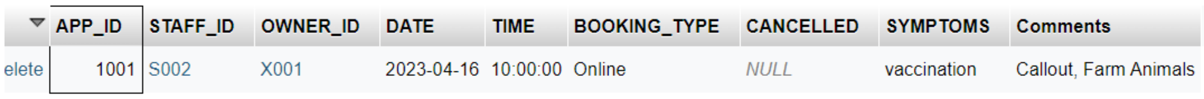
Rule 1: The Information Rule

"All information in a relational database is represented explicitly at the logical level in exactly one way – by values in tables".

All information in the database in represented in tables by columns and tuples. Each tuple has a primary key and these columns are set to be not null and unique. A unique value can be accessed with a unique primary key.

```
SELECT * FROM appointment WHERE APP_ID = '1001';
```



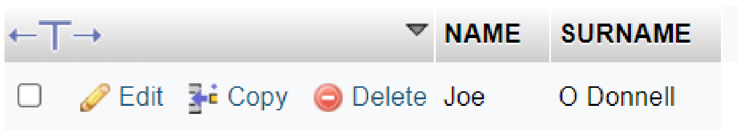
	APP_ID	STAFF_ID	OWNER_ID	DATE	TIME	BOOKING_TYPE	CANCELLED	SYMPTOMS	Comments
elete	1001	S002	X001	2023-04-16	10:00:00	Online	NULL	vaccination	Callout, Farm Animals

Rule 2: The Guaranteed Access Rule

"Each and every datum (atomic value) in a relational database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name."

Only three items are needed to locate specific data: the column(s) name, table name and primary key. All tables have been created so that primary key column is set to not null and unique to guarantee access.

```
SELECT NAME, SURNAME FROM staff WHERE STAFF_ID = 'S001';
```



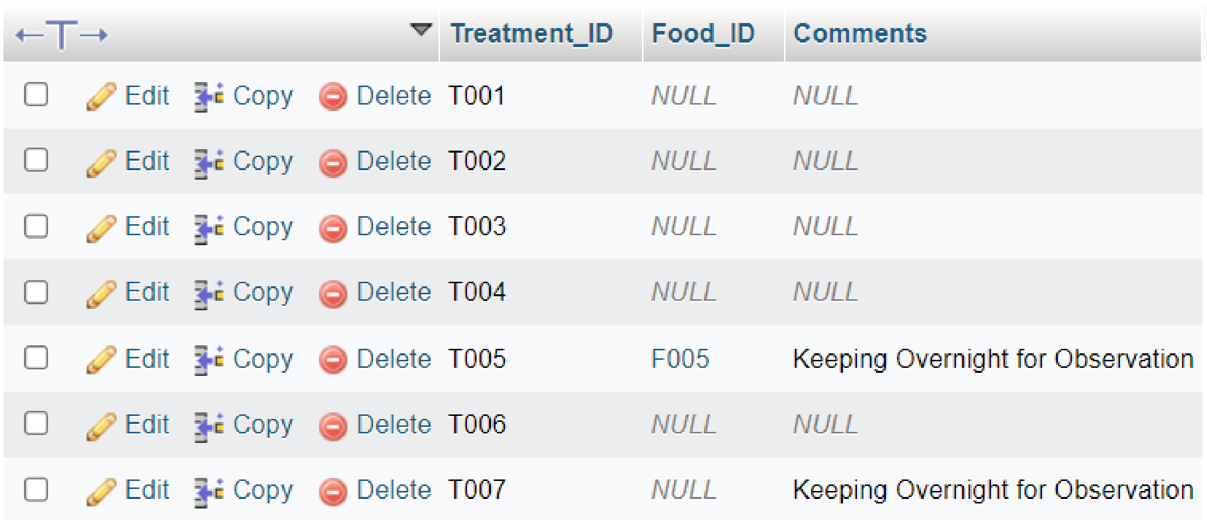
	NAME	SURNAME
Edit Copy Delete	Joe	O Donnell

Rule 3: Systematic Treatment of Null Values

"Null values (distinct from the empty character string or a string of blank characters or any other number) are supported in the fully relational DBMS for representing missing information in a systematic way, independent of data type."

