Northeastern University

ALY6030

Data Warehousing and SQL

Instructor: Valeriy Shevchenko

Northeastern

Lecture 1 (Week 1)

The Relational Model and Basic SQL

Instructor: Valeriy Shevchenko

Northeastern University

Learning Objectives

This session will explain:

- What is SQL
- Concepts of relational databases.
- Data modeling and entity relationships diagrams.
- SQL fundamentals.
- Basic SQL queries.

Relational Databases and SQL

- Relational databases have been used to store large amounts of data for the past couple of decades.
- Often when someone refers to a database, they are talking about a relational database.
- Since so much data is stored in relational databases, it’s important to understand how they work.
- More importantly for an analytical role, it is important to know how to get data out of them.
- Until now, we have manipulated data that is stored in accessible files (.xlsx, .csv) using tools like Excel and R.

Relational Databases and SQL

- With a relational database, we can not easily access the data by opening the file and inspecting it.

- Once we learn query language, however, we will be able to pull the data that we want out of a database and subsequently use it in any of the analysis tools with which we have already become familiar.

- Because databases can be so complex, with large amounts of data stored in many tables, we also need an understanding of how to join, filter and manipulate data stored in a database to extract only the data of interest.

- This allows us to get the data we need in a form that makes it easy to use.

Relational Databases and SQL

What is SQL?

Anyone?

Relational Databases and SQL

- SQL is an abbreviation for **Structured Query Language**, and pronounced either "see-kwell" or as separate letters.
- SQL is a standardized query language for requesting information from a database.
- The original version called SEQUEL (structured English query language) was designed by an IBM research center in 1974.
- SQL was first introduced as a commercial database system in 1979 by Oracle Corporation.
- Historically, SQL has been the favorite query language for database management systems running on minicomputers and mainframes.

Relational Databases and SQL

- Increasingly, however, SQL is being supported by PC database systems because it supports distributed databases (databases that are spread out over several computer systems).
- Although there are different dialects of SQL, it is nevertheless the closest thing to a standard query language that currently exists.
- In 1986, ANSI approved a rudimentary version of SQL as the official standard, but most versions of SQL since then have included many extensions to the ANSI standard.
- In 1991, ANSI updated the standard. The new standard is known as SAG SQL.

Dialects of SQL

- PSQL – Procedural SQL: used by Interbase / Firebird Databases.
- T-SQL – Transact SQL: used by Microsoft SQL Server. (former Sybase DB)
- SQL/PSM – SQL/Persistent Stored Module: used by MySQL, MonetDB, MariaDB, etc.
- PL/SQL – Procedural Language/SQL (based on Ada): used by Oracle DB
- PL/pgSQL - Procedural Language/PostgreSQL: used by PostgressSQL
- SPL – Stored Procedural Language: used by Teradata

Relational DBs

- To motivate the need for relational databases, let’s consider the situations where you choose to use Microsoft Word versus Microsoft Excel for a task.
- Assume you have been keeping track of your weight for a period of time.
- Every so often, you take note of your weight, and make a note such as whether you ate healthily or exercised a lot that day.
- If you put this information into a Word document, it would be very easy to input the information - you could just start typing.

Relational DBs

- However, if you want to find the difference in your weight between two different days, you would have to find those two days in the running document and then do the math in your head or with a calculator.

- There are also no constraints on how you enter the data.

- This means you could leave out the note or the date for some of the entries, or you could include additional information, such as the number of calories burned.

- Alternatively, you could put this information into Excel.

Relational DBs

- You might specify one column for the date, one column for your weight, and one column for your comments.
- The specification of what goes into each column makes the data more structured.
- All of the entries in a given column are the same type of data, and all of the entries in a given row are related.
- Suppose you weigh yourself once a week and want to record Date, Weight and some notes (comments)

Relational DBs

- An example of this spreadsheet could look like this:

Date	Weight(lbs.)	Comments
02/01/2019	235	First day of the new life
02/08/2019	235	Junk food excluded
02/15/2019	236	More exercises needed
02/22/2019	234	New set of exercises started
03/01/2019	233	Exercises extended to few new tricks

- In this example, the Excel spreadsheet is a table of data.

- Let's extend the example.
- Consider the case where you have another table, that is, another spreadsheet, where you're keeping track of any exercise you performed in a given week as well as how many calories you burned per day on average that week.

Relational DBs

- In this table, you have the date, a note about the activities or exercise you performed that week, and the average calories you burned per day during that week.

- An example of this table is shown below:

Date	Activities	Avg Calories Burned
02/01/2019	Zumba	1540
02/08/2019	Yoga	1200
02/15/2019	Fitness	1350
02/22/2019	Jogging	2650
03/01/2019	Weights	1000

- There is a **relationship** between these two tables.

Relational DBs

- If you wanted to see all of this information together in one table, you could join the two tables.
- To do so, you have to match up the dates in each table and create a new combined table.
- The result would be one table with both the average number of calories you burned per day during that week and your measured weight on the date listed.

Relational DBs

- An example of the combined tables is shown below:

Date	Weight (lbs.)	Comments	Activities	Avg Calories Burned
02/01/2019	235	First day of the new life	Zumba	1540
02/08/2019	235	Junk food excluded	Yoga	1200
02/15/2019	236	More exercises needed	Fitness	1350
02/22/2019	234	New set of exercises started	Jogging	2650
03/01/2019	233	Exercises extended to few new tricks	Weights	1000

- The two original Excel spreadsheets, "Fitness progress chart" and "Exercise tracking" are an example of data that could be stored in a relational database.

- We have two tables, each storing different information.

Relational DBs

- There is a relationship between the tables:
  - the date allows the matching of rows in one table to rows in the other table.

- To put this in the terminology of databases - the date is known as the **key field**.

- While this example showed how Excel can be used to store data relationally Excel is not a relational database.

- This example, however, demonstrates some of the characteristics of a relational database.

Characteristics of Relational DB

- Good relational databases have the following characteristics:
  - Optimized to store large amounts of data.
  - Designed to store and retrieve data in a **uniform**, **structured** way.
  - Provide a way to efficiently **query** large datasets
  - Can easily **join** or merge data from different datasets to answer questions

Characteristics of Relational DB

- Think of all the information you may want to find out about your business:
  - What types of questions will you need to answer for other stakeholders?
  - What information will you need to generate quarterly or annual reports?
- Some examples of questions you may want to answer about a business:
  - What were our sales goals for the quarter?
  - How have actual sales exceeded or fallen short of our sales goals?
  - Who are our best customers?
  - How many widgets did we sell during this quarter last year?

Characteristics of Relational DB

- Below are some examples of questions you may want to answer about a non-profit:
  - Who are our biggest donors?
  - What events have been most attended by our donors?
  - How many donations have we received from our past three events?
  - What is the zip code or area code where we receive the greatest amount of donations?
- All of these questions can be answered by considering the details of a company broken into its smallest components.

Characteristics of Relational DB

- What is the smallest component of a company?

- There are people (employees), who each have a:
    - first name,
    - last name,
    - title,
    - work address,
    - work phone number,
    - home address,
    - home phone number,
    - department to which they belong.

- You could break this information down further, for example: area code, street address, city, state, and zip code.

Characteristics of Relational DB

- Each of these components could be stored separately.
- Some employees may have customers or clients, and each client will have contact information, a record of past purchases and perhaps a history of each interaction between the employee and the client.
- It is important to take care when deciding how information is stored in a relational database.
- A good starting place is to consider natural high-level groupings, usually nouns, which are important to an organization.

Characteristics of Relational DB

- In the case of a non-profit, the groupings may be: donations, events, employees, volunteers and clients.

- If we were storing information about these groupings in Excel, information about each grouping would be stored in its own worksheet within the Excel workbook.

- There are a few key concepts behind the relational database model.

- We will use Excel to help explain the structure of relational databases.

- Imagine the structure of an Excel workbook.

