SQL Server - Module 2

Overview:

This week is an introduction to SQL with SQL Server and it is designed to give you an understanding of the core fundamentals. You are already familiar with MySQL so conceptually you can think of them as the same with some difference in syntax because they are championed by different companies for the same relational database type. We will cover data types, syntaxes, clauses, CRUD, how to query a database, aggregate data and generate output, utilise functions to manipulate data and much more. You won't have a project this week as every Module and every chapter is followed by practise exercises to test your understanding. You will first practice these exercises in SQL Server and once you're happy with your results, you will then submit your answers. Your HS team will be here to assist you when needed.

Created and curated by:
Manan Shuddo, Cyrille Arnold, Lymeng Chhim, Muoykear Lim, Senghong Soeung, Oojal Jhutti, Ioni Spinu, Sokhna Vor

Contents

Overview

Transact-SQL is an essential skill for data professionals and developers working with SQL databases. With this combination of expert instruction, demonstrations, and practical labs, step from your first SELECT statement through to implementing transactional programmatic logic.

Work through a series of modules that each concentrate on a different aspect of the Transact-SQL language, with a particular emphasis on querying and changing data in Microsoft SQL Server or Azure SQL Database. The exercise will use a sample database, so you can practice Transact-SQL on your local database server.

The level of difficulty increases as you go through the modules.

What you'll learn:

- **Querying Data with Transact-SQL**
- **Introduction to Data Modeling**
- **Developing SQL Databases**
- **Creating Programmatic SQL Database Objects**

Getting started

Welcome to Week 2 of the DataU SQL Programme, before you get started here are some of the resources you may need in order to complete the study programme:

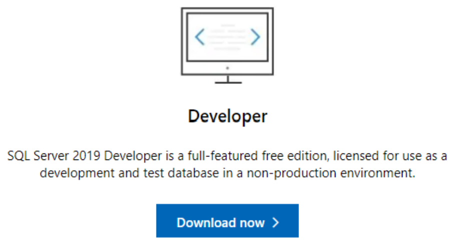
Hardware:
You will need your own computer or laptop device, including for example a Windows, Mac or Linux based desktop or laptop computer. This programme will require you to install software from the internet, as such a tablet may not be a suitable computing solution.

Software:

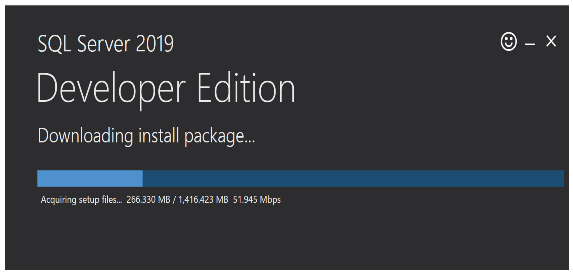
Installation guide:
https://docs.microsoft.com/en-us/sql/database-engine/install-windows/install-sql-server?view=sql-server-ver15

Two major steps to install 1) SQL Server & 2) SQL Server Management Studio (SSMS)

1. **Install SQL Server**
   https://www.microsoft.com/en-us/sql-server/sql-server-downloads?rtc=


Developer
SQL Server 2019 Developer is a full-featured free edition, licensed for use as a development and test database in a non-production environment.
Download now ›

   And you can run the downloaded executable file.



SQL Server 2019
Developer Edition
Downloading install package...
Acquiring setup files... 266.330 MB / 1,416.423 MB 51.945 Mbps

2. **Install SSMS 18**
   https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2018

3. **Now install Sample Database called AdventureWorks 2012.**

   For guide browse to

https://github.com/microsoft/sql-server-samples/tree/master/samples/databases/adventure-works (files at https://github.com/Microsoft/sql-server-samples/releases/tag/adventureworks)

We'll install the file and restore it into your database.

- Create a folder as C:\Samples\AdventureWorks
- Download AdventureWorks files you can now install AdventureWorks files provided by your instructor. Place them in your created folder C:\Samples\AdventureWorks**.**
    - Download AdventureWorks.bak Sample Database (MAIN)
    - Download the AdventureWorksLT2012.bak Sample Database (Exercise)

4. **Now start SQL Server Management Studio**, and when prompted, enter or select the following options and click **Connect**:
    a. **Server type**: Database Engine
    b. **Server name**: (local) (or (local)\instance_name if you installed a named instance)
    c. **Authentication**: SQL Server Authentication
    d. **Login**: sa
    e. **Password**: *The password you specified during installation*

5. Follow this tutorial to finish **restoring the database** and finish your installation. https://qawithexperts.com/article/sql/download-adventureworks-database-and-restore-in-sql-server-s/315

Prerequisites:
Starting from ***Module 1,*** you will be required to have initiated a database that will be provided to you by your instructor.
In order to initiate the database, follow the installation process above.

Connectivity:
This programme of study will require some Internet access, you may be able to complete some sections without it.

Resources:
Assessment quizzes - each chapter will include a quiz to test your understanding. You will need SQL Server to practice your exercises. At times, you will be quizzed after each topic so watch out for that.

Scripts:

It is highly recommended that you save each and every query that you are working on your SSMS during your exercises. Save your query or script in a .sql file in your local machine organized into folders per module. In order to save your query or script into a .sql file, go to File > Save as… or alternatively, for Windows/Linux users press CTRL + S to save the files and for MacOS users press CMD + S to save the files.

Special Instructions:
Some chapters will begin with a Special Instructions Section. You are required to follow those instructions carefully in order to be able to complete the exercises. If you are struggling with following the instructions, please make sure to draw the attention of your instructors for further assistance. If at any time, you get stuck and can't figure out the answer after a couple of attempts please let your instructors know so we can help you move forward.

