INTRODUCTION TO DATABASE SYSTEMS

LibreTexts™

Introduction to Database Systems

The LibreTexts libraries are Powered by NICE CXOne and are supported by the Department of Education Open Textbook Pilot Project, the UC Davis Office of the Provost, the UC Davis Library, the California State University Affordable Learning Solutions Program, and Merlot. This material is based upon work supported by the National Science Foundation under Grant No. 1246120, 1525057, and 1413739.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation nor the US Department of Education.

TABLE OF CONTENTS

Licensing

*A detailed breakdown of this resource's licensing can be found in **Back Matter/Detailed Licensing**.*

CHAPTER OVERVIEW

1: Introduction to Database Systems and SQL

1.1: Introduction and Background

How were things managed before technology became a major influence in our lives? Tracking information was difficult before the employment of databases. There was plenty of room for error with the pen and paper method. It was not until the 1960s when databases were used from a computer-based format. However, most computerized databases still use the principles and methods developed in the previous age.

Databases now are used everywhere to store information. Whether it be in a customer management system or tracking bank information, databases are utilized to store the necessary data for later use. Data is structured in rows and columns featuring different fields for queries and stored in multiple tables to showcase the relationship between them. According to Oracle.com, “Databases have evolved dramatically since their inception in the early 1960s” (Oracle). In the beginning, only navigational databases, such as the hierarchical and network database, were employed. As time went on, new types of databases were created based on the needs of organizations and the management of their data.

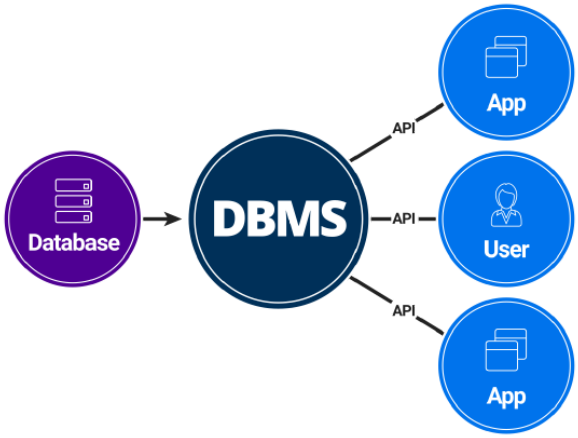


Figure 1.1.1: Database Management. (Copyright; https://www.smartsheet.com/database-management)

In the image shown above, a database is used in a database management system (DBMS) for short, is a form of software that allows an organization to access and manipulate data that will be showcased in a form that is unable to be changed by other applications and users.

1.2: Limitations of Conventional File Processing

Files are used to store specific data for future use and recollection. When computers first became mainstream, files were stored like paper, in the form of flat files. This information was collected in notepads separated by spaces, commas, semicolons or other symbols. Organization of files was often based on their categories, consisting of only related information with specific names. The downside to this is that you were unable to open the files without using a specific coding language to edit it. While it appeared convenient at the time, it is easy to identify the many disadvantages to using this system.

1.3: Data Redundancy

One of the major problems with this system was data redundancy and inconsistency. Since the files and programs jammed into files were created by several different programmers over a long period of time, the files were certain to be in different formats, involving several different programming languages. Most of the information is also constantly duplicated due to how tedious it would be to access others' code and double check the information. For example, if a customer of a bank has two accounts, the data accompanied by these accounts would be stored in two separate files in order to satisfy both accounts as they are made. This leads to data redundancy. This would lead to bigger storage sizes for the same information, increasing the cost.

1.4: Data Accuracy

The countless copies of this data could also have discrepancies, making it impossible to know which information is accurate. Whenever a new value needs to be entered into the database, every single file with this data has to be updated to prevent this. This would lead to tedious work that wasn't 100% accurate in the end. For example, a company could have stored customer data, including name, address, and city. There could be a request in which the record of a customer who lives in a specific city is needed. In order to achieve this, a new program would need to be written and executed, and the file containing the customer's city had to be accessed. Every single customer who belonged to this city would need to be specifically selected and taken out into this new program in order to organize the data. This is neither convenient nor reliable. These copies also contributed to the difficulty involving the creation of new applications, as they may be unable to find the appropriate data. This also ensured atomicity didn't work. Atomicity is a sequence of database processes such that either all occur, or nothing occurs. This could be used to prevent updates to a database occurring only partially; however, atomicity is unable to work unless it is able to read and write to every single file, which in this structure, is extremely difficult.

There was also a difficulty in accessing data due to the "spaghetti code" structure of this system. If a specific set of information is needed to be organized in a new way, unless it was anticipated prior to the initially being created, it was nearly impossible to achieve this. The application needed to display the information in the requested way would not have existed. This system doesn't allow data to be retrieved in a convenient manner, leading to different systems created down the line.

Integrity problems were also created due to the data values in a database needing to satisfy certain types of consistency constraints. Since most of the code involving these files is in different languages, it is almost impossible to change them all to enforce new constraints. The file system also lacks concurrent access. In modern systems, multiple users can update the data simultaneously. This is to ensure a faster response time and to improve the overall performance of the system. The involvement of multiple users may result in inconsistent data, which is normally prevented using supervision. However, in a file processing system, this supervision is lackluster due to the several applications and various languages. It all leads to the same problems in the end.

1.5: Data Security

Security is also a major issue in this system. In a database, every user in the database system shouldn’t be able to access all the data. Each user should be delegated and only allowed to access specific data requiring a password of sorts. In a file processing system since different programmers add their own application, there is either a universal password or so many passwords that the information is scrambled and the people requiring it can’t access it. Since every new file is only added when needed, it is difficult to constantly change the permission for each individual file in order to ensure security standards.

These disadvantages would lead many to convert to a database approach rather than a file system. A database corrected many of these errors reducing the development time and increasing the data integrity of every file. It is true that file processing systems were full of many errors, but they are known as a stepping stone towards more perfected systems of data storage.

1.6: Advantages of Databases

In today's world, data is prevalent in every aspect of our lives as human beings. Data is constantly being created, organized, and stored. With all this data being transferred and exchanged around the world, it is important to have an efficient and organized method to storing this data. This is where databases come in. Databases offer improved efficiency and versatility, they allow categorization and structuring of available data, and they allow multi-user access, creating an organized work environment and newer and better ways to manage data.

Efficiency comes into play specifically with businesses. Databases can handle large amounts of data as well as multiple types of data. Businesses can use databases to have data easily accessible to make operational decisions on a daily basis.

Versatility is also important in terms of accessing data. Databases can be accessed via desktop, laptop, tablet and even mobile devices. This is incredibly helpful in a time where so much importance is placed on accessing things immediately, as data can be easily retrieved at any moment. This benefit is applicable to consumers as well as businesses.

Categorization and organization are both major advantages. They allow the structuring of information in ways that are easily understandable and accessed. Certain DBMS allow relationships between entities in order to simplify the organization of data.

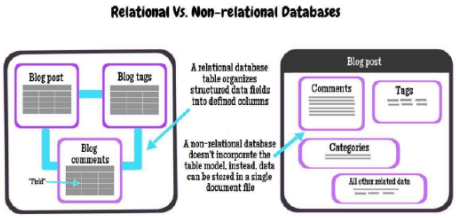


Figure 1.6.1: Databases for Front-End Developers (Copyright; Liz Parody;Medium.com)

Accessing data in a multitude of ways by multiple different users is also a huge advantage that databases have; this is called multi-access. Multi-access is what allows multiple authorized users to have access to the same data. For example, a human resources manager at a company will have access to the same set of potential hires at a certain location as the general manager of that same location. The picture below visually describes the relationship between this shared data and the users that have access to it. (WD, 2005)

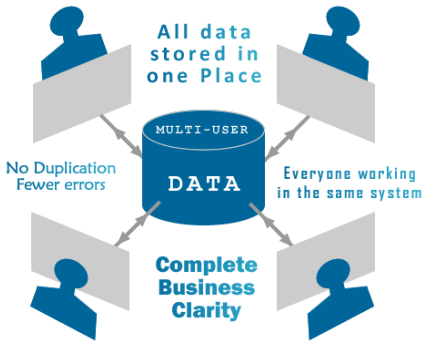


Figure 1.6.1: Relationship Between Shared Data and the Users . (Copyright; https://www.workingdata.co.uk/spread...-1-multi-user/)

Databases offer businesses a smoother operating work situation. The implementation of a database management language such as SQL (Structured Query Language) allows businesses to access and modify data that is stored in a relational database.

Databases are constantly being used and accessed in new ways. With all the advantages that databases offer, uses will continue to grow. The accessibility, versatility and efficiency that a database can provide when paired with a DBMS is the reason why so many successful businesses are using them to this day.

1.7: Costs and Risks of Database Approach

There are obviously many advantages that benefit those who implement a database approach. Organization, efficiency and structure are all some positive elements that can be attributed to the database approach. However, there is always give and take, and there are some risks and costs involved with the database approach as well.

For example, when you decide to implement a database system, you now require personnel who know how to implement and maintain this system. This will most definitely be a significant cost that will be directly attributed to the implementation of the system. There also lies the cost in training individuals who may be new to your system that has already been implemented, and this will not be cheap either. The graph below illustrates the cost one may be dealing with personnel-wise on an annual basis when implementing a Database system.

Another significant cost of the Database approach is the cost of installation and maintenance. When implementing a new database system, it is costly to pay personnel to install and operate it, especially if it is a large and complex database. Installation isn't where the cost stops; maintenance is needed in order to keep that system running, and as you want to expand and maintain, over time, you will require additional hardware. The chart below displays the different costs of operating a given datacenter per month.

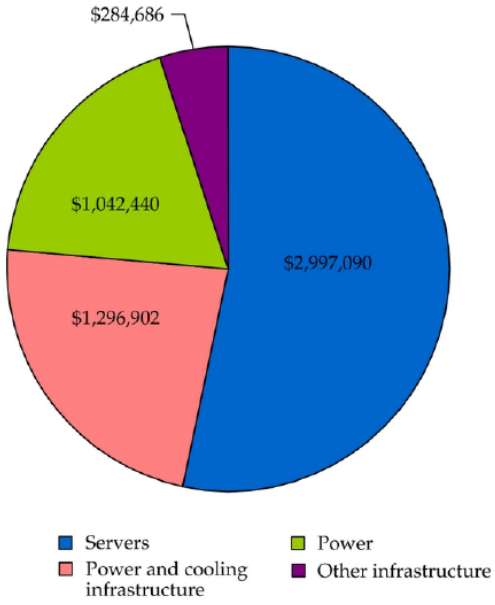


Figure 1.7.1: Costs of Operating a Datacenter Per Month. (Copyright; https://www.researchgate.net/figure/...fig3_258385511)

One must also account for the cost of migration: the cost to transfer the data and functionality of the previous file system over to the new database system accurately and without loss. It may seem as though it is a simple concept, but it is very difficult in a lot of cases and thus costs a substantial amount of money and time to execute.

There are also the costs and risks involved in needing specific backup and recovery systems. In a shared corporate database, there will be large amounts of data being stored. However, there must be backup data in case of software, hardware, or human error. If the data is not backed up, depending on the use of the database, the results could be catastrophic.

Lastly, there is organizational conflict. When implementing a large database with large complexity, it is common for people within the organization to have opposing views on how the data should be stored and how the system should be running. The costs and risks here lie in hiring strong leadership. In order to reach agreements on data definitions and how responsibilities are delegated for accurate data maintenance, the leadership must be strong and defined.

1.8: Components of a Database Environment

A database environment has 5 major components for functionality. The components necessary are: people, hardware, software, data, and procedures (OwlGen, 2019).

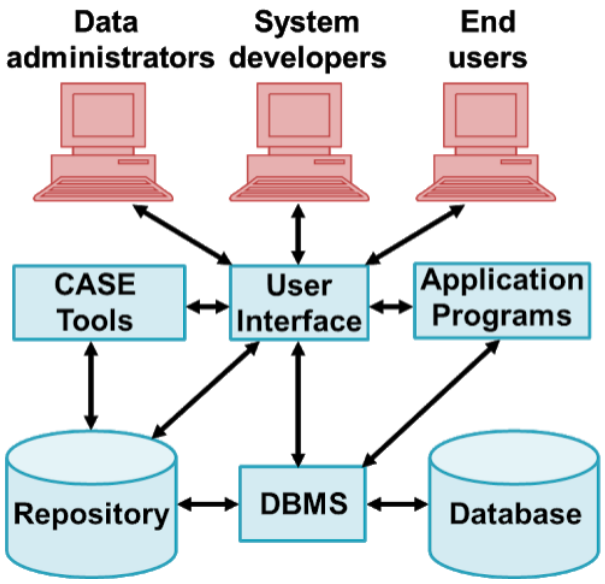


Figure 1.8.1: The 5 Major Components of a Database (Copyright; Source:http://cdn.wagmob.com/subject/G124/h...o00dbms_1.html)

When it comes to people, there are different roles needed to help build the overall database environment. These roles would include but are not limited to system and database administrators, database designers, programmers, analysts, and end users (OwlGen, 2019).

The system administrator is in charge of setting up and managing the system and server. They are needed to make sure there are no server crashes or any missing information within the database (Gite, 2014). Database administrators ensure the physical database is working properly through monitoring the performance and also managing security access and other standards ("What are the functions of a database administrator"). Database designers and programmers code all queries, relationships, and data and make sure they are stored properly within the database management system. Analysts review all the data the designers and programmers have implemented. Finally, end users are the ones that utilize the database management system and make the system more usable for other users.

Hardware and software are the items that make the database environment come to life. Hardware includes the actual computer itself and any sort of networking components needed. Software includes the operating system and any sort of programs needed to build and administer the database (OwlGen, 2019).

Finally, data and procedures go hand in hand. Data includes things like the actual database needed to function in the environment as well as any business procedures and/or rules that manage the system. The procedures are implemented to structure the overall design on how the database should work and regulate all the data that should be going in and coming out of the database (OwlGen, 2019).

1.9: Database Systems Development Life Cycle

The database life cycle (DBLC) consists of six phases. These phases include database primary study planning, analysis, detailed System design, (prototyping), implementation and loading, testing and evaluation, operation, maintenance and evolution. In the database primary study, the researcher examines the current systems operations in the company to determine how and why the current system isn't sustainable. The objective of this study is to analyze the company status, define problems and constraints, define purpose, and define the scope and boundaries. Each section can be broken down in order to further understand the usefulness behind creating this study.

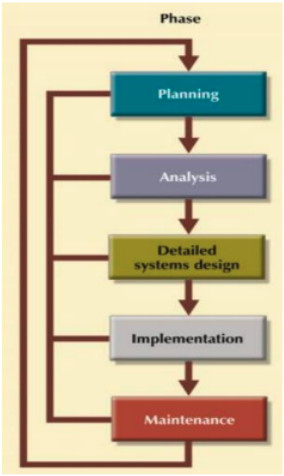


Figure 1.9.1: The Database Life Cycle (DBLC). (Copyright; author via source)

Analyzing the company situation	Pertains to defining the general conditions within a company, including its organization structure and its mission. In order to correctly do this, the designer must discover what the company's operation components are, the way they function, and how they interact
Defining Problems and Constraints	Pertains to the discovery of issues within the company, formally and informally. These problems may appear unstructured; however, problems are usually connected, allowing the designer to overcome them by the end of the process.
Defining Objectives	Is a part of the new proposed database system showing that it is designed to solve the major problems identified previously?
Defining the Scope and Boundaries	Pertains to the engineer recognizing the existence of their limits: scope and boundaries. The system's scope shows the extent of the design according to the requirements. The system also is connected to limits known as boundaries which are external. These boundaries are set by the accompanying hardware and software. Database design is the second phase focusing on the design of the database that supports company operations and objectives in the future. This can be viewed as the most critical DBLC phase.
Implementation and Loading	Pertain to a series of instructions when dealing with the creation of tables, attributes, etc. in the domain. In this phase, the design specifications are installed, creating the exact database required by the parent company. This can be done in 3 phases.
Install the DBMS	Installing a new instance of a DBMS in the system on a server.
Creating the DBMS	Creates the table spaces and file groups accompanied by the database.

Loading and Converting Data	After the database is created, the data must enter the new tables. This requires them to be merged and imported from other databases or the ones previously used in order to ensure the same data is relayed into the newer, better system. Testing and evaluation pertain to the decision made to ensure integrity, security, performance, and recoverability of the database. Following the plans laid out previously, this fine-tunes the database to ensure that it performs as expected. This phase is also divided into three phases, making it easy to follow and accurately test the functionality of the database.
Test the Database	During this step, the database is tested to ensure it has the integrity and security required by the company. This is enforced through the proper use of primary and foreign key rules.
Fine-Tune the Database	This is the editing of the database with the results of the previous step in mind. If no fine-tuning is required, this step can be skipped.
Evaluating the Database	The database must be reviewed thoroughly to ensure that the data contained is protected against loss, promoting the use of a backup.
Operation	The second to last step identifying that the database is fully functional. At this point, the database is complete, and the new system has space to evolve as needed by the developers.
Maintenance and Evolution	The final step. This step is directed by the database administrator allowing them to perform routine maintenance activities regarding the database. Some of these activities include Backup, Corrected Maintenance, Adaptive Maintenance and the Assignment of access permissions to welcome new users and edit old users

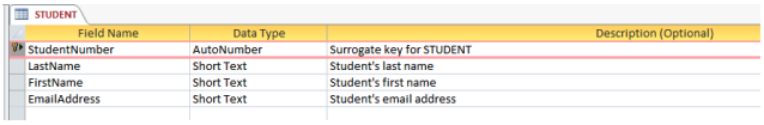
All together these steps make up the Database Life cycle and ensure that a fully functional database is created, allowing for around the clock maintenance within the company and promoting a highly efficient system the meets the guidelines presented at the beginning of the process.

1.10: Single-User and Multi-user Database Applications

Database tables are structured to store data, but a database is not complete unless it also shows the relationships among the tables. To see why this is important, examine Figure 1-4 below (Kroenke, Auer, Vandenberg, Yoder, 2018) the database includes all of the basic data shown together with a GRADE table. Unfortunately, the relationships among the data are missing.

In this format, the GRADE data are useless. It would be the equivalent to a sports commentator who simply announced: “Now for tonight’s baseball scores: 2–3, 7–2, 1–0, and 4–5.” The scores are useless without knowing the teams that earned them. Thus, a database contains both data and the relationships among the data.

This demonstrates is imperative characteristic of database processing. Each row in a table is distinctively identified by a primary key, and the values of these keys are used to create the relationships between the tables. For example, in the STUDENT table StudentNumber serves as the primary key. Each value of StudentNumber is unique and identifies a particular student. Thus, StudentNumber 1 identifies Sam Cooke. For example, ClassNumber in the CLASS table identifies each class. If the numbers used in primary key columns such as StudentNumber and ClassNumber are repeatedly created and assigned in the database itself, then the key is also called a surrogate key (Kroenke, Auer, Vandenberg, Yoder, 2018).


STUDENT

Field Name	Data Type	Description (Optional)
StudentNumber	AutoNumber	Surrogate key for STUDENT
LastName	Short Text	Student's last name
FirstName	Short Text	Student's first name
EmailAddress	Short Text	Student's email address

Figure 1.10.1: Sample Microsoft Access Student Record. (" Figure 1-1: Sample Microsoft Access Student Record " by Dr. Sarah North, Affordable Learning Georgia, Kennesaw State University is licensed under CC BY 4.0)

Figure 1-2 shows each row in a table in specifically known by a primary key, and value of those keys that are used to create a relationship between the tables, such as student IDNumber (primary key). If the numbers used StudentNumber and ClassNumber column and generate and assigned in the database, then the key is also called a surrogate key.

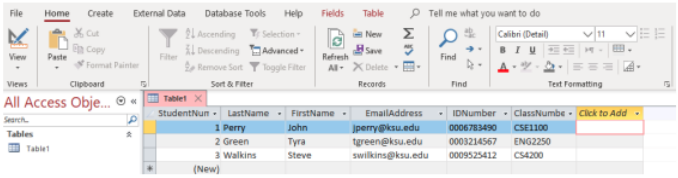
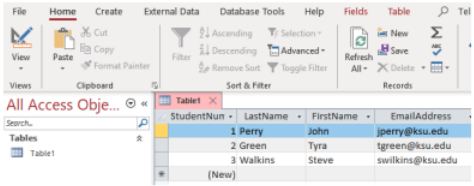
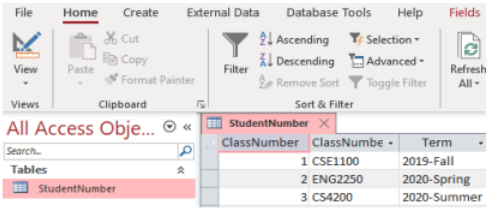


Figure 1.10.2: The Primary Key and Surrogate Key. (" Figure 1-2: The Primary Key and Surrogate Key" by Dr. Sarah North, Affordable Learning Georgia, Kennesaw State University is licensed under CC BY 4.0)

In the table below shows when more than one column in a table are merged to form of the primary key, is known as a composite key. In the GRADE column, StudentNumber and ClassNumber each now serve as a foreign key. A foreign key provides a relationship or link between two tables. Figure 1-3 shows a Microsoft Access 2016 point of view of the tables and their relationships.



StudentNumber Table



ClassNumber Table

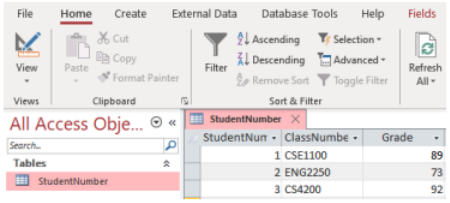


Figure 1.10.3: The Grade table with foreign keys – link to Student ClassNumber Table. (" Figure 1-3: The Grade table with foreign keys – link to Student ClassNumber Table" by Dr. Sarah North, Affordable Learning Georgia, Kennesaw State University is licensed under CC BY 4.0)

Figure 1-4 shows the greater database application, part of a customer relationship management (CRM) system, which manages customers and their contacts, purchases, support requests, and so forth. The CRM system uses software to support a larger company, which may include anywhere from 500 rows to 10 million or more. An enterprise resources planning (ERP) system is an information system that affects every department in a company, including sales, inventory, planning purchasing and other business purposes. SAP (System, Applications & Products in Data Processing) is the vendor used with ERP applications for large companies.

Application	User	Number of Users	Standard Size	Comments
Customer appointment (Doctor dentist)	Manger	20-15	500 rows	Marketing software
Customer relationship Management (CRM)	Senior Manger	10-15	10 million rows	Vendors applications such as Oracle
Data mining	Business Analysts	1-5	1000 to million rows	Data extracted and use by statistical data mining tools.

Figure 1.10.4: A larger database application. (" Figure 1-4: A larger database application" by Dr. Sarah North, Affordable Learning Georgia, Kennesaw State University is licensed under CC BY 4.0)

1.11: Concise Summary

Databases are the foundation of structuring data. When they were first implemented in the 1960s, data became easier to manage and structure. A database is now necessary to hold and manage user data and other personal information. Organizations utilize a database management system, or DBMS, for manipulating and storing the data into the databases while also managing the relationships between the data itself.

As computers and other technology began to become introduced, it became much simpler to manage data and store it in the DBMS. A database has many advantages, including efficiency, versatility, categorization, and organization to name a few. However, there are associated costs and risks to databases. The organization is now required to put funding towards training employees in managing and updating the DBMS as well as for general upkeep of the management system in order for it to remain stable. Various components of the DBMS to enable this include the people, the hardware, the software, the data itself, and the procedures needed to keep the database organized and well managed.

Finally, the life cycle of the database can be defined by six main phases: database initial study, database design, implementation and loading, testing and evaluation, operation, and maintenance and evolution. It is important to consider what needs to be implemented in a database, but also what role it can play for based on the needs of the organization

1.12: Extended Resources

❖ This link gives a detailed description on how database systems are managed and worked through. The speaker brings emphasis on the ER diagram and their relationship in structuring databases and queries. They also discuss database implementation in SQL Servers and briefly define what SQL (Structured Query Language) is. https://www.youtube.com/watch?v=n75iPNrzN-o

❖ This article briefly describes the seven commonly used types of database management systems, explains the origins on how they were structured, and describes how each DBMS is used. https://www.c-sharpcorner.com/Upload...nagement-syste ms/

❖ This article explains the seven best practices on how to protect and secure databases from hackers and other attackers. https://www.esecurityplanet.com/netw...practices.html

❖ This article briefly explains some of the common malpractices in designing a database and database management systems. https://www.toptal.com/database/data...-bad-practices

❖ This article goes into some of the specific types of user interfaces in database management systems. They are designed by mostly UI developers to utilize the information given in the database. They also explain each of the different types to give an idea of which is the best for the needs of each DBMS. https://www.geeksforgeeks.org/interfaces-in-dbms/

❖ Identifying Database Table Relationships One of the huge advantages of a relational database is that, once you have your data held in clearly defined, compact tables, you can connect or relate the data held in different tables. There are three types of relationships between the data you are likely to encounter at this stage in the design: one-to-one, one-to-many, and many-to-many. To be able to identify these relationships, you need to examine the data and have an understanding of what business rules apply to the data and tables. If you're not sure, it can be helpful to meet with someone who does have a thorough knowledge of the data. https://condor.depaul.edu/gandrus/24...ationships.htm

❖ Deciding on Tables and Fields for your Database Design: Each table in your database should hold the information on one subject. You might think of a subject as a collection of related information with common characteristics. For example, if you were creating a database to hold information about the operation of your ice cream stand, you might have an IceCream table. If you decided to sell sundaes as well as cones, you might add a Toppings table. Then, to associate ice cream and toppings in particular combinations and record the prices, you might add a Sundaes table. https://condor.depaul.edu/gandrus/24...les-fields.htm

1.13: References

Introduction to the Module. (n.d.). Retrieved from https://www.cs.uct.ac.za/mit_notes/d...abase-approach

Costs and Risks of Database Approach. (2010). Retrieved from http://www.smartclass.co/2011/02/cos...-approach.html

Hooda, S. (2019, September 17). What are the Advantages of a Database Management System? Retrieved from https://www.goskills.com/Development...agement-system

Keuffel, W. “Battle of the Modeling Techniques.” DBMS Magazine (August 1996).

Kroenke, D., Auer, D., Vandenberg, S., Yoder, R., Database Processing, Fundamental, Design, and Implementation, Fifteenth, Edition, 2018.

Kroenke, D. “Waxing Semantic: An Interview.” DBMS Magazine (September 1994).

Kioko, O. N., & Bilal. (2011, March 28). Disadvantages of conventional file-processing system Retrieved from http://punarvasi.com/disadvantages-o...essing-system/

Multi-user environments: Spreadsheets vs Databases. (2016, April 11). Retrieved from https://www.workingdata.co.uk/spread...d-1-multi-user

Oracle Database Administrator (DBA) Salary Guide. (n.d.). Retrieved from https://education.oracle.com/oracle-...salaries-guide O.

(2019, December 20). What are the Components of Database System Environment? Retrieved from https://www.owlgen.in/what-are-the-c...m-environment/

Parody, L. (2019, September 27). Databases for Frontend Developers. Retrieved from https://medium.com/@lizparody/databa...s-2a4e9f16b7b4

Rahul, & Rahul. (2019, December 27). File Processing System - File System - Disadvantages of File Processing. Retrieved from https://www.tutorialcup.com/dbms/fil...ing-system.htm

Sir, B. (2017, July 26). Disadvantages of File Oriented System. Retrieved from http://niravbaldha.blogspot.com/p/fi...ed-system.html

Thiru. (n.d.). Thiru. Retrieved from http://www.myreadingroom.co.in/notes...dymaterial/65- dbms/506-database-develo pment-life-cycle.html

Vivek Gite. (2014, January 19). What is The Role Of the System Administrator? Retrieved from https://www.cyberciti.biz/faq/what-i...administrator/

WD, (2005) Retrieve from https://www.workingdata.co.uk/spread...-1-multi-user/)

Wang, X., Wang, Y., & Zhu, H. (2012). Energy-Efficient Multi-Job Scheduling Model for Cloud Computing and Its Genetic Algorithm. Mathematical Problems in Engineering, 2012 , 1–16. doi: 10.1155/2012/589243

What are the functions of the Database Administrator. (n.d.). Retrieved March 22, 2020, from http://www.pkirs.utep.edu/cis4365/Tu.../8.00700/1_mul tipart_xF8FF_2_tutorial.htm

What is a database? (n.d.). Retrieved from https://www.oracle.com/database/what-is-database.html

CHAPTER OVERVIEW

2: Data Modelling

2.1: What is a Database Model

High-level Conceptual Data Models

High-level conceptual data models provide concepts for presenting data in ways that are close to the way people perceive data. A typical example is the entity relationship model, which uses main concepts like entities, attributes and relationships. An entity represents a real-world object such as an employee or a project. The entity has attributes that represent properties such as an employee's name, address and birthdate. A relationship represents an association among entities; for example, an employee works on many projects. A relationship exists between the employee and each project.

Record-based Logical Data Models

Record-based logical data models provide concepts users can understand but are not too far from the way data is stored in the computer. Three well-known data models of this type are relational data models, network data models and hierarchical data models.

- The *relational model* represents data as *relations*, or tables. For example, in the membership system at Science World, each membership has many members. The membership identifier, expiry date and address information are fields in the membership. The members are individuals such as Mickey, Minnie, Mighty, Door, Tom, King, Man and Moose. Each record is said to be an *instance* of the membership table.
- The *network model* represents data as record types. This model also represents a limited type of one to many relationship called a *set type*, as shown in Figure 2.1.1.

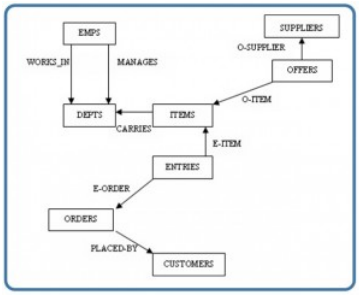


Figure 2.1.1: Network model diagram.

- The *hierarchical model* represents data as a hierarchical tree structure. Each branch of the hierarchy represents a number of related records. Figure 2.1.2 shows this schema in hierarchical model notation.

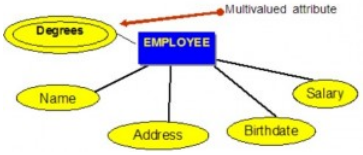


Figure 2.1.2: Hierarchical model diagram.

Key Terms

hierarchical model: represents data as a hierarchical tree structure

instance: a record within a table

network model: represents data as record types

relation: another term for table

relational model: represents data as relations or tables

set type: a limited type of one to many relationship

Exercises

1. What is a data model?
2. What is a high-level conceptual data model?
3. What is an entity? An attribute? A relationship?
4. List and briefly describe the common record-based logical data models.

2.2: Data Abstraction

Data modelling is the first step in the process of database design. This step is sometimes considered to be a high-level and abstract design phase, also referred to as conceptual design. The aim of this phase is to describe:

- The data contained in the database (e.g., entities: students, lecturers, courses, subjects)
- The relationships between data items (e.g., students are supervised by lecturers; lecturers teach courses)
- The constraints on data (e.g., student number has exactly eight digits; a subject has four or six units of credit only)

In the second step, the data items, the relationships and the constraints are all expressed using the concepts provided by the high-level data model. Because these concmepts do not include the implementation details, the result of the data modelling process is a (semi) formal representation of the database structure. This result is quite easy to understand so it is used as reference to make sure that all the user's requirements are met.

The third step is database design. During this step, we might have two sub-steps: one called *database logical design*, which defines a database in a data model of a specific DBMS, and another called *database physical design*, which defines the internal database storage structure, file organization or indexing techniques. These two sub-steps are database implementation and operations/user interfaces building steps.

In the database design phases, data are represented using a certain data model. The *data model* is a collection of concepts or notations for describing data, data relationships, data semantics and data constraints. Most data models also include a set of basic operations for manipulating data in the database.

Degrees of Data Abstraction

In this section we will look at the database design process in terms of specificity. Just as any design starts at a high level and proceeds to an ever-increasing level of detail, so does database design. For example, when building a home, you start with how many bedrooms and bathrooms the home will have, whether it will be on one level or multiple levels, etc. The next step is to get an architect to design the home from a more structured perspective. This level gets more detailed with respect to actual room sizes, how the home will be wired, where the plumbing fixtures will be placed, etc. The last step is to hire a contractor to build the home. That's looking at the design from a high level of abstraction to an increasing level of detail.

The database design is very much like that. It starts with users identifying the business rules; then the database designers and analysts create the database design; and then the database administrator implements the design using a DBMS.

The following subsections summarize the models in order of decreasing level of abstraction.

External models

- Represent the user's view of the database
- Contain multiple different external views
- Are closely related to the real world as perceived by each user

Conceptual models

- Provide flexible data-structuring capabilities
- Present a "community view": the logical structure of the entire database
- Contain data stored in the database
- Show relationships among data including:
  - Constraints
  - Semantic information (e.g., business rules)
  - Security and integrity information
- Consider a database as a collection of entities (objects) of various kinds
- Are the basis for identification and high-level description of main data objects; they avoid details
- Are database independent regardless of the database you will be using

Internal models

The three best-known models of this kind are the relational data model, the network data model and the hierarchical data model. These internal models:

- Consider a database as a collection of fixed-size records
- Are closer to the physical level or file structure
- Are a representation of the database as seen by the DBMS.
- Require the designer to match the conceptual model's characteristics and constraints to those of the selected implementation model
- Involve mapping the entities in the conceptual model to the tables in the relational model

Physical models

- Are the physical representation of the database
- Have the lowest level of abstractions
- Are how the data is stored; they deal with
  - Run-time performance
  - Storage utilization and compression
  - File organization and access methods
  - Data encryption
- Are the physical level – managed by the *operating system (OS)*
- Provide concepts that describe the details of how data are stored in the computer's memory

---

2.3: Data Abstraction Layer

In a pictorial view, you can see how the different models work together. Let's look at this from the highest level, the external model.

The external model is the end user's view of the data. Typically a database is an enterprise system that serves the needs of multiple departments. However, one department is not interested in seeing other departments' data (e.g., the human resources (HR) department does not care to view the sales department's data). Therefore, one user view will differ from another.

The external model requires that the designer subdivide a set of requirements and constraints into functional modules that can be examined within the framework of their external models (e.g., human resources versus sales).

As a data designer, you need to understand all the data so that you can build an enterprise-wide database. Based on the needs of various departments, the conceptual model is the first model created.

