Übungsblatt 2

1. Gespeicherte Programme

a. flighttime

```
DELIMITER //

CREATE FUNCTION `flighttime` (p_departure DATETIME, p_arrival DATETIME)
RETURNS VARCHAR(255)
BEGIN
  DECLARE v_output VARCHAR(255) DEFAULT '';
  DECLARE v_diff TIME;
  DECLARE v_hours, v_minutes INT;
  SET v_diff = TIMEDIFF(p_arrival, p_departure);
  SET v_hours = HOUR(v_diff);
  SET v_minutes = MINUTE(v_diff);

  IF v_hours > 0 THEN SET v_output = CONCAT(v_hours, ' hour ');
  ELSEIF v_hours > 1 THEN SET v_output = CONCAT(v_output, 'hours ');
  END IF;
  SET v_output = CONCAT(v_output, v_minutes, ' minute');
  IF v_minutes <> 1 THEN SET v_output = CONCAT(v_output, 's');
  END IF;

  RETURN v_output;
END //

DELIMITER ;
```

b. book_asap

Leider wurden mit bei meiner Prozedur immer 2 Datensätze in die Tabelle buchung eingefügt. Ich habe leider keine Ahnung warum das so ist.

Des Weiteren wusste ich nicht, welchen Preis ich setzen soll, da dieser nicht als Parameter übergeben wird. Ich habe ihn deshalb einfach auf 0.0 gesetzt.

```
CREATE TABLE `buchung` LIKE `FlughafenDB`.`buchung`;
DELIMITER //
CREATE PROCEDURE `book_asap`(IN p_start CHAR(3), IN p_destination CHAR(3),
IN p_time DATETIME, IN p_pass CHAR(9))
BEGIN
  DECLARE v_flightnr CHAR(8);
  DECLARE v_flight_id INT(11);
  DECLARE no_flight_found CONDITION FOR SQLSTATE '45000';
  DECLARE v_passenger_id CHAR(9);
  DECLARE invalid_pass CONDITION FOR SQLSTATE '45001';

  SELECT `passagier_id` INTO v_passenger_id
    FROM `FlughafenDB`.`passagier`
    WHERE `passnummer` = p_pass
    LIMIT 1;

  IF v_passenger_id IS NULL THEN
    SIGNAL invalid_pass SET MESSAGE_TEXT = 'Invalid pass number';
  END IF;

  SELECT `f`.`flug_id`, `f`.`flugnr` INTO v_flight_id, v_flightnr
    FROM `FlughafenDB`.`flughafen` AS `fhvon`
      INNER JOIN `FlughafenDB`.`flug` AS `f`
        ON `f`.`von` = `fhvon`.`flughafen_id`
      INNER JOIN `FlughafenDB`.`flughafen` AS `fhnach`
        ON `f`.`nach` = `fhnach`.`flughafen_id`
      INNER JOIN `FlughafenDB`.`flugzeug` AS `fz`
        ON `f`.`flugzeug_id` = `fz`.`flugzeug_id`
    WHERE `fhvon`.`iata` = p_start
      AND `fhnach`.`iata` = p_destination
      AND `f`.`abflug` > p_time
      AND `fz`.`kapazitaet` > (SELECT COUNT(*) FROM `FlughafenDB`.`flug` AS
`fi` INNER JOIN `FlughafenDB`.`buchung` AS `bi` ON `fi`.`flug_id` =
`bi`.`flug_id` WHERE `fi`.`flug_id` = `f`.`flug_id`)
    ORDER BY `f`.`ankunft` ASC
    LIMIT 1;

    IF v_flightnr IS NULL THEN
      SIGNAL no_flight_found SET MESSAGE_TEXT = 'Sorry, no flight
available';
    END IF;

    INSERT INTO `buchung` (`flug_id`, `passagier_id`, `preis`) VALUES
(v_flight_id, v_passenger_id, 0.0);

    SELECT CONCAT('A seat from ', p_start, ' to ', p_destination, ' was
booked on flight ', v_flightnr) AS `booked`;
END //
DELIMITER ;
```

