Solutions

Each answer shows the steps, then the **result**.

2.1

- a **candidate key** is any attribute, or set of attributes, that **could** be used as the primary key because it uniquely identifies each record
- a **secondary key** is an attribute used to **search or sort** the table on, and it need not be unique

2.2

- every **foreign key** value must match an existing **primary key** value in the table it refers to
- so a record cannot refer to one that does not exist, and a record still referred to cannot simply be deleted

2.3

- benefit: records can be found by that field without reading every row, so searching and sorting on it are much faster
- drawback: the index takes extra storage, and it must be updated every time a record is inserted, changed or deleted, which slows those operations

2.4

- a row is a **record**, also called a **tuple**
- a column is a **field**, also called an **attribute**

2.5

- it must already be in **1NF**
- and every non-key field must depend on the **whole** primary key, not on part of it – there must be no partial dependency

3.1

(a) `MemberID1`, `MemberID2` and `MemberID3` are a **repeating group**: the same piece of information is held in three numbered fields

- 1NF requires every field to hold a single atomic value and forbids repeating groups; this design also limits an event to three members and wastes space when fewer attend

(b) `EVENT(EventID, EventName, EventDate)`

- `ATTENDANCE(EventID, MemberID)` with the composite key `(EventID, MemberID)`, where `EventID` is a foreign key

3.2

(a) a **partial dependency**

- `ProductName` and `UnitPrice` depend on `ProductID` **alone**, which is only part of the composite key `(OrderID, ProductID)`

(b) `ORDER_LINE(OrderID, ProductID, Quantity)`

- `PRODUCT(ProductID, ProductName, UnitPrice)`, with `ProductID` a foreign key in `ORDER_LINE`

3.3

- the table is not in 3NF because of a **transitive dependency**
- `DepartmentName` depends on `DepartmentID`, which is a **non-key** field, rather than on the primary key `EmployeeID`
- `EMPLOYEE(EmployeeID, Name, DepartmentID)`
- `DEPARTMENT(DepartmentID, DepartmentName)`, with `DepartmentID` a foreign key in `EMPLOYEE`
- the consequence is worth knowing: a department's name is stored once, so it cannot be spelt two ways, and renaming it is one change

3.4

(a) it is a **many-to-many** relationship

- there is nowhere to put the foreign key: a teacher would need many course keys and a course would need many teacher keys, and a field holds one value

(b) add a **link table**, for example `TEACHES(TeacherID, CourseID)`, whose primary key is the two together

- `TeacherID` and `CourseID` are both foreign keys, so the one many-to-many becomes **two one-to-many** relationships: `TEACHER` to `TEACHES` and `COURSE` to `TEACHES`

3.5

- `OWNER` separated out, because the owner's details depend on the owner and not on the visit – a transitive dependency
- `PET` separated out, with `OwnerID` as a foreign key
- `VISIT` holding the date and the pet, with `PetID` as a foreign key
- the treatments are a **repeating group**, so they need a link table `VISIT_TREATMENT` with the composite key
- `TREATMENT` holding the fixed cost, because the cost depends on the treatment alone – a partial dependency in the link table

```
OWNER(OwnerID, OwnerName, Telephone)
PET(PetID, PetName, Species, OwnerID)
VISIT(VisitID, VisitDate, PetID)
VISIT_TREATMENT(VisitID, TreatmentID)
TREATMENT(TreatmentID, TreatmentName, Cost)
```

- foreign keys: `OwnerID` in `PET`; `PetID` in `VISIT`; `VisitID` and `TreatmentID` in `VISIT_TREATMENT`
- names may differ; the marks are for the five tables, their keys and the relationships between them