**Types of data:**

Structure

Semi structure

unstructured

## **What Is a Database?**

A database management system is a software to store, organize, manage, and retrieve data. Think of it like a group of massive spreadsheets that organize information. There's more than one type of database management system, and each is housed on servers, whether in a data center or virtually, on cloud infrastructure (cloud database).

Database management systems come in a variety of shapes, sizes, and flavors, each designed to do different things with different kinds of data. MongoDB is a general-purpose, document-based, distributed database management system built for modern application developers. We also offer an object oriented database ([OOD](https://www.mongodb.com/databases/what-is-an-object-oriented-database" \t "_target)), Realm.

## **How Many Types of Databases Are There?**

there are nearly a dozen types of databases. Some of the more commonly used categories of database include:

### **Hierarchical Databases**

Developed in the 1960s, the hierarchical database looks similar to a family tree. A single object (the “parent”) has one or more objects beneath it (the “child”). No child can have more than one parent. In exchange for the rigid and complex navigation of the parent child structure, the hierarchical database offers high performance, as there’s easy access and a quick querying time. The Windows Registry is one example of this system.

### **Relational Databases**

Relational databases are a system designed in the 1970s. This database commonly uses Structured Query Language (SQL) for operations like creating, reading, updating, and deleting (CRUD) data.

This database stores data in discrete tables, which can be joined together by fields known as foreign keys. For example, you might have a User table that contains data about your users, and join the users table to a Purchases table, which contains data about the purchases the users have made. MySQL, Microsoft SQL Server, and Oracle are examples.

### **Non-Relational Databases**

Non-relational management systems are commonly referred to as NoSQL databases. This type of database matured due to increasingly complex modern web applications. These databases' varieties have proliferated over the last decade. Examples include MongoDB and Redis.

### **Object oriented databases**

Object oriented databases store and manage objects on a database server's disk. Object oriented databases are unique because associations between objects can persist. This means that object oriented programming and the querying of data across complex relationships is fast and powerful. One example of an object oriented database is MongoDB Realm, where the query language constructs native objects through your chosen SDK. Object oriented programming is the most popular programming paradigm.

## All about NoSQL

[NoSQL](https://www.mongodb.com/nosql-explained" \t "_target) is an umbrella term for any alternative system to traditional SQL databases. Sometimes, when we say [NoSQL management systems](https://www.mongodb.com/scale/types-of-nosql-database-management-systems" \t "_target), we mean any database that doesn't use a relational model. NoSQL databases use a data model that has a different structure than the rows and columns table structure used with RDBMS.

NoSQL databases are different from each other. There are four kinds of this database: document databases, key-value stores, column-oriented databases, and graph databases.

### **Document databases**

A [document database](https://www.mongodb.com/document-databases" \t "_target) stores data in [JSON, BSON](https://www.mongodb.com/json-and-bson" \t "_target), or XML documents. Documents in the database can be nested. Particular elements can be indexed for faster querying.

You can access, store, and retrieve documents from your network in a form that is much closer to the data objects used in applications, which means less translation is required to use and access the data in an application. SQL data must often be assembled and disassembled when moving between applications, storage, or more than one network.

Document databases are popular with

## 2.Introduction to oracle:?

Oracle database is a relational database management system. It is also called **OracleDB**, or simply **Oracle**. It is produced and marketed by **Oracle Corporation**. It was created in **1977** by **Lawrence Ellison** and other engineers. It is one of the most popular relational database engines in the IT market for storing, organizing, and retrieving data.

Oracle database was the first DB that designed for **enterprise grid computing** and data warehousing. Enterprise grid computing provides the most flexible and cost-effective way to manage information and applications. It uses SQL queries as a language for interacting with the database.

3.tables, DBMS, RDBMS

Table is a collection of data, organized in terms of rows and columns. In DBMS term, table is known as relation and row as tuple.

## What is a Database Management System(DBMS)?

Database Management System (DBMS) is software used to identify, manage, and create a database that provides administered access to the data.

## What is a Relational Database Management System (RDBMS)?

Relational Database Management System (RDBMS) is a more advanced version of a DBMS system that allows access to data in a more efficient way. It is used to store or manage only the data that are in the form of tables.