NULL has a special meaning in a database, it can represent unknown, missing, or inapplicable data. In the treatment table, NULL has been used to represent inapplicable data as not all tuples have data relating to every column.

```
SELECT Treatment_ID, Food_ID, Comments FROM treatment;
```



Treatment_ID	Food_ID	Comments
T001	NULL	NULL
T002	NULL	NULL
T003	NULL	NULL
T004	NULL	NULL
T005	F005	Keeping Overnight for Observation
T006	NULL	NULL
T007	NULL	Keeping Overnight for Observation

Statements are possible where column values may be NULL:

```
SELECT Invoice_ID, Treat_Fee + IFNULL(LateCancel_Fee, 0) FROM billing;
```

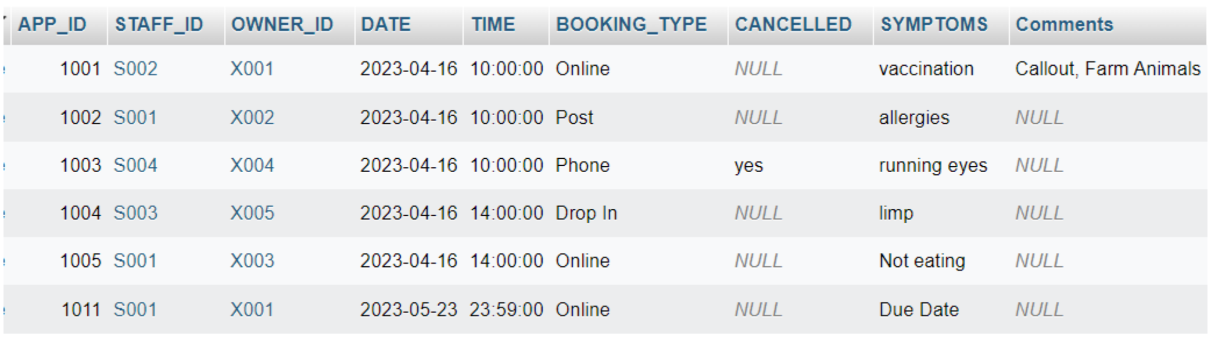
Tables in the database have been created so that important columns such as primary keys are not allowed to be NULL, other columns have set so that if no data is entered the default will be NULL. For example:

```
CREATE TABLE staff (
  STAFF_ID varchar(50) NOT NULL UNIQUE,
  NAME varchar(50) DEFAULT NULL,
  SURNAME varchar(50) DEFAULT NULL,
  ROLE varchar(50) DEFAULT NULL,
  PRIMARY KEY (STAFF_ID)
  );
```

Rule 4: Dynamic online catalogue based on the relational model.

"The data base description is represented at the logical level in the same way as ordinary data, so that authorised users can apply the same relational language to its interrogation as they apply to regular data."

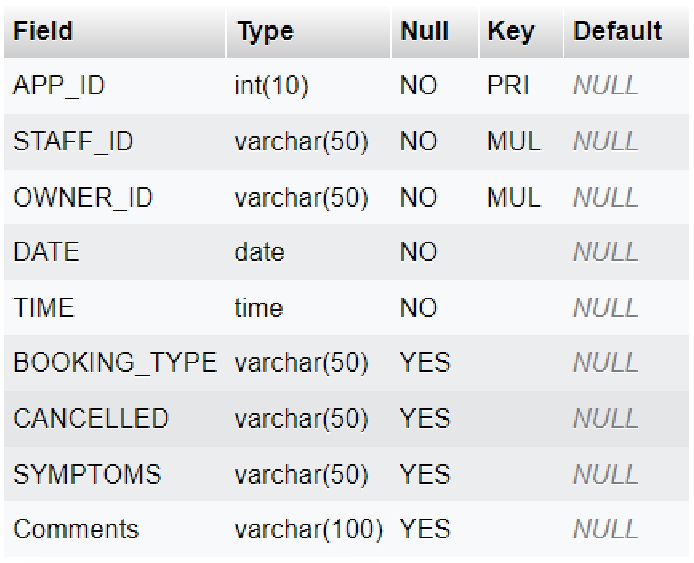
The database should be self-describing. Users cannot modify the tables directly, Data Manipulation Language (DML) must be used to modify the structure. The structure is stored in an up-to-date data dictionary.



