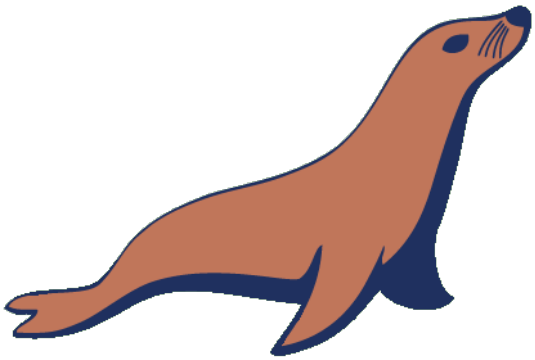
Data Modelling Tools

AUTM08016

Topic 2
Introduction to Databases



Dr Diarmuid Ó Briain
Version 1.0 [06 September 2023]

TUS
Ollscoil Teicneolaíochta na Sionainne:
Lár Tíre, An tIarthar Láir
Technological University of the Shannon:
Midlands Midwest

Dr Diarmuid Ó Briain

Table of Contents

Table of Figures

1. Introduction

In today's data-driven world, where organisations collect, store, and analyse vast amounts of information, databases and data modelling tools have emerged as essential tools for managing and extracting value from this data. Databases provide a structured repository for storing and organising data, while data modelling tools assist in designing and creating these databases efficiently and effectively. Studying databases and data modelling tools equips learners with the knowledge and skills necessary to design, implement, and maintain efficient and scalable data storage solutions that can power business operations, scientific research, and decision-making across various industries.

1.1 Objectives

At the end of this topic the learner will:

- Distinguish Data Modelling Tools
- Write simple File-Based Data Models
- Classify different database types
- List the 12 Rules of Relational Databases
- Discuss Fundamental Database Concepts - ACID
- List databases available on the market.

2. What are Data Modelling Tools?

Data modelling tools are software applications that assist in the creation, modification, and reverse engineering of data models. They are essential for the design and maintenance of efficient and accurate data storage and retrieval systems. Data models represent the structure and relationships between data entities, providing a clear visual representation of the data landscape.

2.1 Create, Edit, and View Data Models

These tools enable users to create and modify various types of data models, including conceptual, logical, and physical models. They provide a Graphical User Interface (GUI) for designing and editing data entities, relationships, and attributes.

2.2 Reverse Engineering

Data modelling tools can extract and analyse existing database structures to generate data models. This helps in understanding the current data layout and identifying potential issues or optimisations.

2.3 Code Generation

Some data modelling tools can generate Structured Query Language (SQL) scripts or code from data models. This simplifies the process of implementing data models in various databases.

2.4 Collaboration and Sharing

Data modelling tools often support collaboration features, allowing teams to work on the same models simultaneously and share their work effectively.

2.5 Documentation and Reporting

These tools can generate comprehensive documentation of data models, including diagrams, detailed descriptions, and traceability information.

2.6 Benefits of Using Data Modelling Tools

Accurate data models ensure that data is structured correctly and relationships are maintained, leading to improved data quality and integrity. Additionally, well-designed data models can optimise database performance, enabling efficient data storage, retrieval, and analysis. Data modelling tools can automate many tasks, saving time and effort in the data modelling process, ultimately reducing overall development costs. Clear data models facilitate better communication among stakeholders, promoting a shared understanding of the data landscape.

3. File-Based Data Models

File-based data models are a type of data modelling approach where data is stored in separate files, each with its own structure. This approach is often used for small to medium-sized applications where the data volume is relatively low. File-based data models are simple to implement and manage, but they can become inefficient as the data volume increases.

3.1 Key Characteristics of File-Based Data Models:

- **Data Separation**: Each file stores a distinct set of data, often related to a specific application or function.
- **Modular Approach**: Data is divided into independent modules, making it easier to maintain and update specific data areas.
- **Direct File Access**: Applications can directly access and manipulate data within their respective files, reducing reliance on centralised control.
- **Limited Metadata Management**: Metadata, such as data definitions and relationships, is typically managed within the application code rather than a centralised data repository.

File-Based Data Models are simple, easy to understand and implement, requiring minimal knowledge of data modelling concepts. They are flexible and adaptable to changing data requirements without major structural modifications and they are scalable as the can accommodate moderate data growth in the early stages of an application's lifecycle.

However, they are also inefficient with performance degrading as data volume increases due to lack of centralised data management and indexing. Data duplication can easily occur if multiple applications store similar data independently and they offer data consistency challenges to the maintenance of data consistency across multiple files. Additionally, they offer limited data analysis as complex data analysis queries can be inefficient and difficult to implement due to the distributed data structure.

