CS3101 Practical 2 Report

Compilation, Execution & Usage

Due to the time constraints of this practical the only way to test the database is to recreate it using the data from the dump file (copy and paste the create and insert statements into MariaDB) or running it on the host server using my credentials (the host server is Klovia). This is because the user interface was not finished due to (as previously mentioned) time constraints.

Overview

For this practical I was asked to implement a database (DB) using MariaDB with a given schema, populate and query the database, add constraints onto the DB and create a simple graphical interface to interact with the database. I have implemented all, but the final requirement listed. I believe it is implemented to quite a decent standard.

Database Implementation

As mentioned in the overview the database was implemented with the given wedding schema using MariaDB in this section each of the queries run will be shown in figures as well as a small breakdown of each.

Create Statements

There was six proper create statements. There was also the statement to create the database, however that is very simple and not worth discussing. When creating the database, the order the tables were created was somewhat important due to the dependencies between the tables. First the Invitation, Dinner Table and Dietary Requirement tables were created as seen below in Figure 1, 2 and 3 respectively:

```
CREATE TABLE `invitation`(
  `code` char(5) NOT NULL,
  `date_sent` date,
  `address` varchar(300),
  PRIMARY KEY (`code`)
);
```

Figure 1: Create Statement for Invitation Table

There is not much to comment on here, the types and primary key are just in accordance with the given schema. The only thing of note is the code field. It has the not null constraint as it is the primary key and

the fact code does not use varchar, but just char as with the given data the invitation codes were always five characters.

```
CREATE TABLE `dinner_table` (
  `table_no` int(11) NOT NULL,
  `capacity` int(11),
  PRIMARY KEY (`table_no`)
);
```

Figure 2: Create Statement for Dinner Table... Table

Again, nothing much to comment on aside from the not null constraint as it is the primary key of the table.

```
CREATE TABLE `dietary_requirement` (
  `short_name` varchar(20) NOT NULL,
  `description` varchar(200),
  PRIMARY KEY (`short_name`)
);
```

Figure 3: Create Statement for Dietary Requirement Table

Same again as the above two. Afterwards however, the person table was created to as it was also needed for dependencies but has its own dependencies so had to be created after the initial three. This is seen below in figure 4:

```
CREATE TABLE `person` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `full_name` varchar(30)  NOT NULL,
 `response` Boolean,
 `notes` varchar(200),
 `invitation_code` char(5),
`table_no` int(11),
PRIMARY KEY (`id`),
FOREIGN KEY (`invitation_code`) REFERENCES `invitation` (`code`) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (`table_no`) REFERENCES `dinner_table` (`table_no`) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Figure 4: Create Statement for Person Table

With this table again a lot of it is just according to the given schema, however the important things to note are the id field has an auto increment feature because it is a simple integer id and they aren't in the given data so this is an easy way to create the field and not have to worry about creating a unique one when populating the DB. Id also has not null as it has the primary key as well as full name because it is reasonable to assume that a person should have a name if they are being invited to a wedding. Then of course there is the foreign keys according to the schema. All foreign keys cascade upon update and delete as there is no reason to restrict. One last thing to note is the response field is created as a Boolean, but it will become a tiny integer when made due to how MariaDB handles Booleans, this is important for DB population and is discussed in that section. Finally, the other two tables were created, Guest Diet and Organiser (seen in figure 5 and 6 respectively).

```
CREATE TABLE `guest_diet` (
`person_id` int(11) NOT NULL,
`dietary_requirement_name` varchar(20) NOT NULL,
PRIMARY KEY (`person_id`,`dietary_requirement_name`),
FOREIGN KEY (`person_id`) REFERENCES `person` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (`dietary_requirement_name`) REFERENCES `dietary_requirement` (`short_name`) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Figure 5: Create Statement for Guest Diet Table