Major Differences between MySQL and SQL Server

For your information here is a summary of the major differences between these popular relational database systems. Because they are championed by two of the world's largest companies, we'd expect some syntax and feature differences.

General difference

	SQL Server	MySQL
Website	SQL Server	MySQL
Licensing	It is proprietary software.	It is free and open source under GPL v2 license, as well as distributed as proprietary software.
Developer	It is developed by Microsoft. Since SQL server is designed by Microsoft, it is also often called MS SQL Server.	It is developed by Oracle Corporation.
Written in	It is written in c and C++	It is also written in C and C++.
Supported Platforms	Supports Linux, Mac OS X, Microsoft Windows Server, and Microsoft Windows operating systems. It was originally developed for Windows exclusively. However, it is also available on Linux and Mac OSX via docker. But, SQL server on Linux or	Smooth Support for Linux, Solaris, Windows, macOS and FreeBSD operating systems. Runs nearly on every popular OS.

	Mac OS X will definitely lack certain features.	
Supported Programming Languages	Supports multiple programming languages including Java, PHP, VB, Delphi, Go, Python, Ruby, C++, and R.	It supports all programming languages supported by SQL Server. Furthermore, MySQL supports some additional languages including Perl, Scheme, Eiffel, Tcl, and Haskel. This makes MySQL very popular among developer communities.
Syntax	SQL server syntax is simple and easy to use.	It is observed that MySQL syntax is a bit complex.
Multilingual	Available in multiple languages	Available only in English language.
Storage Engine	Single storage Engine which is its native engine.	Multiple storage Engine support. Also has an option for using a plug-in storage engine.
Filtering	Supports row-based filtering which filters out the records on a database by database way. Gives the advantage of filtering multiple rows without considering a number of databases. Moreover, the filtered data is kept in a separate distribution database.	Allows to filter out the tables, rows, and users in a variety of ways. However, MySQL supports filtering only on individual database basis. So, developers have to filter database tables individually by executing multiple queries.
Backup	In SQL Server, while backing up the data, the database is not blocked. This allows users to complete the backup and data restoration process completed in less time and efforts.	Backup of the data can be taken by extricating all the data as SQL statements. During the backup process, the database is blocked. This prevents the instances of data corruption while migrating from one version of MySQL to another. However, increases the total time and efforts in the data restoration process because of running multiple SQL statements.
Option to stop Query Execution	Can truncate the query execution without killing the whole process. It utilizes a transactional engine to hold the state consistent.	Can't cancel or kill the query execution without killing the entire process

Security	Both SQL Server and MySQL are built as binary collections. However, SQL server is more secure than MySQL. It does not let any process to access and manipulate the database files at run time. Users need to perform specific functions or manipulate files by executing an instance. This prevents hackers to access or manipulate the data directly.	It allows developers to manipulate database files through binaries at the run time. It also allows other processes to access and manipulate the database files at the run time.

Syntax difference

There are some variations in the commonly used SQL code in both of these relational database management systems.

Factors	MS SQL Server	MySQL
Length function	`SELECT LEN(data_string) FROM TableName`	`SELECT CHARACTER_LENGTH(data_string) FROM TableName`
Concatenation function	`SELECT ('SQL' + 'SERVER')`	`SELECT CONCAT ('My', 'SQL')`
Select top n records from a table	`SELECT TOP 10 * FROM TableName WHERE id = 2`	`SELECT * FROM TableName WHERE id = 2 LIMIT 10`
Generate GUID (Global Unique Identifier)	`SELECT NEWID()`	`SELECT UUID()`
Get current date and time	`SELECT GETDATE()`	`SELECT NOW()`

Case Sensitive Collation	In SQL Server, if the database is defined with case sensitive collation then the table names and column names become case sensitive. Let us take an example here. Suppose you have created a table in a case sensitive collation database : Create table Engineers(SNo int, EngineerName Varchar(80), Salary money) Observe the capital E in the table name. Now if I run the following query: Select * from engineers Then it will give the following error: Invalid object name ‘engineers’ You need to write the table name in the query in the same case as it was mentioned at the time of table creation: Select * from Engineers	In MySQL, there is no case sensitiveness in identifier names.

Now let’s start with SQL Server. Make sure you have sample databases ready: **AdventureWorks** (full version) and **AdventureWorksLT2012** (lightweight version

Module 2 - Introduction to Database Modelling

Chapter 1: Database Introduction

Theory: In this module, you will be introduced to database modelling and key concepts

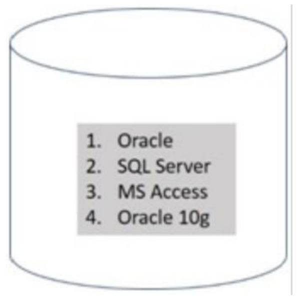
What is Database?

A collected information which is in an organized form for easier access, management, and various updating is known as a database.

Before going into a further discussion of databases, we must have a prior knowledge of exactly what is a DATA? Data can be defined as a collection of facts and records on which we can apply reasoning or can-do discussion or some calculation. The data is always easily available and is in plenty. It can be used for processing some useful information from it. Also, it can be redundant, and can be irrelevant. Data can exist in the form of graphics, reports, tables, text, etc. that represents every kind of information, that allows easy retrieval, updating, analysis, and output of data by systematically organized or structured repository of indexed information.

Containers having a huge amount of data are known as databases, for example, a public library stores books. Databases are computer structures that save, organize, protect, and deliver data.

