Team member:

Kyle Jia, jj3708

Junxiao Wang, jw8193

Nikhil Diddee, nd2564

Milestone 3: Database Setup and SQL

1. Database Server

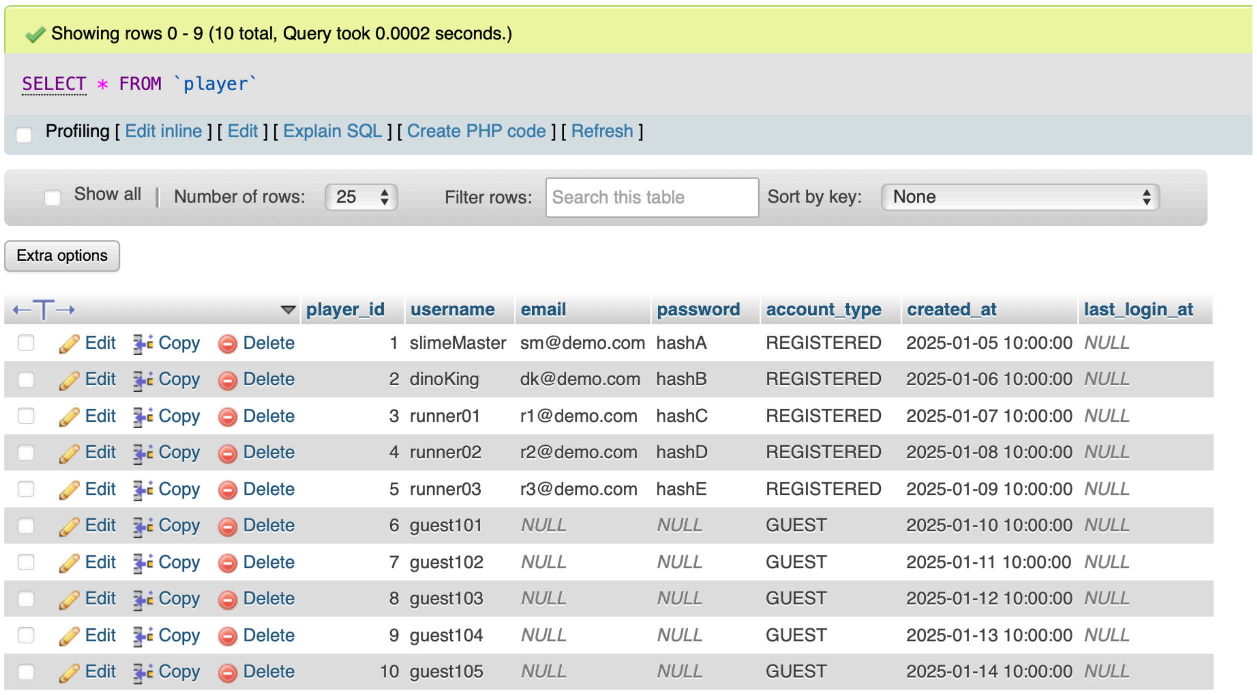
We are hosting our database on phpMyAdmin running locally. The database name is slime_runner_db and it contains all 10 tables required for our Slime Runner endless runner game project.

2. Database Tables and Sample Data

This section demonstrates that all required tables have been successfully created and populated with sufficient realistic data.

2.1 Player Table

The player table stores information about both registered users and guest players. It contains 10 rows with usernames, emails, passwords, account types, and timestamps. Registered players have email and password while guest accounts have NULL values for these fields.


Showing rows 0 - 9 (10 total, Query took 0.0002 seconds.)

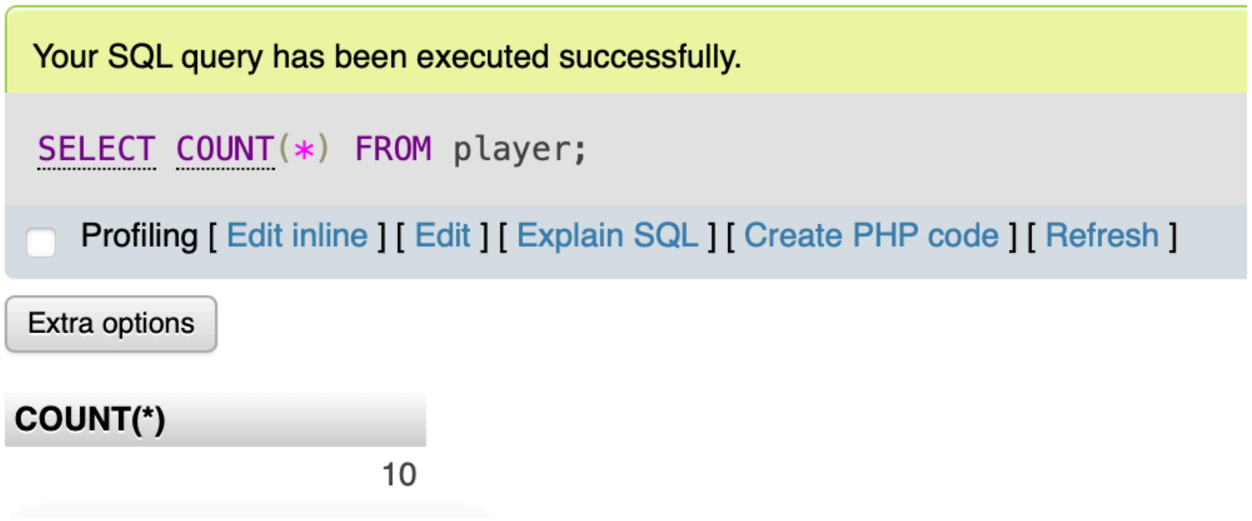
```
SELECT * FROM `player`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

player_id	username	email	password	account_type	created_at	last_login_at
1	slimeMaster	sm@demo.com	hashA	REGISTERED	2025-01-05 10:00:00	NULL
2	dinoKing	dk@demo.com	hashB	REGISTERED	2025-01-06 10:00:00	NULL
3	runner01	r1@demo.com	hashC	REGISTERED	2025-01-07 10:00:00	NULL
4	runner02	r2@demo.com	hashD	REGISTERED	2025-01-08 10:00:00	NULL
5	runner03	r3@demo.com	hashE	REGISTERED	2025-01-09 10:00:00	NULL
6	guest101	NULL	NULL	GUEST	2025-01-10 10:00:00	NULL
7	guest102	NULL	NULL	GUEST	2025-01-11 10:00:00	NULL
8	guest103	NULL	NULL	GUEST	2025-01-12 10:00:00	NULL
9	guest104	NULL	NULL	GUEST	2025-01-13 10:00:00	NULL
10	guest105	NULL	NULL	GUEST	2025-01-14 10:00:00	NULL


Your SQL query has been executed successfully.

```
SELECT COUNT(*) FROM player;
```

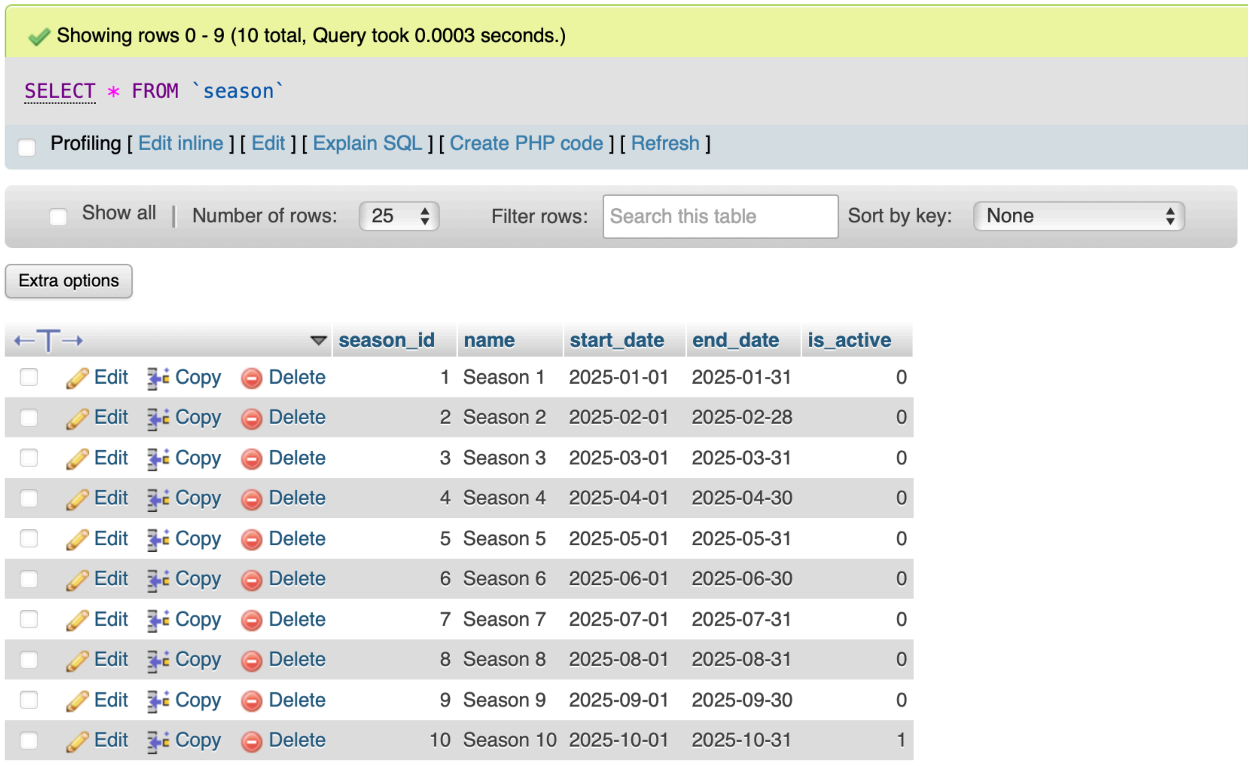
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Extra options

COUNT(*)
10

2.2 Season Table

The season table tracks different competition periods spanning January to October 2025. It contains 10 rows with season names, start dates, end dates, and an is_active flag indicating Season 10 is currently active.


Showing rows 0 - 9 (10 total, Query took 0.0003 seconds.)

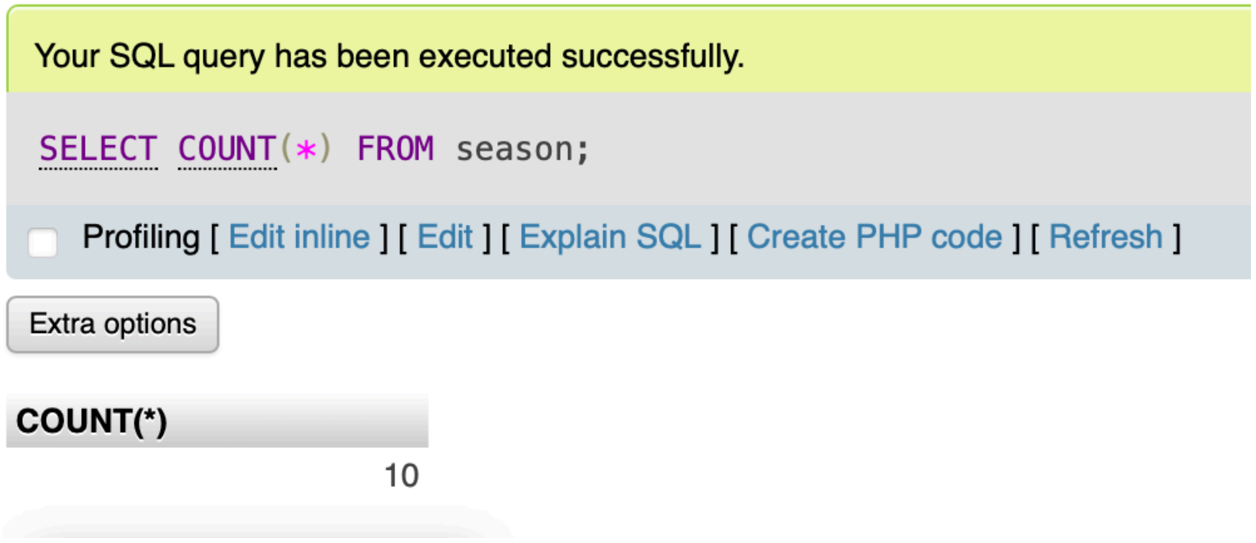
```
SELECT * FROM `season`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	season_id	name	start_date	end_date	is_active
Edit Copy Delete	1	Season 1	2025-01-01	2025-01-31	0
Edit Copy Delete	2	Season 2	2025-02-01	2025-02-28	0
Edit Copy Delete	3	Season 3	2025-03-01	2025-03-31	0
Edit Copy Delete	4	Season 4	2025-04-01	2025-04-30	0
Edit Copy Delete	5	Season 5	2025-05-01	2025-05-31	0
Edit Copy Delete	6	Season 6	2025-06-01	2025-06-30	0
Edit Copy Delete	7	Season 7	2025-07-01	2025-07-31	0
Edit Copy Delete	8	Season 8	2025-08-01	2025-08-31	0
Edit Copy Delete	9	Season 9	2025-09-01	2025-09-30	0
Edit Copy Delete	10	Season 10	2025-10-01	2025-10-31	1


