Database Basics

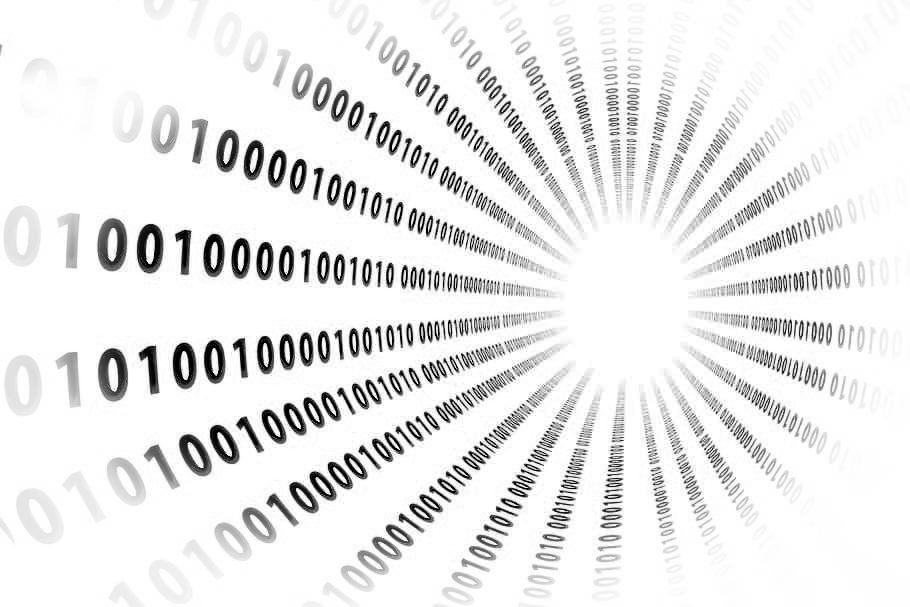


Table of Contents

[1. Introduction 3](#_Toc45717242)

[2. What is a database? 3](#_Toc45717243)

[2.1 Comparing MS SQL Server to MS Excel 3](#_Toc45717244)

[3. Why is a database better? 4](#_Toc45717245)

[3.1 Unique ID 4](#_Toc45717246)

[3.2 Relational data 5](#_Toc45717247)

[3.3 Transactions 6](#_Toc45717248)

[3.4 Data Rule Enforcement 6](#_Toc45717249)

[3.5 Backups 7](#_Toc45717250)

[3.6 Data centralization 7](#_Toc45717251)

[3.7 Size 8](#_Toc45717252)

[3.8 Product comparison summary 9](#_Toc45717253)

[4. When should you use a database? 9](#_Toc45717254)

[4.1 Size 9](#_Toc45717255)

[4.2 Complexity 10](#_Toc45717256)

[4.3 Cost 10](#_Toc45717257)

[5. How a database works 11](#_Toc45717258)

[5.1 Under the hood 11](#_Toc45717259)

[5.2 Data flow 12](#_Toc45717260)

[6. Introduction to SQL 13](#_Toc45717261)

[6.1 DDL (Data Definition Language) 13](#_Toc45717262)

[6.2 DML (Data Definition Language) 14](#_Toc45717263)

[7. Performance 16](#_Toc45717264)

[7.1 Indexes 16](#_Toc45717265)

[7.2 Execution plans 17](#_Toc45717266)

[8. Conclusion 19](#_Toc45717267)

[9. Appendix 20](#_Toc45717268)

[10. Disclaimer 20](#_Toc45717269)

[11. Copyright 21](#_Toc45717270)

# Introduction

You have questions about databases, and I am here to answer them. I have nearly 20 years of professional experience working with Microsoft (MS) SQL Server, Oracle, Teradata, and MS Access. I intend to share with you what a database is, how to decide when you need a database and cover the basics of how to use one.

The database I selected for use in this document is MS SQL Server. I like its balance of ease-of-use and powerful features. It is very widely used and is the database product I worked with the most; over 13 years of developing applications with its technology.

You will find that a database is the most powerful option for storing, serving, and organizing data. The flexibility, reliability, and performance offered in an MS SQL Server database cannot be matched by other methods of storing data, like MS Excel, or lower-tier database offerings like MS Access.

Any set of data from sales records, statistics, or employee performance reports to airplane routes, music scores, or recipes can benefit from the features a database can provide. People around the world leverage the structure, reliability, and power of databases. So can you.

In addition, if you plan on pursuing a career in databases, or even just want to try things out to further your education, I think you will find the information here to be of great interest.