Any system that manages databases is called a database management system, or DBM. The typical diagram representation for a database is a cylinder.



Inside a database, the data is recorded in a table which is a collection of rows, columns, and it is indexed so that finding relevant information becomes an easier task. As new information is added, data gets updated, expanded and deleted. The various processes of databases create and update themselves, querying the data they contain and running applications against it.

There are several different types of database models that have been developed so far, for example, flat, hierarchical, network and relational. These models describe the operations that can be performed on them as well as the structure of the conforming databases. Normally there is a database schema which describes the exact model, entity types, and relationships among those entities.

Flat Databases have the following characteristics −

- simple
- long and dominant
- useful for very small scale and simple applications.

A **Relational Database** has the following characteristics −

- organizes data such that it appears to the user to be stored in a series of interrelated tables
- used for high-performance applications
- efficient
- ease of use
- ability to perform a variety of useful tasks

Why Do We Need a Database?

When you have some data, and you want to store this data somewhere. This data could be anything. It could be about customers, products, employees, orders, …etc. This data could be in text format, numeric, dates, document files, images, audio, or video.

Maybe if you have data about the customers in your company, the first thing that comes in mind is to open a spreadsheet. Then you start to write whatever the data you want to store.
It could be the customer name, id, position, and so on. You may add as many as customers, delete any of them later, or even modify on them.

Now, we have kind of data, and you stored them in spreadsheets in a way that satisfies your needs. That might be OK, because just having data is not a good enough reason to need a database, and it's not the problem. The problem is what comes next, and there's a lot of potential problems.

What if you have a bunch of data, maybe 10,000 customer, "Are you going to scroll down in the spreadsheet to get the 9999 customer?!", What if the security was a concern, "Do you care

about if someone else got an access to your data?", What if you accidentally put a redundant information, "Is it fine to have duplicate information along the spreadsheet?".
This takes us to the next question, "When do we actually need a database?". Consider the following potential problems:

- **Size**: You may have thousands or million rows of customers, or any piece of information.
- **Accuracy**: "Do you care if someone entered incorrect data?". If yes, nothing could actually prevent me from typing incorrect data into a spreadsheet.
- **Security**: If the data is sensitive, and you need to restrict the access to the data; It doesn't need to be shared with everyone. In addition, "Do you need to know who made every change at every point?".
- **Redundancy**: If the redundant data (having multiple copies of the same data) will lead to conflict, you would need to have only a non-repeated unique data.
- **Importance**: "What if you had a disconnect or a crash, and you lost your data?". You've probably felt that pain before. And it's unacceptable to lose important data like orders of a customer, allergies of a patient, flight bookings, …etc.
- **Overwriting**: How about having more than one person overwriting the same data at the same time. How about 10 at the same time or 100 people at the same time?. You'll end up with everybody overwriting everybody else's changes.

Database Management System (DBMS)

We often mistakenly say our database is Oracle, MySQL, SQL Server, MongoDB. But, they aren't databases, they are database management systems (DBMS). The DBMS is the software that would be installed on your personal computer or on a server, then you would use it to manage one or more databases.

The database has your actual data and the rules about that data, while the DBMS is the program that surrounds and manages your actual data, and it enforces the rules you specified on your data. The rules for example could be the type of the data, like integer or string, or the relationship between them.

In practice it's very common to have multiple databases. The database that deals with your order and customer information might be completely independent from your database that deals with human resource information. And in many organizations, you don't just have multiple databases but multiple DBMS. Sometimes it's because one DBMS is better at something than the other.

There are different DBMS, and they are categorized under:

- Relational Database Management Systems
- Hierarchical Database Systems
- Network Database Systems

- Object-Oriented Database Systems
- NoSQL Database Systems

We are going to focus on the relational database management systems (RDBMS). Relational databases are:

- the most common used one.
- the principles we are going to discuss here are usable across all of them.
- If you know you are going to jump into NoSQL databases, most of the introductions assume you already understand relation database concepts and will use these concepts to explain what's offered by NoSQL databases.

  RDBMS are like Oracle, MySQL, SQL Server, SQLite, DB2, …etc.

OLTP and OLAP

OLTP and OLAP are approaches to processing data, and they help define the way data is going to flow, be structured, and stored. If you figure out which fits your business case, designing your database will be much easier. OLTP stands for Online Transaction Processing. OLAP stands for Online Analytical Processing. As the names hint, the OLTP approach is oriented around transactions, while the other is oriented around analytics.

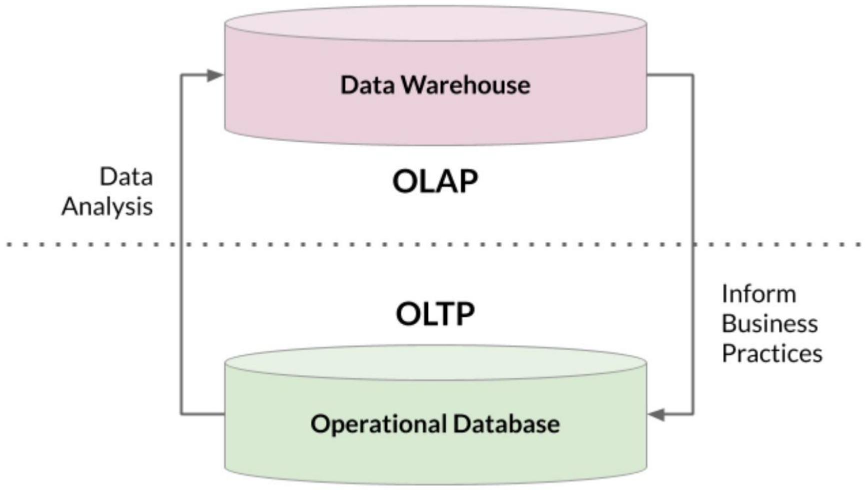
Examples: Let's look at some examples of how OLTP and OLAP can be applied

