What is a database?

A database is a collection of data that can be searched in a systematic way to maintain and retrieve information A database can either be paper based or computerized.   
In a computerized database, the database exist in digital form, so it is a computer-based record keeping system

Characteristics of a database

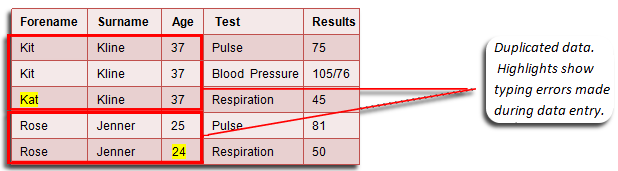
* A database has related data because the individual items of data have a connection
* A database is **logical** because it uses a clear and structured design in storing data
* A database is **ordered** because the data is stored in a very structured way, using **tables**, **records** and **fields** so that users and data handling applications can easily add, delete, edit, search and manipulate the data.

Flat-file Database Model

The flat model is the earliest, simplest data model. It simply lists all the data in a **single table**, consisting of columns and rows.

**Characteristics**

* **Data redundancy** – the needless and unnecessary duplication of data
* **Data inconsistency** – errors in the values or format of data that should be identical
* **Access inefficient:** In order to access or manipulate the data, the computer has to read the entire flat file into memory
* **Simple database:** Can only be used for small data set

Example of a flat file database  
A doctor wishes to use a database to keep track of his patients’ test results. It needs to record:

* + patient name
  + patient age
  + test type
  + test results.

**Relational Database Model**

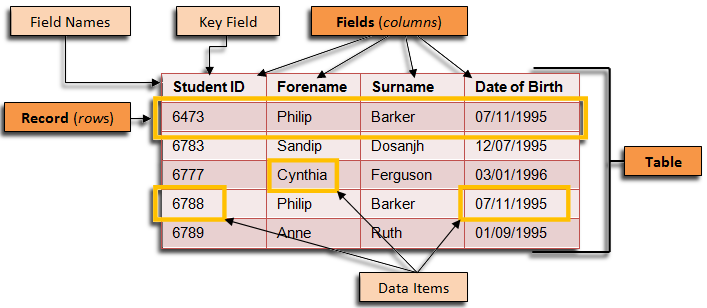
Relational Databases use two or more tables linked together (to form a relationship). Relational Databases do not store all the data in the same table. Repeated data is moved into its own table

Relational databases are designed to address the limitations of a file-based approach by providing a structured and efficient way to store, retrieve, and manipulate data.

Some key features of relational databases that overcome the limitations of file-based systems:

* Relational databases organize data into tables, consisting of rows and columns, providing a structured format. This structure allows for easy categorization and organization of data, making it more accessible and manageable.
* Enforces data integrity through the use of constraints such as primary keys, foreign keys, and data types. These constraints ensure that data is accurate, consistent, and valid, preventing duplicate or inconsistent entries.
* Built on the relational model, which enables the establishment of relationships between tables using primary and foreign keys. This allows for the representation of complex relationships among different entities, facilitating efficient data retrieval and analysis.
* Provide a powerful query language, (such as SQL), which allows users to retrieve, filter, sort, and aggregate data based on specific criteria. This enables complex data retrieval operations and supports the generation of customized reports.
* Provide mechanisms for data security and access control. User authentication, authorization, and role-based permissions ensure that only authorized individuals can access and modify the data. Additionally, integrity capabilities ensure that changes to the database are performed reliably and consistently.
* Designed to handle large volumes of data and support high-performance operations. Indexing and optimization techniques are employed to speed up data retrieval operations, and the ability to scale vertically (adding more resources to a single server) or horizontally (distributing data across multiple servers) allows for increased performance and capacity.
* Provide a level of abstraction between the physical representation of data and the applications that use it. This allows for data independence, meaning that the structure of the database can be modified without affecting the applications that rely on it, providing flexibility and easier maintenance.
* Offer features for data backup and recovery, ensuring that data can be restored in case of hardware failures, software errors, or other contingencies. Regular backups and transaction logs enable point-in-time recovery and data restoration to a consistent state.