# What is a database?

At its most basic, a database is a system that imposes structure on data. While that may not mean software 100% of the time, you won’t see a paper “database” very often anymore.

There is a wide variety of database software available today. Modern databases can be geared to be easy to use (MS Access) or have free licenses (PostgreSQL, MySQL, MongoDB) or are the more traditional enterprise-level platforms that have huge flexibility, scalability, and support (Microsoft SQL Server, Oracle, IBM DB2).

What they all have in common is that they take that idea of organizing data, and they build on it extensively. Even the free database options offer a huge number of features and a wide variety of programmability.

## Comparing MS SQL Server to MS Excel

Why am I comparing MS SQL Server to MS Excel? Many people are familiar with using Excel to store organized data, while few have worked directly with databases. It offers a point of comparison that people can use as a basis for understanding databases.

Why am I not talking about MS Access? If you are here to learn database basics for a career, there are far more jobs using MS SQL Server than MS Access. Alternately, if you are here looking for business solutions, I would encourage you to skip MS Access. It doesn't offer as many features, and if you are outgrowing Excel, Access is not likely to be a long-term solution.

### Similarities

Conceptually, you could look at SQL Server tables as each a single spreadsheet. However, SQL Server tables and Excel spreadsheets both offer many features that the other doesn’t have.

A spreadsheet and a database table both store data in rows and columns. Both products offer tools that allow you to sort your data, filter your data, or search for specific pieces of data within your larger set of data.

They both offer a wide array of functions to manipulate your data, though the functions offered in each differ substantially. I feel specifics about those functions go beyond the scope of an introduction, so we won’t be covering those here.

### Differences

Excel is data presentation software even more than it is data storage software. The flexibility it offers reflects that. An Excel spreadsheet can store and display many differently structured chunks of data combined on the same sheet. SQL Server’s tables are each a single grid of information with a single structure.

Excel offers visualization tools, like graphs and pivot tables. SQL Server is intended as a backend data source for other applications that would provide the visualization, including Excel.

Excel offers simplicity and user-friendliness that a database by itself can’t offer. Anyone can open a new Excel spreadsheet and start typing. Database tables take more time and training to set up and use.

The simplicity that Excel offers reflects its place, as I see it. If you have simple data or need to present data to users, Excel is often an excellent solution. As your data needs grow, Excel often cannot fulfill them anymore, and it’s time to look at a database.

# Why is a database better?

I’d like to talk about the advantages of MS SQL Server and give you some ideas on why you might prefer to use a database over something like Excel. There are many powerful features offered by a database that a spreadsheet simply can’t match. Most of these features form the “structure” we are imposing on the data we are storing in the database. You can think of the structure as a set of laws intended to impose order on an otherwise “wild west” town made of data.

## Unique ID

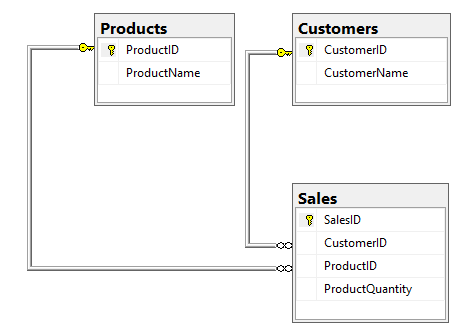
One thing seldom found in software outside a database is a primary key. A primary key is a unique code that enforces that uniqueness onto the data. No duplicates are allowed.

Social Security numbers, a UPC code on a product, or bank account numbers are all examples of primary keys. For example, there should never be two people with the same Social Security number. You can easily see the problems that would be caused if two people were trying to collect Social Security income utilizing the same number. They need to be unique or the process can’t work.

Creating and enforcing something like a primary key is very hard to do in Excel. In SQL Server, it is a short [SQL statement](#_Introduction_to_SQL) that developers include in nearly every table.

## Relational data

One of the most important features of database software is that it will allow you to interrelate your data. In this section, we will use an example of a list of customers, a list of products, and a table that relates those two, showing which customers bought which products, called “sales”.



You can set up connections between tables that enforce rules between them. These are referred to as foreign keys. A foreign key is a data integrity rule that effectively makes that connection between two tables. These tell the database, and a developer, how the data relates between the tables.

In the sales table, only the ID of the customer and the ID of the product will be stored, not their names. This comes in very handy when a product name changes. You just update one record in the products table, and any applications or reports that use SQL based on these tables will automatically pick up the change.