Some examples of OLTP tasks: ● Find the price of a book ● Update latest customer transaction ● Keep track of employee hours	Some examples of OLAP tasks: ● Calculate books with best profit margin ● Find most loyal customers ● Decide employee of the month

	OLTP	**OLAP**
Purpose	support daily transactions	report and analyze data
Design	application-oriented	subject-oriented
Data	up-to-date, operational	consolidated, historical
Size	snapshot, gigabytes	archive, terabytes

Queries	simple transactions & frequent updates	complex, aggregate queries & limited updates
Users	thousands	hundreds

OLAP and OLTP systems work together; in fact, they need each other. OLTP data is usually stored in an operational database that is pulled and cleaned to create an OLAP data warehouse. Without transactional data, no analyses can be done in the first place. Analyses from OLAP systems are used to inform business practices and day-to-day activity, thereby influencing the OLTP databases.



Structuring Data

Data can be stored in three different levels. The first is structured data, which is usually defined by schemas. Data types and tables are not only defined, but relationships between tables are also defined, using concepts like foreign keys. The second is unstructured data, which is schemaless and data in its rawest form, meaning it's not clean. Most data in the world is unstructured. Examples include media files and raw text. The third is semi-structured data, which does not follow a larger schema, rather it has an ad-hoc self-describing structure. Therefore, it has some structure. This is an inherently vague definition as there can be a lot of variation between structured and unstructured data. Examples include NoSQL, XML, and JSON.

Examples: Let's look at some examples of how you would structure data

Structured data	**Unstructured data**	**Semi-structured data**
Follows a schema Defined data types & relationships *e.g., SQL, tables in a relational database*	Schemaless Makes up most of data in the world *e.g., photos, chat logs, MP3*	Does not follow larger schema Self-describing structure *e.g., NoSQL, XML, JSON*

Because it's clean and organized, structured data is easier to analyze. However, it's not as flexible because it needs to follow a schema, which makes it less scalable. These are trade-offs to consider as you move between structured and unstructured data.

Practice time!: Using Microsoft SQL Server Management Studio, complete the exercises described in the relevant section of your Google Forms Exercise Sheet for this Chapter before moving to the next chapter. Please go to your google classroom to find the form link and do the exercise
Steps:

- Read the question
- Go to Microsoft SQL Server Management Studio
- Create your script on workbench
- Hit execute
- Get result
- Use that result to answer the questions in the Google Form

If you get stuck please ask your support team for help.

Chapter 2: Database Design

What is Database Design?

Database design determines how data is logically stored. This is crucial because it affects how the database will be queried, whether for reading data or updating data. There are two important concepts to know when it comes to database design: Database models and schemas.

Database models are high-level specifications for database structure. The relational model, which is the most popular, is the model used to make relational databases. It defines rows as records and columns as attributes. It calls for rules such as each row having unique keys.

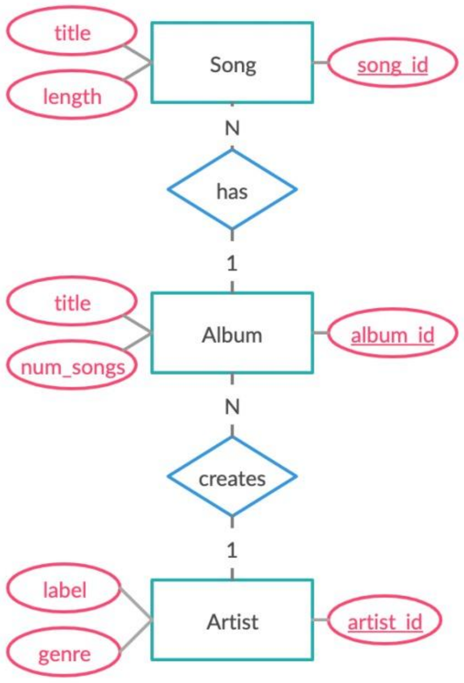
There are other models that exist that do not enforce the same rules. A schema is a database's blueprint. In other words, the implementation of the database model. It takes the logical structure more granularly by defining the specific tables, fields, relationships, indexes, and views a database will have. Schemas must be respected when inserting structured data into a relational database.

Data Modeling

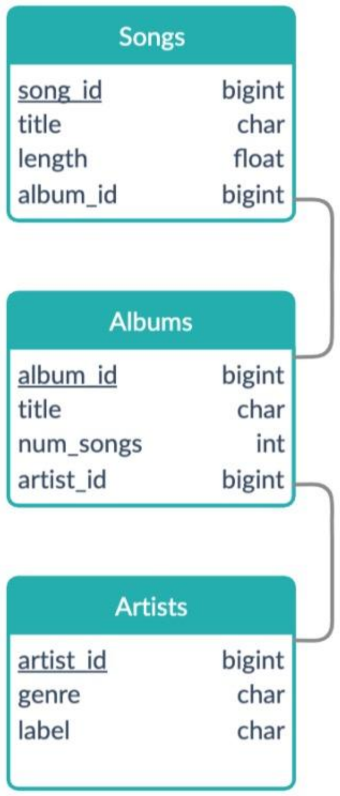
The first step to database design is data modeling. This is the abstract design phase, where we define a data model for the data to be stored.