At this stage, the conceptual model is independent of both software and hardware. It does not depend on the DBMS software used to implement the model. It does not depend on the hardware used in the implementation of the model. Changes in either hardware or DBMS software have no effect on the database design at the conceptual level.

Once a DBMS is selected, you can then implement it. This is the internal model. Here you create all the tables, constraints, keys, rules, etc. This is often referred to as the *logical design*.

The physical model is simply the way the data is stored on disk. Each database vendor has its own way of storing the data. (please excuse the blurry image - the original was to small, so this is scaled out a bit, but introduced the fuzzyness)

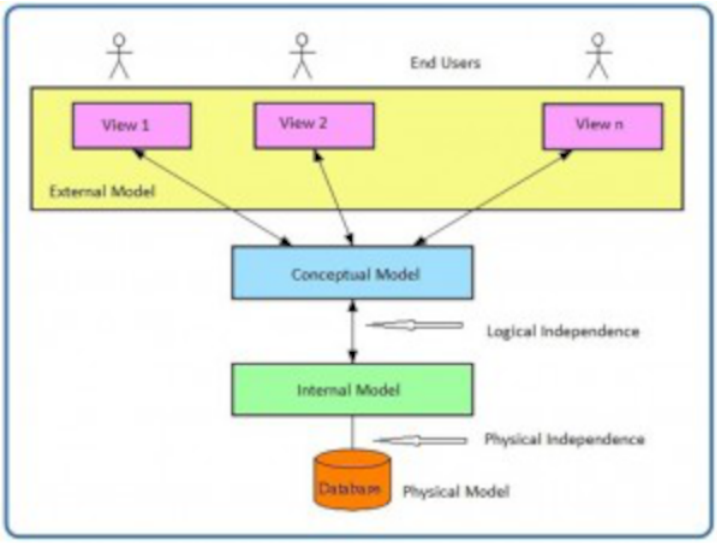


Figure 2.3.1: *Data abstraction layers.* (CC-BY-SA; Adrienne Watt via Website)

2.4: Schemas

Schemas

A *schema* is an overall description of a database, and it is usually represented by the *entity relationship diagram (ERD)*. There are many subschemas that represent external models and thus display external views of the data. Below is a list of items to consider during the design process of a database.

- External schemas: there are multiple
- Multiple subschemas: these display multiple external views of the data
- Conceptual schema: there is only one. This schema includes data items, relationships and constraints, all represented in an ERD.
- Physical schema: there is only one

Logical and Physical Data Independence

Data independence refers to the immunity of user applications to changes made in the definition and organization of data. Data abstractions expose only those items that are important or pertinent to the user. Complexity is hidden from the database user.

Data independence and operation independence together form the feature of data abstraction. There are two types of data independence: logical and physical.

Logical data independence

A *logical schema* is a conceptual design of the database done on paper or a whiteboard, much like architectural drawings for a house. The ability to change the logical schema, without changing the *external schema* or user view, is called *logical data independence*. For example, the addition or removal of new entities, attributes or relationships to this *conceptual schema* should be possible without having to change existing external schemas or rewrite existing application programs.

In other words, changes to the logical schema (e.g., alterations to the structure of the database like adding a column or other tables) should not affect the function of the application (external views).

Physical data independence

Physical data independence refers to the immunity of the internal model to changes in the physical model. The logical schema stays unchanged even though changes are made to file organization or storage structures, storage devices or indexing strategy.

Physical data independence deals with hiding the details of the storage structure from user applications. The applications should not be involved with these issues, since there is no difference in the operation carried out against the data.

2.5: Database Classification Types

Database management systems can be classified based on several criteria, such as the data model, user numbers and database distribution, all described below.

Classification Based on Data Model

The most popular data model in use today is the relational data model. Well-known DBMSs like Oracle, MS SQL Server, DB2 and MySQL support this model. Other traditional models, such as hierarchical data models and network data models, are still used in industry mainly on mainframe platforms. However, they are not commonly used due to their complexity. These are all referred to as *traditional models* because they preceded the relational model.

In recent years, the newer *object-oriented data models* were introduced. This model is a database management system in which information is represented in the form of objects as used in object-oriented programming. Object-oriented databases are different from relational databases, which are table-oriented. Object-oriented database management systems (OODBMS) combine database capabilities with object-oriented programming language capabilities.

The object-oriented models have not caught on as expected so are not in widespread use. Some examples of object-oriented DBMSs are O2, ObjectStore and Jasmine.

Classification Based on User Numbers

A DBMS can be classification based on the number of users it supports. It can be a *single-user database system*, which supports one user at a time, or a *multiuser database system*, which supports multiple users concurrently.

Classification Based on Database Distribution

There are four main distribution systems for database systems and these, in turn, can be used to classify the DBMS.

Centralized systems

With a *centralized database system*, the DBMS and database are stored at a single site that is used by several other systems too. This is illustrated in Figure 2.5.1.

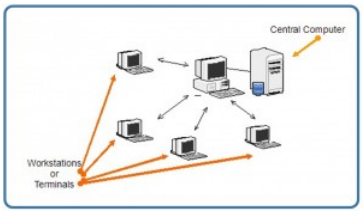


Figure 2.5.1: *Example of a centralized database system.*

In the early 1980s, many Canadian libraries used the GEAC 8000 to convert their manual card catalogues to machine-readable centralized catalogue systems. Each book catalogue had a barcode field similar to those on supermarket products.

Distributed database system

In a *distributed database system*, the actual database and the DBMS software are distributed from various sites that are connected by a computer network, as shown in Figure 2.5.2.

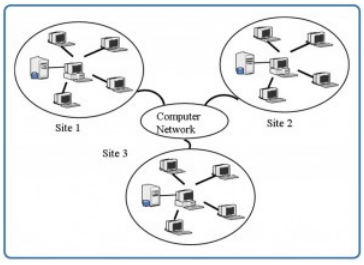


Figure 2.5.2: *Example of a distributed database system.*

Homogeneous distributed database systems

Homogeneous distributed database systems use the same DBMS software from multiple sites. Data exchange between these various sites can be handled easily. For example, library information systems by the same vendor, such as Geac Computer Corporation, use the same DBMS software which allows easy data exchange between the various Geac library sites.

Heterogeneous distributed database systems

In a *heterogeneous distributed database system*, different sites might use different DBMS software, but there is additional common software to support data exchange between these sites. For example, the various library database systems use the same machine-readable cataloguing (MARC) format to support library record data exchange.

CHAPTER OVERVIEW

3: The Relational Data Model

3.1: Relational Model Explained

The relational data model was introduced by E. F. Codd in 1970. Currently, it is the most widely used data model.

The relational model has provided the basis for:

- Research on the theory of data/relationship/constraint
- Numerous database design methodologies
- The standard database access language called s*tructured query language (SQL)*
- Almost all modern commercial database management systems

The relational data model describes the world as “a collection of inter-related relations (or tables).”

Fundamental Concepts in the Relational Data Model

Relation

A *relation*, also known as a *table* or *file*, is a subset of the Cartesian product of a list of domains characterized by a name. And within a table, each row represents a group of related data values. A *row*, or record, is also known as a *tuple*. The columns in a table is a field and is also referred to as an attribute. You can also think of it this way: an attribute is used to define the record and a record contains a set of attributes.

The steps below outline the logic between a relation and its domains.

1. Given *n* domains are denoted by D1, D2, … Dn
2. And *r* is a relation defined on these domains
3. Then $r \subseteq D1 \times D2 \times \ldots \times Dn$

Table

A database is composed of multiple tables and each table holds the data. Figure 7.1.1 shows a database that contains three tables.

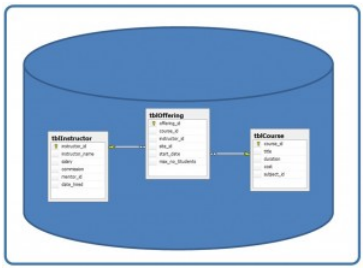


Figure 3.1.1: *Database with three tables.*

Column

A database stores pieces of information or facts in an organized way. Understanding how to use and get the most out of databases requires us to understand that method of organization.

The principal storage units are called *columns* or *fields* or *attributes*. These house the basic components of data into which your content can be broken down. When deciding which fields to create, you need to think generically about your information, for example, drawing out the common components of the information that you will store in the database and avoiding the specifics that distinguish one item from another.

Look at the example of an ID card in Figure 7.1.2 to see the relationship between fields and their data.

Field Name	Data
First Name	Isabelle
Family Name	Whelan
Nationality	British
Salary	109,900
Date of Birth	15 September 1983
Marital Status	Single
Shift	Mon, Wed
Place of issue	Addis Ababa
Valid until	17 December 2003

Figure 3.1.2: *Example of an ID card by A. Watt.*

Domain

A *domain* is the original sets of atomic values used to model data. By *atomic value*, we mean that each value in the domain is indivisible as far as the relational model is concerned. For example:

- The domain of Marital Status has a set of possibilities: Married, Single, Divorced.
- The domain of Shift has the set of all possible days: {Mon, Tue, Wed…}.
- The domain of Salary is the set of all floating-point numbers greater than 0 and less than 200,000.
- The domain of First Name is the set of character strings that represents names of people.

In summary, a domain is a set of acceptable values that a column is allowed to contain. This is based on various properties and the data type for the column. We will discuss data types in another chapter.

Records

Just as the content of any one document or item needs to be broken down into its constituent bits of data for storage in the fields, the link between them also needs to be available so that they can be reconstituted into their whole form. Records allow us to do this. *Records* contain fields that are related, such as a customer or an employee. As noted earlier, a tuple is another term used for record.

Records and fields form the basis of all databases. A simple table gives us the clearest picture of how records and fields work together in a database storage project.

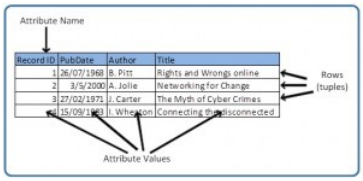


Figure 3.1.3: *Example of a simple table by A. Watt.*

The simple table example in Figure 7.1.3 shows us how fields can hold a range of different sorts of data. This one has:

- A Record ID field: this is an ordinal number; its data type is an integer.
- A PubDate field: this is displayed as day/month/year; its data type is date.
- An Author field: this is displayed as Initial. Surname; its data type is text.
- A Title field text: free text can be entered here.

You can command the database to sift through its data and organize it in a particular way. For example, you can request that a selection of records be limited by date: 1. all before a given date, 2. all after a given date or 3. all between two given dates. Similarly, you can choose to have records sorted by date. Because the field, or record, containing the data is set up as a Date field, the database reads the information in the Date field not just as numbers separated by slashes, but rather, as dates that must be ordered according to a calendar system.

Degree

The *degree* is the number of attributes in a table. In our example in Figure 7.1.3, the degree is 4.

Properties of a Table

- A table has a name that is distinct from all other tables in the database.
- There are no duplicate rows; each row is distinct.
- Entries in columns are atomic. The table does not contain repeating groups or multivalued attributes.
- Entries from columns are from the same domain based on their data type including:
  - number (numeric, integer, float, smallint,…)
  - character (string)
  - date
  - logical (true or false)
- Operations combining different data types are disallowed.
- Each attribute has a distinct name.
- The sequence of columns is insignificant.

- The sequence of rows is insignificant.

Key Terms

atomic value: each value in the domain is indivisible as far as the relational model is concerned**attribute**: principle storage unit in a database

column: see *attribute*

degree: number of attributes in a table

domain: the original sets of atomic values used to model data; a set of acceptable values that a column is allowed to contain

field: see *attribute*

file:see *relation*

record: contains fields that are related; *see tuple*

relation: a subset of the Cartesian product of a list of domains characterized by a name; the technical term for table or file

row: *see tuple*

structured query language (SQL): the standard database access language

table:see *relation*

tuple: a technical term for row or record

Terminology Key

Several of the terms used in this chapter are synonymous. In addition to the Key Terms above, please refer to Table 3.1.1 below. The terms in the Alternative 1 column are most commonly used.

Formal Terms (Codd)	Alternative 1	Alternative 2
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

Table 3.1.1: . *Terms and their synonyms by A. Watt.*

Exercises

Use Table 3.1.2 *to answer questions 1-4.*

1. Using correct terminology, identify and describe all the components in Table 3.1.2.
2. What is the possible domain for field EmpJobCode?
3. How many records are shown?
4. How many attributes are shown?
5. List the properties of a table.

EMPLOYEE

EMPID	EMPLNAME	EMPINIT	EMPFNAME	EMPJOBCODE
123455	Friedman	A.	Robert	12
123456	Olanski	D.	Delbert	18
123457	Fontein	G.	Juliette	15
123458	Cruazona	X.	Maria	18

Table 3.1.2: . *Table for exercise questions, by A. Watt.*

3.2: Entities

The *entity relationship (ER) data model* has existed for over 35 years. It is well suited to data modelling for use with databases because it is fairly abstract and is easy to discuss and explain. ER models are readily translated to relations. ER models, also called an ER schema, are represented by ER diagrams.

ER modelling is based on two concepts:

- Entities, defined as tables that hold specific information (data)
- *Relationships*, defined as the associations or interactions between entities

Here is an example of how these two concepts might be combined in an ER data model: Prof. Ba (entity) teaches (relationship) the Database Systems course (entity).

For the rest of this chapter, we will use a sample database called the COMPANY database to illustrate the concepts of the ER model. This database contains information about employees, departments and projects. Important points to note include:

- There are several departments in the company. Each department has a unique identification, a name, location of the office and a particular employee who manages the department.
- A department controls a number of projects, each of which has a unique name, a unique number and a budget.
- Each employee has a name, identification number, address, salary and birthdate. An employee is assigned to one department but can join in several projects. We need to record the start date of the employee in each project. We also need to know the direct supervisor of each employee.
- We want to keep track of the dependents for each employee. Each dependent has a name, birthdate and relationship with the employee.

Entity, Entity Set and Entity Type

An *entity* is an object in the real world with an independent existence that can be differentiated from other objects. An entity might be

- An object with physical existence (e.g., a lecturer, a student, a car)
- An object with conceptual existence (e.g., a course, a job, a position)

Entities can be classified based on their strength. An entity is considered weak if its tables are existence dependent.

- That is, it cannot exist without a relationship with another entity
- Its primary key is derived from the primary key of the parent entity
    - The Spouse table, in the COMPANY database, is a weak entity because its primary key is dependent on the Employee table. Without a corresponding employee record, the spouse record would not exist.

An entity is considered strong if it can exist apart from all of its related entities.

- Kernels are strong entities.
- A table without a foreign key or a table that contains a foreign key that can contain nulls is a strong entity

Another term to know is *entity type* which defines a collection of similar entities.

An *entity set* is a collection of entities of an entity type at a particular point of time. In an entity relationship diagram (ERD), an entity type is represented by a name in a box. For example, in Figure 3.2.1, the entity type is EMPLOYEE.

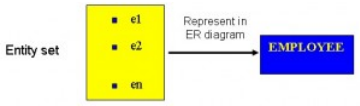


Figure 3.2.1: *ERD with entity type EMPLOYEE.*

Existence dependency

An entity's existence is dependent on the existence of the related entity. It is existence-dependent if it has a mandatory foreign key (i.e., a foreign key attribute that cannot be null). For example, in the COMPANY database, a Spouse entity is existence - dependent on the Employee entity.

Kinds of Entities

You should also be familiar with different kinds of entities including independent entities, dependent entities and characteristic entities. These are described below.

Independent entities

Independent entities, also referred to as kernels, are the backbone of the database. They are what other tables are based on. *Kernels* have the following characteristics:

- They are the building blocks of a database.
- The primary key may be simple or composite.
- The primary key is not a foreign key.
- They do not depend on another entity for their existence.

If we refer back to our COMPANY database, examples of an independent entity include the Customer table, Employee table or Product table.

Dependent entities

Dependent entities, also referred to as *derived entities*, depend on other tables for their meaning. These entities have the following characteristics:

- Dependent entities are used to connect two kernels together.
- They are said to be existence dependent on two or more tables.
- Many to many relationships become associative tables with at least two foreign keys.
- They may contain other attributes.
- The foreign key identifies each associated table.
- There are three options for the primary key:
  1. Use a composite of foreign keys of associated tables if unique
  2. Use a composite of foreign keys and a qualifying column
  3. Create a new simple primary key

Characteristic entities

Characteristic entities provide more information about another table. These entities have the following characteristics:

- They represent multivalued attributes.
- They describe other entities.
- They typically have a one to many relationship.
- The foreign key is used to further identify the characterized table.
- Options for primary key are as follows:
  1. Use a composite of foreign key plus a qualifying column
  2. Create a new simple primary key. In the COMPANY database, these might include:
     - Employee (EID, Name, Address, Age, Salary) – EID is the simple primary key.
     - EmployeePhone (EID, Phone) – EID is part of a composite primary key. Here, EID is also a foreign key.

3.3: Attributes

Each entity is described by a set of attributes (e.g., Employee = (Name, Address, Birthdate (Age), Salary).

Each attribute has a name, and is associated with an entity and a domain of legal values. However, the information about attribute domain is not presented on the ERD.

In the entity relationship diagram, shown in Figure 8.2.1, each attribute is represented by an oval with a name inside.

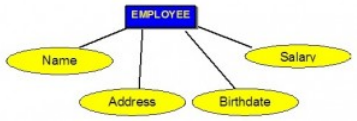


Figure 3.3.1: *How attributes are represented in an ERD.*

Types of Attributes

There are a few types of attributes you need to be familiar with. Some of these are to be left as is, but some need to be adjusted to facilitate representation in the relational model. This first section will discuss the types of attributes. Later on we will discuss fixing the attributes to fit correctly into the relational model.

Simple attributes

Simple attributes are those drawn from the atomic value domains; they are also called *single-valued attributes*. In the COMPANY database, an example of this would be: Name = {John} ; Age = {23}

Composite attributes

Composite attributes are those that consist of a hierarchy of attributes. Using our database example, and shown in Figure 8.2.2, Address may consist of Number, Street and Suburb. So this would be written as → Address = {59 + 'Meek Street' + 'Kingsford'}

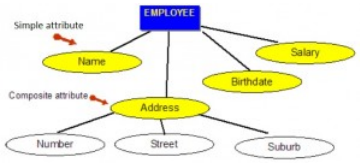


Figure 3.3.2: *An example of composite attributes.*

Multivalued attributes

Multivalued attributes are attributes that have a set of values for each entity. An example of a multivalued attribute from the COMPANY database, as seen in Figure 8.2.3, are the degrees of an employee: BSc, MIT, PhD.

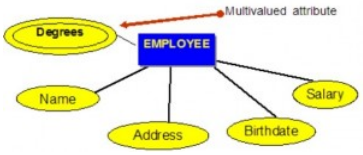


Figure 3.3.3: *Example of a multivalued attribute.*

Derived attributes

Derived attributes are attributes that contain values calculated from other attributes. An example of this can be seen in Figure 8.2.4. Age can be derived from the attribute Birthdate. In this situation, Birthdate is called a *stored attribute*, which is physically saved to the database.

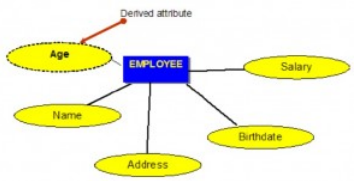


Figure 3.3.4: *Example of a derived attribute.*

3.4: Keys

An important constraint on an entity is the key. The *key* is an attribute or a group of attributes whose values can be used to uniquely identify an individual entity in an entity set.

Types of Keys

There are several types of keys. These are described below.

Candidate key

A *candidate key* is a simple or composite key that is unique and minimal. It is unique because no two rows in a table may have the same value at any time. It is minimal because every column is necessary in order to attain uniqueness.

From our COMPANY database example, if the entity is **Employee**(EID, First Name, Last Name, *SIN*, Address, Phone, BirthDate, Salary, DepartmentID), possible candidate keys are:

- EID, SIN
- First Name and Last Name – assuming there is no one else in the company with the same name
- Last Name and DepartmentID – assuming two people with the same last name don't work in the same department

Composite key

A *composite key* is composed of two or more attributes, but it must be minimal.

Using the example from the candidate key section, possible composite keys are:

- First Name and Last Name – assuming there is no one else in the company with the same name
- Last Name and Department ID – assuming two people with the same last name don't work in the same department

Primary key

The primary key is a candidate key that is selected by the database designer to be used as an identifying mechanism for the whole entity set. It must uniquely identify tuples in a table and not be null. The primary key is indicated in the ER model by underlining the attribute.

- A candidate key is selected by the designer to uniquely identify tuples in a table. It must not be null.
- A key is chosen by the database designer to be used as an identifying mechanism for the whole entity set. This is referred to as the primary key. This key is indicated by underlining the attribute in the ER model.

In the following example, EID is the primary key:

Employee(<u>EID</u>, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID)

Secondary key

A *secondary key* is an attribute used strictly for retrieval purposes (can be composite), for example: Phone and Last Name.

Alternate key

Alternate keys are all candidate keys not chosen as the primary key.

Foreign key

A *foreign key (FK)* is an attribute in a table that references the primary key in another table OR it can be null. Both foreign and primary keys must be of the same data type.

In the COMPANY database example below, DepartmentID is the foreign key:

Employee(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID)

3.5: Nulls

A *null* is a special symbol, independent of data type, which means either unknown or inapplicable. It does not mean zero or blank. Features of null include:

- No data entry
- Not permitted in the primary key
- Should be avoided in other attributes
- Can represent
  - An unknown attribute value
  - A known, but missing, attribute value
  - A “not applicable” condition
- Can create problems when functions such as COUNT, AVERAGE and SUM are used
- Can create logical problems when relational tables are linked

NOTE: The result of a comparison operation is null when either argument is null. The result of an arithmetic operation is null when either argument is null (except functions that ignore nulls).

Example of how null can be used

Use the Salary table (Salary_tbl) in Figure 3.5.1 to follow an example of how null can be used.

Salary_tbl

emp#	jobName	salary	commission
E10	Sales	12500	32090
E11	Null	25000	8000
E12	Sales	44000	0
E13	Sales	44000	Null

Figure 3.5.1: *Salary table for null example, by A. Watt.*

To begin, find all employees (emp#) in Sales (under the jobName column) whose salary plus commission are greater than 30,000.

```
SELECT emp# FROM Salary_tbl
WHERE  jobName = Sales AND
(commission + salary) > 30,000  –> E10 and E12
```

This result does not include E13 because of the null value in the commission column. To ensure that the row with the null value is included, we need to look at the individual fields. By adding commission and salary for employee E13, the result will be a null value. The solution is shown below.

```
SELECT emp# FROM Salary_tbl
WHERE jobName = Sales AND
commission > 30000 OR
salary > 30000 OR
commission + salary) > 30,000  –>E10 and E12 and E13
```

3.6: Relationships

Relationships are the glue that holds the tables together. They are used to connect related information between tables.

Relationship strength is based on how the primary key of a related entity is defined. A weak, or non-identifying, relationship exists if the primary key of the related entity does not contain a primary key component of the parent entity. Company database examples include:

- Customer(**CustID**, CustName)
- Order(**OrderID**, CustID, Date)

A strong, or identifying, relationship exists when the primary key of the related entity contains the primary key component of the parent entity. Examples include:

- Course(**CrsCode**, DeptCode, Description)
- Class(**CrsCode, Section**, ClassTime…)

Types of Relationships

Below are descriptions of the various types of relationships.

One to many (1:M) relationship

A one to many (1:M) relationship should be the norm in any relational database design and is found in all relational database environments. For example, one department has many employees. Figure 3.6.1 shows the relationship of one of these employees to the department.

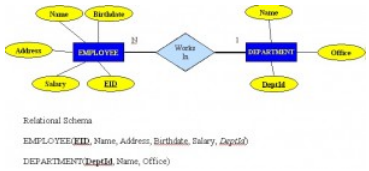


Figure 3.6.1: *Example of a one to many relationship.*

One to one (1:1) relationship

A one to one (1:1) relationship is the relationship of one entity to only one other entity, and vice versa. It should be rare in any relational database design. In fact, it could indicate that two entities actually belong in the same table.

An example from the COMPANY database is one employee is associated with one spouse, and one spouse is associated with one employee.

Many to many (M:N) relationships

For a many to many relationship, consider the following points:

- It cannot be implemented as such in the relational model.
- It can be changed into two 1:M relationships.
- It can be implemented by breaking up to produce a set of 1:M relationships.
- It involves the implementation of a composite entity.
- Creates two or more 1:M relationships.
- The composite entity table must contain at least the primary keys of the original tables.
- The linking table contains multiple occurrences of the foreign key values.
- Additional attributes may be assigned as needed.
- It can avoid problems inherent in an M:N relationship by creating a composite entity or bridge entity. For example, an employee can work on many projects OR a project can have many employees working on it, depending on the business rules. Or, a student can have many classes and a class can hold many students.

Figure 3.6.2 shows another another aspect of the M:N relationship where an employee has different start dates for different projects. Therefore, we need a JOIN table that contains the EID, Code and StartDate.

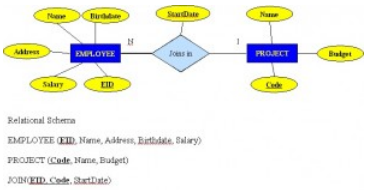


Figure 3.6.2: *Example where employee has different start dates for different projects.*

Example of mapping an M:N binary relationship type

- For each M:N binary relationship, identify two relations.
- A and B represent two entity types participating in R.
- Create a new relation S to represent R.
- S needs to contain the PKs of A and B. These together can be the PK in the S table OR these together with another simple attribute in the new table R can be the PK.
- The combination of the primary keys (A and B) will make the primary key of S.

Unary relationship (recursive)

A *unary relationship*, also called *recursive*, is one in which a relationship exists between occurrences of the same entity set. In this relationship, the primary and foreign keys are the same, but they represent two entities with different roles. See Figure 3.6.3an example.

For some entities in a unary relationship, a separate column can be created that refers to the primary key of the same entity set.

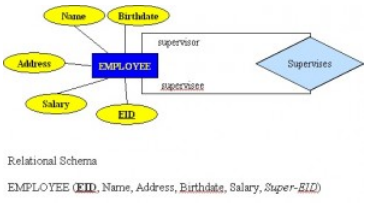


Figure 3.6.3: *Example of a unary relationship.*

Ternary Relationships

A *ternary relationship* is a relationship type that involves many to many relationships between three tables.

Refer to Figure 3.6.4 for an example of mapping a ternary relationship type. Note *n-ary* means multiple tables in a relationship. (Remember, N = many.)

- For each n-ary (> 2) relationship, create a new relation to represent the relationship.
- The primary key of the new relation is a combination of the primary keys of the participating entities that hold the N (many) side.
- In most cases of an n-ary relationship, all the participating entities hold a **many** side.

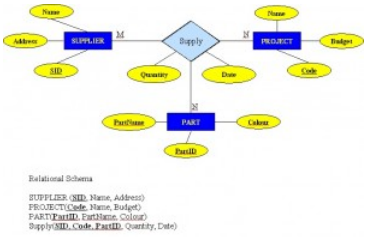


Figure 3.6.4: *Example of a ternary relationship.*

Key Terms

alternate key: all candidate keys not chosen as the primary key

candidate key: a simple or composite key that is unique (no two rows in a table may have the same value) and minimal (every column is necessary)

characteristic entities: entities that provide more information about another table

composite attributes: attributes that consist of a hierarchy of attributes

composite key: composed of two or more attributes, but it must be minimal

dependent entities: these entities depend on other tables for their meaning

derived attributes: attributes that contain values calculated from other attributes

derived entities: see *dependent entities*

EID: employee identification (ID)

entity: a thing or object in the real world with an independent existence that can be differentiated from other objects

entity relationship (ER) data model: also called an ER schema, are represented by ER diagrams. These are well suited to data modelling for use with databases.

entity relationship schema: see *entity relationship data model*

entity set:a collection of entities of an entity type at a point of time

entity type: a collection of similar entities

foreign key (FK): an attribute in a table that references the primary key in another table OR it can be null

independent entity: as the building blocks of a database, these entities are what other tables are based on

kernel: see *independent entity*

key: an attribute or group of attributes whose values can be used to uniquely identify an individual entity in an entity set

multivalued attributes: attributes that have a set of values for each entity

n-ary: multiple tables in a relationship

null: a special symbol, independent of data type, which means either unknown or inapplicable; it does not mean zero or blank

recursive relationship: see *unary relationship*

relationships: the associations or interactions between entities; used to connect related information between tables

relationship strength: based on how the primary key of a related entity is defined

secondary key an attribute used strictly for retrieval purposes

simple attributes: drawn from the atomic value domains

SIN: social insurance number

single-valued attributes: see *simple attributes*

stored attribute: saved physically to the database

ternary relationship: a relationship type that involves many to many relationships between three tables.

unary relationship: one in which a relationship exists between occurrences of the same entity set.

Exercises

1. What two concepts are ER modelling based on?

2. The database in Figure 3.6.5 is composed of two tables. Use this figure to answer questions 2.1 to 2.5.

DIRECTOR

DIRNUM	DIRNAME	DIRDOB
100	J.Broadway	01/08/39
101	J.Namath	11/12/48
102	W.Blake	06/15/44

PLAY

PLAYNO	PLAYNAME	DIRNUM
1001	Cat on a cold bare roof	102
1002	Hold the mayo, pass the bread	101
1003	I never promised you coffee	102
1004	Silly putty goes to Texas	100
1005	See no sound, hear no sight	101
1006	Starstruck in Biloxi	102
1007	Stranger in parrot ice	101

Figure 3.6.5: *Director and Play tables for question 2, by A. Watt.*

1. Identify the primary key for each table.
2. Identify the foreign key in the PLAY table.
3. Identify the candidate keys in both tables.
4. Draw the ER model.
5. Does the PLAY table exhibit referential integrity? Why or why not?

3. Define the following terms (you may need to use the Internet for some of these):
   schema
   host language
   data sublanguage
   data definition language
   unary relation
   foreign key
   virtual relation
   connectivity
   composite key
   linking table
4. The RRE Trucking Company database includes the three tables as shown in Figure 3.6.6. Use Figure 8.5.6 to answer questions 4.1 to 4.5.

TRUCK

TNUM	BASENUM	TYPENUM	TMILES	TBOUGHT	TSERIAL
1001	501	1	5900.2	11/08/90	aa-125
1002	502	2	64523.9	11/08/90	ac-213
1003	501	2	32116.0	09/29/91	ac-215
1004		2	3256.9	01/14/92	ac-315

BASE

BASENUM	BASECITY	BASESTATE	BASEPHON	BASEMGR
501	Dallas	TX	893-9870	J. Jones
502	New York	NY	234-7689	K. Lee

TYPE

TYPENUM	TYPEDESC
1	single box, double axle
2	tandem trailer, single axle

Figure 3.6.6: *Truck, Base and Type tables for question 4, by A. Watt.*

1. Identify the primary and foreign key(s) for each table.
2. Does the TRUCK table exhibit entity and referential integrity? Why or why not? Explain your answer.
3. What kind of relationship exists between the TRUCK and BASE tables?
4. How many entities does the TRUCK table contain ?
5. Identify the TRUCK table candidate key(s).

Customer

CustID	CustName	AccntNo.
100	Joe Smith	010839
101	Andy Blake	111248
102	Sue Brown	061544

BookOrders

OrderID	Title	CustID	Price
1001	The Dark Tower	**102**	**12.00**
1002	Incubus Dreams	**101**	**19.99**
1003	Song of Susannah	**102**	**23.00**
1004	The Time Traveler's Wife	**100**	**21.00**
1005	The Dark Tower	**101**	**12.00**
1006	Tanequil	**102**	**15.00**
1007	Song of Susannah	**101**	**23.00**

Figure 3.6.7: *Customer and BookOrders tables for question 5, by A. Watt.*

5. Suppose you are using the database in Figure 3.6.7, composed of the two tables. Use Figure 3.6.7 to answer questions 5.1 to 5.6.
    1. Identify the primary key in each table.
    2. Identify the foreign key in the BookOrders table.
    3. Are there any candidate keys in either table?
    4. Draw the ER model.
    5. Does the BookOrders table exhibit referential integrity? Why or why not?
    6. Do the tables contain redundant data? If so which table(s) and what is the redundant data?
6. Looking at the student table in Figure 3.6.8, list all the possible candidate keys. Why did you select these?

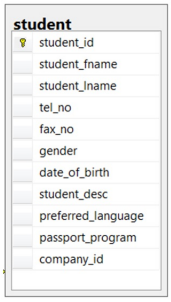


Figure 3.6.8: *Student table for question 6, by A. Watt.*

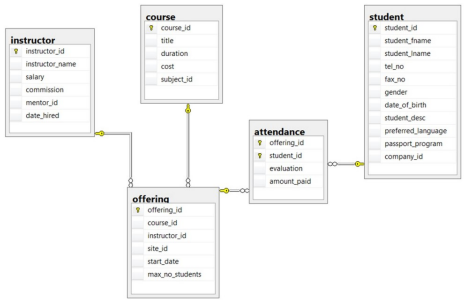


Figure 3.6.9: *ERD of school database for questions 7-10, by A. Watt.*

Use the ERD of a school database in Figure 3.6.9 to answer questions 7 to 10.

7. Identity all the kernels and dependent and characteristic entities in the ERD.
8. Which of the tables contribute to weak relationships? Strong relationships?
9. Looking at each of the tables in the school database in Figure 3.6.9, which attribute could have a NULL value? Why?
10. Which of the tables were created as a result of many to many relationships?

Also see *Appendix B: Sample ERD Exercises*

CHAPTER OVERVIEW

4: Integrity Rules, Constraints and Functional Dependencies

4.1: Constraints and Intergrity

Constraints are a very important feature in a relational model. In fact, the relational model supports the well-defined theory of constraints on attributes or tables. Constraints are useful because they allow a designer to specify the semantics of data in the database. *Constraints* are the rules that force DBMSs to check that data satisfies the semantics.

Domain Integrity

Domain restricts the values of attributes in the relation and is a constraint of the relational model. However, there are real-world semantics for data that cannot be specified if used only with domain constraints. We need more specific ways to state what data values are or are not allowed and which format is suitable for an attribute. For example, the Employee ID (EID) must be unique or the employee Birthdate is in the range [Jan 1, 1950, Jan 1, 2000]. Such information is provided in logical statements called *integrity constraints*.

There are several kinds of integrity constraints, described below.

Entity integrity

To ensure *entity integrity*, it is required that every table have a primary key. Neither the PK nor any part of it can contain null values. This is because null values for the primary key mean we cannot identify some rows. For example, in the EMPLOYEE table, Phone cannot be a primary key since some people may not have a telephone.