3.2 Example File-based Data Model

An example of a file-based data model for an order management system is illustrated in Figure 1. This file stores data about orders placed by customers. Each row in the file represents a single order, and the columns represent the order ID, customer ID, product ID, quantity, and price.

```
Order ID|Customer ID|Product ID|Quantity|Price
---------|------------|------------|--------|--------
1001|1234|5678|10|10.00
1002|4321|7654|20|25.00
1003|9876|3456|30|32.50
```

Figure 1: orders.txt

The file in Figure 2 stores data about customers who have placed orders. Each row in the file represents a single customer, and the columns represent the customer ID, name, address, and phone number.

```
Customer ID|Name|Address|Phone Number
-----------|--------|------------|---------
1234|John Ryan|3 Mulgrave Street|087-456-7890
4321|Tomas Smith|21 Sarsfield Street|086-678-9012
9876|Peter Gleeson|35 Main Street|087-890-1234
```

Figure 2: customers.txt

This file in Figure 3 stores data about products that are available for purchase. Each row in the file represents a single product, and the columns represent the product ID, name, description, and price.

```
Product ID|Name|Description|Price
---------|--------|------------|--------
1|Laptop|A powerful laptop for work and play.|1000.00
2|Phone|A high-end smartphone with a great camera.|500.00
3|TV|A 4K Ultra HD TV with HDR.|1200.00
```

Figure 3: products.txt

This example shows how a file-based data model can be used to store data about orders, customers, and products. However, this approach is not as efficient as a relational database for larger applications with more complex data relationships.

3.3 Summary

In summary, file-based data models are suitable for simple applications with low data volumes and limited data sharing needs. However, as data complexity and volume grow, file-based models become inefficient and may hinder data analysis and reporting capabilities. More sophisticated data modelling techniques, such as relational databases, are typically recommended for larger, enterprise-level applications.

4. Introduction to Databases

4.1 What Is a Database?

A database, in the most general sense, is an organised collection of data. More specifically, a database is an electronic system that allows data to be easily accessed, manipulated and updated.

Databases are the backbone of modern business efficiency. They provide a structured and controlled way to store, manage, and retrieve information, enabling organisations to harness the power of data for improved decision-making and innovation.

Databases extend beyond mere data storage; they are strategic business tools that empower organisations to extract meaningful insights and gain competitive advantages. By effectively utilising databases, businesses can optimise operations, identify market trends, and drive growth.

4.2 History of Databases

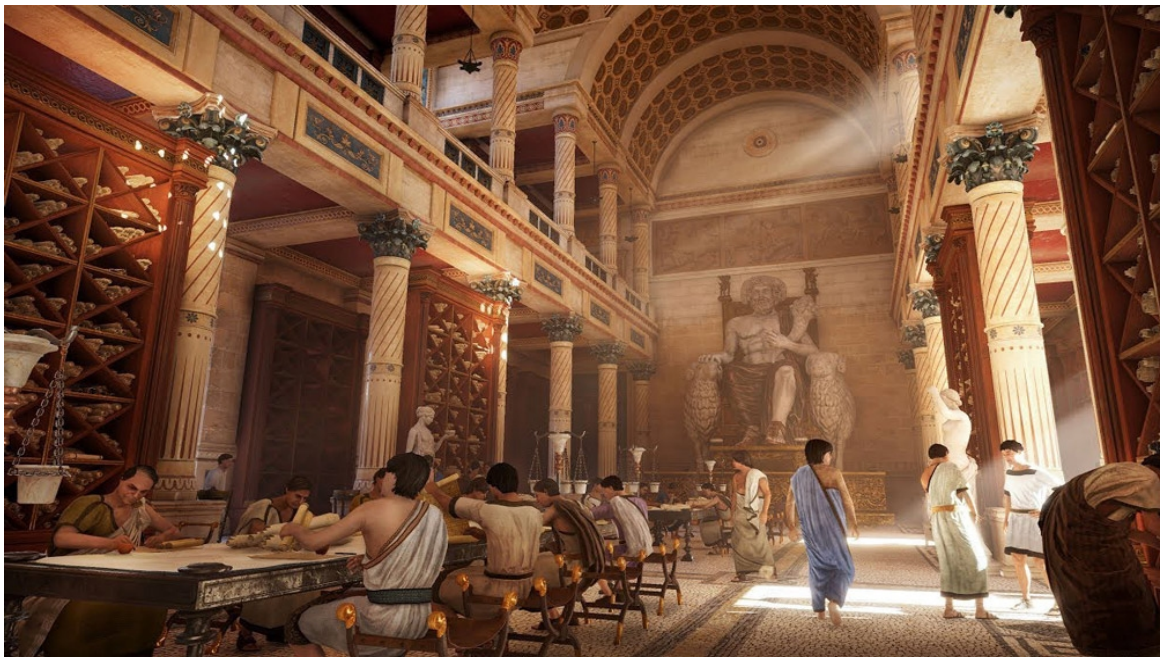


Figure 4: Library of Alexandria

Databases are not a recent invention. They have been around for centuries, from the ancient Egyptians to the Library of Alexandria, established during the reign of Ptolemy II Philadelphus (285–246 BC). However, the computerisation of databases in the 1960s has revolutionised their capabilities.

Early databases were navigational, relying on pointers to navigate through data. However, these were inefficient and limited in their ability to handle complex data relationships. The relational model, introduced by Edgar Frank "Ted" Codd, provided a more structured and efficient way to store and retrieve data [1].

Relational databases are now the standard for modern applications. They are powerful, efficient, and easy to use.