There are three levels to a data model:

- **A conceptual data model** describes what the database contains, such as its entities, relationships, and attributes.



- **A logical data model** decides how these entities and relationships map to tables.



Songs	
song_id	bigint
title	char
length	float
album_id	bigint

Albums	
album_id	bigint
title	char
num_songs	int
artist_id	bigint

Artists	
artist_id	bigint
genre	char
label	char

- **A physical data model** looks at how data will be physically stored at the lowest level of abstraction.

These three levels of a data model ensure consistency and provide a plan for implementation and use.

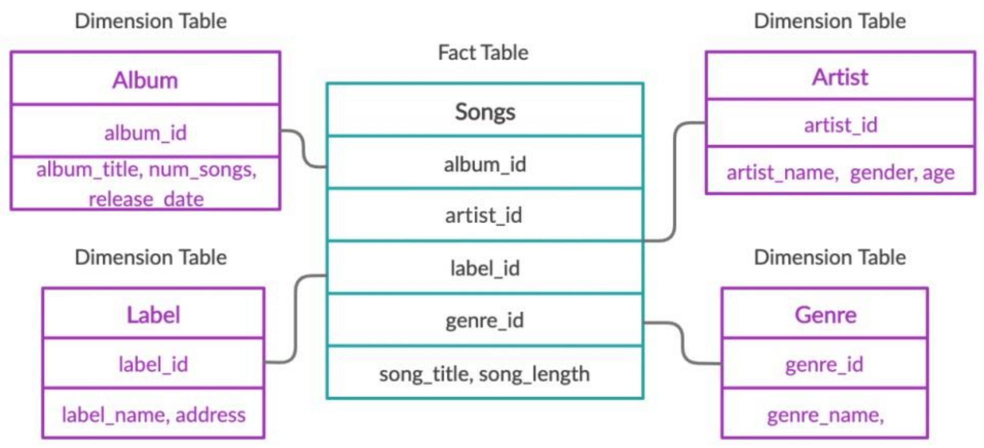
Elements of dimensional modeling

Dimensional models are made up of two types of tables: **fact** and **dimension** tables.

Fact table holds is decided by the business use-case. It contains records of a key metric, and this metric changes often. Fact tables also hold foreign keys to dimension tables.

Dimension tables hold descriptions of specific attributes and these do not change as often.

Examples: Let's look at some examples of how you would analyse songs



The turquoise table is a fact table called songs. It contains foreign keys to purple dimension tables. These dimension tables expand on the attributes of a fact table, such as the album it is in and the artist who made it. The records in fact tables often change as new songs get inserted. Albums, labels, artists, and genres will be shared by more than one song - hence records in dimension tables won't change as much.

Summing it up, to decide the fact table in a dimensional model, consider what is being analyzed and how often entities change.

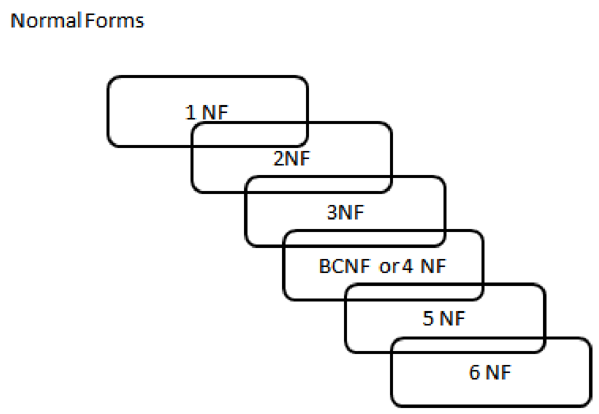
Chapter 3: Normalization Relationship / Carnality

What is Normalization?

Normalization is a technique that divides tables into smaller tables and connects them via relationships. The goal is to reduce redundancy and increase data integrity. So how does this happen? There are several forms of normalization, which we'll delve into later. But the basic idea is to identify repeating groups of data and create new tables for them. Let's go back to our example and to see how these tables were normalized.

The database normalization process is further categorized into the following types:

- First Normal Form (1 NF)
- Second Normal Form (2 NF)
- Third Normal Form (3 NF)
- Boyce Codd Normal Form or Fourth Normal Form (BCNF or 4 NF)
- Fifth Normal Form (5 NF)
- Sixth Normal Form (6 NF)



One of the driving forces behind database normalization is to streamline data by reducing redundant data. Redundancy of data means there are multiple copies of the same information spread over multiple locations in the same database.

The drawbacks of data redundancy include:

1. Data maintenance becomes tedious – data deletion and data updates become problematic
2. It creates data inconsistencies
3. Insert, Update and Delete anomalies become frequent. An update anomaly, for example, means that the versions of the same record, duplicated in different places in the database, will all need to be updated to keep the record consistent

4. Redundant data inflates the size of a database and takes up an inordinate amount of space on disk

Normal Forms

First Normal Form (1NF):

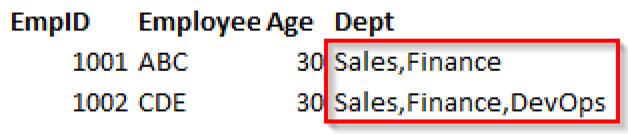
The first normal form requires that a table satisfies the following conditions:

- Rows are not ordered
- Columns are not ordered
- There is duplicated data
- Row-and-column intersections always have a unique value
- All columns are "regular" with no hidden values