## Uses of RDBMS

Relational database management systems are frequently used in disciplines such as manufacturing, human resources and banking. The system is also useful for airlines that need to store ticket service and passenger documentation information as well as universities maintaining student databases.

### What is the Difference between DBMS and RDBMS?

DBMS stands for Database Management System, and RDBMS is the acronym for the Relational Database Management system. In DBMS, the data is stored as a file, whereas in RDBMS, data is stored in the form of tables.

### Difference between RDBMS and DBMS

|  |  |
| --- | --- |
| **RDBMS** | **DBMS** |
| Data stored is in table format | Data stored is in the file format |
| Multiple data elements are accessible together | Individual access of data elements |
| Data in the form of a table are linked together | No connection between data |
| Normalisation is not achievable | There is normalization |
| Support distributed database | No support for distributed database |
| Data is stored in a large amount | Data stored is a small quantity |
| Here, redundancy of data is reduced with the help of key and indexes in RDBMS | Data redundancy is common |
| RDBMS supports multiple users | DBMS supports a single user |
| It features multiple layers of security while handling data | There is only low security while handling data |
| The software and hardware requirements are higher | The software and hardware requirements are low |
| Oracle, SQL Server. | XML, Microsoft Access. |

**SQL stands for Structured Query Language**andis a computer language that we use to interact with a relational database. SQL is a tool for organizing, managing, and retrieving archived data from a computer database. The original name was given by IBM as Structured English Query Language, abbreviated by the acronym SEQUEL. When data needs to be retrieved from a database, SQL is used to make the request. The [DBMS](https://www.geeksforgeeks.org/introduction-of-dbms-database-management-system-set-1/)processes the SQL query retrieves the requested data and returns it to us. Rather, SQL statements describe how a collection of data should be organized or what data should be extracted or added to the database.

## What are SQL commands?

Developers use structured query language (SQL) commands, which are specific keywords or SQL statements, to work with data stored in relational databases. The following are categories for SQL commands.

### **1. Data Definition Language**

* SQL commands used to create the database structure are known as data definition language (DDL). Based on the needs of the business, database engineers create and modify database objects using [DDL](https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/). The CREATE command, for instance, is used by the database engineer to create database objects like tables, views, and indexes.

| **Command** |
| --- |
| **1. CREATE**  **Creates a new table, a view of a table, or other object in the database.** |
| **2. ALTER**  **Modifies an existing database object, such as a table** |
| **DROP**  **Deletes an entire table, a view of a table, or other objects in the database.** |

### **2. Data Manipulation Language**

* A relational database can be updated with new data using data manipulation language ([DML](https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/)) statements. The [INSERT](https://www.geeksforgeeks.org/sql-insert-statement/)command, for instance, is used by an application to add a new record to the database.

| **Command** |
| --- |
| **1. SELECT**  **Retrieves certain records from one or more tables.** |
| **2. INSERT**  **Creates a record.** |
| **3. UPDATE**  **Modifies records.** |
| **4. DELETE**  **Deletes records.** |

### **3. Data Query Language**

* Data retrieval instructions are written in the data query language ([DQL](https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/)), which is used to access relational databases. The SELECT command is used by software programs to filter and return particular results from a SQL table.

### **4. Data Control language**

* Data control language (DCL) is a programming language used by database administrators to control or grant other users access to databases. For instance, they can allow specific applications to manipulate one or more tables by using the GRANT command.

| **Command** |
| --- |
| **1. GRANT**  **Gives a privilege to the user.** |
| **2. REVOKE**  **Takes back privileges granted by the user.** |

### **5. Transaction Control Language**

* To automatically update databases, the relational engine uses transaction control language (TCL). For instance, the database can reverse a mistaken transaction using the ROLLBACK command.