**Components of a Relational Database**

Data is stored in **TABLES**, which consists of many **RECORDS** and each record consists of several **FIELDS** which has many different **DATA TYPES**.

A Database Management System (DBMS) allows us to create a relational database using the following database elements

* **Diagram, table

  Description automatically generatedObject/entity**- An entity is used to represent an object in the real world that can be distinguished from other objects. An entity can be a physical object (such as a person or a place) or a concept (such as an activity or a task) for which we need to record data in the database. For example, a physical object could be an employee, a customer or a product, and a concept could be an online order, a school course or a booking.
* **Tuples** – A single entry in a table. It is also called a record or a row
* **Attributes**- An attribute is used to represent a property, a quality or a characteristic that describes an entity. For example, the name of an employee, or the date and time that a booking was submitted. The column are called attributes, which describes each tuple. Also refers to as fields
* **Index** is a data structure technique which allows you to quickly retrieve records from a database file.

Advantages of relational databases?

* Reduce data redundancy
* Reduce updating
* Data inconsistency – as data is not duplicated, there is no risk of the same data item being stored differently in another record.
* Greater data integrity
* Interdependence from other application programs
* Improved data security
* Reduce data entry, storage and retrieval cost
* Facilitate development of new application program
* Flexibility – A relational database can be queried with greater flexibility than a flat-file system.

Disadvantages

* Database systems are complex, difficult and time consuming to design
* Substantial hardware and software start up cost
* Damage to database affects virtually all application program
* Extensive conversion cost in moving from a flat file database model to relational database model
* Initial training required for all programmers and users

**How do you form the relationship? (Link the tables)**

Relational Databases use two or more tables linked together (to form a relationship) because it does not store all the data in the same table. Repeated data is moved into its own table. The relationship between the tables is formed by using a common field which appears in BOTH tables.   
Database keys are used in establishing a relationship between these tables.

What is a database key?  
A database key in the relational database is a normal attribute(field) or group of attributes that can be used to **uniquely identify** or sort records(tuple) in a table(relation).

**Types of database keys**

**Primary Key (PK)**

The primary key is the most important key in the database. There can be only one primary key in a table. It will not accept duplicate or null values because it contains unique values  
It is a column(s) within a relational database table that uniquely represents each record in the table. For example, the ideal primary key for a table of students would be their ID number, as this would uniquely identify each student in the table.

*What Is the Purpose of the Primary Key (PK)?*

Each table within a database will have its own primary key. The main purpose of having a primary key is to **identify** each unique record in a particular table. This makes it far easier to search for any particular record in any given table. It also makes it much easier to identify a particular record that may have certain data in common with other records.

**Candidate/secondary/alternate Key**

A candidate key is any field in the table that could be used as a primary key because it is unique. It can be one or multiple columns in a database table. It can identify a record uniquely just like a primary key.

**Alternate key**

An alternate key is any of the candidate keys that are not used as the primary key.

**Foreign Key**

A foreign key is a column or set of columns in a table that refers to a primary key in another table. They are used to specify that a record in one table is related to that record in another table. It is also called a reference key.

*The foreign key definition is a column or set of columns that is used to refer to another table’s primary key.*

The main reason of having a foreign key is to create logical relationships between two tables. Primary keys are used to identify a record in a table. Foreign keys are used to relate one record to another record in a different table.

**Composite Key**

A composite key is a primary key, or unique identifier, that is made up or more than one attribute.

It is a combination of two or more columns in a table that can be used to uniquely identify each row in the table. Being unique can only be done when the columns are combined; If the column are taken individually by themselves then the columns are not unique.

Types of relationships

In a relational database, cardinality refers to the number of instances or occurrences of one entity that can be associated with the instances of another entity through a relationship. It describes the relationship between two tables or entities in terms of how many records can be related to each other.

Cardinality in a relational database helps define the nature and constraints of relationships between tables, providing a way to establish connections and maintain data integrity. By understanding and correctly representing cardinality, database designers can create efficient and structured databases that accurately reflect the real-world relationships between entities.