Referential integrity

Referential integrity requires that a foreign key must have a matching primary key or it must be null. This constraint is specified between two tables (parent and child); it maintains the correspondence between rows in these tables. It means the reference from a row in one table to another table must be valid.

Examples of referential integrity constraint in the Customer/Order database of the Company:

- Customer(**CustID**, CustName)
- Order(**OrderID**, CustID, OrderDate)

To ensure that there are no orphan records, we need to enforce referential integrity. An *orphan record* is one whose foreign key FK value is not found in the corresponding entity – the entity where the PK is located. Recall that a typical join is between a PK and FK.

The referential integrity constraint states that the customer ID (CustID) in the Order table must match a valid CustID in the Customer table. Most relational databases have declarative referential integrity. In other words, when the tables are created the referential integrity constraints are set up.

Here is another example from a Course/Class database:

- Course(**CrsCode**, DeptCode, Description)
- Class(**CrsCode, Section**, ClassTime)

The referential integrity constraint states that CrsCode in the Class table must match a valid CrsCode in the Course table. In this situation, it's not enough that the CrsCode and Section in the Class table make up the PK, we must also enforce referential integrity.

When setting up referential integrity it is important that the PK and FK have the same data types and come from the same domain, otherwise the relational database management system (RDBMS) will not allow the join. RDBMS is a popular database system that is based on the relational model introduced by E. F. Codd of IBM's San Jose Research Laboratory. Relational database systems are easier to use and understand than other database systems.

Referential integrity in Microsoft Access

In Microsoft (MS) Access, referential integrity is set up by joining the PK in the Customer table to the CustID in the Order table. See Figure 4.1.1 for a view of how this is done on the Edit Relationships screen in MS Access.

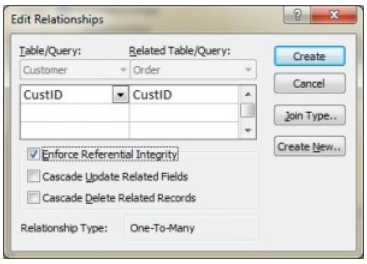


Figure 4.1.1. *Referential access in MS Access, by A. Watt.*

Referential integrity using Transact-SQL (MS SQL Server)

When using Transact-SQL, the referential integrity is set when creating the Order table with the FK. Listed below are the statements showing the FK in the Order table referencing the PK in the Customer table.

```
CREATE TABLE Customer
( CustID INTEGER PRIMARY KEY,
CustName CHAR(35) )
```

```
CREATE TABLE Orders
( OrderID INTEGER PRIMARY KEY,
CustID INTEGER REFERENCES Customer(CustID),
OrderDate DATETIME )
```

Foreign key rules

Additional foreign key rules may be added when setting referential integrity, such as what to do with the child rows (in the Orders table) when the record with the PK, part of the parent (Customer), is deleted or changed (updated). For example, the Edit Relationships window in MS Access (see Figure 9.1) shows two additional options for FK rules: Cascade Update and Cascade Delete. If these are not selected, the system will prevent the deletion or update of PK values in the parent table (Customer table) if a child record exists. The child record is any record with a matching PK.

In some databases, an additional option exists when selecting the Delete option called Set to Null. In this is chosen, the PK row is deleted, but the FK in the child table is set to NULL. Though this creates an orphan row, it is acceptable.

Enterprise Constraints

Enterprise constraints – sometimes referred to as semantic constraints – are additional rules specified by users or database administrators and can be based on multiple tables.

Here are some examples.

- A class can have a maximum of 30 students.
- A teacher can teach a maximum of four classes per semester.
- An employee cannot take part in more than five projects.
- The salary of an employee cannot exceed the salary of the employee's manager.

4.2: Business Rules

Business rules are obtained from users when gathering requirements. The requirements-gathering process is very important, and its results should be verified by the user before the database design is built. If the business rules are incorrect, the design will be incorrect, and ultimately the application built will not function as expected by the users.

Some examples of business rules are:

- A teacher can teach many students.
- A class can have a maximum of 35 students.
- A course can be taught many times, but by only one instructor.
- Not all teachers teach classes.

Cardinality and connectivity

Business rules are used to determine cardinality and connectivity. *Cardinality* describes the relationship between two data tables by expressing the minimum and maximum number of entity occurrences associated with one occurrence of a related entity. In Figure 4.2.2, you can see that cardinality is represented by the innermost markings on the relationship symbol. In this figure, the cardinality is 0 (zero) on the right and 1 (one) on the left.

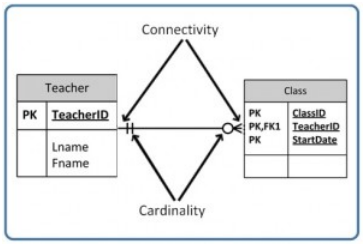


Figure 4.2.1: *Position of connectivity and cardinality on a relationship symbol, by A. Watt.*

The outermost symbol of the relationship symbol, on the other hand, represents the connectivity between the two tables. *Connectivity* is the relationship between two tables, e.g., one to one or one to many. The only time it is zero is when the FK can be null. When it comes to participation, there are three options to the relationship between these entities: either 0 (zero), 1 (one) or many. In Figure 4.2.2, for example, the connectivity is 1 (one) on the outer, left-hand side of this line and many on the outer, right-hand side.

Figure 4.2.2. shows the symbol that represents a one to many relationship.

Figure 4.2.2:

In Figure 4.2.4, both inner (representing cardinality) and outer (representing connectivity) markers are shown. The left side of this symbol is read as minimum 1 and maximum 1. On the right side, it is read as: minimum 1 and maximum many.

Figure 4.2.3:

4.3: Relationship Types

The line that connects two tables, in an ERD, indicates the *relationship type* between the tables*:* either identifying or non-identifying. An *identifying relationship* will have a solid line (where the PK contains the FK). A *non-identifying relationship* is indicated by a broken line and does not contain the FK in the PK. See the section in Chapter 8 that discusses weak and strong relationships for more explanation.

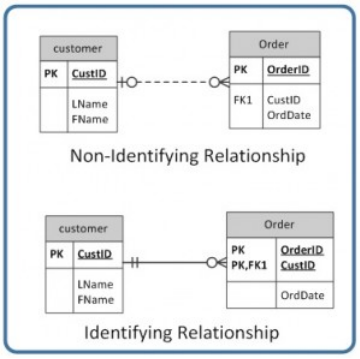


Figure 4.3.1: *Identifying and non-identifying relationship, by A. Watt.*

Optional relationships

In an *optional relationship*, the FK can be null or the parent table does not need to have a corresponding child table occurrence. The symbol, shown in Figure 4.3.2, illustrates one type with a zero and three prongs (indicating many) which is interpreted as zero OR many.

Figure 4.3.2: Zero or Many. ("Zero or Many" by Patrick McClanahan is licensed under CC BY-SA 4.0)

For example, if you look at the Order table on the right-hand side of Figure 4.3.3, you'll notice that a customer doesn't need to place an order to be a customer. In other words, the **many side** is optional.

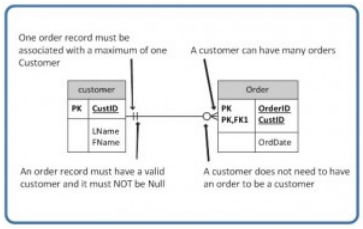


Figure 4.3.3: *Example usage of a zero to many optional relationship symbol, by A. Watt.*

The relationship symbol in Figure 4.3.3 can also be read as follows:

- Left side: The order entity must contain a minimum of one related entity in the Customer table and a maximum of one related entity.
- Right side: A customer can place a minimum of zero orders or a maximum of many orders.

Figure 4.3.4 shows another type of optional relationship symbol with a zero and one, meaning zero OR one. The **one side** is optional.

Figure 4.3.4:

Figure 4.3.5 gives an example of how a zero to one symbol might be used.

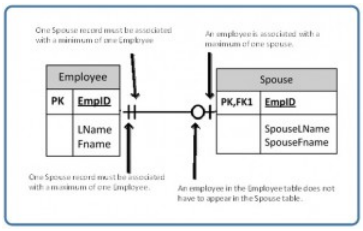


Figure 4.3.5: *Example usage of a zero to one optional relationship symbol, by A. Watt.*

Mandatory relationships

In a *mandatory relationship*, one entity occurrence requires a corresponding entity occurrence. The symbol for this relationship shows *one and only one* as shown in Figure 4.3.6. The one side is mandatory.

Figure 4.3.6:

See Figure 4.3.7 for an example of how the one and only one mandatory symbol is used.

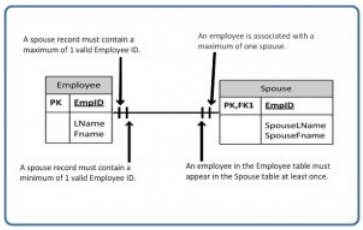


Figure 4.3.7: *Example of a one and only one mandatory relationship symbol, by A. Watt.*

Figure 4.3.8 illustrates what a one to many relationship symbol looks like where the **many side** is mandatory.

Figure 4.3.8:

Refer to Figure 4.3.9 for an example of how the one to many symbol may be used.

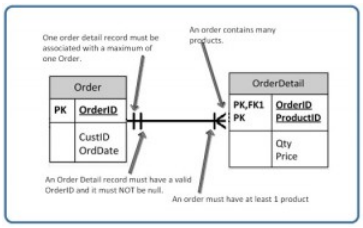


Figure 4.3.9: *Example of a one to many mandatory relationship symbol, by A. Watt.*

So far we have seen that the innermost side of a relationship symbol (on the left-side of the symbol in Figure 4.3.10) can have a 0 (zero) cardinality and a connectivity of many (shown on the right-side of the symbol in Figure 4.3.10), or one (not shown).

Figure 4.3.10:

However, it cannot have a connectivity of 0 (zero), as displayed in Figure 4.3.11. The connectivity can only be 1.

Figure \PageIndex11}:

The connectivity symbols show maximums. So if you think about it logically, if the connectivity symbol on the left side shows 0 (zero), then there would be no connection between the tables.

The way to read a relationship symbol, such as the one in Figure 4.3.12, is as follows.

- The CustID in the Order table must also be found in the Customer table a minimum of 0 and a maximum of 1 times.
- The 0 means that the CustID in the Order table may be null.
- The left-most 1 (right before the 0 representing connectivity) says that if there is a CustID in the Order table, it can only be in the Customer table once.
- When you see the 0 symbol for cardinality, you can assume two things: T
  1. the FK in the Order table allows nulls, and
  2. the FK is not part of the PK since PKs must not contain null values.

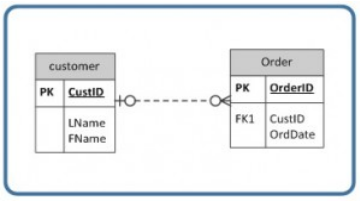


Figure 4.3.12: *The relationship between a Customer table and an Order table, by A. Watt.*

Key Terms

business rules: obtained from users when gathering requirements and are used to determine cardinality

cardinality: expresses the minimum and maximum number of entity occurrences associated with one occurrence of a related entity

connectivity: the relationship between two tables, e.g., one to one or one to many

constraints: the rules that force DBMSs to check that data satisfies the semantics

entity integrity: requires that every table have a primary key; neither the primary key, nor any part of it, can contain null values

identifying relationship: where the primary key contains the foreign key; indicated in an ERD by a solid line

integrity constraints: logical statements that state what data values are or are not allowed and which format is suitable for an attribute

mandatory relationship:one entity occurrence requires a corresponding entity occurrence.

non-identifying relationship: does not contain the foreign key in the primary key; indicated in an ERD by a dotted line

optional relationship: the FK can be null or the parent table does not need to have a corresponding child table occurrence

orphan record: a record whose foreign key value is not found in the corresponding entity – the entity where the primary key is located

referential integrity: requires that a foreign key must have a matching primary key or it must be null

relational database management system (RDBMS): a popular database system based on the relational model introduced by E. F. Codd of IBM's San Jose Research Laboratory

relationship type: the type of relationship between two tables in an ERD (either identifying or non-identifying); this relationship is indicated by a line drawn between the two tables.

Exercises

Read the following description and then answer questions 1-5 at the end.

The swim club database in Figure 9.5.16 has been designed to hold information about students who are enrolled in swim classes. The following information is stored: students, enrollment, swim classes, pools where classes are held, instructors for the classes, and various levels of swim classes. Use Figure 9.5.16 to answer questions 1 to 5.

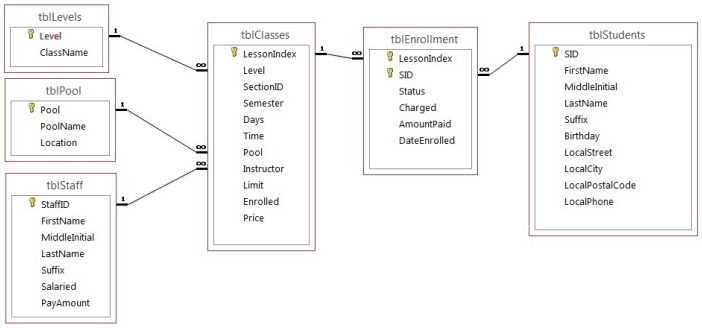


Figure 4.3.16: *ERD for questions 1-5. (Diagram by A. Watt.)*

The primary keys are identified below. The following data types are defined in the SQL Server.

tblLevels
Level – Identity PK
ClassName – text 20 – nulls are not allowed

tblPool
Pool – Identity PK
PoolName – text 20 – nulls are not allowed
Location – text 30

tblStaff
StaffID – Identity PK
FirstName – text 20
MiddleInitial – text 3
LastName – text 30
Suffix – text 3
Salaried – Bit
PayAmount – money

tblClasses
LessonIndex – Identity PK
Level – Integer FK
SectionID – Integer
Semester – TinyInt
Days – text 20
Time – datetime (formatted for time)
Pool – Integer FK
Instructor – Integer FK
Limit – TinyInt
Enrolled – TinyInt
Price – money

tblEnrollment
LessonIndex – Integer FK
SID – Integer FK (LessonIndex and SID) Primary Key
Status – text 30
Charged – bit
AmountPaid – money
DateEnrolled – datetime

tblStudents
SID – Identity PK
FirstName – text 20

MiddleInitial – text 3
LastName – text 30
Suffix – text 3
Birthday – datetime
LocalStreet – text 30
LocalCity – text 20
LocalPostalCode – text 6
LocalPhone – text 10

Implement this schema in SQL Server or access (you will need to pick comparable data types). Submit a screenshot of your ERD in the database.

1. Explain the relationship rules for each relationship (e.g., tblEnrollment and tblStudents: A student can enroll in many classes).
2. Identify cardinality for each relationship, assuming the following rules:
   - A pool may or may not ever have a class.
   - The levels table must always be associated with at least one class.
   - The staff table may not have ever taught a class.
   - All students must be enrolled in at least one class.
   - The class must have students enrolled in it.
   - The class must have a valid pool.
   - The class may not have an instructor assigned.
   - The class must always be associated with an existing level.
3. Which tables are weak and which tables are strong (covered in an earlier chapter)?
4. Which of the tables are non-identifying and which are identifying?

4.4: Functional Dependencies

A *functional dependency* (FD) is a relationship between two attributes, typically between the PK and other non-key attributes within a table. For any relation R, attribute Y is functionally dependent on attribute X (usually the PK), if for every valid instance of X, that value of X uniquely determines the value of Y. This relationship is indicated by the representation below :

X ————> Y

The left side of the above FD diagram is called the *determinant*, and the right side is the *dependent*. Here are a few examples.

In the first example, below, SIN determines Name, Address and Birthdate. Given SIN, we can determine any of the other attributes within the table.

SIN ————-> Name, Address, Birthdate

For the second example, SIN and Course determine the date completed (DateCompleted). This must also work for a composite PK.

SIN, Course ————> DateCompleted

The third example indicates that ISBN determines Title.

ISBN ————> Title

Rules of Functional Dependencies

Consider the following table of data r(R) of the relation schema R(ABCDE) shown in Table 4.4.1: .

A	B	C	D	E
a1	b1	c1	d1	e1
a2	b1	C2	d2	e1
a3	b2	C1	d1	e1
a4	b2	C2	d2	e1
a5	b3	C3	d1	e1

Table R

Table 4.4.1: *Functional dependency example, by A. Watt.*

As you look at this table, ask yourself: *What kind of dependencies can we observe among the attributes in Table R?* Since the values of A are unique (a1, a2, a3, etc.), it follows from the FD definition that:

$A \rightarrow B, \quad A \rightarrow C, \quad A \rightarrow D, \quad A \rightarrow E$

- It also follows that $A \rightarrow BC$ (or any other subset of ABCDE).
- This can be summarized as $A \rightarrow BCDE$.
- From our understanding of primary keys, A is a primary key.

Since the values of E are always the same (all e1), it follows that:

$A \rightarrow E, \quad B \rightarrow E, \quad C \rightarrow E, \quad D \rightarrow E$

However, we cannot generally summarize the above with $ABCD \rightarrow E$ because, in general, $A \rightarrow E$, $B \rightarrow E$, $AB \rightarrow E$.

Other observations:

1. Combinations of BC are unique, therefore $BC \rightarrow ADE$.
2. Combinations of BD are unique, therefore $BD \rightarrow ACE$.
3. If C values match, so do D values.
    1. Therefore, $C \rightarrow D$
    2. However, D values don't determine C values
    3. So C does not determine D, and D does not determine C.

Looking at actual data can help clarify which attributes are dependent and which are determinants.

Inference Rules

Armstrong's axioms are a set of inference rules used to infer all the functional dependencies on a relational database. They were developed by William W. Armstrong. The following describes what will be used, in terms of notation, to explain these axioms.

Let R(U) be a relation scheme over the set of attributes U. We will use the letters X, Y, Z to represent any subset of and, for short, the union of two sets of attributes, instead of the usual X U Y.

Axiom of reflexivity

This axiom says, if Y is a subset of X, then X determines Y (see Figure 4.4.1).

$$\text{If } Y \subseteq X, \text{ then } X \rightarrow Y$$

Figure 4.4.1: . *Equation for axiom of reflexivity.*

For example, **PartNo —> NT123** where X (PartNo) is composed of more than one piece of information; i.e., Y (NT) and partID (123).

Axiom of augmentation

The axiom of augmentation, also known as a partial dependency, says if X determines Y, then XZ determines YZ for any Z (see Figure 4.4.2).

$$\text{If } X \rightarrow Y, \text{ then } XZ \rightarrow YZ \text{ for any } Z$$

Figure 4.4.2: . *Equation for axiom of augmentation.*

The axiom of augmentation says that every non-key attribute must be fully dependent on the PK. In the example shown below, StudentName, Address, City, Prov, and PC (postal code) are only dependent on the StudentNo, not on the StudentNo and Grade.

StudentNo, Course —> StudentName, Address, City, Prov, PC, Grade, DateCompleted

This situation is not desirable because every non-key attribute has to be fully dependent on the PK. In this situation, student information is only partially dependent on the PK (StudentNo).

To fix this problem, we need to break the original table down into two as follows:

- Table 1: StudentNo, Course, Grade, DateCompleted
- Table 2: StudentNo, StudentName, Address, City, Prov, PC

Axiom of transitivity

The axiom of transitivity says if X determines Y, and Y determines Z, then X must also determine Z (see Figure 4.4.3).

$$\text{If } X \rightarrow Y \text{ and } Y \rightarrow Z, \text{ then } X \rightarrow Z$$

Figure 4.4.3: . *Equation for axiom of transitivity.*

The table below has information not directly related to the student; for instance, ProgramID and ProgramName should have a table of its own. ProgramName is not dependent on StudentNo; it's dependent on ProgramID.

StudentNo —> StudentName, Address, City, Prov, PC, ProgramID, ProgramName

This situation is not desirable because a non-key attribute (ProgramName) depends on another non-key attribute (ProgramID).

To fix this problem, we need to break this table into two: one to hold information about the student and the other to hold information about the program.

- Table 1: StudentNo —> StudentName, Address, City, Prov, PC, ProgramID
- Table 2: ProgramID —> ProgramName

However we still need to leave an FK in the student table so that we can identify which program the student is enrolled in.

Union

This rule suggests that if two tables are separate, and the PK is the same, you may want to consider putting them together. It states that if X determines Y and X determines Z then X must also determine Y and Z (see Figure 4.4.4).

$$\text{If } X \rightarrow Y \text{ and } X \rightarrow Z \text{ then } X \rightarrow YZ$$

Figure 4.4.4: . *Equation for the Union rule.*

For example, if:

- SIN —> EmpName
- SIN —> SpouseName

You may want to join these two tables into one as follows:

SIN –> EmpName, SpouseName

Some database administrators (*DBA*) might choose to keep these tables separated for a couple of reasons. One, each table describes a different entity so the entities should be kept apart. Two, if SpouseName is to be left NULL most of the time, there is no need to include it in the same table as EmpName.

Decomposition

Decomposition is the reverse of the Union rule. If you have a table that appears to contain two entities that are determined by the same PK, consider breaking them up into two tables. This rule states that if X determines Y and Z, then X determines Y and X determines Z separately (see Figure 4.4.5).

$$\text{If } X \rightarrow YZ \text{ then } X \rightarrow Y \text{ and } X \rightarrow Z$$

Figure 4.4.5: *Equation for decompensation rule.*

Dependency Diagram

A dependency diagram, shown in Figure 4.4.6 , illustrates the various dependencies that might exist in a *non-normalized table*. A non-normalized table is one that has data redundancy in it.

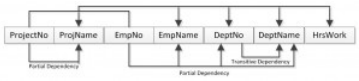


Figure 4.4.6: . *Dependency diagram.*

The following dependencies are identified in this table:

- ProjectNo and EmpNo, combined, are the PK.
- Partial Dependencies:
  - ProjectNo —> ProjName
  - EmpNo —> EmpName, DeptNo,
  - ProjectNo, EmpNo —> HrsWork
- Transitive Dependency:
  - DeptNo —> DeptName

Key Terms

Armstrong's axioms: a set of inference rules used to infer all the functional dependencies on a relational database

DBA: database administrator

decomposition: a rule that suggests if you have a table that appears to contain two entities that are determined by the same PK, consider breaking them up into two tables

dependent: the right side of the functional dependency diagram

determinant: the left side of the functional dependency diagram

functional dependency (FD): a relationship between two attributes, typically between the PK and other non-key attributes within a table

non-normalized table: a table that has data redundancy in it

Union: a rule that suggests that if two tables are separate, and the PK is the same, consider putting them together

---

CHAPTER OVERVIEW

5: ER Modeling

5.1: Relationships

One important theory developed for the entity relational (ER) model involves the notion of functional dependency (FD). The aim of studying this is to improve your understanding of relationships among data and to gain enough formalism to assist with practical database design.

Like constraints, FDs are drawn from the semantics of the application domain. Essentially, *functional dependencies* describe how individual attributes are related. FDs are a kind of constraint among attributes within a relation and contribute to a good relational schema design. In this chapter, we will look at:

- The basic theory and definition of functional dependency
- The methodology for improving schema designs, also called normalization

Relational Design and Redundancy

Generally, a good relational database design must capture all of the necessary attributes and associations. The design should do this with a minimal amount of stored information and no redundant data.

In database design, redundancy is generally undesirable because it causes problems maintaining consistency after updates. However, redundancy can sometimes lead to performance improvements; for example, when redundancy can be used in place of a *join* to connect data. A *join* is used when you need to obtain information based on two related tables.

Consider Figure 5.1.1 customer 1313131 is displayed twice, once for account no. A-101 and again for account A-102. In this case, the customer number is not redundant, although there are deletion anomalies with the table. Having a separate customer table would solve this problem. However, if a branch address were to change, it would have to be updated in multiple places. If the customer number was left in the table as is, then you wouldn't need a branch table and no join would be required, and performance is improved .

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hil	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hil	Horseneck	8000000

Bank Accounts

Figure 5.1.1: *An example of redundancy used with bank accounts and branches.*

Insertion Anomaly

An *insertion anomaly* occurs when you are inserting inconsistent information into a table. When we insert a new record, such as account no. A-306 in Figure 5.1.2, we need to check that the branch data is consistent with existing rows.

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hil	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hil	Horseneck	8000000
A-306	800	1111111	Round Hil	Horseneck	8000800

Insertion anomaly - Insert account A-306 at Round Hill

Figure 5.1.2: *Example of an insertion anomaly.*

Update Anomaly

If a branch changes address, such as the Round Hill branch in Figure 5.1.3, we need to update all rows referring to that branch. Changing existing information incorrectly is called an *update anomaly*.

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Palo Alto	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000

Update Anomaly - Round Hill branch address

Figure 5.1.3: *Example of an update anomaly.*

Deletion Anomaly

A *deletion anomaly* occurs when you delete a record that may contain attributes that shouldn't be deleted. For instance, if we remove information about the last account at a branch, such as account A-101 at the Downtown branch in Figure 5.1.4, all of the branch information disappears.

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000

Deletion anomaly - Bank Account

Figure 5.1.4: *Example of a deletion anomaly.*

The problem with deleting the A-101 row is we don't know where the Downtown branch is located and we lose all information regarding customer 1313131. To avoid these kinds of update or deletion problems, we need to decompose the original table into several smaller tables where each table has minimal overlap with other tables.

Each bank account table must contain information about one entity only, such as the Branch or Customer, as displayed in Figure 5.1.5.

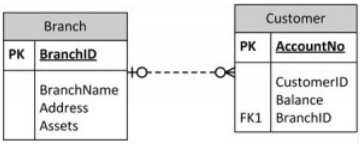


Figure 5.1.5: *Examples of bank account tables that contain one entity each, by A. Watt.*

Following this practice will ensure that when branch information is added or updated it will only affect one record. So, when customer information is added or deleted, the branch information will not be accidentally modified or incorrectly recorded.

Example: employee project table and anomalies

Figure 5.1.6 shows an example of an employee project table. From this table, we can assume that:

1. EmpID and ProjectID are a composite PK.
2. Project ID determines Budget (i.e., Project P1 has a budget of 32 hours).

EmpID	Budget	ProjectID	Hours
S75	32	P1	7
S75	40	P2	3
S79	32	P1	4
S79	27	P3	1
S80	40	P2	5
	17	P4	

Figure 5.1.6: *Example of an employee project table, by A. Watt.*

Next, let's look at some possible anomalies that might occur with this table during the following steps.

1. Action: Add row {S85,35,P1,9}
2. Problem: There are two tuples with conflicting budgets
3. Action: Delete tuple {S79, 27, P3, 1}

4. Problem: Step #3 deletes the budget for project P3
5. Action: Update tuple {S75, 32, P1, 7} to {S75, 35, P1, 7}
6. Problem: Step #5 creates two tuples with different values for project P1's budget
7. Solution: Create a separate table, each, for Projects and Employees, as shown in Figure 5.1.7.

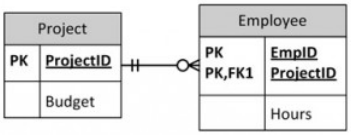


Figure 5.1.7: *Solution: separate tables for Project and Employee, by A. Watt.*

How to Avoid Anomalies

The best approach to creating tables without anomalies is to ensure that the tables are normalized, and that's accomplished by understanding functional dependencies. FD ensures that all attributes in a table belong to that table. In other words, it will eliminate redundancies and anomalies.

Example: separate Project and Employee tables

Project table

ProjectID	Budget
P1	32
P2	40
P3	27
P4	17

Employee table

EmpID	ProjectID	Hours
S75	P1	7
S75	P2	3
S79	P1	4
S79	P3	1
S80	P2	5

Figure 5.1.8:*Separate Project and Employee tables with data, by A. Watt.*

By keeping data separate using individual Project and Employee tables:

1. No anomalies will be created if a budget is changed.
2. No dummy values are needed for projects that have no employees assigned.
3. If an employee's contribution is deleted, no important data is lost.
4. No anomalies are created if an employee's contribution is added.

Key Terms

deletion anomaly: occurs when you delete a record that may contain attributes that shouldn't be deleted

functional dependency (FD): describes how individual attributes are related

insertion anomaly: occurs when you are inserting inconsistent information into a table

join: used when you need to obtain information based on two related tables

update anomaly: changing existing information incorrectly

Exercises

1. Normalize Figure 5.1.9.

Attribute Name	Sample Value	Sample Value	Sample Value
StudentID	1	2	3
StudentName	John Smith	Sandy Law	Sue Rogers
CourseID	2	2	3
CourseName	Programming Level 1	Programming Level 1	Business
Grade	75%	61%	81%
CourseDate	Jan 5th, 2014	Jan 5th, 2014	Jan 7th, 2014

Figure 5.1.9: *Table for question 1, by A. Watt.*

2. Create a logical ERD for an online movie rental service (no many to many relationships). Use the following description of operations on which your business rules must be based:The online movie rental service classifies movie titles according to their

type: comedy, western, classical, science fiction, cartoon, action, musical, and new release. Each type contains many possible titles, and most titles within a type are available in multiple copies. For example, note the following summary:TYPE TITLE
Musical My Fair Lady (Copy 1)
My Fair Lady (Copy 2)
Oklahoma (Copy 1)
Oklahoma (Copy 2)
Oklahoma (Copy 3)
etc.

3. What three data anomalies are likely to be the result of data redundancy? How can such anomalies be eliminated?

Also see *Appendix B: Sample ERD Exercises*

CHAPTER OVERVIEW

6: Relationship Diagram for Data Analysis-ER and SQL

6.1: Introduction and Background

ERD stands for Entity Relationship Diagram. An ER diagram illustrates the logic within a database and how individual components relate to one another (Visual Paradigm). Similar to a story board for film, an ER diagram lays out the wireframe of internal database structures to aid in understanding, improving efficiency, and debugging logical errors. The key pieces of an ER diagram are the entities, attributes, and the relationships between them. For example, within the complex systems of Amazon, they would have an entity for customer listing attributes such as name, address, payment, and email. Once an order is placed and ready for shipment, the shipping database would access the customer database to retrieve information to complete the order and ship out the package. An ER diagram would show how the shipping department can access many customers within the database, and the information it would require to complete shipment.

In the 1960s and 1970s, there was a rise in data modeling needs and published works on different models. Peter Chen is attributed with inventing and promoting the entity-relationship model stemming from a paper he published in 1976, “The Entity-Relationship Model – Toward a Unified View of Data” (Kempe, 2013). As databases and businesses grew more complex, a unified modeling system became more important so that structures could be modeled across industries consistently. After the original ERD was developed by Peter Chen, many of his colleagues adapted and made small alterations to improve and simplify the overall model. The key components are still present; the look has been simplified to accommodate more complex systems. Below we see examples of both the original and newer ERD models.

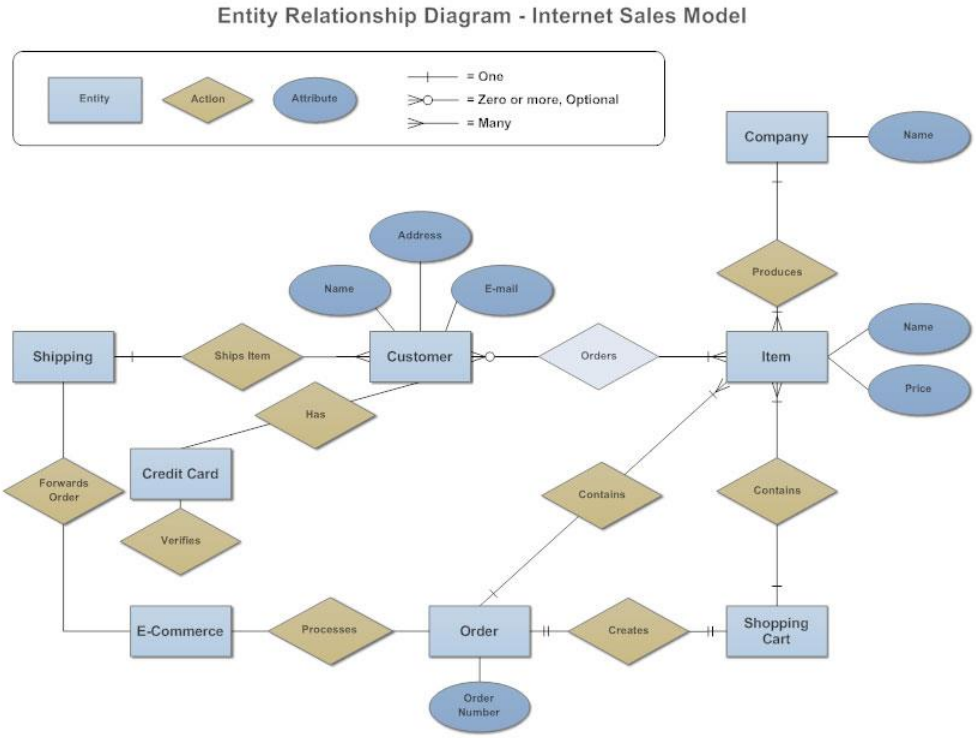


Figure 6.1.1: Original Style ERD (“Module_2 - Relationship Diagram for Data Analysis & SQL” by Dr Sarah North, Affordable Learning Georgia)

1. This is an example of an older ERD version depicting e-commerce Company. We can tell because there are action diamonds between entities, and attributes are each listed separately in individual ovals.

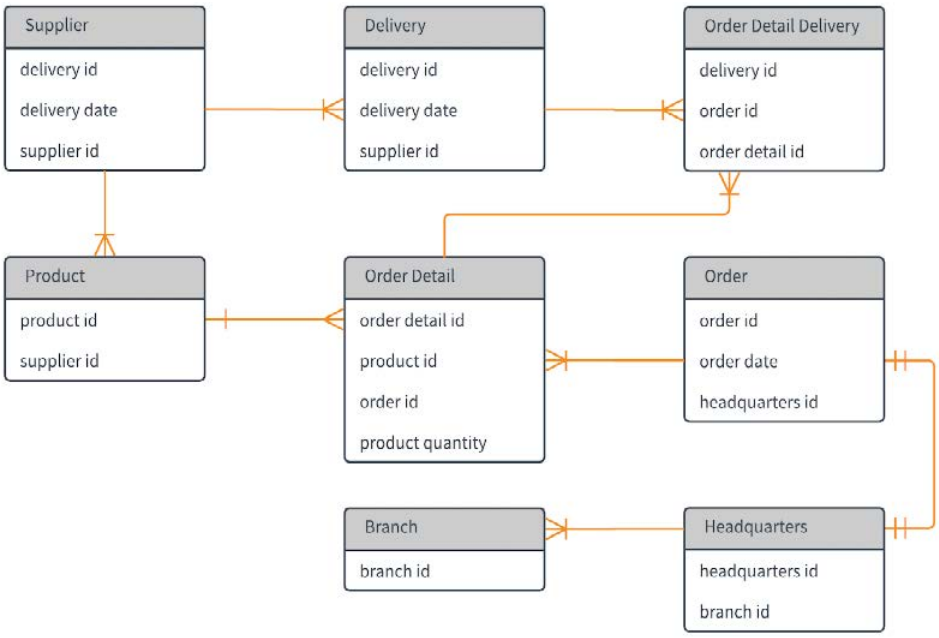


Figure 6.1.2: New Style ERD. ("Module_2 - Relationship Diagram for Data Analysis & SQL" by Dr Sarah North, Affordable Learning Georgia)

2. In this newer ERD e-commerce example, we see attributes listed within each entity, and no actions are present. If depicted in the older model, we would see 21 attribute ovals and a minimum of 8 actions. This would create visual clutter, making the ERD harder to examine, detracting from its usefulness.

