



ACCA5026 – DATABASE & APPLICATION DEVELOPMENT

Database Design & Implementation



AUGUST 14, 2026

STUDENT: ANDREI-STEFANITA STOICA (2413927)

Lecturer: Dr. Fahad Ahmad

Contents

1. Introduction	2
2. Part A – Database Design & Implementation	2
2.1 Database Design	2
2.1.1 Entities and Attributes	2
2.1.2 Relationships and ERD	4
2.1.3 Normalization	5
2.2 Database Implementation	5
2.2.1 Table Creation and Constraints	5
2.2.2 Sample Data Insertion.....	8
2.2.3 Stored Procedures, Triggers, and Functions	11
2.2.4 Transaction Handling (BEGIN, COMMIT, ROLLBACK).....	13
2.2.5 Performance Tuning & Indexing	16
2.3 Justification and Literature Support	17
3. Part B – PHP Website Development	18
3.1 Overview of Website Topic.....	18
3.2 Database Design for Website	18
3.2.1 Tables and Relationships	18
3.2.2 Sample Data	19
3.3 PHP Implementation	20
3.3.1 Dynamic Content Display	20
3.3.2 CRUD Operations	20
3.3.3 Search/Filter Functionality.....	21
3.3.4 Security Measures (Login, Input Validation).....	21
3.4 SQL Injection Overview	22
3.4.1 Explanation and Examples.....	22
3.4.2 Vulnerabilities in Website Code.....	23
3.4.3 Anti-SQL Injection Measures Implemented	23
3.4.4 Screenshots of Testing and Results	24
4. Conclusion and Reflection.....	27
5. References	29
6. Supporting Material	29

1. Introduction

This report presents the design and implementation of a relational database prototype for MedAssist Clinic, a private healthcare facility in the UK offering outpatient medical services. As the clinic modernizes its operations to digitize patient records, appointment scheduling and treatment histories, this database solution provides a secure, multi-user friendly system which enforces data integrity principles. The primary objective was to create a normalized database schema supporting patient and doctor management, department organization, appointment scheduling, treatment recording and feedback collection complete with transaction handling mechanisms and performance optimizations. This prototype serves as the foundation for a comprehensive clinic management system which improves service efficiency and patient care coordination.

2. Part A – Database Design & Implementation

2.1 Database Design

2.1.1 Entities and Attributes

The database consists of six core entities with the following attributes:

1. DEPARTMENT

- `department_id` (INT, PK, AI)
- `name` (VARCHAR(100), NOT NULL, UNIQUE)
- `description` (TEXT)
- `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)
- `updated_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)

2. DOCTOR

- `doctor_id` (INT, PK, AI)
- `first_name` (VARCHAR(50), NOT NULL)
- `last_name` (VARCHAR(50), NOT NULL)
- `specialty` (VARCHAR(100))
- `phone` (VARCHAR(20))
- `email` (VARCHAR(100), UNIQUE)
- `department_id` (INT, FK)
- `hire_date` (DATE, NOT NULL)

- `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)
- `updated_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)

3. PATIENT

- `patient_id` (INT, PK, AI)
- `first_name` (VARCHAR(50), NOT NULL)
- `last_name` (VARCHAR(50), NOT NULL)
- `date_of_birth` (DATE, NOT NULL)
- `gender` (ENUM('Male','Female','Other'), NOT NULL)
- `phone` (VARCHAR(20))
- `email` (VARCHAR(100), UNIQUE)
- `address` (TEXT)
- `registration_date` (DATE, NOT NULL)
- `emergency_contact_name` (VARCHAR(100))
- `emergency_contact_phone` (VARCHAR(20))
- `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)
- `updated_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)

4. APPOINTMENT

- `appointment_id` (INT, PK, AI)
- `patient_id` (INT, FK, NOT NULL)
- `doctor_id` (INT, FK, NOT NULL)
- `appointment_date` (DATE, NOT NULL)
- `appointment_time` (TIME, NOT NULL)
- `reason_for_visit` (TEXT, NOT NULL)
- `status` (ENUM('scheduled','completed','cancelled','no-show'), DEFAULT 'scheduled')
- `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)
- `updated_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)

5. TREATMENT

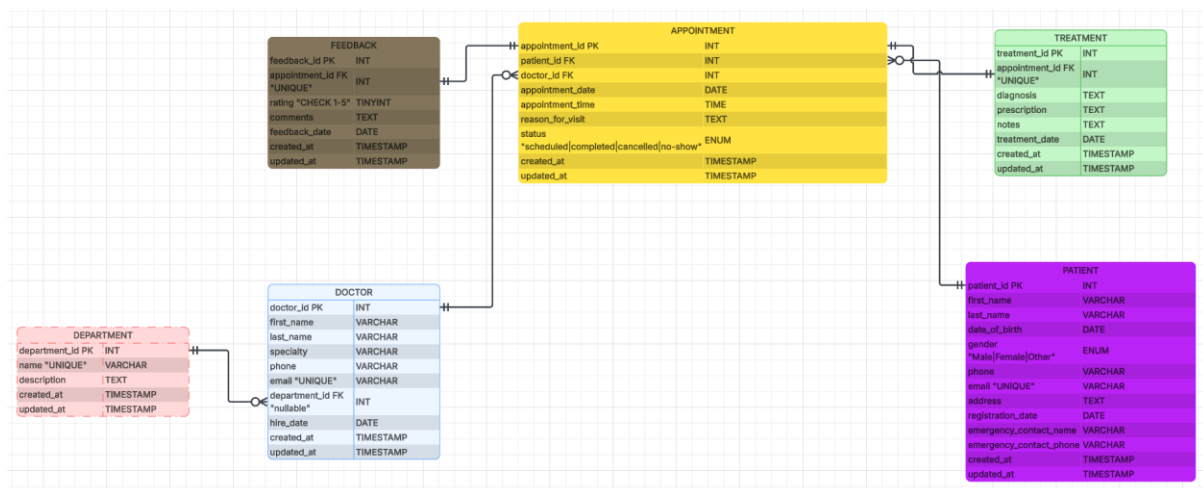
- `treatment_id` (INT, PK, AI)
- `appointment_id` (INT, FK, NOT NULL, UNIQUE)
- `diagnosis` (TEXT, NOT NULL)
- `prescription` (TEXT)
- `notes` (TEXT)
- `treatment_date` (DATE, NOT NULL)
- `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)

- `updated_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)

6. FEEDBACK

- `feedback_id` (INT, PK, AI)
- `appointment_id` (INT, FK, NOT NULL, UNIQUE)
- `rating` (TINYINT, NOT NULL, CHECK between 1-5)
- `comments` (TEXT)
- `feedback_date` (DATE, NOT NULL)
- `created_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)
- `updated_at` (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP) [1]

2.1.2 Relationships and ERD



The Entity-Relationship Diagram [2] illustrates the following relationships with cardinalities:

- DEPARTMENT 1:M DOCTOR: One department can have many doctors; a doctor belongs to at most one department (nullable FK allows unassigned doctors).
- PATIENT 1:M APPOINTMENT: One patient can have many appointments; each appointment belongs to exactly one patient.
- DOCTOR 1:M APPOINTMENT: One doctor can have many appointments; each appointment belongs to exactly one doctor.
- APPOINTMENT 1:1 TREATMENT: Each appointment has at most one treatment record; each treatment belongs to exactly one appointment.
- APPOINTMENT 1:1 FEEDBACK: Each appointment has at most one feedback record; each feedback belongs to exactly one appointment [3].

2.1.3 Normalization

The database design achieves Third Normal Form (3NF) through:

- Elimination of repeating groups: Each entity contains only atomic values (1NF satisfied).
- Elimination of partial dependencies: All non-key attributes depend on the whole primary key (2NF satisfied).
- Elimination of transitive dependencies: Non-key attributes depend only on the primary key, not on other non-key attributes.