In addition, the connection between the tables prevents you from recording a sale for a customer who doesn’t exist in the “customers” table, or a product that isn’t in the products table. In other words, it prevents clearly incorrect data from being recorded.

If you were storing data like this in Excel, automatic updates and this type of rule enforcement are unavailable. You would likely need to update multiple records, increasing work and the likelihood of error.

## Transactions

Transactions are the basis for trustworthy data. Transactions are a validation that any update to database data was completed successfully.

When a change is submitted to a database the application submitting the change isn’t just throwing out an update and trusting that the data has been changed. The application gets back either a completion message or an error, so it can move on to the next line of code (if any) or handle any error generated by the attempted database update.

Multiple updates to a database can be linked together with code into a single transaction. This means that if any one of the updates fails the whole transaction fails, and no changes are committed to the database data.

The classic database transaction example is a bank transfer of funds between two accounts. Money is deducted from one account and added to another. Both updates must be completed successfully or neither update is kept.

What if an application deducted money from one account but the computer running the code was turned off before the application could add it to the other account? If the application is not properly coded, the money just disappears.

With a database transaction linking both updates, it guarantees the money cannot be lost. The transaction says that the withdrawal cannot happen if the deposit doesn’t happen too, and the application attempting the update will have an error.

## Data Rule Enforcement

There is an old saying in computer programming: “garbage in, garbage out”. This saying implies that no matter how good your program is, if your users give it bad input, you will get bad results. This applies very much with unstructured data. Unstructured data is likely to have “garbage” entered in, which can cause many issues when it comes time to consume that data.

Millions of people around the world store important data in Excel. While it is possible to automatically enforce rules in Excel, very few users do so, and even fewer do it properly.

The first level of data rule enforcement is data type (e.g. integer, decimal, text, date, binary). You assign a data type to every column in a table, and if the wrong type of data is attempted to be inserted into even one of the columns, then the whole insert attempt is rejected.

As discussed under [Relational data](#_Relational_data), foreign keys are another type of data rule enforcement.

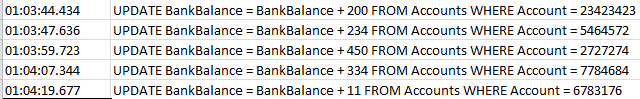
Data rule enforcement is built into good database software. Beyond the fundamental types of rule enforcement I just mentioned, SQL Server gives you the ability to apply incredibly sophisticated rules, should you desire it.

## Backups

Database backups are more robust than the sorts of backup solutions with which most people are familiar. In addition to file backups done to your database, every single transaction hitting a database is logged as it completes successfully. This allows not only reliable data but also means you can restore a database to a specific point in time.

You can think of the transaction log as a list of every single INSERT, UPDATE, and DELETE command issued to the database in the order in which they happened.

Conceptual example of a transaction log



Imagine a scenario where a developer updates an application with buggy code that inadvertently deletes thousands of rows of vital data. After the developer halts the bad updates, they can combine the latest database file backup with the database transaction logs up to the point in time when things started to go wrong. This will effectively take the database back in time to the point they indicate.

## Data centralization

Data centralization is not the physical centralization of data, but rather the process of setting up your different datasets so that they can be accessed together. This usually means loading many datasets onto a single database server (or linked servers) so the data can be combined, correlated, or compared.

Bringing your data together in one location solves crucial problems, saves works, and creates new opportunities. Data centralization is a task that gets more and more important when the number of different datasets, and their size, grow larger.

### Solving problems

Imagine a company that has grown rapidly recently. A single group of employees used to maintain the sales records in Excel, but that had to be split into regional groups to accommodate the number of sales, as the company grew. Each region was given a copy of the employee spreadsheet, the customer spreadsheet, and must track their own sales spreadsheet.

* Overhead: Having multiple people updating lists of employees and customers is time wasted that could be spent doing other things.
* Consistency: Multiple different people in the company trying to maintain lists of employees, customers, and sales is going to create a lot of errors. Lost time to correct errors, mishandled sales, and lost customers all add up to money wasted.
* Reporting: Every time the CEO wants a report on sales, they must wait for each region to report their sales and then have all the data combined.

The decision is made to move to a database, and I.T. has been tasked to keep centralized tables updated with all employee and customer information. The regional sales groups are now only responsible for recording their sales in the database.

Manhours are saved so the sales groups can focus on sales. The data will be consistent across groups and more accurate. Reporting will be quick and easy, allowing management to make decisions faster and with greater confidence.