Examples: Let's look at some examples of normal forms

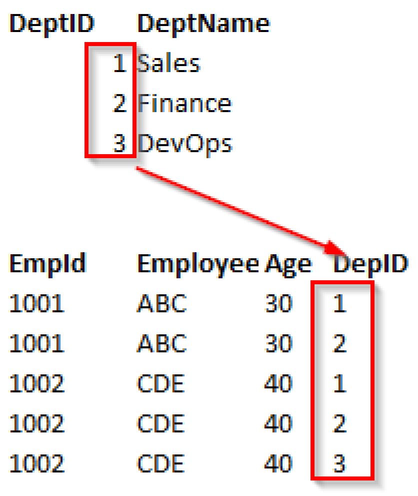
In the following example, the first table clearly violates the 1 NF. It contains more than one value for the Dept column. So, what we might do then is go back to the original way and instead start adding new columns, so, Dept1, Dept2, and so on. This is what's called a repeating group, and there should be no repeating groups. In order to bring this First Normal Form, split the table into the two tables. Let's take the department data out of the table and put it in the dept table. This has the one-to-many relationship to the employee table.

Let's take a look at the employee table:



EmpID	Employee	Age	Dept
1001	ABC	30	Sales,Finance
1002	CDE	30	Sales,Finance,DevOps

Now, after normalization, the normalized tables Dept and Employee looks like below:



DeptID	DeptName
1	Sales
2	Finance
3	DevOps

EmpId	Employee	Age	DepID
1001	ABC	30	1
1001	ABC	30	2
1002	CDE	40	1
1002	CDE	40	2
1002	CDE	40	3

Second Normal Form and Third Normal Form are all about the relationship between the columns that are the keys and the other columns that aren't the key columns.

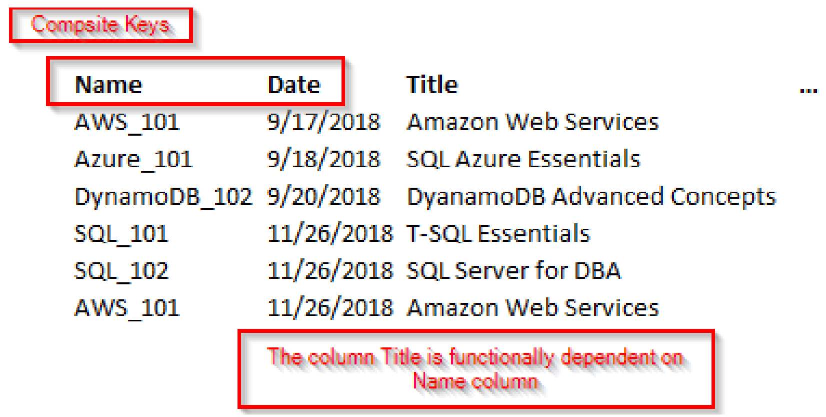
Second Normal Form (2NF):

An entity is in a second normal form if all of its attributes depend on the whole primary key. So this means that the values in the different columns have a dependency on the other columns.

- The table must be already in 1 NF and all non-key columns of the tables must depend on the PRIMARY KEY
- The partial dependencies are removed and placed in a separate table

Note: Second Normal Form (2 NF) is only ever a problem when we're using a composite primary key. That is, a primary key made of two or more columns.

The following example, the relationship is established between the Employee and Department tables.

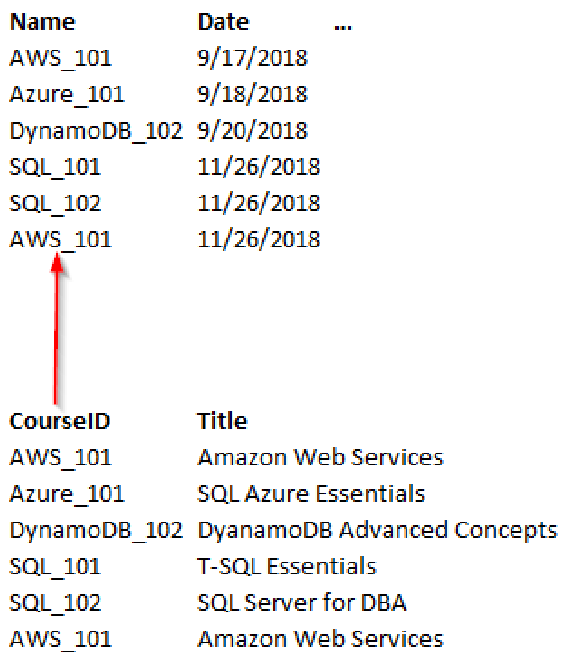


Compsite Keys

Name	Date	Title	...
AWS_101	9/17/2018	Amazon Web Services	
Azure_101	9/18/2018	SQL Azure Essentials	
DynamoDB_102	9/20/2018	DyanamoDB Advanced Concepts	
SQL_101	11/26/2018	T-SQL Essentials	
SQL_102	11/26/2018	SQL Server for DBA	
AWS_101	11/26/2018	Amazon Web Services	

The column Title is functionally dependent on Name column

In this example, the Title column is functionally dependent on Name and Date columns. These two keys form a composite key. In this case, it only depends on Name and partially dependent on the Date column. Let’s remove the course details and form a separate table. Now, the course details are based on the entire key. We are not going to use a composite key.