APP_ID	STAFF_ID	OWNER_ID	DATE	TIME	BOOKING_TYPE	CANCELLED	SYMPTOMS	Comments
1001	S002	X001	2023-04-16	10:00:00	Online	NULL	vaccination	Callout, Farm Animals
1002	S001	X002	2023-04-16	10:00:00	Post	NULL	allergies	NULL
1003	S004	X004	2023-04-16	10:00:00	Phone	yes	running eyes	NULL
1004	S003	X005	2023-04-16	14:00:00	Drop In	NULL	limp	NULL
1005	S001	X003	2023-04-16	14:00:00	Online	NULL	Not eating	NULL
1011	S001	X001	2023-05-23	23:59:00	Online	NULL	Due Date	NULL

Structure of each table is easily accessible with a statement as follows:

```
DESC appointment;
```



Field	Type	Null	Key	Default
APP_ID	int(10)	NO	PRI	NULL
STAFF_ID	varchar(50)	NO	MUL	NULL
OWNER_ID	varchar(50)	NO	MUL	NULL
DATE	date	NO		NULL
TIME	time	NO		NULL
BOOKING_TYPE	varchar(50)	YES		NULL
CANCELLED	varchar(50)	YES		NULL
SYMPTOMS	varchar(50)	YES		NULL
Comments	varchar(100)	YES		NULL

Rule 5: The comprehensive data sub language rule.

"A relational system may support several languages and various modes of terminal use (for example, fill-in-theblanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all of the following items:

- *Data definition (DDL)*
- *View definition. (DML)*
- *Data manipulation*
- *Integrity constraints*
- *Transaction boundaries"*

My database supports DDL and DML through SQL as a language to create and modify data structures and query them. A new table can be created with the following example (all tables in the database can be altered in the same fashion):

```
CREATE TABLE example (
  id varchar(10) not null unique,
  name varchar(10) default null,
  surname varchar(10) default null,
  module varchar(10) default null,
  primary key (id)
  );
```

Table can be modified, here is an example of adding extra columns:

```
ALTER TABLE example ADD location varchar(10) default null;
ALTER TABLE example ADD extra varchar(10) default null;
```

Also, possible to delete columns:

```
ALTER TABLE example DROP COLUMN extra;
```

Insert data into the table:

```
INSERT INTO example (id, name, surname, module, location) VALUES ('G00246442', 'Ri
chard', 'Daly', 'Databases', 'Galway');
```

Select specific data from the table:

```
SELECT * FROM example WHERE id = 'G00246442';
```

Create a View and be able to view the table from that View:

```
CREATE VIEW studentView AS SELECT id, name, surname FROM example WHERE id = 'G0024
6442';

SELECT * FROM studentview;
```

Delete the View, the data and the table entirely:

```
DROP VIEW studentview;

DELETE FROM example WHERE id = 'G00246442';

DROP TABLE example;
```

Rule 6: The view updating rule.

"All views that are theoretically updateable are also updateable by the system."

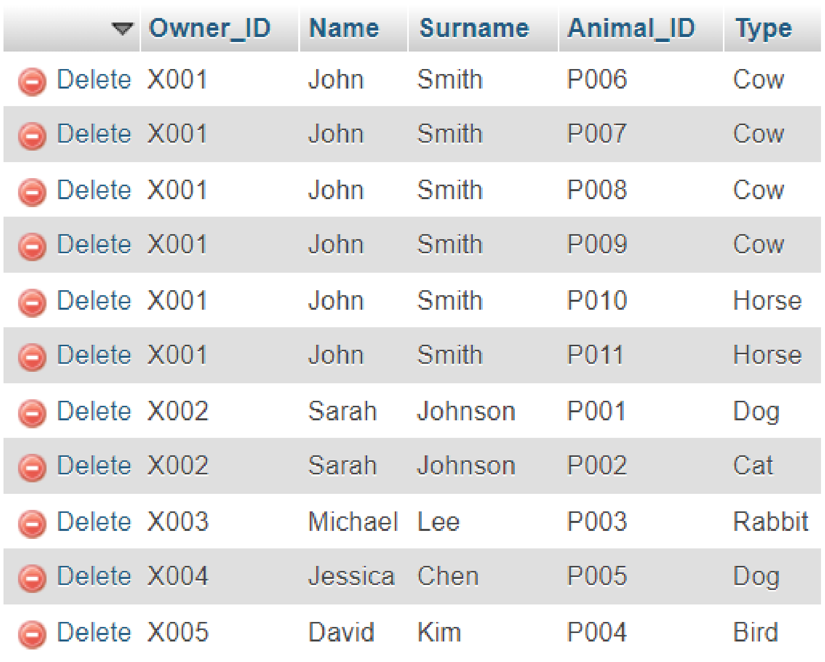
By creating a view, a table can be made up of one or many base tables with specified columns to provide a unique view of a dataset. The database system should allow any alteration made in the view to be related to the base tables.

