Project Report

I started populating some real world data into my data models tables by making the below statements/orders and then set about applying them to my database. Care was taken to make sure that the parts they ordered were matched against the cars they have. i.e. cars table and parts_cars table.

Statements;

Barry has two orders of id (1 & 2) for headlights, front bumper and back bumper for his VW Golf. The first order (1) has the two parts front and back bumper. The second order having the Headlights.

James has two orders of id (3 & 5) for two brake pads and an exhaust for his Ford Focus. Order three has two brake pads. Order 5 has the exhaust.

Orla has one order 4 for a bonnet and her car is a Nissan Leaf.

I found that after building the tables, attributes and constraints for my data model on paper that populating the data was relatively easy. I also found myself gaining a better understanding of the relations between the tables. When populating each row from the statements above into a table I would move to the other tables, populating them, in the steps of their relations. This is when the business rules become clearer. The same applies when reading the data from my data model to confirm everything was setup correctly. Ie. The parts matched in each table and the cost of each.

SQL for creating Tables, Attributes and Pk/FK

Customer

```
CREATE TABLE `g00012318`.`Customer` ( `Cust_ID` INT NOT NULL AUTO_INCREMENT , `Sta
tus` ENUM('1','2','3') NOT NULL DEFAULT '1' , `Fname` VARCHAR(255) NOT NULL , `Lna
me` VARCHAR(255) NOT NULL , `Address` VARCHAR(255) NOT NULL , `Town` VARCHAR(255)
NOT NULL , `Post_Code` VARCHAR(255) NULL , PRIMARY KEY (`Cust_ID`)) ENGINE = InnoD
B
```

Status

```
CREATE TABLE `g00012318`.`Customer_Status` ( `Status` ENUM('1','2','3') NULL , `De
sc` ENUM('Active','Inactive','Suspended') NOT NULL , PRIMARY KEY (`Status`)) ENGIN
E = InnoDB;
```

Customer to status foreign key link

```
ALTER TABLE customer add constraint fk_cust_status foreign key (status) references
 customer_status(status);
```

Car_Manufacturer

```
CREATE TABLE `g00012318`.`Car_Manufacturer` ( `Man_ID` INT NOT NULL AUTO_INCREMENT
 , `Name` VARCHAR(255) NOT NULL , PRIMARY KEY (`Man_ID`)) ENGINE = InnoDB;
```

cars

```
CREATE TABLE `g00012318`.`cars` ( `car_id` INT NOT NULL AUTO_INCREMENT , `man_id`
INT NOT NULL , `model` VARCHAR(255) NOT NULL , PRIMARY KEY (`car_id`)) ENGINE = In
noD
```

cars to car_manufacturer foreign key link

```
ALTER TABLE cars add constraint fk_cars_Man foreign key (man_id) references car_ma
nufacturer(man_id);
```

orders

```
CREATE TABLE `g00012318`.`orders` ( `order_id` INT NOT NULL , `cust_id` INT NOT NU
LL , `amount_due` INT NOT NULL , PRIMARY KEY (`order_id`)) ENGINE = InnoDB;
```

orders to customer foreign key link

```
ALTER TABLE orders add constraint fk_orders_cust foreign key (cust_id) references
customer(cust_id)
```

part_suppliers

```
CREATE TABLE `g00012318`.`part_suppliers` ( `part_supplier_id` INT NOT NULL , `par
t_id` INT NOT NULL , `supplier_id` INT NOT NULL , PRIMARY KEY (`part_supplier_id`)
) ENGINE = InnoDB;
```

parts

```
CREATE TABLE `g00012318`.`parts` ( `part_id` INT NOT NULL , `supplier_id` INT NOT
NULL , `part_name` VARCHAR(255) NOT NULL , `cost_to_us` INT NOT NULL , `cost_to_cu
st` INT NOT NULL , PRIMARY KEY (`part_id`)) ENGINE = InnoDB;
```