6.2: Understanding the Importance of Data Modeling

Keeping a well-documented model of a database is important for overall business efficiency. ER diagrams are easy to use and understand, allowing for simple cross-departmental understanding. As new changes or expansions are proposed, an understanding of current workflow and capabilities is key to managing these projects. Also, as new team members join, an ERD allows for quick and efficient training. New and current team members can revisit this often as a reference while working on individual pieces to understand the overall effect.

While the foundation of an ERD is to show the relationship between entities, they can also be helpful in other instances. Designing and debugging databases can be very complex tasks, but having ER diagrams can simplify the process and reduce the chance of error. In designing an entirely new system, an ER diagram can aid in planning the logic and requirements of a database before any part of it is built. This can save money and hours, and it ultimately cuts down on redundancies and time spent on unneeded builds. Visualizing the overall structure allows programmers to locate any errors in logic or flaws in the overall design before creation. Once a system is already created, an ERD is still useful in debugging as errors occur. Because developers can see how all entities and attributes are linked, it is easier to trace down where any error stems from. In fixing the issue, the ERD also helps in seeing if any side effects may occur (Visual Paradigm).

ER diagrams are especially useful in organizations with bigger and more complex database systems that have many different entities, data points, and relationships between them. A good example of this can be found in any major hospital. All hospitals have doctors and patients, but the larger ones deal with doctors across different specialized departments and all the components that interact. When a patient makes a yearly checkup appointment online, their appropriate insurance is checked and logged into their profile. When they arrive, the doctor is able to access all previous medical records and current medication, even if prescribed by a different doctor. Any samples taken are processed by the laboratory, who have editing access to the patient profile in order to update the results.

The information is then posted to the account, and both the doctor and patient are notified. Patients have view-only capabilities, while doctors can edit and add on prescriptions as needed. If a prescription is added, the in-house pharmacy is able to view and notify patients when filled. While this process is going on, the billing department is able to see activities in all other departments as it relates to the individual case, viewing insurance information to forward and process bills. They are able to communicate between both the insurance provider and the patient to fulfill total costs.

In this example, the process is not linear, and each department has different levels of access and editing capabilities. Not included in the example, but still present in the situation, are all of the back-end human resources for the hospital as an organization with employees. This becomes increasingly complex as more patients, doctors, specializations, and privacy concerns are added into the mix. In health care professions where certain situations can literally be life or death, efficiency is of the utmost importance. ER diagrams are essential to keeping hospital databases well-organized and free from error. While the figure below depicts a simplified and basic interaction with a doctor’s office, we can already see many entities and complex relationships between them. A full-service hospital ER diagram would be much
larger.

6.3: Naming and Definitions

Now that we have covered what an ER diagram is, let's go into more detail regarding the finer points of data modeling. ER diagrams are used to visually represent data relationships in many settings, but they are most commonly used in business environments. These data representations allow many companies to better use their resources and make well-informed decisions.

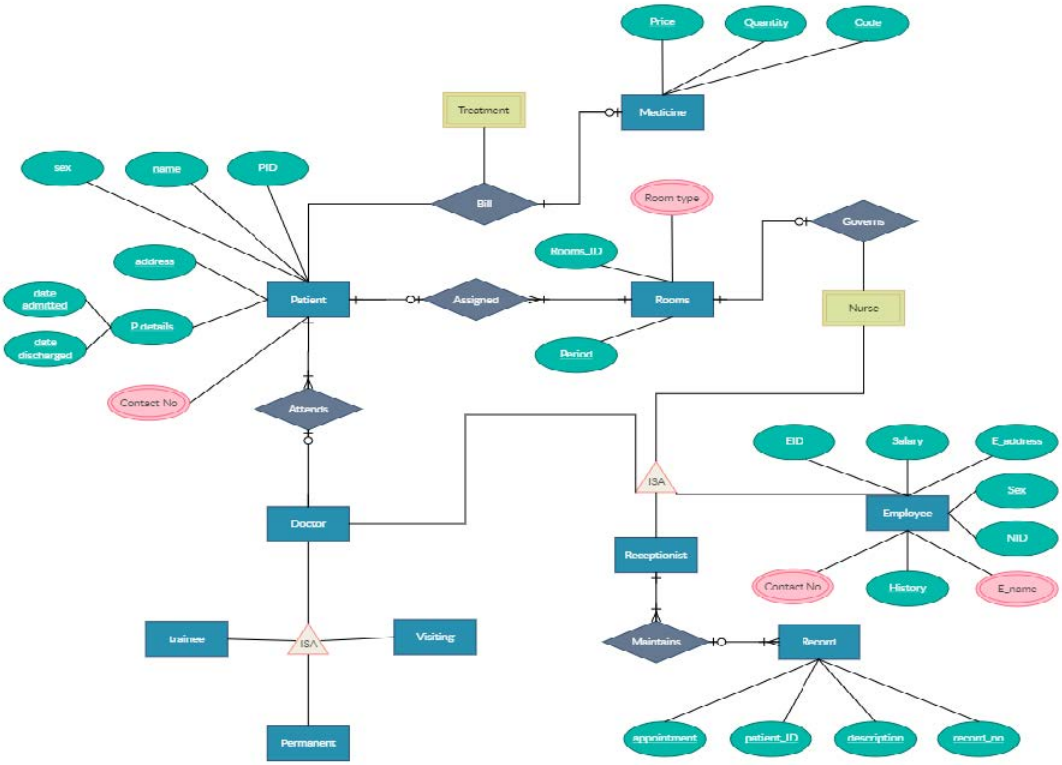


Figure 6.3.1: Basic Hospital ERD. ("Module_2 - Relationship Diagram for Data Analysis & SQL" by Dr Sarah North, Affordable Learning Georgia)

These diagrams should be easily representable and understandable by a wide majority of employees, as they represent data and/or inner functions of the company. Therefore, it is obvious that such diagrams must be easily readable and functional to allow quick decision-making.

Speaking of readability, in these ER diagrams, naming values and objects is half the battle. Something that shares this need is programming, as creating variables and methods with explanatory yet short names helps to increase readability of the code. Understanding this, writing ER diagrams brings many similar concepts, so writing an ER diagram is nearly the same as writing the code that drives it. The value of self-explanatory variable names cannot be understated, as they help keep the company and its employees informed. However, explaining such concepts without an example is needlessly abstract, so let's start by making a few tables and bringing them up to snuff.

Figure 6.3.2: Entity Relation Tables. ("Module_2 - Relationship Diagram for Data Analysis & SQL" by Dr Sarah North, Affordable Learning Georgia)

Now with these tables, we can start to fix problems, but first let's start with some baselines. Firstly, we can see an empty value in table 6.3.1, and with no relations, table names, or variable names these tables do not convey any meaningful information. Choosing an example set of data, we can display the relationship between a Student, a College, and their Professors.

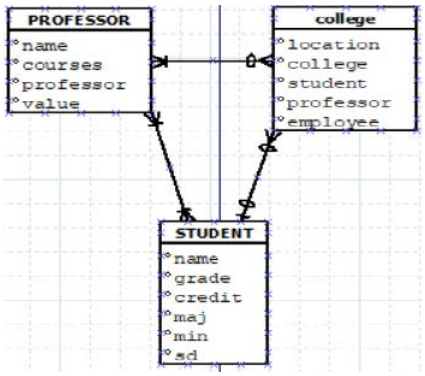


Figure 6.3.3: College, Professor, Student ER Diagram. ("Module_2 - Relationship Diagram for Data Analysis & SQL" by Dr Sarah North, Affordable Learning Georgia)

Now this table is still far from ideal. Tables share variables, and these names are terrible, and would require a large amount of explanation to make sense. Let's start fixing it step by step. Our first objective will be getting the data to a more readable state. The variable names currently in the table are atrociously low on elaboration or are so general that anything could fit where they are. Other variables are even repeated, so if they were used as a unique identifier or a primary key the program would give irrelevant or hindering information. This would lead to many problems, both in coding and in viewing. Let's make some edits to the table to better represent the variables that are in there right now.

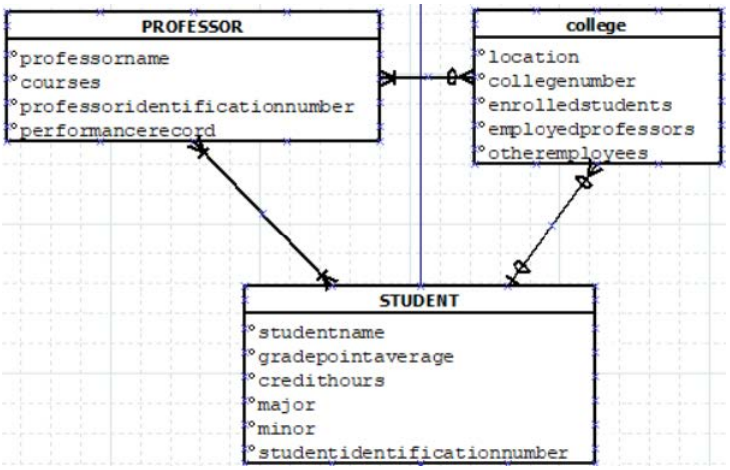


Figure 6.3.4: ER Diagram with Corrected Attribute Names. ("Module_2 - Relationship Diagram for Data Analysis & SQL" by Dr Sarah North, Affordable Learning Georgia)

Now that the variables have been clarified, we can begin fixing some finer details like the relations between each table. The student and college relation is not indicative of the general format. A student at college will only have one college, and a college will have far more than one student enrolled. Additionally, professors that teach generally tend to stay at one college, so changing that correlation will better fit the represented data. Fixing this will better clarify the tables' reliance on each other, as well as opening more relation options between them.

=

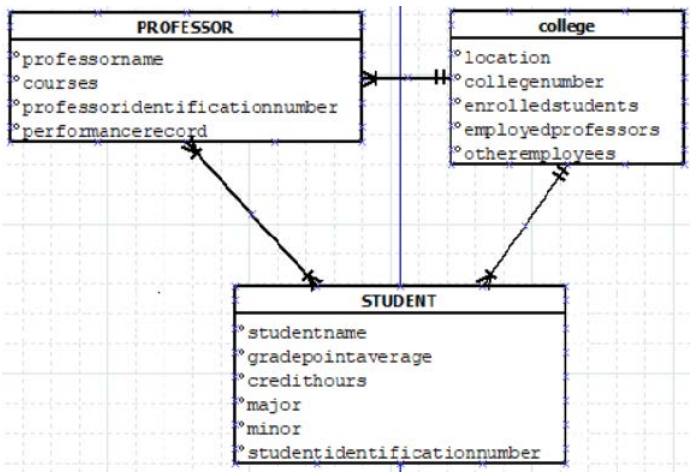


Figure 6.3.5: ER Diagram With Corrected Relation Types. ("Module_2 - Relationship Diagram for Data Analysis & SQL" by Dr Sarah North, Affordable Learning Georgia)

Now with the inter-table relations fixed, our focus can shift to the formatting of variables and table names. Some variables have far too long a name to be practically used or understood. Be careful when fixing these names, however, as cutting off too much or using unorthodox acronyms can lead to the same confusion regarding the variables. Also, no one wants to read super long variable names, and doing so in a conference or meeting puts an unnecessarily large amount of time spent explaining what a simple diagram should be.

In programming, such variables would never be used, mainly because no one wants to type out student identification nnumber, and then find out they misspelled it. Additionally, such values are memory inefficient, wasting space that could be used to make processes run faster. So, now to fix the table, look for shorter variable names or acronyms to replace the names already there, but remember, going for shortness over functionality could lead to a similarly cryptic graph.

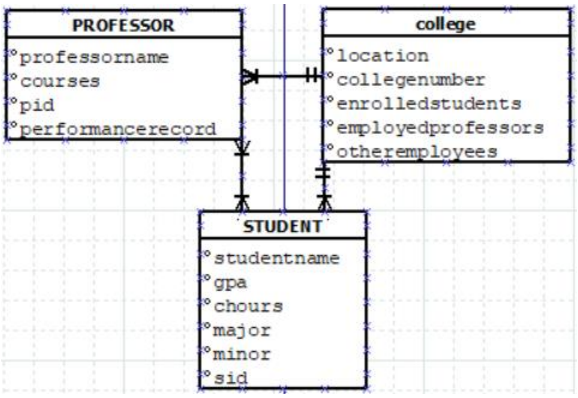


Figure 6.3.6: ER Diagram With Corrected Relation Formatting. ("Module_2 - Relationship Diagram for Data Analysis & SQL" by Dr Sarah North, Affordable Learning Georgia)

Now that these variable names are much shorter, let's format them more professionally. Capitalization is an easy way to better present data in a professional way. Capitalizing acronyms and oneword variables is one more layer to add to the credibility of the relation. Additionally, we can format variables in a better way than just a long string of characters. There are many naming conventions common in programming that can be applied outside of code to ER diagrams, but for this example, we will use the Camel case convention. Camel case uses capital letters after the first word to increase readability of longer variable names (ex: numapplepie to numApplePie).

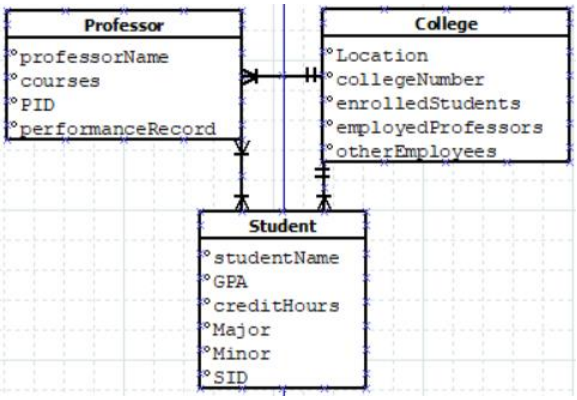


Figure 6.3.7: ER Diagram With Corrected Attribute Format. ("Module_2 - Relationship Diagram for Data Analysis & SQL" by Dr Sarah North, Affordable Learning Georgia)

This table has come a long way, and now that it is properly formatted, let's focus on details like the primary key for each table. For formatting, the primary key should go at the top of the table variables to better represent its role. Adding to the layers of complexity, let's examine some of the variables and their use. What about student names? Attempting to separate the name down into 2-3 parts is a tactic that is required, as most universities and colleges order role via last name.

So let's add some clarification to that name. Remember if we apply some convention to the visual, it should carry through, so if we edit studentName, we must edit the professorName variable as well. Another topic to cover is sub tables, that is tables that elaborate on certain variables within another linked table. In the next example, we will add the dependent variable to the Professor table as well as a sub table. One other topic to cover is useless data. Look at the college table and see that there are two employee variables. That's a waste of both space and data, so they can be combined into one variable.

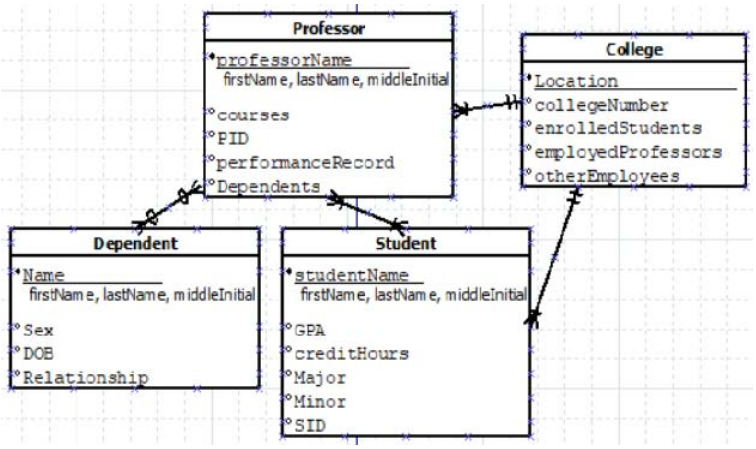


Figure 6.3.8: ER Diagram with Corrected Keys. ("Module_2 - Relationship Diagram for Data Analysis & SQL" by Dr Sarah North, Affordable Learning Georgia)

Now that all these edits have been applied, this table has been fixed from its confusing original state to one that could be shown to a conference room of businesspeople. All the changes were meant to improve both the appearance and readability of the data. Of course, some sacrifices were made to fit tables like these into this form. Ideally, an ER diagram should have a good bit of white space in between
the tables to better separate them from each other. But stepping away from the massive amount of refining we had to do, these should not be applied after the table was written, but during the process. This allows for less overall time spent making the table, and more time available to improve or enhance the model or programming behind it. But let's step away from the complexity and formatting of these tables to better grasp the concept of modeling entities and their relationships.

When modeling new or different entities, it is best to approach the problem step by step. Start by splitting up the work into different categories: the initial entity, the attributes of that entity, and the entity's relationships to other values. Applying this logic to a new table, let's use a shipping company as an example. Firstly, let's identify the company's entities, which will be the tables necessary to understand its inner workings. Entities like orders and storage are very important, but there are far too many entities to model, so

depending on what the meeting entails, tailoring the data to fit the occasion is a necessity. For instance, showing what customer data you get in a meeting discussing storage employee importance is unimportant to the current topic.

Speaking of customer data, the next step in making a comprehendible ER diagram revolves around getting the attributes and variables that each entity has. Remember to make this data easily recognizable by its name. That name should represent the data contained within, because while a variable named Ren_and_Stimpy is very recognizable, if it contained an order number, that would be terribly confusing. Customer data might include things like their name, address, and so forth. Customer data should, however, not include extraneous details like their sex or skin color. While elaborating on what each entity has, be careful not to program personal details to be stored carelessly, as data leaks have become more and more common over the years. Keeping the data secure is important, but remember, the less you collect, the easier it is to encrypt or protect from outside influence.

6.4: Modeling

Finally, speaking of data, entities and variables might have relationships to other entities, so this should be the last phase of ER diagram assembly. Take, for example, the order and customer entities in the context of the company. The relationship is as such, for each customer, they may have multiple orders, while each order will only have one customer who made it. The storage facility may have multiple incoming orders or requests, but the order only references one facility.

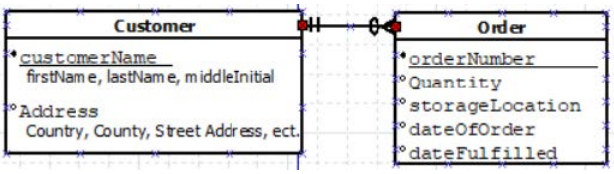


Figure 6.4.1: Entity-to-Entity Relation. ("Module_2 - Relationship Diagram for Data Analysis & SQL" by Dr Sarah North, Affordable Learning Georgia)

Thus we can see the importance of properly laying out and describing ER relations to better represent the data. However, let's address some smaller, more technical issues that may arise when creating a diagram. Given a diagram, sometimes concepts or variables put onto the board that others are not familiar with. Say you use some in-house acronyms at a large event, and then people ask questions about those topics. That is wasted time that could have been spent asking better questions that you prepared to answer. Remember that while you want to inform your audience, it is important to present that in an easily comprehendible way. Following some commonly used conventions, as well as using some advice outlined above, will allow you to work smarter, not harder.

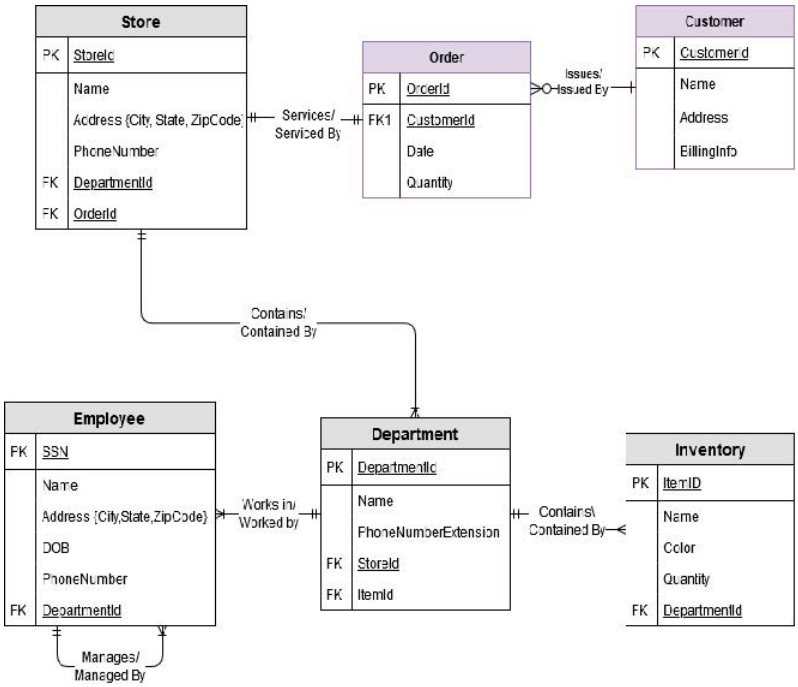


Figure 6.4.1: Store ER Diagram. ("Module_2 - Relationship Diagram for Data Analysis & SQL" by Dr Sarah North, Affordable Learning Georgia)

6.5: Business Intelligence Systems and Data Warehouse

Business Intelligence (BI) Systems are any database system that has data, programs, and personnel and are specialized for the preparation of data for BI processing. A Data Warehouse is a system used for reporting and data analysis, and is considered a core component of business intelligence. This includes day-to-day transactions and other internal data or external data from sources by the Extract, Transform, and load (ETL) System.

Database SKUs (A stock-keeping unit (SKU) is a scannable bar code, most often seen printed on product labels in a retail store) where the primary key consists of one or more column will uniquely identify each row or record in the table. Some tables may have one or more foreign keys that are used to create relationship between tables logically link the tables together.

6.6: Introduction to Structure Query Language (SQL)

SQL is the universal query language of relational database management systems (DBMS) that is almost always behind user-friendly GUIs. In this section, we will briefly talk about SQL queries. We will visit SQL in more detail in later chapters.

SQL statements can also include a SQL comment, which is a block of text used to document the SQL statement but not executed as part of the statement. SQL comments are enclosed in the symbols /* and */, and any text between these symbols is ignored when the SQL statement is executed. Here is an example:

/* SQL-Query-Ch02 */

The fundamental statement of SQL query can apply to Microsoft Access, SQL Server, Oracle Database, and MySQL. The basic form of SQL queries uses SQL SELECT – FROM – WHERE framework. Here are some basic specifications:

- The SQL SELECT statement specifies which columns are to be listed in the query results.
- The SQL FROM statement specifies which tables are to be listed in the query results.
- The SQL WHERE statement specifies which rows are to be listed in the query results.

Reading Specified Columns from Single Table – to obtain values from the SKU_Data table, we write a SQL SELECT statement that contains all of the column names in the table:

/* SQL-Query-Ch_02 */

	SKU	SKU_Description	Department	Buyer
1	200201	Women's T-Shirts	Clothing	Tyra Perry
2	200202	Mans's T-Shirts	Clothing	Bradly Cooper
3	200203	Children T-Shirt	Clothing	Cora Mathis

```
SELECT    SKU, SKU_Description, Department, Buyer
FROM      SKU_DATA;
```

Below is a SQL queries from a single table, which obtains just the value of the Department and Buyer columns of the SKU_Data table:

Department	Buyer
Clothing	Tyra Perry
Clothing	Bradly Cooper
Clothing	Cora Mathis

```
SELECT    Department, Buyer
FROM      SKU_DATA;
```

The Catalog_SKU_2019 table shows DateOnWebSite. To see these items, we use the follow query: /* SQL-Query-Ch_02 */

	SKU	SKU_Description	Department	Buyer	Date on WebSite
1	200201	Women's T-Shirts	Clothing	Tyra Perry	2020-10-10
2	200202	Mans's T-Shirts	Clothing	Bradly Cooper	2019-11-11
3	200203	Children T-Shirt	Clothing	Cora Mathis	2020-20-20

```
SELECT    *
FROM      CATALOG_SKU_2020
WHERE     DateOnWebSite = '10-Oct-2020';
```

6.7: Submitting SQL Statement to the DBMS

Depending on what DBMS is being used, the process to submit and execute SQL queries differ. Some DBMS have graphical interfaces that make it easy to create and execute an SQL query. Others have command line capabilities for typing in the SQL query directly.

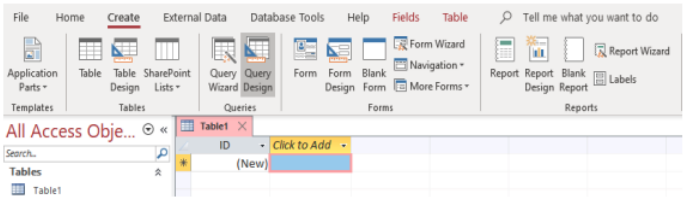


Figure 6.7.1: Microsoft Access 2016 Design view. ("Module_2 - Relationship Diagram for Data Analysis & SQL" by Dr Sarah North, Affordable Learning Georgia)

6.8: Concise Summary

An Entity Relationship Diagram, or ER diagram, is a visual representation of a database and the interactions between its entities and attributes. The need for ER diagrams arose from the need to structure complex databases in the 1970s. Aspects of ER diagrams that are appealing include their not only being easy to understand but also easy to create and teach.

One of the earliest models was developed by Peter Chen, which included entities (rectangles), attributes (ovals) and their relationships (diamonds). Newer ER diagrams (using Unifying Modeling Language) simplify this by attaching attributes to their entities in a rectangle and by having relationships designated by lines with specific end-connection symbols that represent the following:

- One-to-one
- One-to-many
- Many-to-one
- Many-to-many

All four of the above relationships are also combined with a symbol for mandatory or optional cardinality.

There are a few naming conventions that have evolved since Chen's original model. Entity names are typically one word, bolded, and put in the upper rectangle (the case can be interchangeable, but most use either Pascal or Camel). When keys are introduced, primary keys are camel-case and underlined, and secondary keys are just camel-case; all keys must have the same name as their corresponding entities. Attributes are camel-case, unique, and as short as possible.

ER diagrams allow a prebuild for databases which saves time, effort, and confusion for programming teams. They are also a great tool when used to help diagnose errors in the database. These simple diagrams are crucial when creating a database as efficiently and quickly as possible.

6.9: Extended Resources

1. This explains ER diagrams in depth. They go over each of the main parts; entity, attribute, and relationship, and each of the variations of them.
https://beginnersbook.com/2015/04/e-r-model-in-dbms/

2. https://www.tutorialspoint.com/dbms/...esentation.htm
This site shows the old method of making ER diagrams and gives a good description of each of the types of connections between the entities.

3. This resource shows two pictures; one shows possible individual components of an ERD, and the other depicts an example of an ERD using all the components.
https://www.oreilly.com/library/view...6lev1sec5.html

4. This video covers ER basics; what ER diagrams are, when to use them, and useful tips for creating them. The video shows an example of an ERD being drawn out in 'smart draw' to get a better understanding on how one is made from scratch
https://www.youtube.com/watch?v=dUJp0Yoq5eg

5. In this article they go over a brief history of ER diagrams. They discuss the different people involved with the creation and changes made to the models over time.
https://www.dataversity.net/a-short-...tion-modeling/

6. ER Diagram Template: Old version
https://www.lucidchart.com/pages/tem...agram-template

7. ER Diagram Template: New version
https://www.lucidchart.com/pages/tem...agram-template

6.10: References

SmartDraw. Entity Relationship Diagram. (n.d.). Retrieved from https://www.smartdraw.com/entityrela...ram/#whatIsERD

Kempe, S. (2013, November 20). A Short History of the ER Diagram and Information Modeling. Retrieved from https://www.dataversity.net/a-short-...ion-modeling/#

Lucidchart. Template: Entity Relationship Diagram. (n.d.). Retrieved from https://lucidchart.zendesk.com/hc/en...onship-Diagram

Visual Paradigm. What is Entity Relationship Diagram (ER)? (n.d.). Retrieved from https://www.visualparadigm.com/guide...nship-diagram/

Creately 2008. R Diagram for Hospital Management System (Entity Relationship Diagram). (n.d.). Retrieved March 22, 2020, from https://creately.com/diagram/example/hwj0b7oy1/E-R Diagram for Hospital Management System

CHAPTER OVERVIEW

7: Mapping ER to Schema, Normalization

7.1: Introduction and Background

Mapping an Entity Relationship (ER) model gives a good overview of the design of a system with the goal of making the system easier to understand at a technical level. The ER diagrams can be mapped to a relation schema, which means we can clearly display the relationship between its members. The database **schema** is a structure that describes in a formal language the association of data as a blueprint of how the database can be constructed.

To understand this process, we will review below what an ER model is. An ER model is used to illustrate the logical interpretation of the system and consists of three components: Entity, Entity Type and Entity Set.

Entity can refer to a person, a concept, an object, a virtual file, or it can represent an idea that can be quantified, such as a company, a job, or a document. An entity consists of an Entity Type and everything that consists of an entity type is called an Entity Set. The example below will clarify the power of design a schema showcasing an entity type, “Car,” and an entity set consisting of all cars.

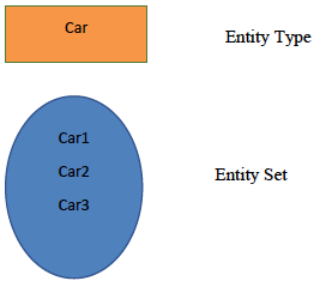


Figure 7.1.1: An Entity Set of All Cars. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

The next important step in understanding the mapping is an Attribute. Attributes are properties which define the entity type. For example, a person may have the attributes name, date of birth, and address. The attribute in an ER diagram is represented by an oval.

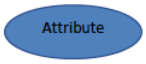


Figure 7.1.1: Attribute in an ER Diagram. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

The attribute that uniquely identifies each entity is called a Key Attribute and is represented by an oval with underlining. A key attribute can be “ID,” which identifies each entity by a unique number.

Figure 7.1.1: Key Attribute in an ER Diagram. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

For complex designs, there are multiple different types of attributes: Composite Attributes, consisting of many individual attributes, Multivalued Attributes, consisting more than one value, and Derived Attributes, which are derived from other attributes as the name suggests.

Between entities, there are relationships. A relationship has a type and can be part of a sets. The latter consists of one, two or many relationship types: unary, binary or N-ary.

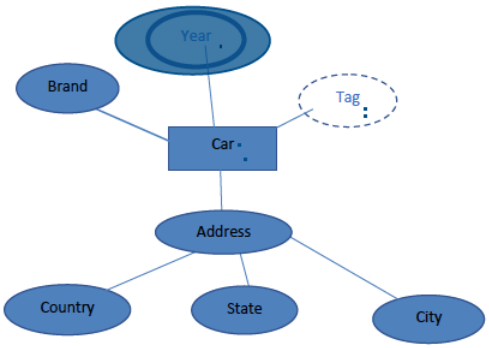


Figure 7.1.1: Multiple Different Types of Attributes. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

Relationship Type:

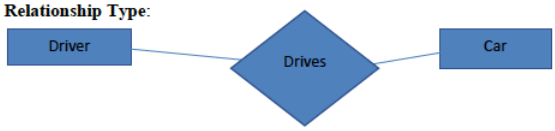


Figure 7.1.1: Relationship Type. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

Relationship Set:

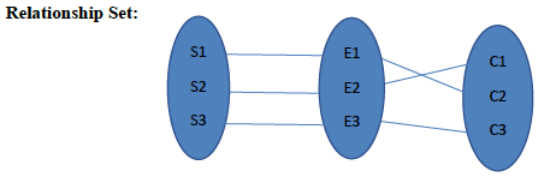


Figure 7.1.1: Relationship Set. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

Unary, Binary Relationship:

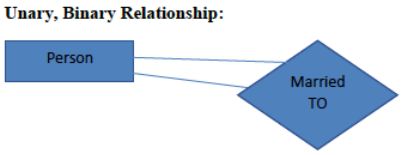


Figure 7.1.1: **Unary, Binary Relationship** ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

An ER diagram can also have a Weak Entity. A weak entity is an entity type for which a key attribute can't be defined. A company can store information of dependents (Parents, Children, Spouse) of an Employee but the dependents don't exist without the Employee; we call this entity a weak entity.

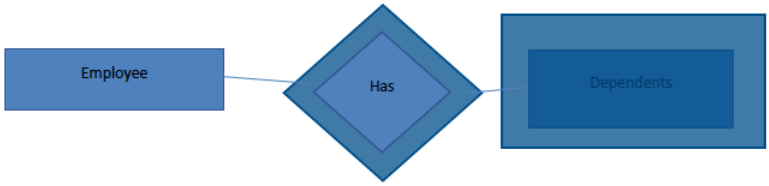


Figure 7.1.1: Weak Entity. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

7.2: List five properties of relations

In database systems, there exist relations in a two-dimensional table of data. This typically has multiple named columns and an indeterminate quantity of rows for data to be entered. A common misconception is a database relation being mistaken for a relational database. The main point to remember is that a relation specifically refers to an individual table in a relational database. There are specific properties that define relations so that they can be easily identified by a user. These relations have many properties that identify and separate themselves from non-relational tables.

Properties of relations:

1. Each table in a database has a unique identity (name).
2. Any entry at the connection of each row and column has a single value. There can be only one value that is related with each attribute on a specific row of a table; no multivalued attributes are allowed in a relation.
3. Each row is unique; no two rows or tables in the same relation can be identical.
4. Each attribute (or column) within a table has a unique name.
5. The sequence of columns (left to right) is insignificant. The order of the columns in a relation can be changed without changing the meaning or use of the relation.
6. The sequence of rows (top to bottom) is insignificant. As with columns, the order of the rows of a relation may be changed or stored in any sequence.

A few of these properties tie into one another, namely, the first, fourth, and sixth. The columns having unique attributes and the individual rows being arbitrarily organized very simply explains the core properties of relations.

