**CHAPTER 11 DATABASE**

## DATA STORAGE

**File based approach**

Data is stored in one or more separate computers files.

**Relational Database**

It is a way of structuring information in the tables, rows and columns.

**What is the limitation of using a file-based?**

- Data redundancy (data is repeated in more than one file)
- Data dependency (change to data means changes to program accessing the data)
- Lack of data integrity (entries that should be same can be different in different places)
- Lack of data privacy (all users have access to all data if a single flat file)
- Data inconsistency (If one file is updated but others aren't, the data becomes unreliable.

**What are the benefits of Relational database instead of flat file?**

- Reduced data redundancy
- Reduced data dependency
- Improved data integrity
- Improved data privacy
- Program-data independence
- Ability to create ad hoc queries.(queries designed for a particular purpose)

**Described the features of relational database that address the limitations of file based approach.**

- **Multiple tables are linked together**
  Which reduces data redundancy
  Increase data integrity
  Referential integrity can be enforced
- **Program-data independence means that**
  Structure of data can change and does not affect program
  Structure of programs can change and does not affect data.

- **Complex queries can be more easily written**

  To find a specific data.

- **Different users can be given different access rights**

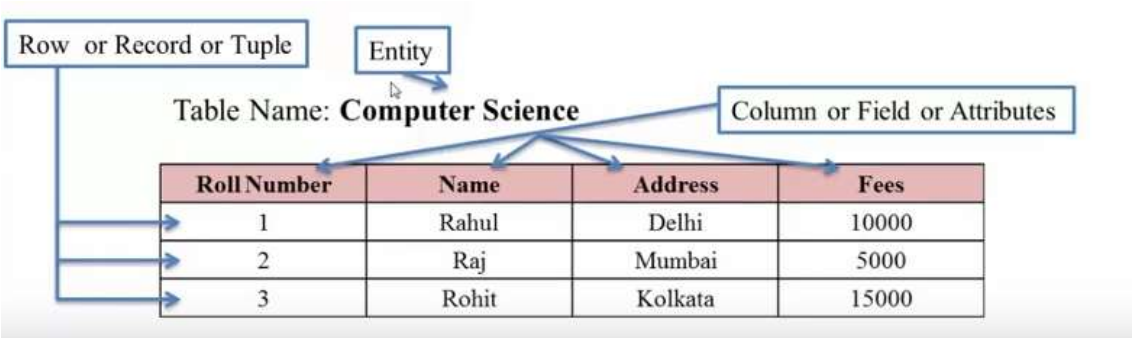
  Which improves security.

- **Different users can be given different views of the data**

  So they do not see confidential information and data privacy is maintained

## DATABASE TERMS

**DBMS:** Database Management System




| Term | Definition |
|---|---|
| Entity | A real-world object or concept that data is stored about (e.g. Student, Book) |
| Table | A collection of data about an entity, organised in rows and columns |
| Record (Tuple) | A single row in a table representing one instance of an entity |
| Field (Attribute) | A single column in a table, storing one piece of data about the entity |
| Primary key | A unique identifier for each record in a table (e.g. StudentID) |
| Candidate key | A field (or combination of fields) that could be used as a primary key |
| Secondary key | A field used for searching or sorting, but not necessarily unique |
| Foreign key | A field that links to the primary key in another table to create relationships |
| Relationship | A logical connection between two tables/entities |
| One-to-One | One record in a table relates to one record in another table |
| One-to-Many | One record in a table relates to many records in another table |
| Many-to-Many | Records in one table relate to many records in another, and vice versa |
| Referential integrity | Ensures foreign keys match a primary key in the related table to prevent broken links |
| Indexing | A technique to speed up searching in a table by creating an ordered list of key fields |

**Examples of Candidate key:** RollNo, CNIC, Passport Number

**What is data redundancy?**

Repeated data

**Explain how a relational database can help to reduce data redundancy?**

- Because each record of data is stored once and is referenced by a primary key.
- Because data is stored in individual tables.
- And the tables are linked by relationships.
- By the proper use of primary and foreign keys.
- By enforcing referential integrity.
- By going through the normalization process.

**How primary key and foreign key are used to link?**

Name is the primary key in table 1 links to Name which is foreign key in table 2.

**What is data integrity and how can you ensure data integrity?**

- Ensure data is consistent
- By enforcing referential integrity, validation rules.

**Example:**

The SOFTWARE_MANAGEMENT database has the following tables:

CUSTOMER_DETAILS(CustomerID, CompanyName, Address1, Address2, City)

SOFTWARE_PURCHASED(SoftwareName, SoftwareDescription, CustomerID,
LicenceType, LicenceCost, RenewalDate)

| Term | Example |
|---|---|
| Entity | CUSTOMER_DETAILS, SOFTWARE_PURCHASED |
| Foreign key | CustomerID |
| Attribute | SoftwareDescription |

**Example:**

The bank uses a relational database, ACCOUNTS, to store the information about customers and their accounts.

The database stores the customer's first name, last name and date of birth.