### Interconnection

In that same company, I.T. is tasked with keeping phone records for the sales group. They will see how much total time the sales employees spend on the phone and how long they spend on each call. I.T. links up both sets of data by employee, so they can now relate sales to phone statistics in reports. Now that I.T. can tie company sales records to phone records, employees can see information like who is closing sales in the least amount of time, or the most. This reporting allows management to provide rewards or additional training where it might be needed.

This sort of data interconnection is getting more advance and complex all the time. Interconnecting information on potential customers is one of the fastest areas of growth in the tech world.

## Size

The sheer size of data a database can handle can’t be matched by spreadsheet software. The number of rows available in Excel is limited to just over 1,000,000. SQL Server tables can and do hold billions of rows. Its real limit is how much storage space is available.

## Product comparison summary

|  |  |  |
| --- | --- | --- |
|  | Microsoft Excel | Microsoft SQL Server |
| Ease of use | Very user-friendly | Need training |
| Data presentation | Yes, excellent tools | No built-in tools, but many external options. |
| Unique ID | Not really possible. You could cobble something together that would look like a unique ID, but it wouldn’t be something that Excel would enforce. | Fundamental to database design |
| Relational data | Not possible. There are lookup functions, but it’s not the same thing at all. | Fundamental to database design |
| Transactions | No such thing | Easily implemented |
| Data rule enforcement | Simple rules | Fundamental to database design.  Rules can be as complex as you like. |
| Backups | Manual | Automatable and can be restored at the transactional level |
| Data centralization | Only works if you have a small amount of data to centralize. Even then, there would be user-contention/ownership issues constantly. | Ideal. There are multiple tool suites that facilitate this process. |
| Size | 1,000,000 rows | Only limited by hard drive space |

# When should you use a database?

Choosing when to make the move to a database is difficult. If you’ve been dependent on spreadsheets, it’s going to be a big change to the way you and your employees have been doing things. As a database developer, I am biased, but I would say that the sooner you make the move the better. The longer you and your employees continue with your old methods, the harder they are to change. Consider a few of the key elements in this section to help you make the call.

## Size

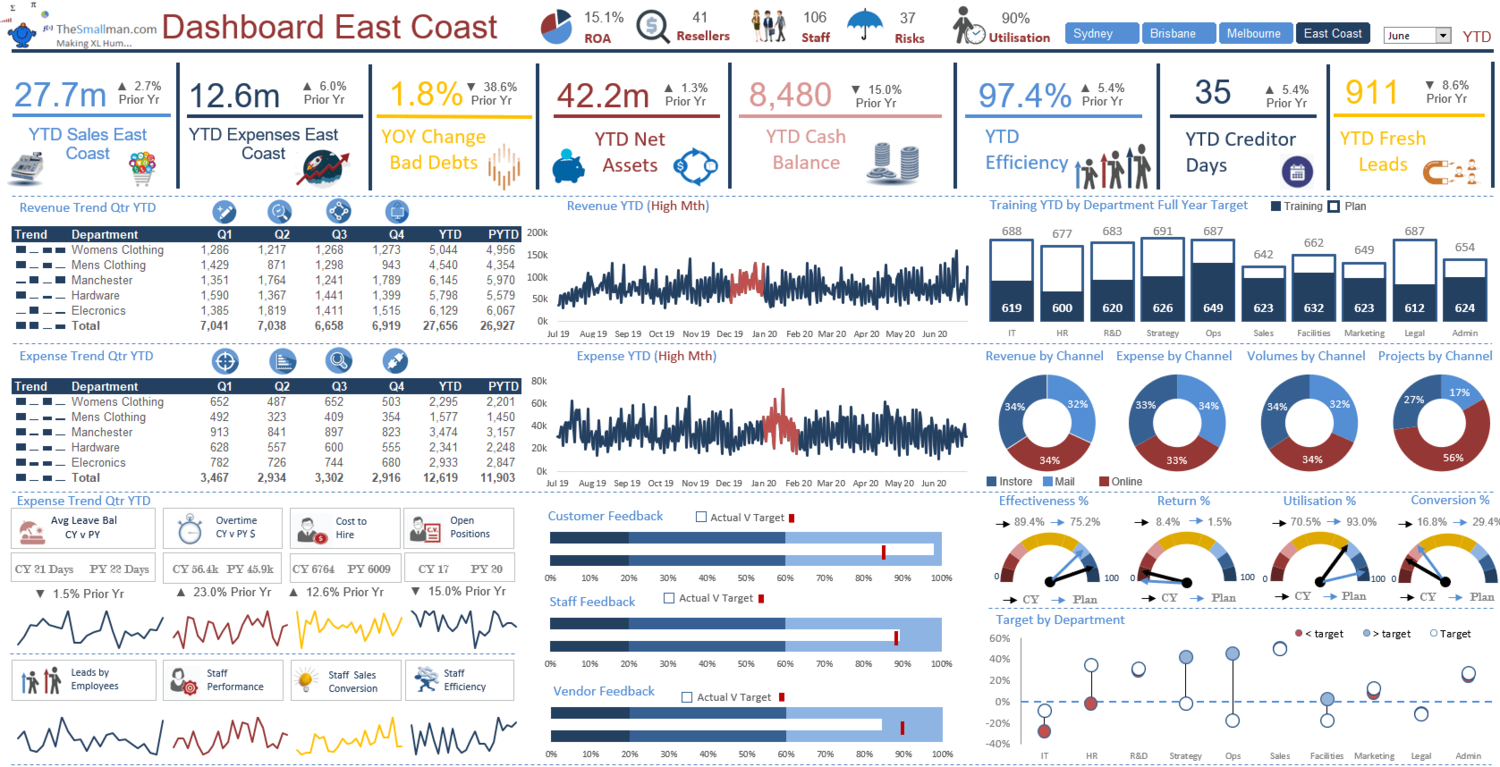
Microsoft Excel can hold just over 1,000,000 rows of data, but you should move to a database long before coming close to that limit. Stability and speed alone will represent issues in a spreadsheet containing hundreds of thousands of rows.