Characteristics of Relational DB

- In order to help visualize database structures, here is a comparison of what something is called in an Excel spreadsheet and what it is called in the vernacular of relational databases:

Database	• Is analogous to a workbook
Table / Entity)	• Is analogous to a worksheet
Database	• Is a collection of tables as a Workbook is a collection of Worksheets
Record	• Is an instance or tuple is analogous to a row
Attribute	• Is analogous to a column
Entry	• Is analogous to the value in a cell
Tables are linked	• By values for a specified attribute, as worksheets may be linked by values for a specific column

Characteristics of Relational DB

- Think about the rows and columns of an Excel spreadsheet.
- If we want to create a well structured database, we must ensure that each row contains information about only one record, such as one employee's contact information.
- Each column is labeled with a title to describe a single attribute of that record, such as the employee’s first name, last name, title, the department to which the employee belongs and the employee's home phone number.
- In relational databases, it is necessary to make each record unique, so we must also give this employee a unique employee identification number.

Characteristics of Relational DB

- This unique ID is generally represented as the first entry of the record, however the ordering of the attributes does not matter.
- Tables are then connected, linked or related (this is the relational part of a relational database) using the information found in this unique column.
- In the table below, for example, the unique ID would be the region, specified as Northeast, Southeast, Central or West.

January Data	Revenue	Cost	Profit
Northeast	$30,500	$11,000	$19,500
Southwest	$24,000	$7,000	$17,000
Central	$42,500	$15,000	$27,000
West	$32,500	$12,500	$19,500

Definitions of Relational DB

- Now we can formally define relational database terms, previously defined in terms of Microsoft Excel.

- Formal definition of relational database terms:

Database	• A collection of tables
Table	• A collection of records or rows • Each row represents a single record • Each column holds one specific piece of information about that record
Entity	• Object, concept or event – usually a noun
Attribute	• A characteristic of an entity
Record	• Specific characteristics of one instance of an entity in the DB • Each record is made up of multiple attributes • Attributes described in the column headings
Entry	• Specific value of an attribute for a given record

Definitions of Relational DB

Example of a Database?

Anyone?

Northeastern University

Definitions of Relational DB

Example of a Database?

Company ERP

Definitions of Relational DB

Example of a Entity?

Sales Orders

Definitions of Relational DB

Example of a Table?

Anyone?

Definitions of Relational DB

Example of a Table?

SalesOrderHeader
SalesOrderLine
SalesOrderRelease

Northeastern University

Definitions of Relational DB

Example of a Sales Order Attributes?

Anyone?

Definitions of Relational DB

Example of a Sales Order Attributes?

OrderDate
OrderNumber
SoldToCustomer
ShipToCstomer
BillToCustomer

Definitions of Relational DB

Example of a Record (Row)?

Anyone?

Definitions of Relational DB

Example of a Record (Row)?

01/01/2019, 100123, NEU, NEU, NEU

Entity vs Table in Relational DB

- In the relational model, we start by determining the entities.
- What concepts do we care about in this dataset?
  - For example, the sales order is a purchase request or shipment?
- Those entities have attributes, such as:
  - The date order was placed
  - The unique number in the system for the placed order
  - Who is the buyer?
  - Who should we ship it to?
  - Who will be paying for it?

Entity vs Table in Relational DB

- The entities are then represented in the database as a table broken up into attributes which each describe different aspects of the entity.
- Each record in the table represents an instance of that entity, such as the specific Sales Order placed by NEU.
- A record in a table is sometimes referred to as a tuple, which is simply a collection of attribute values for an instance of an entity.
- In a relational database, the ordering of the records (rows) and attributes (columns) is not important.
- Tables names are singular or plural nouns with no spaces in relational databases.
- Attributes are also usually named with no spaces or special characters.

Concepts of Relational DB

- Let’s look at another example, consider the following table:

Customers:

Name	DOB	Address	Phone
John Smith	09/12/1985	5 Hickory Way, Revere, MA 02151	(781) 339-5224
Jane Smith	06/01/1989	52 Washington Drive, Taunton, MA 02718	(508) 724-3671
Ronda Li	05/25/2002	74 Sunset Drive, Mansfield, MA 02031	(508) 541-2134
John Smith	10/12/1978	900 Sparrow St, Revere MA 02151	(339) 862-9071

- This is an example of a table in a database.
- The table is called Customers, and represents the entity "customer.“
- The attributes of the customer entity are Name, DOB, Address and Phone, which are the headers for the attributes of this table.

Concepts of Relational DB

- Let's compare this table to another table that represents a customer entity in a different way:

CustomersRedesigned:

ID	LName	FName	DOB	StreetAddress	City	State	Zip	Phone
756	Smith	John	09/12/1985	5 Hickory Way	Revere	MA	02151	(781) 339-5224
326	Smith	Jane	06/01/1989	52 Washington Drive	Taunton	MA	02718	(508) 724-3671
129	Li	Ronda	05/25/2002	74 Sunset Drive	Mansfield	MA	02031	(508) 541-2134
655	Smith	John	09/12/1985	900 Sparrow St	Revere	MA	02151	(339) 862-9071

- This table is called CustomersRedesigned, and this entity has the following attributes: ID, LName, FName, DOB, StreetAddress, City, State, Zip and Phone.

Concepts of Relational DB

- Considering the first Customers table, how can we uniquely identify a single customer?
- Some customers may have the same name, others customers may have the same DOB.
- John Smith has the same name and DOB, but these records have different addresses. Is this the same customer?
- In the second table, CustomersRedesigned, each customer has a unique attribute: ID.
- Using this attribute, the database user can easily distinguish between two different customers with the same name.

Concepts of Relational DB

- The CustomersRedesigned table is also preferable due to the granularity of the attributes.
- Consider the case where we want to find all of the customers who live in the city of Revere.
- In the first table, we would have to look at each address to determine if "Revere" is somewhere in the text entries of the Address attribute.
- In the second table, the City attribute makes it easy to find which customers are from Revere.
- In other situations, we may be interested in the first name only of some customers, these are easily distinguished when First Name and Last Name are stored as separate attributes.

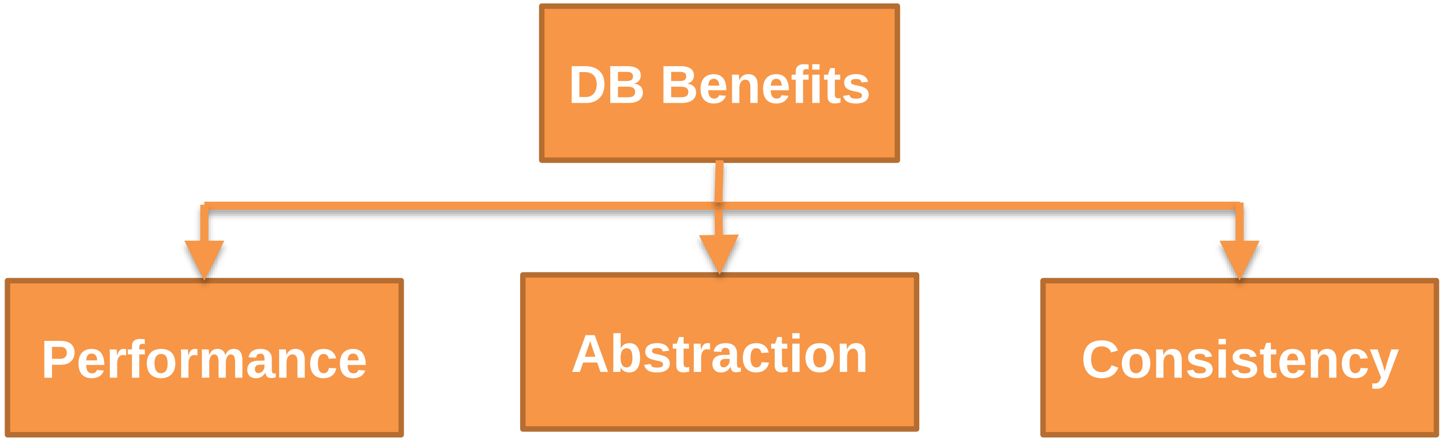
Concepts of Relational DB

- The difference between these two tables is in the **data model**.
- In the first table, each customer is modeled by four attributes, while in the second table, each customer is modeled by the same information plus a unique ID, using nine attributes in total.
- When it comes to the design of a data model, there is not one right answer.
- The best data model is determined based on the context of the problem and how the data is being used.
- There are some best practices for data modeling, and applying these best practices can reduce the storage required for the database and can improve performance for certain database operations.

Benefits of Databases

- Edgar Codd, a computer scientist who worked for IBM, defined these best practices, broadly known as normalization.

- We will learn a more about concepts in database normalization.