The bank has several different types of account. Each account type has a unique ID number, name (for example, regular or saving) and bonus (for example, $5.00, $10.00 or $15.00).

A customer can have more than one account.

Each customer's account has its own ID number and stores the amount of money the customer has in that account.

The bank creates a normalised, relational database to store the required information. There are three tables:

- CUSTOMER
- ACCOUNT_TYPE
- CUSTOMER_ACCOUNT

**(i)** Write the attributes for each table to complete the database design for the bank.

CUSTOMER (CustomerID, FirstName, LastName, DateOfBirth)

ACCOUNT_TYPE (AccountID, Name, Bonus)

CUSTOMER_ACCOUNT (ID, CustomerID, AccountID, Amount)

**Identify the primary key for each table that you designed in part (d)(i).**

CUSTOMER CustomerID

ACCOUNT_TYPE AccountID

CUSTOMER_ACCOUNT ID

[2]

**Identify one foreign key in one of the tables that you designed in part (d)(i).**

Table name Customer_Account

Foreign key CustomerID, AccountID

[1]

| Definition | Term |
|---|---|
| All the data about one entity | Table |
| The data in one row of a table | Tuple / Record |
| A column or field in a table | Attribute |

**Describe the role of a primary and a foreign key in a database relationship?**

- Primary key uniquely identifies each tuple.
- Primary key can be used as a foreign key in another table.
- To form relationship between the tables.

# STRUCTURES QUERY LANGUAGE

**Data types in SQL**

VARCHAR, BOOLEAN, INTEGER, REAL, DATE, TIME

## Data Definition Language (DDL)

Data Definition Language (DDL) is the part of SQL provided for creating or altering tables. These commands only create the structure. They do not put any data into the database.

**1. DDL Statement to create a database**

CREATE DATABASE <Name>;

**(i)** Write a Data Definition Language (DDL) statement to create the `EMPLOYEES` database.

CREATE DATABASE EMPLOYEES;

**2. DDL Statement to create a table**

```
CREATE TABLE <Name> (
        <attribute name> <datatype>,
        <attribute name> <datatype>,
        <attribute name> <datatype>,
        PRIMARY KEY (name) );
```

**Example:**

| EmployeeID | FirstName | LastName | DateOfBirth | Gender | DepartmentNumber |
|---|---|---|---|---|---|
| 156FJEK | Harvey | Kim | 12/05/1984 | Male | S1 |
| 558RRKL | Catriona | Moore | 03/03/1978 | Female | F2 |
| 388LMDV | Oscar | Ciao | 01/01/1987 | Male | F2 |

**Write a DDL statement to define the table EXPLOYEE_DATA, and declare EmployeeID as the primary key.**

CREATE TABLE EMPLOYEE_DATA (

EmployID VARCHAR (7),

FirstName VARCHAR,

LastName VARCHAR,

DateOfBirth DATE,

Gender VARCHAR (5),

DepartmentNumber VARCHAR (2),

PRIMARY KEY (EmployeeID) NOT NULL);

**3. DDL Statement to add new field**

ALTER TABLE <table name>
ADD <field name> <datatype>;

---

**(d)** The table STUDENT needs an additional field to store the student's telephone number, for example 012-3456.

Write a Data Definition Language (DDL) statement to add the new field to the table STUDENT.

ALTER TABLE STUDENT

ADD TelNum VARCHAR;

## DATA MANIPULATION LANGUAGE(DML)

There are three categories of use for Data Manipulation Language (DML)

- The insertion of data into the tables when the database is created
- The modification or removal of data in the database
- The reading of data stored in the database

**1. Add data into the table**

- **Inserts data into all columns**

  INSERT INTO <table name>

  VALUES (<data in column1 >, <data in column2>,.....);

- **Inserts data into specific columns**

  INSERT INTO <table_name> (column1, column2)

  VALUES (data1, data2);

**Example:**

| RoomNumber | RoomType |
|---|---|
| 1 | Standard |
| 2 | Double |
| 3 | Executive |
| 4 | Standard |

**(ii)** Room number **5** is a **Double** room.

Complete the Data Manipulation Language (DML) statement to add the details for room number 5 to the table ROOM.

INSERT ..... **INTO** ..... ROOM

VALUES (..... **5, "Double"** .....);

[2]

**(iii) The existing shop with ID 8765 has just used the existing supplier SUP89 for the first time. Write an SQL statement to add this information to the database.**

```
INSERT INTO SHOP_SUPPLIER (ShopID, SupplierID)
VALUES (8765, "SUP89")
```

## 2. Update data / record in a table

UPDATE <table name>

SET <column1> = <value1>, <column2> = <value2>, <column3> = <value3>…

WHERE <condition>;

**(ii) Fatima Woo is an Area B nurse with the nurse ID of 076. She has recently married, and her new family name is Chi. Write an SQL command to update her record.**

```
UPDATE AreaB
SET FaimilyName = "Chi"
WHERE NurseID = "076";
```
[3]

## 3. Display / Return a Value

SELECT <column1>, <column1>, ……
FROM <table name>
WHERE <condition>;