Examples:

- The appointment table depends directly on its composite key (patient_id, doctor_id, appointment_date, appointment_time) for its attributes.
- Doctor details are stored in the doctor table rather than duplicated in appointment.
- Patient details are normalized to the patient table rather than repeated across appointments.
- Lookup data like department names is centralized to avoid redundancy.

This structure eliminates update, insertion, and deletion anomalies while ensuring data integrity [4] [5] [6].

2.2 Database Implementation

2.2.1 Table Creation and Constraints

Implementation begins with table creation using appropriate data types and constraints to enforce business rules.

Primary Keys: Auto-incrementing integers ensure unique record identification in all tables.

Foreign Key Constraints: Maintain referential integrity:

- `appointment.patient_id` → `patient.patient_id` (ON DELETE CASCADE)
- `appointment.doctor_id` → `doctor.doctor_id` (ON DELETE CASCADE)

- `treatment.appointment_id` → `appointment.appointment_id` (ON DELETE CASCADE)
- `feedback.appointment_id` → `appointment.appointment_id` (ON DELETE CASCADE)
- `doctor.department_id` → `department.department_id` (ON DELETE SET NULL)

Unique Constraints: Prevent duplicate entries:

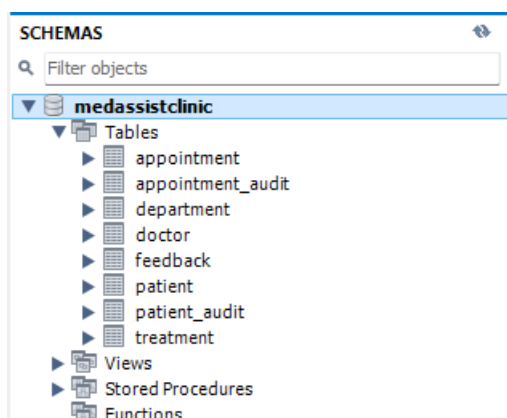
- `department.name` ensures department names are unique
- `doctor.email` ensures unique doctor contact information
- `patient.email` ensures unique patient contact information
- Composite key (`doctor_id`, `appointment_date`, `appointment_time`) in `appointment` prevents double booking
- `appointment_id` in `treatment` and `feedback` ensures one treatment/feedback per appointment

Check Constraints: Validate data domain:

- `gender` restricted to 'Male', 'Female', 'Other'
- `status` limited to valid appointment states
- `rating` in `feedback` constrained to values 1-5

Default Values: Timestamps automatically track record creation and modification:

- `created_at` set to `CURRENT_TIMESTAMP` on insert
- `updated_at` updates to `CURRENT_TIMESTAMP` on modification



Shows the eight tables created: six core tables (department, doctor, patient, appointment, treatment, feedback) plus two audit tables (patient_audit, appointment_audit) for tracking changes.

medassistclinic.department							
Info Columns Indexes Triggers Foreign keys Partitions Grants DDL							
Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra
department_id	int		NO			select,insert,update,references	auto_increment
name	varchar(100)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
description	text		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
created_at	timestamp	CURRENT_TIMESTA...	YES			select,insert,update,references	DEFAULT_GENE...
updated_at	timestamp	CURRENT_TIMESTA...	YES			select,insert,update,references	DEFAULT_GENE...

medassistclinic.department							
Info Columns Indexes Triggers Foreign keys Partitions Grants DDL							
Indexes in Table					Index Details		
Visible	Key	Type	Uni...	Columns	Key Name:	Packed:	Unique:
☒	PRIMARY	BTREE	YES	department_id	Index Type:		
☒	name	BTREE	YES	name	Allows NULL:		
					Cardinality:		
					Comment:		
					User Comment:		
Columns in table							
Column	Type	Nullable	Indexes				
department_id	int	NO	PRIMARY				
name	varchar(100)	NO	name				
description	text	YES					
created_at	timestamp	YES					
updated_at	timestamp	YES					

Department table showing primary key constraint on department_id, unique constraint on name, and appropriate data types for all attributes.

medassistclinic.appointment									
Info Columns Indexes Triggers Foreign keys Partitions Grants DDL									
Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra	Comments	
appointment_id	int		NO			select,insert,update,references	auto_increment		
patient_id	int		NO			select,insert,update,references			
doctor_id	int		NO			select,insert,update,references			
appointment_date	date		NO			select,insert,update,references			
appointment_time	time		NO			select,insert,update,references			
reason_for_visit	text		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references			
status	enum('scheduled','c...	scheduled	YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references			
created_at	timestamp	CURRENT_TIMESTA...	YES			select,insert,update,references	DEFAULT_GENE...		
updated_at	timestamp	CURRENT_TIMESTA...	YES			select,insert,update,references	DEFAULT_GENE...		

routine checkups, specialist consultations, condition-specific treatments and patient feedback mechanisms.

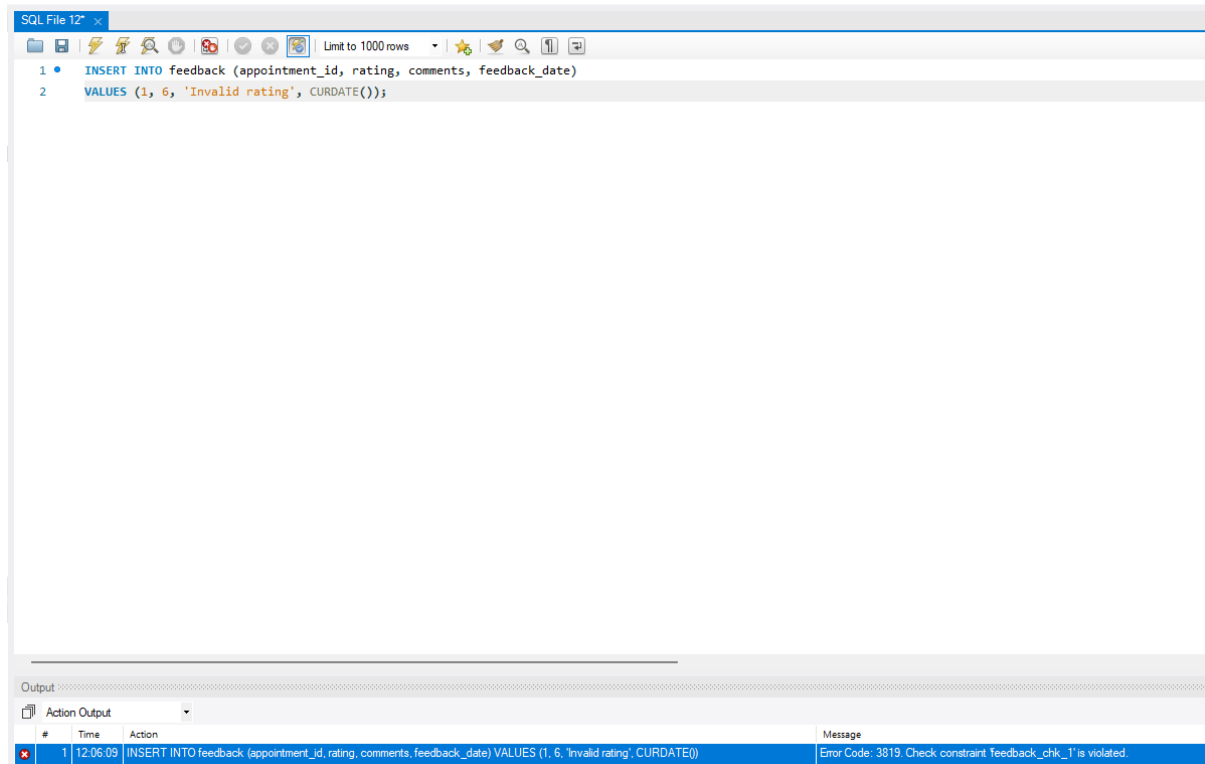
Result Grid						Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:
	department_id	name	description	created_at	updated_at				
▶	1	General Medicine	Primary care and general health consultations	2026-08-08 03:13:03	2026-08-08 03:13:03				
	2	Dermatology	Skin, hair, and nail conditions	2026-08-08 03:13:03	2026-08-08 03:13:03				
	3	Physiotherapy	Physical rehabilitation and movement therapy	2026-08-08 03:13:03	2026-08-08 03:13:03				
	4	Cardiology	Heart and cardiovascular system	2026-08-08 03:13:03	2026-08-08 03:13:03				
	5	Pediatrics	Medical care for infants, children, and adolescents	2026-08-08 03:13:03	2026-08-08 03:13:03				
★	NULL	NULL	NULL	NULL	NULL				