5.Operators in SQL

**Types of SQL Operators**

* Arithmetic operator
* [Comparison operator](https://www.geeksforgeeks.org/comparison-operators-in-sql/)
* [Logical operator](https://www.geeksforgeeks.org/sql-and-and-or-operators/)

**Arithmetic Operators**

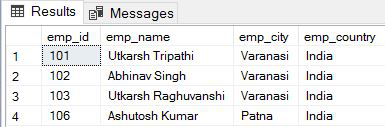
We can use various arithmetic operators on the data stored in the tables. Arithmetic Operators are:

| **Operator** | **Description** |
| --- | --- |
| + | The addition is used to perform an addition operation on the data values. |
| – | This operator is used for the subtraction of the data values. |
| / | This operator works with the ‘ALL’ keyword and it calculates division operations. |
| \* | This operator is used for multiplying data values. |
| % | Modulus is used to get the remainder when data is divided by another. |

**Example Query:**

SELECT \* FROM employee WHERE emp\_city NOT LIKE 'A%';

**Output:**



**Comparison Operators**

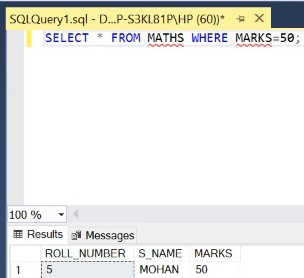
Another important operator in SQL is a comparison operator, which is used to compare one expression’s value to other expressions. SQL supports different types of comparison operator, which is described below:

| **Operator** | **Description** |
| --- | --- |
| = | Equal to. |
| > | Greater than. |
| < | Less than. |
| >= | Greater than equal to. |
| <= | Less than equal to. |
| <> | Not equal to. |

**Example Query:**

SELECT \* FROM MATHS WHERE MARKS=50;

**Output:**



**Logical Operators**

The Logical operators are those that are true or false. They return true or false values to combine one or more true or false values.

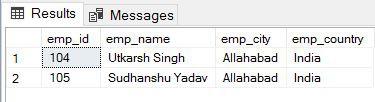
| **Operator** | **Description** |
| --- | --- |
| [AND](https://www.geeksforgeeks.org/sql-and-and-or-operators/) | Logical AND compares two Booleans as expressions and returns true when both expressions are true. |
| [OR](https://www.geeksforgeeks.org/sql-and-and-or-operators/) | Logical OR compares two Booleans as expressions and returns true when one of the expressions is true. |
| [NOT](https://www.geeksforgeeks.org/sql-not-operator/) | Not takes a single Boolean as an argument and change its value from false to true or from true to false. |

**Example Query:**

SELECT \* FROM employee WHERE emp\_city =

'Allahabad' AND emp\_country = 'India';

**Output:**



**Special Operators**

| **Operators** | **Description** |
| --- | --- |
| [ALL](https://www.geeksforgeeks.org/sql-all-and-any/) | ALL is used to select all records of a SELECT STATEMENT. It compares a value to every value in a list of results from a query. The ALL must be preceded by the comparison operators and evaluated to TRUE if the query returns no rows. |
| [ANY](https://www.geeksforgeeks.org/sql-all-and-any/) | ANY compares a value to each value in a list of results from a query and evaluates to true if the result of an inner query contains at least one row. |
| [BETWEEN](https://www.geeksforgeeks.org/sql-between-in-operator/) | The SQL BETWEEN operator tests an expression against a range. The range consists of a beginning, followed by an AND keyword and an end expression. |
| [IN](https://www.geeksforgeeks.org/sql-between-in-operator/) | The IN operator checks a value within a set of values separated by commas and retrieves the rows from the table that match. |
| [EXISTS](https://www.geeksforgeeks.org/sql-exists/) | The EXISTS checks the existence of a result of a subquery. The EXISTS subquery tests whether a subquery fetches at least one row. When no data is returned then this operator returns ‘FALSE’. |
| [SOME](https://www.geeksforgeeks.org/sql-some/) | SOME operator evaluates the condition between the outer and inner tables and evaluates to true if the final result returns any one row. If not, then it evaluates to false. |
| UNIQUE | The UNIQUE operator searches every unique row of a specified table. |

**Example Query:**

SELECT \* FROM employee WHERE emp\_id BETWEEN 101 AND 104;

**Output:**



SQL gives you options for retrieving, analyzing, and displaying the information you need with the GROUP BY, HAVING, and ORDER BY clauses. Here are some examples of how you can use them.