This is a quite simple table made according to the schema, not much of note design wise.

```
CREATE TABLE `organiser` (
  `person_id` int(11) NOT NULL,
  `password` varchar(20),
  PRIMARY KEY (`person_id`),
  FOREIGN KEY (`person_id`) REFERENCES `person` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Figure 6: Create Statement for Organiser Table

Similar to the above as these are quite simple tables made using the given schema.

Database Population

There is not too much to say about populating the database as we were given the data (much like the schema. The only important thing of note was the significant amount of time it took to create the insert statements due to the confusing and inaccurate to the schema formatting of the given spreadsheet. And a part of MariaDB which initially caused several issues when inserting the response into person table. As MariaDB handles Booleans by representing true as ‘1’ and false as ‘0’ (*BOOLEAN*, no date). I was not aware of this at first. However once fixed it was relatively easy. Below in Figure 7-12 are the insert statements used to populate the DB.

```
INSERT INTO `invitation`

VALUES ('AWGHZ','2023-02-25','2 Woodleigh Road, Coventry, CV4 8GT'),('CAUMN','2023-02-23','35 Northway, Leamington Spa, CV31 2BW'),('CIXGN','2023-02-23','3 Dunmere Cottages, Par, PL24 2PD'),('HTJPT','2023-03-01','42 Manor Road, Ducklington, OX29 7YA'),('IXKJB','2023-03-07','9 Heol Dolfain, Ynysforgan, SA6 6QF'),('JHAOJ','2023-02-23','86 St Bartholomews Road, Nottingham, NG3 3ED'),('JQBZV','2023-03-07','Charter House, 100 Broad Street, Birmingham, B15 1AE'),('JUQOK','2023-02-23','3 Marchmont Gardens, Richmond, TW10 6ET'),('KTANU','2023-03-01','117 Hamstead Road, Handsworth, B20 2BT'),('KYMTY','2023-02-23','8 Farriers Court, Coleford, GL16 8AB'),('LHGFM','2023-03-12','4 Victoria Road, Cemaes Bay, LL67 0HR'),('LLBQC','2023-02-23','21 Crescent Gardens, Swanley, BR8 7HE'),('LTOQV','2023-02-23','126 Jersey Road, Blaengwynfi, SA13 3TE'),('LUXNR','2023-03-01','12 Helyg Road, Penmaenmawr, LL34 6HE'),('MCZFI','2023-02-23','11 Cragie Walk, Gillingham, ME8 9JH'),('MIDZS','2023-02-25','Dormouse, Tithe Laithe, Hoyland, S74 9DQ'),('MSNNS','2023-03-07','20 Teal Close, Askam-In-Furness, LA16 7JF'),('NNNND','2023-03-01','5 Sunningdale Avenue, Brigg, DN20 8QD'),('NQWUC','2023-03-01','6 The Causeway, Needham Market, IP6 8BD'),('QGCJI','2023-02-23','Glebe House, The Street, Itteringham, NR11 7AX'),('RGWBV','2023-03-01','Orchard Cottage, Ingram Lane, Grassthorpe, NG23 6RA'),('RVQLU','2023-02-23','215 Southcote Lane, Reading, RG30 3AY'),('SADDT','2023-03-01','40 Honey Lane, Buntingford, SG9 9BQ'),('UAZEX','2023-02-23','45 Harpes Road, Oxford, OX2 7QJ'),('UBIHF','2023-02-23','1 Red Gables, Puddington Village, Puddington, CH64 5ST'),('ULYDN','2023-02-23','11 Rutherglen Road, Sunderland, SR5 5LW'),('UTZAF','2023-03-03','11 Mayfield, Ivybridge, PL21 9UE'),('VFAIW','2023-02-25','Carreg Grwca, Glandwr, SA34 0UD'),('VZLAR','2023-03-02','11 Denton View, Blaydon-On-Tyne, NE21 4DZ'),('ZTPAA','2023-02-25','6 Almsgate, Compton, GU3 1JG'),('ZYBXM','2023-03-12','79 Hayling Rise, Worthing, BN13 3AG');
```

Figure 7: Insert Statement to Populate Invitation