parts_n_orders

```
CREATE TABLE `g00012318`.`parts_n_orders` ( `parts_n_order_id` INT NOT NULL , `ord
er_id` INT NOT NULL , `part_supplier_id` INT NOT NULL , `sales_price` INT NOT NULL
 , `qty` INT NOT NULL , PRIMARY KEY (`parts_n_order_id`)) ENGINE = InnoDB;
```

part_suppliers to parts foreign key link

```
ALTER TABLE part_suppliers add constraint fk_partSup_parts foreign key (part_id) r
eferences parts(part_id)
```

part_n_orders to orders foreign key link

```
ALTER TABLE parts_n_orders add constraint fk_partsOrder_orders foreign key (order_
id) references orders(order_id)
```

part_n_orders to part_suppliers foreign key link

```
ALTER TABLE parts_n_orders add constraint fk_partsOrder_partSup foreign key (part_
supplier_id) references part_suppliers(part_supplier_id)
SELECT * FROM `part_suppliers`
```

suppliers

```
CREATE TABLE `g00012318`.`suppliers` ( `supplier_id` INT NOT NULL , `sup_name` VAR
CHAR(255) NOT NULL , `address` VARCHAR(255) NOT NULL , `town` VARCHAR(255) NOT NUL
L , `county` VARCHAR(255) NOT NULL , `post_code` VARCHAR(255) NOT NULL , PRIMARY K
EY (`supplier_id`)) ENGINE = InnoDB;
```

Parts to suppliers foreign key link

```
ALTER TABLE parts add constraint FK_parts_Sup foreign key (supplier_id) references
 suppliers(supplier_id)
```

part_suppliers to suppliers foreign key link

```
ALTER TABLE part_suppliers add constraint fk_partSup_suppliers foreign key (suppli
er_id) references suppliers(supplier_id)
```

parts_cars

```
CREATE TABLE `g00012318`.`parts_cars` ( `part_id` INT NOT NULL , `car_id` INT NOT
NULL , PRIMARY KEY (`part_id`, `car_id`)) ENGINE = InnoDB;
```

parts_cars to cars foreign key link

```
ALTER TABLE parts_cars add constraint fk_partCars_cars foreign key (car_id) refere
nces cars(car_id)
```

parts_cars to parts foreign key link

```
ALTER TABLE parts_cars add constraint fk_partCars_parts foreign key (part_id) refe
rences parts(part_id)
```

SQL for creating test data

Customer_status

```
INSERT INTO customer_status VALUES (1,'Active')
INSERT INTO customer_status VALUES (2,'inactive')
INSERT INTO customer_status VALUES (3,'suspended')
```

Customer

```
INSERT INTO customer VALUES (1,1,'James','Taylor','101, Main st','Athenry','ABC
123');
INSERT INTO customer VALUES (2,1,'Barry','Murphy','5, Shop st','Galway','EFG
456');
INSERT INTO customer VALUES (3,1,'John','kelly','12 Dame st','Loughrea','7HG
JA1');
INSERT INTO customer VALUES (4,3,'Ruth','Holland','6 High st','Tuam','9QJ YMC');
INSERT INTO customer VALUES (5,2,'Orla','Johnson','399, Ocean
Drive','Clifden','AJB 24M');
```

Car_manufacturer

```
INSERT INTO car_manufacturer VALUES (1,'Ford');
INSERT INTO car_manufacturer VALUES (2,'VW');
INSERT INTO car_manufacturer VALUES (3,'Kia');
INSERT INTO car_manufacturer VALUES (4,'Nissan');
INSERT INTO car_manufacturer VALUES (5,'Toyota');
```

Cars

```
INSERT INTO cars VALUES (1,2,'Golf');
INSERT INTO cars VALUES (2,2,'Polo');
INSERT INTO cars VALUES (3,4,'Leaf');
INSERT INTO cars VALUES (4,1,'Focus');
INSERT INTO cars VALUES (5,1,'Fiesta');
```

