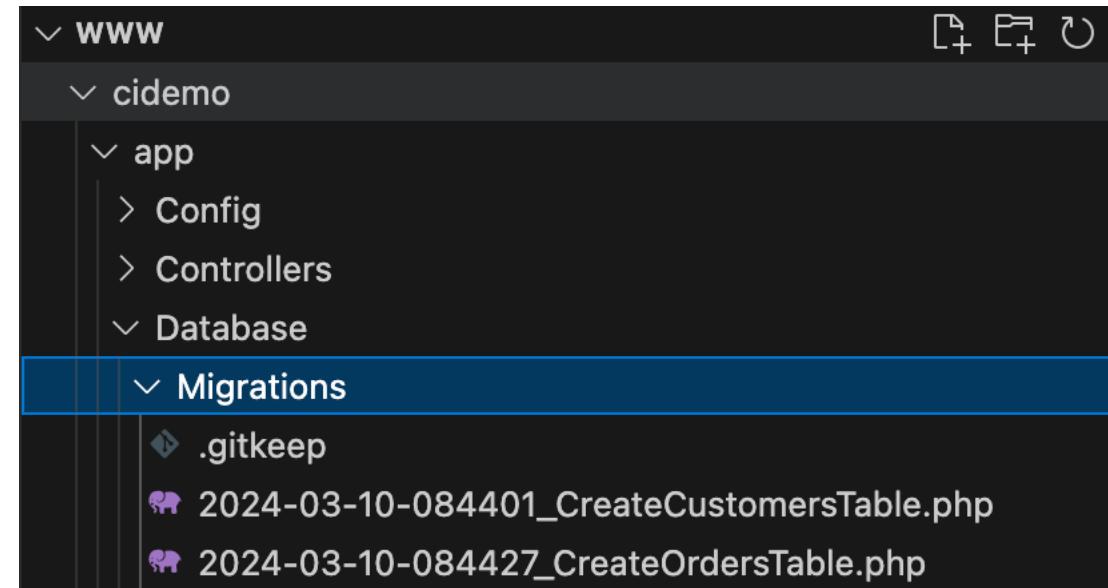
Encryption
  key:generate     Generates a new encryption key and writes it in an '.env' file.

Generators
  make:cell         Generates a new Controlled Cell file and its view.
  make:command     Generates a new spark command.
  make:config       Generates a new config file.
  make:controller  Generates a new controller file.
  make:entity      Generates a new entity file.
  make:filter       Generates a new filter file.
  make:migration   Generates a new migration file.
  make:model        Generates a new model file.
  make:scaffold    Generates a complete set of scaffold files.
  make:seeder       Generates a new seeder file.
  make:validation  Generates a new validation file.
  migrate:create   [DEPRECATED] Creates a new migration file. Please use "make:migration" instead.
  session:migration [DEPRECATED] Generates the migration file for database sessions, Please use "make:migration --session" instead.

Housekeeping
  debugbar:clear   Clears all debugbar JSON files.
  logs:clear       Clears all log files.
```

Create a Migration

- Use the spark make:migration command
 - `php spark make:migration create_customers_table`
 - `php spark make:migration create_orders_table`
- This will create a new file in the app/Databases/Migrations folder
 - The file will have a data time stamp and be versioned
 - You need to add the fields that need to be created in the table
 - The up() method defines the schema for creating the tables.
 - The down() method defines the schema for dropping the tables.



Create a Migration – The Forge Class

The screenshot shows a web browser displaying the CodeIgniter documentation at codeigniter4.github.io/CodeIgniter4/dbmgmt/forge.html. The page title is "Database Forge Class". The left sidebar contains a navigation menu with sections like Welcome to CodeIgniter4, Installation, Build Your First Application, CodeIgniter4 Overview, General Topics, Controllers and Routing, Building Responses, Working with Databases, Modeling Data, Managing Databases (which is expanded), Database Manipulation with Database Forge (which is also expanded), Library Reference, Helpers, Testing, Command Line Usage, Extending CodeIgniter, and Official Packages. The main content area starts with a brief description: "The Database Forge Class contains methods that help you manage your database." followed by a bulleted list of methods:

- Initializing the Forge Class
- Creating and Dropping Databases
 - \$forge->createDatabase('db_name')
 - \$forge->dropDatabase('db_name')
 - Creating Databases in the Command Line
- Creating Tables
 - Adding Fields
 - Adding Keys
 - Adding Foreign Keys
 - Creating a Table
- Dropping Tables
 - Dropping a Table
- Modifying Tables
 - Adding a Field to a Table
 - Dropping Fields From a Table
 - Modifying a Field in a Table
 - Adding Keys to a Table
 - Dropping a Primary Key
 - Dropping a Key
 - Dropping a Foreign Key
 - Renaming a Table
- Class Reference

- Reference for making field types

Create a Migration – The Customers Table

- Fields for the Customers Table