```
INSERT INTO `dinner_table`

VALUES (1,11),(2,8),(3,10),(4,10),(5,10),(6,4),(7,8),(8,8),(9,6),(10,8),(99,1);
```

Figure 8: Insert Statement to Populate Dinner Table

```
INSERT INTO `dietary_requirement`

VALUES ('Gluten-free','No gluten, e.g. wheat products.'),('Halal','Alcohol, some meats and a few other things to be avoided. Check https://thehalallife.co.uk/halal-check/'),('Vegan','No animal products of any kind, except maybe honey (check!).'),('Vegetarian','No meat or fish.');
```

Figure 9: Insert Statement to Populate Dietary Requirement

```
INSERT INTO `person` (full_name, response, notes, invitation_code, table_no)

VALUES ('Vicky Hernando',1,'','UAZEX',1),('Solomiya Hernando',1,'','UAZEX',1),('Davorka
Hernando',1,'','UAZEX',1),('Driskoll Devine',NULL,'','RVQLU',1),('Sophie Devine',NULL,'','RVQLU',1),('Anas
Devine',NULL,'','RVQLU',1),(Theo Hester',NULL,'','MCZFI',1),('Morgan Martin',NULL,'Best man','KYMTY',1),('Jody
Sims',1,'Giving speech','LTOQV',1),('Islay Sims',0,'Giving speech','LTOQV',1),('Ashlyn Sims',0,'','JHAOJ',2),('Ryan
Sims',1,'Thank Ryan for the voucher he sent us.','LTOQV',2),('Chung Mercado',NULL,'','ULYDN',2),('Carmen
Turner',NULL,'','QGCJI',2),('Theresa Butler',1,'','LLBQC',2),('Edmund Kaufman',1,'','UBIHF',2),('Mason
Gill',NULL,'','JUQOK',3),('Fletcher Gill',1,'Doesnt talk to Keira any more: keep them apart!','JUQOK',3),('Tameka Gill',0,'Has
guide dog called Amos, a very good boy.','CIXGN',3),('Elvin Nielsen',0,'','CAUMN',3),('Lyssa Nielsen',0,'','CAUMN',3),('Keira
Arnold',NULL,'','CAUMN',3),('Benedict Arnold',NULL,'','CAUMN',3),('Andrea Cardenas',NULL,'','MIDZS',4),('Stephen
Cardenas',NULL,'','MIDZS',4),('Chris Cardenas',NULL,'','VFAIW',4),('Margaret Garza',NULL,'','ZTPAA',4),('Joann
Alvarado',NULL,'','AWGHZ',4),('Deana Alvarado',0,'','AWGHZ',4),('Raymond Cox',0,'','SADDT',5),('Don
Cox',1,'','SADDT',5),('Jimmie Frye',1,'Might try to play accordion: avoid this at all costs.','RGWBV',5),('Lacey
Frey',1,'','HTJPT',5),('Priscilla Todd',NULL,'Wheelchair access needed','HTJPT',5),('Yesenia
Hall',NULL,'','HTJPT',5),('Angelina Hall',1,'','HTJPT',5),('Fran Spears',1,'','NQWUC',6),('Lorraine
Hahn',NULL,'','KTANU',6),('Sherri Evans',1,'','NNNND',6),('Kathryn Bates',1,'','NNNND',6),('Mitchel
Wyatt',1,'','LUXNR',7),('Aaron Wyatt',1,'','LUXNR',7),('Tara Wyatt',0,'','LUXNR',7),('Dwayne
```

Figure 10: Insert Statement to Populate Person

```
INSERT INTO `organiser`

VALUES (266,'iheartpokemon1991'),(269,'abc123');
```

Figure 11: Insert Statement to Populate Organiser

