**WEEK 3: EXCEL & DATABASE DESIGN**

**Task 2: Ms Excel : Key Data Analyst Tool**

Although the charm of MS-Excel for data analysis may not be as exciting as coming technologies like SQL & Python, it still remains an **invaluable tool for most of the day-to-day data management tasks** most organizations rely on including Payroll, Employee Management, etc.

Thus, learning this **classic skill is absolutely essential** to be called an analyst. And if you master excel, you have easily mastered the basic tool required to start data analysis.

Your Tasks:

1. Complete the Concept Builder video
2. View the case study by clicking on button below
3. Solve the case study and submit it as your assignment

Video - <https://youtu.be/Vl0H-qTclOg>

Assignment - Case Study Virtual Gaming Company - <https://docs.google.com/spreadsheets/d/1SbiV2ds9ZzV5pu_Kc_8lpoAlfl56P6uG3su4slZ9FhI/edit#gid=1069306441>

# Task 3: Data Warehouse

Think of yourself as a data analyst working for a company that has three departments: Marketing, Sales and Finance. Now, let’s assume that each department maintains a separate database.

This could lead to a situation where each department has its own version of the facts. For a question like “What is the total revenue of the last quarter?”, every department might have a different answer. This occurs because each department draws information from a different database.

This is where a data warehouse can help in creating a **single version** of the **truth** and **facts**. A data warehouse would thus be the **central repository** of data of the entire enterprise.

A **Data Warehouse** is a collection of data. It demonstrates the following properties:

* **Subject-oriented:** A data warehouse should contain information about a few – well-defined subjects rather than the enterprise.
* **Integrated:** A data warehouse is an integrated repository of data. It contains information from various systems within an organization.
* **Non-volatile:** The data values in a database cannot be changed without a valid reason.
* **Time-variant:** A data warehouse contains historical data for analysis.

# How does a data warehouse work?

A **Data warehouse** may contain **multiple** databases. Within each database, data is organized into tables and columns. Within each column, you can define a **description of the data**, such as integer, data field, or string. Tables can be organized inside of **schemas**, which you can think of as folders. When data is ingested, it is stored in various tables described by the schema. **Query** tools use the schema to determine which data tables to access and analyze.

# What are the benefits of using a data warehouse?

**Benefits of a data warehouse include the following**:

* Informed decision making
* Consolidated data from many sources
* Historical data analysis
* Data quality, consistency, and accuracy
* Separation of analytics processing from transactional databases, which improves performance of both systems

# How is a data warehouse architected ?

A **data warehouse architecture** is made up of tiers. The top tier is the **front-end** client that presents results through reporting, analysis, and data mining tools. The middle tier consists of the **analytics engine** that is used to access and analyze the data. The bottom tier of the architecture is the **database server,** where data is loaded and stored. Data is stored in two different types of ways:

1. Data that is accessed frequently is stored in very fast storage (like SSD drives) and
2. Data that is infrequently accessed is stored in a cheap object store, like Amazon S3.

The data warehouse will automatically make sure that frequently accessed data is moved into the “**fast**” storage so query speed is optimized.

# How do data warehouses, databases, and data lakes work together?

Typically, businesses use a **combination of a database**, a **data lake,** and a **data warehouse** to store and analyze data. Amazon Redshift’s [lake house architecture](https://aws.amazon.com/redshift/lake-house-architecture/) makes such an integration easy.

As the volume and variety of data increases, it’s advantageous to follow one or more common patterns for working with data across your database, data lake, and data warehouse:

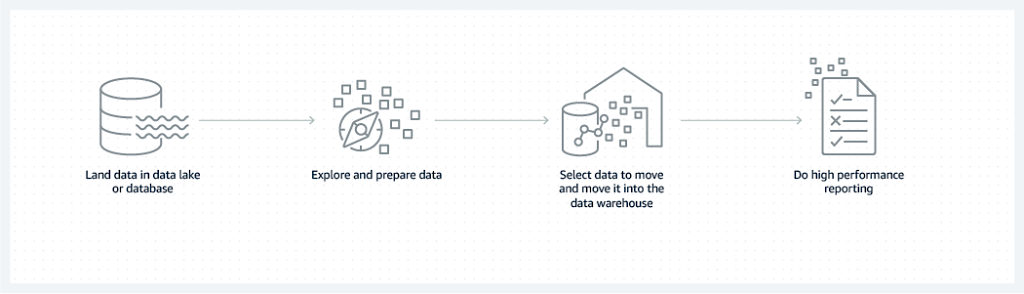
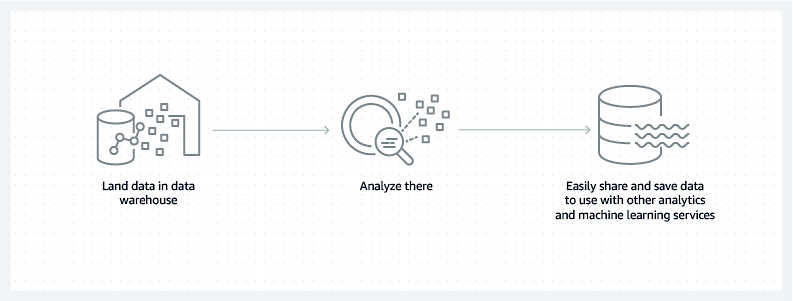


Image (above): Land data in a database or datalake, prepare the data, move selected data into a data warehouse, then perform reporting.

Image (above): Land data in a data warehouse, analyze the data, then share data to use with other analytics and machine learning services.

A **data warehouse** is specially designed for **data analytics**, which involves reading **large amounts of data** to understand relationships and trends across the data. A database is used to capture and store data, such as recording details of a transaction.

Unlike a data warehouse, a **data lake** is a **centralized repository** for all data, including structured, semi-structured, and unstructured. A data warehouse requires that the data be organized in a tabular format, which is where the **schema** comes into play. The tabular format is needed so that SQL can be used to query the data. But not all applications require data to be in tabular format. Some applications, like big data analytics, full text search, and machine learning, can access data even if it is ‘semi-structured’ or completely unstructured.

# Data warehouse vs Data Lake

| **CHARACTERISTICS** | **DATA WAREHOUSE** | **DATA LAKE** |
| --- | --- | --- |
| **Data** | Relational data from transactional systems, operational databases, and line of business applications | All data, including structured, semi-structured, and unstructured |
|  |  |  |
| **Schema** | Often designed prior to the data warehouse implementation but also can be written at the time of analysis (schema-on-write or schema-on-read) | Written at the time of analysis (schema-on-read) |
|  |  |  |
| **Price/Performance** | Fastest query results using local storage | Query results getting faster using low-cost storage and decoupling of compute and storage |
| **Data quality** | Highly curated data that serves as the central version of the truth | Any data that may or may not be curated (i.e. raw data) |
| **Users** | Business analysts, data scientists, and data developers | Business analysts (using curated data), data scientists, data developers, data engineers, and data architects |
| **Analytics** | Batch reporting, BI, and visualizations | Machine learning, exploratory analytics, data discovery, streaming, operational analytics, big data, and profiling |

|  |
| --- |
|  |

Top of Form

Bottom of Form

# Data warehouse vs database

| **CHARACTERISTICS** | **DATA WAREHOUSE** | **TRANSACTONAL DATABASE** |
| --- | --- | --- |
| **Suitable workloads** | Analytics, reporting, big data | Transaction processing |
| **Data source** | Data collected and normalized from many sources | Data captured as-is from a single source, such as a transactional system |
| **Data capture** | Bulk write operations typically on a predetermined batch schedule | Optimized for continuous write operations as new data is available to maximize transaction throughput |
| **Data normalization** | Denormalized schemas, such as the Star schema or Snowflake schema | Highly normalized, static schemas |
| **Data storage** | Optimized for simplicity of access and high-speed query performance using columnar storage | Optimized for high throughout write operations to a single row-oriented physical block |
| **Data access** | Optimized to minimize I/O and maximize data throughput | High volumes of small read operations |

# How does a data mart compare to a data warehouse?

A **data mart** is a data warehouse that serves the needs of a **specific team** or **business unit**, like finance, marketing, or sales. It is smaller, more focused, and may contain **summaries of data** that best serve its community of users. A data mart might be a portion of a data warehouse, too.