Orders

```
INSERT INTO orders VALUES (1,2,1000);
INSERT INTO orders VALUES (2,2,800);
INSERT INTO orders VALUES (3,1,500);
INSERT INTO orders VALUES (4,5,600);
INSERT INTO orders VALUES (5,1,600);
```

Parts

```
INSERT INTO parts VALUES (100,30,'Headlights',450,800);
INSERT INTO parts VALUES (101,20,'Brake pad',100,250);
INSERT INTO parts VALUES (102,20,'Exhaust',410,600);
INSERT INTO parts VALUES (103,30,'Front Bumper',400,500);
INSERT INTO parts VALUES (104,30,'Back Bumper',400,500);
INSERT INTO parts VALUES (105,10,'Bonnet',400,600);
```

Suppliers

```
INSERT INTO suppliers VALUES (10,'VW Ireland','Block C, Liffey Valley','Dublin
22','Ireland','YMC A33');
INSERT INTO suppliers VALUES (20,'Motorpark','Headford Rd','Galway','Ireland','CNN
B14');
INSERT INTO suppliers VALUES (30,'General Supplies ltd','High
st','Cork','Ireland','FOX X57');
```

Part_suppliers

```
INSERT INTO part_suppliers VALUES (1,100,30);
INSERT INTO part_suppliers VALUES (2,101,20);
INSERT INTO part_suppliers VALUES (3,102,20);
INSERT INTO part_suppliers VALUES (4,103,30);
INSERT INTO part_suppliers VALUES (5,104,30);
INSERT INTO part_suppliers VALUES (6,105,10);
```

Parts_cars

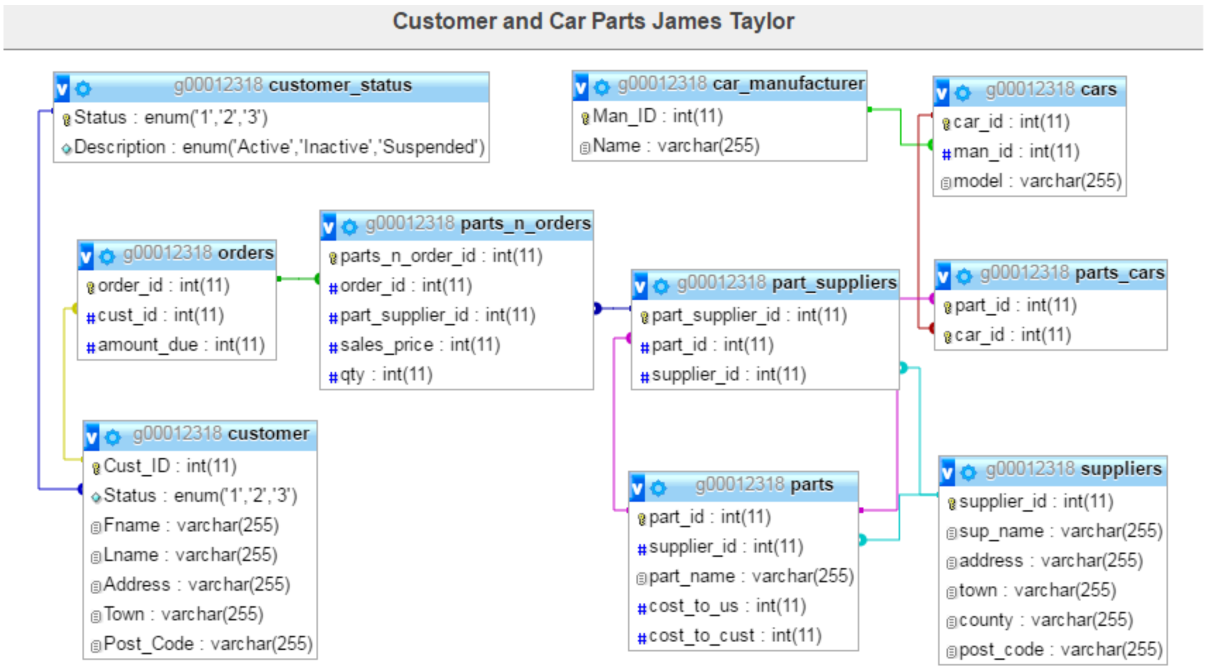
```
INSERT INTO parts_cars VALUES (103,1);
INSERT INTO parts_cars VALUES (104,1);
INSERT INTO parts_cars VALUES (100,1);
INSERT INTO parts_cars VALUES (101,4);
INSERT INTO parts_cars VALUES (102,4);
INSERT INTO parts_cars VALUES (105,3);
```