Despite that 1,000,000-row limit seeming a long way off, Excel is not really intended to be used for data that large. If you have a spreadsheet coming up on 100,000 rows, it’s time to think about using a database. Who is going to consume 100,000 rows of data in Excel, except to summarize it? If you are summarizing all that data, you are already better off in a database and just outputting to Excel for your users. A database is going to do it better and faster.

## Complexity

My professional experience has been that as soon as you try to interconnect Excel spreadsheets, you are in trouble. Before I was trained to become a database developer, I worked in a department that was trying to consolidate data for management from a variety of sources. I shudder now to think about the number of links present on the main dashboard of that spreadsheet.

Over-complicated Excel spreadsheet example



Our department owned some of the spreadsheets linked from that dashboard, but some were owned by other departments or were autogenerated from applications. Every morning was spent collecting data from different sources and tracking down issues. Sometimes an application failed to produce the spreadsheet on time, or a spreadsheet had been changed so the cell we were aiming for wasn’t there, or the whole file was missing.

Everything we had should have been moved over to a database a long time prior to it getting to that point. That’s my lived worst-case scenario, but where is the threshold? When is it time to move to a database?

I can’t recommend it highly enough to move sooner rather than later. The benefits gained are huge, and the learning curve for basic database functionality isn’t as high as you think. As soon as you have multiple people updating similar data, or you are trying to link spreadsheets together, you need to seriously consider moving to a database.

## Cost

Cost is usually the biggest factor holding people back from making the switch. You have three basic factors to consider: hardware, software, and salaries. The costs of SQL Server are very low for a small user base but escalate quickly when you scale up. For specific cost information, see the link under [License costs](#_License_costs).

### Scenarios

1. You are a business owner with 15 employees, and you have the time and inclination to learn how to use SQL Server yourself. You have a large amount of data to deal with but have no plans to serve customers data directly. From a cost perspective, this is an almost ideal situation to migrate to SQL Server. You can use SQL Server Express, which is free. This will leave you with the cost of a server, but for 15 employees, almost any PC will work fine as a server.
2. You are the CEO of a small company of 75 employees. You do not have time to work with SQL Server yourself, and you want to offer a website directly to your customers where they can read about your large selection of products. You can get SQL Server Standard Edition. It’s going to cost you at least $8,000 for a license that will last for years and $3,000 for a good server.
3. You are the CTO of a company of 350 employees spread out all over the country, your employees need all kinds of data handled, and need to be processing sales directly with customers. It’s time to go to the cloud. You’ll be looking at a data warehouse and web serving configuration. You won’t have to pay for servers yourself, or the staff to maintain them, but the cloud services are going to be $5,000/month and are going to increase as your user base increases in size.

# How a database works

## Under the hood

I have put a link in the [Materials section](#_Materials) of this document where you can download SQL Server’s free Express Edition. If you chose to do so, also download and install SQL Server Management Studio to provide you with a graphical user interface.

