500080 Database Design – ACW 2018-19

1. Understanding an existing database structure

a)

Cap (**NAME**, **LANGU**)
The primary key involves both attributes 'NAME' and 'LANGU' as individually can have duplicates, therefore a combination is required to form a composite primary key.

Course (COURSE_NAME, **COURSE_NUMBER**, CREDIT_HOURS, OFFERING_DEPT)
The COURSE_NUMBER is the primary key as it is unique, not null, and will not change. The COURSE_NAME may change in the future and hence should not be used as the PK.

Department_to_major (**Dcode**, DNAME)
The Dcode attribute is the PK for this table as it is unique and not null - DNAME contains null values.

Dependent (**PNO**, **DNAME**, RELATIONSHIP, SEX, AGE)
The PK is a composite of PNO and DNAME, as both these fields individually can have duplicates, but combined they are unique.

Grade_report (**STUDENT_NUMBER**, **SECTION_ID**, GRADE)
STUDENT_NUMBER and SECTION_ID form a composite PK as a student has many grades (repeated student_number) and a section_id can be repeated for many students.

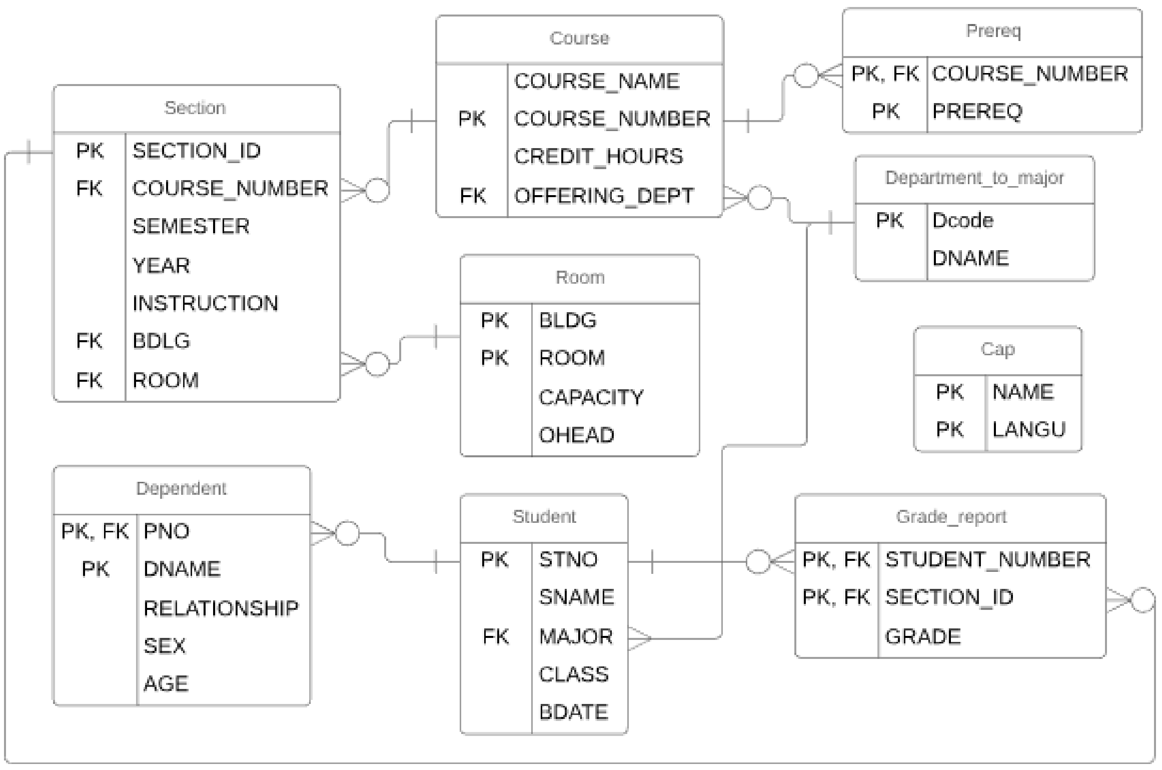
Prereq (**COURSE_NUMBER**, **PREREQ**)
The primary key is a composite of both attributes as it is possible for a course to have multiple prerequisites.

Room (**BLDG**, **ROOM**, CAPACITY, OHEAD)
BLDG and ROOM must be combined to form a unique composite PK. All attributes individually do not fulfil the criteria to be a PK.

