Database Exam Solutions – MySQL (Sakila Database)

Question 1 – Solution (Function)

```
DELIMITER $$
CREATE FUNCTION get_customer_active_rentals(p_customer_id SMALLINT UNSIGNED)
RETURNS INT
BEGIN
  DECLARE v_count INT DEFAULT 0;
  DECLARE v_exists INT DEFAULT 0;
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    RETURN -1;
  END;

  SELECT COUNT(*) INTO v_exists FROM customer WHERE customer_id = p_customer_id;
  IF v_exists = 0 THEN
    RETURN -1;
  END IF;

  SELECT COUNT(*) INTO v_count
  FROM rental r
  WHERE r.customer_id = p_customer_id
    AND r.return_date IS NULL;

  RETURN v_count;
END$$
DELIMITER ;
```

Question 2 – Solution (Procedure)

```
DELIMITER $$
CREATE PROCEDURE return_film(
  IN p_rental_id INT,
  IN p_staff_id TINYINT UNSIGNED
)
BEGIN
  DECLARE v_customer_id SMALLINT UNSIGNED;
  DECLARE v_inventory_id MEDIUMINT UNSIGNED;
  DECLARE v_film_id SMALLINT UNSIGNED;
  DECLARE v_rental_rate DECIMAL(5,2);
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Unexpected SQL error.';
  END;

  SELECT customer_id, inventory_id, return_date
  INTO v_customer_id, v_inventory_id, @tmp_return
  FROM rental WHERE rental_id = p_rental_id;

  IF v_customer_id IS NULL THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Rental not found.';
  END IF;

  IF @tmp_return IS NOT NULL THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Rental already returned.';
  END IF;

  SELECT f.film_id, f.rental_rate
  INTO v_film_id, v_rental_rate
  FROM inventory i
  JOIN film f ON f.film_id = i.film_id
  WHERE i.inventory_id = v_inventory_id;

  IF v_film_id IS NULL THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Film not found.';
  END IF;

  IF NOT EXISTS (SELECT 1 FROM staff WHERE staff_id = p_staff_id) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid staff.';
  END IF;

  START TRANSACTION;
    UPDATE rental SET return_date = NOW() WHERE rental_id = p_rental_id;
    INSERT INTO payment (customer_id, staff_id, rental_id, amount, payment_date)
    VALUES (v_customer_id, p_staff_id, p_rental_id, v_rental_rate, NOW());
  COMMIT;
END$$
DELIMITER ;
```

Question 3 – Solution (Trigger)

```
CREATE TABLE customer_status_log (
  log_id INT AUTO_INCREMENT PRIMARY KEY,
  customer_id SMALLINT UNSIGNED NOT NULL,
  old_status TINYINT(1) NOT NULL,
  new_status TINYINT(1) NOT NULL,
  change_date DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (customer_id) REFERENCES customer(customer_id)
);

DELIMITER $$
CREATE TRIGGER after_customer_status_change
AFTER UPDATE ON customer
FOR EACH ROW
BEGIN
  IF OLD.active <> NEW.active THEN
    INSERT INTO customer_status_log (customer_id, old_status, new_status, change_date)
    VALUES (OLD.customer_id, OLD.active, NEW.active, NOW());
  END IF;
END$$
DELIMITER ;
```

Question 4 – Solution (Procedure + Loop)

```
DELIMITER $$
CREATE PROCEDURE add_film_to_store(
  IN p_film_id SMALLINT UNSIGNED,
  IN p_store_id TINYINT UNSIGNED,
  IN p_quantity INT
)
BEGIN
  DECLARE v_i INT DEFAULT 0;
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error adding inventory.';
  END;

  IF p_quantity <= 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Quantity must be > 0';
  END IF;

  IF NOT EXISTS (SELECT 1 FROM film WHERE film_id = p_film_id) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Film not found.';
  END IF;

  IF NOT EXISTS (SELECT 1 FROM store WHERE store_id = p_store_id) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Store not found.';
  END IF;

  START TRANSACTION;
    WHILE v_i < p_quantity DO
      INSERT INTO inventory (film_id, store_id, last_update)
      VALUES (p_film_id, p_store_id, NOW());
      SET v_i = v_i + 1;
    END WHILE;
  COMMIT;
END$$
DELIMITER ;
```

Question 5 – Solution (View)

```
CREATE VIEW top_renting_customers AS
SELECT
  cu.customer_id,
  CONCAT(cu.first_name, ' ', cu.last_name) AS customer_name,
  COUNT(r.rental_id) AS num_rentals,
  ci.city AS city
FROM customer cu
JOIN rental r ON cu.customer_id = r.customer_id
JOIN address a ON cu.address_id = a.address_id
JOIN city ci ON a.city_id = ci.city_id
GROUP BY cu.customer_id, customer_name, ci.city;
```