Figure 7.2.1: Example of Property One. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

The second property speaks to Codd's original notion from 1969 that each and every attribute in individual tuples inside of a relation must only consist of a single value and not allow any different multivalued values of the kind supported in databases.

EMPLOYEE1

EmpID	Name	DeptName	Salary
100	Margaret Simpson	Marketing	48,000
140	Allen Beeton	Accounting	52,000
110	Chris Lucero	Info Systems	43,000
190	Lorenzo Davis	Finance	55,000
150	Susan Martin	Marketing	42,000

Figure 7.2.2: Example of Property Two. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

The third property speaks to the uniqueness within the table itself and how each individual row can only have values of data that are unique to the row itself. This ensures that no two rows will be identical.

EmpID	Name	DeptName	Salary	CourseTitle	DateCompleted
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/2015
				Surveys	10/7/2015
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/2015
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/2015
				C++	4/22/2015
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/16/2015
				Java	8/12/2015

Figure 7.2.3: Example of Property Three. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

The fifth property specifically describes how the order does not matter in regard to the columns themselves. In Figure 7.2.3, the order of columns can be arbitrary in the structure itself.

Entities, Relations, and Characteristics

- Each row contains data about an entity (table).
- Each column contains data related to attributes of the entities.

- All entries in a column are of the same kind.
- Each cell of the table holds a single value.
- The order of the columns is not important.
- The order of the rows is not important
- No two rows may be identical.

EmpolyeerNumber	FirsName	Email
10	Judy	[illegible]
20	Lily	[illegible]
30	Sam	[illegible]

Table 7.2.1: Entities Relations Characteristics. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

Table 2 below shows three columns: the buyer's name, the **SKU** (stock keeping unit), and the names of the buyer's college major. The Buyers_Name may be associated with more than one SKU, and they can have multiple Departments.

	Buyers_Name	SKU_Numbers	Department
1	Judy Griffin	Griffin	Payroll
2	Lily Murphy	Murphy	Security
3	Sam Moore	Moore	Accounting

Table 7.2.2: Buyer and SKU_Numbers. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

In the relational model of Databases:

- A **candidate key** is an element that controls all the other columns in a relation. The SKU_DATA relation has two candidate keys: SKU and SKU_Description. Buyer is an element, but it is not a candidate key because it controls only Department.
- **A primary key** is a specific choice of a minimal set of attributes (Columns) that is a single row of a table in a database relation. A table has only one primary key. The primary key can have one column, or it can be a composite. The definition of a table is given as **(SKU, SKU_Description, Department, Buyer)**, where **SKU** is the primary key of SKU_DATA and the (**OrdrNumber, SKU**) is the primary key of ORDER_ITEM.
- A **surrogate key** is an artificial column that is added to a table to serve as the primary key when the primary key is large. RENTAL_PROPERTY is the relationship below.
- A **foreign key** is a column that is the primary key of a table. The term occurs because it is a key of a **table *foreign*** to the one in which it appears **as the primary key**. In the below two tables, COMPANY.CompanyName is the primary key of COMPANY, and EMPLOYEE.Company is a foreign key. In this text, we will show ***foreign keys in italics***:
- **Anomalies** are problems that can occur in poorly planned database systems, unnormalised **databases** where the majority of the data are stored in one table **(a flat-file database)**. An **Insertion Anomaly** may be such that it is not possible to add a required piece of data unless another piece is unavailable data that is removed. When we delete one row, the structure of this table forces to lose facts about two different things: a machine and a repair. This condition is called a **deletion anomaly**.

7.3: Define first (1NF), second (2NF), and third normal (3NF) form

Normalization is a technique in database design used to organize tables in a way that reduces redundancy and dependency of data. The use of this technique splits first into a larger table then into smaller tables, and links all by using entity relationships. The designer of the relational model, Edgar Codd, came up with this theory of normalization that introduces the First Normal (1NF) form and then continues to extend the theory into a second (2NF) and third normal form (3NF). The main idea with this theory is that a table is about a specific topic and only supporting topics are included, which minimizes duplicate data, avoids data modification issues, and simplifies queries.

The **first normal form (1NF)**, like all normal forms, has specific requirements to be validated as 1NF. A table, as below, must be two dimensional with rows and columns, and each row must contain data that relates to something. Each column must contain data for a single attribute of the thing it's describing, each cell of the table must have only a single value, entries in any column must have the same type of data (Ex. If the entry in one row of a column contains hair color, all the other rows must contain hair color as well), each column must have a unique name, all rows should be uniquely identified (has to have some unique ID), and the order of the columns and rows must have no significance. Tables in first normal form are subject to deletion and insertion anomalies; they may prove useful in some applications but can be unreliable in others.

Customer_ID	Customer Name	Item	Price	Supplier	Supplier Phone
allen_d	Allen Davidson	Xbox One	250	Microsoft	1-800-BUY-XBOX
cameron123	Cameron Cherry	PlayStation 4	300	Sony	1-800-BUY-SONY
trinity9	Trinity Wilson	PS Vita	200	Sony	1-800-BUY-SONY

Figure 7.3.1: 1st Normal Form ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

Second normal form (2NF) must have all attributes or non-key columns dependent on the key. For example, if the data is based on making an order as shown in Figure 1, the price of the item isn't determined by the primary key or unique identifier of the customer, so this would break the second normal form. In this case, the table can be split into two, with one table having the primary key as Customer_ID and all of their personal information and the second being a table with the primary key as the item name and the supplier information and prices. All of this information in each table is now correctly dependent on the primary key.

Customer_ID	Customer Name
allen_d	Allen Davidson
cameron123	Cameron Cherry
trinity9	Trinity Wilson

Item	Supplier	Supplier Phone	Price
Xbox One	Microsoft	1-800-BUY-XBOX	250
PlayStation 4	Sony	1-800-BUY-SONY	300
PS Vita	Sony	1-800-BUY-SONY	200

Figure 7.3.2: 2nd Normal Form. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

Since the second normal form has two separate tables, a junction table is necessary where both keys are represented in the same table to show who bought what. This can also be used as a compound key where two primary keys are conjoined to represent each other.

Customer_ID	Item
allen_d	Xbox One
cameron123	PlayStation 4
trinity9	PS Vita

Figure 7.3.3: Junction Table. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

In the **third normal form (3NF),** all columns can be determined only by the key in the table and no other column. Having this type of normalization gets rid of redundancy and is especially vulnerable to some types of modification anomalies. Take a look at Figure 7.3.2 where the items table has duplicate supplier phone numbers and supplier names. If Sony were to have more products under their item key, the supplier phone number would keep repeating because it's directly related to the supplier name and will always be that same phone number.

Item	Supplier	Supplier Phone	Price
Xbox One	Microsoft	1-800-BUY-XBOX	250
PlayStation 4	Sony	1-800-BUY-SONY	300
PS Vita	Sony	1-800-BUY-SONY	200

Figure 7.3.4: Split this table. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

Separating these two tables into individual representations makes all of the columns dependent on only the key in the table and no other column, making the third normal form valid.

Item	Supplier	Price
Xbox One	Microsoft	250
PlayStation 4	Sony	300
PS Vita	Sony	200

Supplier	Supplier Phone
Microsoft	1-800-BUY-XBOX
Sony	1-800-BUY-SONY

Figure 7.3.1: 3rd Normal Form. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

- Some researchers discovered that anomalies could occur, and the conditions for **Boyce-Codd Normal Form (BCNF)** were identified. These normal forms are **well-defined so the relation in BCNF is in 3NF**, a relation in 3NF is in 2NF, and a relation in 2NF is in 1NF can occur.
- Therefore, the normal forms 2NF through BCNF concern anomalies that happen from functional dependencies. They led to the definition of **fourth normal form (4NF)** and **fifth normal form (5NF)**.
- Table 7.3.1 shows other anomalies which occurred because of another kind of dependency called a **multivalued dependency.** Those anomalies could be eliminated by placing each multivalued dependency in a relation of its own, a condition known as 4NF.

Anomaly	Norma Forms	Design Principles
Functional Dependencies	1NF, 2NF, 3NF, BCNF	BCNF: Design tables so that every determinant is a candidate key.
Multivalued Dependencies	4NF	4NF: Move each multivalued dependency to a table of its own.
Data Constrained	5NF, DK/NF	DK/NF: Make every constraint a logical consequence of candidate keys and domains.

Table 7.3.1: Form of Normalizations Theory . ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

The **fifth normal form (5NF),** also known as **Project-Join Normal Form (PJ/NF)** can lead to an anomaly where a table can be **split apart but not correctly joined back together**. However, the conditions this condition is rather complex, and generally, if a relation is in 4NF it is in 5NF. For more information about 5NF, consult this Wikipedia article.

7.4: State Two Properties of Candidate Keys

A candidate key is a conglomerate of attributes that identify a database record in a unique way without referencing any other key data from the database. A table may contain one or more candidates and one of those candidate keys has to be referred to as the primary key. The absolute requirement for a table to have is the primary key, but the maximum number of **candidate keys** is unlimited by any constraints.

The naming of a candidate key is coming from a super key with no redundant attribute and they are respectively selected from the set of super keys. Another name for **candidate key** could be **minimal super key.**

All candidate keys have common properties. One of the most important candidate key properties is that the attribute that is used for identification must remain the same for its lifetime. Another basic but important property of candidate keys is that the value of the attribute cannot be a null value. It's a form of identification; therefore, if the value has a null value, there is no way a query can be applied to this row and find the desired information. Each candidate key must contain minimum fields to ensure uniqueness.

StudID	Roll No	FirstName	LastName	Email
1	11	Ben	RandomA	abc@gmail.com
2	12	David	RandomB	xyz@gmail.com
3	13	Cameron	RandomC	mnoprq@gmail.com

Table 7.4.1: Example Identifies Each Property in a Unique and Standard Way. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

This example identifies each property in a unique and standard way. The studID, Roll Number and Email are unique attributes, therefore all three of them are candidate keys. Most importantly studID is a candidate key and also primary key. To uniquely identify each student, a query can use the **primary key**, which also serves as a candidate key. One of the other options would be to identify it by Roll Number or by the email, as only one email and Roll Number can exist.

Other examples of important candidate keys : Social Security Number, International Standard Book Numbers, Bank account numbers, Serial Numbers, Driver License Numbers, National Provider ID, Phone numbers and so on.

7.5: Concise Summary

An Entity Relationship (ER) model is a diagram that gives a good overview of the design of a database system with the aim of making it easier to be understood at a technical level. This model can be mapped to a relational schema which shows the relationship between its members. An ER diagram consists of multiple components: the entity, entity type, entity set, and attributes.

The entity of the ER diagram is an object or concept about which you want to store information. This component is the overall subject of the entity set, whilst the entity set is within the entity which holds the attributes of the entity's type. For example, in an entity called "Car", an entity set would consist of attributes "Car ID", "Car color", "Car model" and "Car brand". To explain this better, here is a visual representation.

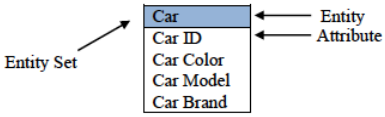


Figure 7.5.1: ERD Component. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

The next important step in understanding ER mapping is attributes. These components are properties which define the entity type. There are many examples to describe this, including Name, DOB, Address, etc. The attribute that uniquely identifies each entity is called a **key attribute**, usually identifying each entity with a unique number. In the example above, Car ID would be the key attribute which can also be called "Primary Key" or PK. Car ID is the key attribute because every Car ID is unique and can identify each entity separately without duplicates. For more complex designs, there are multiple types of attributes: Composite attributes (compose of two or many combined attributes), Multivalued Attributes (consists of more than one value) and Derived attributes (attributes derived from other attributes).

A weak entity is an entity type for which a key attribute can't be defined. A company can store information of dependents (Parents, Children, Spouse) of an Employee but the dependents don't exist without employee.

A relational model for database management system is an approach to managing data using a table structure that is consistent with specific logic and properties. In database systems, there exist relations in a two-dimensional table of data. This typically has multiple named columns and an unknown quantity of rows for data to be entered. There are specific properties that define relations so that they can be easily identified by a user. There are six properties of relations:

1. Each relation (or table) in a database has a unique name.
2. Each entry at the intersection of each row and column is atomic (or single valued). (There can be only one value associated with each attribute on a specific row of a table; no multivalued attributes are allowed in a relation.)
3. Each row is unique; no two rows in a relation can be identical.
4. Each attribute (or column) within a table has a unique name.
5. The sequence of columns (left to right) is insignificant. The order of the columns in a relation can be changed without changing the meaning or use of the relation.
6. The sequence of rows (top to bottom) is insignificant. As with columns, the order of the rows of a relation may be changed or stored in any sequence.

As stated previously, these relational tables use a certain structure that is consistent with specific logic and properties: this is called **normalization**. This process is a technique in database design that is used to organize tables in a manner that reduces redundancy and dependency of data. The use of this technique divides larger tables into smaller tables and links them using relationships. The creator of the relational model, Edgar Codd, came up with the theory of normalization with the introduction of the First Normal Form and then continued to extend this theory into a second and third normal form.

The first normal form, like all normal forms, has specific requirements to be validated as first normal form. A table in first normal form must be two dimensional with rows and columns, each row must contain data that pertains to something, each column must contain data for a single attribute of the thing it's describing, each cell of the table must have only a single value, entries in any column has to have the same type of data (Ex. If the entry in one row of a column contains hair color, all the other rows must contain hair color as well), each column must have a unique name, all rows should be uniquely identified (has to have some unique

ID), and the order of the columns and rows has no significance. Tables in first normal form are subject to deletion and insertion anomalies and may prove useful in some applications but can be unreliable in others.

Customer_ID	Customer Name	Item	Price	Supplier	Supplier Phone
allen_d	Allen Davidson	Xbox One	250	Microsoft	1-800-BUY-XBOX
cameron123	Cameron Cherry	PlayStation 4	300	Sony	1-800-BUY-SONY
trinity9	Trinity Wilson	PS Vita	200	Sony	1-800-BUY-SONY

Figure 7.5.2: 1st Normal Form. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

Second Normal Form

Must have all attributes or non-key columns dependent on the key.

Customer_ID	Customer Name
allen_d	Allen Davidson
cameron123	Cameron Cherry
trinity9	Trinity Wilson

Item	Supplier	Supplier Phone	Price
Xbox One	Microsoft	1-800-BUY-XBOX	250
PlayStation 4	Sony	1-800-BUY-SONY	300
PS Vita	Sony	1-800-BUY-SONY	200

Figure 7.5.3: 2nd Normal From. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

Third Normal Form

All columns can be determined only by the key in the table and no other column.

Customer_ID	Customer Name
allen_d	Allen Davidson
cameron123	Cameron Cherry
trinity9	Trinity Wilson

Supplier	Supplier Phone
Microsoft	1-800-BUY-XBOX
Sony	1-800-BUY-SONY

Item	Supplier	Price
Xbox One	Microsoft	250
PlayStation 4	Sony	300
PS Vita	Sony	200

Figure 7.5.4: 3rd Normal From. ("Module_3: Mapping ER to Schema, Normalization" by Dr Sarah North, Affordable Learning Georgia is licensed under CC BY 4.0)

Having this type of normalization gets rid of redundancy and is especially vulnerable to some types of modification anomalies. If Sony were to have more products under their item key, the supplier phone number would keep repeating because it's directly related to the supplier name and will always be that same phone number. Separating these two tables into individual representations makes all of the columns dependent on only the key in the table and no other

column, making the third normal form valid.

Candidate keys are a collective of attributes that identify a database record in a unique way without referencing any other key data from the database. A table may contain one or more candidates and one of those candidate keys has to be referred to as the primary key. The absolute requirement for a table to have is the primary key, but the maximum number of candidate keys is unlimited by any constraints.

7.6: Extended Resources

1. A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

   A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.

   Tutorials Points Lesson on Schemas

2. This video shows converting ER Diagrams to Schemas | SQL - Discuss the role of designing databases in the analysis and design of an information system. Learn how to transform an entity-relationship (ER) Diagram into an equivalent set of well-structured relations

   A YouTube video on Converting ER Diagrams into Schemas

3. Normalization? 1NF, 2NF, 3NF, BCNF Database Example - The inventor of the relational model Edgar Codd proposed the theory of normalization with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form. Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form.

   From Guru 99 a good lesson about Database Normalization

4. Video on Normalizations has three steps: 1nf, 2nf, and 3nf. These stand for first normal form, second normal form, and third normal form. Each step has rules on what is allowed or not allowed in our database design. Each normal form gets progressively more strict.

   YouTube video on Normalizations

5. Video on Multivalued dependencies According to database theory, a multivalued dependency is a full constraint between two sets of attributes in a relation. In contrast to the functional dependency, the multivalued dependency requires that certain tuples be present in a relation.

   Wiki page on Multivalued Dependency : https://en.wikipedia.org/wiki/Multivalued_dependency

   A 2 part YouTube video set on Multivaried Dependency:

   - Part_1
   - Part_2

7.7: References

Database Normalization (Explained in Simple English). (2020, February 19). Retrieved from the link provided.

Taylor, A. G. (n.d.). SQL First, Second and Third Normal Forms. Retrieved from the link provided.

Normalization - 1NF, 2NF, 3NF and 4NF. (2015). Retrieved from the link provided.

Mapping from ER Model to relational Model (2020, February 20). Retrieved from the link provided.

Introduction of ER Model (2020, February 20). Retrieved from the link provided.

Hoffer, Jeffrey A, V Ramesh, and Heikki Topi. *Modern Database Management*. , 2011. Print.

CHAPTER OVERVIEW

8: SQL and ER

8.1: Introduction and Background

In the world of today, record keeping is used amongst all types of people and entities. Whether for individuals, businesses, or governments, record keeping is used to organize, store, and change data. One of the main tools today that allows record keeping to be more efficient and practical is SQL.

SQL, or Standard Query Language, is a type of language used to store, manipulate, and retrieve data in and from a database. SQL has its origins in 1970 when Dr. Edgar F. "Ted" Codd created the relational model of database management. SQL itself would appear 4 years later in 1974 when it was created at the IBM Research Laboratory in San Jose, California based on Codd's idea of a relational database. In 1978, a product called System/R was created by IBM to further develop these concepts. In 1986, the first standard prototype of SQL was created by IBM, and in that same year, SQL was considered the official standard by the American National Standards Institute (SQL – Overview, 2020).

Unlike Java, Python, or C, SQL is not a procedural language. Another contrast is that SQL is a 4th-generation language, meaning it is more similar to regular human language and speech and is able to be understood by an untrained person unlike Java, C, Python, or others considered 3rd generation languages (Williams, 2018).

Since SQL allows for easier use, especially when it comes to some of the large organizations that use it. Benefits include reduced training costs and increased productivity, because workers can focus on a single language, making it easier to train new employees. Workers are able to learn SQL proficiently, allowing them to increase their productivity and maintain existing programs. The use of SQL on multiple machines in a company also allows for application portability. By using SQL, application longevity is also improved, as standard languages do not go through frequent, large changes. This means that rewriting of older applications is not common with the use of SQL. Another benefit of using SQL is cross-system communication, which allows for the managing of data through multiple applications. These benefits mainly pertain to the use of SQL by corporations. Due to the fact that SQL can be used by almost anyone and is efficient in record keeping, it is known as one of the best (if not *the* best) language in its category (Ramesh, 2011).

The first ANSI SQL standards were published in 1986 and have been updated every few years after that. The updated ANSI SQL standards of 1992 are known to have greatly revised its structure. The structure now features three levels: Entry, Intermediate, and Full. These revisions have made SQL what it is today, constantly improving to better perform at the top level (Ramesh, 2011).

SQL has many major components within its process that enable it to work or process data and execute tasks:

- • Query Dispatcher
- • Optimization Engines
- • Classic Query Engine
- • SQL Query Engine

With these components, the way a user interacts with SQL is by commands. These commands allow the user to execute a multitude of actions, including Create, Alter, Drop, Select, Insert, Update, Delete, Grant, and Revoke. Commands can be modified by clauses, which set the conditions for a command. These clauses include Where, From, and Using. These commands and clauses are just one example of how SQL can be understood by an average person who has little experience in programming (SQL – Overview, 2020).

The highest ranking and/or most popular database management systems today are Oracle, Microsoft SQL Server, MySQL, MongoDB, and PostgreSQL (2019 Database Trends - SQL vs. NoSQL, Top Databases, Single vs. Multiple Database Use., 2019). Oracle is ranked the highest out of all database systems because of its level of functionality, portability, performance, recovery, speed, multiple database support, and reliability (Khamlichi, 2018). Many of these languages are capable of linking or syncing with other types of software and languages like Java. These attributes, among others, help make SQL the leading data organization and collection language in the world.

8.2: Define a Database Using SQL Data Definition Language

It is common knowledge to most who are familiar in the coding world that SQL is used universally to define databases and perform certain actions to interact with the content in those databases. Of these, there exist 4 major categories: Data Definition Languages, Data Query Languages, Data Manipulation Languages, and Data Control Languages.

A Data Definition Language, also known as DDL, is what consists of the "SQL commands that are used to define the database schema" (Varshini, D., 2019, August 26). DDL is also used to modify the content within the database: the database objects as well as the schemas. There are many different commands, including CREATE, DROP, ALTER, TRUNCATE, COMMENT, and RENAME. Each of these commands has its own unique utility and function when implemented as a SQL query. For instance, CREATE and DROP are used to simply create and delete database tables. ALTER can be used to add new columns to a table. So for example, if we had the table below:

EmployeeName	EmployeeID	Position
John	123	Engineer
Jeff	456	Accountant
Jason	789	QA

Figure 8.2.1: Employee (Kashefi 2020)

If we wanted to add each employee's salary, we could use the command:

```
ALTER TABLE Employee
ADD Salary
```

Which would give us this table:

EmployeeName	EmployeeID	Position	Salary
John	123	Engineer	
Jeff	456	Accountant	
Jason	789	QA	

Figure 8.2.2: Employee (Kashefi 2020)

And because the salary column was not filled, we are left with a new, blank column. If we wanted to clear all the information but leave the table and columns, we could use TRUNCATE:

```
TRUNCATE TABLE Employee
```

Which would leave us with this table:

EmployeeName	EmployeeID	Position	Salary

Figure 8.2.3: Employee (Kashefi 2020)

The next category used to define a database with SQL is Data Query Language, or DQL. This category of language is actually used to perform manipulations to the data within schemas rather than the schemas themselves as mentioned above. The primary example of DQL is the SELECT command which selects a set of data which you want to perform your operation on and does this by retrieving the data from the database (Varshini, D., 2019, August 26). It is also worth noting that the primary purpose of DQL is to get a schema relation based on the query submitted. More information on DQL, its commands, and its uses can be found in Section 5.3.

DML, or Data Manipulation Language, is the next major category, which deals with data present in the database. This is different from DQL, because DQL only takes into consideration data from the schema object. It is also worth noting that this category includes most of the commonly used SQL statements such as INSERT, DELETE, and UPDATE. These allow users to actually modify data in small increments, rather than make sweeping changes that would be seen in DDL. For example, let's look back at the table in Figure Figure 8.2.3.

Currently this table is empty, but what if we wanted to add a new employee who has a name, ID, position, and salary? We would use this command:

```
INSERT INTO Employee(EmployeeName,EmployeeID,Position,Salary)
VALUES (“Jonah”, 159, Janitor, 1000000)
```

Which would leave us with this table:

EmployeeName	EmployeeID	Position	Salary
Jonah	159	Janitor	1000000

Figure 8.2.4: Employee (Kashefi 2020)

Next is DCL, which stands for Data Control Language. This category is mostly concerned with the rights and permissions of other users on the network database and has commands which manage these. These commands such as GRANT and REVOKE simply grant or revoke permissions to a specified user on the database system. This is important for both data security and data integrity. It ensures that data can only be seen by people who must see it, and it also ensures data cannot be changed by someone who does not have an understanding of what is happening in the database.

The last category we will discuss is TCL. TCL was not mentioned above, as there is debate on whether or not TCL is actually something that should be considered a major category within the data definition namespace. However, to be thorough we will touch on it briefly here. It essentially deals with transactions of data within the database. It does this through the use of commands such as COMMIT, ROLLBACK, SAVEPOINT, and SET TRANSACTION. All of these perform a certain action by specified keyword on a transaction. This can give users the ability to control versions of their databases. This can become important in the case of incorrect information or even a corrupted database.

8.3: Write a single table query in SQL

In data science, a query is a call for a specific set, group, or combination of data. In order to query a database, we need to use a language the database can understand (Gibbs, n.d.). Tables, on the other hand, are objects within a database that include some, or all, of the data from the database. This data is organized into a grouping made up of rows and columns. The rows each represent a unique item from the database records. The columns each represent different attributes that the item contains (Cai et al., n.d.).

StudentName	StudentID	ClassID	Grade
John	123	987	90
Jeff	456	654	83
Jason	789	321	97

Figure 8.3.1: Student (Kashefi 2020)

Above is an example table named Student. The first column represents the student's name. This student name is what we will use to name the unique items in the rows. For example, John is the name of a student. Every other column of that row contains data that describes him. No other row will have the exact same data. This makes his, and every other row unique.

In the example table, the columns represent attributes in the data. Every student has a name, a student ID, a class ID, and a grade.

In order to query from this table, we must use specific SQL commands. We must use the SELECT command to choose the columns in the table we are querying from. To select the correct table, the FROM command must be used. In order to ask for the entire table as a query, therefore:

```
SELECT * FROM Student
```

The asterisk is a shorthand way to ask for every column. This shorthand is also used in other database and programming languages. This command would be read in plain English as "select all columns from the student table." This would return everything from the above table, as the table is named *Student* and all of the information is in one column or another.

Now what if we just wanted to know the name and grade for each student? This is how it would look:

```
SELECT StudentName, Grade FROM Student
```

In this example, we used SELECT to choose columns. Instead of an asterisk used for "all columns," we use the specific column names "StudentName" and "Grade," which are intuitively the name and grade of the student, and the columns are separated by a comma. The FROM command underneath once again chooses the correct table. In plain English, this is "select the columns student name and grade from the student table." This would be the returned table:

StudentName	Grade
John	90
Jeff	83
Jason	97

Figure 8.3.2: Student (Kashefi 2020)

This query can be done with any combination of columns in the table, as long as the same formatting is used.

The last staple command of SQL is WHERE. This command will only return a row if the information in one of the specific columns fits a specific condition of the query(“Learn SQL:

Queries Reference Guide,” n.d.). To allow a better understanding, here is an example:

```
SELECT StudentName, Grade
FROM Student
WHERE Grade < 95
```

For this example, we have the same queried table as above, except that any row with a grade above 95 is removed. In plain English, this would be “select the columns student name and grade from the student table if the student’s grade is below 95.” This would be the final queried table:

StudentName	Grade
John	90
Jeff	83

Figure 8.3.1: Student (Kashefi 2020)

This is just a small part of all the possible ways to query in SQL, but they are the main ones that can allow basic table queries to be practiced.

8.4: Establish referential integrity using SQL

Referential integrity is the accuracy and consistency of data in a relationship. Referential integrity is a subset of data integrity, which is concerned with the accuracy and consistency of data as whole in a database. When relationships occur, data is linked between two or more tables. A primary key is the key or specific column in a parent table, and a foreign key is the key in a child table that references the primary key. Referential integrity requires that a foreign key references a primary key (Ian, 2016).

The tables below will show how referential integrity works within a database management system:

Child Table

Department	
Employee ID	Department
6789	Marketing
5632	R&D

Parent Table

Employee		
Employee ID	Age	Salary
6789	25	56000
5632	29	83000

Figure 8.4.1: Parent / Child Tables. (Copyright; author via source)

As you can see in the tables above, Employee is the parent table and Department is the child table. The primary key is the Employee ID and is referenced in the child table, therefore making it a foreign key in this table. The relationship is denoted by a curved line connecting the columns in the two different tables. As a result of this relationship a user will be prevented from:

- Adding information in the child table if the same information is not also in the parent table
- Changing data in the primary table that result in parentless keys in the child table (orphaned)
- Deleting records from the parent table if they exist in the child table

A lack of Referential integrity can result in records being lost and/or inaccurate or confusing. This can have serious and negative effects for entities that make use of database systems (Ian, 2016).

To ensure or establish referential integrity within SQL (Microsoft SQL Servers), it is necessary to establish variables (columns) as a primary key when creating tables. After creating the child table, expand the database in project explorer and look at “table.” After finding the table tree, expand it and right click on “keys.” After right clicking, it will provide options, one of them being a “new foreign key.” Click “new foreign key” and expand “Tables and columns specification.” After expanding, click on the 3 dots on the right side of the box. This will allow the user to enter a primary or parent able for foreign keys along with the specific column. After doing this, the tables will be linked to one another. The user will only be able to perform tasks that are within the constraints of referential integrity.

8.5: Concise Summary

SQL, a non-procedural query language, is a major database management system that helps organize and log data. Commands and clauses like CREATE, SELECT, etc. are what makes SQL work and gives it functionality. Using these commands and clauses, it is possible to create databases, create tables, edit data within tables, add columns and rows, create keys and so on.

The intuitive words and language that make up these tables allow people who may not be skilled in computer science to be able to use the system. Some of the commands are powerful in another sense as well: they allow the user to quickly scan through all of the given information and return only a specific set based on any number of parameters that the user decides are necessary.

Using other commands, data can be linked between two tables, using characteristics such as Referential Integrity, primary keys, and foreign keys. Primary keys uniquely identify records in tables, while the foreign keys reference the primary key. The linking that results from this is called a relationship. Referential integrity helps prevent the duplication of already existing data and it helps make sure that necessary records aren't accidently deleted. Referential integrity gives SQL a way to make sure that the data that is in the table is correct and updated due to constraints on what can be changed, and who it can be changed by. The many features of SQL enable it to be the most popular query language in the world.

Furthermore, this module discusses the process of transforming a data model into a database design as various aspects of data models and designs, how they relate to each other, and how they relate to the systems analysis and design process in general and to the systems development life cycle (SDLC) in particular. Transforming a data model into a database design requires three major tasks: replacing each entity with a table and each attribute with a column, representing relationships and maximum cardinality by placing foreign keys, and representing minimum cardinality by defining actions to constrain activities on values of primary and foreign keys.

Four uses for ID-dependent entities are N:M relationships, association relationships, multivalued attributes, and archetype/instance relationships. An association relationship differs from an intersection table because the ID-dependent entity has no key data. In all ID-dependent entities, the key of the parent is already in the child

8.6: Extended Resources

1. This is Microsoft's SQL documentation. This resource helped us gain information on different commands and queries and when they should be used. The information is detailed and in depth, albeit dense to read in some parts.
2. This W3Schools resource explains what referential integrity is. It is also a good source in showing examples of joining with referential integrity.
3. This Code Academy page lists every query command that exists in SQL. It also gives examples of the use of each one.
4. This W3Schools resource gives examples of the usage of multiple queries and operations to clear any confusion that may come from a specific one or even just the syntax.
5. This website allows for users to practice many SQL functions online. It also has videos for users to look at to help them with their exercises.
6. This website allows users to learn about SQL and how it works. It also serves as a site where users can practice SQL and look at tutorials to help with their practice.
7. This website allows users to learn the fundamentals of SQL and database concepts.
8. This source lists 18 different sites that can help users learn about and practice SQL.

8.7: References

SQL - Overview. (n.d.).(2020) Retrieved February 22, 2020, from https://www.tutorialspoint.com/sql/sql-overview.htm

SEQUEL (SQL) is Developed1974. (2020). Retrieved February 22, 2020, from http://www.historyofinformation.com/detail.php?id=910

SQL clauses. (n.d.). Retrieved February 22, 2020, from https://docs.oracle.com/javadb/10.8....efclauses.html

2019 Database Trends - SQL vs. NoSQL, Top Databases, Single vs. Multiple Database Use. (2019, March 4). Retrieved February 23, 2020, from https://scalegrid.io/blog/2019-datab...-database-use/

Khamlichi, M. (2018, February 6). Why oracle database is the best DBMS solution. Retrieved February 23, 2020, from https://www.osradar.com/oracle-datab...-dbms-solution /

Ian. (2016. May 28). What is Referential Integrity? Retrieved February 24, 2020, from https://database.guide/what-is-referential-integrity/

Borsecnik, J., Guyer, C., Miliner, G., Hamilton, B., Masha, Macauley, E., … Woodard, J. (n.d.). Lesson 1: Create and query database objects. Retrieved February 23, 2020, from https://github.com/MicrosoftDocs/sql...ase-objects.md

Guyer, C., Miliner, G., Ray, M., Hamilton, B., Cai, S., Kumar, S., & Stein, S. (n.d.). Tables. Retrieved February 23, 2020, from https://github.com/MicrosoftDocs/sql...s/tables/table s.md

What is a Query in SQL? (n.d.). Retrieved February 22, 2020, from https://study.com/academy/lesson/wha...ry-in-sql.html

Varshini, D., & GeeksForGeeks. (2019, August 26). SQL: DDL, DQL, DML, DCL and TCL Commands. Retrieved from https://www.geeksforgeeks.org/sql-dd...-tcl-commands/ Learn SQL: Queries Reference Guide. (n.d.). Retrieved March 23, 2020, from https://www.codecademy.com/learn/lea...ies/cheatsheet

Hoffer, J. A., Ramesh, V., & Topi, H. (2011). *Modern database management*. Upper Saddle River, NJ: Prentice Hall.

CHAPTER OVERVIEW

9: SQL - Structured Query Language

9.1: What is SQL?

Structured Query Language (SQL) is a database language designed for managing data held in a relational database management system. SQL was initially developed by IBM in the early 1970s (Date 1986). The initial version, called *SEQUEL* (Structured English Query Language), was designed to manipulate and retrieve data stored in IBM's quasi-relational database management system, System R. Then in the late 1970s, Relational Software Inc., which is now Oracle Corporation, introduced the first commercially available implementation of SQL, Oracle V2 for VAX computers.

Many of the currently available relational DBMSs, such as Oracle Database, Microsoft SQL Server (shown in Figure 15.1), MySQL, IBM DB2, IBM Informix and Microsoft Access, use SQL.