**Example:**

**(i) Write an SQL query to display the Nurse ID and family name for all Area B nurses with a specialism of 'THEATRE'.**

```
SELECT NurseID, FamilyName
FROM AreaB
WHERE Specialism = "THEATRE";
```
[3]

**Example:**

LESSON(LessonID, StudentID, InstructorID, LessonDate, LessonTime)

**(e) Write a Data Manipulation Language (DML) statement to return the date and time of all future lessons booked with the instructor whose InstructorID is Ins01.**

```
SELECT LessonDate, LessonTime
FROM Lesson
WHERE InstructorID = "Ins01"
AND LessonDate > #1/12/2026# ;
```

## 4. Delete data / record in a table

DELETE FROM <table name>

WHERE <condition>;

**Example:**

This is EXAM_DATA table, delete ExamID 00956124.

| ExamID | Subject | Level | TotalMarks |
|---|---|---|---|
| 00956124 | Computer Science | 2 | 75 |
| 00956125 | Computer Science | 3 | 120 |
| 00956126 | Mathematics | 2 | 100 |
| 00956127 | Mathematics | 3 | 150 |
| 00956128 | Physics | 2 | 70 |
| 00956129 | Physics | 3 | 80 |

DELETE FROM EXAM_DATA

WHERE ExamID = “00956124”;

**Example:**

**(c)** The database has the following tables:

CUSTOMER(CustomerID, CompanyName)

SOFTWARE(SoftwareID, SoftwareName, OperatingSystem, Description)

LICENCE(LicenceID, CustomerID, SoftwareID, DateOfPurchase, LicenceType, Cost, ExpiryDate)

**(iii)** The company needs a list of all software licences that have an expiry date on or before 31/12/2019.

Write an SQL query to return the fields CustomerID, SoftwareID, LicenceType, Cost and ExpiryDate for all licences that expire on, or before 31/12/2019. Group the output by CustomerID, and in ascending order of cost.

SELECT CustomerID, SoftwareID, LicenceType, Cost, ExpiryDate

FROM LICENCE

WHERE ExpiryDate <= #31/12/2019#

GROUP BY CustomerID ORDER BY Cost ASC;

## REFERENTIAL INTEGRITY

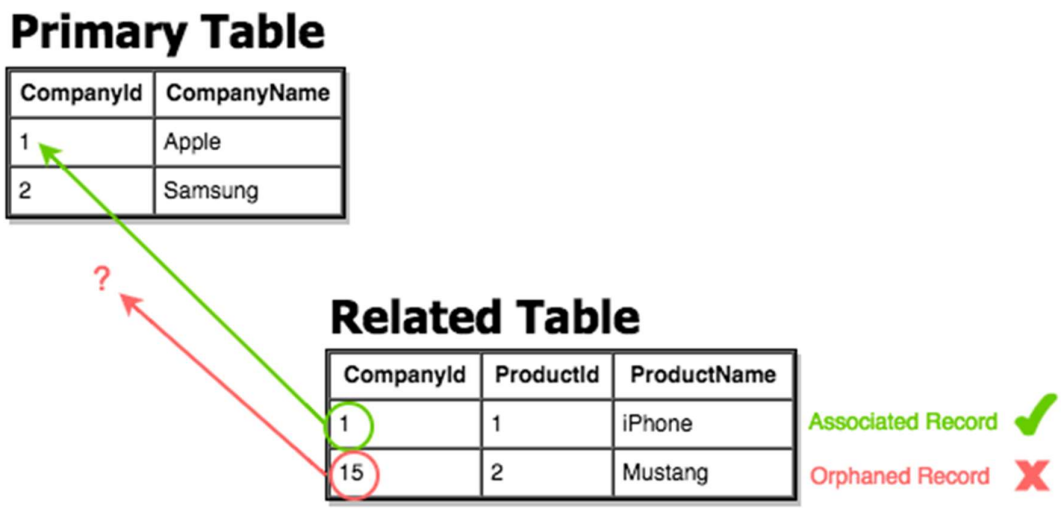
**What is meant by Referential Integrity?**

- Referential Integrity makes sure all data is up-to-date.
- Referential integrity ensures that every foreign key has a corresponding primary key.
- Referential Integrity prevents records from being added / deleted / modified incorrectly.
- Referential Integrity makes sure that if data is changed in one place the change is reflected in all related records.
- Referential Integrity makes sure any queries return accurate and complete results.
- Referential Integrity makes sure data is consistent.

**Example: Creating tables with Referential Integrity**

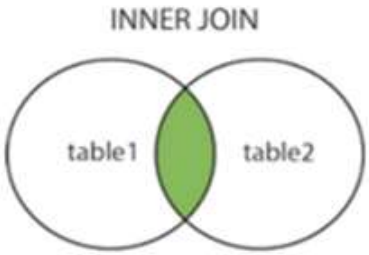
```
-- Create the Customer table first
CREATE TABLE Customer (
   CustomerID INT PRIMARY KEY,
   Name VARCHAR (50)
);

-- Create the Order table with a foreign key
CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   CustomerID INT,
   Product VARCHAR (50),
   -- Referential Integrity rule
   FOREIGN KEY (CustomerID) REFERENCES Customer (CustomerID)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);
```



## INNER JOIN

The `INNER JOIN` keyword selects records that have matching values in both tables.




```
SELECT <column_name>
FROM <table1>
INNER JOIN <table2>
ON table1.column_name = table2.column_name
WHERE <condition>;
```

**Example:**

A database, `FILMS`, stores information about films and actors.