Parts_n_orders

```
INSERT INTO parts_n_orders VALUES (1,1,4,500,1);
INSERT INTO parts_n_orders VALUES (2,1,5,500,1);
INSERT INTO parts_n_orders VALUES (3,2,1,800,1);
INSERT INTO parts_n_orders VALUES (4,3,2,500,2);
INSERT INTO parts_n_orders VALUES (5,4,6,600,1);
INSERT INTO parts_n_orders VALUES (6,5,3,600,1);
```

Relational Schema map-flow

(This is also in the Customer and Car Parts schema.pdf in more detail)



Queries

Select the parts available by id and name along with its supplier name for car model Golf.

```
SELECT cars.model, parts_cars.part_id, parts.part_name, parts.supplier_id, supplie
rs.sup_name from cars, parts_cars, parts, suppliers where cars.car_id = parts_cars
.car_id and parts_cars.part_id = parts.part_id and parts.supplier_id = suppliers.s
upplier_id and cars.model = 'Golf';
```

Select all the parts in the orders made by Barry.

```
SELECT DISTINCT part_suppliers.part_id from part_suppliers , parts_n_orders, orders, customer
where part_suppliers.part_supplier_id = parts_n_orders.part_supplier_id and
parts_n_orders.order_id = orders.order_id and orders.cust_id = (SELECT cust_id FROM customer
where Fname = 'Barry');
```

Insert a new supplier called Best Buy parts from Jump street, Derry, N.Ireland and post code UPC 666.

```
insert into suppliers VALUES (40,'Best Buy Parts','Jump
St','Derry','N.Ireland','UPC 666');
```

insert 4 new users into the customers table

```
INSERT INTO customer (Cust_id,Status,Fname,Lname,Address,Town) VALUES
(99,1,'Santa','Clause','North Pole','FairyLand'),
(100,1,'Bunny','Rabbit','HillTop','Fields'),
(101,2,'Tooth','Fairy','Cloud 9','Sky'),
(102,3,'Satan','Son of Dawn','Downunder','Hell');
```

Update the address of user John to Lname Kelly, address Turloughmore, Co.Galway.

```
UPDATE customer SET Lname = 'Collins', Address='Turloughmore', Town='Co.Galway', P
ost_Code= null where cust_id = 3
```

update the 4 users that were created above

```
UPDATE customer SET Town = 'Bin' WHERE Cust_ID IN (99,100,101,102);
```

Delete the customer Ruth from the customers table.

```
DELETE FROM customer where Fname = 'Ruth'
```

Delete the 4 customers that were created above.

```
DELETE FROM customer WHERE Cust_ID IN (99,100,101,102);
```

Codds Rules

Rule 1: Information Rule

"The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format."

Explanation: Each cell of data (each specific piece of data in a table cell) is stored in a unique(PK) row (tuplet) in a certain column (attribute) in a certain table.

SQL: `SELECT Fname,cust_id from customer where Fname='James'`

The data cell 'james' is stored in the table customer under the attribute Fname and tuplet 1. Here cust_id is the primary key which ensures each unique row/tuplet.

Rule 2: Guaranteed Access Rule

"Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data."