```
cidemo > app > Database > Migrations > 2024-03-10-084401_CreateCustomersTable.php
1  <?php
2
3  namespace App\Database\Migrations;
4
5  use CodeIgniter\Database\Migration;
6
7  class CreateCustomersTable extends Migration
8  {
9      public function up()
10     {
11         $this->forge->addField([
12             'id' => [
13                 'type' => 'INT',
14                 'constraint' => 11,
15                 'unsigned' => true,
16                 'auto_increment' => true,
17             ],
18             'name' => [
19                 'type' => 'VARCHAR',
20                 'constraint' => 255,
21             ],
22             'email' => [
23                 'type' => 'VARCHAR',
24                 'constraint' => 255,
25             ],
26         ]);
27         $this->forge->addKey('id', true);
28         $this->forge->createTable('customers');
29     }
30
31     public function down()
32     {
33         $this->forge->dropTable('customers');
34     }
35 }
36
```

Create a Migration – The Orders Table

- Fields for the Orders Table

```
cidemo > app > Database > Migrations > 2024-03-10-084427_CreateOrdersTable.php
1  <?php
2
3  namespace App\Database\Migrations;
4
5  use CodeIgniter\Database\Migration;
6
7  class CreateOrdersTable extends Migration
8  {
9      public function up()
10     {
11         $this->forge->addField([
12             'id' => [
13                 'type' => 'INT',
14                 'constraint' => 11,
15                 'unsigned' => true,
16                 'auto_increment' => true,
17             ],
18             'customer_id' => [
19                 'type' => 'INT',
20                 'constraint' => 11,
21                 'unsigned' => true,
22             ],
23             'order_date' => [
24                 'type' => 'DATE',
25             ],
26             'total_amount' => [
27                 'type' => 'DECIMAL',
28                 'constraint' => '10,2',
29             ],
30         ]);
31         $this->forge->addKey('id', true);
32         $this->forge->addForeignKey('customer_id', 'customers', 'id', 'CASCADE', 'CASCADE');
33         $this->forge->createTable('orders');
34     }
35
36     public function down()
37     {
38         $this->forge->dropTable('orders');
39     }
40 }
```

Run the Migration

- Use Spark to run the Migrations
 - `php spark migrate`
 - This will run and execute create table sql specific for your database.
- Use PHPMyAdmin to check that the tables have been created

- There will also be a Migrations table. This is where the versioning of the migrations are stored.
- In Lab 3 you will learn how to use migrations to add data, but in the demo we'll run out insert statements.

The screenshot shows the phpMyAdmin interface for the 'cicustomers' database. The left sidebar displays the database structure with several tables: customers, migrations, orders, and information_schema. The main panel shows a table of three tables: customers, migrations, and orders. The 'customers' table has 0 rows, is InnoDB, and uses utf8mb3_general_ci collation. The 'migrations' table has 2 rows, is InnoDB, and uses utf8mb3_general_ci collation. The 'orders' table has 0 rows, is InnoDB, and uses utf8mb3_general_ci collation. A 'Create new table' dialog is open at the bottom, prompting for a table name (with a placeholder 'customers') and the number of columns (set to 4), with a 'Create' button.