Result Grid										Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:
	appointment_id	patient_id	doctor_id	appointment_date	appointment_time	reason_for_visit	status	created_at	updated_at				
▶	1	1	1	2024-01-15	09:00:00	Annual checkup	completed	2026-08-08 03:13:03	2026-08-08 03:13:03				
	2	2	2	2024-01-16	10:30:00	Skin rash consultation	completed	2026-08-08 03:13:03	2026-08-08 03:13:03				
	3	3	3	2024-01-17	14:00:00	Back pain treatment	completed	2026-08-08 03:13:03	2026-08-08 03:13:03				
	4	4	4	2024-01-18	11:00:00	Chest pain evaluation	completed	2026-08-08 03:13:03	2026-08-08 03:13:03				
	5	5	5	2024-01-19	15:30:00	Child vaccination	completed	2026-08-08 03:13:03	2026-08-08 03:13:03				
	6	6	1	2024-01-20	09:30:00	Flu symptoms	scheduled	2026-08-08 03:13:03	2026-08-08 03:13:03				
	7	7	2	2024-01-21	13:00:00	Acne treatment	scheduled	2026-08-08 03:13:03	2026-08-08 03:13:03				
	8	1	6	2024-01-22	10:00:00	Follow-up consultation	scheduled	2026-08-08 03:13:03	2026-08-08 03:13:03				
★	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL				

Result Grid											Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:
	doctor_id	first_name	last_name	specialty	phone	email	department_id	hire_date	created_at	updated_at				
▶	1	John	Smith	General Practitioner	07911123456	john.smith@medassist.co.uk	1	2020-01-15	2026-08-08 03:13:03	2026-08-08 03:13:03				
	2	Emily	Jones	Dermatologist	07911123457	emily.jones@medassist.co.uk	2	2019-03-22	2026-08-08 03:13:03	2026-08-08 03:13:03				
	3	Michael	Brown	Physiotherapist	07911123458	michael.brown@medassist.co.uk	3	2021-07-10	2026-08-08 03:13:03	2026-08-08 03:13:03				
	4	Sarah	Davis	Cardiologist	07911123459	sarah.davis@medassist.co.uk	4	2018-11-05	2026-08-08 03:13:03	2026-08-08 03:13:03				
	5	David	Wilson	Pediatrician	07911123460	david.wilson@medassist.co.uk	5	2020-09-18	2026-08-08 03:13:03	2026-08-08 03:13:03				
	6	Lisa	Anderson	General Practitioner	07911123461	lisa.anderson@medassist.co.uk	1	2021-02-28	2026-08-08 03:13:03	2026-08-08 03:13:03				
	7	Robert	Taylor	Dermatologist	07911123462	robert.taylor@medassist.co.uk	2	2022-01-10	2026-08-08 03:13:03	2026-08-08 03:13:03				
★	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL				

Result Grid								Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:
	feedback_id	appointment_id	rating	comments	feedback_date	created_at	updated_at				
▶	1	1	5	Excellent service, doctor was very thorough an...	2024-01-15	2026-08-08 03:13:03	2026-08-08 03:13:03				
	2	2	4	Good treatment, slight wait time but worth it	2024-01-16	2026-08-08 03:13:03	2026-08-08 03:13:03				
	3	3	5	Physiotherapist was very helpful, pain significan...	2024-01-17	2026-08-08 03:13:03	2026-08-08 03:13:03				
	4	4	4	Cardiologist was knowledgeable, explained ever...	2024-01-18	2026-08-08 03:13:03	2026-08-08 03:13:03				
	5	5	5	Great with children, made my son feel comforta...	2024-01-19	2026-08-08 03:13:03	2026-08-08 03:13:03				
★	NULL	NULL	NULL	NULL	NULL	NULL	NULL				

Result Grid												Filter Rows:	Edit:	Export/Import:	Wrap Cell Content:
	patient_id	first_name	last_name	date_of_birth	gender	phone	email	address	registration_date	emergency_contact_name	emergency_contact_phone				
▶	1	Alice	Johnson	1985-04-12	Female	07700123456	alice.johnson@email.com	123 Oak Street, London	2023-01-15	Mark Johnson	07700123457				
	2	Bob	Smith	1972-08-25	Male	07700123458	bob.smith@email.com	45 Elm Avenue, Manchester	2023-02-20	Sarah Smith	07700123459				
	3	Carol	Davis	1990-11-03	Female	07700123460	carol.davis@email.com	78 Pine Road, Birmingham	2023-03-10	Tom Davis	07700123461				
	4	David	Clark	1965-02-18	Male	07700123462	david.clark@email.com	34 Cedar Lane, Leeds	2023-01-30	Jane Clark	07700123463				
	5	Emma	Taylor	2010-06-30	Female	07700123464	emma.taylor@email.com	56 Birch Drive, Liverpool	2023-04-05	David Taylor	07700123465				
	6	Frank	Miller	1988-09-14	Male	07700123466	frank.miller@email.com	89 Walnut Street, Sheffield	2023-03-22	Laura Miller	07700123467				
	7	Grace	Lee	1995-12-07	Female	07700123468	grace.lee@email.com	12 Poplar Court, Bristol	2023-05-12	James Lee	07700123469				
★	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL				



Testing constraint validation - the CHECK constraint on feedback.rating ensures only valid ratings (1-5) are accepted.

2.2.3 Stored Procedures, Triggers, and Functions

The implementation includes stored procedures for CRUD operations with transaction handling and triggers for audit trails.

Stored Procedures:

- `sp_insert_patient`, `sp_update_patient`, `sp_delete_patient`: Handle patient records with transaction safety
- `sp_insert_appointment`, `sp_update_appointment_status`: Manage appointment lifecycle
- `sp_add_treatment`, `sp_add_feedback`: Record clinical outcomes and patient feedback
- `sp_book_appointment_with_treatment`: Demonstrates multi-step transactional scenario
- `sp_book_appointment_check_patient`: Demonstrates conditional rollback based on preconditions

Audit Triggers: Automatically track changes to critical tables:

- `trg_patient_after_insert/update/delete`: Records patient modifications in `patient_audit`

- `trg_appointment_after_insert/update/delete`: Records appointment changes in `appointment_audit`

Triggers capture: patient/appointment ID, operation type

(INSERT/UPDATE/DELETE), timestamp, user who made the change (via

`CURRENT_USER()`), and before/after data states in TEXT format.