c. erreichbare_flughaefen

```
DELIMITER //
CREATE PROCEDURE `erreichbare_flughaefen`(IN p_id SMALLINT(6), IN p_hops
TINYINT, IN p_recursion TINYINT)
BEGIN
  DECLARE v_airport_id SMALLINT(6);
  DECLARE v_from SMALLINT(6);
  DECLARE v_done TINYINT DEFAULT 0;
  DECLARE c_airport CURSOR FOR
    SELECT DISTINCT `von`, `nach`
      FROM `FlughafenDB`.`flugplan` AS `fp`
      WHERE `fp`.`von` = p_id AND `fp`.`von` IS NOT NULL AND `fp`.`nach` IS
NOT NULL;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_done = 1;

  IF p_recursion = 0 THEN
    SET max_sp_recursion_depth = p_hops + 1;
    DROP TABLE IF EXISTS `hops`;
    CREATE TABLE `hops` (
      `flughafen_id` SMALLINT(6) NOT NULL,
      `hops` TINYINT NOT NULL
    )ENGINE=MEMORY;
  END IF;

  IF p_recursion <= p_hops THEN
    OPEN c_airport;
    l_fetch_data: LOOP
      FETCH c_airport INTO v_from, v_airport_id;
      IF v_done THEN
        CLOSE c_airport;
        LEAVE l_fetch_data;
      END IF;
      INSERT INTO `hops` VALUES (v_airport_id, p_recursion);
      CALL erreichbare_flughaefen(v_airport_id, p_hops, (p_recursion + 1));
    END LOOP l_fetch_data;
  END IF;

END //
DELIMITER ;
```

2. Triggers uns Views

a. View

Beim Insert wird der Fehler „`CHECK OPTION failed 'mdeutschl_FlughafenDB.v_buchung'`“ geworfen. Die `WITH CHECK` Klausel wendet alle Constraints der `WHERE` Klausel und validiert diese gegen alle Statements, die auf die View abgesetzt werden und failt somit wenn versucht wird, einen ungültigen Wert einzufügen.

```
CREATE OR REPLACE VIEW `v_buchung` AS
  SELECT * FROM `buchung` WHERE `buchung_id` BETWEEN 10 AND 1000
  WITH CHECK OPTION;
```