Explanation: Here the data cell 'Golf' is accessed using table **cars**, attribute **model** and **primary key** 1

SQL: `SELECT model FROM cars WHERE car_id = 1`

Rule 3: Systematic Treatment of NULL Values

"The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following – data is missing, data is not known, or data is not applicable."

Explanation: Take the example of a NULL value present on a tuplet, and also in another tuplet located on a different table in the same database. One maybe missing an integer data type so it was set to NULL not 0. The other, a varchar data type, was not needed and so has no value. When a transaction is made against these two tables, the treatment of how the NULL value is read needs to be treated the same way independent of data type. Null != 0. Null != empty string or missing data.

SQL: Only the **first query** returns the rows of data with the null value. Showing it is treated differently form and empty string and a whitespace.

```
SELECT Post_Code FROM customer WHERE Post_Code is null
SELECT Post_Code FROM customer WHERE Post_Code = ' ' //whitespace
SELECT Post_Code FROM customer WHERE Post_Code = '' // empty string
```

One issue I observed was selecting the Post_code column where value is 0. This returned all the post_code rows with a value and not those that contained Null , '' or ' '.

```
SELECT Post_Code FROM customer WHERE Post_Code = 0
```

Rule 4: Active Online Catalog

"The structure description of the entire database must be stored in an online catalog, known as data dictionary, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself."

Explanation: The INFORMATION_SCHEMA provides information about the database like constraints, data types of columns, access privileges if any..etc. like a system catalog here you can find out everything you need about any database managed by the DBMS. Example below will list all the table constraints in my database g00012318. The second SQL shows information on the orders table.

SQL:

```
select * from information_schema.table_constraints where constraint_schema = 'g000
12318'
select * from information_schema.tables where TABLE_NAME = 'orders'
```

Rule 5: Comprehensive Data Sub-Language Rule

"A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation."

Explanation: For this DB SQL is the sub-language used to access the database and to run TSQL/DML queries. Below SQL list all the parts in the orders made by Barry

SQL: SELECT DISTINCT part_suppliers.part_id from part_suppliers , parts_n_orders, orders, customer where part_suppliers.part_supplier_id = parts_n_orders.part_supplier_id and parts_n_orders.order_id = orders.order_id and orders.cust_id = (SELECT cust_id FROM customer where Fname = 'Barry')

Rule 6: View Updating Rule

"All the views of a database, which can theoretically be updated, must also be updatable by the system."

Explanation: If a view is created of a table then to allow any updates of the view it must be able to update the corresponding base table also. And to do this it must contain all the ***nessacary*** columns/attributes. To prove this, two views were created of the customer table. One without Lname and the other without Post_code. Note only Post_code is free of the NOT NULL constraint in the base table and so is the only column not needed to update the base table.

SQL:

```
CREATE VIEW customer_view_1 as SELECT cust_id, Status, Fname, Address, Town,
Post_code FROM customer;
//An update to this view should not be allowed as Lname has to have a value in the
base table, cannot be set to NULL.

UPDATE customer_view_1 set Fname = 'Jonney' WHERE cust_id = 3
This rule was successful showing that this version if SQL does not obey the rule.
The Lname column in the base table was unchanged but Fname was.
```

```
CREATE VIEW customer_view_2 as SELECT cust_id, Status, Fname, Lname, Address, Town
FROM customer;
//This view can be updated as only post_code is missing and this can be set to
NULL in the base table

UPDATE customer_view_2 set Fname = 'John' WHERE cust_id = 3
This rule was successful, post_code was unchanged in base table.

CREATE VIEW customer_view_3 as SELECT Status, Fname, Address, Town FROM customer;
//The PK Cust_ID is missing from the view and so the base table cannot be updated.

UPDATE customer_view_3 set Fname = 'Jonney' WHERE Fname = ‘John’
This rule was successful, the base table was also updated. PK was auto-incremented
to a unique value in the base table.
```

The same applies for when data is inserted into a view,

```
INSERT INTO customer_view_1 VALUES (10,1,'Paddy','Galway','Tuam','444 aaa')
Again the base table customer was updated with the data and Lname was emnpty!!
```

Findings: This version of my SQL does not fully implement rule 6!!