Procedure	sql_mode	Create Procedure	character_set_client	collation_connection	Database Collation
sp_insert_patient	ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_Z	<pre> CREATE DEFINER='root'@'localhost' PROCEDURE `sp_insert_patient` (IN p_first_name VARCHAR(50), IN p_last_name VARCHAR(50), IN p_date_of_birth DATE, IN p_gender ENUM('Male','Female','Other'), IN p_phone VARCHAR(20), IN p_email VARCHAR(100), IN p_address TEXT, IN p_emergency_contact_name VARCHAR(100), IN p_emergency_contact_phone VARCHAR(20)) BEGIN DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN ROLLBACK; SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error inserting patient. Transaction rolled back.'; END; START TRANSACTION; INSERT INTO patient (first_name, last_name, date_of_birth, gender, phone, email, address, emergency_contact_name, emergency_contact_phone, registration_date) VALUES (p_first_name, p_last_name, p_date_of_birth, p_gender, p_phone, p_email, p_address, p_emergency_contact_name, p_emergency_contact_phone, CURDATE()); COMMIT; END </pre>	utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_ci

Stored procedure for inserting new patients demonstrating transaction handling with `BEGIN`, `COMMIT`, `ROLLBACK` and exception handling.

Trigger	Event	Table	Statement	Timing
trg_appointment_after_insert	INSERT	appointment	<pre> BEGIN INSERT INTO appointment_audit (appointment_id, operation, new_data, changed_by) VALUES (NEW.appointment_id, 'INSERT', CONCAT('{appointment_id:', NEW.appointment_id, ',patient_id:', NEW.patient_id, ',doctor_id:', NEW.doctor_id, ',appointment_date:', NEW.appointment_date, ',', ',appointment_time:', NEW.appointment_time, ',', ',reason_for_visit:', NEW.reason_for_visit, ',', ',status:', NEW.status, '}'), CURRENT_USER()); END </pre>	AFTER

Audit trigger that automatically records changes to patient data, including who made the change (`CURRENT_USER()`) and the data before/after modification.

Result Grid Filter Rows: Export: Wrap Cell Content:				
View	Create View	character_set_client	collation_connection	
vw_doctor_appointments	<pre>CREATE ALGORITHM=UNDEFINED DEFINER='root'@'localhost' SQL SECURITY DEFINER VIEW `vw_doctor_appointments` AS select `a`.`appointment_id` AS `appointment_id`,concat(`d`.`first_name`,` `,`d`.`last_name`) AS `doctor_name`,concat(`p`.`first_name`,` `,`p`.`last_name`) AS `patient_name`,`a`.`appointment_date` AS `appointment_date`,`a`.`appointment_time` AS `appointment_time`,`a`.`reason_for_visit` AS `reason_for_visit`,`a`.`status` AS `status` from ((`appointment` `a` join `doctor` `d` on((`a`.`doctor_id` = `d`.`doctor_id`))) join `patient` `p` on((`a`.`patient_id` = `p`.`patient_id`))) order by `a`.`appointment_date`,`a`.`appointment_time`</pre>	utf8mb4	utf8mb4_0900_ai_ci	

View displaying scheduled appointments with joined doctor and patient names for easy staff reference, ordered by appointment time.

2.2.4 Transaction Handling (BEGIN, COMMIT, ROLLBACK)

Transaction handling ensures data integrity during multi-step operations by treating them as atomic units that either fully complete or fully roll back.

Successful Transaction Scenario:

Calling `sp_book_appointment_with_treatment` with valid parameters executes as a single unit:

1. Inserts appointment record.
2. Adds treatment record.
3. Adds feedback record.
4. Updates appointment status to 'completed'.

All steps commit together, or if any step fails, the entire transaction rolls back leaving the database unchanged.

Conditional Rollback Scenario:

The procedure `sp_book_appointment_check_patient` first verifies patient existence before booking:

- If `patient_id` doesn't exist, it raises an error and rolls back (no appointment created).
- If patient exists, it proceeds with booking and commits the transaction.

This prevents orphaned appointments and maintains referential integrity.

SQL File 12"

1CALL sp_book_appointment_with_treatment(
2 1, -- p_patient_id: Alice Johnson
3 1, -- p_doctor_id: John Smith
4 '2024-02-10', -- p_appointment_date
5 '14:00:00', -- p_appointment_time
6 'Follow-up consultation', -- p_reason_for_visit
7 'hypertension stage 2', -- p_diagnosis
8 'Amlodipine 5mg daily', -- p_prescription
9 'Patient reports improved BP, advise low-salt diet', -- p_notes
10 5, -- p_rating
11 'Very satisfied with the follow-up; doctor was thorough.' -- p_feedback_comments
12);

Output

Action Output

#	Time	Action	Message
1	12:59:05	CALL sp_book_appointment_with_treatment(1, -- p_patient_id: Alice Johnson 1, -- p_doctor_id: John Smith '2024-02-10', -- p_appointment_date ...	0 row(s) affected

SQL File 12"

1SELECT * FROM feedback;

Result Grid

feedback_id	appointment_id	rating	comments	feedback_date	created_at	updated_at
1	1	5	Excellent service, doctor was very thorough and attentive	2024-01-15	2026-08-08 03:13:03	2026-08-08 03:13:03
2	2	4	Good treatment, slight wait time but worth it	2024-01-16	2026-08-08 03:13:03	2026-08-08 03:13:03
3	3	5	Physiotherapist was very helpful, pain significantly reduced	2024-01-17	2026-08-08 03:13:03	2026-08-08 03:13:03
4	4	4	Cardiologist was knowledgeable, explained everything clearly	2024-01-18	2026-08-08 03:13:03	2026-08-08 03:13:03
5	5	5	Great with children, made my son feel comfortable during vaccination	2024-01-19	2026-08-08 03:13:03	2026-08-08 03:13:03
6	9	5	Very satisfied with the follow-up; doctor was thorough.	2026-08-08	2026-08-08 12:59:05	2026-08-08 12:59:05
NULL	NULL	NULL	NULL	NULL	NULL	NULL

feedback 24 x

Output

Action Output

#	Time	Action	Message
1	12:59:05	CALL sp_book_a	CALL sp_book_appointment_with_treatment('2024-02-10', -- p_appointment_date ... 0 row(s) affected
2	13:03:54	SELECT * FROM	1, -- p_patient_id: Alice Johnson 1, -- p_doctor_id: John Smith 9 row(s) returned
3	13:05:09	SELECT * FROM	'2024-02-10', -- p_appointment_date '14:00:00', -- p_appointment_time 'Follow-up consultation', -- p_reason_for_visit 'Hypertension stage 2', -- p_diagnosis 'Amlodipine 5mg daily', -- p_prescription 'Patient reports improved BP, advise low-salt diet', -- p_notes 5, -- p_rating 'Very satisfied with the follow-up; doctor was thorough.' -- p_feedback_comments 6 row(s) returned

Successful transaction where all operations (appointment booking, treatment recording, feedback submission) completed and were committed as a single unit.

SQL File 12*

```
1 USE medassistclinic;
2
3 CALL sp_book_appointment_check_patient(
4     999, -- p_patient_id: intentionally non-existent
5     1, -- p_doctor_id
6     '2024-02-11', -- p_appointment_date
7     '09:00:00', -- p_appointment_time
8     'New patient consultation' -- p_reason_for_visit
9 );
```

Output

#	Time	Action	Message
1	13:08:20	USE medassistclinic	0 row(s) affected
2	13:08:20	CALL sp_book_appointment_check_patient(999, --p_patient_id: intentionally non-existent 1, --p_doctor_id '2024-02-11', --p_appointment_date ...	Error Code: 1644. Error in appointment booking. Transaction rolled back.