b. Trigger

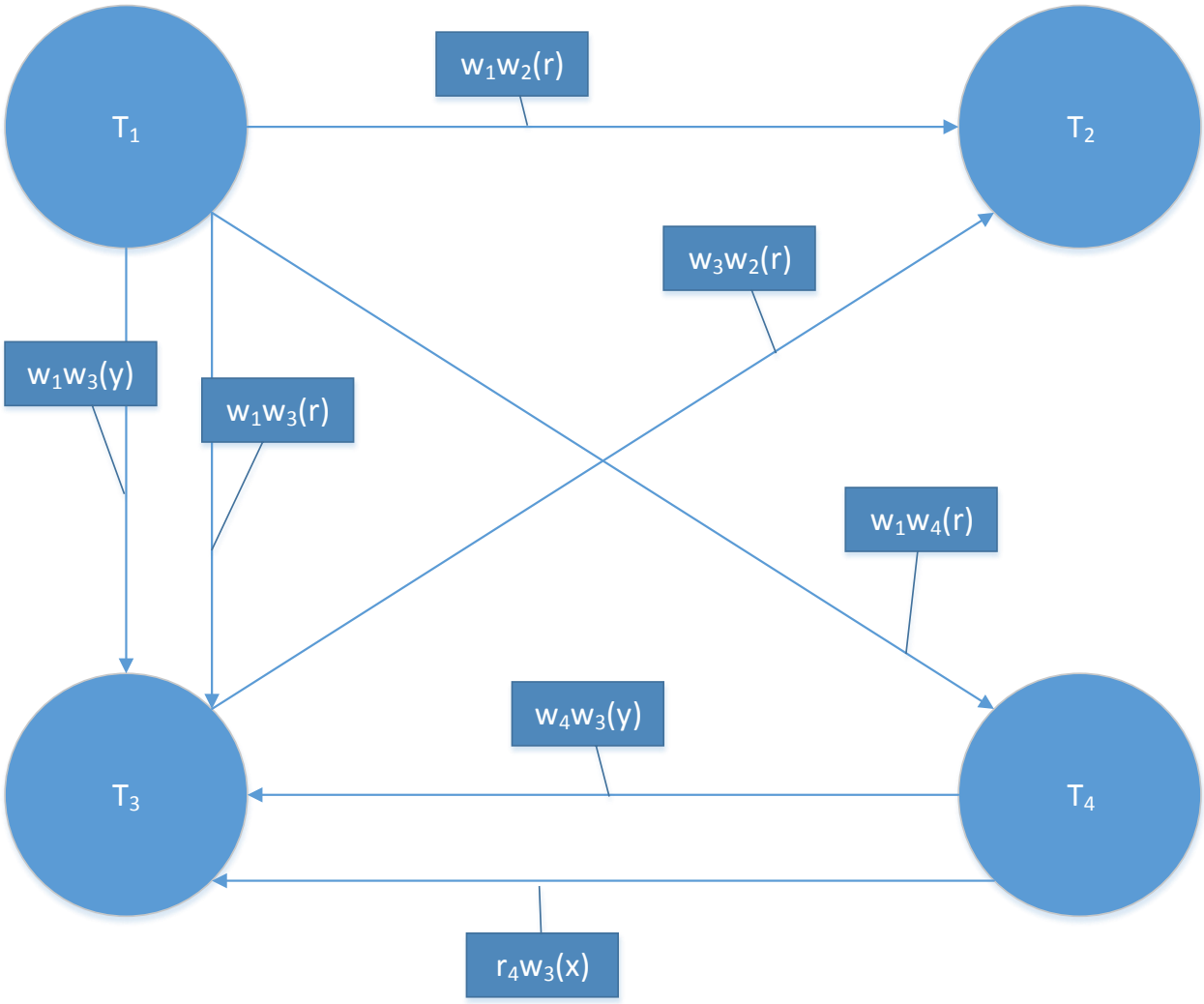
Die Spalte `preis` ist mit dem `NOT NULL` constraint versehen, weshalb sowieso keine `NULL`-Werte für den Preis eingefügt werden können.

```
DELIMITER //
CREATE TRIGGER `t_price_invalid` BEFORE INSERT ON `buchung`
FOR EACH ROW
BEGIN
  IF NEW.`preis` IS NULL OR NEW.`preis` < 0 THEN
    SET NEW.`preis` = 10000.0;
  END IF;
END //
DELIMITER ;
```