4.3 Edgar Codd and the Relational Model

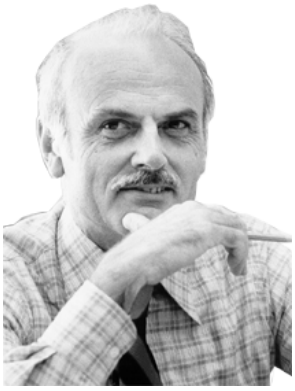


Figure 5: Edgar Codd

Edgar Codd's relational model, published in 1970, introduced a new approach to storing and retrieving data. Instead of relying on hierarchical or network models, which were based on pointers and linked structures, the relational model organises data into tables with rows and columns. This makes it much easier to query and manipulate data, and it paved the way for the development of powerful and efficient relational databases.

Codd's work was initially met with resistance from IBM, which was heavily invested in its own hierarchical database model, Information Management System (IMS). However, Codd's vision eventually prevailed, and relational databases became the standard for commercial applications.

In 1979, Larry Ellison founded Oracle Corporation and adopted Codd's relational model for his database product. Oracle Database (DB) quickly became the most popular relational database.

Figure 6: Larry Ellison

4.4 The Relational Database

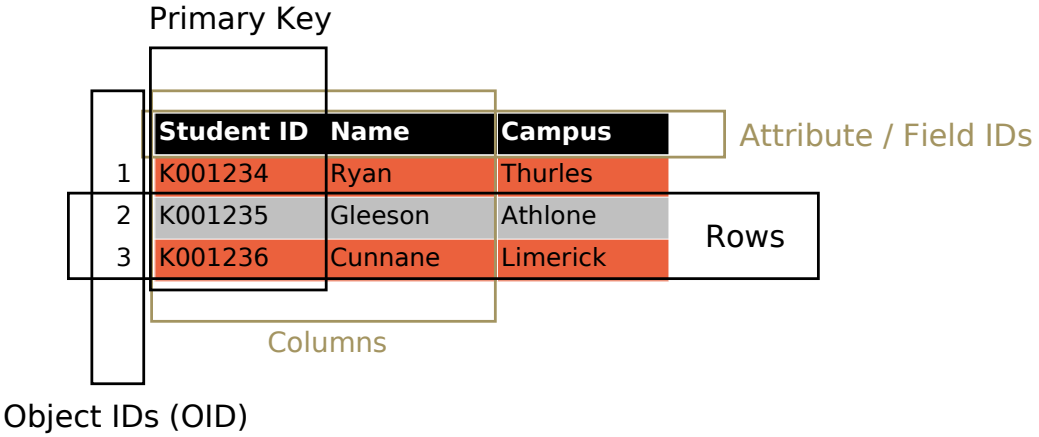


Figure 7: The anatomy of a RDBMS table

A relational database is a collection of tables that store data in a structured and organised way. Each table has rows and columns, and the tables are related to each other by primary and foreign keys.

Referential integrity is a rule that ensures that the relationships between tables are always consistent. This means that any changes to data in one table must also be reflected in the tables that relate to it. For example, if a student is deleted from the **`STUDENT_MASTER`** table, then that students accounts must also be deleted from the **`STUDENT_MASTER`** table. Referential integrity is important because it helps to prevent data from becoming corrupt. It also makes it easier to query and manipulate data.

The relational model is a powerful and efficient way to store and manage data. It is the standard for most modern database systems, and it is likely to remain so for many years to come.

4.4.1 Relational DB example

Consider the tables from a relational DB in Figures 8, 10, and 9. An order is placed by a single customer. This is a one-to-one relationship between the **`Orders`** and **`Customers`** tables. An order contains a single product, this is a one-to-one relationship between the **`Orders`** and **`Products`** tables. A product can be ordered multiple times. These are many-to-one relationships between the **`Orders`** and **`Products`** tables.

This relational database structure allows for efficient data storage and retrieval. For example, to find all orders placed by a particular customer, the **`Orders`** and **`Customers`** tables can be joined on the **`CustomerID`** field. Similarly, a join of the **`Orders`** and **`Products`** tables on both the **`CustomerID`** and **`ProductID`** fields will find all products ordered by a particular customer.

Column Name	Data Type	Description
OrderID	INT	Primary key, Unique ID for each order
CustomerID	INT	Foreign key referencing the CustomerID in the Customers table
ProductID	INT	Foreign key referencing the ProductID in the Products table
Quantity	INT	Quantity of the product ordered
Price	FLOAT	Price of the product

Figure 8: orders table

Column Name	Data Type	Description
CustomerID	INT	Primary key, Unique ID for each customer
Name	VARCHAR(50)	Customer's name
Address	VARCHAR(255)	Customer's address
PhoneNumber	VARCHAR(20)	Customer's phone number

Figure 9: Customers table

Column Name	Data Type	Description
ProductID	INT	Primary key, Unique ID for each product
Name	VARCHAR(50)	Product's name
Description	VARCHAR(255)	Product's description
Price	FLOAT	Price of Product

Figure 10: Customers table

As an example consider the output of the SQL query executed against the orders, in Figure 11, customers, in Figure 12, and products, in Figure 13, tables. SQL and SQL queries are the subject of future topics on this module.

```
SQL> select * from orders
| orderID | customerID | productID | orderDate | status |
| --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 2023-10-04 | pending |
| 2 | 2 | 2 | 2023-10-05 | processing |
| 3 | 3 | 3 | 2023-10-06 | delivered |
```