Section (**SECTION_ID**, COURSE_NUMBER, SEMESTER, YEAR, INSTRUCTION, BLDG, ROOM)
The SECTION_ID is unique, not null, and will not change, hence is the PK for this table.

Student (**STNO**, SNAME, MAJOR, CLASS, BDATE)
A student can have a duplicate SNAME or BDATE and can be in the same major or class. Therefore, STNO is the only unique attribute and hence is the PK.

b)



2. Normalising a bulk data set

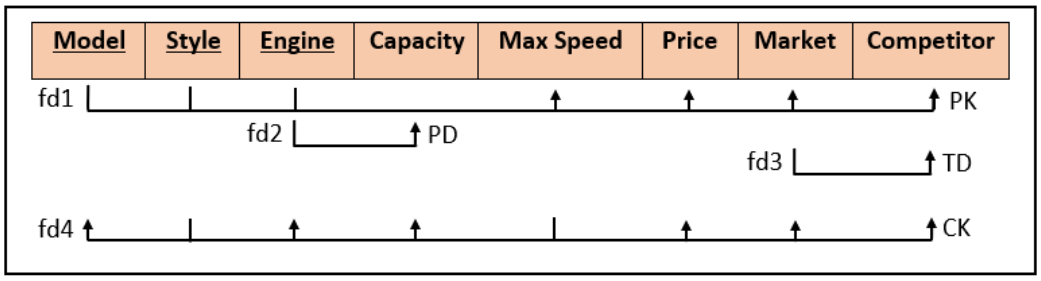
a)

Below is an illustration of the dependencies between attributes within the 'Cars' relation. Capacity is functionally dependent on Engine (Partial Dependency) due to all Capacity values being determined by each respective Engine value which is one of the attributes that makes up the primary key. Competitor is functionally dependent on Market (Transitive Dependency) as each Competitor value can be determined from its respective Market value, with Market being a non-PK attribute.

Model, Style, Engine is the most suitable CK that I have therefore chosen as the primary key. This key combination is unique for each tuple, will not change, and will maintain this uniqueness even if further tuples are added to the relation.

One other candidate key is Style, Max Speed. Although these attributes can uniquely identify each tuple in the existing table, there is potential that a car could be added to the relation that has a duplicate key that matches an existing tuple's Style and Max Speed.

In theory, Price alone could be a candidate key as all the tuple values are unique. However, price fluctuates far too much and realistically is not suitable. The addition of further tuples to the relation is highly likely to cause a duplication of price values against an existing tuple, which would violate the criteria for Price to be a unique key.



b)

<u>Model</u>	<u>Style</u>	<u>Engine</u>	Capacity	Max Speed	Price	Market	Competitor
Macho	Convertible	2900s	2847	155	£12900	Sports	Expresso
Macho	Coupe	2900s	2847	155	£11340	Executive	Gyro
Macho	Coupe	2400s	2395	131	£9990	Executive	Gyro
Fiasco	Saloon	2400s	2395	117	£10950	Executive	Gyro
Fiasco	Estate	2400s	2395	117	£11470	Commuter	Trio
Fiasco	Saloon	2400t	2395	101	£9950	Executive	Gyro
Fiasco	Estate	2400t	2395	101	£10370	Commuter	Trio
Commando	Saloon	2400t	2395	105	£8400	Commuter	Trio
Commando	Estate	2400t	2395	105	£8700	Commuter	Trio
Commando	Saloon	1900	1898	83	£7100	Commuter	Trio
Domino	Saloon	1900	1898	91	£4990	Domestic	Poncho
Domino	Estate	1900	1898	91	£5350	Domestic	Poncho
Domino	Saloon	1400	1365	79	£4450	Domestic	Poncho
Domino	Hatchback	1400	1365	79	£4100	Domestic	Poncho

The Cars table above is in 1NF in its initial state.

To put the data in 2NF, there should be no partial dependencies. As Capacity is partially dependent on Engine in the Cars table, the capacity column is removed, and a new Engine_Capacities table is created.

3NF removes any transitive dependencies, and as Competitor is Transitively dependent on Market, this column is removed from the cars table, and a new Market_Competitors table is created.

For BCNF, each functional dependency (A → B) should have A as a super key.