Part of the database is shown:

```
ACTOR(ActorID, FirstName, LastName, DateOfBirth)
FILM_FACT(FilmID, FilmTitle, ReleaseDate, Category)
FILM_ACTOR(ActorID, FilmID)
```

**(c)** Complete the SQL script to return the IDs of all the actors in the film with the title Cinderella.

```
SELECT ActorID
FROM FILM_ACTOR
INNER JOIN FILM_FACT
ON FILM_FACT.FilmID = FILM_ACTOR.FilmID
WHERE FILM_FACT.FilmTitle = 'Cinderalla' ;
```

[4]

**Example:**

The table shows example data in `GAME_DEVELOPMENT`.

| GameName | Genre | TeamNumber | DevelopmentStage | ManagerID |
|---|---|---|---|---|
| Bunny Hop | Platform | 4 | Analysis | 23KP |
| Fried Eggs | Retro | 2 | Programming stage 1 | 9RTU |
| Create-a-game | Action | 1 | Acceptance testing | 11TF |

**(ii)** Another table, `PRODUCT_MANAGER`, is created.

```
PRODUCT_MANAGER(ManagerID, FirstName, LastName)
```

Complete the Data Manipulation Language (DML) statement to return the game name, genre and team number of all games managed by the product manager with the first name 'James' and the last name 'Fitz'.

```
SELECT GameName, Genre, TeamNumber
FROM GAME_DEVELOPMENT
INNER JOIN PRODUCT_MANAGER
   ON PRODUCT_MANAGER.ManagerID = GAME_DEVELOPMENT.ManagerID
WHERE PRODUCT_MANAGER.FirstName = 'James'
  AND PRODUCT_MANAGER.LastName = 'Fitz';
```

## Aggregate Functions COUNT() AVG() SUM()

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

```
SELECT SUM(column_name
FROM table_name
WHERE condition;
```

**Table Name: Products**

| ProductID | ProductName | SupplierID | CategoryID | Unit | Price |
|---|---|---|---|---|---|
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 22 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 21.4 |

**Write the SQL statement to find the number of products.**

SELECT COUNT (ProductID)
FROM Products;

**Write the SQL statement to find the average price of all products**

SELECT AVG (Price)
FROM Products;

**Write the SQL statement to calculate the sum of all prices**

SELECT SUM (Price)
FROM Products;

**Example:**

A relational database, TECHNOLOGY, stores data about the staff in a company and the computer devices used by the staff.

The database has the following tables:

STAFF(StaffID, FirstName, LastName, DateOfBirth, JobTitle)

DEVICE(DeviceID, Type, DatePurchased, StaffID)

**(b)** The database uses a Data Definition Language (DDL) and Data Manipulation Language (DML).

**(i)** Complete the SQL script to return the number of devices stored in the database for the staff member with the first name 'Ali' and last name 'Khan'.

```
SELECT ..... COUNT ..... (STAFF.StaffID)

FROM ..... STAFF .....

INNER JOIN DEVICE

..... ON ..... STAFF.StaffID = DEVICE.StaffID

WHERE STAFF.FirstName = 'Ali'

..... AND ..... STAFF.LastName = 'Khan';
```

# Entity-relationship (E-R) diagrams

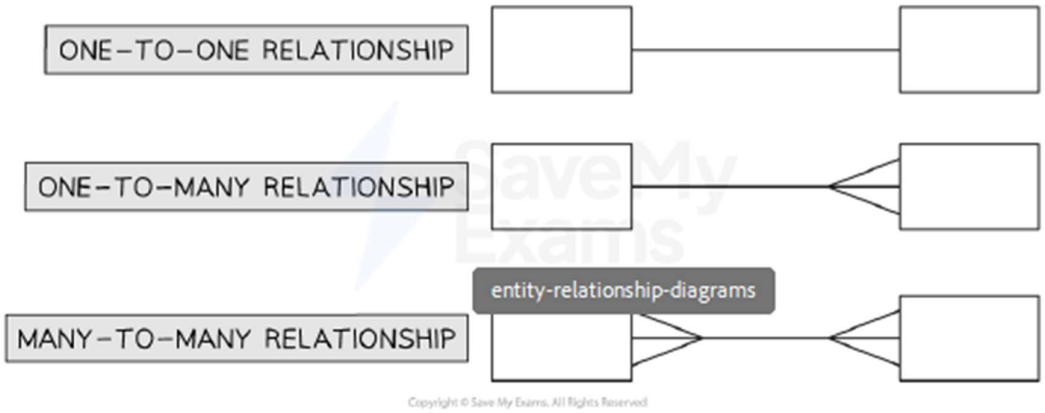
## What is an entity?