Your SQL query has been executed successfully.

```
SELECT COUNT(*) FROM season;
```

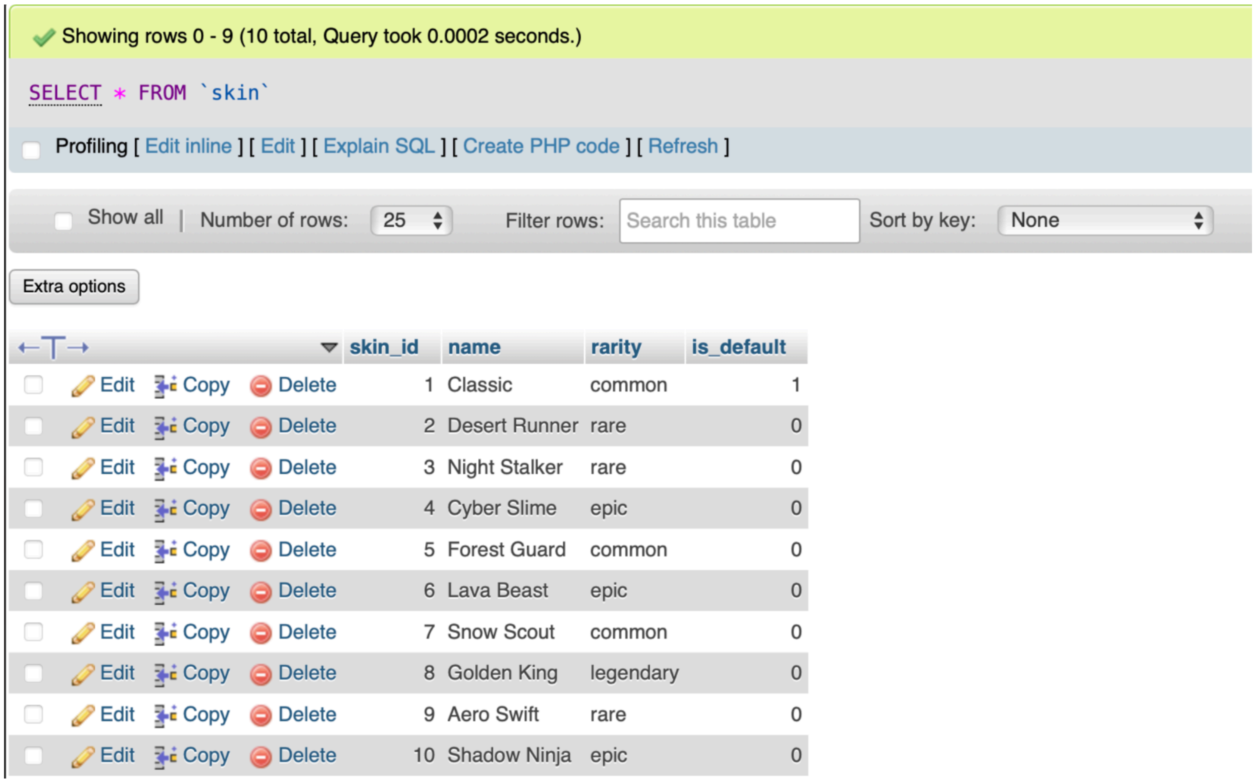
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Extra options

COUNT(*)
10

2.3 Skin Table

The skin table contains 10 different character skins with varying rarity levels from common to legendary. The Classic skin is marked as the default skin that all players own automatically.


Showing rows 0 - 9 (10 total, Query took 0.0002 seconds.)

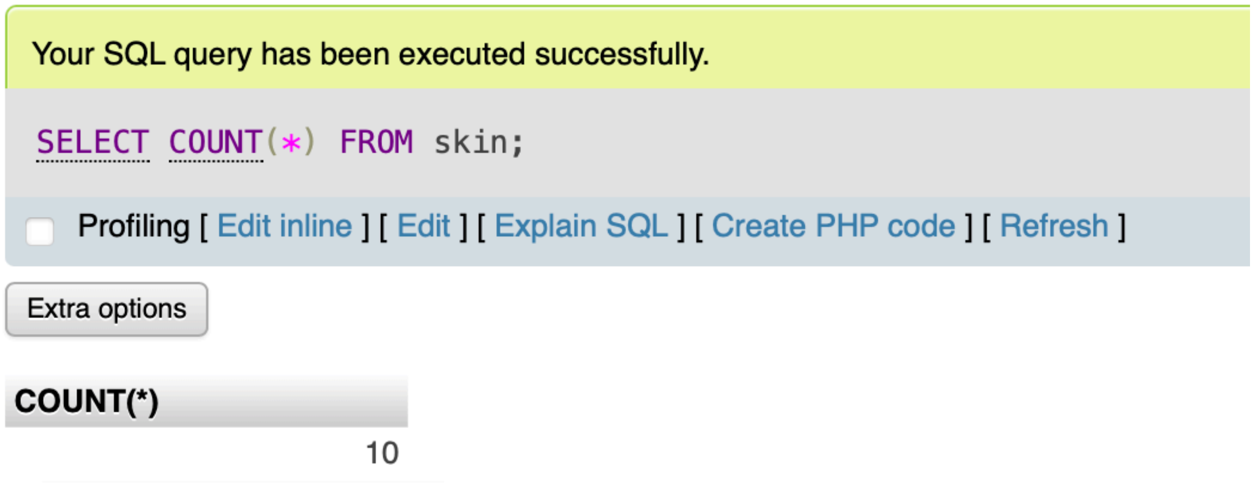
```
SELECT * FROM `skin`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	skin_id	name	rarity	is_default
Edit Copy Delete	1	Classic	common	1
Edit Copy Delete	2	Desert Runner	rare	0
Edit Copy Delete	3	Night Stalker	rare	0
Edit Copy Delete	4	Cyber Slime	epic	0
Edit Copy Delete	5	Forest Guard	common	0
Edit Copy Delete	6	Lava Beast	epic	0
Edit Copy Delete	7	Snow Scout	common	0
Edit Copy Delete	8	Golden King	legendary	0
Edit Copy Delete	9	Aero Swift	rare	0
Edit Copy Delete	10	Shadow Ninja	epic	0


Your SQL query has been executed successfully.