#### **Data warehouse vs data mart**

| **CHARACTERISTICS** | **DATA WAREHOUSE** | **DATA MART** |
| --- | --- | --- |
| **Scope** | Centralized, multiple subject areas integrated together | Decentralized, specific subject area |
| **Users** | Organization-wide | A single community or department |
| **Data Source** | Many sources | A single or a few sources, or a portion of data already collected in a data warehouse |
| **Size** | Large, can be 100’s of gigabytes to petabytes | Small, generally up to 10’s of gigabytes |
| **Design** | Top-down | Bottom-up |
| **Data details** | Complete, detailed data | May hold summarized data |

# How can a data warehouse be deployed on AWS?

**AWS** allows you to take advantage of all of the **core benefits** associated with on-demand computing: accessing seemingly **limitless storage** and compute capacity, **scaling your system** in parallel with your growing amount of data collected, stored, and queried, and paying only for the resources you provision. AWS offers a broad set of managed services that **integrate** seamlessly with each other so that you can quickly deploy an **end-to-end analytics** and **data warehousing solution.**

The following illustration shows the key steps of an end-to-end analytics process, also called a stack. AWS offers a [variety of managed services](https://aws.amazon.com/big-data/datalakes-and-analytics/) at each step.



Image (above): AWS offers a variety of products and services at each step of the analytics process.

[Amazon Redshift](https://aws.amazon.com/redshift/) is **fast**, **fully-managed**, and **cost-effective** data warehouse service. It gives you petabyte-scale data warehousing and exabyte-scale data lake analytics together in one service, for which you [only pay for what you use](https://aws.amazon.com/redshift/pricing/).

# Task 4: Dimensional Modelling

One of the primary methods of designing a data warehouse is called **dimensional modelling**.

The **two key elements** of dimensional modelling are **facts and dimensions**, which are basically the different types of variables used to design a data warehouse. They are arranged in a specific manner known as a **schema diagram**.

Basically, facts are the **numerical** data in a data warehouse and dimensions are the **metadata** (that is, data explaining some other data) attached to the fact variables. Both facts and dimensions are equally important for generating actionable insights from a data set.

# What is Dimensional Modelling in Data Warehouse?

**Dimensional Modeling (DM)** is a data structure technique optimized for data storage in a Data warehouse. The purpose of dimensional modeling is to **optimize** the database for faster retrieval of data. The concept of Dimensional Modelling was developed by **Ralph Kimball** and consists of “**fact**” and “**dimension**” tables.

A dimensional model in data warehouse is designed to **read**, **summarize**, **analyze** numeric information like values, balances, counts, weights, etc. in a data warehouse. In contrast, relation models are optimized for addition, updating and deletion of data in a real-time Online Transaction System.

These dimensional and relational models have their **unique way** of data storage that has specific advantages.

For instance, in the relational mode, normalization and ER models reduce redundancy in data. On the contrary, dimensional model in data warehouse arranges data in such a way that it is easier to retrieve information and generate reports.

Hence, Dimensional models are used in [data warehouse systems](https://www.guru99.com/data-warehousing.html) and not a good fit for relational systems.

# Elements of Dimensional Data Model

### Fact

Facts are the **measurements/metrics** or **facts** from your business process. For a Sales business process, a measurement would be quarterly sales number

### Dimension

Dimension provides the **context surrounding** a business process event. In simple terms, they give who, what, where of a fact. In the Sales business process, for the fact quarterly sales number, dimensions would be

* **Who** – Customer Names
* **Where** – Location
* **What** – Product Name

In other words, a dimension is a **window to view** information in the facts.

### Attributes

The Attributes are the various **characteristics** of the dimension in dimensional data modeling.

In the Location dimension, the attributes can be

* State
* Country
* Zipcode etc.

Attributes are used to search, filter, or classify facts. **Dimension Tables contain Attributes**

### Fact Table

A fact table is a **primary** table in dimension modelling.

A Fact Table contains

1. Measurements/facts
2. Foreign key to dimension table

### Dimension Table

* A dimension table contains **dimensions of a fact**.
* They are **joined** to fact table via a foreign key.
* Dimension tables are **de-normalized** tables.
* The Dimension **Attributes** are the various columns in a dimension table
* Dimensions offers **descriptive characteristics** of the facts with the help of their attributes
* **No set limit** set for given for number of dimensions
* The dimension can also contain one or more **hierarchical relationships**

# Difference between Dimension table vs. Fact table

### **Key Difference**

* Fact table contains **measurements, metrics, and facts** about a business process while the Dimension table is a companion to the fact table which contains **descriptive attributes** to be used as query constraining.
* Fact table is located at the **center** of a star or snowflake schema, whereas the Dimension table is located at the **edges** of the star or snowflake schema.
* Fact table is defined by their **grain** or its most atomic level whereas Dimension table should be **wordy, descriptive, complete,** and **quality assured.**
* Fact table helps to store **report labels** whereas Dimension table contains **detailed data**.
* Fact table **does not** contain a hierarchy whereas the Dimension table contains **hierarchies**.

### **Difference between Dimension table vs. Fact table**

| **PARAMETERS** | **FACT TABLE** | **DIMENSION** |
| --- | --- | --- |
| **Definition** | Measurements, metrics or facts about a business process. | Companion table to the fact table contains descriptive attributes to be used as query constraining. |
| **Characteristic** | Located at the center of a star or snowflake schema and surrounded by dimensions. | Connected to the fact table and located at the edges of the star or snowflake schema |
| **Design** | Defined by their grain or its most atomic level. | Should be wordy, descriptive, complete, and quality assured. |
| **Task** | Fact table is a measurable event for which dimension table data is collected and is used for analysis and reporting. | Collection of reference information about a business. |
| **Type of Data** | Facts tables could contain information like sales against a set of dimensions like Product and Date. | Evert dimension table contains attributes which describe the details of the dimension. E.g., Product dimensions can contain Product ID, Product Category, etc. |
| **Key** | Primary Key in fact table is mapped as foreign keys to Dimensions. | Dimension table has a primary key columns that uniquely identifies each dimension. |
| **Storage** | Helps to store report labels and filter domain values in dimension tables. | Load detailed atomic data into dimensional structures. |
| **Hierarchy** | Does not contain Hierarchy | Contains Hierarchies. For example Location could contain, country, pin code, state, city, etc. |