In the sections below, we are going to talk about some of the underlying pieces of SQL Server, and being able to view them may help your understanding. I would encourage you to try it out.

### Master Database

As the name implies, the master database is the database over all the other databases. It’s essentially a database of databases. All the components of databases you create are listed there, though not the data.

You will find information about:

* Security (e.g. - users, logins)
* Database Objects (e.g. - databases, tables, columns)
* Server Objects (e.g. - database instances, links to other servers)
* Server Settings (e.g. - database replication)

While I encourage you to look around in your own instance of this database, use extreme caution in a professional situation; the risk to cause major problems is high.

### MSDB Database

The MSDB database stores information about major services provided in SQL Server. If you are wondering what “MSDB” stands for, I cannot enlighten you, because Microsoft has failed to make it clear. I think of it as “Microsoft Database,” but that is not official.

Main services controlled from MSDB are:

* SQL Server Agent – Server “jobs” are controlled by this service. Jobs are how we automate processes directly on SQL Server without the need for an external application. For example, you could schedule a nightly data delete from a database or a weekly transfer of data between databases.
* Database Mail – Emails can be generated directly from the server by this service, usually instigated by a job process.
* Service Broker – Advanced functions of built-in workload distribution are handled here.

MSDB also stores logs of backups and restores of databases and supports features of SSMS.

## Data flow

Below is a conceptual diagram of how an application interacts with a database. As you can see, there are multiple layers that information must traverse before an application will know that a submitted database request has been completed, and data (if any) is returned.



# Introduction to SQL

The [American National Standards Institute (ANSI)](https://www.ansi.org/) has set standards for the language the databases use: SQL (Structured Query Language). Nearly all major database platforms conform to ANSI standards.

On top of that, different database technologies provide custom functions and capabilities of their own, but the basic underlying syntax for SQL statements should be consistent if a database claims to be ANSI-compliant.

Common code formatting practices have keywords in all caps, to visually differentiate them from objects like tables and columns. That is why the following sections will have capital letters used for SQL keywords.

These are going to be simple examples, intended to demonstrate the basics. Those basics are going to come up often when working on databases.

## DDL (Data Definition Language)

### CREATE TABLE

This is the basic building block of your database, and it is where most of the database-side data validation will be set. The name of our example table is “HumanResources.Department”. “HumanResources” is the schema name and “Department” is the table name.

A schema is often used as a way of categorizing your tables but has additional functionality too. They are most often used for security and organization.

Below the “schema.table” name is the column names section. Each of the columns in your table receives the data type that is defined here. Data types set the size of memory available for the data stored there and are strictly enforced. If you try to insert or update data to an invalid type, the SQL statement will fail.



To read more about data types, please see [Microsoft’s documentation on SQL Server data types](https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver15).

### CREATE INDEX

Indexes are created on tables to allow data to be found very quickly. To learn more about them see the [Indexes](#_Indexes) section of this document.



### DROP TABLE

This statement is going to be difficult to undo, and all data in the table will be lost, so be very sure before dropping a table.



### ALTER TABLE

This statement is used to add new columns, drop columns, or alter existing columns from a table. If you drop an existing column, all data in it will be lost.



## DML (Data Definition Language)

There are four basic concepts in databases: Create, Read, Update, and Delete. “CRUD” is the common mnemonic to remember them.

The SQL keywords map neatly over to:

* Create = INSERT
* Read = SELECT
* Update = UPDATE
* Delete = DELETE

### INSERT

INSERT is how you add new rows of data to the database. You need to strictly conform to all the data types for every column, but you can exclude columns from the insert statement if the column allows NULL values to be inserted. “NULL” means “no data” and doesn’t have a value.



### UPDATE

UPDATE statements are used to change existing data in your database. You can update any number of columns in a single statement, but only one table at a time. They will succeed or fail together.

The WHERE clause is a part of a query that allows you to restrict the rows impacted or returned. It is not changing any values like the code in the SET clause of the UPDATE statement. You don’t usually need to address every row in a table, so WHERE clauses are very common.



### DELETE

DELETE removes rows from your table. You can delete all data in a table with one statement, so be careful.



### SELECT

SELECT is how you read data from the database. Simple SELECT statements are very common when building an application, or during data validation.



### JOIN

More complex SELECT statements usually involve interrelating the data in multiple tables. You do this by adding JOIN statements to the SELECT. You join tables using a common data element.