```
SELECT COUNT(*) FROM skin;
```

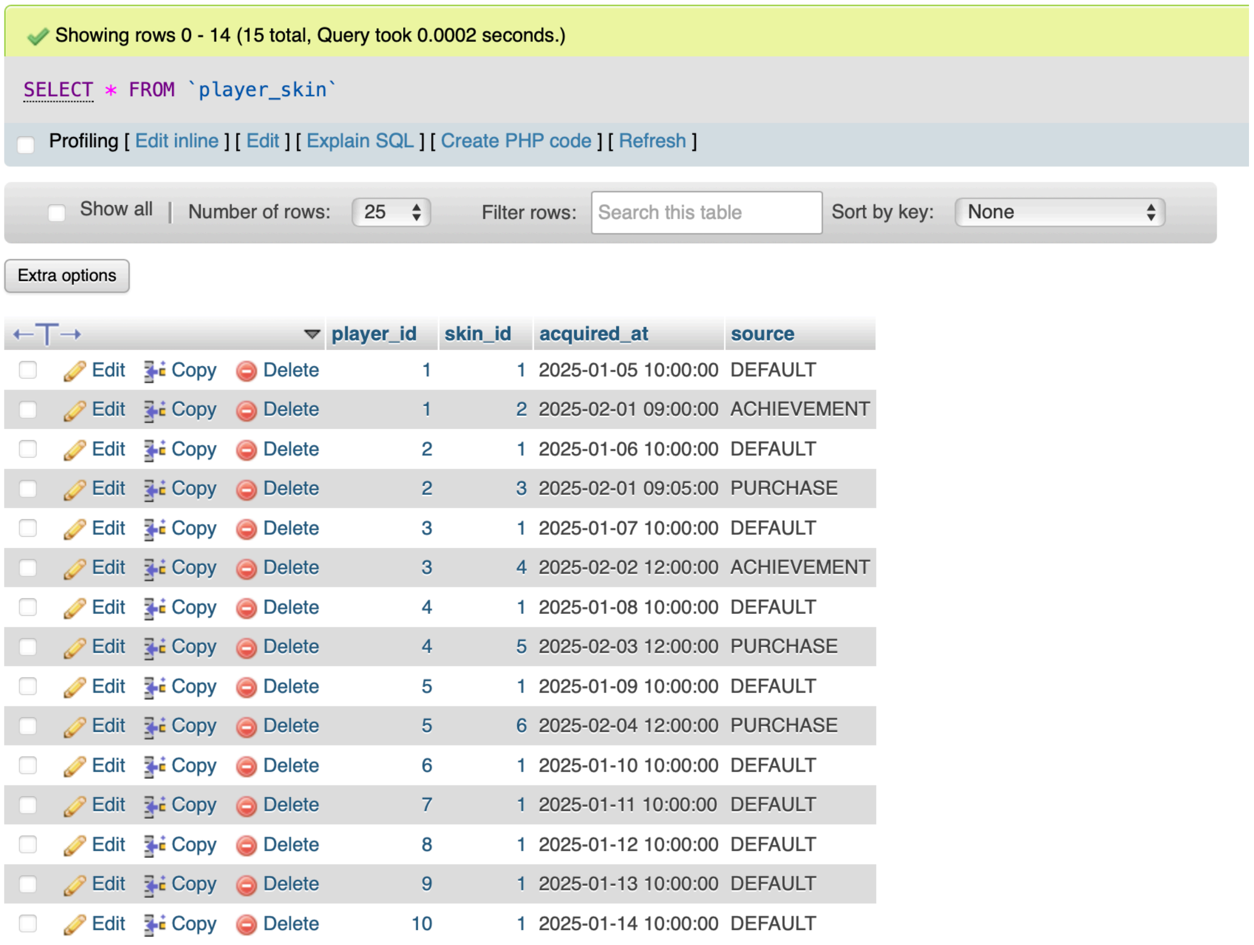
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Extra options

COUNT(*)
10

2.4 PlayerSkin Table

The player_skin table contains 15 rows tracking which skins each player owns and how they acquired them. All players own the default skin and some have acquired additional skins through achievements or purchases.

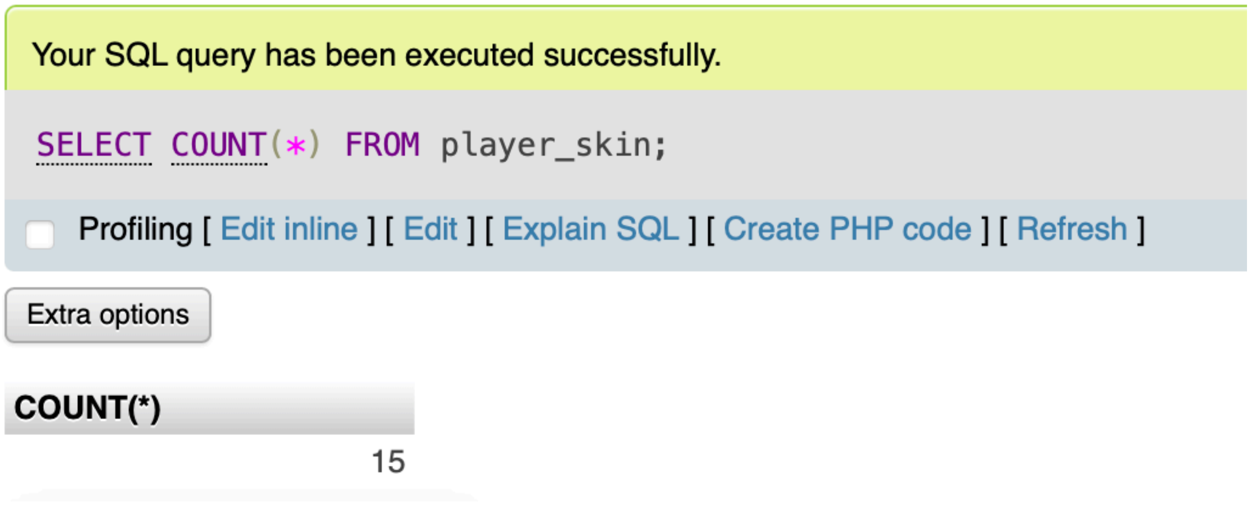


Showing rows 0 - 14 (15 total, Query took 0.0002 seconds.)

```
SELECT * FROM `player_skin`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Extra options

player_id	skin_id	acquired_at	source
1	1	2025-01-05 10:00:00	DEFAULT
1	2	2025-02-01 09:00:00	ACHIEVEMENT
2	1	2025-01-06 10:00:00	DEFAULT
2	3	2025-02-01 09:05:00	PURCHASE
3	1	2025-01-07 10:00:00	DEFAULT
3	4	2025-02-02 12:00:00	ACHIEVEMENT
4	1	2025-01-08 10:00:00	DEFAULT
4	5	2025-02-03 12:00:00	PURCHASE
5	1	2025-01-09 10:00:00	DEFAULT
5	6	2025-02-04 12:00:00	PURCHASE
6	1	2025-01-10 10:00:00	DEFAULT
7	1	2025-01-11 10:00:00	DEFAULT
8	1	2025-01-12 10:00:00	DEFAULT
9	1	2025-01-13 10:00:00	DEFAULT
10	1	2025-01-14 10:00:00	DEFAULT



Your SQL query has been executed successfully.

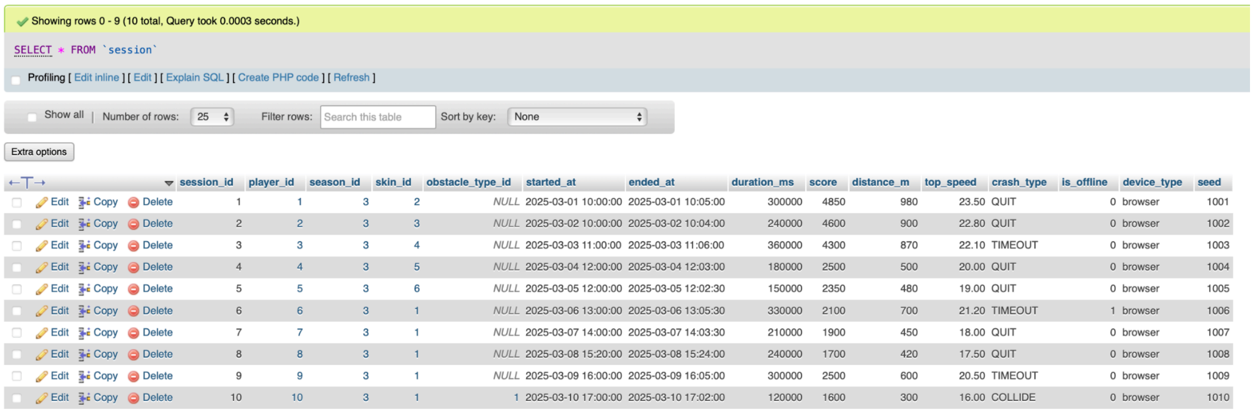
```
SELECT COUNT(*) FROM player_skin;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Extra options

COUNT(*)
15

2.5 Session Table

The session table contains 10 gameplay sessions with realistic score and distance data from Season 3. Each session includes player ID, season ID, skin used, timestamps, scores, distances, top speeds, and crash types. The duration_ms field is automatically calculated.



✓ Showing rows 0 - 9 (10 total, Query took 0.0003 seconds.)

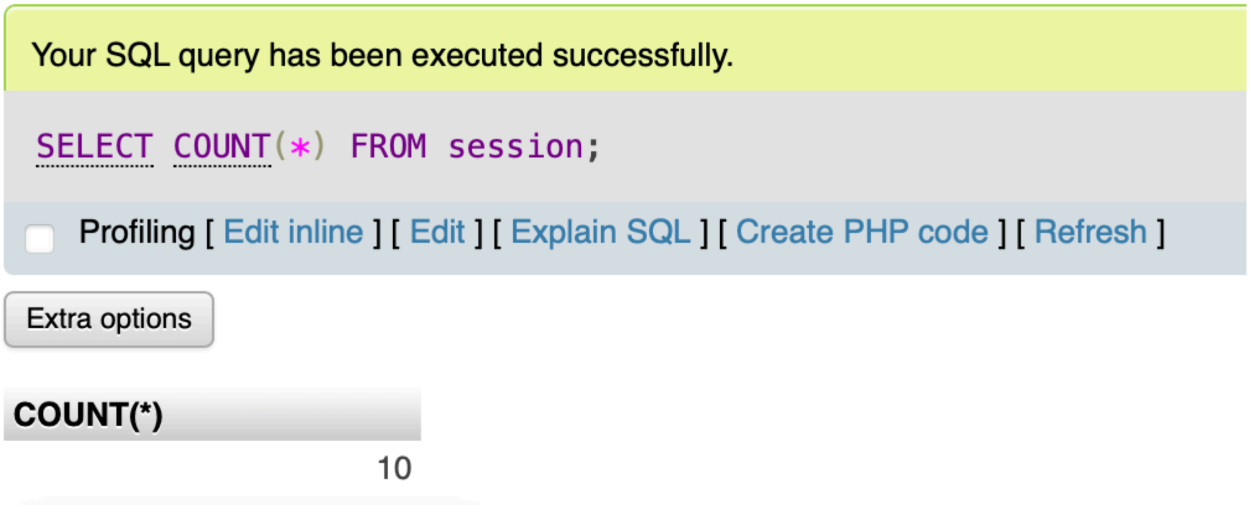
```
SELECT * FROM `session`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	session_id	player_id	season_id	skin_id	obstacle_type_id	started_at	ended_at	duration_ms	score	distance_m	top_speed	crash_type	is_offline	device_type	seed
Edit Copy Delete	1	1	3	2	NULL	2025-03-01 10:00:00	2025-03-01 10:05:00	300000	4850	980	23.50	QUIT	0	browser	1001
Edit Copy Delete	2	2	3	3	NULL	2025-03-02 10:00:00	2025-03-02 10:04:00	240000	4600	900	22.80	QUIT	0	browser	1002
Edit Copy Delete	3	3	3	4	NULL	2025-03-03 11:00:00	2025-03-03 11:06:00	360000	4300	870	22.10	TIMEOUT	0	browser	1003
Edit Copy Delete	4	4	3	5	NULL	2025-03-04 12:00:00	2025-03-04 12:03:00	180000	2500	500	20.00	QUIT	0	browser	1004
Edit Copy Delete	5	5	3	6	NULL	2025-03-05 12:00:00	2025-03-05 12:02:30	150000	2350	480	19.00	QUIT	0	browser	1005
Edit Copy Delete	6	6	3	1	NULL	2025-03-06 13:00:00	2025-03-06 13:05:30	330000	2100	700	21.20	TIMEOUT	1	browser	1006
Edit Copy Delete	7	7	3	1	NULL	2025-03-07 14:00:00	2025-03-07 14:03:30	210000	1900	450	18.00	QUIT	0	browser	1007
Edit Copy Delete	8	8	3	1	NULL	2025-03-08 15:20:00	2025-03-08 15:24:00	240000	1700	420	17.50	QUIT	0	browser	1008
Edit Copy Delete	9	9	3	1	NULL	2025-03-09 16:00:00	2025-03-09 16:05:00	300000	2500	600	20.50	TIMEOUT	0	browser	1009
Edit Copy Delete	10	10	3	1	1	2025-03-10 17:00:00	2025-03-10 17:02:00	120000	1600	300	16.00	COLLIDE	0	browser	1010