Figure 11: orders query

```
SQL> select * from customers
| customerID | customerName | address | email | phoneNo |
| --- | --- | --- | --- | --- |
| 1 | John Ryan | 3 Mulgrave St | john@gmail.com | 087-456-7890 |
| 2 | Thomas Smith | 21 Sarsfield St | tom@micro.org | 086-678-9012 |
| 3 | Peter Gleeson | 35 Main St | peter@itservice.com | 087-890-1234 |
```

Figure 12: customers query

```
SQL> select * from products
| productID | productName | price | description |
| --- | --- | --- | --- |
| 1 | Laptop | €1,000 | A powerful laptop for work and play. |
| 2 | Phone | €500 | A high-end smartphone with a great camera. |
| 3 | TV | €1200 | A 4K Ultra HD TV with HDR. |
```

Figure 13: products query

5. Other Databases

5.1 Spreadsheets

Spreadsheets are often touted as a cost-effective alternative to databases, but they have a number of limitations that make them unsuitable for managing some data situations. Here are some of the key differences between databases and spreadsheets:

- Spreadsheets are not designed for multi-user access, and this can lead to conflicts and data corruption. Databases, on the other hand, can handle multi-user access with ease, allowing multiple users to access and modify data simultaneously without interfering with each other.
- Spreadsheets are not very good at enforcing data validation and integrity rules. This means that it is easy for users to make mistakes or enter invalid data. Databases, on the other hand, can enforce data validation and integrity rules, ensuring that the data is accurate and consistent.
- Spreadsheets can be used to query and report on data, but they are not as powerful as databases. Databases can be used to run complex queries, join multiple tables, and create advanced reports.
- Spreadsheets are not scalable, and they can struggle to handle large amounts of data. Databases, on the other hand, are scalable and can handle millions of rows of data with ease.
- Spreadsheets are generally easier to use than databases. However, databases can be more powerful and efficient for large-scale data management.

In general, spreadsheets are a good choice for small-scale data management or for tasks that do not require a lot of data integrity or security. However, databases are a better choice for large-scale data management, multi-user access, and complex data analysis.

5.2 Non-Relational Databases

A non-relational database (NoSQL) database does not store data in a traditional relational format. Instead, non-relational databases store data in other ways, such as key-value pairs, documents, or graphs. This makes them more flexible and scalable than relational databases, which can be a good thing for certain types of applications.

MongoDB is a cross-platform, document-oriented database management system (DBMS) that stores data in JavaScriptObject Notation (JSON)-like documents. It is a NoSQL database, meaning it does not adhere to the strict schema requirements of traditional relational databases. Instead, MongoDB stores data in flexible, self-describing JSON documents, enabling efficient storage and retrieval of complex data structures.

Example of data access in MongoDB. The example in Figure 14 illustrates a simple python program that will retrieve all documents from a MongoDB database, called **`mydatabase`**, where the products have a price field greater than 100. The **`$gt`** operator is used to perform a greater-than comparison. Additionally the program gets all products with a status of pending, in this case for customer numbers 1, 2, and 3.

The returned data for both queries is in JSON format. JSON is a lightweight data-interchange format that uses key-value pairs to represent data. Each document in the returned results is represented as a JSON object, with each field of the document corresponding to a key-value pair. The **`_id`** field is a unique identifier for each document, and the other fields are the data stored for that document.

```
~$ cat mongodb_example.py
 1  #! /usr/bin/env python3
 2
 3  import pymongo
 4
 5  # Connect to the MongoDB database
 6  client = pymongo.MongoClient()
 7  db = client['mydatabase']
 8
 9  # Get the products collection
10  products = db['products']
11
12  # Find all products with a price greater than 100
13  products_with_high_price = products.find({'price': {'$gt':100}})
14
15  for product in products_with_high_price:
16      print(product)
17
18  # Find all orders that are pending
19  orders_pending = db['orders'].find({'status':'pending'})
20
21  for order in orders_pending:
22      print(order)
23


~$ ./mongodb_example.py
{'_id': 1, 'productName': 'Laptop', 'price': 1000, 'description': 'A
powerful laptop for work and play.'}

{'_id': 3, 'productName': 'TV', 'price': 700, 'description': 'A 4K
Ultra HD TV with HDR.'}

{'_id':29,'customerID':3,'productID':3,'orderDate':2023-10-
06,'status':'pending'}

{'_id':30,'customerID':2,'productID':2,'orderDate':2023-10-
05,'status':'pending'}

{'_id':31,'customerID':1,'productID':1,'orderDate':2023-10-
04,'status':'pending'}
```

Figure 14: Accessing data in MongoDB

The table in summarised the key differences between relational and non-relational databases:

Feature	Relational database	Non-relational database
Data storage	Tables	Key-value pairs, documents, graphs
Data modelling	Stricter	More flexible
Scalability	Less scalable	More scalable
Use cases	Enterprise applications, transactional systems	Web-based applications, cloud computing, social networking

Non-relational databases have the following benefits:

- **Flexible**: Non-relational databases can store data in many different ways, which makes them more flexible than relational databases. This can be a good thing for applications with complex data structures.
- **Scalable**: Non-relational databases can be scaled horizontally, which means that you can add more servers to handle more traffic. This can be a good thing for applications that need to handle a lot of data.
- **Performance**: Non-relational databases can perform better than relational databases for certain types of applications, such as those that require a lot of reads.