- Let's talk broadly about some of the benefits of relational database shown above:

Benefits of Databases

Abstraction

- **Abstraction**: databases help to decouple data use from how and where the data is stored.
- This allows systems to grow and makes them easier to develop and maintain through modularity.
- We will learn a little bit more about this when we discuss the client-server architecture.

Benefits of Databases

Performance

- **Performance**: databases can be tuned or adjusted to improve performance for the task that needs to be done.

- For example, a database which handles credit card transactions may be optimized for many writes, meaning changing or adding new records.

- A database used internally to analyze historic sales data may be optimized to handle many reads, meaning users requesting access to view the data, but not changing or adding new records.

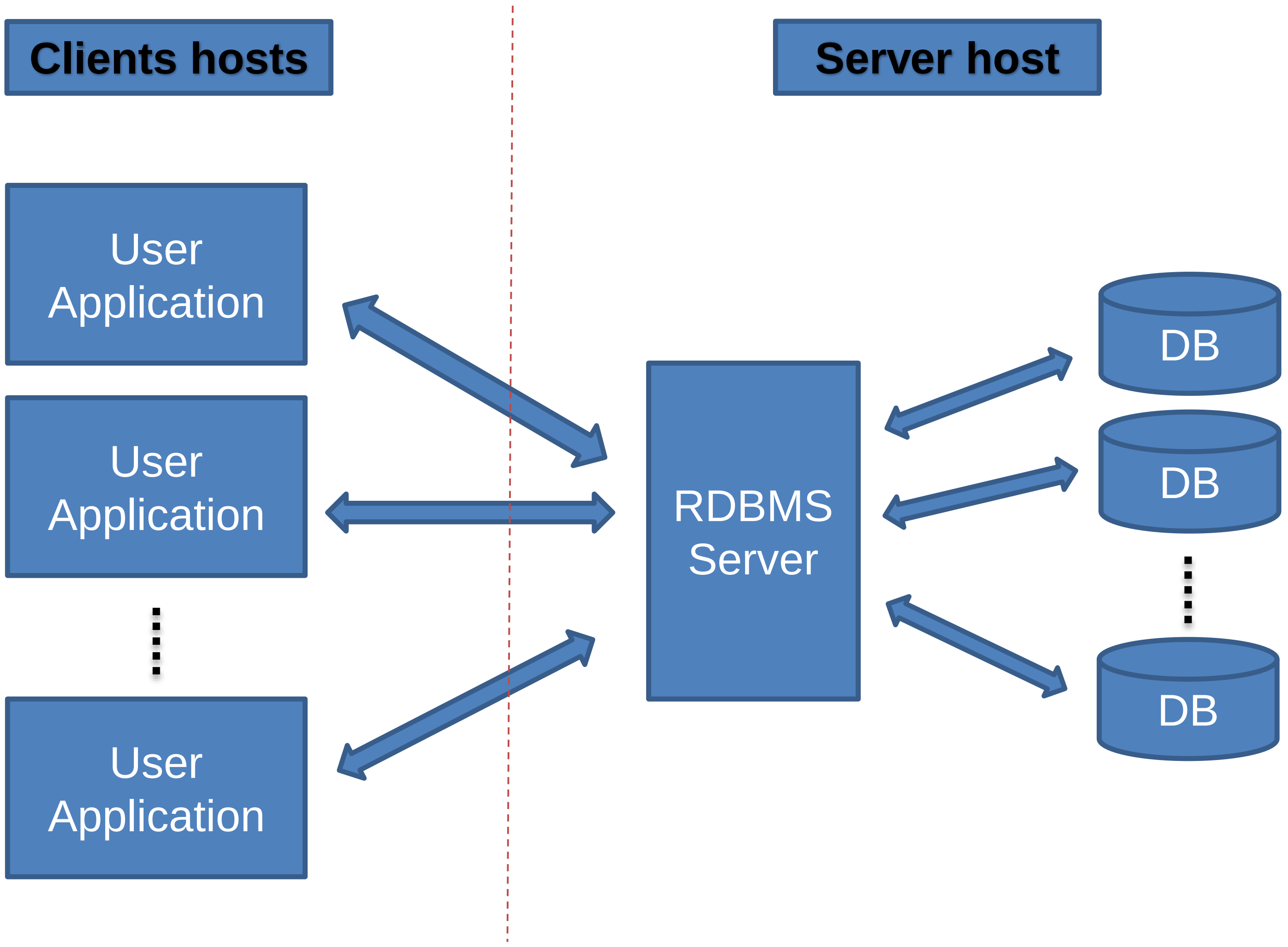
Benefits of Databases

Consistency

- **Consistency:** data is stored in one place and can be accessed and updated by many users or clients.

- The client-server architecture of relational databases will be discussed in more detail.

Client Server Architecture



Client Server Architecture

- Let’s talk about a common network architecture for relational databases, which is responsible for the abstraction, consistency and performance of relational databases.
- Most database systems are designed with a client-server network architecture.
- This means there are two components to the system:
  - The client.
  - The server.
- The client is the front-end, where a user (such as a data analyst) opens an application, which acts as the client, and uses this application to connect to a database.
- Once the connection is made, the user may run some queries or otherwise interact with a database.

Client Server Architecture

- The server is the back-end; this stores the actual database and its settings.
- The server stores the data and manages connections to the database.
- Managing connections means that the server determines whether a given client is allowed to log into the database, what data the client is allowed to see, and how the client is allowed to manipulate the database
  - read from tables.
  - update tables.
  - delete tables.
- Multiple clients can log on to the server and access data at the same time.

Client Server Architecture

- Another example of a client-server architecture is web browsers and web servers.
- Your web browser, such as Chrome or Firefox, is a client.
- It has features which allow you to interact with the Internet.
- When you navigate to a URL, it looks up that URL and connects to a web server, which allows your web browser to display all of the content on that page.
- The server allows other computers to see that content at the same time.
- A client-server architecture is used because it supports the following key database benefits: abstraction, performance, and consistency.

Client Server Architecture

- This architecture supports abstraction by allowing the users to connect to databases of any size, regardless of the storage available on the computers with the client application.
- Similarly, the performance of the database can be tuned on the server side.
- The performance will be independent of the computer accessing the database through the client application.
- Finally, because the data is stored separately on the server, each client will see the most up to date version of the database, users will not need to worry about accessing an older version of the dataset.

Client Server Architecture

- When you think about a client-server architecture, you might think that you need at least two computers:
    - one for the client;
    - one for the server;
- A client-server architecture can also exist using two different processes on a single computer.
- This means that we can run a client-server architecture on a laptop.
- Though this is not realistic in practice, it can be useful for learning and local development and testing.
- This requires running two programs at once.
    - First we must start the server.
    - Then we can launch the client.
    - Finally, we must let the client know the location of the server.

Client Server Architecture

- Running a database locally, on a laptop for example, is nice for when we are learning about databases.

- In these situations, we have complete control over both the server and the client. We can control which data is in the database, as well as when the server is running and when it is not running.

- In practice, such as when working with a company's database, there are one or more database servers which run on computer(s) which are separate from those running instances of the client application.

- Typically an IT professional provides users with the necessary connection information and information about how to connect to the server.

Client Server Architecture

- Similarly, in the example of the web browser, we need a URL, username, and password to allow us to log into a private website.

- There are a variety of different relational database management systems (RDMS) available.

- Some of the more traditional (and proprietary) relational database systems include Oracle Database and Microsoft SQL Server.

- There are also powerful, open-source options, which are popular for personal work and start-ups, including MySQL and PostgreSQL.
- Amazon Web Services (AWS) offers some of these systems as managed, cloud-based software, as well as their own RDMS solution, Aurora.

Client Server Architecture

- For this week, we will use MySQL as the DB server, and MySQL Workbench as the database client.
- MySQL is widely available, free and easy to use.
- The client tool, MySQL Workbench, is nice for creating SQL queries, managing connections and performing data modeling.
- Workbench has a good graphical user interface (GUI) to use when interacting with databases.
- Database GUIs which are not always user-friendly, however MySQL Workbench provides a fairly user-friendly experience.

Client Server Architecture

- Using the instructions provided in the course materials, you will install the software on your computer.
- When working with MySQL locally:
  - We will make sure the server is running (MySQL)
  - We will launch the client (MySQL Workbench)
  - We will connect to the server with the client.
- MySQL Workbench has a tool which allows users with the right privileges to start the server with the Workbench GUI.