A View can be created in the following example (Already created in DB):

```
CREATE VIEW petAndOwner AS SELECT petowner.Owner_ID, petowner.Name,
petowner.Surname, animal.Animal_ID, animal.Type FROM petowner, animal WHERE
petowner.Owner_ID = animal.Owner_ID;
```

The view can be selected as follows:

```
SELECT * FROM petandowner;
```



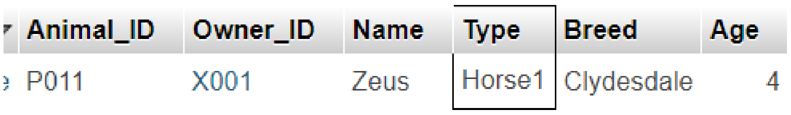
	Owner_ID	Name	Surname	Animal_ID	Type
Delete	X001	John	Smith	P006	Cow
Delete	X001	John	Smith	P007	Cow
Delete	X001	John	Smith	P008	Cow
Delete	X001	John	Smith	P009	Cow
Delete	X001	John	Smith	P010	Horse
Delete	X001	John	Smith	P011	Horse
Delete	X002	Sarah	Johnson	P001	Dog
Delete	X002	Sarah	Johnson	P002	Cat
Delete	X003	Michael	Lee	P003	Rabbit
Delete	X004	Jessica	Chen	P005	Dog
Delete	X005	David	Kim	P004	Bird

Altering the data from the view should also be reflected in the base table. Here the Type column in animal is being altered through the view:

```
UPDATE petandowner SET Type = 'Horse1' WHERE petandowner.Owner_ID = ‘X001' AND pet
andowner.Animal_ID = 'P011';
```

Selecting the information through the base table animal will show that the alteration has affected the base table:

```
SELECT * FROM animal WHERE Type = 'Horse1';
```



Animal_ID	Owner_ID	Name	Type	Breed	Age
P011	X001	Zeus	Horse1	Clydesdale	4

Rule 7: High Level Insert Update and Delete Rule.

“The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data.”

With knowledge of a tables columns and constraints it is possible to do multiple inserts into a table with a single statement:

```
INSERT INTO petowner (Owner_ID, Name, Surname, Address_line1, Address_line2,
Phone, Email) VALUES

('X006', 'Richard', 'Sierra', 'Cashel', 'Athenry', '12348752',
'r.sierra@email.com'),

('X007', 'Tim', 'Bates', 'The sound', 'tuam', '13448956', 'timbates@email.com'),

('X008', 'sorcha', 'freenom', 'keel', 'headford', '95612345',
'freenom.s@email.com'),

('X009', 'bill', 'robson', 'briarhill', 'galway', '53545687', 'b.rob@email.com'),

('X010', 'shuna', 'o reilly', 'salthill', 'galway', '12312394',
's.o.reilly@email.com');
```

Using the primary key, it is also possible to update multiple columns in one statement:

```
UPDATE petowner SET NAME = 'Richard' WHERE Owner_ID BETWEEN 'X006' AND 'X010';
```

Using the primary key, it is also possible to delete multiple columns in one statement:

```
DELETE FROM petowner WHERE Owner_ID BETWEEN 'X006' AND 'X010';
```

Rule 8: Physical Data Independence

"Applications programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods."

We can change the physical level of a database, where it stored. It could be transferred to a new location on the storage device, changed between devices. It is platform independent and should operate the same on windows as it would on a Linux machine. Changes at the physical should not affect the logical or conceptual levels.

Rule 9: Logical Data Independence

"Applications programs and terminal activities remain logically unimpaired when Information-Preserving changes of any kind that theoretically permit impairment are made to the base tables."

Database tables are logically independent of each other, adding a new table will not change queries working against already established tables. Similarly, if a table was altered with a new column, a query that worked previously against that table should also operate the same.