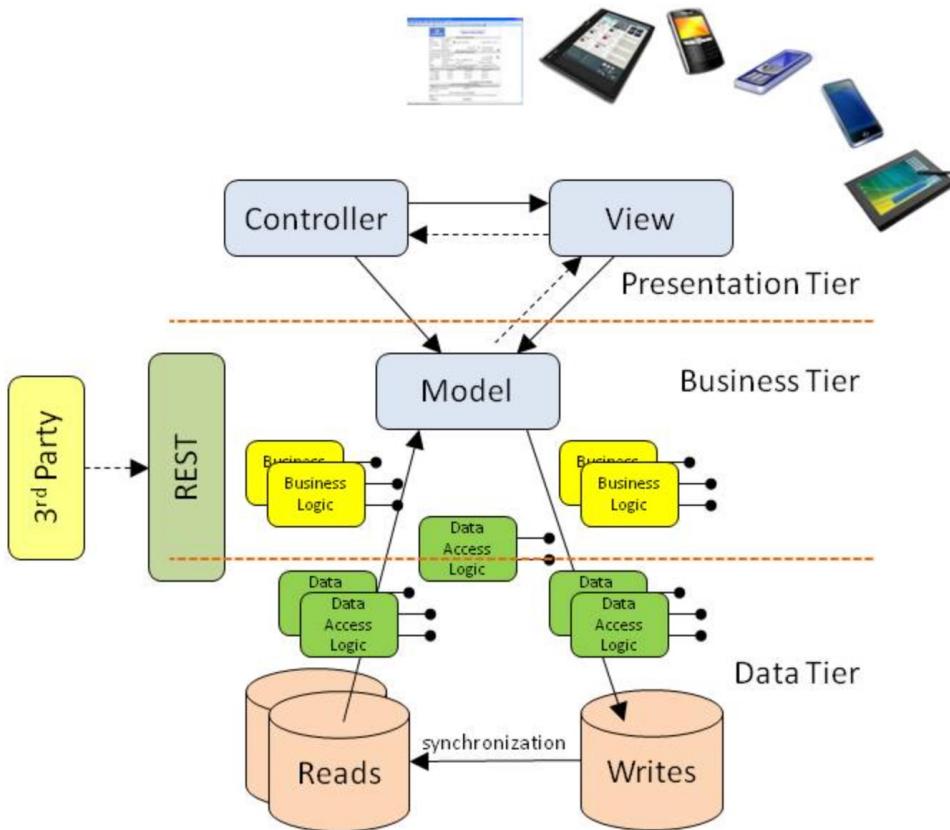
Table	Action	Rows	Type	Collation	Size	Overhead
customers	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb3_general_ci	16.0 KiB	-
migrations	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb3_general_ci	16.0 KiB	-
orders	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb3_general_ci	32.0 KiB	-
3 tables Sum		2	InnoDB	utf8mb4_0900_ai_ci	64.0 KiB	0 B

How does this help developer workflow?

- A few scenarios
 - You develop locally and then deploy to production:
 - You can just run the migration when you deploy and update your application
 - You have multiple developers
 - When a database change is made, the code for the change is committed to the code repository
 - Other developers can then run the migration to update their local databases
 - You release open source software
 - The same migration code and create SQL create and modify statements for multiple databases
 - Each developer that downloads your software can choose their own database server (e.g. Postgres) and not worry about conflicts

Codelgniter Models

Model View Controller (MVC)

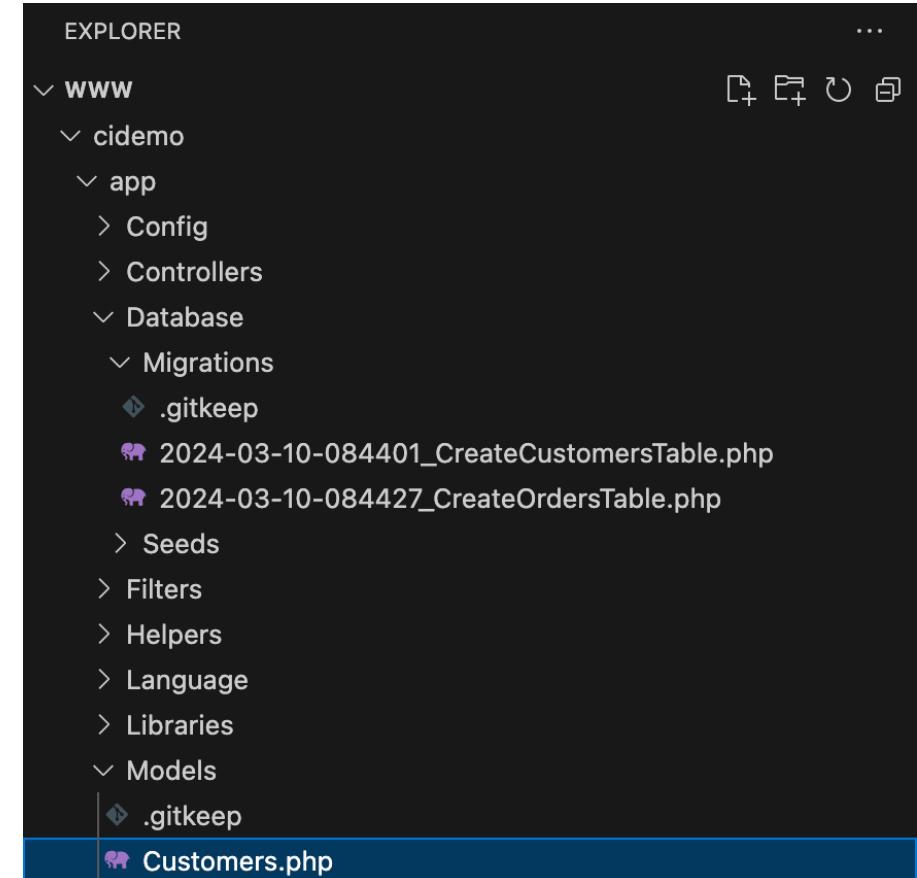


The MVC pattern is a specialized form of 3-layer architecture, making it highly relevant to modern web development practices

Image from https://en.wikiversity.org/wiki/Three-Tier_Architecture#/media/File:MVC3tierArchitecture.jpg

Codelgniter Models

- The Codelgniter's Model offers convenient features and additional functionality to make working with a single table in the database easier.
 - automatic database connection;
 - basic Create Read Update Delete methods;
 - in-model validation;
 - automatic pagination;
 - and more
- Models in Codelgniter 4 are stored in the app/Models directory
- Use Spark to create a new model file e.g.
php spark make:model Customers
- A new file called Customers.php is placed in app/Models
- To access models within your classes, you can create a new instance or use the model() helper function.

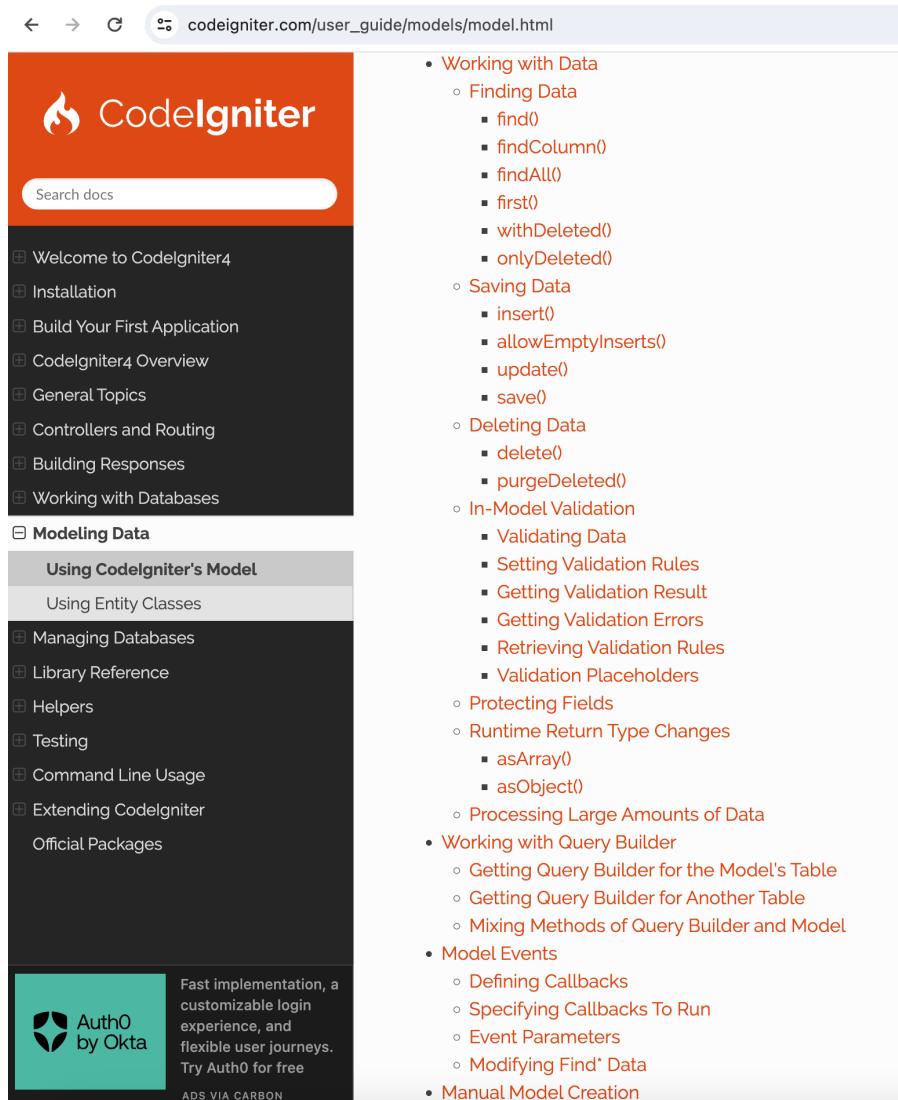


Creating a Model Class

- **Model Configuration:** The model class provides configuration options to facilitate the smooth functioning of its methods.
 - **\$table:** Specifies the database table that this model primarily works with.
 - **\$primaryKey:** Specifies the name of the column that uniquely identifies the records in this table.
 - **\$useAutoIncrement:** Specifies if the table uses an auto-increment feature for \$primaryKey.
 - **\$returnType:** This setting allows you to define the type of data that is returned.
 - **\$allowedFields:** This array should be updated with the field names that can be set during save(), insert(), or update() methods. A