In the below example, we are reading a table that contains information about what department employees served in over a period (HumanResources.EmployeeDepartmentHistory). It stores the employee’s ID for a date range to show which department ID and shift ID applied at the time.

Since that table is mostly just ID codes, we need to join it up to other tables to get a human-readable set of data. We join this history table to the person table, department table, and the shift table using the unique IDs found there.

Once those tables are joined, we can assemble a report that anyone can read. I added in the criteria that the “EndDate IS NULL” because an EndDate being present implies that the employee no longer works in that department, and I want current data.



Please note that these joins are only in scope for this statement and do not represent a permanent structural change like a foreign key does

# Performance

## Indexes

Indexes are how a database finds data quickly. Imagine a reference book in a 3-ring binder with hundreds of pages of information. An index is like adding tabs to sections of the book. The tabs let you flip quickly to the section you need.

You can add multiple indexes to a table; however, you must be careful how many you add. If you index every column in the table, it’s like putting a tab on every page of that book. It adds a lot of extra weight to the book while being nearly useless.

With one exception (clustered index described below) every index takes up hard drive space. Unused indexes will slow down queries on the table, so be prudent in your index choices.

### Clustered Index

A clustered index is the physical order in which the data is stored on the hard drive. You might put the clustered index on the “LastName” column of a table of employees. Each row would be stored alphabetically by last name on the hard drive. This makes accessing data by last name very fast.

Often it is best to put the clustered index on the unique identifier column of each table. Data stored in order by a known unique value helps the database engine with optimizing query efficiency.

A table with no clustered index is known as a “heap”. There are virtually no situations where a heap is better than an ordered table.

### Primary Keys

The primary key is a column or combination of columns that uniquely identifies a row of data in a table. A table can only have one primary key. It cannot be set to a null value. It is often used as the clustered index on the table.

Usually, a value is created for a primary key if the data in the table doesn’t already have one. For example, if you were creating an “employees” table, you might set their social security number as the table’s primary key. However, this is often not permitted, as this might expose employees’ Social Security numbers. So, a new column with a new value would be added to the table to stand as the primary key.

### Other Indexes

You can add more indexes to your tables as your applications and reporting require them. Additional indexes can be flagged as unique or not depending on the data in the column. Flagging an index as unique will force data integrity checks, and non-unique values will not be allowed in that column.

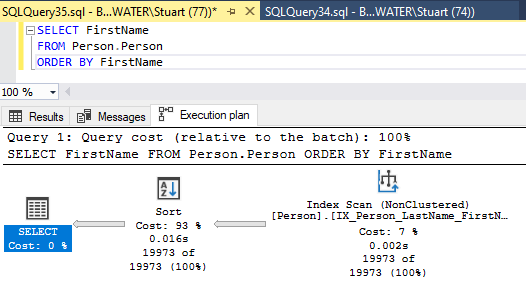
## Execution plans

Demand for tuning your database will grow as its size and user-demand both increase. Performance tuning is crucial to keeping your costs under control and your users happy. There are books and numerous tools that can be utilized to help you tune your database.

The bread-and-butter of tuning is reviewing the execution plan for your SQL. An execution plan shows you how SQL Server is breaking down your SQL into logical chunks and organizes them to give you your results.

### Example Execution Plan 1

This query returns a list of first names from the “Person.Person” table and sorts them in alphabetical order.

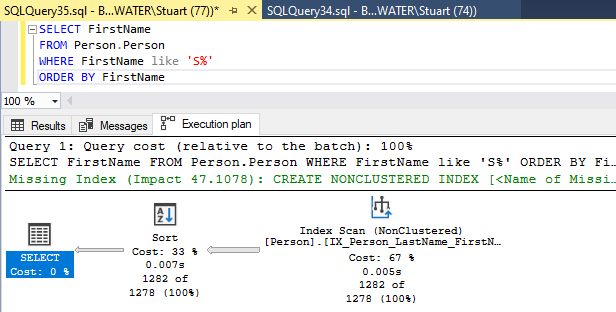


This is a very simple execution plan generated by SQL Server Management Studio (SSMS). The results are read from right to left, as the arrows point. It is showing you that it is retrieving the list of FirstName values from “Person.Person”, then sorting them, and returning them to the screen.

To tune a query, you look at the “Cost” values, and target steps with high costs. In this case, it’s the sort step, but this SQL doesn’t need tuning. Even though the relative cost is high, that is normal and appropriate for a sort in a query this size.