SQL File 12*

```
1 USE medassistclinic;
2
3 SELECT COUNT(*) AS appt_cnt
4 FROM appointment
5 WHERE patient_id = 999;
```

Result Grid

appt_cnt
0

Result 25 x

Output

#	Time	Action	Message
1	13:08:20	USE medassistclinic	0 row(s) affected
2	13:08:20	CALL sp_book_appointment_check_patient(999, --p_patient_id: intentionally non-existent 1, --p_doctor_id '2024-02-11', --p_appointment_date...	Error Code: 1644. Error in appointment booking. Transaction rolled back.
3	13:09:22	USE medassistclinic	0 row(s) affected
4	13:09:22	SELECT COUNT(*) AS appt_cnt FROM appointment WHERE patient_id = 999 LIMIT 0, 1000	1 row(s) returned

"Failed transaction - the procedure detects the patient doesn't exist, raises an error, and rolls back all changes. Verification shows no appointment was created, maintaining data integrity.

2.2.5 Performance Tuning & Indexing

Strategic indexing improves query performance for common clinic operations while balancing write performance overhead.

Index Choices and Justification:

- `idx_appointment_patient` (`patient_id`): Speeds up patient history lookup (frequent for treatment reviews).
- `idx_appointment_doctor` (`doctor_id`): Optimizes doctor schedule viewing (high-use operational query).
- `idx_appointment_date` (`appointment_date`): Accelerates daily schedule checks and date-range reporting.
- `idx_appointment_status` (`status`): Improves filtering for appointment status dashboards.
- `idx_treatment_appointment` (`appointment_id`): Efficient treatment retrieval for given appointments.
- `idx_feedback_appointment` (`appointment_id`): Quick feedback lookup for appointments.
- `idx_patient_last_name` (`last_name`): Speeds up patient searches by name (common front-desk operation).
- `idx_doctor_last_name` (`last_name`): Optimizes doctor lookup by name.
- `idx_doctor_department` (`department_id`): Accelerates department-based doctor assignments and reporting.

Expected Improvements:

- Point lookups by indexed columns change from $O(n)$ full table scans to $O(\log n)$ index searches.
- Date-range queries (e.g., "today's appointments") benefit significantly from `idx_appointment_date`.
- Join operations between appointment and patient/doctor tables execute faster using respective foreign key indexes.
- Reporting queries grouping by status or department see performance gains from relevant indexes.

SQL File 12" x

Limit to 1000 rows

1 • EXPLAIN SELECT * FROM appointment WHERE appointment_date = '2024-01-15';

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [IA](#)

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	appointment	NULL	ref	idx_appointment_date	idx_appointment_date	3	const	1	100.00	NULL

Query execution plan showing how the `idx_appointment_date` index is used (type: `ref`, possible_keys and key showing the index) making date-based searches efficient.

2.3 Justification and Literature Support

The design choices are grounded in established database principles:

- For normalization theory and rules to eliminate redundancy: *M. Fotache, M.-I. Cluci, T. Taipalus, and G. Talaba, "The effects of database normalization on decision support system performance," Information Systems, vol. 136, p. 102636, Feb. 2026, Section 1-2-3-4-5. doi: 10.1016/j.is.2025.102636. Available:*

<https://www.sciencedirect.com/science/article/pii/S030643792500122X>

- Auditing triggers. Follows best practices for immutable audit trails where triggers capture changes at the database level, ensuring all modifications are tracked regardless of application path: *R. Elmasri and S. Navathe, Fundamentals of database systems, 7th ed. Pearson, 2017, ch. 7-26-30.*

- Transaction processing. Implements ACID properties (Atomicity, Consistency, Isolation, Durability) using MySQL's transaction support with proper COMMIT/ROLLBACK handling. Stored procedures encapsulate business logic with exception handling to maintain consistency: *A. Silberschatz, H. F. Korth, and S. Sudarshan, Database System Concepts, 7th ed. McGraw-Hill, 2019, ch. 4-5-14-15-17. Available:* <https://www.mpgcamb.com/wp-content/uploads/2024/12/Abraham-Silberschatz-Henry-F.-Korth-S.-Sudarshan-Database-System-Concepts-McGraw-Hill-Education-2019.pdf>

- Constraint utilization. Employs database-level constraints (NULL, UNIQUE, CHECK, FOREIGN KEY) as the first line of defense for data integrity, complementing application-level validation: *C. J. Date, SQL and Relational Theory: How to Write Accurate SQL Code, 2nd ed. O'Reilly Media, 2011, ch. 1-4-8.*

3. Part B – PHP Website Development

3.1 Overview of Website Topic

The topic chosen for Part B of this assessment is related to videogames and is appropriately named The Video Game Catalogue. It is an information-based web application which enables users to browse, search and view details of video games organized by genre and platform. The main purpose is to demonstrate proficiency in PHP-MySQL integration, including dynamic content retrieval, full CRUD (Create, Read, Update, Delete) functionality, user-friendly search/filter capabilities and essential security controls. The reason for the choice of subject was because it provides a clear domain for relational data (games, genres, platforms) while remaining non-controversial in nature and educationally appropriate as per the instructions of this part of the assessment.

3.2 Database Design for Website

3.2.1 Tables and Relationships

The database follows a normalized relational schema consisting of three tables:

1. **genres** (id PK, name) – lookup table for video game genres.
2. **platforms** (id PK, name) – lookup table for gaming platforms.
3. **games** (id PK, title, genre_id FK, platform_id FK, release_year, description) – main entity table.

Relationships are established via foreign keys: games.genre_id references genres.id and games.platform_id references platforms.id. This design satisfies First Normal Form (atomic columns), Second Normal Form (full functional dependence on primary keys) and Third Normal Form (no transitive dependencies). By separating descriptive

attributes into lookup tables, redundancy is eliminated and referential integrity is enforced [7].

3.2.2 Sample Data

Sample data includes five genres (Action, RPG, Adventure, Sports, Strategy), five platforms (PlayStation 5, Xbox Series X|S, Nintendo Switch, PC, Mobile), and twenty games.

Game Title	Genre	Platform	Release Year	Description
The Legend of Zelda: Tears of the Kingdom	Adventure	Nintendo Switch	2023	The latest entry in the legendary Zelda series...
Among us	Strategy	Mobile	2018	A multiplayer game of teamwork and betrayal...
Animal Crossing: New Horizons	Adventure	Nintendo Switch	2020	Social simulation game where players create...
Call of Duty: Modern Warfare II	Action	Xbox Series X S	2022	The latest installment in the blockbuster military...
Civilization VI	Strategy	PC	2016	Turn-based strategy game where players build an empire...
Cyberpunk 2077	RPG	PC	2020	An open-world, action-adventure story...
Disco Elysium	RPG	PC	2019	A groundbreaking role-playing game about a detective...
Elden Ring	RPG	PC	2022	FromSoftware's acclaimed open-world dark fantasy RPG...
FIFA 22	Sports	PlayStation 5	2021	Football simulation featuring HyperMotion technology...
FIFA 23	Sports	PC	2022	The latest installment in EA's football simulation series...
God of War Ragnarök	Action	PlayStation 5	2022	Kratos and Atreus embark on a perilous journey...
Hogwarts Legacy	RPG	PlayStation 5	2023	An immersive, open-world action RPG...
Hollow Knight	Adventure	PC	2017	A classic 2D action-adventure game...

Mario Kart 8 Deluxe	Sports	Nintendo Switch	2017	The ultimate definitive version of Mario Kart 8...
Pokémon Legends: Arceus	RPG	Nintendo Switch	2018	A new take on the Pokémon franchise...
Red Dead Redemption 2	RPG	PlayStation 5	2018	An epic tale of life in America's unforgiving heartland...
Rocket League	Sports	PC	2015	High-powered hybrid of arcade-style soccer...
Stardew Valley	RPG	PC	2016	Open-ended country-life RPG with farming...
Super Mario Odyssey	Adventure	Nintendo Switch	2017	Mario's globe-trotting 3D adventure featuring...