- An entity in something worthy of **capturing and storing data about** e.g. *students, orders, products, courses, customers*
- Entities become **tables** in a **relational database**
- Relational databases store different entities in **separate tables**
- Linking tables depends on the **relationships between entities**
- There are 3 types of (sometimes called degrees of) **relationships:**
  - **One-to-one**
  - **One-to-many**
  - **Many-to-many**
- Imagine a company has
  - A table of products
  - A table of customers
  - A table of the orders the customers have made
- What is the relationship between a **customer** and an **order**?
  - **One** customer can make multiple (**many**) orders
  - But each order relates to a specific (**one**) customer
  - So the relationship between customer and order is **one-to-many**
- Now consider the relationship between a **product** and an **order**
  - An order could have more than one (**many**) products on it
  - A product could be on more than one (**many**) order
  - So the relationship between order and product is **many-to-many**
- **One-to-one** relationships also exist but are not very common in databases

# What is an entity-relationship (E-R) diagram?

- An entity relationship diagram (**E-R**) is a **diagram that represents the entities** (tables) that will be in a database and the **relationships between these entities**
- The entities are drawn as **boxes** with the **entity** name in
- The relationships are drawn in as what is known as '**crow's feet notation**'

- This is how to draw the relationships in the exam:






- The **names** of the entities would go inside the boxes


(a) Describe the relationship shown above.

Many to One [1]

The relationship for Area B of the hospital is:


(i) Explain what the degree of relationship is between the entities B-NURSE and B-WARD.

Many to Many [1]

(c) The database has the following tables:

CUSTOMER(CustomerID, CompanyName)

SOFTWARE(SoftwareID, SoftwareName, OperatingSystem, Description)

LICENCE(LicenceID, CustomerID, SoftwareID, DateOfPurchase, LicenceType, Cost, ExpiryDate)

(i) Identify the type of relationship that exists between the tables CUSTOMER and LICENCE.

One to Many [1]

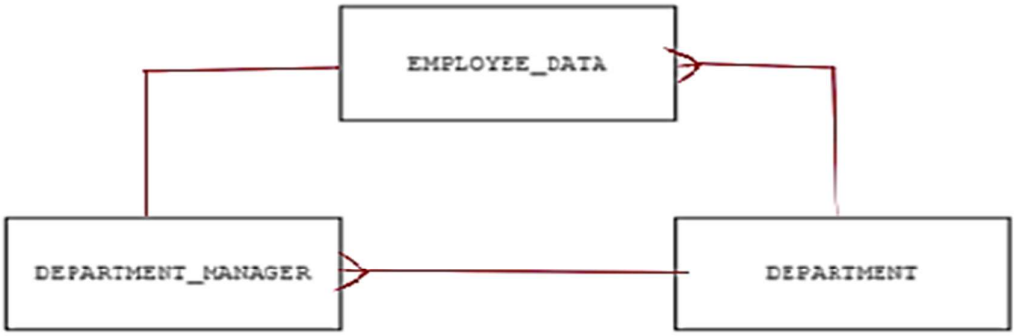
**(c)** In the company:

- An employee can be a manager.
- A department can have several managers and several employees.
- An employee can only belong to one department.

The EMPLOYEES database has three tables:

```
EMPLOYEE_DATA(EmployeeID, FirstName, LastName, DateOfBirth, Gender,
              DepartmentNumber)
DEPARTMENT(DepartmentNumber, DepartmentName)
DEPARTMENT_MANAGER(DepartmentNumber, EmployeeID, role)
```

Complete the entity-relationship (E-R) diagram for the EMPLOYEES database.




[3]

A driving school teaches people how to drive cars. The school has a relational database, DRIVING_SCHOOL, to store information about instructors, students, lessons and the cars used by instructors.

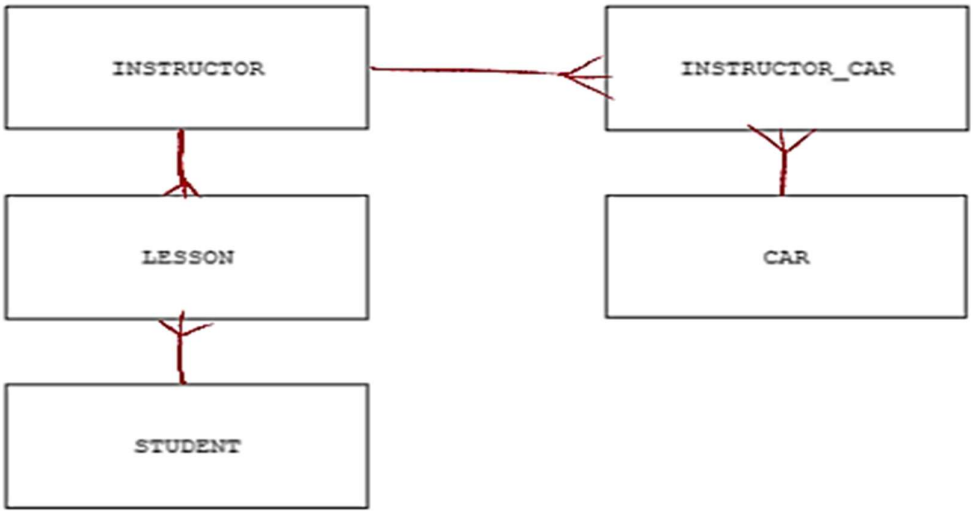
```
INSTRUCTOR(InstructorID, FirstName, LastName, DateOfBirth, Level)

INSTRUCTOR_CAR(InstructorID, Registration)

STUDENT(StudentID, FirstName, LastName, DateOfBirth, Address1)

LESSON(LessonID, StudentID, InstructorID, LessonDate, LessonTime)
```