There are three types of cardinalities commonly used in a relational database:

**A picture containing waterfall chart

Description automatically generatedOne-to-One (1:1) Cardinality**

In a one-to-one relationship, each record in one table is related to exactly one record in another table, and vice versa. This means that for every record in Table A, there is only one corresponding record in Table B, and vice versa. One-to-one relationships are relatively rare and are typically used when the related data is truly distinct and separate.

For example, in a database for employee management, you might have a "Person" table and an "Address" table. Each person can have only one corresponding address, and each address can be associated with only one person.

**A picture containing waterfall chart

Description automatically generatedOne-to-Many (1:N) Cardinality**

In a one-to-many relationship, each record in one table can be related to multiple records in another table, but each record in the second table can only be related to one record in the first table. This is the most common type of relationship in a relational database.

For instance, consider a database for a school. The "Department" table may have a one-to-many relationship with the "Course" table, where each department can offer multiple courses, but each course is associated with only one department.

**Many-to-Many (N:N) Cardinality**

A picture containing waterfall chart

Description automatically generatedIn a many-to-many relationship, each record in one table can be related to multiple records in another table, and vice versa. This type of relationship requires the use of **an intermediary or junction table** to establish the association.

Taking an e-commerce example, you might have a "Customer" table and a "Product" table. Since a customer can purchase multiple products, and a product can be purchased by multiple customers, a many-to-many relationship exists between them. To represent this relationship, a junction table, often called an "Order" table, is used to store the specific combinations of customers and products for each order.

**What is referential integrity?**

Referential integrity is a concept in a relational database that ensures the consistency and accuracy of data relationships between tables. It establishes rules and constraints that maintain the integrity of these relationships. It is enforced through the use of **foreign keys**, which are columns in a table that **reference the primary key of another table**.   
The primary key uniquely identifies each record in the referenced table. The foreign key in another table establishes a relationship between the two tables by referencing the primary key value.

It requires that if a record is added to a table and a value is entered into a foreign key field, the value must exist in the primary key field of another table. **The primary/foreign key relationship ensures that a record in one table is linked to a record in another table**. In some cases, the foreign key can be null to signify that there is no relationship. If null values are not allowed, then there must be a related record.

The key aspects of referential integrity are:

Ensures that a foreign key in one table corresponds to a valid primary key value in another table. This means that the referenced primary key value must exist in the referenced table.

Prevents orphaned records, which are **records in a table that have no corresponding record in the related table**. If a record is deleted from the referenced table, any related records in other tables are also handled appropriately, such as being deleted or updated accordingly.

Ensures that the relationships between tables remain consistent. When updating a primary key value in the referenced table, any related foreign key values in other tables are updated accordingly to maintain the integrity of the relationship.

Integrity constraints can be defined, such as CASCADE, SET NULL, or SET DEFAULT, to specify the action to take when a referenced record is modified or deleted. For example, CASCADE will propagate the changes to related records, SET NULL will set the foreign key to NULL, and SET DEFAULT will set the foreign key to a default value.

Database indexing is a technique used to improve the performance and efficiency of data retrieval operations in a database. It involves creating additional data structures, called indexes, that store a subset of the data in a way that allows for faster searching and retrieval of specific records.

Database Indexing

Indexes are created on one or more columns of a database table. They contain a sorted copy of the values in the indexed column(s), along with a pointer to the corresponding location of the full record in the table. This organized structure allows the database system to quickly locate the desired records based on the indexed values.

When a query is executed that involves a search condition matching the indexed column(s), the database engine can use the index to locate the relevant records more efficiently. Instead of scanning the entire table, it can perform a search operation on the index, which is typically much faster due to its smaller size and optimized structure. Once the matching records are identified, the corresponding full records can be retrieved from the table.

Example

Let's consider a simple example with a database table called "Employees" that contains information about employees in a company. The table has columns such as "EmployeeID," "FirstName," "LastName," "Department," and "Salary."

To improve the performance of queries that involve searching or filtering based on the "LastName" column, an index can be created on that column. This index would store a sorted copy of the last names along with pointers to the corresponding records in the "Employees" table.