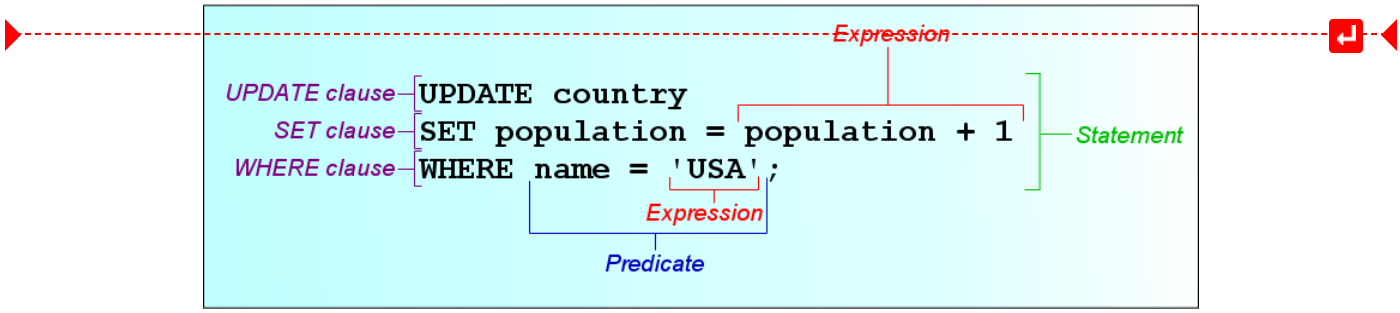


Figure 9.1.1: Example of an SQL Statement. ("Sql statement anatomy" by SqlPac is licensed under CC BY-SA 3.0)

In a DBMS, the SQL database language is used to:

- Create the database and table structures
- Perform basic data management chores (add, delete and modify)
- Perform complex queries to transform raw data into useful information

In this chapter, we will focus on using SQL to create the database and table structures, mainly using SQL as a data definition language (*DDL*). In Chapter 16, we will use SQL as a data manipulation language (*DML*) to insert, delete, select and update data within the database tables.

9.2: CREATE a Database

How to Create a Database

The major SQL DDL statements are CREATE DATABASE and CREATE/DROP/ALTER TABLE. The SQL statement CREATE is used to create the database and table structures.

SQL Statement: CREATE DATABASE SW

A new database named **SW** is created by the SQL statement CREATE DATABASE SW. Once the database is created, the next step is to create the database tables.

The general format for the CREATE TABLE command is:

```
CREATE TABLE <tablename>
(
ColumnName, Datatype, Optional Column Constraint,
ColumnName, Datatype, Optional Column Constraint,
Optional table Constraints
);
```

Tablename is the name of the database table such as **Employee**. Each field in the CREATE TABLE has three parts (see above):

1. ColumnName
2. Data type
3. Optional Column Constraint

ColumnName

The ColumnName must be unique within the table. Some examples of ColumnNames are FirstName and LastName.

Data Type

The data type, as described below, must be a system data type or a user-defined data type. Many of the data types have a size such as CHAR(35) or Numeric(8,2).

Bit –Integer data with either a 1 or 0 value

Int –Integer (whole number) data from -2^31 (-2,147,483,648) through 2^31 – 1 (2,147,483,647)

Smallint –Integer data from 2^15 (-32,768) through 2^15 – 1 (32,767)

Tinyint –Integer data from 0 through 255

Decimal –Fixed precision and scale numeric data from -10^38 -1 through 10^38

Numeric –A synonym for **decimal**

Timestamp –A database-wide unique number

Uniqueidentifier –A globally unique identifier (GUID)

Money – Monetary data values from -2^63 (-922,337,203,685,477.5808) through 2^63 – 1 (+922,337,203,685,477.5807), with accuracy to one-ten-thousandth of a monetary unit

Smallmoney –Monetary data values from -214,748.3648 through +214,748.3647, with accuracy to one-ten-thousandth of a monetary unit

Float –Floating precision number data from -1.79E + 308 through 1.79E + 308

Real –Floating precision number data from -3.40E + 38 through 3.40E + 38

Datetime –Date and time data from January 1, 1753, to December 31, 9999, with an accuracy of one-three-hundredths of a second, or 3.33 milliseconds

Smalldatetime –Date and time data from January 1, 1900, through June 6, 2079, with an accuracy of one minute

Char –Fixed-length non-Unicode character data with a maximum length of 8,000 characters

Varchar –Variable-length non-Unicode data with a maximum of 8,000 characters

Text –Variable-length non-Unicode data with a maximum length of 2^31 – 1 (2,147,483,647) characters

Binary –Fixed-length binary data with a maximum length of 8,000 bytes

Varbinary –Variable-length binary data with a maximum length of 8,000 bytes

Image – Variable-length binary data with a maximum length of 2^31 – 1 (2,147,483,647) bytes

---

9.3: Optional Column Constraints

Optional Column Constraints

The Optional ColumnConstraints are NULL, NOT NULL, UNIQUE, PRIMARY KEY and DEFAULT, used to initialize a value for a new record. The column constraint NULL indicates that null values are allowed, which means that a row can be created without a value for this column. The column constraint NOT NULL indicates that a value must be supplied when a new row is created.

To illustrate, we will use the SQL statement CREATE TABLE EMPLOYEES to create the employees table with 16 attributes or fields.

```
USE SW
CREATE TABLE EMPLOYEES
(
EmployeeNo                    CHAR(10)          NOT NULL          UNIQUE,
DepartmentName                CHAR(30)          NOT NULL          DEFAULT “Huma
FirstName                     CHAR(25)          NOT NULL,
LastName                      CHAR(25)          NOT NULL,
Category                      CHAR(20)          NOT NULL,
HourlyRate                    CURRENCY          NOT NULL,
TimeCard                      LOGICAL           NOT NULL,
HourlySalaried                CHAR(1)           NOT NULL,
EmpType                       CHAR(1)           NOT NULL,
Terminated                    LOGICAL           NOT NULL,
ExemptCode                    CHAR(2)           NOT NULL,
Supervisor                    LOGICAL           NOT NULL,
SupervisorName                CHAR(50)          NOT NULL,
BirthDate                     DATE              NOT NULL,
CollegeDegree                 CHAR(5)           NOT NULL,
CONSTRAINT                    Employee_PK       PRIMARY KEY(EmployeeNo
);
```

The first field is EmployeeNo with a field type of CHAR. For this field, the field length is 10 characters, and the user cannot leave this field empty (NOT NULL).

Similarly, the second field is DepartmentName with a field type CHAR of length 30. After all the table columns are defined, a table constraint, identified by the word CONSTRAINT, is used to create the primary key:

CONSTRAINT EmployeePK PRIMARY KEY(EmployeeNo)

We will discuss the constraint property further later in this chapter.

Likewise, we can create a Department table, a Project table and an Assignment table using the CREATE TABLE SQL DDL command as shown in the below example.

```
USE SW
CREATE TABLE DEPARTMENT
(
   DepartmentName Char(35)  NOT NULL,
   BudgetCode     Char(30)  NOT NULL,
   OfficeNumber   Char(15)  NOT NULL,
   Phone          Char(15)  NOT NULL,
```

```
   CONSTRAINT DEPARTMENT_PK PRIMARY KEY(DepartmentName)
);
```

In this example, a project table is created with seven fields: ProjectID, ProjectName, Department, MaxHours, StartDate, and EndDate.

```
USE SW
CREATE TABLE PROJECT
(
   ProjectID       Int  NOT NULL IDENTITY (1000,100),
   ProjectName     Char(50) NOT NULL,
   Department      Char(35) NOT NULL,
   MaxHours        Numeric(8,2)  NOT NULL DEFAULT 100,
   StartDate       DateTime NULL,
   EndDate         DateTime NULL,
   CONSTRAINT      ASSIGNMENT_PK  PRIMARY KEY(ProjectID)
);
```

In this last example, an assignment table is created with three fields: ProjectID, EmployeeNumber, and HoursWorked. The assignment table is used to record who (EmployeeNumber) and how much time(HoursWorked) an employee worked on the particular project(ProjectID).

```
USE SW
CREATE TABLE ASSIGNMENT
(
   ProjectID       Int  NOT NULL,
   EmployeeNumber  Int  NOT NULL,
   HoursWorked     Numeric(6,2)  NULL,
);
```

9.4: Table Constraints

Table constraints are identified by the CONSTRAINT keyword and can be used to implement various constraints described below.

IDENTITY constraint

We can use the optional column constraint IDENTITY to provide a unique, incremental value for that column. Identity columns are often used with the PRIMARY KEY constraints to serve as the unique row identifier for the table. The IDENTITY property can be assigned to a column with a tinyint, smallint, int, decimal or numeric data type. This constraint:

- Generates sequential numbers
- Does not enforce entity integrity
- Only one column can have the IDENTITY property
- Must be defined as an integer, numeric or decimal data type
- Cannot update a column with the IDENTITY property
- Cannot contain NULL values
- Cannot bind defaults and default constraints to the column

For IDENTITY[(seed, increment)]

- Seed – the initial value of the identity column
- Increment – the value to add to the last increment column

We will use another database example to further illustrate the SQL DDL statements by creating the table tblHotel in this HOTEL database.

```
CREATE TABLE  tblHotel
(
HotelNo                 Int               IDENTITY (1,1),
Name                    Char(50)          NOT NULL,
Address                 Char(50)          NULL,
City                    Char(25)          NULL,
)
```

UNIQUE constraint

The UNIQUE constraint prevents duplicate values from being entered into a column.

- Both PK and UNIQUE constraints are used to enforce entity integrity.
- Multiple UNIQUE constraints can be defined for a table.
- When a UNIQUE constraint is added to an existing table, the existing data is always validated.
- A UNIQUE constraint can be placed on columns that accept nulls. *Only one row can be NULL.*
- A UNIQUE constraint automatically creates a unique index on the selected column.

This is the general syntax for the UNIQUE constraint:

```
[CONSTRAINT constraint_name]
UNIQUE [CLUSTERED | NONCLUSTERED]
(col_name [, col_name2 […, col_name16]])
[ON segment_name]
```

This is an examle using the UNIQUE constraint.

```
CREATE TABLE EMPLOYEES
(
```

```
EmployeeNo                    CHAR(10)          NOT NULL          UNIQUE,
)
```

FOREIGN KEY constraint

The FOREIGN KEY (FK) constraint defines a column, or combination of columns, whose values match the PRIMARY KEY (PK) of another table.

- Values in an FK are automatically updated when the PK values in the associated table are updated/changed.
- FK constraints must reference PK or the UNIQUE constraint of another table.
- The number of columns for FK must be same as PK or UNIQUE constraint.
- If the WITH NOCHECK option is used, the FK constraint will not validate existing data in a table.
- No index is created on the columns that participate in an FK constraint.

This is the general syntax for the FOREIGN KEY constraint:

```
[CONSTRAINT constraint_name]
[FOREIGN KEY (col_name [, col_name2 […, col_name16]])]
REFERENCES [owner.]ref_table [(ref_col [, ref_col2 […, ref_col16]])]
```

In this example, the field HotelNo in the tblRoom table is a FK to the field HotelNo in the tblHotel table shown previously.

```
USE HOTEL
GO
CREATE TABLE  tblRoom
(
HotelNo          Int              NOT NULL ,
RoomNo           Int              NOT NULL,
Type             Char(50)         NULL,
Price            Money            NULL,
PRIMARY KEY (HotelNo, RoomNo),
FOREIGN KEY (HotelNo) REFERENCES tblHotel
)
```

CHECK constraint

The CHECK constraint restricts values that can be entered into a table.

- It can contain search conditions similar to a WHERE clause.
- It can reference columns in the same table.
- The data validation rule for a CHECK constraint must evaluate to a boolean expression.
- It can be defined for a column that has a rule bound to it.

This is the general syntax for the CHECK constraint:

```
[CONSTRAINT constraint_name]
CHECK [NOT FOR REPLICATION] (expression)
```

In this example, the Type field is restricted to have only the types ‘Single’, ‘Double’, ‘Suite’ or ‘Executive’.

```
USE HOTEL
GO
CREATE TABLE  tblRoom
```

```
(
HotelNo           Int               NOT NULL,
RoomNo            Int               NOT NULL,
Type              Char(50)          NULL,
Price             Money             NULL,
PRIMARY KEY (HotelNo, RoomNo),
FOREIGN KEY (HotelNo) REFERENCES tblHotel
CONSTRAINT Valid_Type
CHECK (Type IN (‘Single’, ‘Double’, ‘Suite’, ‘Executive’))
)
```

In this second example, the employee hire date should be before January 1, 2004, or have a salary limit of $300,000.

```
GO
CREATE TABLE SALESREPS
(
Empl_num          Int      Not Null CHECK (Empl_num BETWEEN 101 and 199),
Name              Char (15),
Age               Int               CHECK (Age >= 21),
Quota             Money             CHECK (Quota >= 0.0),
HireDate       DateTime,
CONSTRAINT  QuotaCap                CHECK ((HireDate < “01-01-2004”) OR (Quota <=3000
)
```

DEFAULT constraint

The DEFAULT constraint is used to supply a value that is automatically added for a column if the user does not supply one.

- A column can have only one DEFAULT.
- The DEFAULT constraint cannot be used on columns with a timestamp data type or identity property.
- DEFAULT constraints are automatically bound to a column when they are created.

The general syntax for the DEFAULT constraint is:

```
[CONSTRAINT constraint_name]
DEFAULT {constant_expression | niladic-function | NULL}
[FOR col_name]
```

This example sets the default for the city field to ‘Vancouver’.

```
USE HOTEL
ALTER TABLE tblHotel
Add CONSTRAINT df_city DEFAULT ‘Vancouver’ FOR City
```

9.5: Few Final Tidbits

User Defined Types

User defined types are always based on system-supplied data type. They can enforce data integrity and they allow nulls.

To create a user-defined data type in SQL Server, choose types under “Programmability” in your database. Next, right click and choose ‘New’ –>‘User-defined data type’ or execute the sp_addtype system stored procedure. After this, type:

sp_addtype ssn, ‘varchar(11)’, ‘NOT NULL’

This will add a new user-defined data type called SIN with nine characters.

In this example, the field EmployeeSIN uses the user-defined data type SIN.

```
CREATE TABLE SINTable
(
EmployeeID        INT Primary Key,
EmployeeSIN       SIN,
CONSTRAINT        CheckSIN
CHECK (EmployeeSIN LIKE ‘ [0-9][0-9][0-9] – [0-9][0-9] [0-9] – [0-9][0-9][0-9] ‘)
)
```

ALTER TABLE

You can use ALTER TABLE statements to add and drop constraints.

- ALTER TABLE allows columns to be removed.
- When a constraint is added, all existing data are verified for violations.

In this example, we use the ALTER TABLE statement to the IDENTITY property to a ColumnName field.

```
USE HOTEL
GO
ALTER TABLE  tblHotel
ADD CONSTRAINT unqName UNIQUE (Name)
```

Use the ALTER TABLE statement to add a column with the IDENTITY property such as ALTER TABLE TableName.

```
ADD
ColumnName         int    IDENTITY(seed, increment)
```

DROP TABLE

The DROP TABLE will remove a table from the database. Make sure you have the correct database selected.

```
DROP TABLE tblHotel
```

Executing the above SQL DROP TABLE statement will remove the table tblHotel from the database.

Key Terms

DDL: abbreviation for *data definition language*

DML: abbreviation for *data manipulation language*

SEQUEL: acronym for *Structured English Query Language;* designed to manipulate and retrieve data stored in IBM's quasi-relational database management system, System R

Structured Query Language (SQL): a database language designed for managing data held in a relational database management system

Exercises

1. Using the information for the Chapter 9 exercise, implement the schema using Transact SQL (show SQL statements for each table). Implement the constraints as well.
2. Create the table shown here in SQL Server and show the statements you used.
   Table: Employee

ATTRIBUTE (FIELD) NAME	DATA DECLARATION
EMP_NUM	CHAR(3)
EMP_LNAME	VARCHAR(15)
EMP_FNAME	VARCHAR(15)
EMP_INITIAL	CHAR(1)
EMP_HIREDATE	DATE
JOB_CODE	CHAR(3)

Table 9.5.1: Copy and Paste Caption here. (Copyright; author via source)

3. Having created the table structure in question 2, write the SQL code to enter the rows for the table shown in Table 9.5.1.

	EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
▶	101	News	John	G	08-Nov-00	502
	102	Senior	David	H	12-Jul-89	501
	103	Arbough	June	E	01-Dec-96	500
	104	Ramoras	Anne	K	15-Nov-87	501
	105	Johnson	Alice	K	01-Feb-93	502
	106	Smithfield	William		22-Jun-04	500
	107	Alonzo	Maria	D	10-Oct-93	500
	108	Washington	Ralph	B	22-Aug-91	501
	109	Smith	Larry	W	18-Jul-97	501

Figure 9.5.2: *Employee table with data. by A. Watt.*

Use Figure 9.5.2 to answer questions 4 to 10.

4. Write the SQL code to change the job code to 501 for the person whose personnel number is 107. After you have completed the task, examine the results, and then reset the job code to its original value.
5. Assuming that the data shown in the Employee table have been entered, write the SQL code that lists all attributes for a job code of 502.
6. Write the SQL code to delete the row for the person named William Smithfield, who was hired on June 22, 2004, and whose job code classification is 500. (*Hint*: Use logical operators to include all the information given in this problem.)
7. Add the attributes EMP_PCT and PROJ_NUM to the Employee table. The EMP_PCT is the bonus percentage to be paid to each employee.
8. Using a single command, write the SQL code that will enter the project number (PROJ_NUM) = 18 for all employees whose job classification (JOB_CODE) is 500.

9. Using a single command, write the SQL code that will enter the project number (PROJ_NUM) = 25 for all employees whose job classification (JOB_CODE) is 502 or higher.
10. Write the SQL code that will change the PROJ_NUM to 14 for those employees who were hired before January 1, 1994, and whose job code is at least 501. (You may assume that the table will be restored to its original condition preceding this question.)

Also see *Appendix C: SQL Lab with Solution*

CHAPTER OVERVIEW

10: SQL Data Manipulation Language

10.1: Introduction

The SQL data manipulation language (DML) is used to query and modify database data. In this chapter, we will describe how to use the SELECT, INSERT, UPDATE, and DELETE SQL DML command statements, defined below.

- *SELECT* – to query data in the database
- *INSERT* – to insert data into a table
- *UPDATE* – to update data in a table
- *DELETE* – to delete data from a table

In the SQL DML statement:

- Each clause in a statement should begin on a new line.
- The beginning of each clause should line up with the beginning of other clauses.
- If a clause has several parts, they should appear on separate lines and be indented under the start of the clause to show the relationship.
- Upper case letters are used to represent reserved words.
- Lower case letters are used to represent user-defined words.

10.2: SELECT Statement

The SELECT statement, or command, allows the user to extract data from tables, based on specific criteria. It is processed according to the following sequence:

```
SELECT DISTINCT item(s)
FROM  table(s)
WHERE  predicate
GROUP BY  field(s)
ORDER BY fields
```

We can use the SELECT statement to generate an employee phone list from the Employees table as follows:

```
SELECT  FirstName, LastName, phone
FROM Employees
ORDER BY LastName
```

This action will display employee's last name, first name, and phone number from the Employees table, seen in Table 10.2.1.

Last Name	First Name	Phone Number
Hagans	Jim	604-232-3232
Wong	Bruce	604-244-2322

Table 10.2.1: *Employees table*. (Copyright; author via source)

In this next example, we will use a Publishers table (Table 10.2.2). (You will notice that Canada is mispelled in the *Publisher Country* field for Example Publishing and ABC Publishing. To correct mispelling, use the UPDATE statement to standardize the country field to Canada – see UPDATE statement later in this chapter.)

Publisher Name	Publisher City	Publisher Province	Publisher Country
Acme Publishing	Vancouver	BC	Canada
Example Publishing	Edmonton	AB	Cnada
ABC Publishing	Toronto	ON	Canda

Table 10.2.2: *Publishers table*. (Copyright; author via source)

If you add the publisher's name and city, you would use the SELECT statement followed by the fields name separated by a comma:

```
SELECT PubName, city
FROM Publishers
```

This action will display the publisher's name and city from the Publishers table.

If you just want the publisher's name under the display name city, you would use the SELECT statement with *no comma* separating pub_name and city:

```
SELECT PubName city
FROM Publishers
```

Performing this action will display only the pub_name from the Publishers table with a “city” heading. If you do not include the comma, SQL Server assumes you want a new column name for pub_name.

10.2.1: SELECT with WHERE criteria

Sometimes you might want to focus on a portion of the Publishers table, such as only publishers that are in Vancouver. In this situation, you would use the SELECT statement with the WHERE criterion, i.e., WHERE city = 'Vancouver'.

These first two examples illustrate how to limit record selection with the WHERE criterion using BETWEEN. Each of these examples give the same results for store items with between 20 and 50 items in stock.

Example #1 uses the quantity, *qty BETWEEN 20 and 50*.

```
SELECT StorID, qty, TitleID
FROM Sales
WHERE qty BETWEEN 20 and 50  (includes the 20 and 50)
```

Example #2, on the other hand, uses *qty >=20 and qty <=50* .

```
SELECT StorID, qty, TitleID
FROM Sales
WHERE qty >= 20 and qty  <= 50
```

Example #3 illustrates how to limit record selection with the WHERE criterion using NOT BETWEEN.

```
SELECT StorID, qty, TitleID
FROM Sales
WHERE qty NOT BETWEEN 20 and 50
```

The next two examples show two different ways to limit record selection with the WHERE criterion using IN, with each yielding the same results.

Example #4 shows how to select records using *province=* as part of the WHERE statement.

```
SELECT *
FROM Publishers
WHERE province = 'BC' OR province = 'AB' OR province = 'ON'
```

Example #5 select records using *province IN* as part of the WHERE statement.

```
SELECT *
FROM Publishers
WHERE province IN ('BC', 'AB', 'ON')
```

The final two examples illustrate how NULL and NOT NULL can be used to select records. For these examples, a Books table (not shown) would be used that contains fields called Title, Quantity, and Price (of book). Each publisher has a Books table that lists all of its books.

Example #6 uses NULL.

```
SELECT price, title
FROM Books
WHERE price IS NULL
```

Example #7 uses NOT NULL.

```
SELECT price, title
FROM Books
WHERE price IS NOT NULL
```

10.2.2: Wildcard in LIKE clause

The LIKE keyword selects rows containing fields that match specified portions of character strings. LIKE is used with char, varchar, text, datetime and smalldatetime data. A *wildcard* allows the user to match fields that contain certain letters. For example, the wildcard province = 'N%' would give all provinces that start with the letter 'N'. Table 16.3 shows four ways to specify wildcards in the SELECT statement in regular express format.

%	Any string of zero or more characters
_	Any single character
[ ]	Any single character within the specified range (e.g., [a-f]) or set (e.g., [abcdef])
[^]	Any single character not within the specified range (e.g., [^a – f]) or set (e.g., [^abcdef])

Figure 10.2.2.1: *How to specify wildcards in the SELECT statement*. (Copyright; author via source)

In example #1, LIKE 'Mc%' searches for all last names that begin with the letters "Mc" (e.g., McBadden).

```
SELECT LastName
FROM Employees
WHERE LastName LIKE 'Mc%'
```

For example #2: LIKE '%inger' searches for all last names that end with the letters "inger" (e.g., Ringer, Stringer).

```
SELECT LastName
FROM Employees
WHERE LastName LIKE '%inger'
```

In, example #3: LIKE '%en%' searches for all last names that have the letters "en" (e.g., Bennett, Green, McBadden).

```
SELECT LastName
FROM Employees
WHERE LastName LIKE '%en%'
```

10.2.3: SELECT statement with ORDER BY clause

You use the ORDER BY clause to sort the records in the resulting list. Use *ASC* to sort the results in ascending order and *DESC* to sort the results in descending order.

For example, with ASC:

```
SELECT *
FROM Employees
ORDER BY HireDate ASC
```

And with DESC:

```
SELECT *
FROM Books
ORDER BY type, price DESC
```

10.2.4: SELECT statement with GROUP BY clause

The *GROUP BY* clause is used to create one output row per each group and produces summary values for the selected columns, as shown below.

```
SELECT type
FROM Books
GROUP BY type
```

Here is an example using the above statement.

```
SELECT type AS ‘Type’, MIN(price) AS ‘Minimum Price’
FROM Books
WHERE royalty > 10
GROUP BY type
```

If the SELECT statement includes a WHERE criterion where *price is not null*,

```
SELECT type, price
FROM Books
WHERE price is not null
```

then a statement with the GROUP BY clause would look like this:

```
SELECT type AS ‘Type’, MIN(price) AS ‘Minimum Price’
FROM Books
WHERE price is not null
GROUP BY type
```

Using COUNT with GROUP BY

We can use COUNT to tally how many items are in a container. However, if we want to count different items into separate groups, such as marbles of varying colours, then we would use the COUNT function with the GROUP BY command.

The below SELECT statement illustrates how to count groups of data using the COUNT function with the GROUP BY clause.

```
SELECT COUNT(*)
FROM Books
GROUP BY type
```

Using AVG and SUM with GROUP BY

We can use the AVG function to give us the average of any group, and SUM to give the total.

Example #1 uses the AVG FUNCTION with the GROUP BY type.

```
SELECT AVG(qty)
FROM Books
GROUP BY type
```

Example #2 uses the SUM function with the GROUP BY type.

```
SELECT SUM(qty)
FROM Books
GROUP BY type
```

Example #3 uses both the AVG and SUM functions with the GROUP BY type in the SELECT statement.

```
SELECT ‘Total Sales’ = SUM(qty), ‘Average Sales’ = AVG(qty), stor_id
FROM Sales
GROUP BY StorID ORDER BY  ‘Total Sales’
```

10.2.5: Restricting rows with HAVING

The HAVING clause can be used to restrict rows. It is similar to the WHERE condition except HAVING can include the aggregate function; the WHERE cannot do this.

The HAVING clause behaves like the WHERE clause, but is applicable to groups. In this example, we use the HAVING clause to exclude the groups with the province ‘BC’.

```
SELECT au_fname AS ‘Author”s First Name’, province as ‘Province’
FROM Authors
GROUP BY au_fname, province
HAVING province <> ‘BC’
```

10.3: INSERT statement

The *INSERT statement* adds rows to a table. In addition,

- INSERT specifies the table or view that data will be inserted into.
- Column_list lists columns that will be affected by the INSERT.
- If a column is omitted, each value must be provided.
- If you are including columns, they can be listed in any order.
- VALUES specifies the data that you want to insert into the table. VALUES is required.
- Columns with the IDENTITY property should not be explicitly listed in the column_list or values_clause.

The syntax for the INSERT statement is:

```
INSERT [INTO] Table_name | view name [column_list]
DEFAULT VALUES | values_list | select statement
```

When inserting rows with the INSERT statement, these rules apply:

- Inserting an empty string (‘ ‘) into a varchar or text column inserts a single space.
- All char columns are right-padded to the defined length.
- All trailing spaces are removed from data inserted into varchar columns, except in strings that contain only spaces. These strings are truncated to a single space.
- If an INSERT statement violates a constraint, default or rule, or if it is the wrong data type, the statement fails and SQL Server displays an error message.

When you specify values for only some of the columns in the column_list, one of three things can happen to the columns that have no values:

1. A default value is entered if the column has a DEFAULT constraint, if a default is bound to the column, or if a default is bound to the underlying user-defined data type.
2. NULL is entered if the column allows NULLs and no default value exists for the column.
3. An error message is displayed and the row is rejected if the column is defined as NOT NULL and no default exists.

This example uses INSERT to add a record to the publisher’s Authors table.

```
INSERT INTO Authors
VALUES(‘555-093-467’, ‘Martin’, ‘April’, ‘281 555-5673’, ‘816 Market St.,’ , ‘Vancouv
```

This following example illustrates how to insert a partial row into the Publishers table with a column list. The country column had a default value of Canada so it does not require that you include it in your values.

```
INSERT INTO Publishers (PubID, PubName, city, province)
VALUES (‘9900’, ‘Acme Publishing’, ‘Vancouver’, ‘BC’)
```

To insert rows into a table with an IDENTITY column, follow the below example. Do not supply the value for the IDENTITY nor the name of the column in the column list.

```
INSERT INTO jobs
VALUES (‘DBA’, 100, 175)
```

10.3.1: Specific INSERT Examples

Inserting specific values into an IDENTITY column

By default, data cannot be inserted directly into an IDENTITY column; however, if a row is accidentally deleted, or there are gaps in the IDENTITY column values, you can insert a row and specify the IDENTITY column value.

```
IDENTITY_INSERT option
```

To allow an insert with a specific identity value, the IDENTITY_INSERT option can be used as follows.

```
SET IDENTITY_INSERT jobs ON
INSERT INTO jobs  (job_id, job_desc, min_lvl, max_lvl)
VALUES (19, 'DBA2', 100, 175)
SET IDENTITY_INSERT jobs OFF
```

Inserting rows with a SELECT statement

We can sometimes create a small temporary table from a large table. For this, we can insert rows with a SELECT statement. When using this command, there is no validation for uniqueness. Consequently, there may be many rows with the same pub_id in the example below.

This example creates a smaller temporary Publishers table using the CREATE TABLE statement. Then the INSERT with a SELECT statement is used to add records to this temporary Publishers table from the Publishers table.

```
CREATE TABLE dbo.tmpPublishers (
PubID char (4) NOT NULL ,
PubName varchar (40) NULL ,
city varchar (20) NULL ,
province char (2) NULL ,
country varchar (30) NULL  DEFAULT ('Canada')
)
INSERT  tmpPublishers
SELECT * FROM Publishers
```

In this example, we're copying a subset of data.

```
INSERT tmpPublishers (pub_id, pub_name)
SELECT PubID, PubName
FROM Publishers
```

In this example, the publishers' data are copied to the tmpPublishers table and the country column is set to Canada.

```
INSERT tmpPublishers (PubID, PubName, city, province, country)
SELECT PubID, PubName, city, province, 'Canada'
FROM Publishers
```

10.4: DELETE Statement

The *DELETE statement* removes rows from a record set. DELETE names the table or view that holds the rows that will be deleted and only one table or row may be listed at a time. WHERE is a standard WHERE clause that limits the deletion to select records.

The DELETE syntax looks like this.

```
DELETE [FROM] {table_name | view_name }
[WHERE clause]
```

The rules for the DELETE statement are:

1. If you omit a WHERE clause, all rows in the table are removed (except for indexes, the table, constraints).
2. DELETE cannot be used with a view that has a FROM clause naming more than one table. (Delete can affect only one base table at a time.)

What follows are three different DELETE statements that can be used.

1. Deleting all rows from a table.

```
DELETE
FROM Discounts
```

2. Deleting selected rows:

```
DELETE
FROM Sales
WHERE stor_id = ‘6380’
```

3. Deleting rows based on a value in a subquery:

```
DELETE FROM Sales
WHERE title_id IN
(SELECT title_id FROM Books WHERE type = ‘mod_cook’)
```

10.5: UPDATE statement

The *UPDATE statement* changes data in existing rows either by adding new data or modifying existing data.

This example uses the UPDATE statement to standardize the country field to be Canada for all records in the Publishers table.

```
UPDATE Publishers
SET country = ‘Canada’
```

This example increases the royalty amount by 10% for those royalty amounts between 10 and 20.

```
UPDATE roysched
SET royalty = royalty + (royalty * .10)
WHERE royalty BETWEEN 10 and 20
```

Including subqueries in an UPDATE statement

The employees from the Employees table who were hired by the publisher in 2010 are given a promotion to the highest job level for their job type. This is what the UPDATE statement would look like.

```
UPDATE Employees
SET job_lvl =
(SELECT max_lvl FROM jobs
WHERE employee.job_id = jobs.job_id)
WHERE DATEPART(year, employee.hire_date) = 2010
```

10.6: DELETE Statement

The *DELETE statement* removes rows from a record set. DELETE names the table or view that holds the rows that will be deleted and only one table or row may be listed at a time. WHERE is a standard WHERE clause that limits the deletion to select records.

The DELETE syntax looks like this.

```
DELETE [FROM] {table_name | view_name }
[WHERE clause]
```

The rules for the DELETE statement are:

1. If you omit a WHERE clause, all rows in the table are removed (except for indexes, the table, constraints).
2. DELETE cannot be used with a view that has a FROM clause naming more than one table. (Delete can affect only one base table at a time.)

What follows are three different DELETE statements that can be used.

1. Deleting all rows from a table.

```
DELETE
FROM Discounts
```

2. Deleting selected rows:

```
DELETE
FROM Sales
WHERE stor_id = ‘6380’
```

3. Deleting rows based on a value in a subquery:

```
DELETE FROM Sales
WHERE title_id IN
(SELECT title_id FROM Books WHERE type = ‘mod_cook’)
```

10.7: Built-in Functions

There are many built-in functions in SQL Server such as:

1. *Aggregate:* returns summary values
2. *Conversion:* transforms one data type to another
3. *Date:* displays information about dates and times
4. *Mathematical:* performs operations on numeric data
5. *String:* performs operations on character strings, binary data or expressions
6. *System:* returns a special piece of information from the database
7. *Text and image:* performs operations on text and image data

Below you will find detailed descriptions and examples for the first four functions.

10.7.1: Aggregate functions

Aggregate functions perform a calculation on a set of values and return a single, or summary, value. Table 10.7.1.1 lists these functions.

FUNCTION	DESCRIPTION
AVG	Returns the average of all the values, or only the DISTINCT values, in the expression.
COUNT	Returns the number of non-null values in the expression. When DISTINCT is specified, COUNT finds the number of unique non-null values.
COUNT(*)	Returns the number of rows. COUNT(*) takes no parameters and cannot be used with DISTINCT.
MAX	Returns the maximum value in the expression. MAX can be used with numeric, character and datetime columns, but not with bit columns. With character columns, MAX finds the highest value in the collating sequence. MAX ignores any null values.
MIN	Returns the minimum value in the expression. MIN can be used with numeric, character and datetime columns, but not with bit columns. With character columns, MIN finds the value that is lowest in the sort sequence. MIN ignores any null values.
SUM	Returns the sum of all the values, or only the DISTINCT values, in the expression. SUM can be used with numeric columns only.

Table 10.7.1.1: *A list of aggregate functions and descriptions.* (Copyright; author via source)

Below are examples of each of the aggregate functions listed in Table 10.7.1.1.

Example #1: AVG

```
SELECT AVG (price) AS ‘Average Title Price’
FROM Books
```

Example #2: COUNT

```
SELECT COUNT(PubID) AS ‘Number of Publishers’
FROM Publishers
```

Example #3: COUNT

```
SELECT COUNT(province) AS ‘Number of Publishers’
FROM Publishers
```

Example #3: COUNT (*)

```
SELECT COUNT(*)
FROM Employees
WHERE job_lvl = 35
```

Example #4: MAX

```
SELECT MAX (HireDate)
FROM Employees
```

Example #5: MIN

```
SELECT MIN (price)
FROM Books
```

Example #6: SUM

```
SELECT SUM(discount) AS ‘Total Discounts’
FROM Discounts
```

10.7.2: Conversion function

The conversion function transforms one data type to another.