**(b)** Complete the entity-relationship diagram for the database DRIVING_SCHOOL.



**(b)** The **normalised** relational database, SPORTS_CLUB, has the following table design.

MEMBER(MemberID, FirstName, LastName, MembershipType)

SESSION(SessionID, Description, SessionDate, SessionTime, NumberMembers)

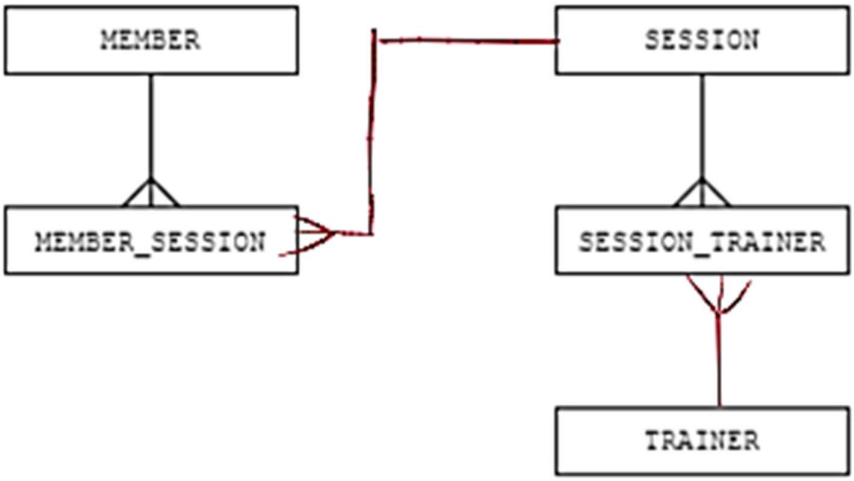
TRAINER(TrainerID, TrainerFirstName, TrainerLastName)

MEMBER_SESSION(MemberID, SessionID)

SESSION_TRAINER(SessionID, TrainerID)

**(i)** Anushka has designed an entity-relationship (E-R) diagram for SPORTS_CLUB.

Complete the entity-relationship (E-R) diagram.




[2]

---

USER(UserName, FirstName, SecondName, DateOfBirth)

PHOTO(PhotoID, UserName, Comment, UploadDate)

TEXTPOST(PostID, UserName, DateOfPost, TheText)

**(ii)** Draw the entity-relationship (E-R) diagram to show the relationships between the three tables.

# Normalised design

## What is a normalised design?

- A **normalised design** is a database structure that:
  - **Organises data efficiently**
  - **Eliminates unnecessary duplication**
  - **Ensures data integrity**
  - **Supports easy updates and accurate relationships**
- You may be asked to produce a normalised design based on:
  - A **written description** of a system
  - A **table of unnormalised data**
  - A set of **existing flat tables**
- You need to be able to:
  - Identify **repeating groups** or **duplicated data**
  - Break down data into **separate related tables**
  - Assign appropriate **primary keys** and **foreign keys**
  - Show the database in **1NF, 2NF, and 3NF**
  - Clearly show **relationships** between the tables

## Step-by-step process

| Step | What to do | Why it matters |
|---|---|---|
| 1 | Read the scenario or table carefully | Understand what entities (real-world things) are involved |
| 2 | Identify the fields and any repeated/duplicated values | These usually indicate the need for more than one table |
| 3 | Apply **1NF** – remove repeating groups, separate fields | Each field should hold a single value |
| 4 | Apply **2NF** – remove partial dependencies | Make sure non-key fields depend on the whole primary key |

| | | |
|---|---|---|
| 5 | Apply **3NF** – remove transitive dependencies | Non-key fields should not depend on other non-key fields |
| 6 | Assign **primary keys** for each table | Every table needs a unique identifier |
| 7 | Use **foreign keys** to link related tables | Ensures relationships and referential integrity |

## Example

- A student is enrolled on **multiple courses**
- Each course is taught by **a teacher**
- The table includes: StudentName, StudentID, CourseID, CourseName, TeacherName
- You would:
  - Identify **Entities**: Student, Course, Teacher
  - Create tables:
    - **Student** (StudentID, StudentName)
    - **Course** (CourseID, CourseName, TeacherID)
    - **Teacher** (TeacherID, TeacherName)
    - A link table: **StudentCourse** (StudentID, CourseID) – handles many-to-many