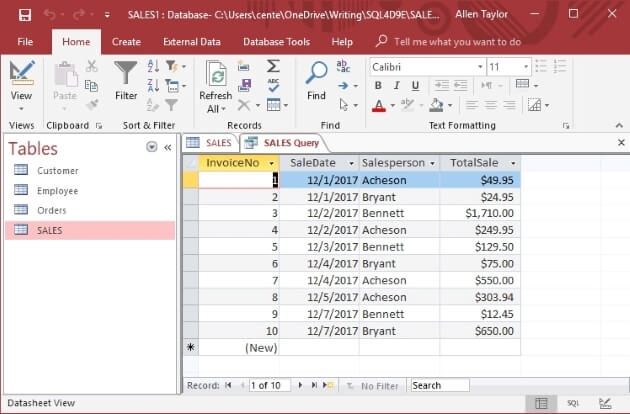
**GROUP BY clauses**

Sometimes, rather than retrieving individual records, you want to know something about a group of records. The GROUP BY clause is the tool you need.

Suppose you’re the sales manager of another location, and you want to look at the performance of your sales force. If you do a simple SELECT, such as the following query:

SELECT InvoiceNo, SaleDate, Salesperson, TotalSale

FROM SALES;



This result gives you some idea of how well your salespeople are doing because so few total sales are involved. However, in real life, a company would have many more sales — and it wouldn’t be so easy to tell whether sales objectives were being met.

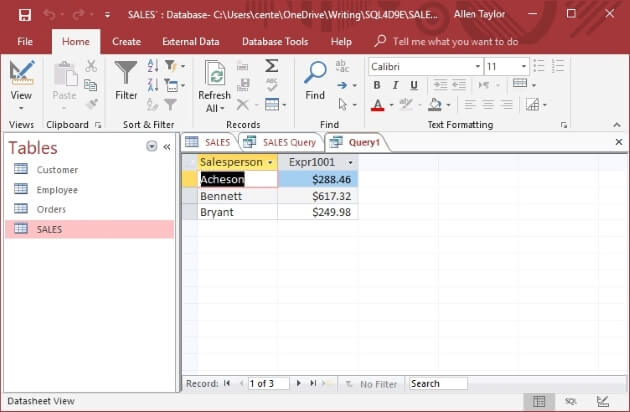
To do the real analysis, you can combine the GROUP BY clause with one of the *aggregate*functions (also called *set functions*) to get a quantitative picture of sales performance. For example, you can see which salesperson is selling more of the profitable high-ticket items by using the average (AVG) function as follows:

SELECT Salesperson, AVG(TotalSale)

FROM SALES

GROUP BY Salesperson;

Running the query with a different database management system would retrieve the same result, but might appear a little different.

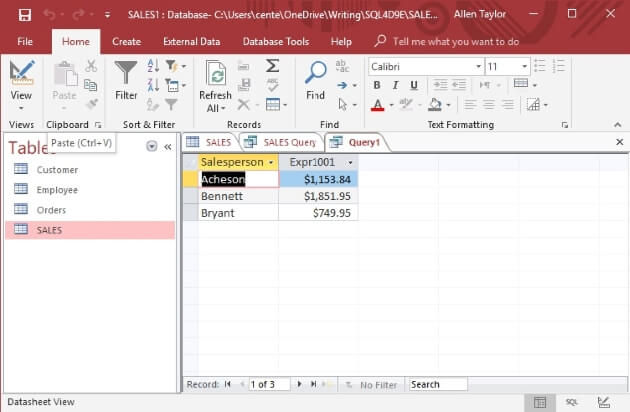


The average value of Bennett’s sales is considerably higher than that of the other two salespeople. You compare total sales with a similar query:

SELECT Salesperson, SUM(TotalSale)

FROM SALES

GROUP BY Salesperson;



Bennett also has the highest total sales, which is consistent with having the highest average sales.

**HAVING clauses**

You can analyze the grouped data further by using the HAVING clause. The HAVING clause is a filter that acts similar to a WHERE clause, but on groups of rows rather than on individual rows. To illustrate the function of the HAVING clause, suppose the sales manager considers Bennett to be in a class by himself.