3.3 PHP Implementation

3.3.1 Dynamic Content Display

Dynamic content is generated using PHP Data Objects (PDO) with prepared statements. A single connection is established in config.php and reused across pages. On index.php, a base query joins games with genres and platforms to obtain readable names. Conditional clauses are appended based on user-supplied search (`$_GET['search']`) and filter (`$_GET['genre']`, `$_GET['platform']`) parameters, then executed via `$pdo->prepare()` and `$execute()`. The result set is iterated to output HTML cards showing title, genre, platform, year, and a truncated description. Individual game details are shown on game_detail.php using a parameterized query where the game ID is bound as an integer, preventing type-juggling attacks [8].

3.3.2 CRUD Operations

CRUD functionality resides in the /admin/ directory:

- Create (add_game.php): Inserts a new record after validating required fields (title, genre_id, platform_id). Uses `$pdo->prepare("INSERT INTO games (title, genre_id, platform_id, release_year, description) VALUES (?, ?, ?, ?, ?)")`.
- Read: Covered in previous section 3.3.1; also includes list_games.php which displays all games in a table for administrative overview.

- Update (edit_game.php): Retrieves existing data via SELECT, then updates with `$pdo->prepare("UPDATE games SET title=?, genre_id=?, platform_id=?, release_year=?, description=? WHERE id=?")`.
- Delete (list_games.php and delete_game.php): Executes `$pdo->prepare("DELETE FROM games WHERE id=?")` after a JavaScript confirmation prompt.

All statements employ bound parameters, ensuring data-code separation [8] [9].

3.3.3 Search/Filter Functionality

Users may search by game title (partial match using LIKE '%?%') and/or filter by exact genre or platform ID. The search term is sanitized with trim() before binding; numeric filters are cast with intval() to guarantee integer values. Query construction appends clauses only when corresponding inputs are present, optimizing performance via indexed columns (genre_id, platform_id). Results are ordered alphabetically by title for consistent presentation.

3.3.4 Security Measures (Login, Input Validation)

Security is implemented in layers:

- Authentication: Admin access requires session-based login (admin/login.php) with hard-coded credentials (admin/password123) for this educational demo. On successful validation, `$_SESSION['admin_logged_in']` is set; admin scripts redirect to login.php if the session is absent or false.
- Input Validation: All incoming data undergo type-specific validation: strings are trimmed, numeric IDs are cast to int, and empty critical fields trigger user-friendly error messages.
- Output Encoding: Any user-derived data echoed in HTML contexts passes through `htmlspecialchars()` to mitigate Cross-Site Scripting (XSS).
- Prepared Statements: As detailed below, PDO prepared statements defend against SQL Injection [10].

3.4 SQL Injection Overview

3.4.1 Explanation and Examples

SQL Injection (SQLi) is a security vulnerability in database-driven web applications which occur when untrusted user input is directly concatenated or interpolated into an SQL query string without proper validation, escaping or parameterization.

When a system fails to maintain data-code separation, the relational database management system (RDBMS) cannot distinguish between the application developer's intended SQL commands (code) and dynamic user inputs (data). An attacker can exploit this structural flaw by supplying crafted input containing SQL meta-characters (such as single quotes, comments or logical operators OR 1 = 1), forcing the database engine to parse and execute commands as part of the SQL query grammar.

SQL injection remains one of the most severe and pervasive threats to enterprise applications due to the depth of control an attacker can achieve. Key risks include:

- **Loss of Confidentiality (Data Exfiltration):** Attackers can bypass access controls to read restricted database tables, extracting sensitive information such as personal identifiable information (PII), payment credentials, intellectual property, or password hashes.
- **Loss of Integrity (Data Tampering):** Attackers can execute UPDATE, INSERT, or DELETE statements, enabling them to alter account balances, modify administrative permissions, or destroy database records.
- **Authentication & Authorization Bypasses:** SQLi vulnerabilities frequently allow attackers to impersonate arbitrary users, including system administrators, granting them elevated privileges within the application context.
- **Denial of Service (DoS):** By injecting resource-intensive queries or database-specific drop commands (e.g., DROP TABLE), attackers can disrupt database availability and application functionality.
- **Remote Code Execution & Host Compromise:** In environments where the database engine runs with elevated system privileges or features extended stored procedures enabled (e.g., xp_cmdshell in MS SQL Server), attackers can pivot from

the database layer to execute shell commands on the underlying server operating system.

3.4.2 Vulnerabilities in Website Code

Upon a review of the developed codebase, no inherent SQL injection vulnerabilities were identified because all database interactions utilize PDO prepared statements with bound parameters. However, for illustrative purposes, a hypothetical vulnerable version of index.php's search block would be:

```
// VULNERABLE (not used)
```

```
$sql = "SELECT * FROM games WHERE title LIKE '%$search%'";
```

Here, a malicious search term such as ' UNION SELECT id, username, password FROM users -- could extract credentials from a user's table if one existed. Such patterns are absent from the final implementation.

3.4.3 Anti-SQL Injection Measures Implemented

The primary defense is PDO Prepared Statements with Parameter Binding. This technique separates SQL syntax from data: the query structure is sent to the MySQL server first, placeholders (?) are recognized, and user supplies are transmitted separately as parameters, never interpreted as SQL. Examples from the codebase:

- Search in index.php:

```
$stmt = $pdo->prepare("SELECT g.*, gen.name AS genre_name, plat.name AS  
platform_name
```

```
FROM games g
```

```
JOIN genres gen ON g.genre_id = gen.id
```

```
JOIN platforms plat ON g.platform_id = plat.id
```

```
WHERE g.title LIKE ?");
```

```
$stmt->execute(["%{$search}%"]);
```

- Genre filter:

```
$stmt->execute([$genre_filter]); // $genre_filter is intval()-cast
```

- Admin insert (add_game.php):

```
$stmt = $pdo->prepare("INSERT INTO games (title, genre_id, platform_id,
release_year, description)

VALUES (?, ?, ?, ?, ?)");

$stmt->execute([$title, $genre_id, $platform_id, $release_year, $description]);
```

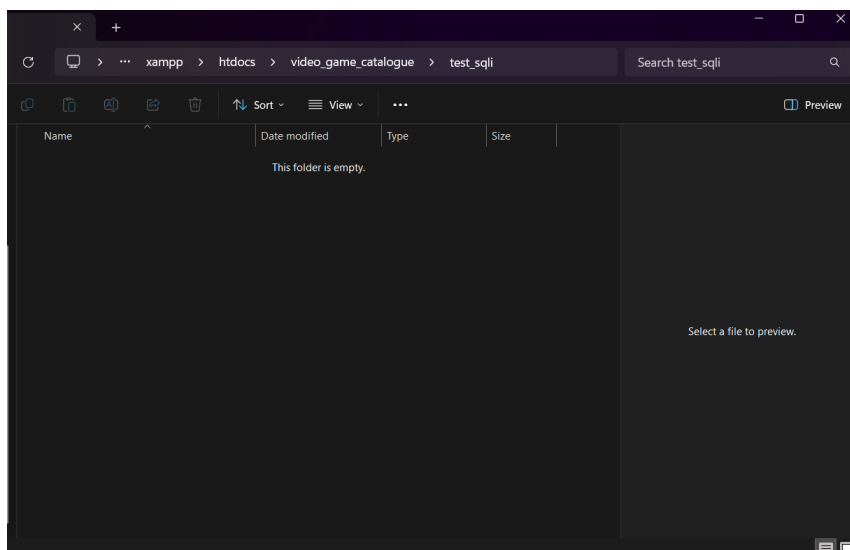
Additional safeguards include:

- Input Typing: Numeric IDs are forced to integers via intval(), eliminating string-based injection vectors.
- Least Privilege: While the demo uses the root account for simplicity, a production system would employ a dedicated MySQL user with only SELECT, INSERT, UPDATE, DELETE privileges on the required tables.
- Error Handling: PDO exceptions are caught and generic messages displayed, preventing leakage of query details that could aid attackers [12].