The following statement will operate the same at the beginning and the end even if a new table was to be created:

```
SELECT NAME, SURNAME FROM staff WHERE STAFF_ID = 'S001';

CREATE TABLE example (

  id varchar(10) not null unique,

  name varchar(10) default null,

  surname varchar(10) default null,

  module varchar(10) default null,

  primary key (id)

  );

INSERT INTO example (id, name, surname, module, location) VALUES ('G00246442', 'Ri
chard', 'Daly', 'Databases', 'Galway');

SELECT NAME, SURNAME FROM staff WHERE STAFF_ID = 'S001';
```

The following will query will work against the table at the beginning and the end if even a new column is added:

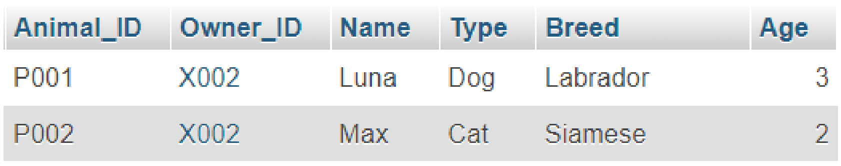
```
SELECT * FROM example WHERE id = 'G00246442';

ALTER TABLE example ADD location varchar(10) default null;

SELECT * FROM example WHERE id = 'G00246442';
```

Rule 10: Integrity Independence

“The declaration of integrity constraints must be part of the language used to define the DB structure, not enforced by application programs. This makes integrity checking a function of the DB, independent of applications.”

Possible errors are prevented by utilisation of primary and foreign keys throughout the database. Integrity is ensured through constraints of a key relating to another table, if data with a foreign key relating to it was to be deleted, the database would prevent this through an error so that integrity is ensured. This is also ensured by the foreign key being a primary key in another table as well as all keys being not null, they must have a value.

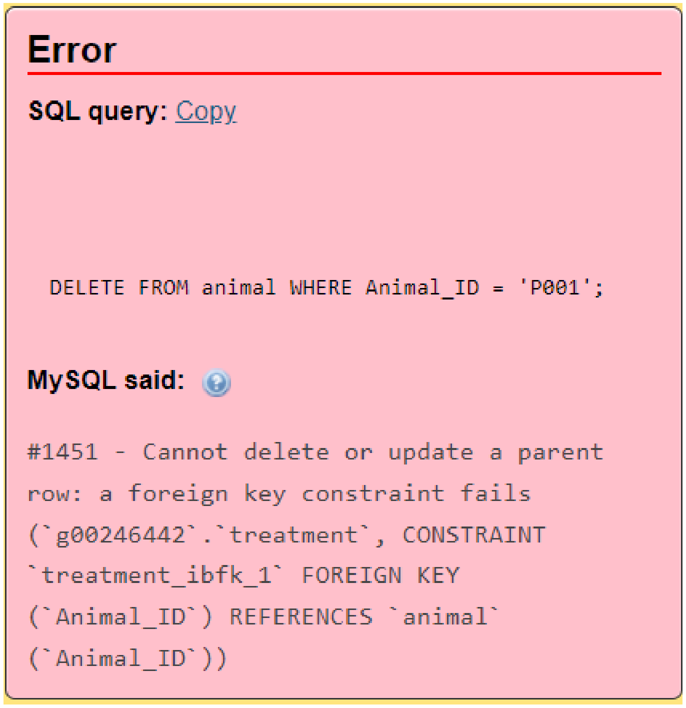
The followed is capture of data from within the animal table of the database, Animal ID is a primary key and Owner ID is a foreign key. Animal ID is referenced in the treatment table as a foreign key.



Animal_ID	Owner_ID	Name	Type	Breed	Age
P001	X002	Luna	Dog	Labrador	3
P002	X002	Max	Cat	Siamese	2

If we attempt to delete row with the primary key of P001 an error will be given due to integrity constraints as it is referenced in the treatment table.

```
DELETE FROM animal WHERE Animal_ID = 'P001';
```



Error

SQL query: Copy

```
DELETE FROM animal WHERE Animal_ID = 'P001';
```

MySQL said:

```
#1451 - Cannot delete or update a parent
row: a foreign key constraint fails
(`g00246442`.`treatment`, CONSTRAINT
`treatment_ibfk_1` FOREIGN KEY
(`Animal_ID`) REFERENCES `animal`
(`Animal_ID`))
```

Rule 11: Distributed Independence

"A relational DBMS has distribution independence."

It is possible that the data of the database could be distributed over various locations. The end user should not be aware of this. If someone was to use the Amazon Store, they are not aware of where their distributed database is stored once it works as intended. Theoretically this database could be stored in various locations and an application could still access it.

Rule 12: Distributed Independence

"If a relational system has a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity or constraints expressed in the higher-level relational language (multiple-records-at-a-time)."

SQL should be the only language that is used to change data in the database. Low level languages should not be able to alter the tables and break the integrity of the database. SQL should be used as it will support the Independence rules. Database administrators should ensure this practice for security reasons.