Name	Date	...
AWS_101	9/17/2018	
Azure_101	9/18/2018	
DynamoDB_102	9/20/2018	
SQL_101	11/26/2018	
SQL_102	11/26/2018	
AWS_101	11/26/2018	

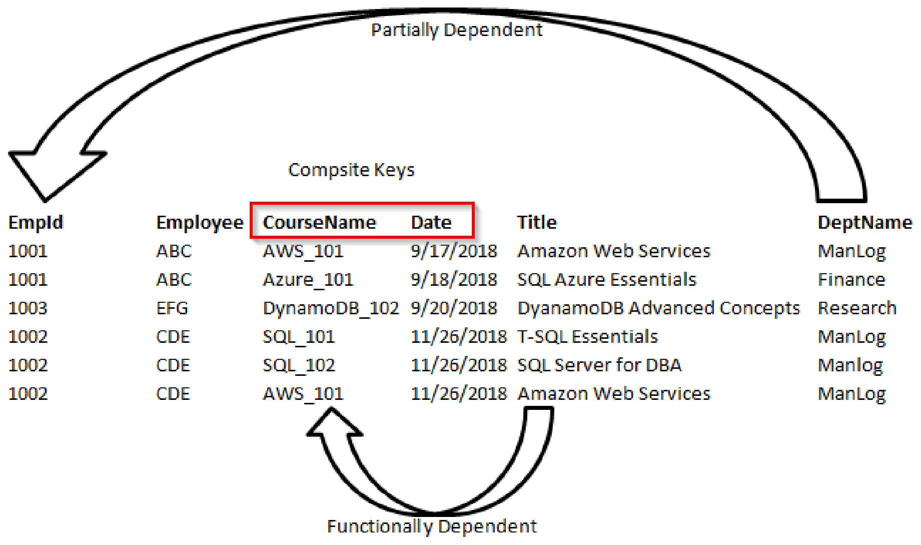
CourseID	Title
AWS_101	Amazon Web Services
Azure_101	SQL Azure Essentials
DynamoDB_102	DyanamoDB Advanced Concepts
SQL_101	T-SQL Essentials
SQL_102	SQL Server for DBA
AWS_101	Amazon Web Services

Third Normal Form (3NF):

The third normal form states that you should eliminate fields in a table that do not depend on the key.

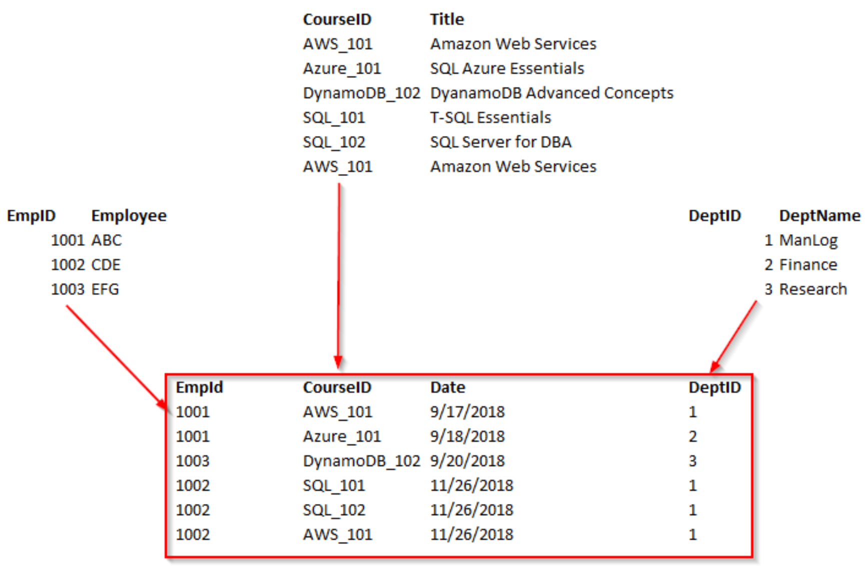
- A Table is already in 2 NF
- Non-Primary key columns shouldn’t depend on the other non-Primary key columns
- There is no transitive functional dependency

Consider the following example, in the table employee; empID determines the department ID of an employee, department ID determines the department name. Therefore, the department name column is indirectly dependent on the empID column. So, it satisfies the transitive dependency. So this cannot be in third normal form.



EmpId	Employee	CourseName	Date	Title	DeptName
1001	ABC	AWS_101	9/17/2018	Amazon Web Services	ManLog
1001	ABC	Azure_101	9/18/2018	SQL Azure Essentials	Finance
1003	EFG	DynamoDB_102	9/20/2018	DyanamoDB Advanced Concepts	Research
1002	CDE	SQL_101	11/26/2018	T-SQL Essentials	ManLog
1002	CDE	SQL_102	11/26/2018	SQL Server for DBA	Manlog
1002	CDE	AWS_101	11/26/2018	Amazon Web Services	ManLog

In order to bring the table to 3 NF, we split the employee table into two.



CourseID	Title
AWS_101	Amazon Web Services
Azure_101	SQL Azure Essentials
DynamoDB_102	DyanamoDB Advanced Concepts
SQL_101	T-SQL Essentials
SQL_102	SQL Server for DBA
AWS_101	Amazon Web Services

EmpID	Employee
1001	ABC
1002	CDE
1003	EFG

DeptID	DeptName
1	ManLog
2	Finance
3	Research

EmpId	CourseID	Date	DeptID
1001	AWS_101	9/17/2018	1
1001	Azure_101	9/18/2018	2
1003	DynamoDB_102	9/20/2018	3
1002	SQL_101	11/26/2018	1
1002	SQL_102	11/26/2018	1
1002	AWS_101	11/26/2018	1

Now, we can see that all non-key columns are fully functionally dependent on the Primary key.

Although a fourth and fifth form does exist, most databases do not aspire to use those levels because they take extra work and they don't truly impact the database functionality and improve performance.