However, there are drawbacks of non-relational databases too. For example non-relational databases are not as good at enforcing data integrity as relational databases.

They also tend to be more complex to use than relational databases.

Overall, non-relational databases are a good choice for applications that need to store a lot of data, are highly scalable, and require flexibility. However, they are not a good choice for applications that require a lot of data integrity or are very complex.

5.3 Summary

	Spreadsheets	Non-relational databases	Relational databases	Data warehouses
Data storage	Flat files	Key-value pairs	Tables	Tables
Data integrity	Limited	Weaker	Stronger	Stronger
Scalability	Not scalable	Highly scalable	Less scalable	Highly scalable
Use cases	Personal data management, basic calculations	Web-based applications, cloud computing, social networking	Enterprise applications, transactional systems	Business intelligence, data analysis, trend forecasting

6. The 12 Rules of Relational Databases

Edgar Codd developed a list of 12 rules, or criteria, that determined whether a database could be called “relational” or not.

Codd's 12 rules, also known as the 12 commandments of relational databases, are a set of guidelines that define what makes a database truly relational [1].

6.1.1 Rule 0: Foundation Rule

“*A Relational Database Management System (RDBMS) must manage its stored data using only its relational capabilities*”

Essentially this rule states that an RDBMS should not rely on any non-relational features, such as hierarchical or network structures. The data in a relational database should be stored and manipulated in a purely relational manner.

6.1.2 Rule 1: Information Rule

“*All information in the database should be represented in one and only one way – as values in a table*”

This rule states that data in a relational database should be normalised, meaning that it should be stored in a way that minimises redundancy and inconsistencies. Each piece of information should be represented once in the database, and there should be a single place to find it.

6.1.3 Rule 2: Guaranteed Access Rule

“*Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name*”

This rule states that any piece of data in a relational database should be able to be retrieved using its primary key value. The primary key is a unique identifier for each row in a table.

6.1.4 Rule 3: Systematic Treatment of Null Values

“*Null values (which are distinct from empty character strings, strings of blank characters, zeros or any other number) are supported in the fully relational DBMS for representing missing information in a systematic way that is independent of data type*”

This rule states that a relational database should support null values, which are used to represent missing or unknown data. Null values should be treated as a distinct data type, separate from other data types such as numbers and strings.

6.1.5 Rule 4: Dynamic Online Catalogue Based on the Relational Model

"*The database description is represented at the logical level in the same way as ordinary data, so authorised users can apply the same relational language to its interrogation as they apply to regular data*"

This rule states that the metadata for a relational database should be stored in the database itself, and that this metadata should be accessible using the same relational language as the data itself. This allows users to query and update the metadata without having to use a separate tool.

6.1.6 Rule 5: Comprehensive Data Sublanguage Rule

"*A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible:*

- *Data definition*
- *View definition*
- *Data manipulation (interactive and by program)*
- *Integrity constraints*
- *Authorisation*
- *Transaction boundaries (begin, commit and rollback)"*

This rule states that a relational database should support a powerful data sublanguage that can be used to perform all of the following tasks:

- Create and modify tables
- Create and modify views
- Insert, update, and delete data
- Define and enforce integrity constraints
- Grant and revoke privileges
- Manage transactions

6.1.7 Rule 6: View Updating Rule

"*All views that are theoretically up-dateable are also up-dateable by the system*"

This rule states that views should be updated using the same relational operations as base tables. This means that users should be able to update views without having to worry about the underlying data structure of the database.

6.1.8 Rule 7: High-Level Insert, Update and Delete

"*The ability to handle a base relation or a derived relation as a single operand applies not only to the retrieval of data, but also to the insertion, update and deletion of data*"

This rule states that the data manipulation language of a relational database should allow users to perform operations on base tables and derived tables in the same way. This means that users should be able to insert, update, and delete data from derived tables without having to explicitly specify the underlying base tables.

6.1.9 Rule 8: Physical Data Independence

"*Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods*"

This rule states that changes to the physical storage of data or the access methods used to retrieve and manipulate data should not affect the logical structure of the database. This means that application programs should not have to be rewritten or recompiled whenever the physical implementation of the database changes.

For example, suppose a relational database is initially stored on magnetic tape. Later, the database is moved to a disk-based system. Application programs that access the database should not need to be changed to work with the disk-based system.

6.1.10 Rule 9: Logical Data Independence

"*Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind – and those that theoretically permit unimpairment – are made to the base tables*"

This rule states that changes to the logical structure of the database, such as the addition or deletion of columns, should not affect the logical structure of the database. This means that application programs should not have to be rewritten or recompiled whenever the logical schema of the database changes.

6.1.11 Rule 10: Integrity Independence

"*Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalogue, not in the application programs*"

This rule states that integrity constraints, which are rules that enforce the consistency and validity of data in a database, should be defined in the relational database language and stored in the database catalogue, not in the application programs. This means that application programs can access and manipulate data without having to know about the specific integrity constraints that are in place.

This rule is important because it ensures that integrity constraints are consistently enforced across all applications that access the database. It also makes it easier to

modify the integrity constraints as the database schema evolves, without having to modify all of the application programs.