Your SQL query has been executed successfully.

```
SELECT COUNT(*) FROM session;
```

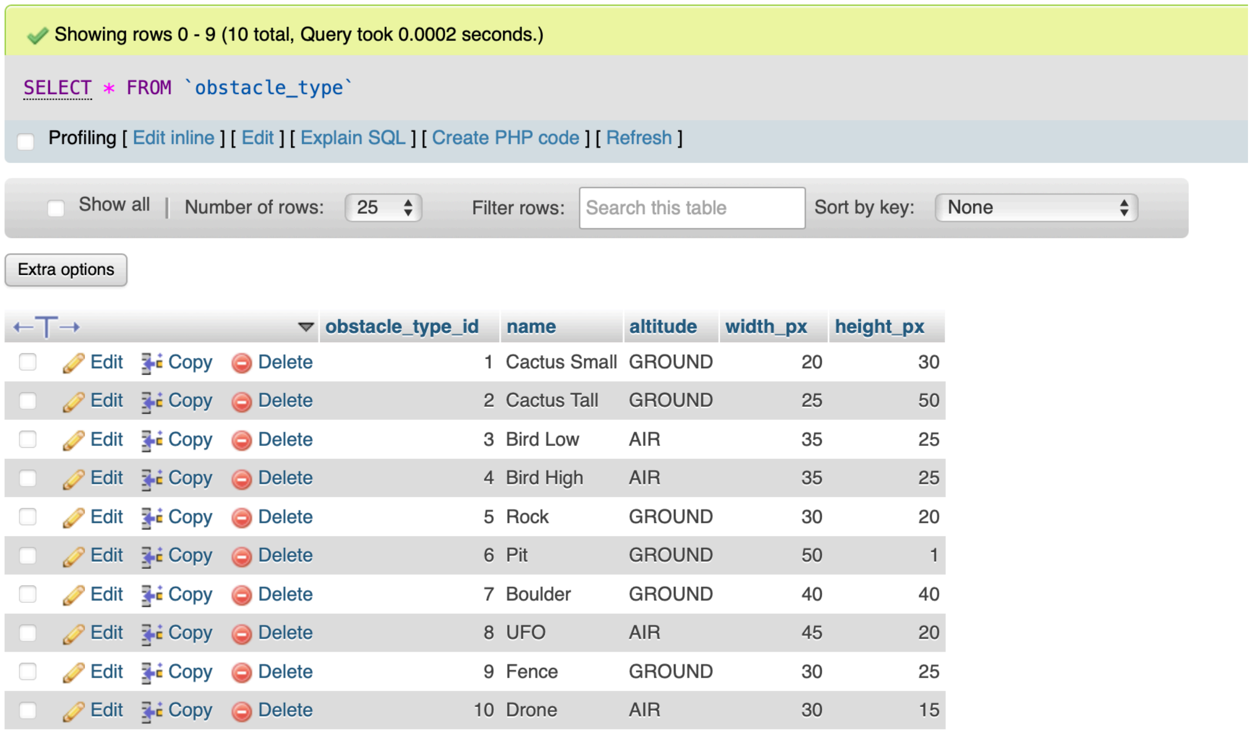
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Extra options

COUNT(*)
10

2.6 ObstacleType Table

The obstacle_type table defines 10 different obstacles with varying sizes and altitudes. Some are ground obstacles like cacti and rocks while others are air obstacles like birds and drones. Each has width and height dimensions in pixels.

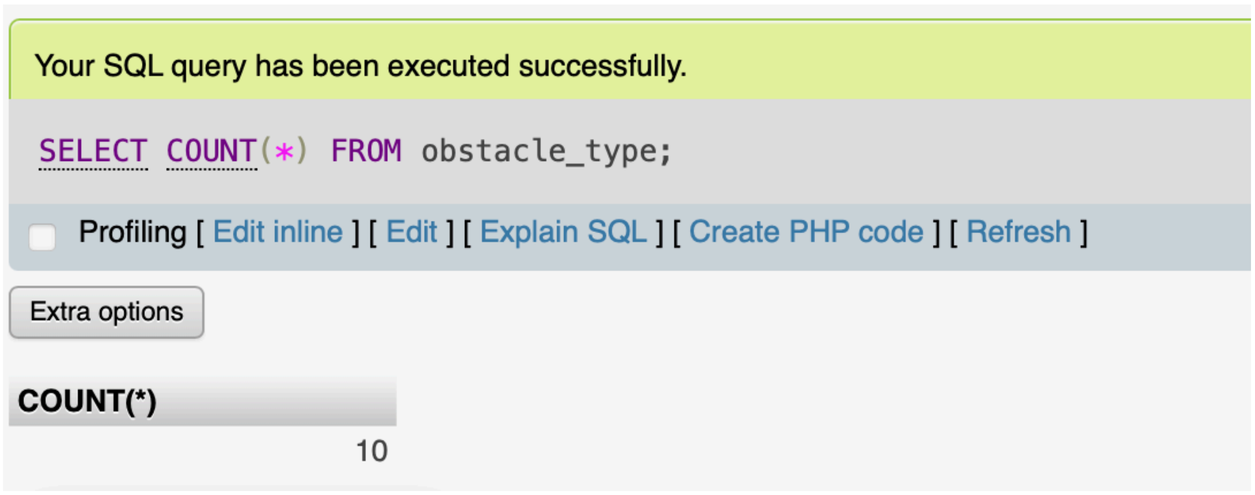


Showing rows 0 - 9 (10 total, Query took 0.0002 seconds.)

```
SELECT * FROM `obstacle_type`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Extra options

obstacle_type_id	name	altitude	width_px	height_px
1	Cactus Small	GROUND	20	30
2	Cactus Tall	GROUND	25	50
3	Bird Low	AIR	35	25
4	Bird High	AIR	35	25
5	Rock	GROUND	30	20
6	Pit	GROUND	50	1
7	Boulder	GROUND	40	40
8	UFO	AIR	45	20
9	Fence	GROUND	30	25
10	Drone	AIR	30	15



Your SQL query has been executed successfully.

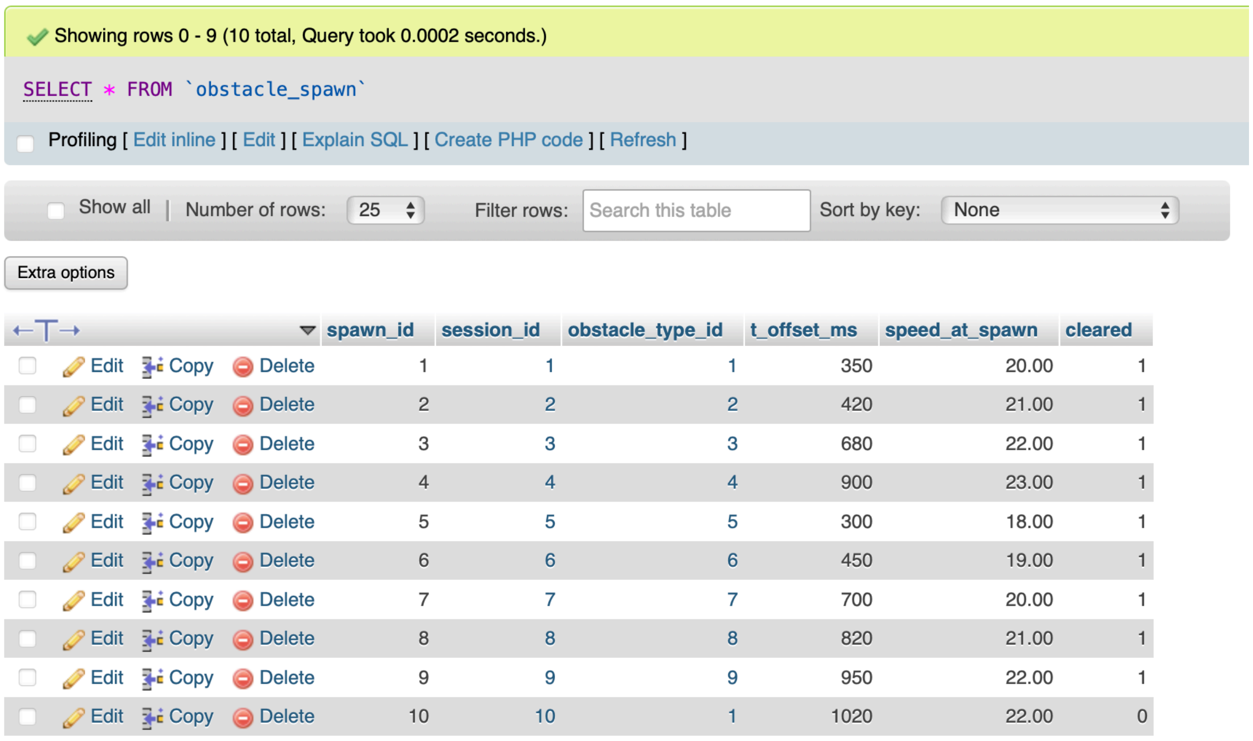
```
SELECT COUNT(*) FROM obstacle_type;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Extra options

COUNT(*)
10

2.7 ObstacleSpawn Table

The obstacle_spawn table contains 10 rows recording when obstacles appeared during gameplay sessions. Each entry includes timing offsets, spawn speeds, and whether the obstacle was cleared. Most obstacles were cleared except one that caused a collision.