**Tips**

- Always check for repeating fields or duplicated values
- Watch for composite keys in link tables (especially many-to-many relationships)
- Check that all fields in a table depend only on the key, the whole key, and nothing but the key
- Label each stage of your work (1NF → 2NF → 3NF)

# First Normal Form (1NF)

## What is first normal form?

- For a **table** to be in first normal form it must:
  - **Contain atomic values**
    - Each column in a table must contain single, indivisible values
  - **Have no repeating groups**
    - Columns must not contain arrays or lists of values
  - **Have unique column names**
    - Each column must have a unique name within the table
  - **Have a unique identifier (primary key)**
    - Each row must have a unique identifier to distinguish it from other rows
- This customers table below has **no primary key** and the name is stored in **one field** so is not atomic
- This table is **not** in first normal form

**Table: Customers**

| name | phone | country |
|---|---|---|
| John Smith | 07373 929122 | UK |
| Iram Iravani | 07234 543422 | Iraq |
| Wu Zhang | 04563 523427 | China |
| Anne James | 09378 482894 | USA |
| Khalid Shirvani | 02343 536522 | France |

- This customers table below **has a primary key** and the name is stored in **two fields** so it is **atomic**
- This table is in **first normal form**

**Table: Customers**

| customer_id | forename | surname | phone | country |
|---|---|---|---|---|
| 1 | John | Smith | 07373 929122 | UK |
| 2 | Iram | Iravani | 07234543422 | Iraq |
| 3 | Wu | Zhang | 04563523427 | China |
| 4 | Anne | James | 09378482894 | USA |
| 5 | Khalid | Shirvani | 02343536522 | France |

# Second Normal Form (2NF)

## What is second normal form?

- For a table to be in second normal form it must:
  - **Fulfil all 1NF requirements**
  - **Only apply to tables with a compound primary key**
  - **Have full functional dependency**
    - **All non-prime attributes (attributes not part of the primary key) must be fully dependent on the primary key**
  - **Have no partial dependencies**
    - Non-prime attributes must not depend on only part of the primary key (in case of a composite primary key)
    - Separate tables should be created for partially dependent attributes
- In this table below, Course Title **only depends on part** of the compound primary key (the course code) and not the Date so this table **is not** in second normal form

**Table: Course**

| Course | Date | Course Title | Room | Capacity | Available |
|---|---|---|---|---|---|
| SQL101 | 03/01/2020 | SQL Basics | 4A | 12 | 4 |
| DB202 | 03/01/2020 | Database Design | 7B | 14 | 7 |
| SQL101 | 04/05/2020 | SQL Basics | 7B | 14 | 10 |
| SQL101 | 15/05/2020 | SQL Basics | 12A | 8 | 8 |
| CS50 | 31/05/2020 | C Programming | 4A | 12 | 11 |

- To turn this table into second normal form we will ensure:
  - Course Title is moved into its own **Course** table
  - A Session table is created to use the full key (Course, Date) for all time-specific info
  - No **partial dependencies** remain

**Table: Course**

| Course | Course Title |
|---|---|
| SQL101 | SQL Basics |
| DB202 | Database Design |
| CS50 | C Programming |

**Table: Session**

| Course | Date | Room | Capacity | Available |
|---|---|---|---|---|
| SQL101 | 03/01/2020 | 4A | 12 | 4 |
| DB202 | 03/01/2020 | 7B | 14 | 7 |
| SQL101 | 04/05/2020 | 7B | 14 | 10 |
| SQL101 | 15/05/2020 | 12A | 8 | 8 |
| CS50 | 31/05/2020 | 4A | 12 | 11 |

# Third Normal Form (3NF)

## What is third normal form?

- For a table to be in third normal form it must:
  - **Fulfil all 2NF requirements**
  - **Have no transitive dependencies**
    - Non-prime attributes must not depend on other non-prime attributes
  - **Have each non-prime attribute dependent solely on the primary key, not on other non-prime attributes**
  - **Have separate tables for attributes with transitive dependencies, and the tables should be linked using a foreign key**
- In this table below, the certificate depends on the title - this a transitive dependency and so this table **is not** in third normal form

| FilmID | Title | Certificate | Description |
|---|---|---|---|
| 12034 | Saw IV | 18 | Eighteen and over |
| 12035 | Spiderman 2 | 12A | Age 12 and over |
| 12036 | Shrek | U | Universal |

- To turn this table into third normal form we will ensure:
  - Transitive dependency removed by **separating** Description into its own table
  - The Film table only stores fields that **directly depend** on the key (FilmID)
  - A **foreign key** (Certificate) is used to maintain the relationship

**Table: Film**

| FilmID | Title | Certificate |
|---|---|---|
| 12034 | Saw IV | 18 |
| 12035 | Spiderman 2 | 12A |
| 12036 | Shrek | U |