For example, if a database has an integrity constraint that states that a customer's name must be unique, this constraint should be defined in the relational database language and stored in the database catalogue. This means that any application that attempts to insert a new customer record with a duplicate name will be prevented from doing so.

This rule is often cited as one of the most important of Codd's 12 rules because it makes it possible to create databases that are more flexible, reliable, and maintainable.

6.1.12 Rule 11: Distribution Independence

"*The data manipulation sub-language of a relational DBMS must enable application programs and terminal activities to remain logically unimpaired whether and whenever data are physically centralised or distributed*"

This rule states that the Data Manipulation Language (DML) of an RDBMS should be able to handle distributed data without application programs being aware of the distribution. This means that application programs should be able to work with data as if it were all stored in one location, even if it is actually spread across multiple locations.

For example, an application that stores customer data could be distributed across multiple servers, with each server containing a subset of the data. The DML of the RDBMS should be able to handle this distribution without the application programs being aware of it.

6.1.13 Rule 12: Non-subversion

"*If a relational system has or supports a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity rules or constraints expressed in the higher-level (multiple-records-at-a-time) relational language*"

This rule ensures that the higher-level DML, which defines the overall structure and behaviour of the database, is the authoritative source for integrity rules. Any low-level DML that attempts to override these rules must be prevented from doing so.

7. Fundamental Database Concepts

Key Database Concepts and Data Objects

This following section lists key DB concepts and data objects. These are essential for managing and manipulating data in relational databases.

- **Tables**: are the basic unit of data storage in a relational database. Each table consists of rows and columns. Rows represent individual items of data, while columns represent the attributes of those items.
- **Rows**: Rows, also known as records, represent individual items of data in a table. Each row in a table must have a unique primary key. The primary key is a column or combination of columns that uniquely identifies each row in a table.
- **Columns**: Columns are the attributes of data in a table. They define the specific values that can be stored in each column. Each column must have a unique name and a data type. The most common data types are numbers, strings, dates, and Boolean values.

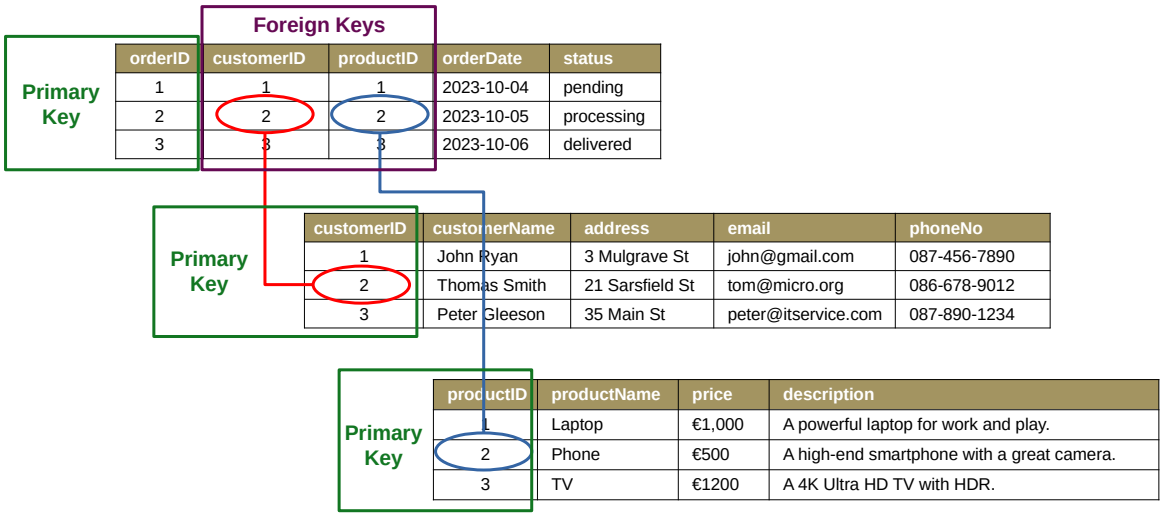


Figure 15: Relationships between tables

- **Relationships**: are the foundation of relational databases. They allow for the linking of data from different tables together. In order to link tables together, a foreign key is required. A foreign key is a column in one table that references the primary key of another table. This allows for the identification of relationships between different pieces of data.
- **Primary Key**: The primary key is a column or combination of columns that uniquely identifies each row in a table. It must be unique for each row, and must not contain any null values. The primary key is used to link tables together and to ensure the integrity of the data.
- **Foreign Key**: A foreign key is a column in one table that references the primary key of another table. The purpose of a foreign key is to link data from different tables together. For example, a table called **`Customer`** and another table called **`Orders`**. The **`Customer`** table would have a primary key called **`CustomerID`**, and the **`Orders`** table would have a foreign key called **`CustomerID`**. This allows for the linking of the customer data to the order data.

```
SQL> SELECT
  o.orderID,
  c.customerName,
  p.productName,
  o.orderDate,
  o.status
FROM orders o
JOIN customers c ON c.customerID = o.customerID
JOIN products p ON p.productID = o.productID;

+---------+---------------+-------------+------------+------------+
| orderID | customerName  | productName | orderDate  |   status   |
+---------+---------------+-------------+------------+------------+
|    1    | John Ryan     | Laptop      | 2023-10-04 | pending    |
|    2    | Thomas Smith  | Phone       | 2023-10-05 | processing |
|    3    | Peter Gleeson | TV          | 2023-10-06 | delivered  |
+---------+---------------+-------------+------------+------------+
```