### Example Execution Plan 2

Let’s look at one with a problem. The below query is very similar to the last, but now we are restricting the results to just employees whose first names begin with the letter “s”.

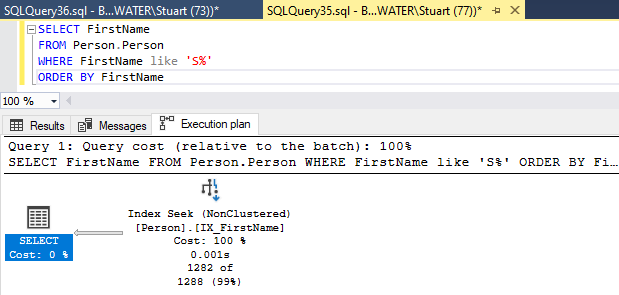


The higher cost has shifted over to the data retrieval, where we would prefer it not to be. Beyond that, SQL Server has even suggested a fix to our issue. The green text there is a SQL statement that it recommends we use to resolve the issue.

Below the SQL statement is easier to read. It just adds an index to the FirstName column.



After the index is added, we see a much better execution plan. The index has not only resolved the performance problem caused by filtering the FirstName column but has resolved the speed issue with the sort. SQL Server goes straight to the new index and does all its work very quickly.



The execution plans offered in SSMS are vital but will not resolve all your problems. As I said before, there are books written on this topic, but even more frequently, just knowing your database well is going to be the key to resolving performance issues.

# Conclusion

I hope this brief look at databases has given you a better idea of what they are like and the purposes they can serve. If you have found this document at all daunting, then I would encourage you to [download SQL Server Express Edition](#_Materials) and use this document as a reference to understand what you are seeing and doing there.

I have included a [link to a sample database](#_Materials) created by Microsoft and am [including some scripts](#_Materials) for you to play around with. Give it a try. I think you’ll be surprised by how much you can do.

# Appendix

### License costs

Link to MS SQL Server license costs: <https://www.microsoft.com/en-us/sql-server/sql-server-2019-pricing>

### Materials

Install SQL Server

1. Go to [SQL Server](https://www.microsoft.com/en-us/sql-server/sql-server-downloads) Downloads
2. Scroll down to the free editions
3. Download either Developer or Express editions
4. Install the application

Install [SQL Server Management Studio (SSMS)](https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms)

Sample Database: [AdventureWorks](https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver15)

Sample SQL for experimentation



### Sources

[Microsoft T-SQL Reference](https://docs.microsoft.com/en-us/sql/t-sql/language-reference) - Guide on the MS SQL Server language

[SQL Server Central](https://www.sqlservercentral.com/) – Excellent reference for authoritative articles about MS SQL Server

[How Databases Work](http://coding-geek.com/how-databases-work/) – Appears to be good for reference information, though a bit difficult to consume.

<https://hub.packtpub.com/how-sql-server-handles-data-under-the-hood/>

Decorative images’ source is labeled in MS Word alt text. All other images are created by the author.

# Disclaimer

Bellevue College, the author of this document, and course instructor take no responsibility for any harm caused by utilization of the information contained in this document, use of the embedded links, or the content therein, or use of the attached database scripts.

# Copyright

©Copyright 2020 by Stuart Ketcham

This document was completed as a requirement for the Technical Writing Certificate program at Bellevue College, Spring 2020

Style Sheet

Name: Stuart Ketcham

Date: 7/10/2020

Title: “Database Basics”

Style Manual: Microsoft Writing Style Guide, Microsoft Corporation, 2018

Dictionary: Microsoft Word for Office 365

|  |  |  |
| --- | --- | --- |
| Item | Decision | Source |
| Serial (Oxford) comma | include | <https://docs.microsoft.com/en-us/style-guide/punctuation/commas> |
| Spelling out numbers versus using numerals | Spelling for less than 10.  Numerals for 10 or greater. | <https://docs.microsoft.com/en-us/style-guide/numbers> |
| Acronyms and abbreviations | SQL  MS  MSDB  ID  CEO  CTO  I.T.  PC  ANSI  SSMS | <https://docs.microsoft.com/en-us/style-guide/acronyms>  Refers to:  <https://ahdictionary.com/> |
| Capitalization in headings and subheadings | Sentence-style capitalization in titles and headings | <https://docs.microsoft.com/en-us/style-guide/capitalization> |
| Specifications for use of hyphens and dashes | See guide | <https://docs.microsoft.com/en-us/style-guide/punctuation/dashes-hyphens/> |
|  |  |  |
|  |  |  |