For instance, let's say the "Employees" table contains the following data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EmployeeID** | **FirstName** | **LastName** | **Department** | **Salary** |
| 1 | John | Smith | Sales | 50000 |
| 2 | Jane | Johnson | HR | 60000 |
| 3 | David | Anderson | Engineering | 70000 |
| 4 | Lisa | Miller | Sales | 55000 |
| 5 | Michael | Davis | Finance | 65000 |

To create an index on the "LastName" column, the database system would build a separate data structure that looks something like this:

|  |  |
| --- | --- |
| **LastName** | **Record Pointer** |
| Anderson | 3 |
| Davis | 5 |
| Johnson | 2 |
| Miller | 4 |
| Smith | 1 |

If you want to retrieve the record(s) of employees with the last name "Johnson." With the index in place, the database engine can quickly locate the relevant record by searching the index for "Johnson" and retrieve the corresponding **record pointer**, which is 2 in this case. It then uses this pointer to fetch the full record from the "Employees" table.

By using the index, the database system can avoid scanning the entire "Employees" table and instead directly access the specific records that match the search criteria. This significantly improves the performance of the query, especially when the table contains a large number of records.

**Can I create multiple indexes on the same attribute (column)**

No, in most relational database systems, you typically cannot create multiple indexes on the same column. Each column in a table can have at most one index associated with it.

The reason for this limitation is that having multiple indexes on the same column would introduce redundancy and unnecessary overhead. Each index requires additional storage space and maintenance resources, and having multiple indexes on the same column would duplicate this overhead.

Composite indexes

composite indexes can be created that span multiple columns, including the column you want to index. A composite index combines multiple columns into a single index structure, allowing for efficient searching and retrieval based on a combination of those columns. This can be useful when queries frequently involve multiple columns in the same condition or when sorting and joining operations involve those columns.

Example

Using the employees table above to create an index that improves the performance of queries involving both the "Department" and "Salary" columns. In this case, we can create a composite index that spans both columns.

The composite index on "Department" and "Salary" would store a sorted copy of the combined values from these columns, along with pointers to the corresponding records in the "Employees" table. The index structure would look something like this:

|  |  |  |
| --- | --- | --- |
| **Department** | **Salary** | **Record Pointer** |
| Engineering | 70000 | 3 |
| Finance | 65000 | 5 |
| HR | 60000 | 2 |
| Sales | 50000 | 1 |
| Sales | 55000 | 4 |

With this composite index in place, queries that involve conditions on both "Department" and "Salary" can benefit from its efficient search capabilities. For example, if you want to retrieve all employees in the "Sales" department with a salary greater than 50,000, the database engine can utilize the composite index to quickly identify the relevant records without having to scan the entire table.

By using composite indexes, you can optimize queries that involve multiple columns, offering improved performance for specific combinations of column values. However, it's important to consider the trade-offs, as creating too many composite indexes can increase storage requirements and impact the performance of data modification operations.

Benefits of database indexing include:

* Speed up data retrieval operations, especially when searching or filtering based on the indexed columns. By directly accessing the relevant subset of data, the database engine can avoid scanning the entire table and retrieve the desired records more quickly.
* Reduce the amount of disk I/O required for query processing. Instead of reading the entire table, the database engine can read only the index pages, which are typically smaller and require fewer disk accesses.
* Improve the performance of sorting and joining operations. When sorting or joining tables based on indexed columns, the database engine can utilize the index's sorted structure to minimize the time and resources required for these operations.
* There is an option to create index-organized tables (IOTs). IOTs store the table data within the index structure itself, eliminating the need for a separate table storage structure. This can further enhance query performance by reducing disk I/O and improving data locality.

While indexing offers significant performance benefits, it also has some considerations:

* Require additional storage space. Each index creates a separate data structure that needs to be maintained and updated as the underlying table data changes. This can increase the overall storage requirements of the database.
* Whenever data is inserted, updated, or deleted in a table with indexes, the corresponding index(es) need to be updated as well. This additional overhead can impact the performance of data modification operations.
* Designing and maintaining appropriate indexes requires careful consideration. Choosing the right columns to index and monitoring the index performance over time is crucial to ensure optimal query performance.