Topic 10 Fundamentals of Databases

Data Models and Entity Relationship Modelling

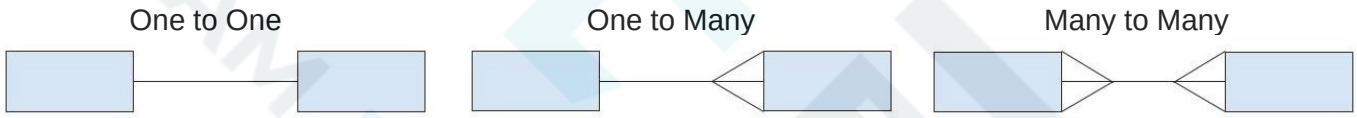
Creating a data model is one of the first steps in designing a database, as the name suggests, it helps us understand the data which needs to be stored and how it is related. Entity relationship modelling is a top down technique which helps to visualise and understand this data, and to produce an entity relationship diagram which can be used to visually show the data and relationships.

When discussing data models and entity relationship modelling there are three key terms to be aware of:

1. Entity; something about which data is stored such as a book, car or customer.
2. Attribute; information about an entity, such as a name, address or colour.
3. Relationship; how one entity links to another, for example one customer may own many cars.

Entity Relationship Diagram

Entity relationship diagrams use rectangles to represent entities and use lines to show relationships between entities. One of three different relationships shown below is possible, and the one to many relationship is the most common. This is also known as the cardinality.



Entity Descriptions

IT is important to be able to use standard notation to describe entities. The standard format is:

EntityName (PrimaryKey, attribute1, attritbute2, …)

The examples below show how this is used, entity names should be written as singular rather than plural, so classroom not classrooms.

Classroom (RoomNumber, Building, Floor, Size)

Student (StudentID, FirstName, LastName, Phone)

It is also possible to create a compound primary key from two attributes as shown below

Qualification (StudentID, Subject, ,Date, Level)

Relational Databases

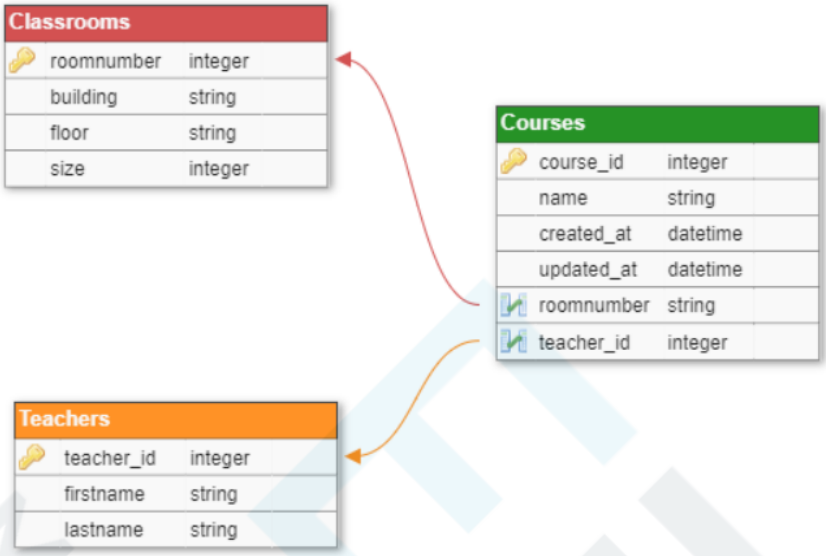
Databases use one or more tables to store information about entities. The database in our example above might have a table for qualifications, one for students and one for classrooms. In a relational database, these tables are linked together using relationships based on one or more common attributes.

Primary, Foreign and Composite Keys

A Primary Key is a special attribute in each table which can uniquely identify each entity in a table. Because each entity must have a unique primary key, this attribute is usually a numeric ID for example a student ID number, or book reference number. In some cases it may not be possible to create a unique primary key from a single attribute, in this case two attributes are used together to form what is known as a compound primary key.

The primary key is also used to manage the relationships between tables. In this case, the primary key from one table, can be stored in a second, related, table and is known as a foreign key.

The example database below shows this in action. There are three tables, classrooms (primary key sroomnumber), teachers (primary key teacher_id) and courses (primary key course_id). The courses table also contains the roomnumber attribute and the teacher_id attribute as foreign keys.