In the example below, a price that contains two 9s is converted into five characters. The syntax for this statement is SELECT ‘The date is ‘ + CONVERT(varchar(12), getdate()).

```
SELECT CONVERT(int, 10.6496)
SELECT title_id, price
FROM Books
WHERE CONVERT(char(5), price) LIKE ‘%99%’
```

In this second example, the conversion function changes data to a data type with a different size.

```
SELECT title_id, CONVERT(char(4), ytd_sales) as ‘Sales’
FROM Books
WHERE type LIKE ‘%cook’
```

10.7.3: Date function

The date function produces a date by adding an interval to a specified date. The result is a datetime value equal to the date plus the number of date parts. If the date parameter is a smalldatetime value, the result is also a smalldatetime value.

The DATEADD function is used to add and increment date values. The syntax for this function is DATEADD(datepart, number, date).

```
SELECT DATEADD(day, 3, hire_date)
FROM Employees
```

In this example, the function DATEDIFF(datepart, date1, date2) is used.

This command returns the number of datepart “boundaries” crossed between two specified dates. The method of counting crossed boundaries makes the result given by DATEDIFF consistent across all data types such as minutes, seconds, and milliseconds.

```
SELECT DATEDIFF(day, HireDate, ‘Nov 30 1995’)
FROM Employees
```

For any particular date, we can examine any part of that date from the year to the millisecond.

The date parts (DATEPART) and abbreviations recognized by SQL Server, and the acceptable values are listed in Table 16.5.

DATE PART	ABBREVIATION	VALUES
Year	yy	1753-9999
Quarter	qq	1-4
Month	mm	1-12
Day of year	dy	1-366
Day	dd	1-31
Week	wk	1-53
Weekday	dw	1-7 (Sun.-Sat.)
Hour	hh	0-23
Minute	mi	0-59
Second	ss	0-59
Millisecond	ms	0-999

Figure 10.7.3.1: *Date part abbreviations and values.* (Copyright; author via source)

10.7.4: Mathematical Functions

Mathematical functions perform operations on numeric data. The following example lists the current price for each book sold by the publisher and what they would be if all prices increased by 10%. The example also shows the usage for the square root (SQRT), the round and the floor functions.

```
SELECT Price, (price * 1.1) AS ‘New Price’, title
FROM Books
SELECT ‘Square Root’ = SQRT(81)
SELECT ‘Rounded‘ = ROUND(4567.9876,2)
SELECT FLOOR (123.45)
```

10.7.5: Joining Tables

Joining two or more tables is the process of comparing the data in specified columns and using the comparison results to form a new table from the rows that qualify. A join statement:

- Specifies a column from each table
- Compares the values in those columns row by row
- Combines rows with qualifying values into a new row

Although the comparison is usually for equality – values that match exactly – other types of joins can also be specified. All the different joins such as inner, left (outer), right (outer), and cross join will be described below.

Inner join

An *inner join* connects two tables on a column with the same data type. Only the rows where the column values match are returned; unmatched rows are discarded.

Example #1

```
SELECT jobs.job_id, job_desc
FROM jobs
INNER JOIN Employees ON employee.job_id = jobs.job_id
WHERE jobs.job_id < 7
```

Example #2

```
SELECT authors.au_fname, authors.au_lname, books.royalty, title
FROM authorsINNER JOIN titleauthor ON authors.au_id=titleauthor.au_id
INNER JOIN books ON titleauthor.title_id=books.title_id
GROUP BY authors.au_lname, authors.au_fname, title, title.royalty
ORDER BY authors.au_lname
```

Left outer join

A *left outer join* specifies that all left outer rows be returned. All rows from the left table that did not meet the condition specified are included in the results set, and output columns from the other table are set to NULL.

This first example uses the new syntax for a left outer join.

```
SELECT publishers.pub_name, books.title
FROM Publishers
LEFT OUTER JOIN Books On publishers.pub_id = books.pub_id
```

This is an example of a left outer join using the old syntax.

```
SELECT publishers.pub_name, books.title
FROM Publishers, Books
WHERE publishers.pub_id *= books.pub_id
```

Right outer join

A *right outer join* includes, in its result set, all rows from the right table that did not meet the condition specified. Output columns that correspond to the other table are set to NULL.

Below is an example using the new syntax for a right outer join.

```
SELECT titleauthor.title_id, authors.au_lname, authors.au_fname
FROM titleauthor
RIGHT OUTER JOIN authors ON titleauthor.au_id = authors.au_id
ORDERY BY au_lname
```

This second example show the old syntax used for a right outer join.

```
SELECT titleauthor.title_id, authors.au_lname, authors.au_fname
FROM titleauthor, authors
WHERE titleauthor.au_id =* authors.au_id
ORDERY BY au_lname
```

Full outer join

A *full outer join* specifies that if a row from either table does not match the selection criteria, the row is included in the result set, and its output columns that correspond to the other table are set to NULL.

Here is an example of a full outer join.

```
SELECT books.title, publishers.pub_name, publishers.province
FROM Publishers
FULL OUTER JOIN Books ON books.pub_id = publishers.pub_id
WHERE (publishers.province <> “BC” and publishers.province <> “ON”)
ORDER BY books.title_id
```

Cross join

A *cross join* is a product combining two tables. This join returns the same rows as if no WHERE clause were specified. For example:

```
SELECT au_lname, pub_name,
FROM Authors CROSS JOIN Publishers
```

Key Terms

aggregate function: returns summary values

ASC: ascending order

conversion function: transforms one data type to another

cross join: a product combining two tables

date function: displays information about dates and times

DELETE statement: removes rows from a record set

DESC: descending order

full outer join: specifies that if a row from either table does not match the selection criteria

GROUP BY: used to create one output row per each group and produces summary values for the selected columns

inner join: connects two tables on a column with the same data type

INSERT statement: adds rows to a table

left outer join: specifies that all left outer rows be returned

mathematical function: performs operations on numeric data

right outer join: includes all rows from the right table that did not meet the condition specified

SELECT statement: used to query data in the database

string function: performs operations on character strings, binary data or expressions

system function: returns a special piece of information from the database

text and image functions: performs operations on text and image data

UPDATE statement: changes data in existing rows either by adding new data or modifying existing data

wildcard: allows the user to match fields that contain certain letters.

Exercises

For questions 1 to 18 use the PUBS sample database created by Microsoft. To download the script to generate this database please go to the following site: http://www.microsoft.com/en-ca/download/details.aspx?id=23654.

1. Display a list of publication dates and titles (books) that were published in 2011.
2. Display a list of titles that have been categorized as either traditional or modern cooking. Use the Books table.
3. Display all authors whose first names are five letters long.
4. Display from the Books table: type, price, pub_id, title about the books put out by each publisher. Rename the column type with ”Book Category.” Sort by type (descending) and then price (ascending).
5. Display title_id, pubdate and pubdate plus three days, using the Books table.
6. Using the datediff and getdate function determine how much time has elapsed in months since the books in the Books table were published.
7. List the title IDs and quantity of all books that sold more than 30 copies.
8. Display a list of all last names of the authors who live in Ontario (ON) and the cities where they live.
9. Display all rows that contain a 60 in the payterms field. Use the Sales table.
10. Display all authors whose first names are five letters long , end in O or A, and start with M or P.
11. Display all titles that cost more than $30 and either begin with T or have a publisher ID of 0877.
12. Display from the Employees table the first name (fname), last name (lname), employe ID(emp_id) and job level (job_lvl) columns for those employees with a job level greater than 200; and rename the column headings to: “First Name,” “Last Name,” “IDENTIFICATION#” and “Job Level.”
13. Display the royalty, royalty plus 50% as “royalty plus 50” and title_id. Use the Roysched table.
14. Using the STUFF function create a string “12xxxx567” from the string “1234567.”
15. Display the first 40 characters of each title, along with the average monthly sales for that title to date (ytd_sales/12). Use the Title table.
16. Show how many books have assigned prices.
17. Display a list of cookbooks with the average cost for all of the books of each type. Use the GROUP BY.

Advanced Questions (Union, Intersect, and Minus)

1. The relational set operators UNION, INTERSECT and MINUS work properly only if the relations are union-compatible. What does union-compatible mean, and how would you check for this condition?
2. What is the difference between UNION and UNION ALL? Write the syntax for each.

3. Suppose that you have two tables, Employees and Employees_1. The Employees table contains the records for three employees: Alice Cordoza, John Cretchakov, and Anne McDonald. The Employees_1 table contains the records for employees: John Cretchakov and Mary Chen. Given that information, what is the query output for the UNION query? List the query output.
4. Given the employee information in question 3, what is the query output for the UNION ALL query? List the query output.
5. Given the employee information in question 3, what is the query output for the INTERSECT query? List the query output.
6. Given the employee information in question 3, what is the query output for the EXCEPT query? List the query output.
7. What is a cross join? Give an example of its syntax.
8. Explain these three join types:
    1. left outer join
    2. right outer join
    3. full outer join
9. What is a subquery, and what are its basic characteristics?
10. What is a correlated subquery? Give an example.
11. Suppose that a Product table contains two attributes, PROD_CODE and VEND_CODE. The values for the PROD_CODE are: ABC, DEF, GHI and JKL. These are matched by the following values for the VEND_CODE: 125, 124, 124 and 123, respectively (e.g., PROD_CODE value ABC corresponds to VEND_CODE value 125). The Vendor table contains a single attribute, VEND_CODE, with values 123, 124, 125 and 126. (The VEND_CODE attribute in the Product table is a foreign key to the VEND_CODE in the Vendor table.)
12. Given the information in question 11, what would be the query output for the following? Show values.
    1. A UNION query based on these two tables
    2. A UNION ALL query based on these two tables
    3. An INTERSECT query based on these two tables
    4. A MINUS query based on these two tables

Advanced Questions (Using Joins)

1. Display a list of all titles and sales numbers in the Books and Sales tables, including titles that have no sales. Use a join.
2. Display a list of authors' last names and all associated titles that each author has published sorted by the author's last name. Use a join. Save it as a view named: Published Authors.
3. Using a subquery, display all the authors (show last and first name, postal code) who receive a royalty of 100% and live in Alberta. Save it as a view titled: AuthorsView. When creating the view, rename the author's last name and first name as 'Last Name' and 'First Name'.
4. Display the stores that did not sell the title *Is Anger the Enemy?*
5. Display a list of store names for sales after 2013 (Order Date is greater than 2013). Display store name and order date.
6. Display a list of titles for books sold in store name "News & Brews." Display store name, titles and order dates.
7. List total sales (qty) by title. Display total quantity and title columns.
8. List total sales (qty) by type. Display total quantity and type columns.
9. List total sales (qty*price) by type. Display total dollar value and type columns.
10. Calculate the total number of types of books by publisher. Show publisher name and total count of types of books for each publisher.
11. Show publisher names that do not have any type of book. Display publisher name only.

CHAPTER OVERVIEW

11: Client/Server Architecture

11.1: Introduction and Background to Client/Server Systems and Multi-tier Architecture

6.1 - Introduction and Background to Client/Server Systems and Multi-tier Architecture

The client/server model is the standard model of networked traffic in today's computing and database systems. The client/server model was designed with world wide web services in mind and to allow for servers to provide service to end user devices all over the network. This methodology has been adopted as the standard for content publishing and receiving by modern network systems.

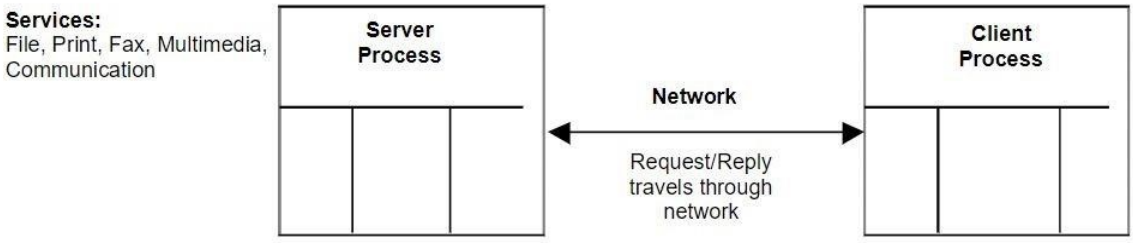


Figure 11.1.1: A graphic demonstrating the architecture of client/server systems. From "*An Introduction to Client/Server Computing*" by Yadav SC, Singh SK, 2009.

For databases, the model of client/server is ideal because the database can easily serve multiple users at the same time through standardized calls for information. If multiple users are looking to fetch the same files on a routine basis, tools can be deployed which monitor the trends of usage and allow for IT administrators to beef up the systems which serve those files, reducing wait times and increasing overall system performance. Similarly, for systems or databases which are accessed less frequently, those costs can be avoided.

Prior to the client/server architecture, peer-to-peer networking was used. This means that for every request to send or receive, devices would communicate one-to-one, and for each transaction of data, this process had to be repeated. By taking on the role of a "server", a system can handle more than one call at once, drastically improving throughput and efficiency. Using the client/server model gives rise to a very specific set of requirements when designing a transport layer protocol. Generally, a transport layer protocol needs to cover all of the following requirements: a connection between a client and server, an interaction between the client and server, authentication of the client and server, and a checksum or other method to maintain data integrity after having sent and received packets.

Multi-Tier Architecture (MTA) can be generalized as a separation and duplication of a server/database system by decentralizing data and compute resources. MTA is used to improve reliability and throughput, similarly to client/server architecture. As such, an MTA environment will almost certainly use client/server communication.

MTA systems are composed of 6 basic layers (Wall, D, Morgan Kaufmann):

- Persistence: The database which serves files to the applications that service the clients or users.
- Accessor: Typically, the SQL server. Not the disks in the database, but the part of the database which does the thinking.Logic: The applications which the users interact with, in turn providing instructions to the database from which files are requested.
- Presentation: Systems applications which package data from the database into web browser languages such as HTML or XML.
- Requester/consumer: The web browser itself; this is on the client side.
- Elsewhere: The sources of information hosted on other platforms, such as AWS, Azure,

  or GCP, or other sources of HTML content that are not native.

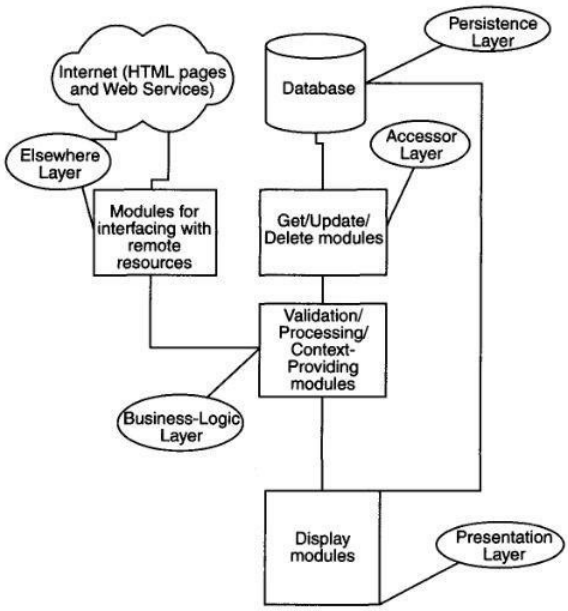


Figure 11.1.1: A graphic demonstrating the architecture of multi tier architecture.

11.2: Three Components of Client/Server Systems

The client/server model is designed to manage data and facilitate the methods by which users can access it. In the client/server model, data is shared between one central server that houses and provides access to it and one or more clients that access the data through either a direct or remote feed. By calling requests to the server, clients can access any information they have access to, but clients cannot access information from other clients, as that would represent a peer-to-peer system, rather than a client/server model.

In client/server systems, there are three main components of logic. Presentation logic controls the inputs and outputs. It "presents" the data to the user in a readable format and also collects information that a user inputs using their system. Processing logic oversees everything around turning the client request into a request the server can use in the next step to find information. The final component is storage logic, which handles physical data storage and processes for retrieving it upon user request.

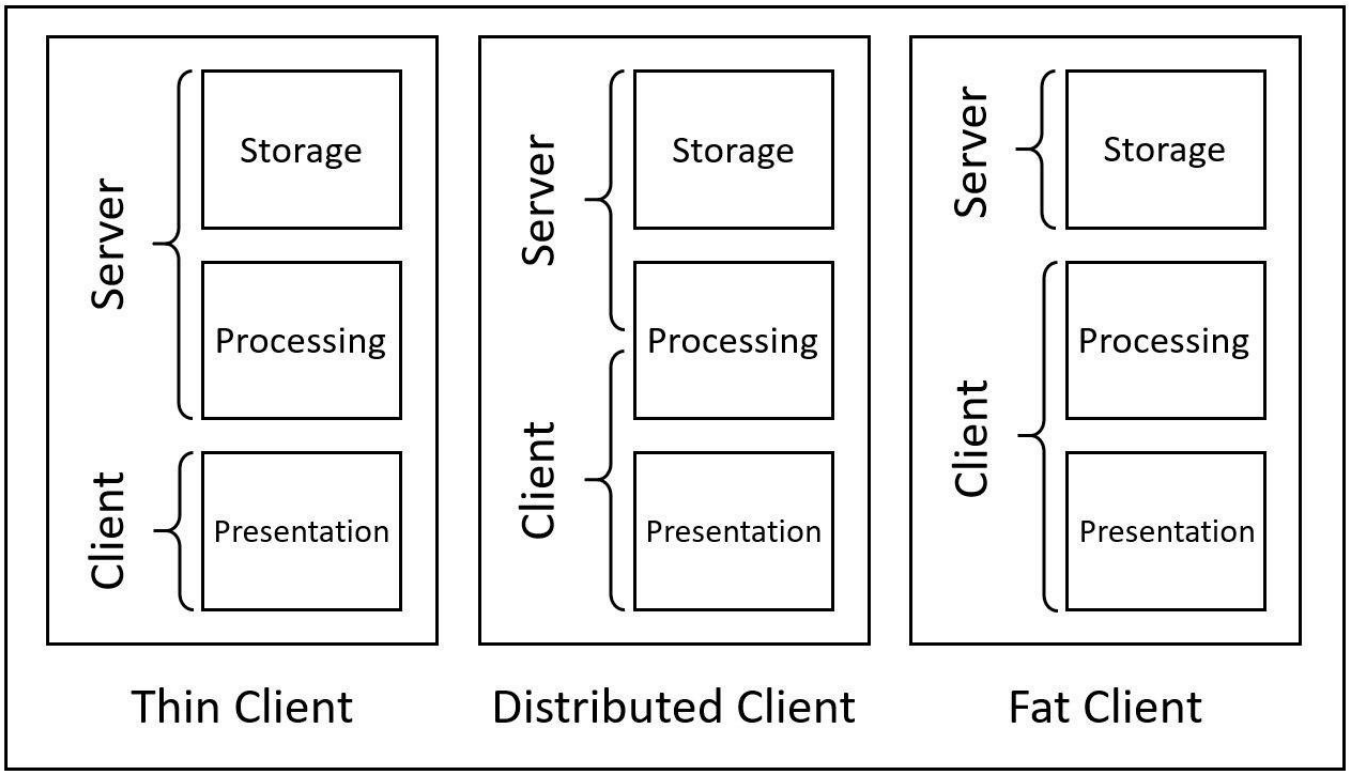


Figure 11.2.1: Client distribution models (made via Microsoft PowerPoint, Michael Nolan 2020)

The way that systems using this model delegate these logical applications defines individual databases further. Two-tier database architecture is divided into three groups named to specify what the client takes care of, as shown in Figure 1. In fat client distribution, presentation and processing are both handled by the client system, while the server exclusively handles the storage logic application. In thin client distribution, the client only handles presentation, while the server takes care of processing the request and dealing with storage. Finally, when a two-tier system is distributed in design, the client and server both handle parts of processing logic. In these sorts of systems, presentation logic is still handled by the client and storage is still handled by the server exclusively.

Further levels of tiered architecture in databases works similarly to the aforementioned design ideas, but splits processing logic applications further between multiple intermediate applications and web servers before the client's request reaches the database server.

a. Presentation

Presentation logic is the high-level interactive portion of the database. It is where the enduser can input requests and receive output on their host device.

Input in the presentation level of the client-server model can be many different things. GUI elements can be added to make it easier, such as text fields that users can modify, copy, or delete, buttons for submitting requests, menus, links, and more. Output is handled by the system to present the server response to user queries in a readable way that gives the user all the information they need without giving them information they did not request or information to which they are not allowed access.

Presentation logic is handled exclusively by client software on the host device. This can be managed by some language such as HTML to quickly take inputs and display outputs in a controlled location.

b. Processing

The processing logic layer is an intermediary layer that translates information back and forth between user language coming from the client software and abstracted language used by the database's storage unit to quickly access the correct information. The processing layer also handles business rules, assuming they are not already built into the organizational DBMS.

Processing logic has many automated measures defined by business logic and business rules that are tailored to the specific company or organization the database is a part of. It is also where data integrity is insured whenever commands are sent to the storage database.

Applications for this are generally coded in Java or C, as those languages can be applied to many kinds of hardware with ease and offer some scalability without slowing down the system. Attention needs to be paid to these processes to make sure that this stage of data transfer does not bottleneck, as the processing logic may have to handle many requests per day.

Processing logic is also where the majority of differences in database architecture lie. Two-tier architecture can be split three ways: fat client, thin client, and distributed. Fat client distribution has all processing logic handled on the client's side, so the user host machine sends a request to the database having already translated the language from the user's input into whatever language that specific database uses for queries. Fat client distribution is the most commonly used organization of database access in the client-server system. Thin client distribution has all processing logic handled by the server system. The goal of this is to keep the user interface as light as possible. Distributed systems have processing logic handled by both the client and server in a way that boosts efficiency. It is more complex than the other two methods to set up.

Multi-tier organization splits processing logic in different ways. Three-tier systems may have processing logic all performed on a separate web server that clients and servers both access. Above that, processing can be handled by multiple web and application servers that each manage smaller parts of the translation. The goal of these larger systems is to be more modular and scalable while still being easy to maintain, since problems can be pinpointed easily.

c. Storage

The storage logic represents the physical database and storage devices. It is where data storage and retrieval is handled in the system, and it is the part of the client-server system where the DBMS interacts with everything else.

Data in the database can be stored in many different ways. Data can be accessed using the information provided by the processing applications very quickly.

11.3: Two-tier and Three-tier Architectural Distinctions

Though many people may not realize it, when a group of tech-savvy individuals develops software, one of the most important decisions at the start is the architecture: “architecture is a key feature which decides that the system will meet its performance and other quality objectives” (Hayat and Akram, 2007). Several main components come into play when it comes to deciding the type of architecture, including performance, availability, complexity, and cost.

Performance depends on the quantity of simultaneous users that can be processed by the database system to ensure smooth processing. Typically, most applications limit the number of users accessing different services of the application by only providing them with restricted data. While choosing an architecture, it is quite important to keep in mind the performance component. Another significant factor is the cost. The selected architecture must be affordable for any organization just starting. Availability is the accessibility of the system to the community of users. Complexity is the extent of difficulty of the system in terms of the user’s and developer’s viewpoint. If the developers feel they want a lower level of difficulty, they can make the user’s level of difficulty rise due to lack of user interface organization/improvements. However, to lower the user’s level of difficulty, the developers may have to increase their effort and continuously improve, revise, and contemplate new ways to make the user interface easier and more accessible.

In two-tier architectures, the interfaces run on a client, which means the client communicates with the server directly, and because it runs on a client, another data layer or structure is then stored on a server, as seen in Figure 11.3.1 below. The client's responsibility covers user interface logic from the presentation, which deals with how objects in the business get shown to other users of the software as well as data processing and rules (algorithms) that dictate the information that gets exchanged between the database system and the user interface. Meanwhile, database servers access and process requests from the clients. This architecture allows for optimization of the processing time in the database servers and is highly recommended for small workgroups. However, clients or servers that use two-tier architecture tend only to support a relatively small number of users, because the transaction of information is usually low, and security is not necessarily the top priority. Because of this, two-tier architectures are not used for applications with large user bases due to their limited flexibility.

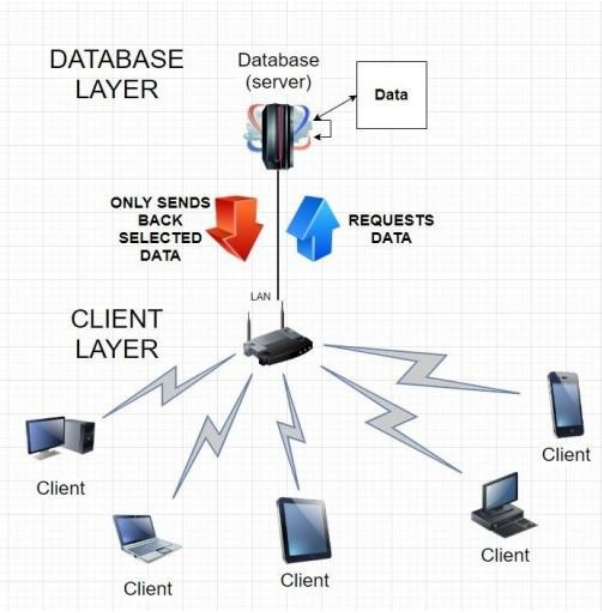


Figure 11.3.1: Visual of two-tier architecture (two layers) (Made via Draw.io, Martin Nguyen, 2020)

In general, three-tier (n-tier) architecture includes another server layer, as seen in Figure 11.3.2 This extra layer make it much easier to scale the system, which makes performance much better, resulting in a highly flexible and reusable structure. The extra layer can have several purposes, such as an application server, or it can be used as local storage and hold local databases. In contrast, the other servers hold more enclosed data. Utilizing a three-tier architecture can lead to better performance, and make it easier to transport application code to other platforms, reducing dependency on commonly domain-specific language. Three-tier systems can often be found in Web-based applications—any program accessible using HTTP.

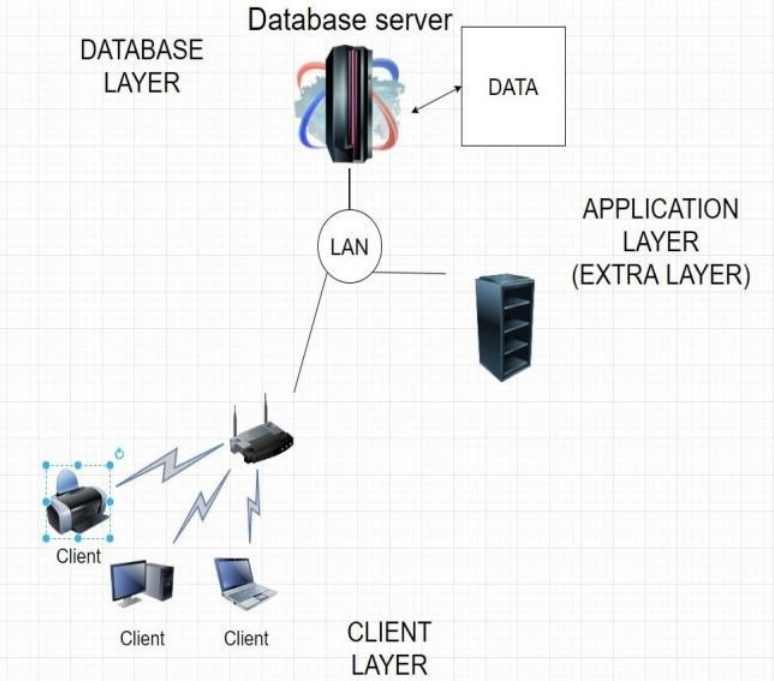


Figure 11.3.2: Visual of three-tier (three or nth layers) (Made via Draw.io, Martin Nguyen, 2020)

11.4: Connecting to databases in three-tier applications

Three-tier application structure is similar to the 7-layer OSI model of telecommunications, in that it breaks up the presentation, data, and application functions into each of their own levels. This segmentation provides for a more streamlined approach to scalability and ease of development, since the developer does not have to produce a new web technology for every instance. Instead, standards such as HTML, CSS, or JavaScript may be used.

When connecting to a three-tier based web application, the user will interact with the presentation tier. This layer is the visual wrapper between the functionality, database, and UI elements. This layer is displayed in a web browser, and served through the World Wide Web or a Local Area Network. Connecting to an IP address using a compatible browser will provide the web page to the user, being provided through a server which hosts the website and has access to the databases which it uses. The presentation tier is largely the visual aspect of a website or application and does not perform the computing or data storage/retrieval. This layer is important, however, because data entry and user intent must be clearly interpreted for the right API calls to be made by the presentation layer, to be sent to the application and data tiers.

By using a three-tier architecture, developers are able to concurrently work on each tier of the application as they are designing it. This parallel development allows for fewer limitations due to older design which cannot easily be changed. This is relevant to the connection to databases, due to the ability of the development teams to agree on which APIs and methods they wish to communicate with between each layer. For the database to be accessed, the user would make a request for a piece of information to be pulled from the database through the presentation layer. The presentation layer would process the request through the application layer (or logic layer) which would request the data from the database. All of this would happen through the use of API calls in a modern application. The database layer would be a version of SQL such as MySQL, Microsoft SQL Server, or another database type.

Once the request reaches the data layer, the database management system (DBMS) is what will actually do the work of finding what the request is looking for. This includes querying the database indexes to see where the information is stored, as databases are usually composed of one or more storage devices with many logical storage endpoints.

11.5: Concise Summary

The client-server model is regularly used today for designing system architecture. At its base, it is a simple back-and-forth transfer of information from client requests to server responses, but it is highly customizable for many functions.

Different client-server implementations can take several forms, depending on how many tiers exist in the design. They range from simple thin and fat client two-tier systems to modular distributed systems, up to any number of tiers of web and application servers separating clients from their servers. The addition of layers of abstraction serves to make the host client's UI as simple to use as possible, making them desirable to many end users.

11.6: Extended Resources

1. Gives the definition and a brief background of client server model/architecture. Explains the topics of advantages and disadvantages of the model as well as protocols and other program relationship models as well. The video goes through and describes as well as gives a visual of client server architectures.
2. This video talks about software multilayered architectures consisting of one tier, two tier, three tier, and n-tier architectures.
3. This slideshow walks you through and visually shows you how to connect database files to tier architecture applications.
4. A history of the client server.
5. Microsoft's client/server application guidelines.
6. A detailed overview of client server applications.
7. This video gives a basic tutorial on client server applications.

11.7: References

Akram, Muhammad. “Qualitative & Quantitative analysis of tiered Architecture of Web- Applications.“ 2007. PDF.

Yadav SC, Singh SK. An Introduction to Client/Server Computing. New Delhi: New Age International; 2009.

Wall, D, Morgan Kaufmann. Multi-Tier Application Programming with PHP: Practical Guide for Architects and Programmers; 2004.

CHAPTER OVERVIEW

12: Physical Database Design and Database Security

12.1: Introduction

In this chapter, we describe physical database design, with special emphasis on the relational data model. Physical database design is the process of transforming logical data models into physical data models. An experienced database designer will make a physical database design in parallel with conceptual data modeling if they know the type of database technology that will be used. Conceptual data modeling is about understanding the organization and gathering the right requirements. Physical database design, on the other hand, is about creating stable database structures correctly expressing the requirements in a technical language. Although there are other data models, we have two reasons for emphasizing the relational data model in this chapter. First, the relational data model is one most commonly used in contemporary database applications. Second, some of the principles of logical database design for the relational model apply to other logical models as well.

We introduced the relational data model informally through simple examples in earlier chapters. It is important, however, to note that the relational data model is a form of logical data model, and as such, it is different from the conceptual data models. Thus, an *Entity-Relationship (E-R)* data model is not a relational data model, and an E-R model may not obey the rules for a well-structured relational data model, called *normalization*. The E-R model was developed for other purposes such as understanding data requirements and business rules about the data rather than structuring the data for sound database processing.

We next describe and illustrate the process of transforming an EER (*Enhanced Entity-Relationship*) model into the relational model. Many CASE tools support this transformation today at the technical level; however, it is important to understand the underlying principles and procedures. We then describe the concepts of normalization in detail. Normalization, which is the process of designing well-structured relations, is an important component of logical design for the relational model. Finally, we describe how to merge relations while avoiding common pitfalls that may occur in this process.

In Chapters two and three, we learned how to describe and model organizational data during the conceptual data modeling and logical database design phases of the database development process. We learned how to use EER notation, the relational data model, and normalization to develop abstractions of organizational data that capture the meaning of data. However, these notations do not explain how the data will be processed or stored. The purpose of physical database design is to translate the logical description of data into the technical specifications for storing and retrieving data. The goal is to create a design for storing data that will provide adequate performance and ensure database integrity, security, and recoverability.

Physical database design does not include implementing files and databases (i.e., creating them and loading data into them). Physical database design produces the technical specifications that programmers, database administrators, and others involved in information systems construction will use during the implementation phase. In this chapter, we study the basic steps required to develop an efficient and high-integrity physical database design. We concentrate in this chapter on the design of a single, centralized database and how to estimate the amount of data that users will require in the database.

e will also learn about choices for storing attribute values and how to select from among these choices to achieve efficiency and data quality. Because of recent U.S. and international regulations (e.g., Sarbanes-Oxley) on financial reporting by organizations, proper controls specified in physical database design are required as a sound foundation for compliance. Hence, we place special emphasis on data quality measures you can implement within the physical design. We will also learn why normalized tables are not always the basis for the best physical data files and how to de-normalize the data to improve the speed of data retrieval. Finally, we learn about the use of indexes, which are important in speeding up the retrieval of data.

When performing physical database design, the decisions made during this stage have a major impact on data accessibility, response times, data quality, security, user friendliness, and similarly important information system design factors. Database administration plays a major role in physical database design. Advanced design will be discussed in this chapter.

Logical	Physical
Entity	Table
Relationship	Foreign Key
Attribute	Column
Unique Identifier	Primary Key

Figure 12.1.1: Comparison between Logical and Physical Design

12.2: Background

The first database system was created in the 1960s as computers became affordable for private organizations. The two most popular data models during this time were the network model CODASYL (Conference/Committee on Data Systems Languages) and the hierarchical model called IMS (Information Management System). These two systems were used by private organizations, but the SABRE (Semi-Automated Business Research Environment) system saw commercial success through International Business Machines (IBM). IBM used the SABRE system to help American Airlines manage reservation data for customers. These database systems began to change in the 1970s when E.F. Codd published a scientific paper that explored the idea of a relational database model. E.F. Codd's ideas about a relational database revolutionized the way people view databases. Codd's relational database schema separated the physical information storage, which became the new standard for database systems.