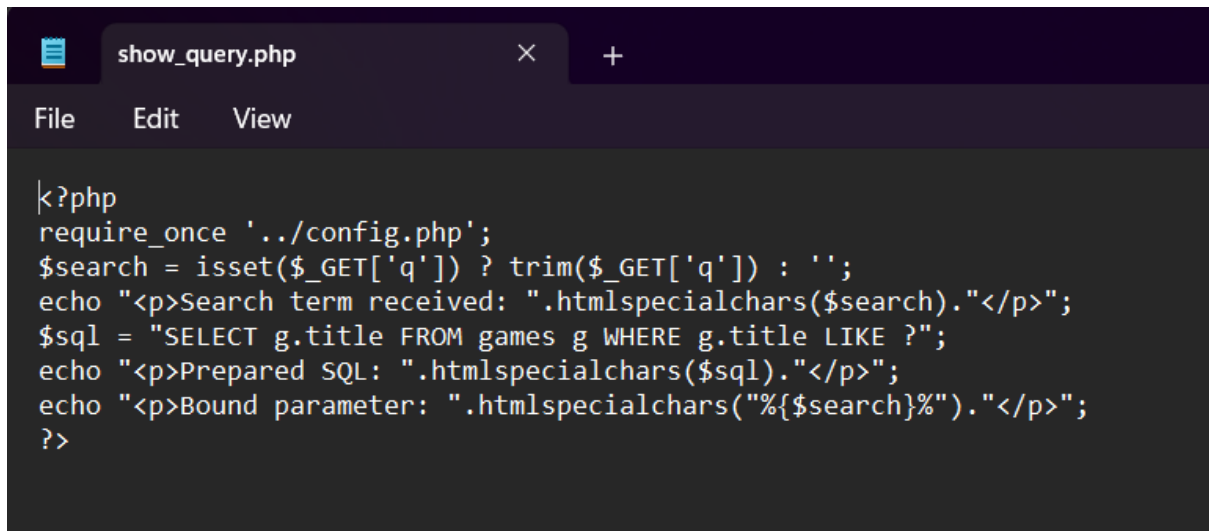
3.4.4 Screenshots of Testing and Results

3.4.4.1 Test 1 - Normal Search (Baseline)

Firstly, before carrying out any tests, it is important to keep test files isolated from the live site.



Test_sql folder is created with that idea in mind.

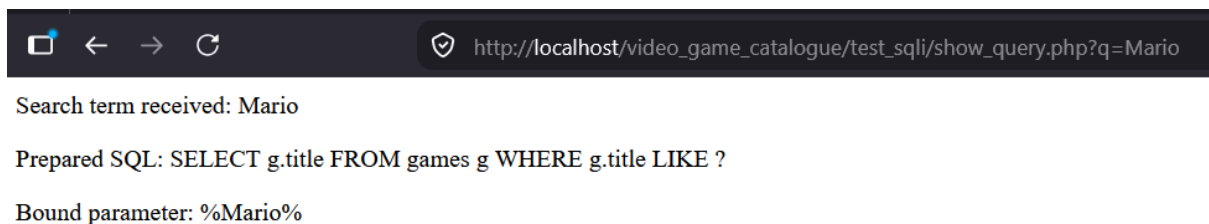
A screenshot of a code editor window titled 'show_query.php'. The editor has a menu bar with 'File', 'Edit', and 'View'. The code is written in PHP and displays search results for a game title. It uses htmlspecialchars to escape the search term and the SQL query.

```
<?php
require_once '../config.php';
$search = isset($_GET['q']) ? trim($_GET['q']) : '';
echo "<p>Search term received: ".htmlspecialchars($search)."</p>";
$sql = "SELECT g.title FROM games g WHERE g.title LIKE ?";
echo "<p>Prepared SQL: ".htmlspecialchars($sql)."</p>";
echo "<p>Bound parameter: ".htmlspecialchars("%{$search}%")."</p>";
?>
```

Created a simple echo page which shows the raw SQL that the site builds, only for testing purposes.

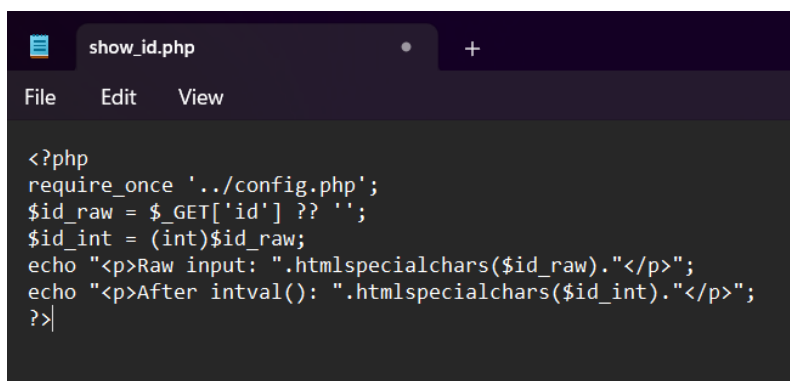
Next step is to request

http://localhost/video_game_catalogue/test_sql/show_query.php?q=Mario in a browser.



The query never concatenates the user input; it stays as a placeholder (?). This is the core of the defense.

3.4.4.2 Test 2 – Injection via Numeric ID (Edit / Delete)

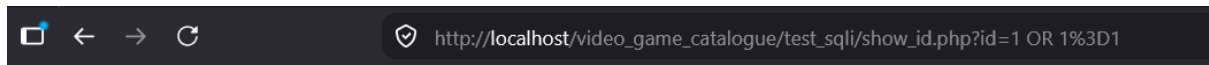
A screenshot of a code editor window titled 'show_id.php'. The editor has a menu bar with 'File', 'Edit', and 'View'. The code is written in PHP and displays the raw input and the value after intval() for a numeric ID.

```
<?php
require_once '../config.php';
$id_raw = $_GET['id'] ?? '';
$id_int = (int)$id_raw;
echo "<p>Raw input: ".htmlspecialchars($id_raw)."</p>";
echo "<p>After intval(): ".htmlspecialchars($id_int)."</p>";
?>
```

Creation of a test script which shows the cast value.