# Types of facts

| **TYPES OF FACTS** | **EXPLANATION** |
| --- | --- |
| **Additive** | Measures should be added to all dimensions. |
| **Semi-Additive** | In this type of facts, measures may be added to some dimensions and not with others. |
| **Non-Additive** | It stores some basic unit of measurement of a business process. Some real-world examples include sales, phone calls, and orders. |

# Types of Dimensions

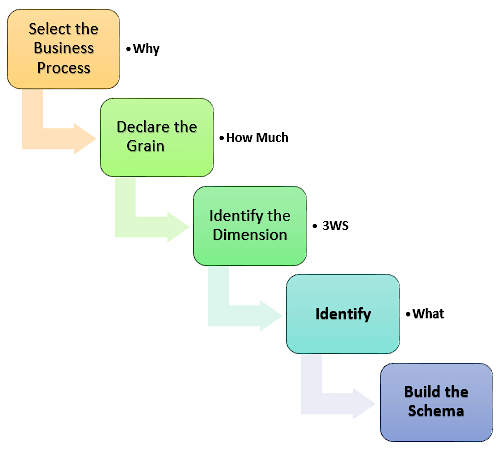
| **TYPES OF DIMENSION** | **DEFINITION** |
| --- | --- |
| **Conformed Dimensions** | Conformed dimensions is the very fact to which it relates. This dimension is used in more than one-star schema or Datamart. |
| **Outrigger Dimensions** | A dimension may have a reference to another dimension table. These secondary dimensions called outrigger dimensions. This kind of Dimensions should be used carefully. |
| **Shrunken Rollup Dimensions** | Shrunken Rollup dimensions are a subdivision of rows and columns of a base dimension. These kinds of dimensions are useful for developing aggregated fact tables. |
| **Dimension-to-Dimension Table Joins** | Dimensions may have references to other dimensions. However, these relationships can be modeled with outrigger dimensions. |
| **Role-Playing Dimensions** | A single physical dimension helps to reference multiple times in a fact table as each reference linking to a logically distinct role for the dimension. |
| **Junk Dimensions** | It a collection of random transactional codes, flags or text attributes. It may not logically belong to any specific dimension. |
| **Degenerate Dimensions** | Degenerate dimension is without corresponding dimension. It is used in the transaction and collecting snapshot fact tables. This kind of dimension does not have its dimension as it is derived from the fact table. |
| **Swappable Dimensions** | They are used when the same fact table is paired with different versions of the same dimension. |
| **Step Dimensions** | Sequential processes, like web page events, mostly have a separate row in a fact table for every step in a process. It tells where the specific step should be used in the overall session. |

# Steps of Dimensional Modelling

The accuracy in creating your Dimensional modeling determines the success of your data warehouse implementation. Here are the steps to create Dimension Model

1. Identify Business Process
2. Identify Grain (level of detail)
3. Identify Dimensions
4. Identify Facts
5. Build Star

The model should describe the Why, How much, When/Where/Who and What of your business process



#### **Step 1) Identify the Business Process**

**Identifying** the actual business process a data-warehouse should cover. This could be Marketing, Sales, HR, etc. as per the [data analysis](https://www.guru99.com/what-is-data-analysis.html) needs of the organization. The selection of the Business process also depends on the **quality of data** available for that process. It is the most important step of the Data Modelling process, and a failure here would have cascading and irreparable defects.

To describe the business process, you can use plain text or use basic Business Process Modelling Notation (BPMN) or Unified Modelling Language (UML).

#### **Step 2) Identify the Grain**

The **Grain** describes the **level of detail** for the business problem/solution. It is the process of identifying the **lowest** level of information for any table in your data warehouse. If a table contains sales data for every day, then it should be daily granularity. If a table contains total sales data for each month, then it has monthly granularity.

During this stage, you answer questions like

1. Do we need to store all the available products or just a few types of products? This decision is based on the business processes selected for Datawarehouse
2. Do we store the product sale information on a monthly, weekly, daily or hourly basis? This decision depends on the nature of reports requested by executives
3. How do the above two choices affect the database size?

**Example of Grain:**

The CEO at an MNC wants to find the sales for specific products in different locations on a daily basis.

So, the grain is “product sale information by location by the day.”

#### **Step 3) Identify the Dimensions**

**Dimensions** are **nouns** like date, store, inventory, etc. These dimensions are where all the **data should be stored**. For example, the date dimension may contain data like a year, month and weekday.

**Example of Dimensions:**

The CEO at an MNC wants to find the sales for specific products in different locations on a daily basis.

Dimensions: Product, Location and Time

Attributes: For Product: Product key (Foreign Key), Name, Type, Specifications

Hierarchies: For Location: Country, State, City, Street Address, Name

#### **Step 4) Identify the Fact**

This step is co-associated with the business users of the system because this is where they get **access to data** stored in the data warehouse. Most of the **fact table rows** are **numerical** values like price or cost per unit, etc.

**Example of Facts:**

The CEO at an MNC wants to find the sales for specific products in different locations on a daily basis.

The fact here is Sum of Sales by product by location by time.

#### **Step 5) Build Schema**

In this step, you implement the **Dimension Model**. A schema is nothing but the **database structure** (arrangement of tables). There are two popular schemas

1. **Star Schema**

The star schema architecture is easy to design. It is called a star schema because diagram resembles a star, with points radiating from a center. The center of the star consists of the fact table, and the points of the star is dimension tables.

The fact tables in a star schema which is third normal form whereas dimensional tables are de-normalized.

1. **Snowflake Schema**

The snowflake schema is an extension of the star schema. In a snowflake schema, each dimension are normalized and connected to more dimension tables.

# Rules for Dimensional Modelling

Following are the rules and principles of Dimensional Modeling:

* **Load** atomic data into dimensional structures.
* **Build** dimensional models around business processes.
* **Need** to ensure that every fact table has an associated date dimension table.
* **Ensure** that all facts in a single fact table are at the same grain or level of detail.
* It’s **essential** to store report labels and filter domain values in dimension tables
* **Need** to ensure that dimension tables use a surrogate key
* **Continuously balance** requirements and realities to deliver business solution to support their decision-making

# Benefits of Dimensional Modeling

* **Standardization** of dimensions allows easy reporting across areas of the business.
* **Dimension tables** store the history of the dimensional information.
* It allows to introduce entirely new dimension **without major disruptions** to the fact table.
* Dimensional also to store data in such a fashion that it is **easier to retrieve** the information from the data once the data is stored in the database.
* Compared to the normalized model dimensional table are **easier to understand.**
* Information is grouped into **clear and simple** business categories.
* The dimensional model is very **understandable** by the business. This model is based on business terms, so that the business knows what each fact, dimension, or attribute means.
* Dimensional models are **deformalized** and **optimized** for fast data querying. Many relational database platforms recognize this model and optimize query execution plans to aid in performance.
* Dimensional modelling in data warehouse creates a schema which is optimized for **high performance**. It means fewer joins and helps with minimized data redundancy.
* The dimensional model also helps to boost **query performance**. It is more de-normalized therefore it is optimized for querying.
* Dimensional models can **comfortably accommodate change**. Dimension tables can have more columns added to them without affecting existing business intelligence applications using these tables.

# What is Multi-Dimensional Data Model in Data Warehouse

**Multidimensional data model** in data warehouse is a model which represents data in the **form of data cubes**. It allows to model and view the data in multiple dimensions and it is defined by dimensions and facts. Multidimensional data model is generally categorized around a **central theme** and represented by a fact table.

