Intro to Databases

A database is a shared collection of logically related data and its description, designed to meet the information needs of an organisation.

The database represents the entities (real-world things), the attributes (their relevant properties), and the logical relationships between the entities. In other words, the database holds data that is logically related.

When we analyse the information needs of an organisation, we attempt to identify entities, attributes, and relationships. An entity is a distinct object (a person, place, thing, concept, or event) in the organisation that is to be represented in the database. An attribute is a property that describes some aspect of the object that we wish to record (name, brand, owner), and a relationship is an association between entities(FaC student has an association with a cohort, a cohort has an association to a location).

The database is a single, possibly extremely large repository of data that can be used simultaneously by many users. Instead of disconnected files with redundant data, all data items are integrated with a minimum amount of duplication. The database holds not only the operational data, but also a description of this data. We'll come back to this in a minute.

Slide 4

Why would you use one?

Why would you use a database when you can already handle data within node?

And not all websites do use databases. So why would you use one? Let's answer this quick question first:

Slide 5

Exercise 1: Why use a library when you already have a bookcase?

Most of you have a collection of books at home. Why would you use a library in that case?

Databases are built in a way that allows them to handle lots of data, they do lots of clever stuff behind the scenes to make sure that when you are asking for something they do it as quickly as possible.
You don't need to have all your data all the time. The majority of database interactions online only need information on one user or one product. Using a database means you can have tens of thousands of millions of users and products in the database and your server only needs to be able to get access to specific ones. So databases help keep things relatively clean and quick.

Slide 6
Let's answer another question.

Exercise 2: Why use a database when you have a spreadsheet?

Slide 7
DBs help to solve the problems of:

1. Size of data - a spreadsheet solution is fine when you have 100 records . It may not work if you have 2 million records. You have to split up the file into multiple subfiles. This will create a problem of speed. It will take you long time to find a record.
2. Ease of updating data - multiple people cannot edit the file at the same time. They will be overwriting everyone else's changes all the time.
3. Accuracy - data accuracy is hard to maintain. There is no validation of data entry and hence accuracy is in question. Anyone can type anything. You can enter wrong spelling and wrong dates for example.
4. Security - you cannot secure data in spreadsheets and text files. Anyone can access the files and can see any data within the files. This solution will not work for payroll and healthcare departments where privacy is pivotal. Data protection is an incredibly important consideration when storing user data and if not understood and adequately implemented can get you in a lot of trouble.
5. Redundancy - the problem of duplication of data. Multiple copies of the same data would eventually find its way into spreadsheets and documents. It is not easy to update multiple copies of the same data at once.

A database prevents these issues. Databases let you store data and are intrinsically set up to allow that data to grow. You can modify that data, and multiple people can do the same modification at the same time.

The data is validated (is the database receiving the information it expected, and in the correct format?).

You can control the security; for example, who can view and who can modify the data. You can track who did what. It is very common to have multiple levels of user privileges from super-admins; who can change and read any data, even delete the entire database. To the lowest level user who can only read data, and only from select tables.

Databases ***should*** also survive crashes and hardware issues without corrupting the data. They allow you to easily make backups and restore from those backups if needed.

Slide 8
How does it all fit together?

Slide 9
Key database concepts

	Real world	Databases
Language	English (UK vs US)	SQL (syntax differences)
Database Management System	Library	PostgreSQL, MySQL
Database	Collection of books	Database
Schema	Classification system	Schema
Client	Library terminal, librarian	PSQL, node-postgres

And actually the library and bookcase example helps in terms of understanding some of the key concepts with databases.

So SQL which is a ‘structured query language’ essentially means that it’s a language which was created to handle relational databases. It’s just a language, it’s not a database system itself, it’s a kind of syntax, a selection of words that are used to communicate with database management systems. But just like with the real world the fact that you speak English doesn’t necessarily mean that everyone else speaks the same English as you do and each dbms speaks a slightly different dialect of sql. There is no guarantee that the SQL you’re familiar with will work perfectly on any other given database. Though any differences will generally be minor syntax changes, SQL is the standard/standardised language for interacting with databases.

A dbms itself is a kind of the library, it’s the overarching system that’s been set up to help you find and access the data that’s inside. It’s a software system that enables you to define, create, maintain, and control access to the database. The one you will be using is Postgresql. There are also a lot of other ones like MySQL. And fundamentally each one of those dbms is the way of setting up the library which you will be dealing with.

The database itself is like the books in the library, it’s the data and organisation of it specifically. So within your dbms you’ll often have multiple databases which is quite confusing at first but you’ll get your head around it after a while.