```
INSERT INTO `guest_diet`

VALUES (267,'Gluten-free'),(267,'Vegetarian'),(268,'Vegan'),(269,'Vegan'),(270,'Gluten-
free'),(270,'Vegan'),(274,'Vegetarian'),(275,'Vegetarian'),(282,'Gluten-
free'),(282,'Vegan'),(285,'Vegetarian'),(286,'Vegetarian'),(287,'Vegetarian'),(289,'Halal'),(290,'Gluten-
free'),(290,'Halal'),(293,'Vegetarian'),(294,'Vegetarian'),(303,'Vegetarian'),(304,'Vegetarian'),(305,'Ve
getarian'),(306,'Gluten-
free'),(306,'Vegetarian'),(307,'Vegetarian'),(308,'Vegan'),(309,'Vegan'),(317,'Vegetarian'),(318,'Veget
arian'):
```

Figure 12: Insert Statement to Populate Guest Diet

Querying the Database & View Creation

For this practical I was required to create 3 queries and then save them using views. The creation of the views was very simple and thus will not be shown in this report (it was simply ‘create view `view_name` as’ then the queries shown below). However, the queries will be shown in the next three figures and explained (Figures 13, 14 & 15).

```
select `full_name` AS `Full Name`,`person`.`id` AS `ID`, `date_sent` AS `Date of Invitation`
from `person`, `invitation`
where `response` is null and `person`.`invitation_code` = `invitation`.`code`
order by `invitation`.`date_sent`,`person`.`full_name`,`person`.`id`
```

Figure 13: Query to Select Data of Those Awaiting Response From

This is a quite simple query selecting the basic data where the response is null and the codes match in each table. Doing the order by in the specification. Without the invitation code condition there is several duplicate records for each person.

```
select `dinner_table`.`table_no` AS `Table Number`,count(`id`) AS `Number of Standard-Diet Guests`
from `dinner_table`, `person`
where `person`.`id` not in (select `guest_diet`.`person_id` from `guest_diet`)) and `response` = 1 and `person`.`table_no` = `dinner_table`.`table_no`
group by `dinner_table`.`table_no`
order by `dinner_table`.`table_no`;
```

Figure 14: Query to Select Data of How Many Have No Diet Requirements at Their Tables

This query makes use of aggregate functions to display the number of guests with no dietary requirements at each table (hence the group by function). It counts the ids for each table, counting only the id's not in the guest diet table as well as guests who responded as going (I.e. the response is 1 because true is represented as 1 in MariaDB. It also has the same condition to prevent repeat records and incorrect counts.

```
select `dinner_table`.`table_no` AS `Table Number`,`person`.`full_name` AS `Full Name`,

group_concat(`dietary_requirement`.`short_name` order by
`dietary_requirement`.`short_name` ASC separator ', ') AS `Dietary Requirement(s)`

from (((`person` join `guest_diet` on(`person`.`id` = `guest_diet`.`person_id`))

join `dietary_requirement` on(`guest_diet`.`dietary_requirement_name` =
`dietary_requirement`.`short_name`))

left join `dinner_table` on(`person`.`table_no` = `dinner_table`.`table_no`))

where `guest_diet`.`dietary_requirement_name` is not null

group by `person`.`id`

order by `dinner_table`.`table_no`,`person`.`full_name`;
```

Figure 15: Query to List the Table Number and Dietary Requirements of All Guests That Apply

This query is a lot more complicated, using the group_concat function (MariaDB Tutorial, 2020) which concatenates all the values in a single column with the comma separator. Then to show my knowledge I used joins to combine the tables and where the guest actually has a dietary requirement. This does something fairly simple on paper but as you can see is a complicated query.

Constraints

For most of the constraints, the tables were simply altered, or a trigger was added as the specification stated we could use any method to enforce the constraints (apart from the final one as it had to be a procedure. Like with the above sections, the code for these are shown below with a brief explanation for each in figures 16-19:

```
ALTER TABLE `dinner_table`

ADD CONSTRAINT CHECK (`capacity` > 0)
```

Figure 16: Alter Table to Check Capacity

This is a very simple statement adding a check to the dinner table, ensuring the capacity is always greater than zero (i.e., a positive integer). Since this constraint was so simple it was way easier to alter the table to add a check to it.