# Summary

* A dimensional model is a data structure technique optimized for Data warehousing tools.
* Facts are the measurements/metrics or facts from your business process.
* Dimension provides the context surrounding a business process event.
* Attributes are the various characteristics of the dimension modelling.
* A fact table is a primary table in a dimensional model.
* A dimension table contains dimensions of a fact.
* There are three types of facts 1. Additive 2. Non-additive 3. Semi- additive .
* Types of Dimensions are Conformed, Outrigger, Shrunken, Role-playing, Dimension to Dimension Table, Junk, Degenerate, Swappable and Step Dimensions.
* Five steps of Dimensional modeling are 1. Identify Business Process 2. Identify Grain (level of detail) 3. Identify Dimensions 4. Identify Facts 5. Build Star
* For Dimensional modelling in data warehouse, there is a need to ensure that every fact table has an associated date dimension table.

# Task 5: Operations in Database

This process includes the typical operations involved in selecting the required data; extracting data from multiple sources; operating on the data so that data from multiple sources is compatible, and loading this data into a data warehouse for analytical purposes.

**SETL – Select, Extract, Transform and Load**

**Select** – Identification of the data that you want to analyze  
**Extract** – Connecting to the particular data source and pulling out the data  
**Transform**– Modifying the extracted data to standardize it  
**Load** – Push the data into the data warehouse

# Overview

**ETL**stands for “extract, transform, and load.”

The process of ETL plays a **key role** in **data integration strategies.** ETL allows businesses to gather data from multiple sources and **consolidate** it into a single, centralized location. ETL also makes it possible for **different types** of data to work together.

A typical ETL process collects and refines different types of data, then delivers the data to a data warehouse such as Redshift, Azure, or BigQuery. ETL also makes it possible to **migrate data** between a variety of sources, destinations, and analysis tools. As a result, the ETL process plays a **critical role** in producing business intelligence and executing broader data management strategies.

# How ETL works

#### **Step 1: Extraction**

Few businesses rely on a single data type or system. Most manage data from a **variety of sources** and use a number of data analysis tools to produce **business intelligence**. To make a complex data strategy like this work, the data must be able to travel freely between systems and apps.

Before data can be moved to a new destination, it must first be extracted from its source. In this**first step** of the ETL process, structured and unstructured data is imported and consolidated into a single repository. Raw data can be extracted from a wide range of sources, including:

* Existing databases and legacy systems
* Cloud, hybrid, and on-premises environments
* Sales and marketing applications
* Mobile devices and apps
* CRM systems
* Data storage platforms
* Data warehouses
* Analytics tools

**Although it can be done manually, hand-coded data extraction can be time-intensive and prone to errors.**[**ETL tools**](https://www.talend.com/resources/etl-tools/)**automate the extraction process and create a more efficient and reliable workflow.**

#### **Step 2: Transformation**

During this phase of the ETL process, rules and regulations can be applied that ensure data quality and accessibility. You can also apply rules to help your company meet reporting requirements. The process of [data transformation](https://www.talend.com/resources/data-transformation-defined/) is comprised of several sub-processes:

* **Cleansing** — inconsistencies and missing values in the data are resolved.
* **Standardization** — formatting rule are applied to the data set.
* **Deduplication** — redundant data is excluded or discarded.
* **Verification** — unusable data is removed and anomolies are flagged.
* **Sorting** — data is organized according to type.
* **Other tasks** — any additional/optional rules can be applied to improve data quality.

Transformation is generally considered to be the most important part of the ETL process. Data transformation improves [data integrity](https://www.talend.com/resources/what-is-data-integrity/) and helps ensure that data arrives at its new destination fully compatible and ready to use.

#### **Step 3: Loading**

The final step in the ETL process is to **load the newly transformed data** into a **new destination**. Data can be loaded all at once (full load) or at scheduled intervals (incremental load).

**Full loading** — In an ETL full loading scenario, everything that comes from the transformation assembly line goes into new, unique records in the [data warehouse](https://www.talend.com/resources/what-is-data-warehouse/). Though there may be times this is useful for research purposes, full loading produces data sets that grow exponentially and can quickly become difficult to maintain.

**Incremental loading** — A less comprehensive but more manageable approach is incremental loading. Incremental loading compares incoming data with what’s already on hand, and only produces additional records if new and unique information is found. This architecture allows smaller, less expensive data warehouses to maintain and manage [business intelligence](https://www.talend.com/resources/what-is-business-intelligence/).

# ETL and business intelligence

**Data strategies** are more **complex** than they’ve ever been, and companies have access to more data from more sources than ever before. **ETL** makes it possible to transform **vast quantities** of data into **actionable** business intelligence. Consider the amount of data [available to a manufacturer](https://www.talend.com/customers/johnson-controls/). In addition to the data generated by sensors in the facility and the machines on an assembly line, the company also collects marketing, sales, logistics, and financial data.

All of that data must be extracted, transformed, and loaded into a new destination for analysis. In this scenario, ETL helps create business intelligence by:

#### **Delivering a single point-of-view**

Managing multiple data sets demands time and coordination, and can result in inefficiencies and delays. ETL combines databases and various forms of data into a **single, unified view**. This makes it easier to analyze, visualize, and make sense of large data sets.

#### **Providing historical context**

ETL allows an enterprise to **combine** legacy data with data collected from new platforms and applications. This produces a long-term view of data, so that older data sets can be viewed alongside more recent information.

#### **Improving efficiency and productivity**

ETL Software automates the process of hand-coded[**data migration**](https://www.talend.com/resources/what-is-data-integration/). As a result, developers and their teams can spend more time on innovation, and less time managing the painstaking task of writing code to move and format data

# Building your ETL strategy

**ETL** can be accomplished in one of **two ways**. In some cases, businesses may task their developers with building their own ETL. However, this process can be time-intensive, prone to delays, and expensive.

Most companies today rely on an ETL tool as part of their **data integration process**. ETL tools are known for their **speed**, **reliability**, and **cost-effectiveness**, as well as their **compatibility** with broader data management strategies. ETL tools also incorporate a broad range of data quality and data governance features.

When evaluating an ETL tool, you’ll want to consider the number and variety of connectors you’ll need, as well as its **portability** and **ease of use**. You’ll also need to determine if an open-source tool is right for your business, since these typically provide **more flexibility** and help users avoid vendor lock-in.

[**Talend Data Fabric**](https://www.talend.com/products/data-fabric/) provides a complete suite of apps that connect all your data, no matter the source or destination.

# Task 6: Data Integrity

To ensure data integrity in the data warehouse, following constraints are used:

1. **Entity**
2. **Referential**
3. **Semantics**

# Overview

The data in a **relational**[**database**](https://www.geeksforgeeks.org/what-is-database/)is stored in **form of a table**. A table makes the data look **organized.** Yet in some cases we might face issues while working with the data like repetition. We might want enforce rules on the data to avoid such technical problems. These rules are called **constraints**. A constraint can be defined as a rule that has to enforced on the data to avoid faults. There are three kinds of constraints: entity, referential and semantic constraints.

# Types of constraints

The three types of constraints –

### **1. Entity constraints :**

These constraints are given within one table. The entity constraints are primary key, foreign key, unique.