3. Transaktionen

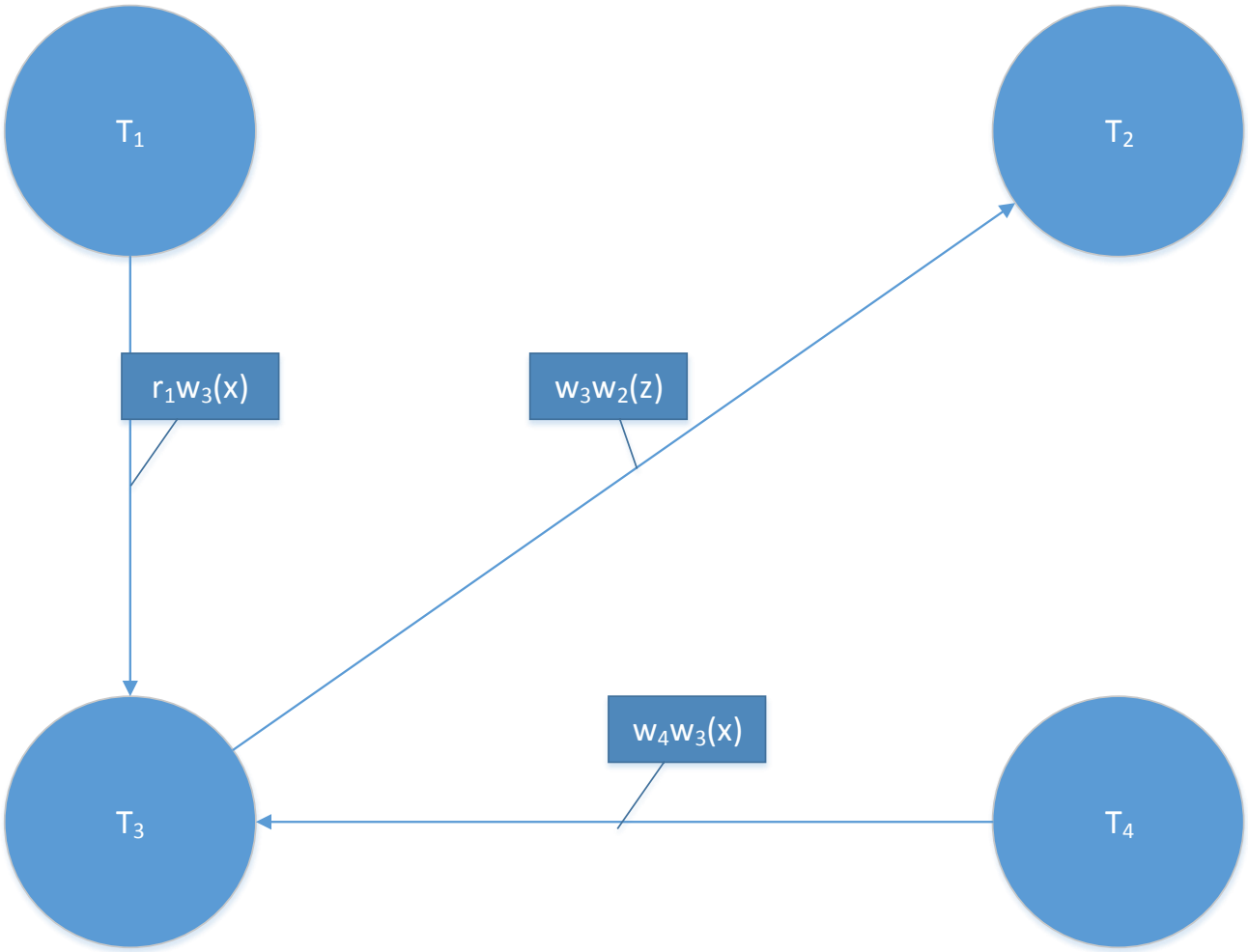
a.

Ausführungsreihenfolge: T_2, T_3, T_4, T_1



b.

Ausführungsreihenfolge: T_2, T_3, T_4, T_1



4. 2-Phasen-Sperrprotokoll

T_a	T_b	T_c	
		r(x)	x++ (1)
w(y)			SET WRITE LOCK y
r(x)			x++ (2)
	w(x)		WAIT (T_b wird unterbrochen)
		r(x)	x++ (3)
		r(z)	z++ (1)
		w(z)	SET WRITE LOCK z
	w(y)		
		w(x)	WAIT
r(y)			y++ (1)
eot			Reevaluierung von T_b
		w(x)	SET WRITE LOCK x
	w(x)		
		eot	

T_b wartet auch nach dem eot von T_a weiter, da auch T_c eine Lesesperre auf x erstellt hat, entsprechend wird T_b immer wieder unterbrochen. Das führt dazu, dass T_a problemlos fertig wird und danach T_c problemlos fertig wird. T_b kann erst nach T_a und T_c ausgeführt werden.