Figure 16: SQL Query

Note: o = orders table, c = customers table p = products table

- **Structured Query Language (SQL)**: SQL is the standard language for managing and manipulating data in relational databases. It can be used to query, insert, update, and delete data. All major relational databases support SQL, which makes it easy for Database Administrators (DBA) to manage data across different platforms.
  - The query, in Figure 16, will join the **orders**, **customers**, and **products** tables using the **`customerID`** and **`productID`** columns, respectively. This will create a single table that contains all of the relevant information from each table, including the **`orderID`**, **`customerName`**, **`productName`**, **`orderDate`**, and order **`status.`**
  - This output shows that John Ryan has placed an order for a laptop, Thomas Smith has placed an order for a phone, and Peter Gleeson has placed an order for a TV. The order status for John Ryan's order is pending, the order status for Thomas Smith's order is processing, and the order status for Peter Gleeson's order is delivered.
- **Indexes**: are an RDBMS is a data structure that works closely with tables and columns to speed up data retrieval operations. It works a lot like a book index. It provides a reference point that allows a user to quickly find and access the data without having to traverse the entire database.
- **Schema**: is the structure behind data organisation. It is a visual overview of how different tables are related to each other. This serves to map out and implement the underlying business rules for which the database is created.

- **Normalisation**: is the process of organising data in a database so that it meets two basic requirements:
  - There is no data redundancy (all data is stored in only one place),
  - Data dependencies are logical (all related data items are stored together).
  - For instance, for a bank's database all customer static data, such as name, address and age, should be stored together. All account information, such as account holder, account type, account branch and so on, should also be stored together; it should also be stored separately from the customer static data.
- **Constraint**: is a restriction on the type of data to be inputted into a certain column. Constraints are always defined on columns. A common constraint is the not-null constraint. It simply specifies that all rows in a table must have a value in the column defined as not null.
- **Transactions (Commits and Rollbacks)**: All items in a series of changes need to be made together. In the case of a simple transfer, if you debit one account, you need to credit anther account.
- **Locking**: facilitate the database to allow multiple users to simultaneous access a set of data. In situations in which two or more users want to access or update the same piece of data. RDBMSs uses locks to isolate transactions. There are different types of locks, such as transactional or data-level locks that simply lock down a single data field; row-level locks that lock down an entire record of data, while table-level locks restrict access to a whole table.

7.1 Atomicity, Consistency, Isolation, Durability (ACID)

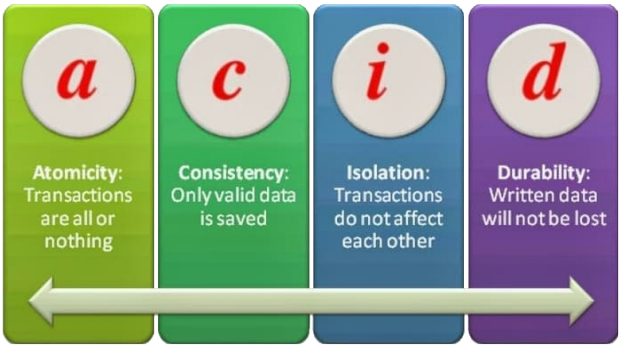


Figure 17: ACID properties of a database

Four highly desirable properties of any RDBMS are:

- **Atomicity**: This refers to a database's ability to either fully process or fully roll back a transaction.
- **Consistency**: The database should ensure that all data written therein follows all rules and constraints specified in the database.
- **Isolation**: Transactions must be processed securely and independently, without interfering with each other.
- **Durability**: The database must ensure that all committed transactions are saved permanently and cannot be accidentally erased, even in a database crash.

8. Databases available on the market

Figure 18: RDBMS on the market

Here is a look at some RDBMS offerings and the companies or organisations behind them:

8.1 Oracle

Oracle is a leading provider of enterprise-level databases, middleware, enterprise resource planning (ERP) systems, and customer relationship management (CRM) offerings. It is the top dog in the RDBMS market with a market share of 48.8%. Oracle DB is a widely used enterprise-level database that comes in different editions to meet different needs. It is fully compliant with the SQL language and also maintains its proprietary version called SQL*Plus. Oracle acquired Sun Microsystems in 2009, which gave it the license holder of MySQL, one of its key competitors. As a result, Oracle now has two RDBMS offerings, but it's unlikely that they will interfere destructively with each other.

8.2 MySQL

MySQL is a RDBMS that is one of the most popular and widely used databases in the world. It is open-source, which means that it is free to use and modify. MySQL is also very scalable, which means that it can be used to store and manage large amounts of data. It is also a reliable database that is known for its stability and performance. As a result, MySQL is a popular choice for a variety of applications, including web applications, e-commerce applications, and enterprise applications.

Some of the key features of MySQL is that it is opensource, and therefore free to use and modify. It is scalable, reliable, easy to use and there is a large community of users and developers with plenty of resources available to help users.

Overall, MySQL is a powerful, versatile, and easy-to-use database that is a popular choice for a variety of applications.