Denormalization

Database normalization is always a starting point for denormalization. Denormalization is a type of reverse engineering process that can apply to retrieve the data in the shortest time possible.

Denormalization is a strategy used on a previously-normalized database to increase performance. In computing, denormalization is the process of trying to improve the read performance of a database, at the expense of losing some write performance, by adding redundant copies of data or by grouping data. It is often motivated by performance or scalability in relational database software needing to carry out very large numbers of read operations.

NOTE: Denormalization should not be confused with Unnormalized form. Databases/tables must first be normalized to efficiently denormalize them.

For additional explanation go here: https://www.youtube.com/watch?v=2OuhVHJGuS0

Cardinality

Relationship

As you already know, a database can consist of many tables, containing certain information. Basically, the information you store in it (in the database) has one direction, for example, contains information about the enterprise work, data about the students and teachers of the Academy, and so on. This information can be logically placed in different tables, but each of these tables will be autonomous and will not depend on others.

How to link them?

As you already know, each relational table should contain a primary key that uniquely characterizes each of its records. But in addition to this function, it allows you to implement the relationships between tables, through which data from one table become available for another table. In this case, if the database contains several tables, it operates more efficiently, data entry in the table becomes easier, the error probability is reduced. Each of the tables contains one field, which has the same value; the relationships are implemented thanks to this. By the way, these fields may have different names, but the unique conformity of the data should be met. A foreign key in one table points to a primary key in another table.

Thus, the foreign key is a field of the table whose values match the value of the field, which is a primary key of another table.

Therefore, thanks to a pair "primary key — foreign key", relationships between data of two tables arise.

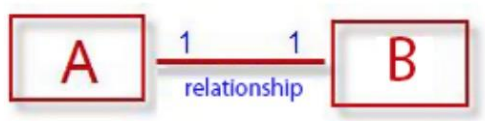
Relationship Types

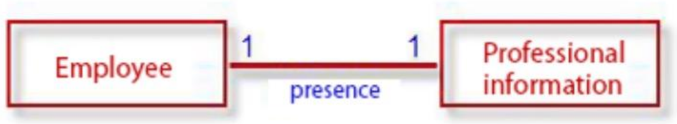
Relationship is a way to explain to the DBMS how to select the information from the database tables.

Examples: Let’s look at some examples relationship types and what they mean

There are four types of relationships between tables:

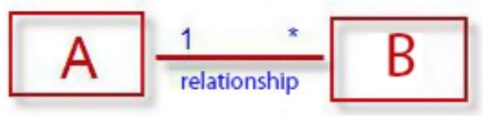
ONE-TO-ONE RELATIONSHIPS (1:1): each entry in the table A can have no more than one matching entry in the table B, and vice versa. This type of relationship is not common because almost all data may be placed only in one table. It may be useful when we need to divide one bulky table. Schematically it can be represented as follows:

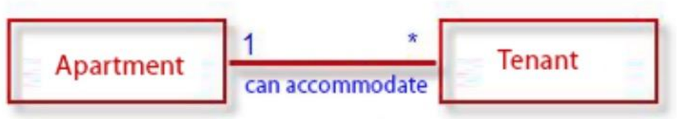




Example: there are two tables, one of which contains data about the employees of the enterprise, and the second one includes the professional information about them. In this case, we can say that there is the one-to-one relationship between these tables, because one person (in the first table) may correspond to only one record comprising professional information in the second table.

ONE-TO-MANY RELATIONSHIPS (1:M): one entry in the table A can be related to many entries in the table B, but an entry in the table B can have only one matching entry in the table A. Here is a schematic representation of the relationship:





Example: an apartment can be empty, or one or more tenants can live in it.

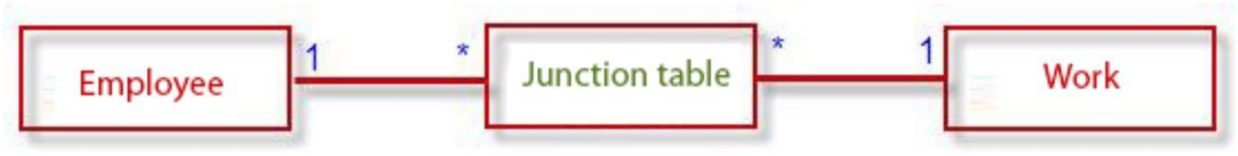
MANY-TO-ONE RELATIONSHIPS (M:1): — the opposite to the previous. Relationship type between objects depends on your point of view. If we consider the ratio of tenants to the apartment, then a relationship many-to-one will be formed.

MANY-TO-MANY RELATIONSHIPS (M:M): This relationship arises between two tables in cases, when:

- One entry from the first table may be associated with more than one entry from the second table;
- One entry from the second table may be associated with more than one entry from the first table.

Such a relationship is uncommon, but if it exists, you can create this one by defining a third table, called a junction table, that consists of the foreign keys only and join two tables.

Example: an enterprise where employees work (one or more). In addition, we can face with a situation when the employees work in several enterprises and this information is stored in the database. In this case, between the tables (Work and Employees), there is the many-to-many relationship.



Practice time!: Using Microsoft SQL Server Management Studio, complete the exercises described in the relevant section of your Google Forms Exercise Sheet for this Chapter before moving to the next chapter. Please go to your google classroom to find the form link and do the exercise
Steps:

- Read the question
- Go to Microsoft SQL Server Management Studio
- Create your script on workbench
- Hit execute
- Get result
- Use that result to answer the questions in the Google Form

If you get stuck please ask your support team for help.