Data Modeling

- Understanding the basics of data modeling is necessary before we can build a relational database to store our data.
- Data modeling requires figuring out all of the data that needs to be stored, determining the format of these data and anticipating what analyses or tasks database users may need to perform with the data.
- In general, there is not a "correct" data model for a given dataset, however there are bad, good and better data models.
- For example, bad data models may have the following characteristics:
    - Stores a single piece of data multiple times;
    - Makes it difficult to access the data that users care about;

Data Modeling

- In a business setting, it is often difficult and time-consuming to change a relational database after it has been designed, tested, piloted, validated and deployed and populated with data.
- Spending the time to design a good data model up-front gives much better chances of avoiding a time-consuming redesign process.
- As we’ve discussed previously, the unstructured data was stored in the Word document, while the Excel spreadsheet provided structure to the data, storing data in columns and rows (which represent records or instances of an entity).

Data Modeling

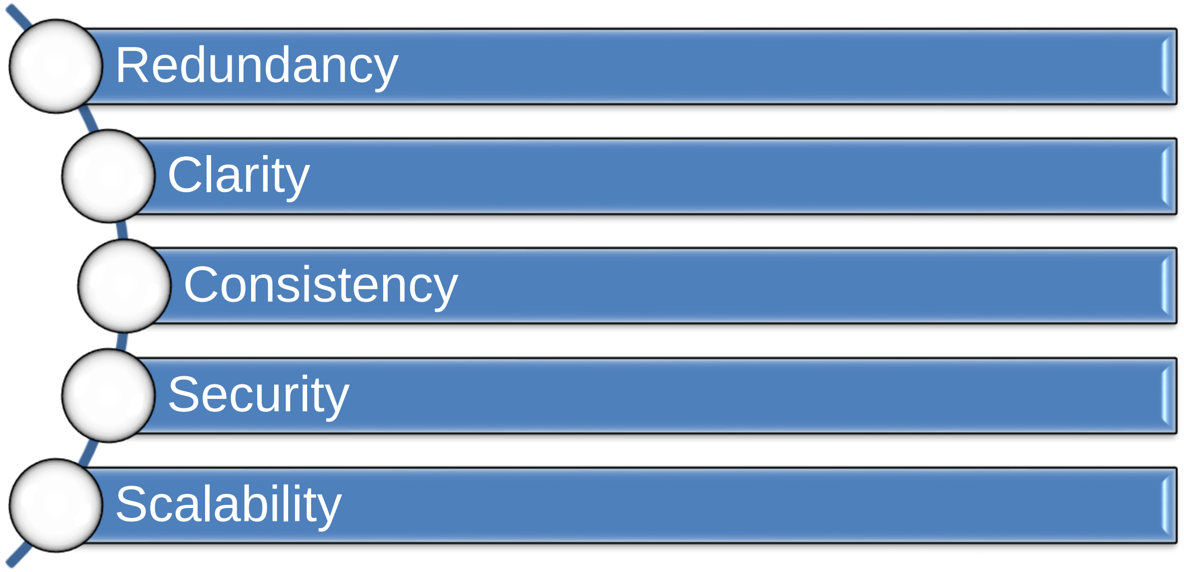
- In the world of data analytics, unstructured data refers to more than just a single Word document.
- Unstructured data may be comprised of a large number of documents.
- These documents may or may not be of the same type or in the same format.
- For example, unstructured data could be stored in a series of Word documents, PowerPoint files, .pdf documents, Excel files, .csv files and web pages.
- In unstructured data, constraints on how data is entered, stored, updated and deleted may not exist.

Data Modeling

- Conversely, structured data refers to any data storage method where we can easily understand the data model.
- There will be purposeful constraints on how data is entered, stored, updated and deleted.
- Typically structured data refers to a relational database, but it can also refer to a well-organized Excel spreadsheet or .csv file.
- Unstructured data storage has some benefits. For example, users can enter data into a document without having to think about how they want to organize it.
- This makes the process of entering data quick and easy.

Characteristics of Datasets

- When it comes to data analytics, however, structured data is generally preferable.
- There are five characteristics of datasets, which we will discuss in terms of structured versus unstructured data:



- These characteristics can be used to assess the quality of a relational database and its associated data model.

Characteristics of Datasets

Redundancy

- Refers to how often a single piece of data is repeated in a dataset.
- In other words: data redundancy is a condition created within a database or data storage technology in which the same piece of data is held in two or more separate places.
- This can mean:
  - Two different fields within a single database;
  - Two different spots in multiple software environments or platforms.
- Whenever data is repeated, this basically constitutes data redundancy. This can occur by accident or can be done deliberately for backup and recovery purposes.

Characteristics of Datasets

Clarity

- Refers to how easy it is to understand the relationships between different elements or entities in the dataset.
- There is should be a balance between the number of entities (tables) and their granularity.
- Relations should be and can be as granular as possible to reflect current or future business needs.

Characteristics of Datasets

Consistency

- Consistency refers to the ability to ensure that a dataset has the correct value for a given entry.
- Many systems in today's environments use and/or collect the same source data.
- Regardless of what source collected the data or where it resides, it cannot contradict a value residing in a different source or collected by a different system.
- There must be a stable and steady mechanism that collects and stores the data without contradiction or unwarranted variance.

Characteristics of Datasets

Security

- Security refers to the ability to control who has access to a given set, subset or element of data.
- This characteristic can be tricky at times due to legal and regulatory constraints.
- Regardless of the challenge, though, individuals need the right level of access to the data in order to perform their jobs.
- This presumes that the data exists and is available for access to be granted.

Characteristics of Datasets

Scalability

- Scalability refers to the ability to allow the dataset to grow in size and to grow quickly.
- Scalability can also refer to the ability to provide consistent access to a dataset for a growing number of users.
- One of the important tasks is to reserve enough data type space for attributes, for example, ones that auto incremented, to make sure enough data can be stored before value overflows.
- Another example is date format and Y2K issue which resulted in significant changes for data, databases and software.

Characteristics of Datasets

- Summary of some of the benefits of structured datasets and drawbacks of unstructured datasets.

DB Characteristic	Structured - e.g. Well-designed relational database	Unstructured - e.g. One or more documents, spreadsheets or files
Redundancy	Each piece of data or entry is only stored once	Data can be represented multiple times in a file
Clarity	Easy to understand the relationships between different pieces of data or entries	Difficult to determine and collect related data, difficult to understand relationships between data
Consistency	Easy to update data, updates will occur for all Users	Difficult to ensure that an entry has the same value in all locations where it appears in files or spreadsheets

Characteristics of Datasets

- Summary of some of the benefits of structured datasets and drawbacks of unstructured datasets. (CONT'D)

DB Characteristic	Structured - e.g. Well-designed relational database	Unstructured - e.g. One or more documents, spreadsheets or files
Security	Ability to provision specific privileges to specific users for each part of the dataset: • some users can read; • some users can read and update; • some users can read only a subset of the dataset)	Can't easily control multi-user access to a file
Scalability	Designed to scale	Difficult to maintain version control if there are hundreds or thousands of users, vendors, brokers or stores accessing the same data

"Good" Data Model

Complete

- All necessary data is represented

No Redundancy

- Same piece of data is not represented more than once

Enforcement of rules

- The data model enforces business rules

Reusability

- Able to be used for different applications

Flexibility

- Model should be able to cope with changes to the business or data requirements

Data Modeling

- There are two high-level steps involved in creating a data model:
  - Identify the **entities** : objects, things, or "nouns" in the model.
  - Define the **relationships** between those entities.

- Each of these high level steps can be broken into a few parts.

- A data model is represented by a diagram called an Entity Relationship Diagram or **ERD**.

- Database schema is another way of saying that it is a schematic of how the data is stored in the database.

Data Modeling and ERD

- There are two high-level steps involved in creating a data model:
  - Identify the **entities** : objects, things, or "nouns" in the model.
  - Define the **relationships** between those entities.
- Each of these high level steps can be broken into a few parts.
- A data model is represented by a diagram called an Entity Relationship Diagram or **ERD**.
- Database schema is another way of saying that it is a schematic of how the data is stored in the database.