**Example :**

create table student (rollnumber int primary key, name varchar2(30), course varchar2(10));

Insert into student values(111, 'ABC', 'Chemical');

Insert into student values(112, 'JJP', 'Mech');

| **Rollnumber** | **Name** | **Course** |
| --- | --- | --- |
| 111 | ABC | Chemical |
| 112 | JJP | Mech |

These values are inserted in the table. Suppose a value given below is inserted :

Insert into student values(111, 'MAB', 'EEE');

It gives an error as the roll-number is enforced a primary key constraint that refrains from duplication. These constraints ensures to maintain uniqueness in the tables to avoid duplication’s.

### **2. Referential constraints :**

These constraints are used for referring other tables to enforce conditions on the data. The widely used referential constraint is foreign key.

**Example :**

create table marks (rollnumber int, name varchar2(30), course varchar2(30)

references student, marks int);

A table is created with a constraint that the marks should be rewarded to those students that are pursuing courses stated in the student table only. If a user tries to enter a value that doesn’t exist, it returns an error.

### **3. Semantic constraints :**

Datatypes are the semantic constraints enforced in a table. Datatypes help the data segregate according to its type.

**Example :**  
A name is a combination of different letters. We can place the name column in the char datatype yet char doesn’t satisfies the condition thereby varchar is preferably used for name.

name varchar2(30);

# Difference between Entity constraints , Referential constraints and Semantic constraints

| **CHARACTERISTICS** | **ENTITY CONSTRAINTS** | **REFERENTIAL CONSTRAINTS** | **SEMANTIC CONSTRAINTS** |
| --- | --- | --- | --- |
| **Definition** | Entity constraints are posed within a table. | Referential constraints are enforced with more than one table. | Semantic constraints are enforced on the values of a specific attribute |
| **Kinds** | The entity constraints are: unique, primary key, NULL. | The referential constraints are foreign key. | The semantic constraints are the datatypes. |
| **Description** | These constraints are used to enforce uniqueness in a table( while NULL is used to define no value) | These constraints are used for referring to another table for analysis of the data. | These constraints are used to divide a set of particular value based on a category. |
| **Functions** | These constraints ensure non duplicate’s in a database. | These constraints ensure that consistency of a database. | These constraints ensure that values are categorized accordingly to avoid confusions. |
| **Syntax** | Primary key: create table( column1 datatype1 primary key…) | Foreign key: create table( column1 datatype1 references tablename…) | column1 varchar2(30) |
| **Examples** | No two students can be designated the same rollnumber. | Rollnumber being referred to the marks table | the marks table. Name is assigned to varchar2 with a precision of 50. |

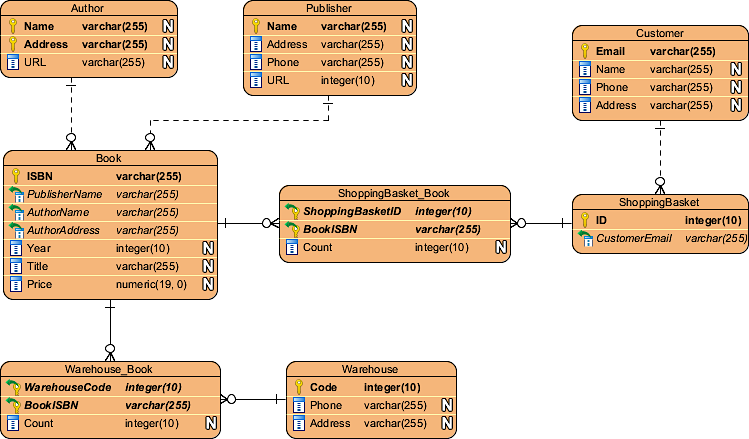
# Task 7: ERD

An **Entity-Relationship Diagram**, or ERD, can be thought of as a **map of the database schema**. We can visualize the structure of the entire schema and answer the following questions just by looking at the ERD:

* What tables does it contain?
* What columns does each table contain?
* What are the data types and constraint(s) (if any) for each column?
* What are the relationships between the various tables?

# What is ERD?

Database is absolutely an **integral part** of software systems. To fully utilize ER Diagram in database engineering guarantees you to produce **high-quality** database design to use in **database creation**, **management**, and **maintenance**. An ER model also provides a **means for communication**.



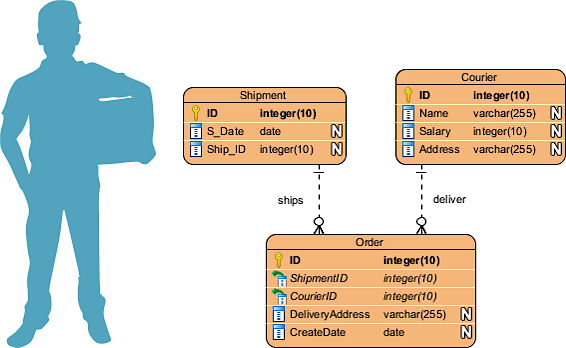
Today we’re going to walk you through everything you need to know about ER Diagramming. By reading this ERD guide, you will get the essential knowledge and skills about ER Diagrams and database design. You will learn things like what is ERD, why ERD, ERD notations, how to draw ERD, etc. along with a bunch of ERD examples.

# What is an ER diagram (ERD)?

**Entity Relationship Diagram**, also known as ERD, ER Diagram or ER model, is a type of structural diagram for use in database design. An ERD contains different symbols and connectors that visualize two important information: **The major entities within the system scope**, and the **inter-relationships among these entities**.

And that’s why it’s called “Entity” “Relationship” diagram (ERD)!

When we talk about entities in ERD, very often we are referring to business objects such as people/roles (e.g. Student), tangible business objects (e.g. Product), intangible business objects (e.g. Log), etc. “Relationship” is about how these entities relate to each other within the system.



In a typical ER design, you can find symbols such as rounded rectangles and connectors (with different styles of their ends) that depict the entities, their attributes, and inter-relationships.

# When to draw ER Diagrams?

So, when do we draw ERDs? While ER models are mostly developed for **designing relational databases** in terms of concept visualization and in terms of **physical database design**, there are still other situations when ER diagrams can help. Here are some typical use cases.

* **Database design** – Depending on the scale of change, it can be risky to alter a database structure directly in a DBMS. To avoid ruining the data in a production database, it is important to plan out the changes carefully. ERD is a tool that helps. By drawing ER diagrams to visualize database design ideas, you have a chance to identify the mistakes and design flaws, and to make corrections before executing the changes in the database.
* **Database debugging** – To debug database issues can be challenging, especially when the database contains many tables, which require writing complex SQL in getting the information you need. By visualizing a database schema with an ERD, you have a full picture of the entire database schema. You can easily locate entities, view their attributes and identify the relationships they have with others. All these allow you to analyze an existing database and to reveal database problems easier.
* **Database creation and patching** – Visual Paradigm, an ERD tool, supports a database generation tool that can automate the database creation and patching process by means of ER diagrams. So, with this ER Diagram tool, your ER design is no longer just a static diagram but a mirror that reflects truly the physical database structure.
* **Aid in requirements gathering** – Determine the requirements of an information system by drawing a conceptual ERD that depicts the high-level business objects of the system. Such an initial model can also be evolved into a physical database model that aids the creation of a relational database, or aids in the creation of process maps and data flow modes.

# ERD notation guide

An ER Diagram contains entities, attributes, and relationships. In this section, we will go through the ERD symbols in detail.