```
cidemo > app > Models > 🐘 Customers.php
1  <?php
2
3  namespace App\Models;
4
5  use CodeIgniter\Model;
6
7  class Customers extends Model
8  {
9      protected $table          = 'customers';
10     protected $primaryKey       = 'id';
11     protected $useAutoIncrement = true;
12     protected $returnType        = 'array';
13     protected $useSoftDeletes    = false;
14     protected $protectFields     = true;
15     protected $allowedFields     = [];
16
17 }
```

Models use Object Oriented Syntax



The screenshot shows a web browser displaying the CodeIgniter User Guide for Models. The URL in the address bar is `https://codeigniter.com/user_guide/models/model.html`. The page has a dark-themed sidebar on the left containing navigation links such as Welcome to CodeIgniter4, Installation, Build Your First Application, CodeIgniter4 Overview, General Topics, Controllers and Routing, Building Responses, Working with Databases, and Modeling Data. Under Modeling Data, there is a section titled "Using CodeIgniter's Model" which includes links for Using Entity Classes, Managing Databases, Library Reference, Helpers, Testing, Command Line Usage, and Extending CodeIgniter. Below this section is a link for Official Packages. The main content area on the right lists various topics under the heading "Working with Data", including Finding Data, Saving Data, Deleting Data, In-Model Validation, Protecting Fields, Runtime Return Type Changes, Processing Large Amounts of Data, Working with Query Builder, Model Events, and Manual Model Creation. Each topic has a corresponding list of methods or steps. At the bottom of the sidebar, there is an advertisement for Auth0 by Okta.

- Working with Data
 - Finding Data
 - find()
 - findColumn()
 - findAll()
 - first()
 - withDeleted()
 - onlyDeleted()
 - Saving Data
 - insert()
 - allowEmptyInserts()
 - update()
 - save()
 - Deleting Data
 - delete()
 - purgeDeleted()
 - In-Model Validation
 - Validating Data
 - Setting Validation Rules
 - Getting Validation Result
 - Getting Validation Errors
 - Retrieving Validation Rules
 - Validation Placeholders
 - Protecting Fields
 - Runtime Return Type Changes
 - asArray()
 - asObject()
 - Processing Large Amounts of Data
- Working with Query Builder
 - Getting Query Builder for the Model's Table
 - Getting Query Builder for Another Table
 - Mixing Methods of Query Builder and Model
- Model Events
 - Defining Callbacks
 - Specifying Callbacks To Run
 - Event Parameters
 - Modifying Find* Data
- Manual Model Creation

https://codeigniter.com/user_guide/models/model.html

Models use Object Oriented Syntax

To support Create, Read, Update, Delete operations, several functions are provided:

- **Finding data:**

- find(): returns a single row given a primary key

```
$user = $userModel->find($user_id);
//You can specify more than one row to return
//by passing an array of primaryKey values instead of just one:
$users = $userModel->find([1, 2, 3]);
```

- findColumn()
- findAll()
- ...