Entities

- Entities are the things, often physical, that have facts, data or attributes associated with them.
- Transactions and customers are examples of entities.
- Processes are almost never entities.
- For example, "order entry," meaning the actual process of entering an order into the system, is not an entity.
- Reports are not entities.
- When developing a data model, it is good practice to have a detailed, extensive description of each entity below the schema.

Entities

- This text should describe what the entity represents, the source of the data and the relationships between the entity and other data within the database.

- Let’s consider the example of a Department as an entity.

- The Sales Department is an occurrence of entity type Department.

- In the context of object-oriented programming, Sales Department is an instance of the class: Department.

- In the context of databases, Sales Department is a record (or row) within the table Department.

Entities

- The first step towards creating an ERD is to identify the entities of interest.
- Inspect the list of columns in the database below.
- The dataset is owned by a credit card company that serves only business credit cards.
- There would be more attributes than what is shown below in a real credit card company's database.
- This example is meant only as a demonstration to learn the concepts related to data modeling.

Entities

- Assume the data for these attributes are all stored in a single spreadsheet.
- If we are interested in instead storing the data in a normalized relational database, what do you think would be the entities in this dataset?

Anyone?

Entities

- Assume the data for these attributes are all stored in a single spreadsheet.
- If we are interested in instead storing the data in a normalized relational database, what do you think would be the entities in this dataset?

Anyone?

Northeastern University

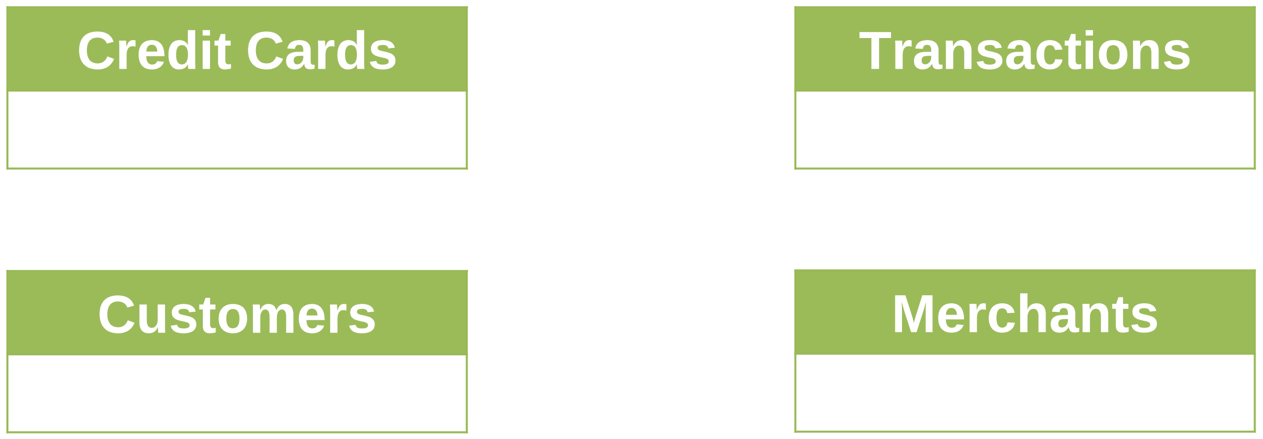
Entities

- Storing Data in a single spreadsheet:

Credit Cards
Customer_ID
TransactionStatus
CreditCard_ID
CustomerContactEmail
TransactionAmount
CreditCardNumber
DateExpired
CreditLimit
InterestRate
Transaction_ID
MerchantCategory
TransactionType
DateIssued
TransactionLocation
Merchant_ID
CustomerCompanyName
MerchantName

Entities

- The entities are represented as boxes in the diagram below.
- Entities are often named as plural nouns with no spaces.



Entities

- Attributes are properties of an entity.
- They are typically nouns, such as a quantity, type or color.
- The values stored in a particular column, however, can be adjectives describing a particular instance of an entity.
- If our entity is a Customer, some possible attributes are ID, name, social security number, address and phone number.
- While identifying the attributes of each entity, it is worthwhile to consider the type of data that is being stored for each attribute.

Entities

- If you are using certain software tools to draw your ERD, you may need to specify a data type for each attribute.
- We will learn more about data types later, but for now think of attributes as either text data (strings) or numeric data.
- The attributes from the credit card company dataset associated with the entities above, are shown below.
- It is a best practice to name each attribute as a single word without spaces.
- As shown below, the attribute names may be shortened from the original dataset.

Entities

- This is especially common practice in cases where the entity name was stored in the attribute name.

- In multi-table queries in SQL, it is best practice to prepend the table name when referring to particular attribute names, therefore storing the entity name in the attribute name can be redundant.

- Some databases may keep the longer version of the attribute names, however, to ensure unique values for all attributes in the database.

- This choice is a matter of preference and/or developer style.

Entities

- Storing Data in normalized Database

Customers
ID
CompanyName
ContactEmail

Credit Cards
ID
CCNumber
Issued
Expiration
Limit
InterestRate

Transactions
ID
Amount
Status
Type
Location

Merchants
ID
Name
Category

- Each entity has specific attributes describing the entity.
- An entity occurrence is an instance of that entity.

Entities

- Storing Data in normalized Database

Transactions
ID
Amount
Status
Type
Location

Transactions

ID	Amount	Status	Type	Location
100245	1200	Posted	Sale	In-person
100573	3795	Pending	Sale	Online
100357	2495	Posted	Sale	Online
100547	-1200	Posted	Return	In-person
100184	3000	Posted	Sale	Online

- Transactions is an entity with attributes ID, Amount, Status, Type and Location.

- This table depicts five occurrences of type Transaction.

Entities

- When creating a database based on an ERD, the entities map to tables, and the entity occurrences are make up the rows or records of the tables.
- In practice, the data modeling process requires a great deal of knowledge about the particular business to which the data belongs as well as the domain.
- In the case of each business, you will be making some assumptions to help you get started.
- Then Database schema might require multiple iterations on different levels until it gets approved.

Keys, relationships and cardinality

- Primary keys are attributes used for identifying records in individual entities.
- A primary key is one or more attributes that uniquely identify a record or row.
- An example of this is a unique customer ID.
- One approach is to choose system usernames as the primary key for users in a given system.
- Usernames must be unique, therefore they have the characteristics of a primary key for a group of users.

Keys, relationships and cardinality

- When choosing a primary key for an entity, it is important to make sure the key is stable and that we have control over it.
- Stability means that the value of the key will not change over time.
- In most cases, it is possible and preferable to use a key generated by the database system.

Entities

- The primary keys for the credit card database are shown in the ERD below, indicated by the PK next to the attribute which uniquely identifies the records in each table.

Customers	
ID	PK
CompanyName	
ContactEmail	

Credit Cards	
ID	PK
CCNumber	
Issued	
Expiration	
Limit	
InterestRate	

Transactions	
ID	PK
Amount	
Status	
Type	
Location	

Merchants	
ID	PK
Name	
Category	

Keys, relationships and cardinality

- After defining the **entities** and their **primary keys**, the second step in the data modeling process is to identify the relationships between the entities.

- An important component of relationships is the cardinality, or how many instances of one entity are related to another entity and vice versa.

- Consider the example of two entities from the credit card database:
    - Customers
    - Credit Cards.

- The relationship can be defined as follows:
    - Each credit card belongs to a single customer;
    - Each customer can have zero, one or many credit cards;

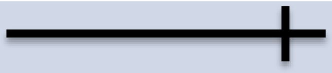
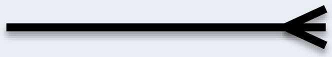
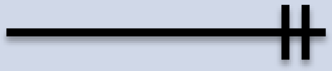
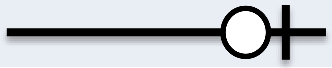
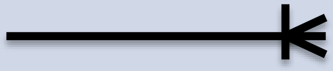
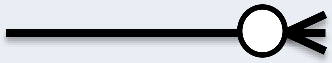
Keys, relationships and cardinality

- This is generalized as a "one to many" relationship.
- Types of relationships include:
    - "one to one“;
    - “one to many”;
    - "many to many“;
- Relationships are depicted as lines between entities in the ERD.
- In addition to a simple line representing the relationship, there are symbols on the ends of the lines indicating the cardinality.
- Cardinality summarizes the maximum number of instances of one entity that are related to another entity and vice versa.