Showing rows 0 - 9 (10 total, Query took 0.0002 seconds.)

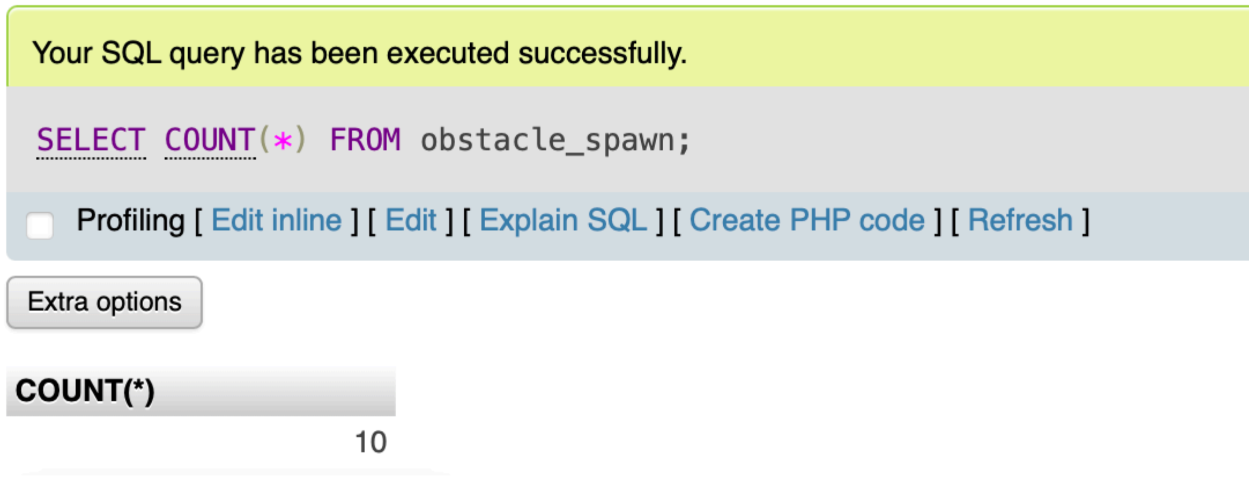
```
SELECT * FROM `obstacle_spawn`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	spawn_id	session_id	obstacle_type_id	t_offset_ms	speed_at_spawn	cleared
Edit Copy Delete	1	1	1	350	20.00	1
Edit Copy Delete	2	2	2	420	21.00	1
Edit Copy Delete	3	3	3	680	22.00	1
Edit Copy Delete	4	4	4	900	23.00	1
Edit Copy Delete	5	5	5	300	18.00	1
Edit Copy Delete	6	6	6	450	19.00	1
Edit Copy Delete	7	7	7	700	20.00	1
Edit Copy Delete	8	8	8	820	21.00	1
Edit Copy Delete	9	9	9	950	22.00	1
Edit Copy Delete	10	10	1	1020	22.00	0


Your SQL query has been executed successfully.

```
SELECT COUNT(*) FROM obstacle_spawn;
```

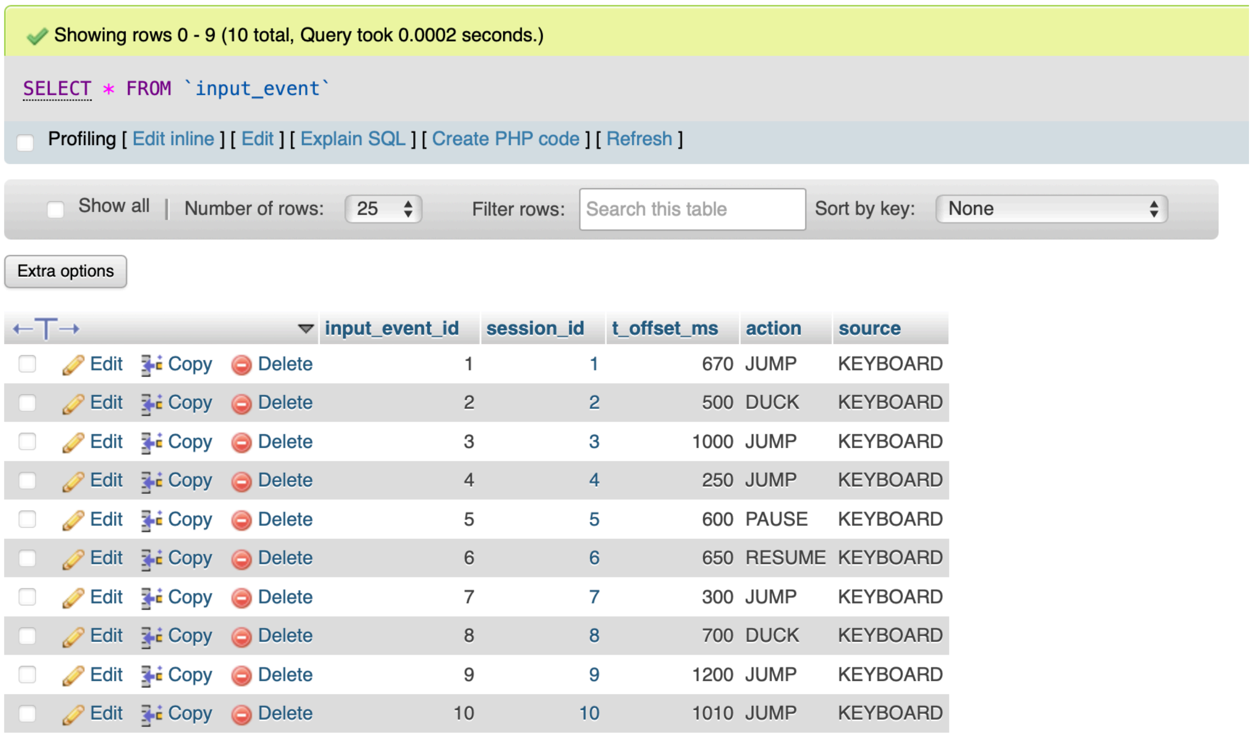
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Extra options

COUNT(*)
10

2.8 InputEvent Table

The input_event table captures 10 player actions with precise timing information. Actions include jumps, ducks, pauses, and resumes with their time offsets and input sources like keyboard or touch.



✓ Showing rows 0 - 9 (10 total, Query took 0.0002 seconds.)

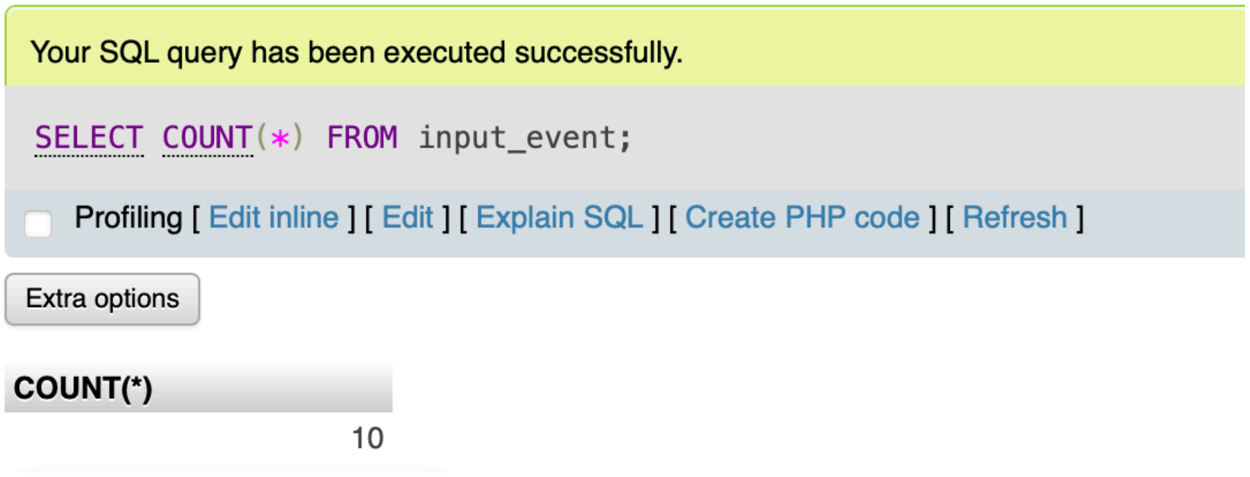
```
SELECT * FROM `input_event`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	input_event_id	session_id	t_offset_ms	action	source
Edit Copy Delete	1	1	670	JUMP	KEYBOARD
Edit Copy Delete	2	2	500	DUCK	KEYBOARD
Edit Copy Delete	3	3	1000	JUMP	KEYBOARD
Edit Copy Delete	4	4	250	JUMP	KEYBOARD
Edit Copy Delete	5	5	600	PAUSE	KEYBOARD
Edit Copy Delete	6	6	650	RESUME	KEYBOARD
Edit Copy Delete	7	7	300	JUMP	KEYBOARD
Edit Copy Delete	8	8	700	DUCK	KEYBOARD
Edit Copy Delete	9	9	1200	JUMP	KEYBOARD
Edit Copy Delete	10	10	1010	JUMP	KEYBOARD



Your SQL query has been executed successfully.

```
SELECT COUNT(*) FROM input_event;
```