Potentially the most important part for us to worry about is the schema. The schema is the classification system for everything within your database. It is not the data itself. The schema exists independently of the data, but it describes all of the different things that you’ll have in your system and what those things are going to look like. It’s the overall description of the database; similar to a map. For a book it might be number of words, author, publication date, genre, those kind of things; and that description is true regardless of whether you’ve actually got a specific book, one book, or all of the books. Generally when you are creating and

setting up the databases for your own projects what you're really doing is designing a schema.

When you want to access the data in your database you use some kind of client, you are not using the database directly, you'll be using another piece of software to access it, to communicate with it. So that's an equivalent of using a library terminal to look through the collection of books, you can ask the librarian or use the library terminal. They are just points of entry into looking through the database. And from your perspective you'll be using a program called psql/pgcli which is an interface allowing you to look at and use your postgres databases. That's quite a lot to take in. Do you have any questions?

Slide 12
As we've mentioned earlier this talk is about relational databases which means it's about databases where things have relationships. Which is a fundamental element of database design that you will be working within. So let's do a little exercise.

Exercise 2: What kind of relationships exist between people? What type of relationships are they?

I mentioned earlier that you have entities in databases which are like the real world things that you'll be dealing with. It can be an animal, a person, a location, all abstract things, like an event or a transaction. And generally speaking a person may have several pets for example, and the nature when entities can have various relationships between each other will dictate how you set up your databases.

Slide 13

So first of all let's just come up with a list of relationships that a car may have with its components. So I'm just going to say Tyres to start it off. Shout out any other.

Relationship	**Type**
Car -> Tyres	One to many
Car -> Steering wheel	One to One
Car -> Driver	Many to Many

Slide 14

In SQL servers, these relationships are defined using **Primary Key-Foreign Key** constraints.

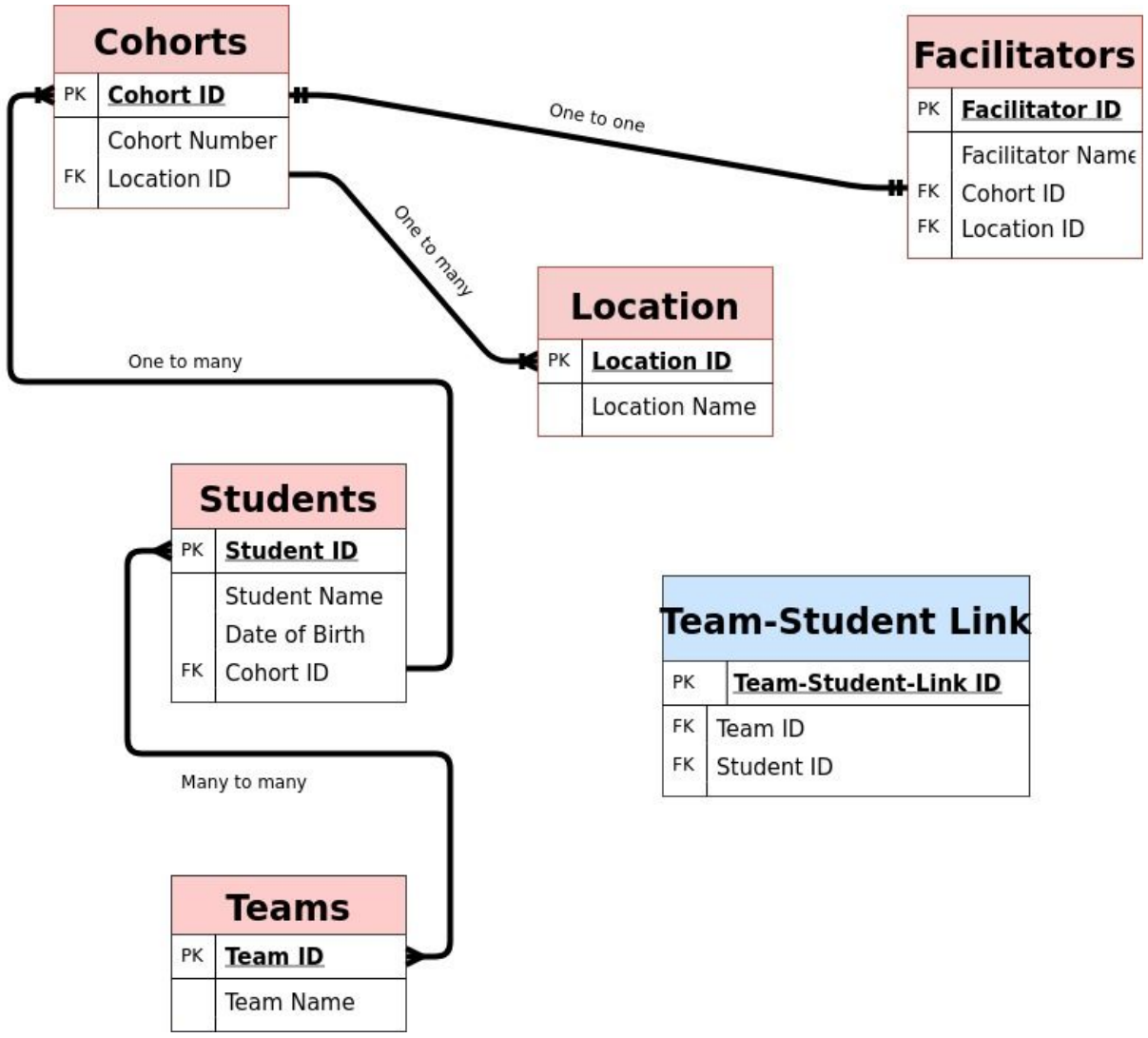
A link is created between two tables where the primary key of one table is associated with the foreign key of another table using database relationships.