```
$user = $userModel->findColumn($column_name);
$users = $userModel->findAll();
```

Models use Object Oriented Syntax

To support Create, Read, Update, Delete operations, several functions are provided:

- Saving data:

- insert(): inserts multiple rows of data

```
$data = [  
    'username' => 'darth',  
    'email'     => 'd.vader@theempire.com',  
];  
// Inserts data and returns inserted row's primary key  
$userModel->insert($data);  
  
// Inserts data and returns true on success and false on failure  
$userModel->insert($data, false);  
  
// Returns inserted row's primary key  
$userModel->getInsertID();  
}
```

The first parameter is an associative array of data to create a new row of data in the database. If an object is passed instead of an array, it will attempt to convert it to an array.

The second parameter is an optional boolean type. If set to false, the method will return a boolean value that indicates the success or failure of the query.

Models use Object Oriented Syntax

To support Create, Read, Update, Delete operations, several functions are provided:

- **Saving data:**

- `save()`: automatically handles **inserting** or **updating** a record based on whether it finds an array key matching the primary key value.

```
<?php
// Defined as a model property
primaryKey = 'id';
// Does an insert()
$data = [
    'username' => 'sen',
    'email'     => 'sen.wang@uq.edu.au',
];

$userModel->save($data);

// Performs an update, since the primary key, 'id', is found.
$data = [
    'id'        => 3,
    'username' => 'tom',
    'email'     => 'tom.zhang@uq.edu.au',
];
$userModel->save($data);
```

Question: What if the "id" =3 for username="sen"?

Models use Object Oriented Syntax

To support Create, Read, Update, Delete operations, several functions are provided:

- **Saving data:**

- update(): updates an existing record in the database. Must specify the primary key.

```
$data = [
    'username' => 'sen',
    'email'     => 'sen.wang@uq.edu.au',
];

$userModel->update($id, $data);|
```

- Can update multiple records, give

```
$data = [
    'fulltime' => 1,
];

$userModel->update([1, 2, 3], $data);|
```

Models use Object Oriented Syntax

To support Create, Read, Update, Delete operations, several functions are provided:

- **Deleting data:**
 - `delete()`: Takes a primary key value(s) as the first parameter and deletes the matching record from the model's table.

```
$userModel->delete(10);  
$userModel->delete([1, 2, 3]);
```

- `purgeDeleted()`: Cleans out the database table by permanently removing all rows that have 'deleted_at IS NOT NULL'.

```
$userModel->purgeDeleted();
```

Raw SQL & QueryBuilder Class

- We can use `query()` function to pass in real SQL scripts as a parameter.

```
$db = db_connect();
$db->query('SELECT * FROM STUDENTS');
```

- Builder can help you generate the raw SQL scripts, which will be sent to `query()`.
- To load the Query Builder, use the `table()` method on the database connection.
- It's important to note that the Query Builder needs to be explicitly loaded if you require.
- Get Data(i.e. `SELECT *:`) `$builder->get()`

Raw SQL – Use QueryBuilder Class

- **From:** \$builder->from('table')

```
$builder = $db->table('users');
$builder->select('username, id, coursename');
$builder->from('courses');
$query = $builder->get();
// Produces: SELECT title, content, date FROM users, courses|
```

- **Join:** \$builder->join()

```
$builder = $db->table('users');
$builder->select('username, id, coursename');
$builder->join('course_enrolment','users.id = course_enrolment.id')
$query = $builder->get();
// Produces: SELECT username, id, coursename JOIN course_enrolment ON
// users.id = course_enrolment.id
```

Raw SQL – Use QueryBuilder Class

- CodeIgniter Model has one instance of the Query Builder for that model's database connection. You can get access to the shared instance of the Query Builder any time you need it:

