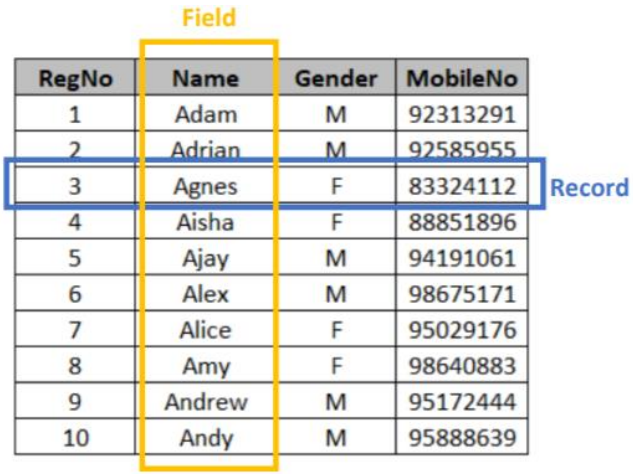
3.3 SQL Database

3.3.1 Relational Database Management System (RDBMS)

A **database** is a collection of data stored in an organised manner.

Four basic functions of persistent storage: **Create, Read, Update, Delete** (CRUD).



RegNo	Name	Gender	MobileNo
1	Adam	M	92313291
2	Adrian	M	92585955
3	Agnes	F	83324112
4	Aisha	F	88851896
5	Ajay	M	94191061
6	Alex	M	98675171
7	Alice	F	95029176
8	Amy	F	98640883
9	Andrew	M	95172444
10	Andy	M	95888639

Record: row
Field: column

3.3.2 Keys

A **candidate key** is an attribute or a combination of attributes that can uniquely identify each record.

A **primary key** is a candidate key that is most suited to become the main key.

A **secondary key** is a candidate key that is not chosen as the primary key.

A **composite key** is a combination of two or more attributes that can uniquely identify each record.

A **foreign key** is an attribute in one table that references the primary key in another table.

3.3.3 Data Redundancy, Data Inconsistency, Data Dependency

Reasons to use RDBMS over flat files such as .txt and .csv:

- **Data redundancy** is where the same data is being stored more than once.
- **Data inconsistency** is where different versions of the same data exist at the same time.

Data dependency:

1. **Functional dependency** is a direct relationship where one attribute uniquely identifies another attribute in the same table.
2. **Transitive dependency** is an indirect relationship formed by two functional dependencies.

3.3.4 Normalisation

Normalisation is the process of organising the tables in a database to reduce data redundancy and inconsistency.

1. **Unnormalized Form (UNF)**
2. **First Normal Form (1NF)**
   Conditions:
   - All attributes should hold only atomic values (each record has to be unique).
3. **Second Normal Form (2NF)**
   Conditions:
   - In 1NF
   - All non-key attributes should be fully dependent on the entire primary key.
4. **Third Normal Form (3NF)**
   Conditions:
   - In 2NF
   - No transitive dependencies.

3.3.5 Entity-Relationship (ER) Diagram

1. One to One

2. One to Many

3. Many to Many

Example: Database of students, their respective CCAs and CCA information

Legend:
___: Primary Key
*: Foreign Key

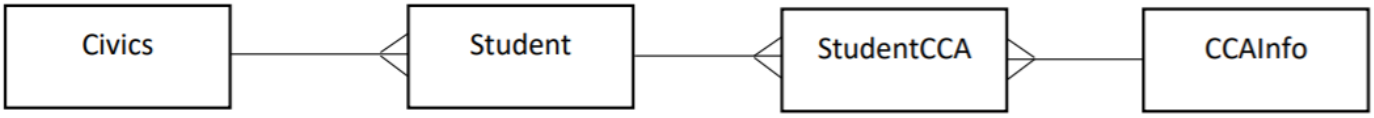
1. UNF
   StudentCCAInfo (MatricNo, Name, Gender, CivicsClass, CivicsTutor, HomeRoom, CCAName1, CCATeacherIC1, CCAName2, CCATeacherIC2, CCAName3, CCATeacherIC3)

2. 1NF
   StudentCCAInfo (MatricNo, Name, Gender, CivicsClass, CivicsTutor, HomeRoom, CCAName, CCATeacherIC)

3. 2NF
   Student (MatricNo, Name, Gender, CivicsClass, CivicsTutor, HomeRoom)

StudentCCA (**MatricNo***, **CCAName***)
CCAInfo (CCAName, CCATeacherIC)

4. 3NF
   Student (MatricNo, Name, Gender, **CivicsClass***)
   Civics (CivicsClass, CivicsTutor, HomeRoom)
   StudentCCA (**MatricNo***, **CCAName***)
   CCAInfo (CCAName, CCATeacherIC)



3.3.8 SQL Query

Example
Person(PersonID, Name, Age)
Employee(EmployeeID, HoursWorked)
Company(**PersonID***, **EmployeeID***)

1. **CREATE TABLE**

```
CREATE TABLE 'Person' (
'PersonID' INTEGER UNIQUE NOT NULL PRIMARY KEY AUTOINCREMENT,
'Name' TEXT NOT NULL,
'Age' INTEGER NOT NULL CHECK(Age > 0 AND Age < 100)
);

CREATE TABLE 'Employee' (
'EmployeeID' INTEGER UNIQUE NOT NULL PRIMARY KEY
AUTOINCREMENT,
'HoursWorked' REAL NOT NULL
);

CREATE TABLE 'Company' (
'PersonID' INTEGER NOT NULL,
'EmployeeID' INTEGER NOT NULL,
PRIMARY KEY('PersonID', 'EmployeeID'),
FOREIGN KEY('PersonID') REFERENCES Person('PersonID'),
FOREIGN KEY('EmployeeID') REFERENCES Employee('EmployeeID')
)
```

   a. **PRIMARY/COMPOSITE KEY**

```
PRIMARY KEY('PersonID', 'EmployeeID')
```

   b. **FOREIGN KEY**

```
FOREIGN KEY('PersonID') REFERENCES Person('PersonID')
```

   c. **CHECK**

```
CHECK(Age > 0 AND Age < 100)
```

2. **INSERT INTO**

```
INSERT INTO Person('Name', 'Age')
VALUES(?, ?)
```

3. **SELECT FROM**

```
SELECT *
FROM Person
```

   a. **DISTINCT**

```
SELECT DISTINCT Person.Name
FROM Person
```

   b. **IS NOT NULL**

```
WHERE Person.Age IS NOT NULL
```

   c. **LIKE**

```
WHERE Person.Name LIKE 'J%'
OR Person.Name LIKE '%s'
```

   d. **ORDER BY**

```
ORDER BY Person.Age ASC
```

   e. **GROUP BY**

```
GROUP BY Person.Name
```

   f. **LIMIT**

```
LIMIT 5
```

   g. **SUM, AVG, COUNT, MIN, MAX**

```
SELECT SUM(Employee.HoursWorked)
FROM Employee
```

4. **UPDATE SET**

```
UPDATE Person
SET Name = 'Pablo', Age = 18
WHERE PersonID = 1
```

5. **DELETE FROM**

```
DELETE FROM Person
WHERE Person.PersonID = 1
```

6. **DROP TABLE**

```
DROP TABLE Company
```