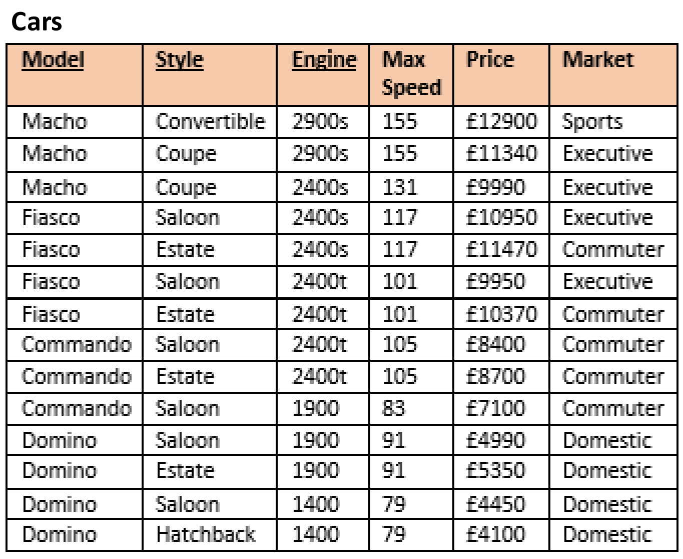
This normalisation ensures the minimisation of redundant data (duplicates) and that data is stored in a logical way.

SQL Statements to achieve normalisation of the 'Cars' data:

```
CREATE TABLE Engine_Capacities (
        Engine varchar(255),
        Capacity int,
);

INSERT INTO Engine_Capacities (Engine, Capacity)
```

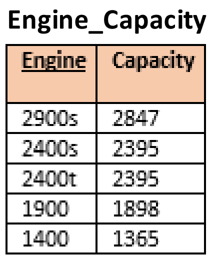
```
SELECT DISTINCT Engine, Capacity FROM Cars;

ALTER TABLE Cars
DROP COLUMN Capacity;

CREATE TABLE Market_Competitors (
        Market varchar(255),
        Competitor varchar(255),
);

INSERT INTO Market_Competitors (Market, Competitor)
SELECT DISTINCT Market, Competitor FROM Cars;

ALTER TABLE Cars
DROP COLUMN Competitor;
```

The final 3 tables (Cars, Engine_Capacities, and Market_Competitors) can be seen below:



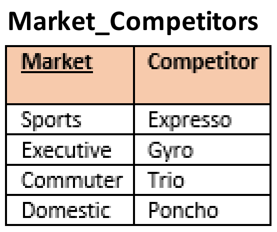
Cars

Model	Style	Engine	Max Speed	Price	Market
Macho	Convertible	2900s	155	£12900	Sports
Macho	Coupe	2900s	155	£11340	Executive
Macho	Coupe	2400s	131	£9990	Executive
Fiasco	Saloon	2400s	117	£10950	Executive
Fiasco	Estate	2400s	117	£11470	Commuter
Fiasco	Saloon	2400t	101	£9950	Executive
Fiasco	Estate	2400t	101	£10370	Commuter
Commando	Saloon	2400t	105	£8400	Commuter
Commando	Estate	2400t	105	£8700	Commuter
Commando	Saloon	1900	83	£7100	Commuter
Domino	Saloon	1900	91	£4990	Domestic
Domino	Estate	1900	91	£5350	Domestic
Domino	Saloon	1400	79	£4450	Domestic
Domino	Hatchback	1400	79	£4100	Domestic



Engine_Capacity

Engine	Capacity
2900s	2847
2400s	2395
2400t	2395
1900	1898
1400	1365



Market_Competitors

Market	Competitor
Sports	Expresso
Executive	Gyro
Commuter	Trio
Domestic	Poncho

You could change Cars.Engine to a EngineID attribute and Cars.Market to a MarketID attribute. You would then add EngineID and MarketID to the Engine_Capacity and Market_Competitors tables respectively as a first column and furthermore primary key.

3. Developing a potential entity model

a)

The following is a likely list of entities and attributes regarding the scheduling of musical performances.