```
$builder = $userModel->builder();
```

- Once you get the Query Builder instance, you can call methods of the Query Builder. However, since Query Builder is not a Model, you cannot call methods of the Model.

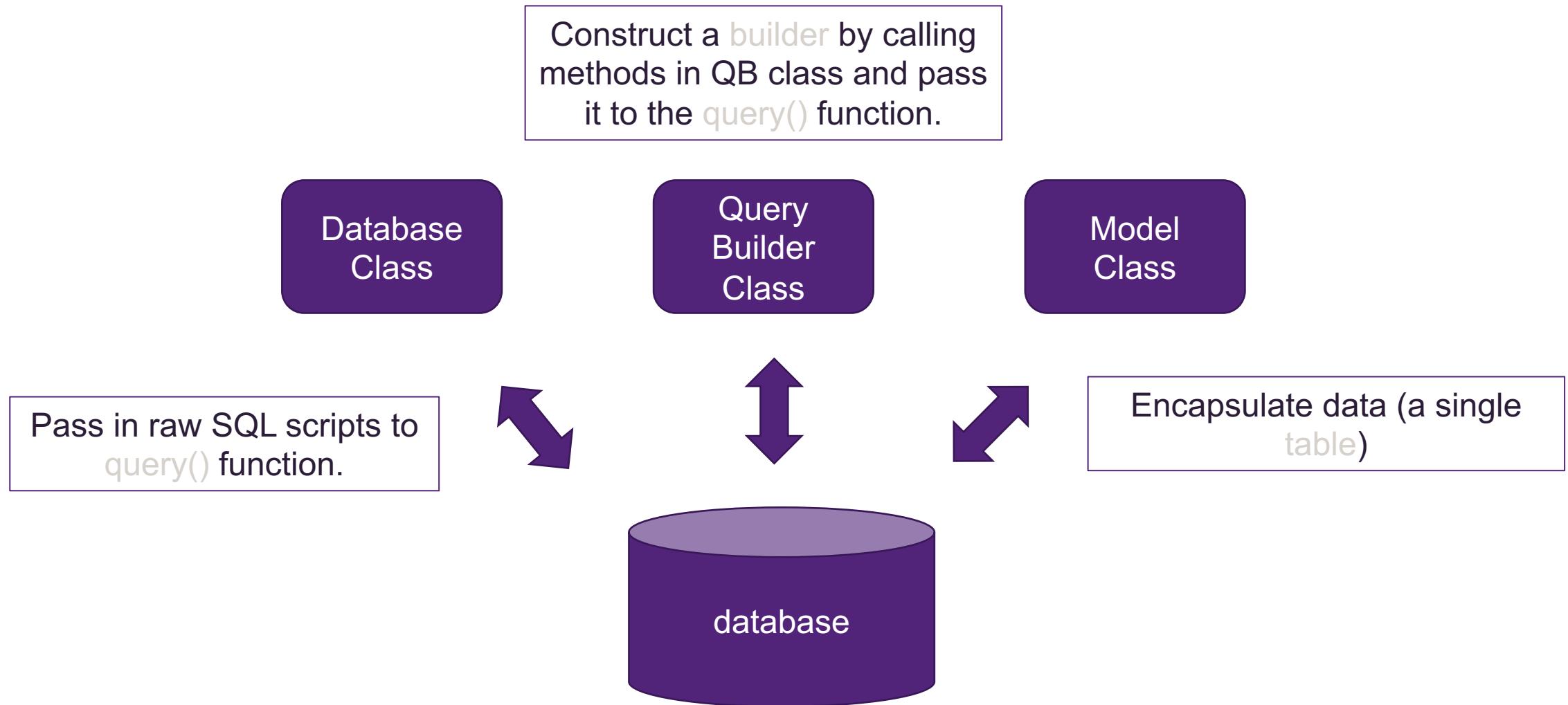
```
$customerBuilder = $userModel->builder('customers');
```

- You can also use Query Builder methods and the Model's CRUD methods in the same chained call, allowing for very elegant use:

