**Databases**

**Relational databases** contain *related data*, all information should be related to each other

Three types of databases:

* **Flat-file Database** 🡪 stores everything in one table, good for small numbers of records related to a single topic
* **Relational Database** 🡪 gives you the ability to separate masses of data into numerous tables, they are linked to each other using *keys*
* **Non-Relational or Big Data** 🡪 used for data analytics and business intelligence
  + E.g., MongoDB, Vertica, etc

**Primary Key**

A PK uniquely identifies each record in the table, most tables should have atleast one PK

* Each table can have more than one column which is part of its primary key, known as a **Composite Key**
* It can either be an attribute that is *guaranteed* to be **unique,** or it can be generated by the DBMS
* The DBMS will enforce the uniqueness of the PK, not allowing repeated records to exist in the table

**Primary key constraints:**

* PK must be unique
* Must *always* have an entry, CANNOT be blank or null
* The value must never change
* A table with numbers and letters

  Description automatically generatedEach table may have a *maximum* of *one* PK

**Two types of Primary Key:**

**Simple** and **Composite**

**Composite** means that there is more than one PK in a table

**Foreign Key**

Foreign keys ensure that the row of information in Table A corresponds to the correct row of information in Table B

Benefits of FK:

* The constraint is used to prevent actions that would destroy links between tables
* A close-up of a list

  Description automatically generatedIt prevents *invalid data* from being inserted into the FK column, because it has to be one of the values contained in the table it points to

FK constraints:

* There is no *uniqueness* constraint
* A table can have *any number* of FKs
* A row cannot be deleted from a reference table if it is in use via a foreign key

A diagram of a database

Description automatically generatedEach table has a relationship to each other through FKs

**Data Modelling**

Details how a database will be *organised* and how they *relate* to each other

Three levels:

* **Conceptual** 🡪 business level of the model
* **Logical** 🡪 define the relationships between them, PK and FKs
* **Physical** 🡪 detailing the specific specifications of the data model

A good model reflects *real-world relationships*

**Entity Relationship Diagrams (ERD)**

* **Entity** refers to *storing objects*
* **Attributes** refers to what the object *has*
* **Relationship**entails how we can relate the information together

ERDs can be used to describe complicated data using tables and lines that display the *type* of relationship, and what is table is related to another

**Crows Feet Notation** 🡪 lines and shapes used to detail the relationship between entities

* a line with three lines on the end that resembles a crow’s foot, it represents a ***many relationship***
* a horizontal line represents *one*, this may be maximum of one or a minimum of one
* A circle represents *zero*, so can be zero or many or zero to one

***A diagram of a number of objects

Description automatically generated***

A black rectangle with white text

Description automatically generated

Where the lines leads *to* is what the relationship is referring to

**Normal Form**

Increased normalisation increases the *depth* of the data which can make it more difficult to change after the fact

* A screenshot of a table

  Description automatically generatedIf a database is going to be changed *frequently* (**high write, low read**) then it may be inefficient to normalise

|  |  |  |
| --- | --- | --- |
| **1st Normal Form** | **2nd Normal Form** | **3rd Normal Form** |
| The following conditions must be met:   * Everything **Atomic** 🡪 data presented as small as possible   + All data should be in *small packages* * There should be *no repeating groups* – no **Redundancy** | The following conditions must be met:   * It is in 1NF * All *non-key* attributes are fully functional *dependent* on the PK | The following conditions must be met:   * It is 2NF * There is no **transitive functional dependency** 🡪 a non-key column is *functionally dependent* on another non-key column, which is *functionally dependent* on the PK |

A table with numbers and words

Description automatically generated

This is not in 2NF as the ‘location’ is not *dependent* on the two PKs (product id and store), it is only dependent on one

* The first row shows a PK of 2 (1+1) which relates to the London location, but row 3 also has a location of London but a PK of 3 – a *contradiction*

A close-up of a table

Description automatically generatedThis is fixed by separating the non-key attribute into another table, so that it is now dependent on the relevant store (only one PK)

A table with text on it

Description automatically generated

This is not in 3NF as genre type and genre id and functionally dependent on each other – they display the same information and genre type is not dependent on the PK, only the genre id which is dependent on the PK – non-key attribute dependent on another non-key attribute

* This is the trickiest the NF but usually will aim for just 1NF

***There are additional normal forms including…***

* Boyce-Codd NF
* 4NF
* 5NF

|  |  |  |
| --- | --- | --- |
| **Boyce-Codd NF (BCNF)** | **4NF** | **5NF** |
| The following conditions must be met:   * It is in 3NF * Must be no non-trivial functional dependencies of attributes on anything other than a subset of a candidate key | The following conditions must be met:   * It is in BCNF * The table should not have any **multi-valued dependency**   **Multivalued dependency** creates redundancies and can contribute to inconsistent data  This is when two attributes in a table are independent of each other, but both depend on a *third* attribute | The following conditions must be met:   * It is in 4NF * It cannot be further non loss decomposed (join dependency)   A relation is said to have **join dependency** if it can be recreated by joining multiple sub relations and each of these sub relations has a subset of the attributes of the original relation |