Slide 15

A primary key is a unique attribute which the database uses to identify a row in a table (a tuple). For our databases we will make sure that it is a unique, auto-incrementing ID which is filled in by the database - in other words it is NEVER NULL and in fact a table must have a primary key; this will help keep everything neat. A primary ID number will only ever be issued once; if, for example, you delete the latest record with an ID of 7 in your table FAC, and then make a new record, the new record's ID will become 8. This is really useful! If we need to refer to a record in separate table, we can reference this ID as a foreign key, to be sure we are referring to the actual record, as every primary key is unique. This will help prevent duplicate data.

Slide 16

Let's have a look at the mock schema of FAC:



Slide 17

1. One-One Relationship (1-1 Relationship)

One-to-One (1-1) relationship is defined as the relationship between two tables where both the tables should be associated with each other based on only one matching row. This relationship can be created using **Primary key-Unique foreign key constraints**.

```
CREATE TABLE cohorts
(
Pk_Cohort_Id INT PRIMARY KEY,
Cohort_Name VARCHAR(255),
);

CREATE TABLE facilitators
(
Pk_Facilitator_Id INT PRIMARY KEY,
Facilitator_Name VARCHAR(255),
Fk_Cohort_Id INT UNIQUE FOREIGN KEY REFERENCES cohorts(Pk_Cohort_Id)
);
```

One-to-One Relationship is implemented using **cohorts(Pk_Cohort_Id)** as the Primary key and **facilitators(fk_cohort_id)** as (Unique Key Constraint-Foreign Key).

Therefore, it will always have only one matching row between the Cohorts-Facilitators table based on the **cohorts(Pk_Cohort_Id)-facilitators(fk_cohort_id)** relationship.

Slide 18

2. One-Many Relationship (1-M Relationship)

The One-to-Many relationship is defined as a relationship between two tables where a row from one table can have multiple matching rows in another table. This relationship can be created using **Primary key-Foreign key relationship**.

```
CREATE TABLE cohorts
(
Pk_Cohort_Id INT PRIMARY KEY,
Cohort_Name VARCHAR(255),
);

CREATE TABLE students
(
Pk_Student_Id INT PRIMARY KEY,
FullName     VARCHAR(255),
```

```
Fk_Cohort_Id   INT FOREIGN KEY REFERENCES cohorts(Pk_Cohort_Id)
);
```

One-to-Many Relationship is implemented using **cohorts(Pk_Cohort_Id) as the Primary Key and Students (Fk_Cohort_Id) as (Foreign Key)**. Thus, it will always have only One-to-Many (One Cohort-Multiple Students) matching rows between the Cohorts-Students table based on the Cohorts (Pk_Cohort_Id)-Students(Fk_Cohort_Id) relationship.

Slide 19

3. Many - Many Relationship (M-M Relationship)

M-M relationship is defined as a relationship when one or more rows in a table are associated with one or more rows in another table.

```
CREATE TABLE students
(
Pk_Student_Id INT PRIMARY KEY,
Cohort_Name VARCHAR(255),
);

CREATE TABLE teams
(
Pk_Team_Id INT PRIMARY KEY,
Team_Name     VARCHAR(255),
);

CREATE TABLE connections
(
Fk_Student_Id INT FOREIGN KEY REFERENCES students(Pk_Student_Id),
Fk_Team_Id INT FOREIGN KEY REFERENCES teams(Pk_Team_Id),
);
```