Database Normalisation

Normalising a database involves removing any redundant or repeated data held within it. This improves the efficiency of the database without compromising data. Normalising the database reduces the sizes of tables within the database, making searching and sorting data much quicker. It also makes the database easier to maintain by reducing data duplication and improving consistency. This reduces the potential for errors or anomalies to occur in the data.

First Normal Form (1NF)

Databases conforming to first normal form, sometimes written 1NF, must meet three rules:

1. Every record must have a primary key.
2. There must be no repeating groups of attributes.
3. The data in each column must be atomic. This means that the data cannot be sensibly split down any further.

The example below shows a basic database of teachers, to normalise this database to first normal form, we must check and apply the three rules above.

StaffID	Name	Subject	Room
001	John Smith	Maths	B1
002	Sally Richards	Geography History	A5
003	Amanda Jones	Maths	C2
004	Richard White	Maths English	D2

The StaffID field is the primary key and is present for all records, rule 1 is met. The Subject column contains repeating attributes, which means rule 2 is not met, we could fix this by creating a separate Subjects table with a relationship to the Teachers table. The name field contains both the first and last name, meaning the data is not atomic and rule 3 is not met, we could fix this by splitting this record into First Name and Last Name.

The example below shows the same database normalised to first normal form. We can see the name column has been split into FirstName and LastName and a new Subjects table created. The subjects table uses a composite primary key made of the SubjectID and room.

Teachers		
StaffID	**FirstName**	**LastName**
001	John	Smith
002	Sally	Richards
003	Amanda	Jones
004	Richard	White

Subjects			
SubjectID	**Room**	**Name**	**Teacher**
001	B1	Maths	001
002	A5	History	002
003	A5	Geography	002
004	D2	English	004
005	C2	Maths	003
006	D2	Maths	004

Second Normal Form (2NF)

Databases in second normal form must meet all the same rules as first normal form but must also have no partial key dependencies. A partial dependency happens when one a non-key column within a table is dependent on the value of one or more of the columns which make up a composite primary key.

This can only happen when a table uses a composite primary key, so we can see that the Teachers table in the example above is already in second normal form.

We must now check the Subjects table for dependencies, neither the name or teacher columns are dependent on the SubjectID or Room columns and so we can say that the database is in second normal form.

Third Normal Form (3NF)

Databases in third normal form must meet all the rules for first and second normal forms, in addition, there must be no non-key dependencies. This means that no column must depend on another column which is not the primary key for the table.

Structured Query Language (SQL)

SQL, short for Structured Query Language, is a programming language used to work with database. It is a declarative language, meaning it describes the required result rather than the process required to get there.

SQL SELECT Command

The SELECT command is used to retrieve data from the database, it uses the following form (The WHERE and ORDER BY parts are optional):

```
SELECT <attribute> FROM <table> WHERE <condition>
ORDER BY <ASC/DESC>
```

SELECT is an incredibly powerful and versatile command and can be used in many different ways. The example below will return all data held in the Customers table. The * is known as a wildcard, and means that all attributes are returned. Because we have not specified a condition using WHERE, all records are returned.

```
SELECT * FROM Customers
```

The example below will return only the CustomerID, FirstName and LastName fields from the Customers table. Only records where the FirstName column is equal to Bob will be returned. Notice how quote marks are used around the name Bob.

```
SELECT CustomerID,FirstName,LastName FROM Customers WHERE FirstName
='Bob'
```

The example below will return OrderID and Date from the Orders table, but will only return records where the OrderSize is greater than 100.

```
SELECT OrderID,Date FROM Orders WHERE OrderSize > 100
```

This example builds on the above, but adds ORDER BY, meaning the results will be displayed in descending order by data.

```
SELECT OrderID,Date FROM Orders WHERE OrderSize > 100 ORDER BY Date DESC
```

SQL UPDATE Command

The UPDATE command is used to change the attributes existing records in the table, it uses this form:

```
UPDATE <table> SET <attribute> = <value> WHERE
<attribute> = <value>
```