### **Entity**

An ERD entity is a **definable thing or concept within a system**, such as a person/role (e.g. Student), object (e.g. Invoice), concept (e.g. Profile) or event (e.g. Transaction) (note: In ERD, the term “entity” is often used instead of “table”, but they are the same). When determining entities, think of them as nouns. In ER models, an entity is shown as a rounded rectangle, with its name on top and its attributes listed in the body of the entity shape. The ERD example below shows an example of an ER entity.

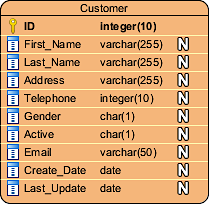


### **Entity Attribute**

Also known as a column, an attribute is a **property or characteristic of the entity that holds it**.

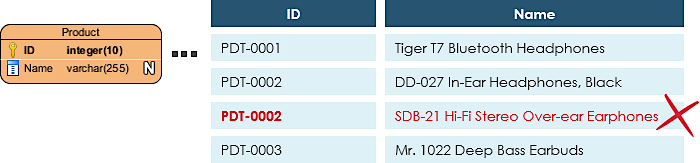
An attribute has a name that describes the property and a type that describes the kind of attribute it is, such as varchar for a string, and int for integer. When an ERD is drawn for physical database development, it is important to ensure the use of types that are supported by the target RDBMS.

The ER diagram example below shows an entity with some attributes in it.



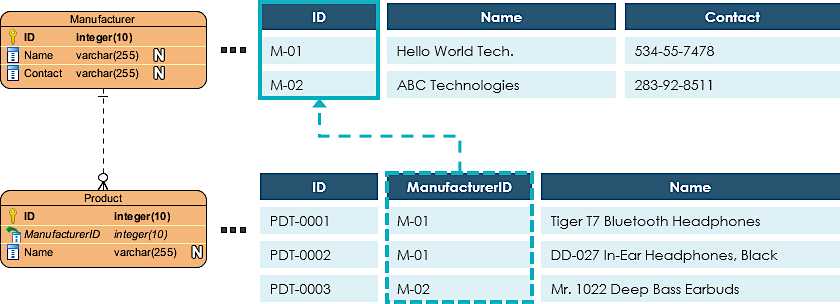
### **Primary Key**

Also known as PK, a primary key is a special kind of entity attribute that **uniquely defines a record in a database table**. In other words, there must not be two (or more) records that share the same value for the primary key attribute. The ERD example below shows an entity ‘Product’ with a primary key attribute ‘ID’, and a preview of table records in the database. The third record is invalid because the value of ID ‘PDT-0002’ is already used by another record.



### **Foreign Key**

Also known as FK, a foreign key is a **reference to a primary key in a table**. It is used to identify the relationships between entities. Note that foreign keys need not be unique. Multiple records can share the same values. The ER Diagram example below shows an entity with some columns, among which a foreign key is used in referencing another entity.



### **Relationship**

A relationship between two entities signifies that the **two entities are associated with each other somehow**. For example, a student might enroll in a course. The entity Student is therefore related to Course, and a relationship is presented as a connector connecting between them.

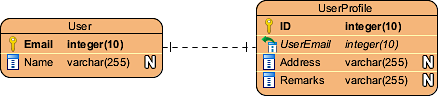
### **Cardinality**

Cardinality defines the **possible number of occurrences in one entity which is associated with the number of occurrences in another**. For example, ONE team has MANY players. When present in an ERD, the entity Team and Player are inter-connected with a one-to-many relationship.

In an ER diagram, cardinality is represented as a crow’s foot at the connector’s ends. The three common cardinal relationships are one-to-one, one-to-many, and many-to-many.

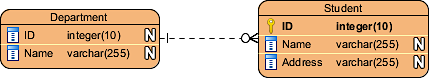
### **One-to-One cardinality example**

A one-to-one relationship is mostly used to split an entity in two to provide information concisely and make it more understandable. The figure below shows an example of a one-to-one relationship.



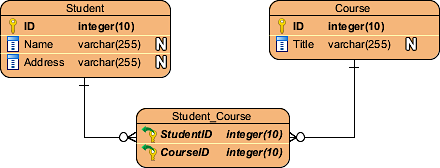
### **One-to-Many cardinality example**

A one-to-many relationship refers to the relationship between two entities X and Y in which an instance of X may be linked to many instances of Y, but an instance of Y is linked to only one instance of X. The figure below shows an example of a one-to-many relationship.



### **Many-to-Many cardinality example**

A many-to-many relationship refers to the relationship betweentwo entitiesX and Y in which X may be linked to many instances of Y and vice versa. The figure below shows an example of a many-to-many relationship. Note that a many-to-many relationship is split into a pair of one-to-many relationships in a physical ERD. You will know what a physical ERD is in the next section.



# Conceptual , Logical and Physical data models

An ER model is typically drawn at up to **three levels** of abstraction:

* [Conceptual ERD / Conceptual data model](https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/#erd-data-models-conceptual)
* [Logical ERD / Logical data model](https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/#erd-data-models-logical)
* [Physical ERD / Physical data model](https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/#erd-data-models-physical)

While all the three levels of an ER model contain entities with attributes and relationships, they **differ in the purposes** they are created for and the audiences they are meant to target.

A general understanding to the three data models is that business analyst uses a **conceptual** and **logical model to model** the business objects exist in the system, while database designer or database engineer elaborates the conceptual and logical ER model to produce the physical model that presents the physical database structure ready for database creation. The table below shows the difference between the three data models.

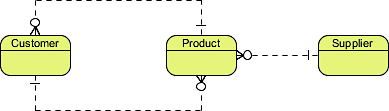
Conceptual model vs Logical model vs Data model:

| **ERD feature** | **Conceptual** | **Logical** | **Physical** |
| --- | --- | --- | --- |
| Entity (Name) | Yes | Yes | Yes |
| Relationship | Yes | Yes | Yes |
| Columns |  | Yes | Yes |
| Column’s Types |  | Optional | Yes |
| Primary Key |  |  | Yes |
| Foreign Key |  |  | Yes |

### **Conceptual Data model**

Conceptual ERD models the **business objects that should exist in a system and the relationships between them**. A conceptual model is developed to present an overall picture of the system by recognizing the business objects involved. It defines what entities exist, NOT which tables. For example, ‘many to many’ tables may exist in a logical or physical data model but they are just shown as a relationship with no cardinality under the conceptual data model.

#### Conceptual data model example

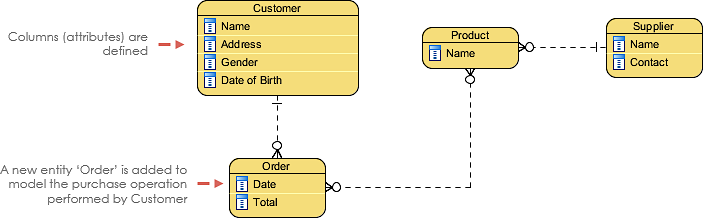


NOTE: Conceptual ERD supports the use of generalization in modeling the ‘a kind of’ relationship between two entities, for instance, Triangle, is a kind of Shape. The usage is like generalization in UML. Notice that only conceptual ERD supports generalization.

### **Logical Data model**

Logical ERD is a **detailed version of a Conceptual ERD**. A logical ER model is developed to enrich a conceptual model by defining explicitly the columns in each entity and introducing operational and transactional entities. Although a logical data model is still independent of the actual database system in which the database will be created, you can still take that into consideration if it affects the design.