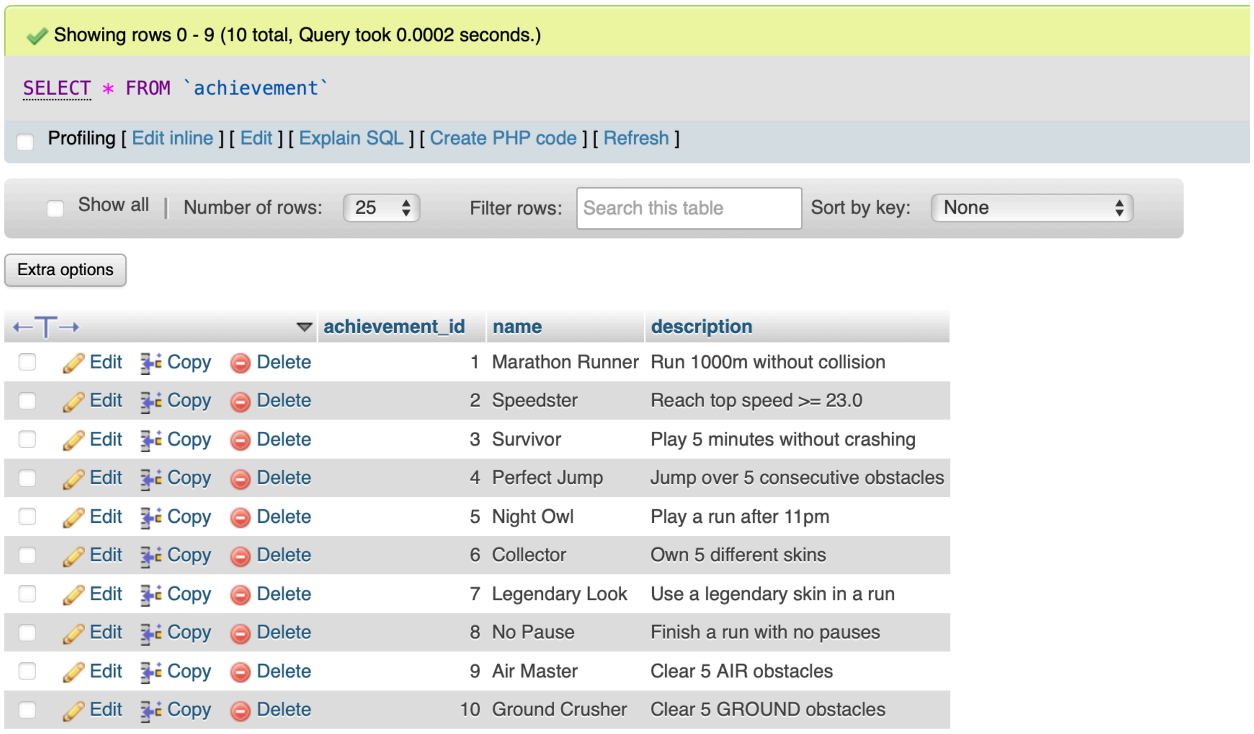
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Extra options

COUNT(*)
10

2.9 Achievement Table

The achievement table defines 10 unlockable achievements with varying difficulty levels. These include distance milestones, speed challenges, collection goals, and gameplay requirements.


Showing rows 0 - 9 (10 total, Query took 0.0002 seconds.)

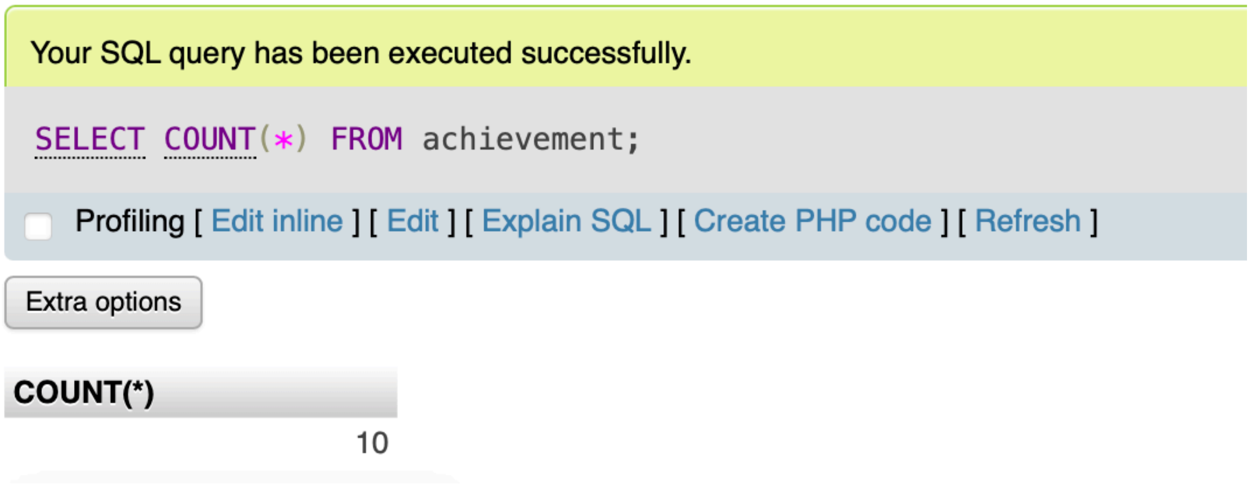
```
SELECT * FROM `achievement`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

achievement_id	name	description
1	Marathon Runner	Run 1000m without collision
2	Speedster	Reach top speed >= 23.0
3	Survivor	Play 5 minutes without crashing
4	Perfect Jump	Jump over 5 consecutive obstacles
5	Night Owl	Play a run after 11pm
6	Collector	Own 5 different skins
7	Legendary Look	Use a legendary skin in a run
8	No Pause	Finish a run with no pauses
9	Air Master	Clear 5 AIR obstacles
10	Ground Crusher	Clear 5 GROUND obstacles


Your SQL query has been executed successfully.

```
SELECT COUNT(*) FROM achievement;
```

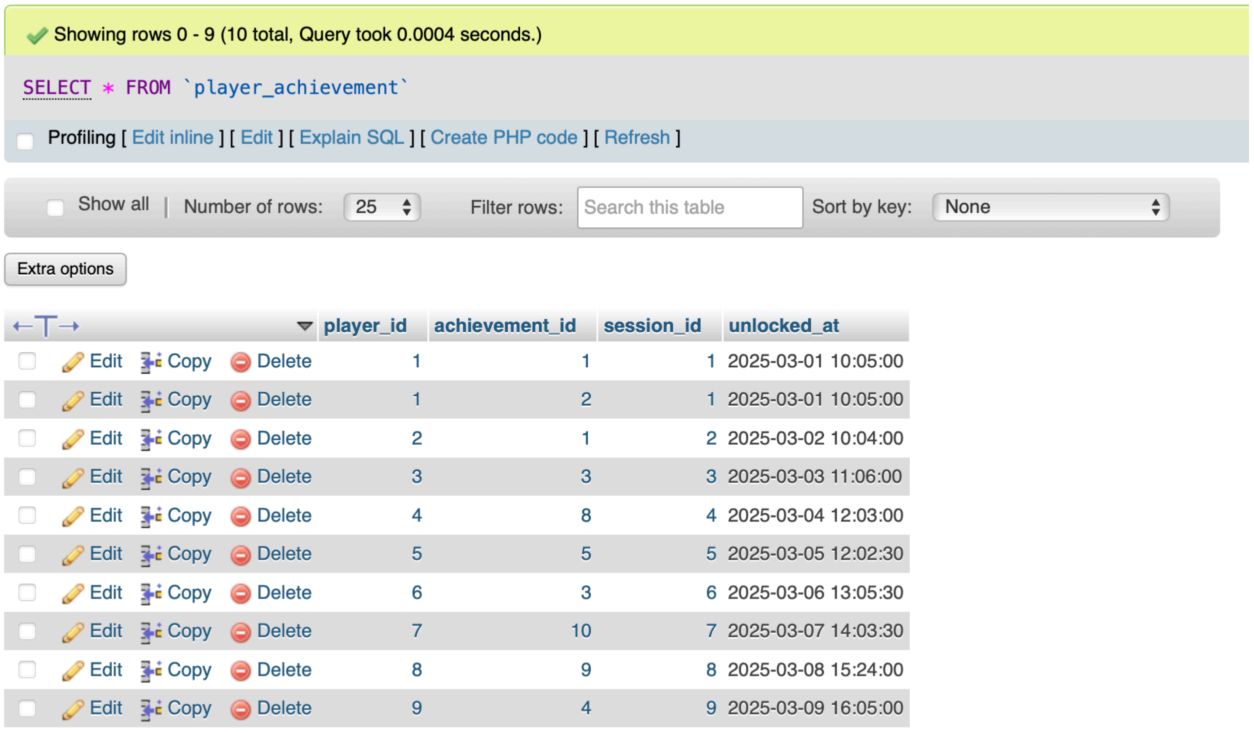
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Extra options

COUNT(*)
10

2.10 PlayerAchievement Table

The player_achievement table contains 10 rows showing which achievements players have unlocked. Each entry links to the specific player, achievement, session where it was earned, and timestamp.


Showing rows 0 - 9 (10 total, Query took 0.0004 seconds.)

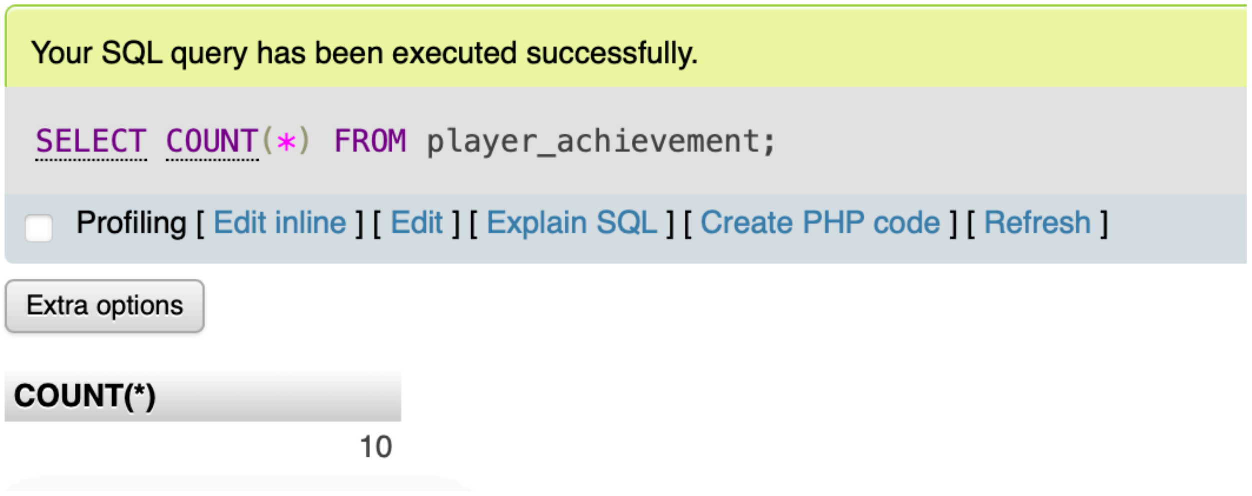
```
SELECT * FROM `player_achievement`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

player_id	achievement_id	session_id	unlocked_at
1	1	1	2025-03-01 10:05:00
1	2	1	2025-03-01 10:05:00
2	1	2	2025-03-02 10:04:00
3	3	3	2025-03-03 11:06:00
4	8	4	2025-03-04 12:03:00
5	5	5	2025-03-05 12:02:30
6	3	6	2025-03-06 13:05:30
7	10	7	2025-03-07 14:03:30
8	9	8	2025-03-08 15:24:00
9	4	9	2025-03-09 16:05:00