Performance (Performance_ID, Date, VenueID, OrchestraID, ConductorID)
Venue (VenueID, Venue_Name, Venue_Manager)
Orchestra (OrchestraID, Email, Phone, Fee)
Conductor (ConductorID, Email, Phone, Fee)
Works_Performed (Performance_ID, Work_Cat_Num)
Work_Soloists_Required (Catalogue_Num, SoloistID)

Work_Information (Catalogue_Num, Title, Composer)
Soloist_Information (SoloistID, Name, Phone, Fee)

The Performance relation contains Performance_ID (the PK), date, and Foreign keys VenueID, OrchestraID, and ConductorID – each FK having a 0…* to 1 relationship. Thus, this reduces repeated data.

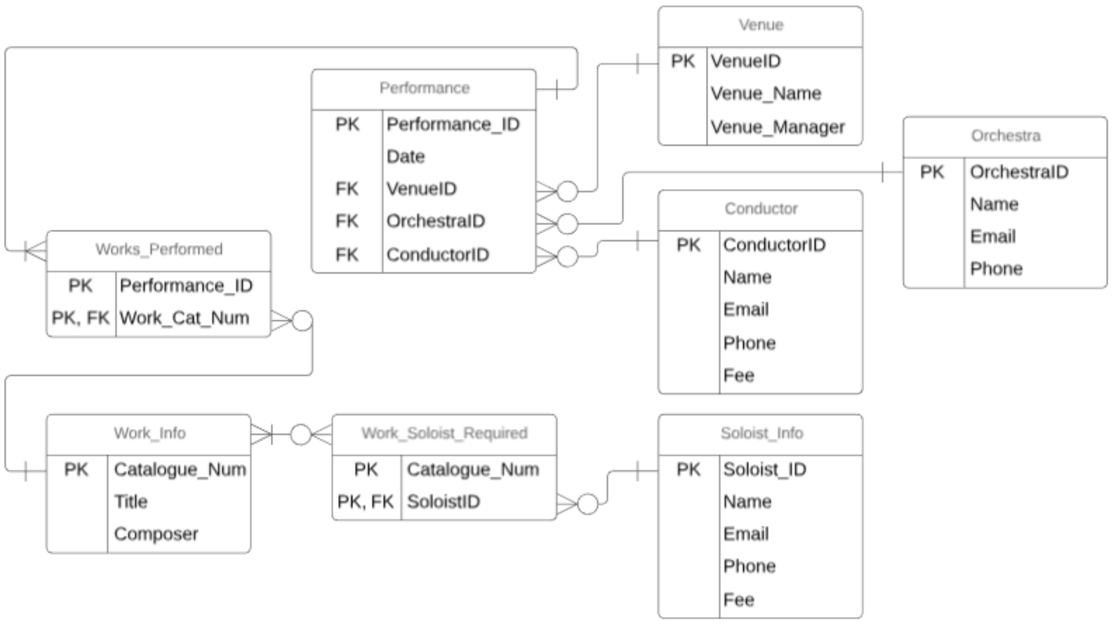
For each Performance_ID, there is a 1 to 1…* relationship to Works_Performed, so multiple works can be in connection with one performance. Both attributes in Works_Performed create a composite PK to avoid PK duplicates.

The Work_Cat_Num attribute in Works_Performed is a foreign key to the Work_Info relation in a 0…* to 1 relationship. Work_Information contains one tuple per catalogue_number to prevent duplication. Different performances can have the same work and performances can have many works.

This can be further extended in a 1…* to 0…* relationship to Work_Soloists_Required. A work can require either none, one or many soloists, and many different works can require the same soloist. This relation has a composite PK of the Catalogue_Num and SoloistID.

The Soloist_Info relation contains all the appropriate information about individual soloists, thus reducing duplicate information. The relationship between work_soloists_required and soloist_info is a 0…* to 1, as multiple works can point to the same soloist information tuple, but not every soloist needs to have a work pointing to it i.e. soloists can exist separately and are not reliant on performing a work to exist in the database.

b)


Venue
PK VenueID
Venue_Name
Venue_Manager
Performance
PK Performance_ID
Date
FK VenueID
FK OrchestraID
FK ConductorID
Orchestra
PK OrchestraID
Name
Email
Phone
Conductor
PK ConductorID
Name
Email
Phone
Fee
Works_Performed
PK Performance_ID
PK, FK Work_Cat_Num
Work_Info
PK Catalogue_Num
Title
Composer
Work_Soloist_Required
PK Catalogue_Num
PK, FK SoloistID
Soloist_Info
PK Soloist_ID
Name
Email
Phone
Fee