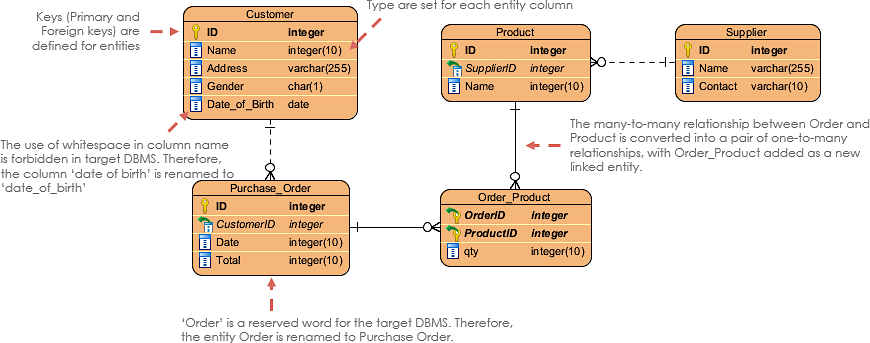
#### Logical data model example



### **Physical Data model**

Physical ERD represents the **actual design blueprint of a relational database**. A physical data model elaborates on the logical data model by assigning each column with type, length, nullable, etc. Since a physical ERD represents how data should be structured and related in a specific DBMS it is important to consider the convention and restriction of the actual database system in which the database will be created. Make sure the column types are supported by the DBMS and reserved words are not used in naming entities and columns.

#### Physical data model example



# How to draw an ER diagram

If you find it difficult to get started with drawing an ER diagram, don’t worry. In this section, we will give you some ERD tips. Try to follow the steps below to understand how to draw an ER diagram effectively.

1) Make sure you are clear about the **purpose of drawing** the ERD. Are you trying to present an overall system architecture that involves the definition of business objects? Or are you developing an ER model ready for database creation? You must be clear about the purpose to develop an ER diagram at the right level of detail (Read the section Conceptual, Logical and Physical Data Models for more details)

2) Make sure you are clear about the **scope to model**. Knowing the modeling scope prevents you from including redundant entities and relationships in your design.

3) Draw the **major entities** involved in the scope.

4) **Define** the properties of entities by adding columns.

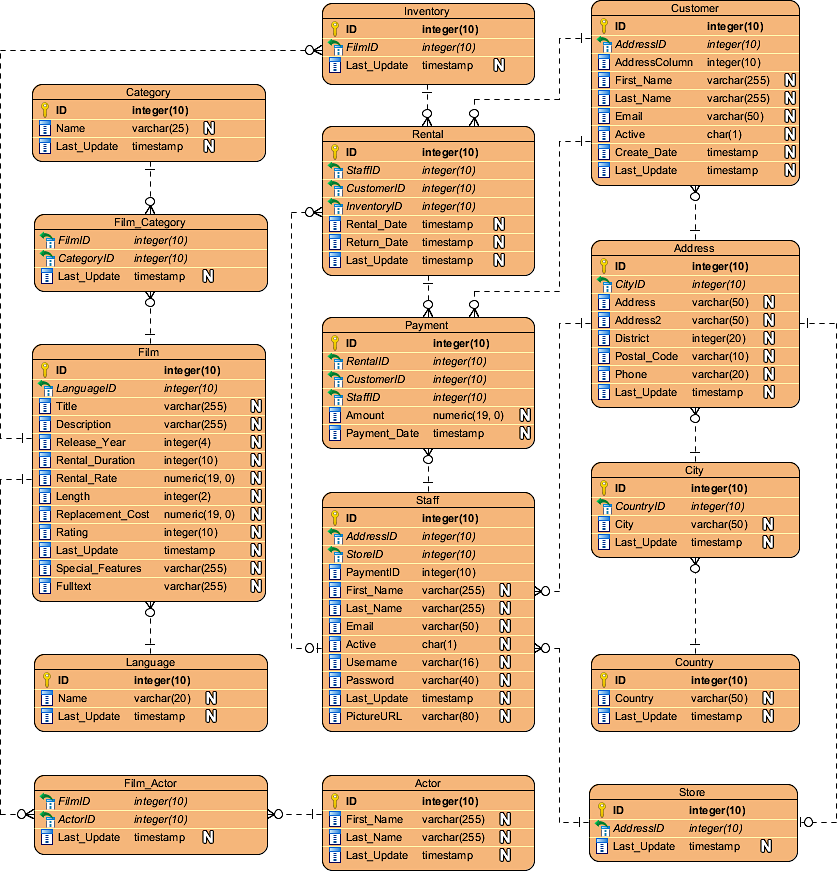
5) **Review** the ERD carefully and check if the entities and columns are enough to store the data of the system. If not, consider adding additional entities and columns. Usually, you can identify some transactional, operational and event entities in this step.

6) Consider the **relationships** between all entities and relate them with proper cardinality (e.g A one-to-many between entity Customer and Order). Don’t worry if there are orphan entities. Although it’s not common, it’s legit.

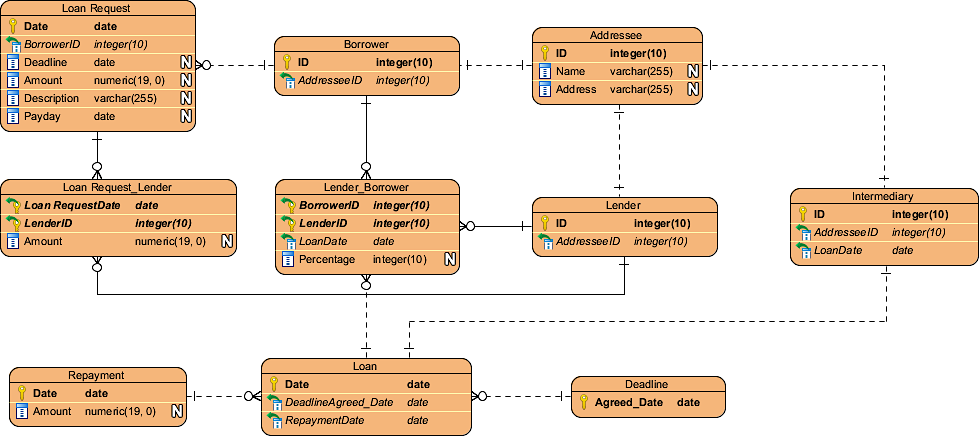
7) **Apply the technique** of database normalization to re-structure the entities in a way that can reduce data redundancy and improve data integrity. For example, the details of the manufacturer might be stored under the Product entity initially. During the process of normalization, you may find that the detail keeps repeating records over records, then you can split it as a separate entity Manufacturer, and with a foreign key that links between Product and Manufacturer.