The UPDATE command should be used with care since it has the potential to change existing data. As an illustration of this point, the example below would change the FirstName of all records in the student table with an age greater than 10 to Jane. Clearly this would not be ideal!

```
UPDATE students SET FirstName = 'Jane' WHERE Age > 10
```

This example, would change the email of the student with StudentID 172666 to james@hotmail.com notice again how quote marks are used around the email address to indicate it is a string and not part of the code. Also notice how the quote marks are not needed around the StudentID as this is numeric data.

```
UPDATE students SET Email = 'james@hotmail.com' WHERE StudentID = 172666
```

This example sets the Status field in the orders table to Dispatched for the OrderID 8871

```
UPDATE orders SET Status = 'Dispatched' WHERE OrderID = 8871
```

SQL DELETE Command

The DELETE Command removes records from a table and takes this form:

```
DELETE FROM <table> WHERE <condition>
```

Once again, this command must be used with care since it removes data. This example would delete all records from the orders table where the Status is Received.

```
DELETE FROM orders WHERE Status = Received
```

This example would delete all records from the Students table where the Age column was greater than 20

```
DELETE FROM Students WHERE Age > 20
```

SQL INSERT Command

The INSERT command is used to add a new record into an existing database table. It can take one of two formats.

This format specifies both the column names and value names, this allows us to insert records where one or more column names are blank, or where the columns are not in the same order as they are in the table.

```
INSERT INTO <table> (<column1>, <column2>, …) VALUES
(<value1>, <value2>, …)
```

In this format, we specify just the values without the column names. This is a much shorter statement, but it requires that all values are specified for the record and they are specified in the correct order.

```
INSERT INTO <table> VALUES (<value1>, <value2>, …)
```

Defining Tables With SQL

The SQL Create command is a special command since it is used to create a new table in a database, it includes all the information needed to create the table such as its name, attributes and their data type along with identifiers such as the primary key. It uses this format

```
CREATE TABLE <name> (<attribute1> <format>,
<attribute2> <format>, PRIMARY KEY (<attribute1))
```

The example below would create a table called Students with FirstName, LastName, Age, DOB and StudentID Fields and set the StudentID Field as the primary key.

```
CREATE TABLE Students (FirstName VARCHAR(50), LastName VARCHAR(50), Age
INT(3), DOB DATE, StudentID INT(6), PRIMARY KEY (StudentID))
```

SQL Data Types

The example above includes a number of different data types, the table below shows the data types you need to be aware of in SQL.

Data type	SQL	Description
Fixed length string	CHAR(size)	A string with the number of characters specified by size
Variable length string	VARCHAR(size)	A string with any number of characters up to the number specified by size
Integer	INT(size)	A whole number stored using the number of bits specified by size
Number with fractional part	FLOAT(size, precision)	A number stored using the number of bits specified by size with digits after the decimal point up to the number specified by precision
Date	DATE	A date in the format YYYY-MM-DD
Date and time	DATETIME	A date and time combined in the format YYYY-MM-DD HH:MM:SS
Time	TIME	A time in the format HH:MM:SS
Year	YEAR	A year in one of the two formats YY or YYYY

Client Server Databases

Client server databases store the database on a central server, allowing several clients to access the database at once. This is incredibly useful since it allows, for example, several sales people to share a database of customers and to access and update it at the same time.

Client server databases have additional features to deal with the problem of concurrent access, which happens when two clients attempt to access or change the same field at the same time. If not well managed, concurrent access can result in data loss or corruption. Client server databases use one or more of the techniques below to avoid this

Record Locks

This technique places a lock on a record as soon as it is accessed by a client. The lock prevents any other client from accessing or modifying the record until the first client closes the record, removing the lock.

Serialisation

Instead of locking a record, this technique places requests from users into a queue, with the requests being executed in order. This means that if two clients attempt to modify the same record at the same time, the first client's changes are executed first, followed by the second client's changes.

Timestamp Ordering

This technique assigns a timestamp to all commands sent to the database, recording the time at which the command was initialised. Commands can then be carried out in the order they were initialised, oldest first.

Commitment Ordering

This technique uses an algorithm to work out the most efficient order to execute a number of commands on the same field. The algorithm considers several factors including the impact of commands on other tables, records and fields in order to prevent any issues from occurring. The commands are then executed in the order deemed most appropriate by the algorithm.