Your SQL query has been executed successfully.

```
SELECT COUNT(*) FROM player_achievement;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Extra options

COUNT(*)
10

3. SQL Commands Used in Our Application

This section documents all SQL commands that our application uses to interact with the database.

3.1 Non-Advanced SQL Commands

Create Commands

All 10 tables were created using CREATE TABLE statements with appropriate data types, primary keys, foreign keys, and constraints. The complete statements are in COMMANDS.sql.

Insert Commands

We populated each table with realistic sample data. Examples include:

```
INSERT INTO player (username, email, password, account_type, created_at) VALUES
('slimeMaster','sm@demo.com','hashA','REGISTERED','2025-01-05 10:00:00');
```

```
INSERT INTO season (name, start_date, end_date, is_active) VALUES ('Season
1','2025-01-01','2025-01-31',0);
```

```
INSERT INTO skin (name, rarity, is_default) VALUES ('Classic', 'common', 1);
```

All INSERT statements are included in COMMANDS.sql.

Select Commands

Our application uses various SELECT queries:

Get all registered players:
```
SELECT * FROM player WHERE account_type = 'REGISTERED';
```

Get leaderboard for a specific season:
```
SELECT p.username, s.score, s.distance_m, s.top_speed FROM session s JOIN player p ON
s.player_id = p.player_id WHERE s.season_id = 3 ORDER BY s.score DESC LIMIT 10;
```

Get achievements for a player:
```
SELECT a.name, a.description, pa.unlocked_at FROM player_achievement pa JOIN
achievement a ON pa.achievement_id = a.achievement_id WHERE pa.player_id = 1;
```

Get skins owned by a player:
```
SELECT s.name, s.rarity, ps.acquired_at, ps.source FROM player_skin ps JOIN skin s ON
ps.skin_id = s.skin_id WHERE ps.player_id = 1;
```

Get active season:
```
SELECT * FROM season WHERE is_active = 1;
```

Get session history:
```
SELECT session_id, score, distance_m, started_at, crash_type FROM session WHERE
player_id = 1 ORDER BY started_at DESC;
```

Update Commands

Update player last login:
```
UPDATE player SET last_login_at = NOW() WHERE player_id = 1;
```

Update session score:
```
UPDATE session
```

```
SET ended_at = NOW(),

    score = 5000,

    distance_m = 1000,

    top_speed = 25.5,

    crash_type = 'COLLIDE',

    obstacle_type_id = 1    -- keep crash info consistent

WHERE session_id = 1;
```

Activate new season:

```
UPDATE season SET is_active = 0 WHERE is_active = 1;
UPDATE season SET is_active = 1 WHERE season_id = 10;
```

Delete Commands

Delete guest player:

```
DELETE FROM player WHERE player_id = 10 AND account_type = 'GUEST';
```

Delete old sessions:

```
DELETE FROM session s

WHERE s.season_id < 3

  AND NOT EXISTS (

    SELECT 1

    FROM player_achievement pa

    WHERE pa.session_id = s.session_id

  );
```

3.2 Advanced PL/SQL Commands

Advanced Feature 1: STORED PROCEDURE

Purpose: Validates that crash data in a session is consistent with the obstacle spawn records.

Parameters:

p_session_id (INT): The session ID to validate

Returns: None (raises an error if validation fails)

Business Logic: This procedure enforces complex validation rules for session crashes. If a session crashed due to collision (COLLIDE), there must be exactly one uncleared obstacle spawn and the obstacle_type_id must match between the session and the uncleared spawn. For non-collision crashes (QUIT or TIMEOUT), all obstacle spawns must be marked as cleared. This ensures data consistency between the session and obstacle_spawn tables.

```
-- USE dino_runner;

DELIMITER //

CREATE PROCEDURE sp_validate_session_crash(IN p_session_id INT)

BEGIN

DECLARE v_crash VARCHAR(10); DECLARE v_obst INT; DECLARE v_uncleared INT;
DECLARE v_match INT;

SELECT crash_type, obstacle_type_id INTO v_crash, v_obst FROM session WHERE
session_id = p_session_id;

IF v_crash IS NULL THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Session not
found'; END IF;

IF v_crash = 'COLLIDE' THEN

SELECT SUM(CASE WHEN cleared = 0 THEN 1 ELSE 0 END) INTO v_uncleared FROM
obstacle_spawn WHERE session_id = p_session_id;

IF v_uncleared <> 1 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'COLLIDE
must have exactly one uncleared spawn'; END IF;

SELECT COUNT(*) INTO v_match FROM obstacle_spawn WHERE session_id = p_session_id
AND cleared = 0 AND obstacle_type_id = v_obst;

IF v_match <> 1 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Crash
obstacle_type_id mismatch'; END IF;

ELSE

SELECT SUM(CASE WHEN cleared = 0 THEN 1 ELSE 0 END) INTO v_uncleared FROM
obstacle_spawn WHERE session_id = p_session_id;

IF v_uncleared <> 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Non-collide
session cannot contain uncleared spawns'; END IF;
```

END IF;

END//

DELIMITER ;

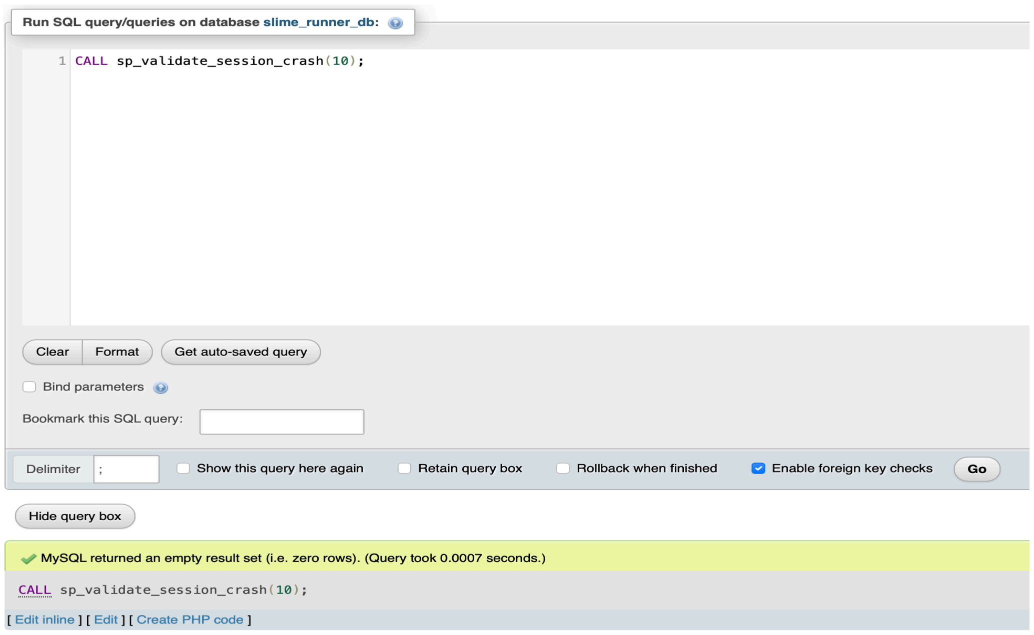
Procedure Testing:

Test 1: CALL sp_validate_session_crash(10);

Result: Success because session 10 has crash_type COLLIDE with exactly one uncleared obstacle spawn and matching obstacle_type_id

Test 2: CALL sp_validate_session_crash(1);

Result: Success because session 1 has crash_type QUIT with all obstacle spawns cleared


Run SQL query/queries on database slime_runner_db:

```
CALL sp_validate_session_crash(10);
```

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0007 seconds.)

```
CALL sp_validate_session_crash(10);
```

Advanced Feature 2: FUNCTION

fn_player_owns_skin Function

Purpose: Checks whether a player owns a specific skin at a given point in time.

Parameters:
p_player_id (INT): The player ID to check
p_skin_id (INT): The skin ID to verify
p_at (DATETIME): The timestamp to check ownership

Returns: BOOLEAN (TRUE if player owns the skin, FALSE otherwise)

The function first checks if the skin is the default skin and returns TRUE since all players own it. Otherwise it queries the player_skin table to verify if the player acquired that skin before the specified time.

Code:

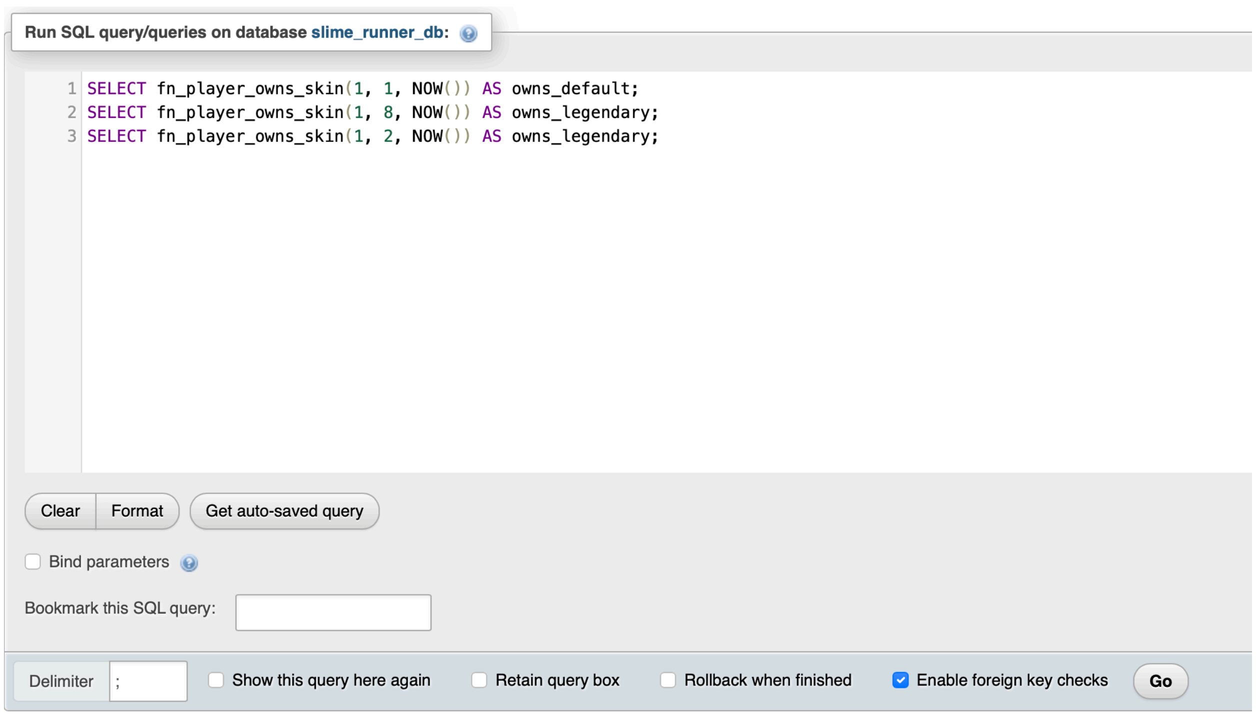
```
DELIMITER //
CREATE FUNCTION fn_player_owns_skin(p_player_id INT, p_skin_id INT, p_at DATETIME)
RETURNS BOOLEAN
DETERMINISTIC
BEGIN
DECLARE has_default BOOLEAN;
DECLARE owned_count INT;