Keys, relationships and cardinality

- There are different ways to indicate cardinality.
- In this course, we will use **crow's foot notation**.
- Often, in conceptual and logical data models, only the top two symbols are used to define relationships between entities.
- These are more general, whereas the lower four symbols are more specific and apply more constraints to the data.
- We will consider only the "One" and "Many" cardinalities.
  - "One" encompasses "One (and only one)" and "Zero or one.“
  - "Many" encompasses "One or many" and "Zero or many."

Keys, relationships and cardinality

crow's foot notation	
	One
	Many
	One (and only one)
	Zero or none
	One or many
	Zero or many

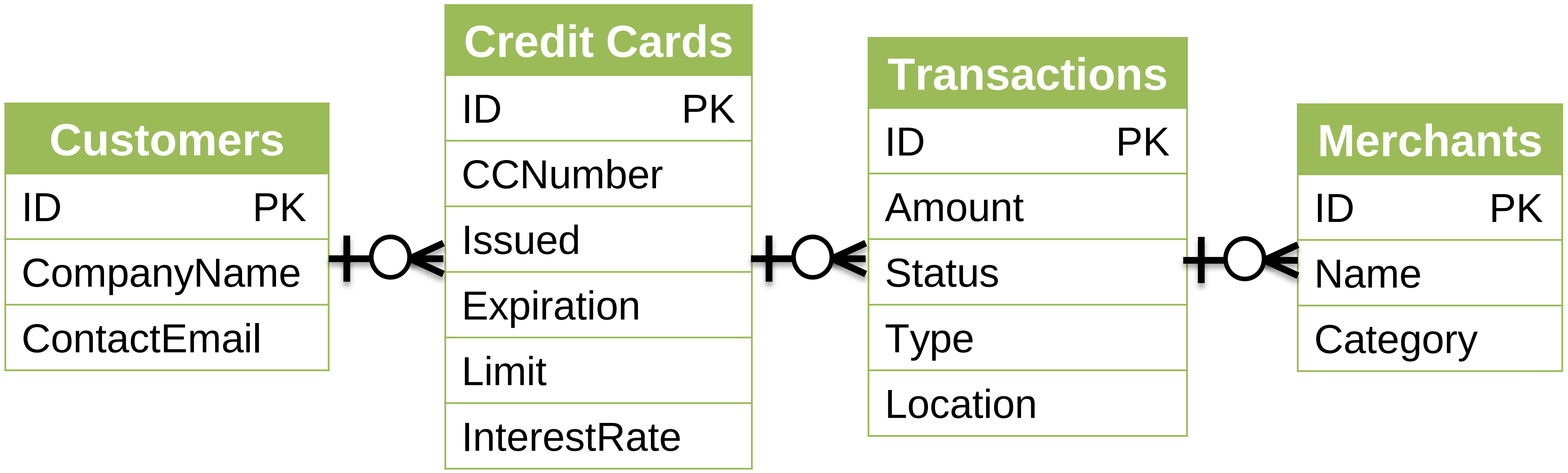
- Another approach to indicate cardinality, which you might see in practice, involves using numbers.

Keys, relationships and cardinality

- Instead of the marks shown for "Zero or many" in the diagram above, we can annotate the appropriate side of the connector line with: "0..N," which indicates that there can be "0 or many" instances of that entity per related entity.
- Similarly, the "One (and only one)" cardinality in the diagram above could be indicated as a line labeled with: "1", which indicates there can be only one instance of that entity per related entity.
- We can add the relationships to our credit card ERD with crow's foot notation.
- Note that the "Many" cardinality is shown as "Zero or many," which we shall consider to be equivalent in this case.

Keys, relationships and cardinality

- Credit Card ERD with relationships



- These relationships constrain the data in a way that should reflect the rules of the business.

Keys, relationships and cardinality

- Each customer can have zero to many credit cards, however each credit card can belong to only one customer;
- Each credit card can have zero or many transactions, however each transaction can only be related to one credit card;
- Each merchant can have zero or many transactions, however each transaction can only be related to one merchant;

Keys, relationships and cardinality

- **Primary keys** are used to uniquely identifying single records in a table.
- **Foreign keys** are a different type of key which are used to implement relationships in a data model.
- Relationship in an ERD is drawn by a line between two entities.
- In order for the relationship to be implemented by the database system, an attribute from one of the entities must be stored with the related entity.
- This attribute, when stored with an entity which it does not describe, is known as a **foreign key**.

Keys, relationships and cardinality

- In a "One to many" relationship, the entity associated with the "one" side of the relationship is known as the independent entity or the parent entity in terms of a specific relationship.

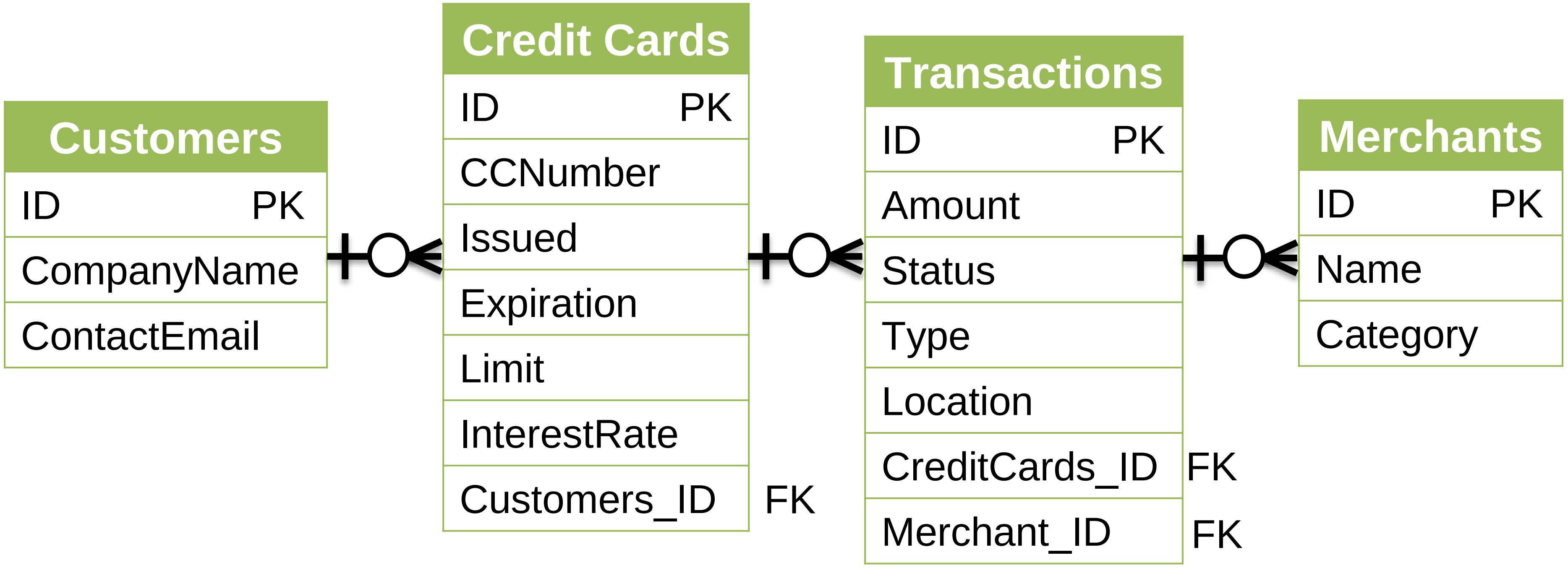
- The entity associated with the "many" side of the relationship is known as the dependent entity or the child entity in terms of a specific relationship.

- For a given one to many relationship:
    - the foreign key is stored in the dependent entity;
    - the foreign key attribute is usually the primary key from the independent entity in the relationship.

- *An independent table has no foreign keys and stands alone without information from other tables.*

Keys, relationships and cardinality

- Building on the credit card company database from above, the foreign keys are added in the appropriate tables below, and are indicated by FK:



- In practice, not every possible relationship in a database will be implemented.

Keys, relationships and cardinality

- For example:
    - each customer can have zero or many transactions and each transaction is associated with one customer.
    - Because this relationship can be derived by joining customers to credit cards, and then credit cards to transactions, it is not necessary or considered a good practice to store the redundant foreign key, Customers_ID, in the transactions table.
    - This extra relationship would also form a "loop" between the three tables, which should be avoided when possible.
    - In some cases, however, when the relationship that closes the loop has another meaning and cannot be derived through the other relationships in the ERD, the additional relationship is necessary and acceptable.