Rule 7: High-Level Insert, Update, and Delete Rule

“A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.”

Explaination: SQL allows us to change multiple rows. One way is to run the update/insert/delete queries is in a batch fiile. In Toad for Oracle, Here you can list the queries and at the first line have “begin tran” (This may vary for other commercial SQL’s like Microsoft “begin”) and at the last line have “commit”. A semicolon has to be at the end of each query. Example below of inserting , updating and deleting two rows of data in the customer table of g00012318 DB.

SQL:

```
INSERT INTO customer (Cust_id,Status,Fname,Lname,Address,Town) VALUES
(99,1,'Santa','Clause','North Pole','FairyLand'),
(100,1,'Bunny','Rabbit','HillTop','Fields'),
(101,2,'Tooth','Fairy','Cloud 9','Sky'),
(102,3,'Satan','Son of Dawn','Downunder','Hell');

UPDATE customer SET Town = 'Bin' WHERE Cust_ID IN (99,100,101,102);
DELETE FROM customer WHERE Cust_ID IN (99,100,101,102);
```

Rule 8: Physical Data Independence

“The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.”

Explanation: The DB can be exported to another drive/folder on the PC or to another PC. When accessed in its new location there is no change to its structure or how it is accessed using mySQL. This is proved when the DB is exported and loaded onto another machine. My DB was exported and installed on another students (Alex) laptop with all the data incl.

SQL: When the table was exported to another students pc it was able to run the same select/update/delete sql queries.

Also the DB can be moved locally to any location as long as the reference is updated in the my.ini file. This database is stored under C:\xampp\mysql\data\ in the folder g0012318. If this was changed then the below line in C:\xampp\mysql\bin\my.ini needs to be updated to the same.

`datadir = "C:/xampp/mysql/data"` change this with " datadir = New Directory Path ". Then restart wamp serever.

Rule 9: Logical Data Independence

“The logical data in a database must be independent of its user’s view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rules to apply.”

Explanation: When changes are made to the database with its constraints or attributes the DBMS checks these first to make sure all the data meets the constraints and fail if not. Below we try to add a constraint, FK, to a non unique or PK column. This should fail as FK have to link to PK/unique column/s. Same would apply if two tables were joined as one. Or seperated from one to two tables.

SQL: ALTER TABLE `car_manufacturer add constraint FK_car_Man foreign key (name) references cars(model); //Failed as Foreign key constraint is incorrectly formed. There is no index in the referenced table where the referenced columns appear.`

Rule 10: Integrity Independence

“A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.”

Explanation: No NULL values allowed in pirmary key attributes. The below SQL fails due to the FK link to Cust_id in the orders table.Also because it is a PK. Also this rules enforces keys to be unique. Which is why the second query fails.

SQL:

```
UPDATE customer set Cust_ID = null where Fname = 'James'
UPDATE customer set Cust_ID = 1 where Fname IN ('Barry','James')
```

Rule 11: Distribution Independence

"The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems."

Explanation: Like the data in this database or the email in my gmail account, when data is accessed and retrived the use does not know where the data is physically stored. In the case of gmail, some data retrived by a user can be physically located in various locations and/or disks but appear as one database in the users application..

Rule 12: Non-Subversion Rule

"If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints."

Explanation: If a DB allows low-level language access, not recomended, Then user access rights should be in place to restrict permissions to this. Allowing the manipulation of a DB free of the DBMS constraints can be very harmful to the DB if not done correctly. The Integrity rules need to be upheld. Example would be adding in NULL values where they are not allowed or inputing duplicate data where they should be unique ie, PK values.