SELECT is_default INTO has_default FROM skin WHERE skin_id = p_skin_id;
IF has_default = 1 THEN
RETURN TRUE;
END IF;

SELECT COUNT(*) INTO owned_count FROM player_skin WHERE player_id = p_player_id
AND skin_id = p_skin_id AND acquired_at <= p_at;

RETURN owned_count > 0;
END//
DELIMITER ;
```

Function Testing:

Test 1: SELECT fn_player_owns_skin(1, 1, NOW()) AS owns_default;
Result: 1 (TRUE) because player 1 owns the default skin

Test 2: SELECT fn_player_owns_skin(1, 8, NOW()) AS owns_legendary;
Result: 0 (FALSE) because player 1 does not own the legendary skin

Test 3: SELECT fn_player_owns_skin(1, 2, NOW()) AS owns_rare;
Result: 1 (TRUE) because player 1 acquired skin 2 on 2025-02-01


Run SQL query/queries on database slime_runner_db:

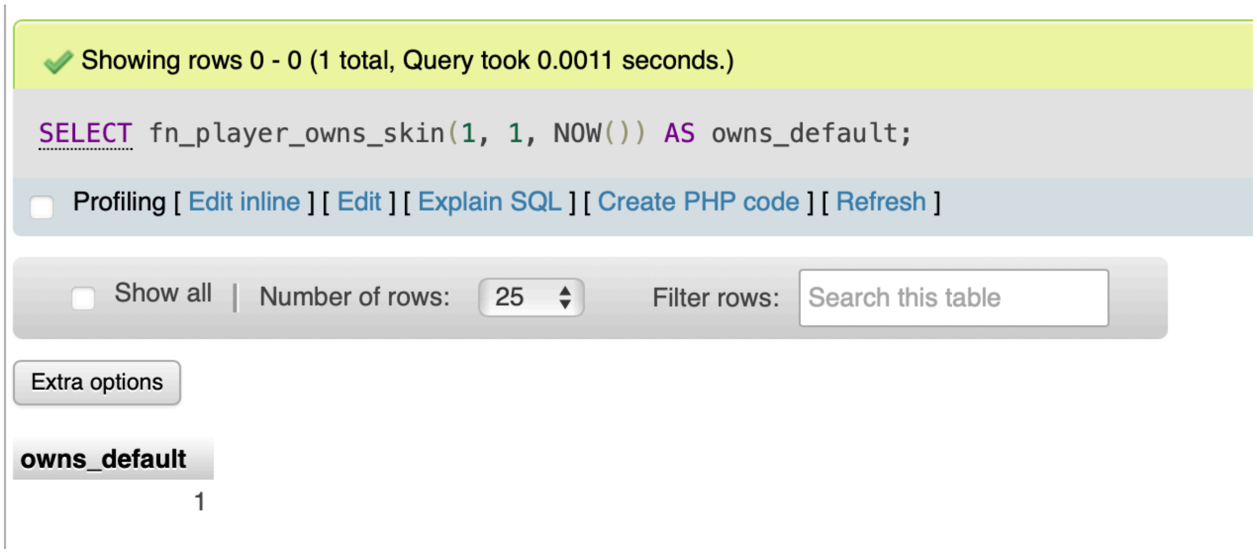
```sql
SELECT fn_player_owns_skin(1, 1, NOW()) AS owns_default;
SELECT fn_player_owns_skin(1, 8, NOW()) AS owns_legendary;
SELECT fn_player_owns_skin(1, 2, NOW()) AS owns_legendary;
```

[ ] Bind parameters

Bookmark this SQL query:

Delimiter ;

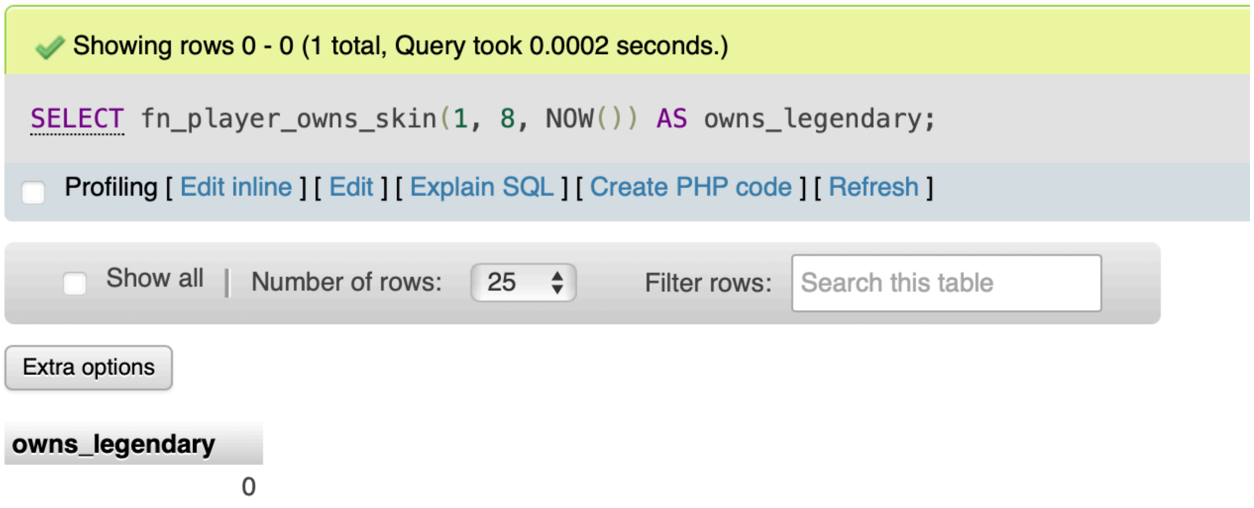
[ ] Show this query here again
[ ] Retain query box
[ ] Rollback when finished
[✓] Enable foreign key checks


Showing rows 0 - 0 (1 total, Query took 0.0011 seconds.)

```sql
SELECT fn_player_owns_skin(1, 1, NOW()) AS owns_default;
```

[ ] Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
[ ] Show all | Number of rows: 25 Filter rows: Search this table
Extra options

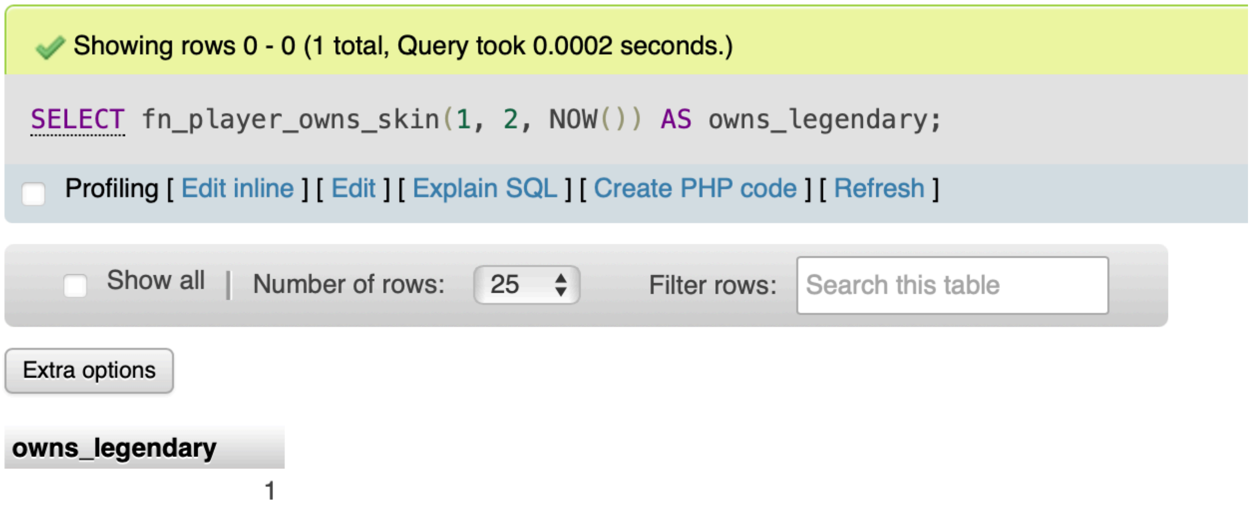
owns_default
1


Showing rows 0 - 0 (1 total, Query took 0.0002 seconds.)

```
SELECT fn_player_owns_skin(1, 8, NOW()) AS owns_legendary;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25 Filter rows: Search this table
Extra options

owns_legendary
0


Showing rows 0 - 0 (1 total, Query took 0.0002 seconds.)

```
SELECT fn_player_owns_skin(1, 2, NOW()) AS owns_legendary;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25 Filter rows: Search this table
Extra options

owns_legendary
1

All tests passed successfully confirming the function works correctly.

4. Conclusion

We have successfully set up our MySQL database with all 10 required tables and populated them with realistic data. All tables contain at least 10 rows of practical data demonstrating our game functionality. We have implemented comprehensive SQL commands including CREATE, INSERT, SELECT, UPDATE, and DELETE operations. We implemented two different advanced SQL features: a stored procedure and a function (we also use CHECK constraints for integrity).