Referential Integrity

- Referential integrity refers to maintaining the validity of foreign keys when the primary key in the parent table changes.
- The database performs this maintenance automatically, but it is important to understand this concept as it relates to keys.
- Every foreign key must either match a primary key or be NULL.
- Consider the following:
  - Users cannot add a new entry to the CreditCards table with an invalid value for Customers_ID.
  - Depending on the constraints set when setting up the foreign key, it may possible to have a NULL value for Customers_ID
  - This would refer to a credit card that has been added to the database but has not yet been assigned to a customer.

Referential Integrity

- Foreign keys can also be set up such that they cannot be equal to NULL, which would be likely be preferred for this case in practice. The database enforces the applied constraints when users add each new credit card to the database.
- It is also important to understand what will happen when records are removed from a database.
- For example:
    - What happens when a merchant ends their relationship with the credit card company?
    - If the merchant is removed from the credit card company's database, what will happen to the transactions associated with that merchant?

Restrict

- The referential action **restrict** specifies that records in the primary key table cannot be deleted unless all related records in tables where that primary key is used as a foreign key have already been deleted.

- In the case of transactions and merchants, with a restrict referential action, the merchant cannot be deleted until all of the transactions associated with that have been assigned to new merchants or have been removed from the database.

- Put another way, no foreign keys may point to the merchant that we are trying to delete.

- If we deleted the merchant without assigning the transactions to a new merchant the transactions would become "orphans."

Cascade

- The referential action cascade, specifies that when records in the primary key table are deleted, all related records in tables where that primary key is used as a foreign key are also deleted.

- In the case of the transactions and merchants example, with a cascade referential action, when a merchant is deleted, all transactions related to that merchant are also removed from the database.

- A cascade referential action should be used when the data stored in the child entity will never be used after the associated parent record is deleted.

- There will be orphaned data in a cascade referential action, however, this data will be deleted such that it can never be misused.

SQL Fundamentals

- Lets see how to use a data model to build a relational database.
- This is implemented with the programming language commonly used to interact with relational databases: **SQL**.
- SQL stands for Structured Query Language.
- Though many modern relational database client software have a graphical interface which can be used in place of writing SQL code and queries.
- Learning SQL is straightforward and may give you more flexibility as you create and query your database because only certain options may be implemented in any particular GUI.

SQL Fundamentals

- Structured Query Language or SQL, is the language used to interact with relational databases.
- It is comprised of keywords or statements that enable users to create and manipulate databases and tables as well as select or retrieve data from a relational database.
- SQL is a programming language, but it’s not a complete language such as Java, R or Python.
- SQL consists of only about 30 statements.
- Though SQL is a powerful language, it has several inconsistencies.

SQL Fundamentals

- In particular, NULL values are problematic in SQL.
- It is portable across operating systems, however, and is somewhat portable across vendors.
- In another words, SQL is a declarative language, not a procedural language like Java or C++.
- This means the language is designed to let the user express what they want to do, e.g. create a new table, or retrieve some data, but does not allow the user to specify how to do it.
- This "procedure" is carried out by the database.

SQL Fundamentals

- In particular, NULL values are problematic in SQL.
- It is portable across operating systems, however, and is somewhat portable across vendors.
- In another words, SQL is a declarative language, not a procedural language like Java or C++.
- This means the language is designed to let the user express what they want to do, e.g. create a new table, or retrieve some data, but does not allow the user to specify how to do it.
- This "procedure" is carried out by the database.

SQL Fundamentals

- Some of the other aspects of SQL that vary among SQL implementations are:
    - Error codes
    - Data types supported (dates/times, currency, string/text variations)
    - System tables; the structure of the database itself
    - Interactive SQL
    - Programming interfaces
    - Dynamic SQL used for report writers and query tools
    - Implementer-defined variations within the standard
    - Database initialization, opening, and connection
    - Whether case matters (upper case, lower case)

SQL Fundamentals

- All flavors of SQL are designed to allow for:
  - Data definition: Operations to **CREATE** tables and views (virtual tables)
  - Data manipulation: **INSERT**, **DELETE**, **UPDATE** or retrieve (**SELECT**) data
  - Data integrity: referential integrity and transactions; enforces keys (primary/foreign)
  - **Access** control/security
  - Data sharing by **concurrent** users

SQL Fundamentals

- When creating a database, it is best practice to follows these steps:

1. Create the data model
2. Create a new database using the selected database software.
3. Create the tables within the new database.
4. Insert data into the database.
5. Write queries to retrieve a subset of data from the database.

SQL Fundamentals

1 Create the data model

- This provides a template for what tables are required, and which attributes are stores in each of the tables.
- We've already discussed and covered this step

SQL Fundamentals

2 Create a new database using the selected database software.

- The exact approach depends on which software is selected.
- The database software can access multiple databases, analogous to Excel accessing multiple workbooks.
- This step depends on the database software, so we will be using MySQL in this module.

SQL Fundamentals

3 Create the tables within the new database.

- Using the data model, begin by creating the tables.
- Table names should be specified based on the data model (i.e. the entities in the entity relationship diagram).
- Column names should be specified based on the attributes in the data model, as well as the data type that will be stored in each column.

SQL Fundamentals

4 Insert data into the database.

- Inserting data into Database can be done using variety of different ways:
    - Restore from data backup
    - Import from another system
    - Import from previously created text/binary files
    - Enter data manually
    - Use pre-created scripts with data insertion commands

SQL Fundamentals

5 Write queries to retrieve a subset of data from the database.

- Querying data uses SQL language by:
  - Running SQL scripts
  - Using GUI
  - Using user client/server applications

MySQL DB Management

- MySQL Workbench uses the term **schema** throughout the GUI.
- In MySQL, schema is synonymous with **database**.
- This varies among databases: in Oracle, a database contains many schemas, and each schema contain multiple tables, schema is just a layer of organization.
- For the purpose of this course, while in MySQL we will use the terms schema and database interchangeably.

MySQL DB Management

CREATE DATABASE

- Using MySQL Workbench, enter the following syntax into the query window and execute the following command to create the credit card company database:

```
CREATE DATABASE creditcardco;
```

- This creates the database named **creditcardco**.

MySQL DB Management

SHOW DATABASES

- SHOW is used to list the databases on the MySQL server.
- The syntax for this is shown below:

```
SHOW DATABASES;
```

- Execute the above command; a list of all databases is returned.
- We should see the **creditcardco** database, which we just created.
- We will also see some other databases, such as:
  - information_schema
  - performance_schema.
- Those are databases that MySQL uses for performance purposes.

MySQL DB Management

USE database

- The USE command specifies to the database server that we are using a particular database.
- After specifying which database we are using with USE, all commands we send to the database server will be applied to the database in use.
- If we don't specify which database to use, we have to specify the database or schema name in every query.
- Run the following command to specify that we will use the **creditcardco** database:

USE creditcardco;

MySQL DB Management

DROP DATABASE

- In database vernacular, DROP refers to deletion.
- Run the following command to DROP or delete the **creditcardco** database from the MySQL server:

```
DROP DATABASE IF EXISTS creditcardco;
```

- The DROP keyword specifies that we are going to delete something, DATABASE specifies that we are dropping a database (as opposed to TABLE or something else).
- The IF EXISTS keyword is optional

MySQL Data Types

- Every SQL implementation has a core selection of common data types.
- Additionally, each flavor of SQL may have additional available data types.
- Data types are applied to each column in a table and specify how MySQL should store data in that column and what operations can be performed on that data, much like the data types we in R or Microsoft Excel.
- For example, if we put numbers in a text field, we won’t be able to perform mathematical operations on those numbers.