His performance distorts the overall data for the other salespeople. (Aha — a curve-wrecker.) You can exclude Bennett’s sales from the grouped data by using a HAVING clause as follows:

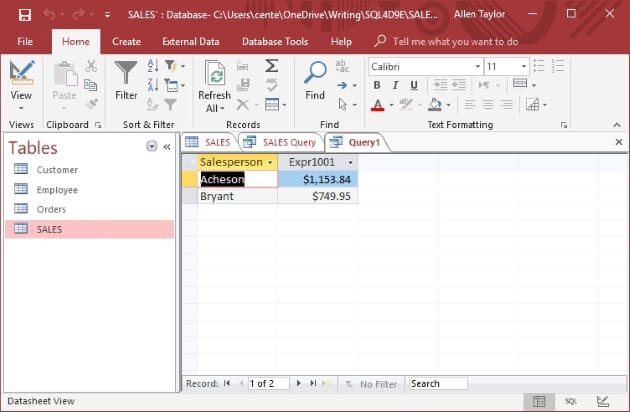
SELECT Salesperson, SUM(TotalSale)

FROM SALES

GROUP BY Salesperson

HAVING Salesperson <>'Bennett';

Only rows where the salesperson is not Bennett are considered.



**ORDER BY clauses**

Use the ORDER BY clause to display the output table of a query in either ascending or descending alphabetical order. Whereas the GROUP BY clause gathers rows into groups and sorts the groups into alphabetical order, ORDER BY sorts individual rows. The ORDER BY clause must be the last clause that you specify in a query.

If the query also contains a GROUP BY clause, the clause first arranges the output rows into groups. The ORDER BY clause then sorts the rows within each group. If you have no GROUP BY clause, then the statement considers the entire table as a group, and the ORDER BY clause sorts all its rows according to the column (or columns) that the ORDER BY clause specifies.

To illustrate this point, consider the data in the SALES table. The SALES table contains columns for InvoiceNo, SaleDate, Salesperson, and TotalSale. If you use the following example, you see all the data in the SALES table — but in an arbitrary order:

SELECT \* FROM SALES ;

In one implementation, this may be the order in which you inserted the rows in the table; in another implementation, the order may be that of the most recent updates. The order can also change unexpectedly if anyone physically reorganizes the database. That’s one reason it’s usually a good idea to specify the order in which you want the rows.

You may, for example, want to see the rows in order by the SaleDate like this:

SELECT \* FROM SALES ORDER BY SaleDate ;

This example returns all the rows in the SALES table in order by SaleDate. For rows with the same SaleDate, the default order depends on the implementation. You can, however, specify how to sort the rows that share the same SaleDate. You may want to see the sales for each SaleDate in order by InvoiceNo, as follows:

SELECT \* FROM SALES ORDER BY SaleDate, InvoiceNo ;

This example first orders the sales by SaleDate; then for each SaleDate, it orders the sales by InvoiceNo. But don’t confuse that example with the following query:

SELECT \* FROM SALES ORDER BY InvoiceNo, SaleDate ;

This query first orders the sales by INVOICE\_NO. Then for each different InvoiceNo, the query orders the sales by SaleDate. This probably won’t yield the result you want, because it’s unlikely that multiple sale dates will exist for a single invoice number.

The following query is another example of how SQL can return data:

SELECT \* FROM SALES ORDER BY Salesperson, SaleDate ;

This example first orders by Salesperson and then by SaleDate. After you look at the data in that order, you may want to invert it, as follows:

SELECT \* FROM SALES ORDER BY SaleDate, Salesperson ;

This example orders the rows first by SaleDate and then by Salesperson.

All these ordering examples are in ascending (ASC) order, which is the default sort order. The last SELECT shows earlier sales first — and, within a given date, shows sales for ‘Adams’ before ‘Baker’. If you prefer descending (DESC) order, you can specify this order for one or more of the order columns, as follows:

SELECT \* FROM SALES

ORDER BY SaleDate DESC, Salesperson ASC ;

This example specifies a descending order for sale dates, showing the more recent sales first, and an ascending order for salespeople, putting them in alphabetical order. That should give you a better picture of how Bennett’s performance stacks up against that of the other salespeople.