Next step is to request

http://localhost/video_game_catalogue/test_sql/show_id.php?id=1%20OR%201%3D1



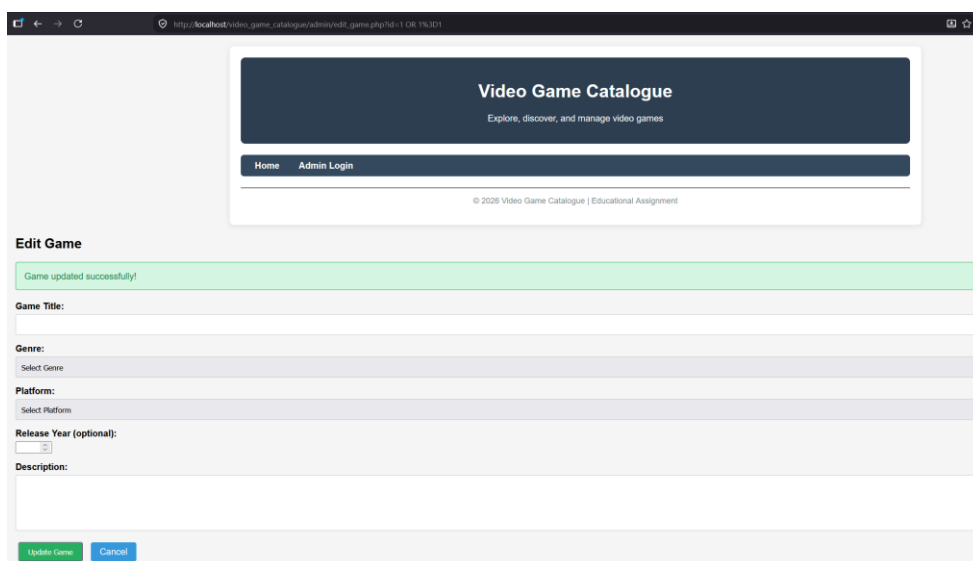
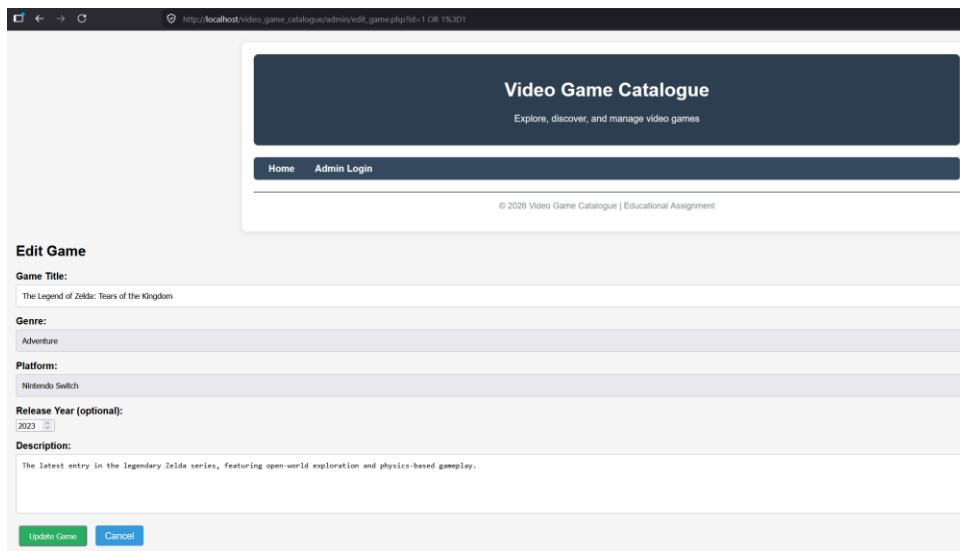
Raw input: 1 OR 1=1

After intval(): 1

The non-numeric part is discarded, leaving only the leading integer.

Now, an attempt to edit a game:

http://localhost/video_game_catalogue/admin/edit_game.php?id=1%20OR%201%3D1



Video Game Catalogue

Explore, discover, and manage video games

Home Admin Login

© 2026 Video Game Catalogue | Educational Assignment

Edit Game

Game Title:
The Legend of Zelda: Tears of the Kingdom

Genre:
Adventure

Platform:
Nintendo Switch

Release Year (optional):
2023

Description:
The latest entry in the legendary Zelda series, featuring open-world exploration and physics-based

Update Game Cancel

Numeric-ID injection – intval() reduces to safe integer.

The editing attempt was not successful [8] [10] [12].

4. Conclusion and Reflection

Completing this assignment has been a challenging but highly valuable learning experience. Developing a MySQL database and PHP-based web application required me to apply technical knowledge across several different areas, including database design, SQL scripting, PHP programming, database connectivity, form handling, validation, and debugging. Although I encountered numerous obstacles throughout the development process, successfully completing the assignment has given me a much stronger understanding of both database-driven applications and the development process as a whole.

One of the biggest challenges I faced was working with SQL and understanding how to write effective and correctly structured SQL scripts. At the beginning of the assignment, I found it difficult to understand how different SQL statements worked together and how to structure queries appropriately for the requirements of the application. I often had to research different approaches, study examples, identify errors in my code and test solutions repeatedly before I could achieve the desired results. This process was sometimes frustrating, particularly when a small error prevented an entire section of the application from working. However, the process of

identifying and correcting these problems gradually improved my understanding of SQL and gave me greater confidence in working with databases.

The PHP development aspect was equally challenging. I had difficulty understanding how to structure the code properly and how the different components of the PHP application should interact with the MySQL database. Since I was unable to attend classes and therefore did not have the same baseline knowledge or continuous guidance that would normally be available, I had to rely heavily on independent research and experimentation. I spent considerable time studying documentation, tutorials, examples, and alternative solutions before understanding how to implement particular features. Debugging became an important part of my learning process, as each error provided an opportunity to understand what I had done incorrectly and how the underlying code worked.

Despite these difficulties, one of the most significant outcomes of the assignment was the amount I learned through practical problem-solving. Rather than simply following instructions or reproducing code, I had to investigate problems independently and develop my ability to analyze why something was not working. This helped me become more persistent and resourceful when facing unfamiliar technical problems. I also learned that programming rarely works perfectly on the first attempt and that debugging, testing, and refinement are essential parts of the development process.

Overall, I am pleased with the outcome of the assignment and with the progress I made throughout the development process. What initially seemed overwhelming became increasingly manageable as my knowledge and confidence developed. The experience has also highlighted areas where I need to continue developing, particularly in writing efficient SQL queries, structuring PHP applications, and following best practices. Most importantly, this assignment has provided me with a stronger foundation on which I can build my future programming and database development skills.

5. References

- [1] A. Watt, "Chapter 8 The Entity Relationship Data Model," BCcampus, Oct. 2016. [Online]. Available: <https://opentextbc.ca/dbdesign01/chapter/chapter-8-entity-relationship-model/>
- [2] <https://app.diagrams.net/>
- [3] I. Belcic and C. Stryker, "What is an Entity Relationship Diagram? | IBM," 2024. [Online]. Available: <https://www.ibm.com/think/topics/entity-relationship-diagram>
- [4] Y. Bai, distributor IEEE Xplore, and publisher Wiley, *SQL server database programming with visual basic.net : concepts, designs and implementations*, First edition. Hoboken, New Jersey: John Wiley & Sons, Inc., 2020. doi: 10.1002/9781119608493.
- [5] A. Gomstyn and A. Jonker, "Database normalization," June 2025. Available: <https://www.ibm.com/think/topics/database-normalization>
- [6] GeeksforGeeks, "Normal Forms in DBMS," July 2015. Available: <https://www.geeksforgeeks.org/dbms/normal-forms-in-dbms/>
- [7] C. J. Date, *Database design and relational theory : normal forms and all that jazz*. Berkeley, CA: Apress, 2019. Available: <https://content.e-bookshelf.de/media/reading/L-13198798-9beae0c728.pdf>
- [8] L. E. Ullman, *PHP and MySQL for dynamic web sites*. San Francisco: Peachpit Press, 2018.
- [9] R. Nixon, *LEARNING PHP, MYSQL & JAVASCRIPT : a step-by-step guide to creating dynamic websites*. S.L.: O'reilly Media, 2021.
- [10] OWASP, "SQL Injection Prevention · OWASP Cheat Sheet Series," Owasp, 2025. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- [11] W. G. J. Halfond and A. Orso, "Combining static analysis and runtime monitoring to counter SQL-injection attacks," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, p. 1, July 2005, doi: 10.1145/1082983.1083250.
- [12] PHP, "PHP: PDO - Manual," 2019. Available: <https://www.php.net/manual/en/book.pdo.php>

6. Supporting Material

- 1 - SQL practice https://www.sqlzoo.net/wiki/SQL_Tutorial
- 2 - SQL theory and practice <https://www.w3schools.com/MYSQL/>

- 3 - UWTSD Database & Application Development course material
- 4 - PHP Full Course <https://www.youtube.com/watch?v=zZ6vybT1HQs>
- 5 - General debugging <https://stackoverflow.com/questions>