MySQL Data Types

- Selected numeric data types in MySQL

Numeric Data Types	Description
INT	4-Byte integer
BIGINT	8-Byte integer
DECIMAL	Number with fixed number of digits before and after the decimal point
FLOAT	Single-precision floating-point number
DOUBLE	Double-precision floating-point number
BIT	Bit value storage

MySQL Data Types

- Selected string/text data types in MySQL

String/Text Data Types	Description
CHAR	Fixed-length string
VARCHAR	Variable-length string
MEDIUMTEXT	Variable length string, with greater max length than varchar. Maximum 16MB
LONGTEXT	Variable length string, with greater max length than varchar. Maximum 4GB

MySQL Data Types

- Selected date and time data types in MySQL

Date and Time Data Types	Description
DATE	A date stored in 'YYYY-MM-DD' format
TIME	A time stored in 'HH:MM:SS' format, can be used for elapsed time as well as time of day data; put another way, this data type is not limited to the 24-hour cycle
DATETIME	A date and time stored in 'YYYY-MM-DD HH:MM:SS‘ format
TIMESTAMP	A date and time stored in 'YYYY-MM-DD HH:MM:SS' UTC format
YEAR	A year stored in ‘YYYY’ format

Basic Table Management

CREATE TABLE

- Let’s create a table with no data. This can be done by specifying the name of the table, the names of the columns, and the column data types.
- The basic syntax is shown below:

```
CREATE TABLE schema_name.table_name (
        column1_name column1_type,
        column2_name column2_type
        [more columns as needed, optionally primary key,
        default  value, nullable, index, and foreign key
        specifications]
);
```

Basic Table Management

CREATE TABLE

- To create table Customers the following syntax can be used:

```
USE creditcardco;
CREATE TABLE Customers (
        ID INT,
        CompanyName VARCHAR(255),
        ContactEmail VARCHAR(255)
);

ALTER TABLE Customers
ADD PRIMARY KEY (ID);
```

Customers	
ID	PK
CompanyName	
ContactEmail	

Basic Table Management

CREATE TABLE

- The USE statement lets MySQL know that we want to work in the **creditcardco** database.
- The CREATE statement creates a table called **Customers**, with three fields (columns/attributes).
- ID is an INT, a number with no decimal points.
- All of the other columns are of type VARCHAR(255), meaning these are text fields or strings with a maximum length of 255 characters.
- The ALTER statement specifies that the ID attribute is the primary key in this table.

Basic Table Management

CREATE TABLE

- Let's DROP this empty table, and then recreate it with more detail.
- Consider the code below:

```
USE creditcardco;
DROP TABLE IF EXISTS Customers;
CREATE TABLE Customers (
        ID INT NOT NULL,
        CompanyName VARCHAR(255) NOT NULL,
        ContactEmail VARCHAR(255),
        PRIMARY KEY (ID)
);
```

Basic Table Management

CREATE TABLE

- First, the original Customers table is dropped.
- The syntax here is as follows: DROP specifies we want to drop (delete) something from the database.
- TABLE specifies that it's a table, specifically, that we want to delete as opposed to an entire database.
- Finally, Customers indicates which table to drop.
- The IF EXISTS statement is optional.
- We add the IF EXISTS clause to indicate that the database should only drop the table if it already exists.
- NOT NULL creates a condition for inserting data, the entries in these particular columns must have values and cannot be empty.

INSERT, UPDATE and DELETE Records

- INSERT, UPDATE, and DELETE are SQL commands used to add, modify or remove data in tables.
- INSERT adds data to a table.
- UPDATE modifies existing data in a table.
- DELETE removes data from a table.
- When dealing with data in a relational database table, we can think in terms of records.

INSERT, UPDATE and DELETE Records

INSERT

- Data values can be inserted into a table one row at a time using the following syntax:

  INSERT INTO table_name [(col_name,...)]
  {VALUES | VALUE} ({entry | DEFAULT}, ...), (...), ...;

- The **INSERT INTO** and **VALUES** commands assume attribute values will be listed in the same order as the order that the attributes were specified in the **CREATE TABLE** command.

INSERT, UPDATE and DELETE Records

INSERT

- To insert values out of order or to insert entries to only certain columns, this can be accomplished by specifying column names after table name, separated by commas, followed by a list of comma-separated values in the order of the previously specified column names.

- The VALUE and VALUES commands are syntactically equivalent, meaning either can be used.

- {entry | DEFAULT} means that the user may insert a value for an attribute, or if no value is specified it will take on the default value for that attribute.

INSERT, UPDATE and DELETE Records

INSERT

- Execute the commands below to insert three customers into the Customers table:

```
INSERT INTO Customers (ID, CompanyName, ContactEmail)
VALUES (302, 'Meyer, LLC', 'l.meyer@email.com');

INSERT INTO Customers
VALUES (479, 'Pourbemani Products',
'pourbemaniproducts@email.com'),
(128, 'Lebowitz, Incorporated', 'j.lebowitz@email.com'),
(105, 'Leonard Co.'; 's.leonard@email.com');
```

INSERT, UPDATE and DELETE Records

UPDATE

- Table data values can be modified using UPDATE.
- For a single table update, use the following syntax:

```
UPDATE table_name
SET column_name1 = {expression1 | DEFAULT}
[, column_name2 = {expression2 | DEFAULT}, ...]
[WHERE where_condition];
```

INSERT, UPDATE and DELETE Records

UPDATE

- Let’s assume that we put the wrong email address into one of the records in the Customers table with our last INSERT statement.
- Pourbemani Product's email should be 's.pourbemani@email.com.'
- Execute the following command to correct this using the above syntax as follows:

```
UPDATE Customers
SET ContactEmail = 's.pourbemani@email.com'
WHERE CompanyName = 'Pourbemani Products';
```

INSERT, UPDATE and DELETE Records

UPDATE

- The above query doesn’t use the ID in the WHERE clause.
- Think about the unintended consequences of this query.
- If Pourbemani Products is not a unique value in the CompanyName field, then all of the email addresses associated with that company name would be updated to s.pourbemani@email.com.
- If this query gave you an error about safe mode, try running it again with the following work around:

```
UPDATE Customers
SET ContactEmail = 's.pourbemani@email.com'
WHERE ContactName = 'Pourbemani Products' AND ID <> 0;
```

INSERT, UPDATE and DELETE Records

UPDATE

- The addition of the second condition in the WHERE clause is saying to replace the email everywhere the contact name is Pourbemani Products AND where the ID is any value other than zero.
- This trivial condition gets around MySQL safe mode and will allow you to update the email address.
- A better approach would be to specify the ID in the condition properly as shown below.
- This syntax ensures that only the single record is updated:

```
UPDATE Customers
SET ContactEmail = 's.pourbemani@email.com'
WHERE ID = 479;
```

INSERT, UPDATE and DELETE Records

DELETE

- DELETE is a statement that removes records from a table.
- Single table syntax is shown here:

```
DELETE FROM table_name
[WHERE where_condition];
```

- To delete "Leonard, Co." company from our Customers table we can use the following syntax:

```
DELETE FROM Customers
WHERE ID = 105;
```

INSERT, UPDATE and DELETE Records

SELECT

- The SELECT statement is used to retrieve data from databases. SELECT statements are used for many of the database queries performed by an analyst.
- SELECT is the tool that is used to get data out of a database in the form in which we are most interested.
- A basic SELECT statement uses the following syntax:

```
SELECT columns
FROM table
WHERE specific records fit some condition(s);
```

INSERT, UPDATE and DELETE Records

SELECT

- SELECT statements can be made even more complicated with more clauses using the following syntax:

SELECT columns/attributes
INTO new table
FROM table
WHERE specific records fit some condition(s)
GROUP BY grouping conditions (columns)
HAVING group-property (specific rows)
ORDER BY ordering criterion ASC | DESC;

INSERT, UPDATE and DELETE Records

SELECT

- To select all records from Customers table, the following statement can be used:

```
SELECT  *
FROM Customers;
```

- In this module, we will learned:
  - how to retrieve data from a relational database.
  - about how to structure and organize data.
  - how to create a database based.
  - how to put data into a database.
  - How to update table and data in table
  - How to delete database, table and records from table

References

MySQL Explained: Your Step-by-Step Guide, 2015,
by Andrew Comeau

Shlaer-Mellor Method: The OOA96 Report:
http://ooatool.com/docs/OOA96.pdf

MySQL Data Types
https://dev.mysql.com/doc/refman/5.7/en/data-types.html

Summary

In this session we've learned:

- What is SQL
- Concepts of relational databases.
- Data modeling and entity relationships diagrams.
- SQL fundamentals.
- How to manipulate databases, tables and records in MySQL
- Basic SQL queries.

Q & A