8.3 MariaDB

MariaDB is a fork of MySQL that was created by former MySQL developers. MariaDB is a community-developed project that is committed to providing a freely available, high-performance, scalable, and reliable alternative to MySQL. MariaDB is fully compatible with MySQL, and it can be used as a drop-in replacement for MySQL in most applications. MariaDB also includes a number of new features that are not available in MySQL, such as row-level locking and support for foreign keys. MariaDB is a popular choice for both small and large organisations, and it is one of the fastest-growing open-source database projects in the world.

8.4 Postgres

PostgreSQL is an open-source, Object-Relational Database Management System (ORDBMS). It is a highly scalable and flexible database that is well-suited for a variety of applications, including web applications, e-commerce applications, and enterprise applications. PostgreSQL is fully compliant with the SQL language and supports a wide range of advanced features, such as stored procedures, triggers, and user-defined functions. It is also a highly reliable database that is known for its stability and performance. PostgreSQL is a popular choice for both large and small organisations, and it is one of the most widely used ORDBMSs in the world.

8.5 IBM DB2

IBM DB2, a RDBMS developed and published by IBM, holds a prominent position among the world's most popular and widely used database systems. It is renowned for its scalability, performance, and data integrity features, making it particularly suitable for demanding enterprise-level applications. DB2 comes equipped with a comprehensive range of capabilities, encompassing data management, data mining, and business intelligence functions. Its high availability and security features make it an ideal choice for mission-critical applications requiring 24/7 uptime and data protection.

IBM DB2, a relational database management system (RDBMS) developed and published by IBM, holds a prominent position among the world's most popular and widely used database systems. It is renowned for its scalability, performance, and data integrity features, making it particularly suitable for demanding enterprise-level applications. DB2 comes equipped with a comprehensive range of capabilities, encompassing data management, data mining, and business intelligence functions. Its high availability and security features make it an ideal choice for mission-critical applications requiring 24/7 uptime and data protection.

8.6 Microsoft

Microsoft is another player in the RDBMS market, with SQL Server as its flagship database product. SQL Server is widely used in enterprise-level databases and comes in different editions to meet different needs. It is fully compliant with the SQL language and also maintains its proprietary version called T-SQL. SQL Server is consistently ranked second in the RDBMS market, with a market share of 33.7%. Microsoft acquired Sybase, another major RDBMS vendor, in 2001, which gave it access to Sybase's database technologies and expertise. As a result, Microsoft now has two RDBMS offerings, but it's unlikely that they will interfere destructively with each other, as they play in slightly different market spaces and cater to slightly different needs.

9. Laboratory #1 - Building Data Models

Objective: To practice creating file-based and spreadsheet-based data models for storing student information.

Materials:

- Text editor
- Spreadsheet software (e.g., LibreOffice Sheets, Microsoft Excel, Google Sheets)
- Sample student data

Instructions:

1. **Create the main data file**:
   a) Create a text file named '**`students.txt`**'.
   b) Define the following fields for each student:
      - Student ID: A unique identifier for each student
      - Given Name: The student's first name
      - Family Name: The student's last name
      - Country of Origin: The student's country of origin
      - Programme: The student's program of study
   c) Add sample student data to the file. For example:

```
Student_ID,Given_Name,Family_Name,Country_Origin,Programme
K0012345,Alice,Murphy,Uganda,Computer Engineering
K0054321,John,Ryan,Scotland,Robotics & Automation
```

2. **Create the sports data file**:
   a) Create a text file named '**`sports.txt`**'.
   b) Define the following fields for each student's sports activities:
      - Student ID: A reference to the student's ID in the main data file
      - Sport Played: The name of the sport the student plays
      - Sport Watched: The name of the sport the student watches
   c) Add sample data to the file. For example:

```
Student_ID,Sport_Played,Sport_Watched
K0012345,Rugby,Hurling
K0054321,Handball,Football
```

3. **Create the academic data file**:
   a) Create a text file named '**`academics.txt`**'.
   b) Define the following fields for each student's academic courses:
      - Student ID: A reference to the student's ID in the main data file
      - Course Code: The code of the course
      - Course Title: The title of the course
      - Semester: The semester the course was taken
      - Grade: The student's grade in the course
   c) Add sample data to the file. For example:

```
Student_ID,Course_Code,Course_Title,Semester,Grade
K0012345,AUTM08017,Object Oriented Programming,CY23-24 S1,72
K0054321,AUTM08016,Data Modelling Tools,CY23-24 S2,65
```

4. **Convert the data files to spreadsheets**:
   a) Open each data file in a spreadsheet software.
   b) Format the data appropriately, including headers, alignment, and formatting.

5. **Perform data analysis**: Use the spreadsheet software's functionality to perform basic data analysis tasks, such as:
   a) Finding the most popular sport among students.
   b) Calculating the average grade for each programme.
   c) Identifying the most challenging courses for students.

6. **Reflection**: Compare and contrast the file-based and spreadsheet-based data models:
   a) Which data model was easier to create?
   b) Which data model is more suitable for data analysis?
   c) What are the advantages and disadvantages of each data model?
   d) Discuss the limitations of using text files and spreadsheets for storing and managing large amounts of data.
   e) What field is the most appropriate to connect data from one file to another file?
   f) What ASCII character can you not use in the fields within the files and why?
   g) Suggest how this problem could be circumnavigated?