```
$users = $userModel->where('status', 'active')
    ->orderBy('last_login', 'asc')
    ->findAll();
```

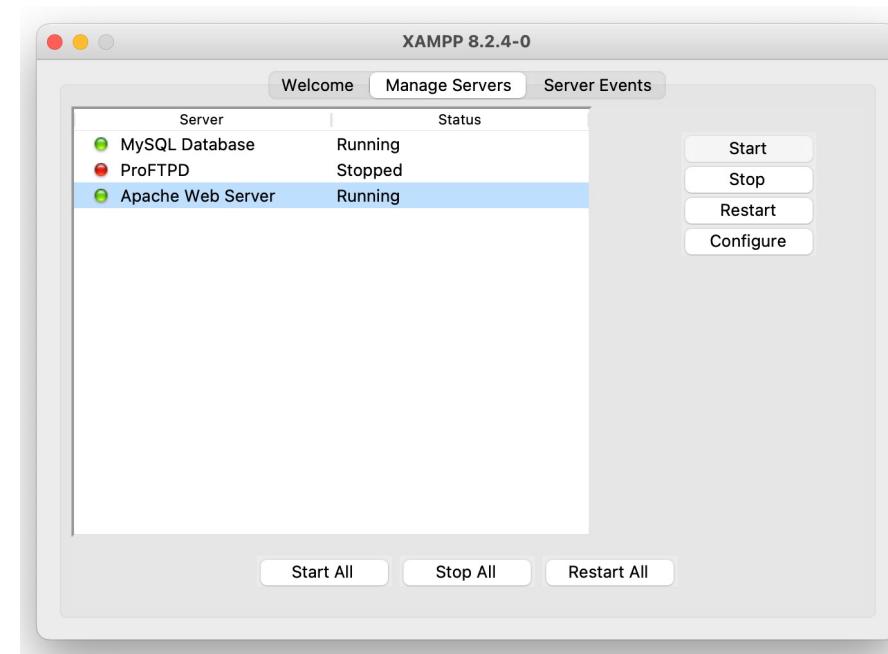
In this case, it operates on the shared instance of the Query Builder held by the model.

CodeIgniter Database Access



Local Database Options

- If you installed XAMPP
 - <https://www.apachefriends.org/>
 - You get both MySQL and PHPMyAdmin
- You can also use Sqllite
 - It is a file system database
 - Just change the .env
- MySQL Workbench
 - <https://www.mysql.com/products/workbench/>
 - Install locally and access your UQCloud Zone MySQL server

A screenshot of the MySQL Workbench interface. At the top, it shows the URL "localhost/phpmyadmin/" in the address bar. The main area is divided into several panes:

- Left pane:** Shows a tree view of databases: New, information_schema, mysql, performance_schema, phpmyadmin, and test.
- General settings (top right):** Server connection collation: utf8mb4_unicode_ci, with a "More settings" link.
- Appearance settings (bottom right):** Language: English, Theme: pmahomme, with a "View all" link.
- Database server (right sidebar):**
 - Server: Localhost via UNIX socket
 - Server type: MariaDB
 - Server connection: SSL is not being used
 - Server version: 10.4.28-MariaDB - Source distribution
 - Protocol version: 10
 - User: root@localhost
 - Server charset: UTF-8 Unicode (utf8mb4)
- Web server (bottom right sidebar):**
 - Apache/2.4.56 (Unix) OpenSSL/1.1.1t PHP/8.2.4 mod_perl/2.0.12 Perl/v5.34.1
 - Database client version: libmysql - mysqlnd 8.2.4
 - PHP extension: mysqli curl mbstring
 - PHP version: 8.2.4

Code Walkthrough: Creating a Customer Search Page

Online Store

Customers

Search Name

ID	Name	Email
1	Raj Patel	raj@example.com
2	Wei Chen	wei@example.com
3	Emma Müller	emma@example.com
4	Priya Gupta	priya@example.com
5	Liam Smith	liam@example.com

© 2024 Online Store. All rights reserved.

- Could you improve the search?
E.g. Allow searches for partial names with %

Designing a Database

Database Design Goals

- Each table must represents a single subject
- No data item will be unnecessarily stored in more than one table, i.e., No data redundancy
- All non-key attributes in a table are dependent on the primary key
- Each table is void of insertion, update, deletion anomalies
- Objective of normalisation is to ensure that all tables are in at least 3rd Normal Form (3NF)

Let's look at an example

StudentID	StudentName	CourseID	CourseName	CourseCredits	InstructorName
1	John Doe	CS101	Programming	3	Dr. Smith
1	John Doe	MAT201	Calculus	4	Prof. Johnson
2	Jane Smith	CS101	Programming	3	Dr. Smith
3	Michael Lee	CS101	Programming	3	Dr. Smith
3	Michael Lee	PHY301	Physics	4	Dr. Brown

- We'll walk through the process of normalizing a single table with duplicates to 3rd Normal Form (3NF) and explain the different types of relationships that emerge.

Let's look at an example

StudentID	StudentName	CourseID	CourseName	CourseCredits	InstructorName
1	John Doe	CS101	Programming	3	Dr. Smith
1	John Doe	MAT201	Calculus	4	Prof. Johnson
2	Jane Smith	CS101	Programming	3	Dr. Smith
3	Michael Lee	CS101	Programming	3	Dr. Smith
3	Michael Lee	PHY301	Physics	4	Dr. Brown