**Table: Certificate**

| Certificate | Description |
|---|---|
| 18 | Eighteen and over |
| 12A | Age 12 and over |
| U | Universal |

## Worked Example

Match each normal form to its definition:

| | |
|---|---|
| First Normal Form (1NF) | All fields are fully dependent on the primary key. |
| Second Normal Form (2NF) | There are no repeating groups of attributes. |
| Third Normal Form (3NF) | There are no partial dependencies. |

[1]

**Answer**

| | |
|---|---|
| First Normal Form (1NF) | There are no repeating groups of attributes. |
| Second Normal Form (2NF) | There are no partial dependencies. |
| Third Normal Form (3NF) | All fields are fully dependent on the primary key. |

A software development company has a relational database, SOFTWARE_MANAGEMENT. The database stores details of the customers who have purchased software, as well as the software and licences that customers have purchased.

The SOFTWARE_MANAGEMENT database has the following tables:

CUSTOMER_DETAILS(CustomerID, CompanyName, Address1, Address2, City)

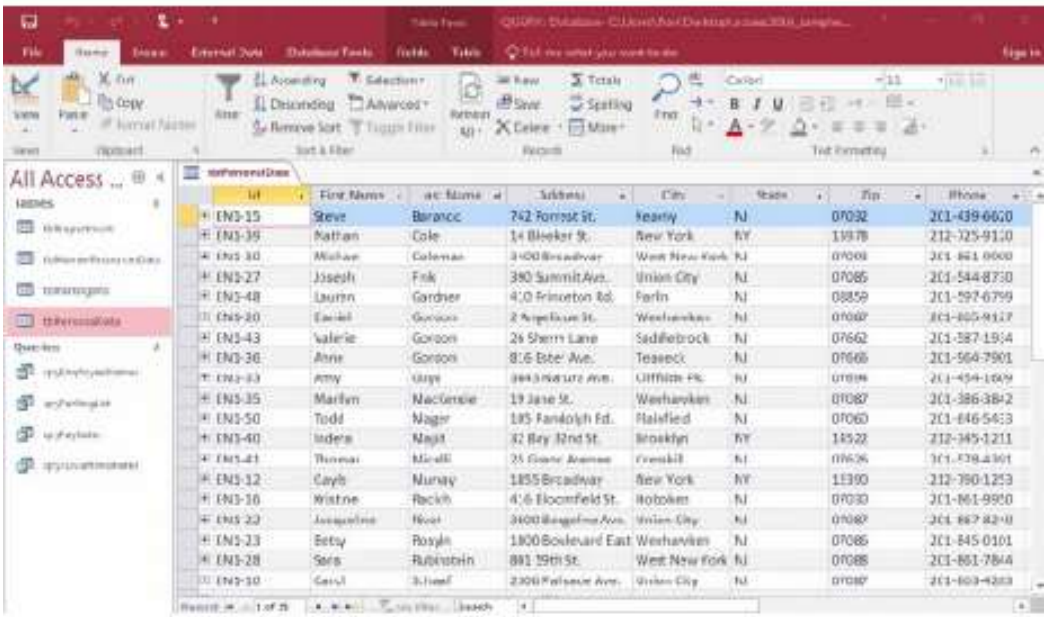
SOFTWARE_PURCHASED(SoftwareName, SoftwareDescription, CustomerID, LicenceType, LicenceCost, RenewalDate)

**(a)** Explain why this database is **not** in Third Normal Form (3NF). Refer to the tables in your answer.

- There are partial dependencies in the software purchase table. Softwaredescription only depends on software name and does not depend on both.

There is a non-key dependency in the software purchase table. Licence cost depends on licence type and does not depends on primary keys.

## DATABASE MANAGEMENT SYSTEM



**Describe the DBMS tools?**

**Developer Interface**

- To create user friendly features e.g. forms to enter the new booking
- To create outputs e.g. report of bookings on a given date
- To create interactive features e.g. buttons or menus.

**Query Processor**

- To create SQL queries
- To search for data that meets set criteria e.g. all bookings for next week.
- To perform calculation of extracted data, e.g. number of empty rooms.
- Organizes the results to be displayed**.**

**DBMS has a data dictionary. Describe what data dictionary stores.**

- Stores all the information about database
- For e.g. fields, datatypes, keys.

**Tasks performed by DBMS developer interface.**

- Create a table
- Set up relationships between tables
- Create a form
- Create a report
- Create a query.

**How DBMS software is used to ensure the security of the data?**

**Issue usernames and passwords**

- Stops unauthorized access to the data
- Strong password should be used.

**Access Rights**

- So that only certain usernames can read certain part of the data.
- Can be read only or full access.
- E.g. backup at the end of each day.

**Encryption of data**

If there is unauthorized access to the data it cannot be understood.

**Name and describe levels of schema of database.**

**External Schema**

The individual's view of database

**Conceptual Schema**

Describe the views which user of database might have.

**Logical schema**

Describe how the relationships will be implemented in the logic structure of the database.

**Physical/ Internal schema**

Describe how the data will be stored on the physical media.