```
DELIMITER ;;

CREATE trigger check_table_capacity  before insert on person for each row

begin

        declare current_cap int;

        declare i int;

        select capacity into current_cap

                from dinner_table

                where table_no = new.table_no;

        select COUNT(*) into i

                from person

                where table_no = new.table_no;

        if i + 1 > current_cap then

                signal SQLSTATE '45001'

                set MESSAGE_TEXT = 'Cannot add to table as table is at capacity';

        end if;

end ;;

DELIMITER ;
```

Figure 17: Trigger That Checks Tables Are Not Going Over Capacity

This is a little more complicated than before, as it is a trigger that checks each insert on person for each row, checking that when adding a new person to a table it will not go over the table's capacity. First gets the capacity of the table that the person is being added to, then counts the number of people already at that table, if adding another person to that table would exceed the capacity then an error occurs, otherwise the check goes through, and the insertion is done. There is also a delimiter change in order to make the check work with the several declare and select statements.

```
DELIMITER ;;
CREATE trigger check_date before insert on invitation for each row
begin
        if NEW.date_sent > NOW() then
                signal SQLSTATE '45001'
                set MESSAGE_TEXT = 'Date sent cannot be set in the future';
        end if;
end ;;
DELIMITER ;
```

Figure 18: Trigger to Check the Invitation Date is not in the Future

Initially I wanted to make this a simple alter table statement like in the first constraint essentially taking the if statements condition and putting it into a check. However, this resulted in an error, no matter which version of now I tried it did not work. I believe this is because of how checks differ from triggers. I.e., the now function could not be used as much as a check would cause it. However, since a trigger only happens when an insertion happens it can be used. So, this is a quite simple trigger, does a single if check and if the date sent is not valid then it sends an error message.

```
DELIMITER ;;

CREATE PROCEDURE `proc_add_person`(IN new_full_name VARCHAR(255), IN new_notes VARCHAR(255), IN
new_invitation_code VARCHAR(255), IN new_table_no INT)

BEGIN

        DECLARE v_invitation_code VARCHAR(255); DECLARE v_table_no INT;

        SELECT code INTO v_invitation_code FROM invitation WHERE code = new_invitation_code;

        IF v_invitation_code IS NULL THEN

                INSERT INTO invitation (code, address, date_sent) VALUES (new_invitation_code, NULL, NULL);

        END IF;

        SELECT table_no INTO v_table_no FROM dinner_table WHERE table_no = new_table_no;

        IF v_table_no IS NULL THEN

                INSERT INTO dinner_table (table_no, capacity) VALUES (new_table_no, '1');

        END IF;

        INSERT INTO person (full_name, response, notes, invitation_code, table_no)

        VALUES (new_full_name, NULL, new_notes, new_invitation_code, new_table_no);

END ;;

DELIMITER ;
```

Figure 19: Procedure to Validate Entry into Person Table

This is fairly complicated looking procedure but ultimately is very simple. It checks if the invitation the person has exists, if not then insert a new one in with null values, then checks if the table number exists, if not then insert one with the correct values, then inserts the new person in. This is just to ensure referential constraints are being met.

GUI Implementation

This was not completed due to time constraints however this does mean that no effort was put into it. As I at least chose what framework that would be used which will be discussed. React was chosen to create the GUI, even though the GUI was not completed. This is because it is quite simple to develop on, very well used and does well with database interaction (Gundaniya, no date). This is why it is what I would have used if given the time to do the GUI.

Conclusion

Ultimately, I am quite happy with what has been achieved, while the GUI was not completed the work on the Database is quite satisfactory, completing all requirements in terms of creating, populating, querying, and constraining it.

References

1. *BOOLEAN* (no date). Available at: https://mariadb.com/kb/en/boolean/.
2. MariaDB Tutorial (2020) *MariaDB group_concat() Aggregate Function By Practical Examples*. Available at: https://www.mariadbtutorial.com/mariadb-aggregate-functions/mariadb-group_concat/.
3. Gundaniya, N. (no date) *The benefits of ReactJS and reasons to choose it for your project*. Available at: https://www.peerbits.com/blog/reasons-to-choose-reactjs-for-your-web-development-project.html.