- In this table, we can observe data duplication. The student names and course details are repeated for each enrollment. This design violates the principles of normalization and can lead to data anomalies and inconsistencies.

Step 1: First Normal Form (1NF)

- To achieve 1NF, we need to ensure that each column contains atomic values and there are no repeating groups.
- In this case, we can split the table into two separate tables: "Students" and "Enrollments".

Students Table:

StudentID	StudentName
1	John Doe
2	Jane Smith
3	Michael Lee

Enrollments Table:

EnrollmentID	StudentID	CourseID	CourseName	CourseCredits	InstructorName
1	1	CS101	Programming	3	Dr. Smith
2	1	MAT201	Calculus	4	Prof. Johnson
3	2	CS101	Programming	3	Dr. Smith
4	3	CS101	Programming	3	Dr. Smith
5	3	PHY301	Physics	4	Dr. Brown

Step 2: Second Normal Form (2NF)

- To achieve 2NF, we need to ensure that all non-key columns depend on the entire primary key.
- In the "Enrollments" table, the primary key is a combination of "StudentID" and "CourseID".
- However, the course details (CourseName, CourseCredits, InstructorName) depend only on the "CourseID".
- To resolve this, we can split the "Enrollments" table further into "Enrollments" and "Courses" tables.

Enrollments Table:

EnrollmentID	StudentID	CourseID
1	1	CS101
2	1	MAT201
3	2	CS101
4	3	CS101
5	3	PHY301

Courses Table:

CourseID	CourseName	CourseCredits	InstructorName
CS101	Programming	3	Dr. Smith
MAT201	Calculus	4	Prof. Johnson
PHY301	Physics	4	Dr. Brown

Step 3: Third Normal Form (3NF)

- To achieve 3NF, we need to ensure that there are no transitive dependencies. In the "Courses" table, the "InstructorName" depends on the "CourseID" but not directly. It depends on the instructor associated with the course. To resolve this, we can split the "Courses" table into "Courses" and "Instructors" tables.

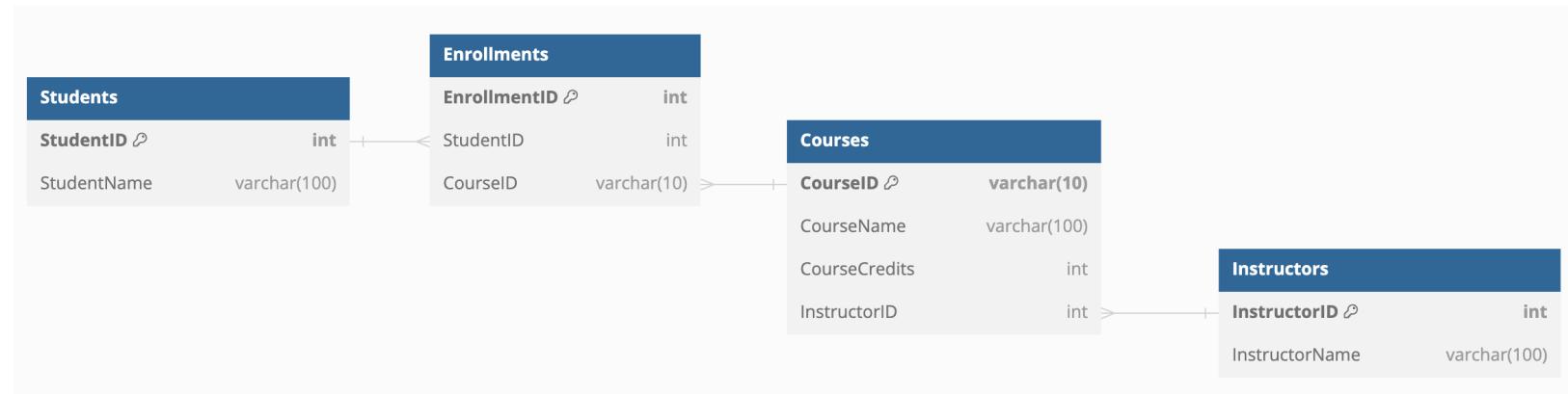
Courses Table:

CourseID	CourseName	CourseCredits	InstructorID
CS101	Programming	3	1
MAT201	Calculus	4	2
PHY301	Physics	4	3

Instructors Table:

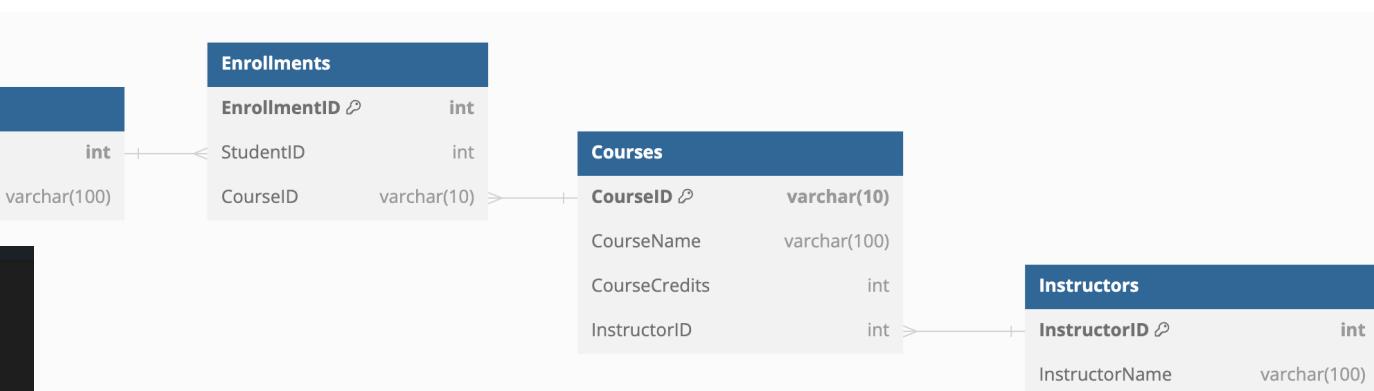
InstructorID	InstructorName
1	Dr. Smith
2	Prof. Johnson
3	Dr. Brown

The 3NF Schema



The 3NF Schema – Created with DBML

```
1 // Use DBML to define your database structure
2 // Docs: https://dbml.dbdiagram.io/docs
3
4 Table Students {
5     StudentID int [pk]
6     StudentName varchar(100)
7 }
8
9 Table Courses {
10    CourseID varchar(10) [pk]
11    CourseName varchar(100)
12    CourseCredits int
13    InstructorID int [ref: > Instructors.InstructorID]
14 }
15
16 Table Instructors {
17     InstructorID int [pk]
18     InstructorName varchar(100)
19 }
20
21 Table Enrollments {
22     EnrollmentID int [pk]
23     StudentID int [ref: > Students.StudentID]
24     CourseID varchar(10) [ref: > Courses.CourseID]
25 }
```



- Represent the table in Database Markup Language (DBML)
- Rendered visually in <https://www.dbdiagram.io>
- You could also use Kroki.io or PHPMyAdmin or any other diagramming tool.

Week 4: Todo

- Weekly RiPPL E task is due in Week 4 on Friday at 3pm
- Complete Week 4 Lab 3
 - It is a challenging lab but by the end you will have created a database with multiple tables and a results-detail set of pages
- Continue with your Design Document Assessment Item
 - You should now be starting with the database design



CREATE CHANGE

Q&A



CREATE CHANGE

Thank you