# Data model examples

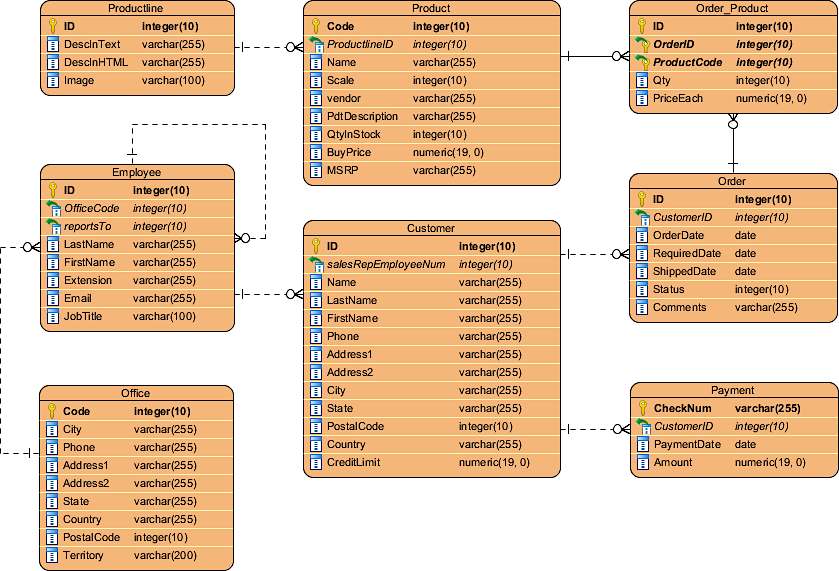
### ERD example – Movie Rental System



### ERD example – Loan System

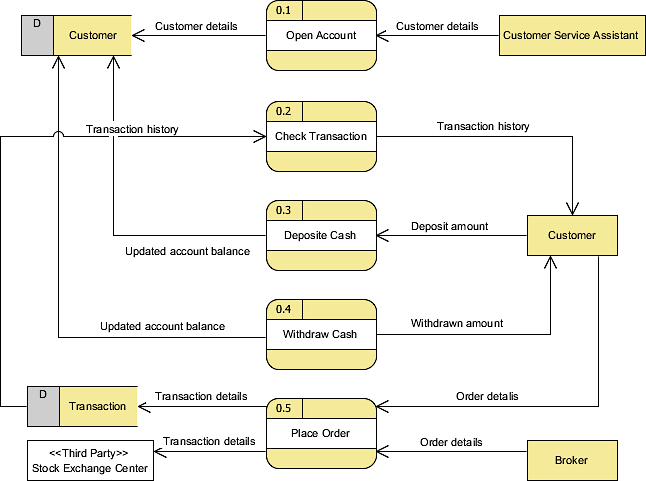


### ERD example – Online Shop

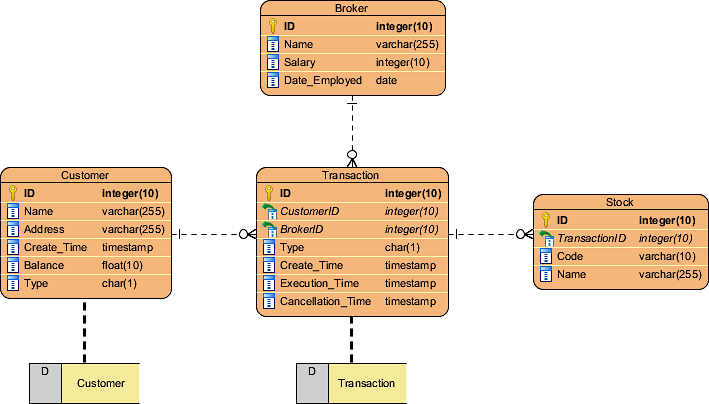


# Using ERD and Data Flow Diagram (DFD)

In system analysis and design,[**Data Flow Diagram (DFD)**](https://www.visual-paradigm.com/features/data-flow-diagram-tool) can be drawn to **visualize** the flow of information within system processes. In a Data Flow Diagram, there is a symbol called **Data Store**, which represents a database table that provides the information needed by the system.

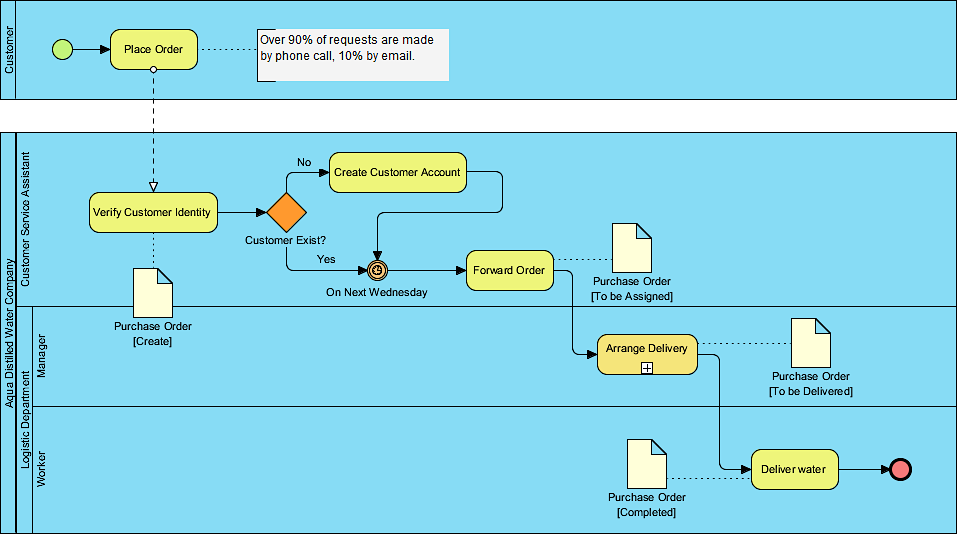


Since a physical ER Diagram provides a **blueprint** of an actual database, the entities in such an ERD are aligned with datastores in a DFD. You can draw ERD as a complement to DFD by representing the structure of information that flows within a system, or, on the contrary, to draw DFD in complementing an ERD by showing how the data will be utilized by the system in runtime.

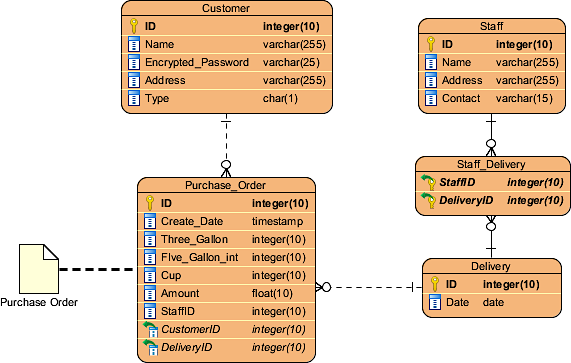


# Using ERD and BPMN Business Process Diagram (BPD)

In business process mapping,[**BPMN Business Process Diagram (BPD)**](https://www.visual-paradigm.com/features/bpmn-diagram-and-tools) can be drawn to **visualize business workflows**. In a Business Process Diagram, there is a symbol called **Data Object,** which represents the data input into / output from process activities.



Since a conceptual and logical data model provides a **high-level view** of business objects within a system, the entities in such ERDs are aligned with data objects in BPD. You can draw ERD as a complement to BPD by representing the structure of data objects needed by a business workflow, or, on the contrary, to draw BPD in complementing an ERD by showing how the data will be utilized throughout a business process.



# Choosing an ERD tool

It takes time and effort to develop a data model with ERD. A helpful database design tool should be able to **reduce your time** and **effort spent**. **Visual Paradigm** provides you with not only an ERD tool but also a set of visual modeling features that helps you draw **faster** and **easier.** It supports most of the popular relational database management systems in the market today both in terms of database design, database generation, and ERD reversal.



The ERD designer is available in [Visual Paradigm Modeler](https://www.visual-paradigm.com/editions/modeler/), which costs [only US $6 per month](https://www.visual-paradigm.com/shop/vp.jsp). We would recommend you [download and have a try](https://www.visual-paradigm.com/download/). 30 days of FREE evaluation is offered. No credit card required.

Top of Form

Bottom of Form