Two major relational database systems emerged in 1974 and 1977. One of the databases was called Ingres, which was created at UBC; the second was called System R, developed at IBM. Ingres used a query language known as QUEL, and it led to the creation of systems such as Ingres Crop., MS SQL Server, Sybase, Wang's PACE, and Britton-Lee (Quick Base). On the other hand, System R used the SEQUEL query language, and it contributed to the development of SQL/DS, DB2, All base, Oracle, and Non-Stop SQL (Quick Base). Relational Database System became the standard and a recognized term in industry.

The next evolution in database systems came in 1976 through the Entity-Relationship (E-R) model. Entity-Relationship database model was created by P. Chen. The E-R model allowed developers to focus less on logic table structure and more on research data application. In 1980, Structured Query Language (SQL) became the standard query language among databases. Also during this time, computer sales saw commercial success, which aided the database market. This increase in sales led to the decline in legacy network and hierarchical database models. With the development of the Internet, the database industry saw substantial growth. Databases started to be accessed from client-servers. Online business was becoming popular, and with the demand came the need for internet database connectors increased. Security also played an important part in the development of databases.

Before 1980, Government organizations such as the Department of Defense were the first to invest heavily into security of data. This was due to the type of data they were storing, such as military data and census data. Most organizations framed security policies around the few vulnerabilities they detected. During this time, physical threats were understood and preventive measurements were put in place. Logical threats or digit threats were difficult to understand and were very weak during this time. Mainstream research was focused on statistical databases. Access Control was the first security control to come out of this research. "Access Control for databases was to be expressed in terms of logical data model with authorizations in terms of relations and tuples. It also had to be content-aware to allow the system to determine whether access should be granted based on the content of the data item" (Lesov, 2010). This type of access control became known as the Bell-LaPadula model or BLP. Systems were now required to store shared resources which also increased the need for security across those resources. Two new models were created: the Mandatory Access Control (MAC) and the Discretionary Access Controls (DAC). However, both of these did not create great success and were later reinforced with encryptions. Two designs for encryptions were the Access Control Kernels and Encrypted Databases. Kernels isolate and contain security policies inside different modules. Cryptography was used through keys to encrypt and store data to maximize security. Paul Lesov from University of Minnesota writes, "Access control was not sufficient by itself to address the issue of DBA being able to exercise complete control over the data residing in the database. Encryption provides a way to encrypt data in the database and store an encryption key external to the database thereby preventing the DBA from accessing the data" (2010).

The digital environment went through a massive transformation that was driven by commercialization of the digital space. Windows was developed and adopted during this time along with the World Wide Web. The idea of using the web as a place to conduct commerce made the security required of the early nineties very different from the late nineties. Another major development during this time was the idea of Object-Oriented Programming. Object-Oriented Programming allowed for more complex and efficient ways to deal with complex data. Early security for databases connected through the web took the form of firewalls which control access to internal servers. Firewalls provided protection against direct attacks on the database but the front-end were left susceptible to SQL Injections.

SQL Injections allowed users to insert scripts into the database which would retrieve information. The idea of data mining extended the threat to individual privacy. Government regulations such as the Health Insurance Portability and Accountability Act (HIPAA) and the Federal Financial Institutions Examination Council (FFIEC) were enacted during this time to protect individual privacy. Paul Lesov stated, "It is important to note that many problems with securing data stored in a database is not due to the lack of research but lacking security in implementation of the database product or an application front ending the database. The shift from

full trust to partial trust was driven in part by natural tendency to not provide full trust to anyone single individual based on dual control principle but also due to the inability of the users to keep their own PC computer secure and database frontend not being able to detect malicious attacks such as SQL injections" (2010). Since then, the growth of the databases outgrew the pace of security research. Due to the amount of data and complexity around data security, research was slow. Today research is still being conducted on database security and is ever evolving year to year.

12.3: Physical DB Design Process

The design of a physical database is heavily influenced on integrity and performance for the system (Hoffer et al., 2015). To maximize efficiency for user interactions with an information system, minimizing time constraints associated with user interactions must be taken into account. According to Adrienne Watt, database design starts with a conceptual data model and produces a specification of a logical schema; this will determine the specific type of database system (network, relational, object-oriented) that is required. The relational representation is still independent of any specific DBMS; it is another conceptual data model (Watt, n.d.). The database design process is initialized from the logical data model that will be used in the database design and can be represented as an E-R (Entity-Relationship) diagram (Figure 1).

Physical database designs require preliminary pieces of information that are collected prior to development. For a physical file design and database design, normalized relations, attribute definition, descriptions of data handling, the expectations for response times, data security, backup, recovery, retention, and integrity of data, and descriptions of the technology are utilized to implement the database (Hoffer et al., 2015). Before commencing analysis and implementation of the database design, designers must first consult with customers to acquire relevant documents and data to be used in analysis and implementation to achieve the desired results based on the customers' needs.

Physical database design is also concerned with the design of fields. A field is the smallest unit of application data recognized by system software. Fields are associated with the columns found in a relationship table and rely on the data types, coding techniques, data integrity, and handling of empty data entry. Efficient database performance relies on adequately defined fields and subsequently relies on adequate specifications for each field.

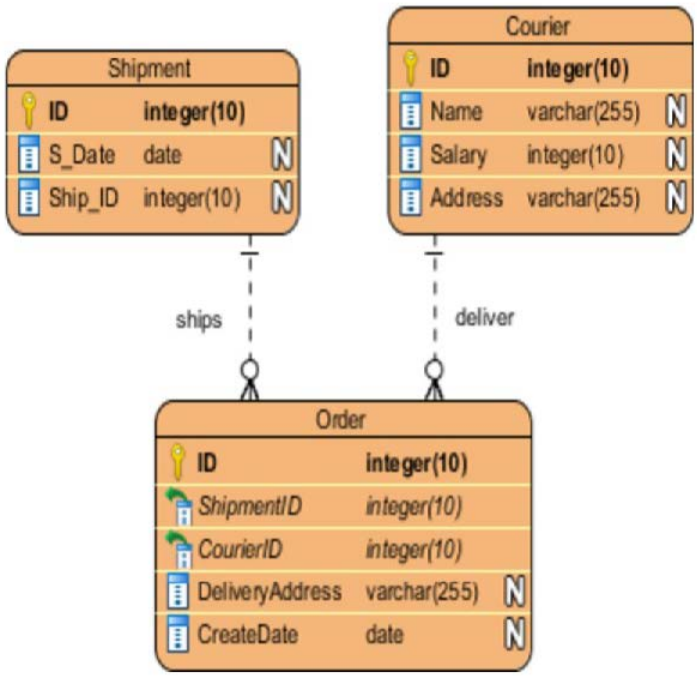


Figure 12.3.1: Copy and Paste Caption here. (Copyright; author via source)

The format for data storage is an imperative decision to be made when considering storage space and data-related integrity as shown in Figure 1. This format can be integrated with the selection of data types that are only needed for the database design to be implemented. Data types can be categorized as a string, numerical, and datetime data values, and can be further decomposed into varying lengths of data for their respective categories as shown in Figure 12.3.2

String	Numeric	Date/Time
CHAR	SMALLINT	DATE
VARCHAR	INTEGER	TIME
CLOB	DOUBLE	TIMESTAMP

Figure 12.3.2: Copy and Paste Caption here. (Copyright; author via source)

Coding and compression techniques are critical for handling data storage within fields. Allowing for smaller-sized codes reduces the space usage of data values within databases. An example of such a coding technique could involve cross-referencing to other tables in order to retrieve a field value. Aside from using data types as a form of data integrity, other forms of data integrity to be considered in a physical database design include default values, which are automatically assigned unless specified otherwise by a user; range control, which a preset range limit specifies the set of values a field can be assigned; and null control, which restricts null values.

12.4: Data Partitioning

Partitioning is a concept in databases in which very large tables and data are partitioned into smaller, individual tables, and queries which helps the data process more quickly and efficiently. One form of partitioning is called horizontal partitioning. Horizontal partitioning is the classification of the rows based on common characteristics into several, separate tables. For example, students with scores less than 70 might be stored in the FAIL table, while students who scored greater than 70 might be stored in the PASS table. The two partitioned tables would be FAIL and PASS. In doing so, the database is partitioned into two tables, both of which will be processed more quickly than a single table. The two typical methods of horizontal partitioning are to partition a sole column value and to partition the date (as the date helps organize the query chronologically). The keyword SELECT represents a horizontal partition in SQL. Below is an example of what horizontal partitioning would look like in SQL in which the first name, last name, class number and grade are specifically chosen as desired values:

SELECT viewClassGradeReport.[FirstName],

viewClassGradeReport.[LastName],

viewClassGradeReport.[ClassName],

viewClassGradeReport.[Grade] FROM viewClassGradeReport;

The Oracle DBMS (Database Management System) provides support for multiple forms of partitioning. The first one is range partitioning. In range partitioning, each partitioned portion is characterized by a range of values for one or multiple columns, such as IDs or dates. For instance, rows for a column titled COUNTRY containing India, Sri Lanka, Pakistan, Nepal, and Bangladesh could be a partition for South Asian countries

The next form of partitioning is hash partitioning. Hash partitioning is the spreading of data in even partitions autonomous of the key value. Hash partitioning outperforms the uneven distributions or rows.

The next partitioning is known as list partitioning. List partitioning is a technique where a list of distinct values is defined as the partitioning key in the characterization for each partition.

The best use of list partitioning is when one requires specifically to highlight rows established on discrete values. For instance, a global distributor may only want data regarding deliveries to China and Japan while ignoring deliveries to other nations.

Oracle provides another form of partitioning known as composite partitioning. Composite partitioning is a combination of the general data allocation methods in which a table is partitioned using one allocation method, and then each partition is subdivided into smaller partitions through another general allocation method.

The counterpart of horizontal partitioning, vertical partitioning is another form of partitioning. Vertical partitioning, as opposed to horizontal partitioning, is the distribution of columns based on their commonalities and separating them into independent tables. Vertical partitioning helps identify the dynamic data from the stable, immutable data. Vertical partitioning is represented with the keyword PROJECT.

Partitioning is a useful tool that helps with the design of the database and has many advantages. Partitioning is practical and helps manage the table because partitioning helps identify the area where maintenance is needed and saves storage space. Partitioning is also secure, as only the relevant and necessary data can be specifically chosen and accessed by the user. Another benefit of implementing partitioning is that backing up and securing files is easier due to their smaller size, and if one file is corrupted, the other is still accessible. Partitioning works similarly with cars and replaceable parts: if one piece of equipment is damaged, the rest of the car will still be functioning. Partitioning also helps with balancing the load. The partitioned files can be designated to different storage locations, reducing conflict and maximizing performance.

Although partitioning proves to be quite handy, it does have its drawbacks. The first drawback of partitioning is the inconsistency of the access speed that it provides. All partitions are not identical; therefore, depending on the data the specific partition consists, access speeds will differ. Another drawback is complexity. Due to the complex nature of partitions, the code required to program will need to be more complex and challenging to maintain. The final disadvantage of partitioning is the excess storage space and time. Data can be replicated multiple times, which in turn takes up storage and can affect the time taken to process.

Partitioning is a useful tool that helps with the design of the database and has many advantages. Partitioning is practical and helps manage the table because partitioning helps identify the area where maintenance is needed and saves storage space. Partitioning is also secure, as only the relevant and necessary data can be specifically chosen and accessed by the user. Another benefit of

implementing partitioning is that backing up and securing files is easier due to their smaller size, and if one file is corrupted, the other is still accessible. Partitioning works similarly with cars and replaceable parts: if one piece of equipment is damaged, the rest of the car will still be functioning. Partitioning also helps with balancing the load. The partitioned files can be designated to different storage locations, reducing conflict and maximizing performance.

Although partitioning proves to be quite handy, it does have its drawbacks. The first drawback of partitioning is the inconsistency of the access speed that it provides. All partitions are not identical; therefore, depending on the data the specific partition consists, access speeds will differ. Another drawback is complexity. Due to the complex nature of partitions, the code required to program will need to be more complex and challenging to maintain. The final disadvantage of partitioning is the excess storage space and time. Data can be

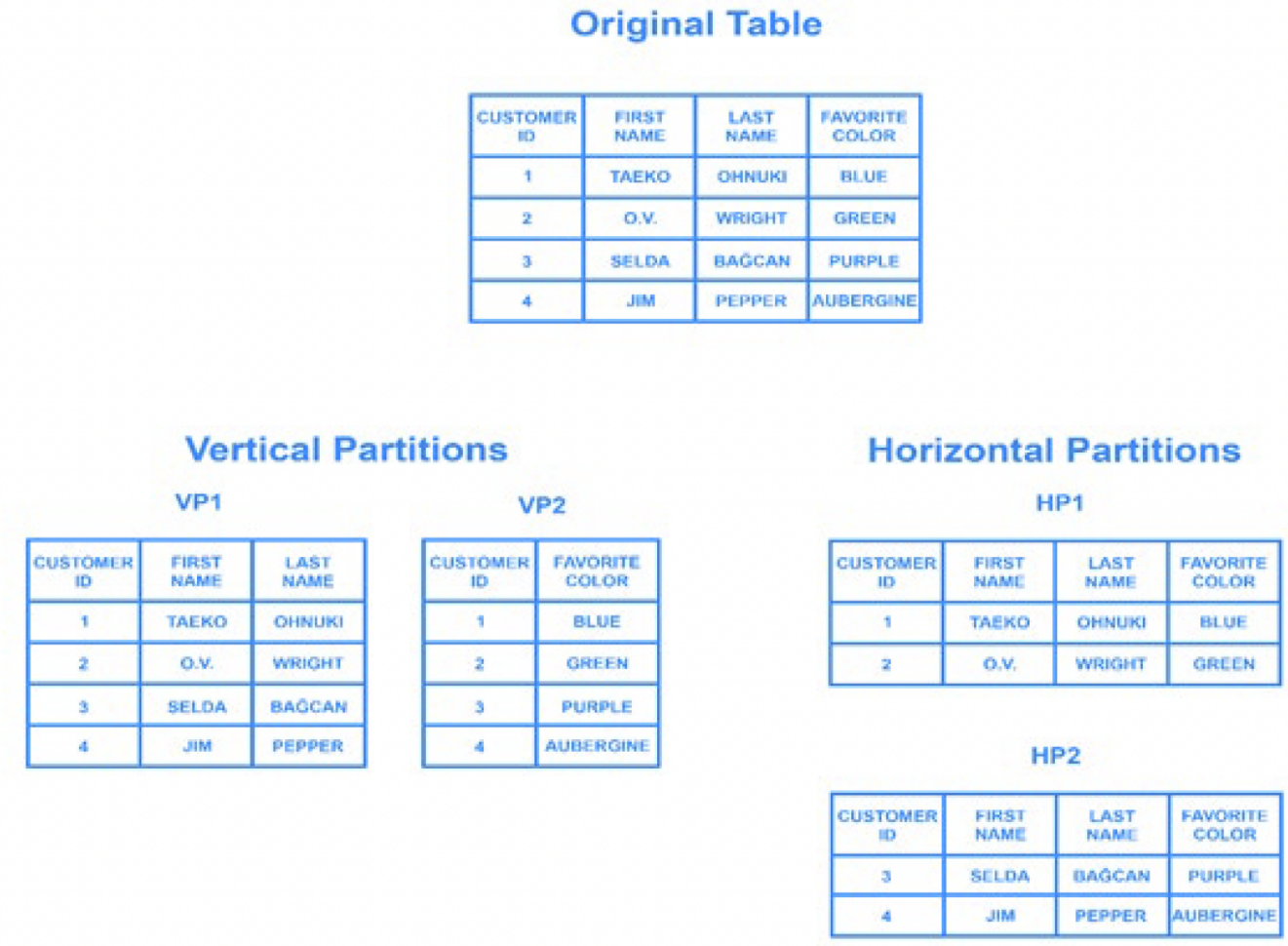


Figure 12.4.1:

Sharding, by Digital Ocean, Copyright 2020 by Digital Ocean

The user view, in a database, is a tool that helps visualize the tables in a database. Using the user view, physical tables can be partitioned coherently. The main intention of utilizing the user view is that it simplifies query editing and develops a secure database. In Oracle, a form of user view is offered called partition view, which displays physically partitioned tables that can be logically merged into one using the SQL union operator. This manner of partitioning does have its limitations. Firstly, there should not be any global index. Second, the physical tables should be independently handled. Lastly, fewer choices are at your disposal with partition view.

12.5: Describe Three Types of File Organization

A physical file contains the actual data that is stored on the system and a description of how the data is to be presented to or received from a program. Physical files can be separated into extents. Extents are a spreadable section of disk storage space. Many database management systems store many different kinds of data in one operating system file. According to (Venkataraman, R., Topi, H. 2011) a "*tablespace* is a named logical storage unit in which data from one or more database tables, views, or other database objects may be stored." A tablespace consists of one or several physical operating system files. Thus, "Oracle has responsibility for managing the storage of data inside a tablespace, whereas the operating system has many responsibilities for managing a tablespace, but they are all related to its responsibilities related to the management of operating system files" (Venkataraman, R., Topi, H. 2011).

Because an instance of Oracle usually supports many databases for many users, a database administrator usually will create many user tablespaces, which helps to achieve database security, since the administrator can give each user selected rights to access each tablespace. As stated by Venkataraman, R. and Topi, H. (2011) "each tablespace consists of logical units called *segments* consisting of one table, index, or partition, which, in turn, are divided into extents .... these consist of a number of contiguous *data blocks*, which are the smallest unit of storage". Each table, index, or other so-called schema object belongs to a single tablespace, but a tablespace may contain one or more tables, indexes, and other schema objects. Physically, each tablespace can be stored in one or multiple data files, but each data file is associated with only one tablespace and only one database.

Modern database management systems have an active role in managing the use of the physical devices and files on them; for example, the allocation of schema objects (e.g., tables and indexes) to data files is typically fully controlled by the DBMS. A database administrator does have the ability to manage the disk space allocated to tablespaces and a number of parameters related to the way free space is managed within a database. Because this is not a text on Oracle, we do not cover specific details on managing tablespaces; however, the general principles of physical database design apply to the design and management of Oracle tablespaces, as they do to whatever the physical storage unit is for any database management system

File Organization

A "*file organization* is a technique for arranging the records of a file on secondary storage devices" (Venkataraman, R., Topi, H. 2011). With modern relational DBMS it is not necessary to design file organizations, but you are to be allowed to select an organization and its parameters for a table or physical file. In choosing a file organization for a particular file in a database, consider seven important factors: fast data retrieval, high throughput for processing data input and maintenance transactions, efficient use of storage space, protection from failures or data loss, minimizing need for reorganization, accommodating growth, and security from unauthorized use.

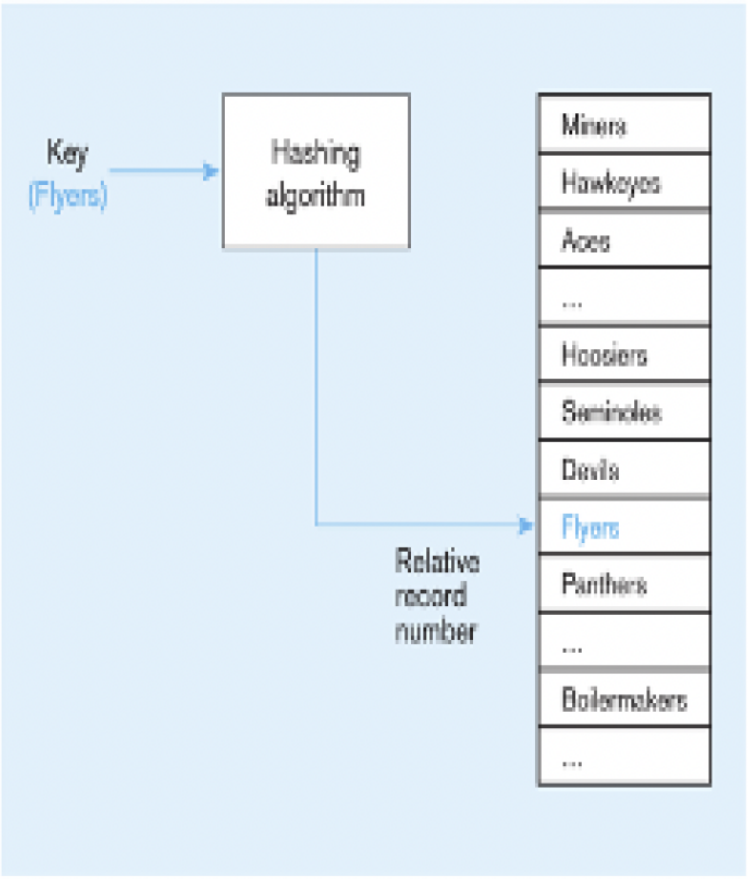
SEQUENTIAL FILE ORGANIZATIONS

In a sequential file organization, the records in the file are stored in sequence according to a primary key value. To locate a particular record, a program must normally scan the file from the beginning until the desired record is located. A common example of a sequential file is the alphabetical list of persons in the white pages of a telephone directory. A comparison of the capabilities of sequential files with the other two types of files can be seen in Figure 1.3. "Because of their inflexibility, sequential files are not used in a database but may be used for files that back up data from a database" (Venkataraman, R., Topi, H. 2011).

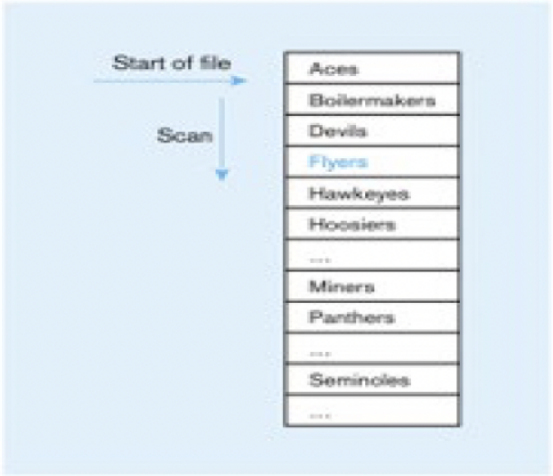
INDEXED FILE ORGANIZATIONS

In an index file organization, records are stored either sequentially or nonsequentially, and an index is created that allows the application software to locate individual records. "A card catalog in a library, an *index* is a table that is used to determine in a file the location of records that satisfy some condition" (Venkataraman, R., Topi, H. 2011). Each index entry matches a key value with one or more records. An index can point to unique records or to potentially more than one record, and an index that allows each entry to point to more than one record is called a *secondary key* index. Secondary key indexes are important for supporting many reporting requirements and for providing rapid ad hoc data retrieval. An example would be an index on the ProductFinish column of a Product table. Because indexes are extensively used with relational DBMSs, the choice of what index and how to store the index entries matters greatly in database processing performance.

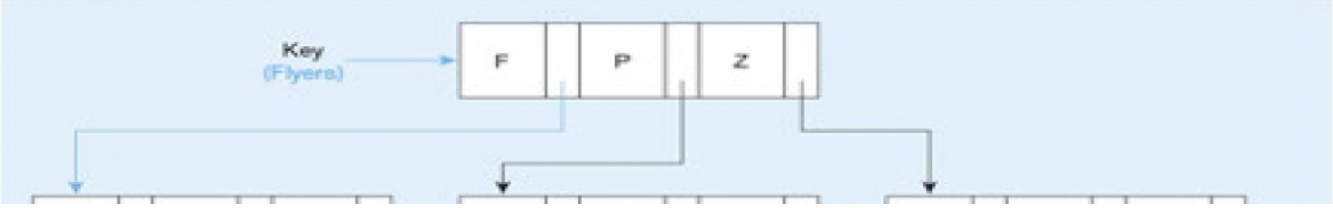
(c) Hashed



(a) Sequential



(b) Indexed



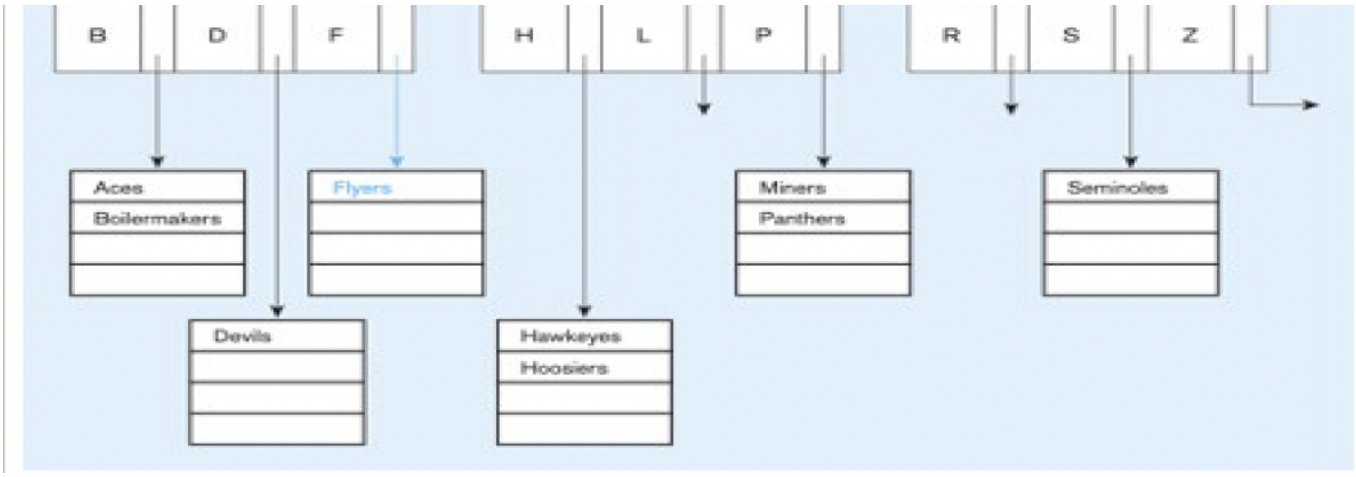


Figure 12.5.1: Three Types of File Organizations (Venkataraman, R., Topi, H. (2011))

When the address of a record within a file is to be determined or computed, the technique of hash file organization may be considered and implemented as an algorithm or function. A hash algorithm is an algorithm that takes an input of random size and proceeds to transform the input such that the hash result is an output of fixed length. Once the output is determined or computed, the hash result is irreversible, meaning that the algorithm can only process data in one direction. The use of hashing algorithms is commonly found in databases for practically any website that requires a password to login to an account and is illustrated in Figure 12.5.2

Figure 12.5.2:

Hashing Algorithm, by jscrambler. Copyright 2020 by jscrambler

In other hashing algorithms, the primary key value of a record is typically divided by a prime number that is suitable for use and the remainder of the divided value is used as a relative storage location. Due to limitations in which only one key is used for storage retrieval through hashing, hashing and indexing are used in combination to address this issue. According to Jeffrey A. Hoffer, Venkataraman Ramesh, and Heikki Topi, a hash index table uses hashing to map a key into a location in an index (sometimes called a scatter index table), where there is a pointer (a field of data indicating a target address that can be used to locate a related field or record of data) to the actual data record matching the hash key (Hoffer et al., 2015).

In an effort to preserve memory space, database management systems may allow for several rows of related tables to be joined together and store the same amount of units of storage (data block). An example of this is seen when a common primary key between related tables such as a CustomerID or ItemID are utilized to join rows of separate tables together which are related by

these two primary keys. According to Jeffrey A. Hoffer, Venkataraman Ramesh, and Heikki Topi, time is reduced because related records will be closer to each other than if the records are stored in separate files in separate areas of the disk. Defining a table to be in only one cluster reduces retrieval time for only those tables stored in the same cluster. This technique of file organization is known as clustering files and is illustrated in Figure 12.5.3

```
CREATE CLUSTER Ordering (CustomerID CHAR(25));
```

The term Ordering names the cluster space; the attribute CustomerID specifies the attribute with common values.

Then tables are assigned to the cluster when the tables are created, as in the following example:

```
CREATE TABLE Customer_T(
  CustomerID                VARCHAR2(25) NOT NULL,
  CustomerAddress           VARCHAR2(15)
  )
  CLUSTER Ordering (CustomerID);
CREATE TABLE Order_T (
  OrderID                   VARCHAR2(20) NOT NULL,
  CustomerID                VARCHAR2(25) NOT NULL,
  OrderDate                 DATE
  )
  CLUSTER Ordering (CustomerID);
```

Figure 12.5.3:

A clustering file for Ordering, by Hoffer et al., 2015, Modern Database Management 12e (E-Reader Version). Copyright 2015 by Pearson Education, Inc

When considering security to protect files from damage, types of control are useful elements of database files to use for unforeseen file corruptions. Database files are stored in a proprietary format by the database which allows for access controls over the files. Some useful procedures to consider are backups to ensure that stored data may be retrieved in the event that data may be compromised. Another technique employs the utilization of encryption to encrypt data contained within files and allow for only programs with access to decrypt them. There are two main methods of encryption: symmetric encryption and asymmetric encryption. Symmetric encryption makes use of a single key for all parties communicating and is used for both encrypting data and decrypting data. Asymmetric encryption makes use of two keys for all parties communicating where the first key is used for encryption and the second key is used for decryption.

12.6: Translate a Database Model into Efficient Structures

A majority of database manipulations demand the location of a row or a collection of rows that satisfies a condition. Given the magnitude of a database, searching for data can be quite the laborious task. Hence, using indexes can vastly increase the speed of the process and reduce the time and work. The usage and definition of indexes are a crucial spoke on the wheel of physical database design. Indexes are defined as either a primary key, secondary key, or both. It is ordinary to define an index for the primary key of a table. The index is formed of two columns: one column for the key and the other column for the address of the record that consists of the key value. In the case of a primary key, the index will only have one entry for each key value.

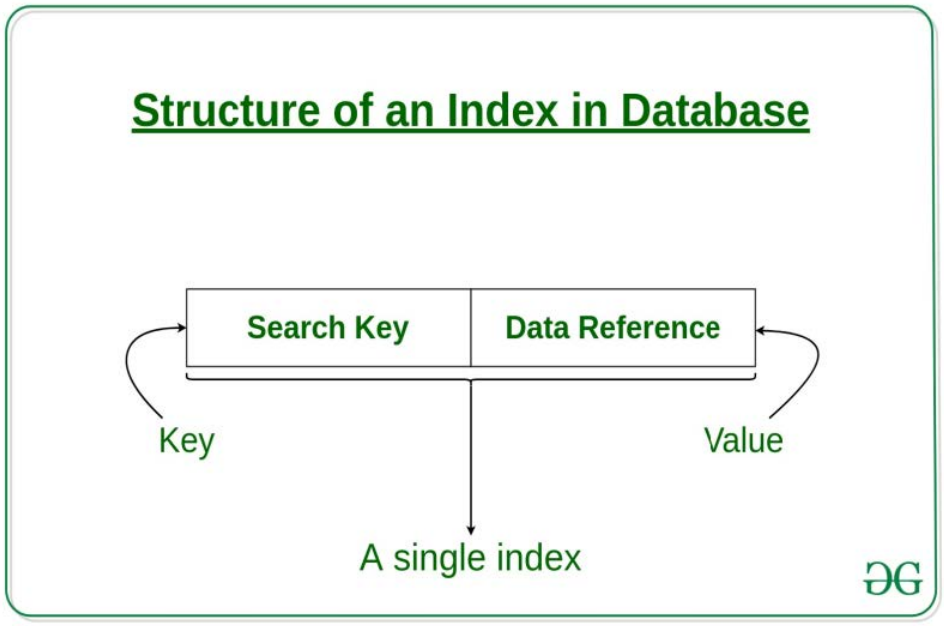


Figure 12.6.1:

Indexing in Databases, by GeeksforGeeks. Copyright 2020 by GeeksforGeeks.org

CREATING A UNIQUE KEY INDEX

The syntax to create a unique key index in SQL is "CREATE [UNIQUE] INDEX index_name ON table_name(column1, ... column_n);". The UNIQUE modifier specifies the values in the indexed columns. Creating a non-unique key index is equivalent to a secondary key index. The term UNIQUE isn't used to create a secondary key index, because values can be repeated.

WHEN TO USE INDEXES

It is important to know when to use an index and which attributes to use when creating an index. Using indexes come at the price of performance. Performance is compromised when using indexes due to the overload for maintenance for insertions, deletions, and updating records. For this reason, indexes should be utilized mainly for data retrieval (Hoffer, Venkataraman, & Topi, 2011). Here are some rules or conditions that suggest the use of indexes.

1. Indexes are a lot more efficient and practical for substantial tables.
2. Indexes are useful when there is a need to set out a unique index for the primary key.
3. Indexes are frequently used for columns that appear in WHERE modifiers of SQL commands.
4. Indexes should be used when for attributes referenced in ORDER BY and GROUP BY statements.
5. Indexes are convenient when there is diversity in the values of an attribute. For Oracle's standards, it is unproductive to use an index when an attribute has fewer than 30 values.
6. One point to keep in mind is to consider developing a compressed version of the values. Doing this will ensure that the index isn't slower to process.
7. If the index is used for finding the location of where the record will be stored, make sure the key of this index is a surrogate key to ensure the records will be fairly spread across the storage space.
8. Make sure to check the limit of indexes on the DBMS because some systems do not allow for more than 16 indexes.

9. Find a way to index attributes that have null values because rows with a null value won't be referenced.

12.6.1: Designing a Database for Optimal Query Performance

Databases are used every day, which makes it important to optimize the performance of database processing. Database processing can include adding, deleting, and modifying a database along with methods of retrieving data. Since databases have more traffic retrieving data from the database than maintenance, it is important to optimize query performance. Producing reports and ad hoc screens for users is the primary goal. The amount of work required to optimize query performance heavily relies on DBMS. Some DBMS give little control to the developer on how the database is designed and where the query can be processed. This can limit how optimized data reads and writes are in the database. However, other systems give complete control to the developer and require a lot of setup work. Since workloads often vary, it is difficult to reach peak performance from a database unless the workload is focused. “For example, the Teradata DBMS is highly turned for parallel processing in a data warehousing environment. In this case, rarely can a database designer or query writer improve on the capabilities of the DBMS to store and process data” (Hoffer, Venkataraman, & Topi, 2011). This situation is rare; thus, it is important as a database designer to consider ways to improve the processing capabilities of the database.

Since computer architecture has changed greatly over the years, the use of multiple processors in database servers has become standard. Symmetric multiprocessor (SMP) architecture is commonly used in database servers to allow parallel processing. DBMS that use parallel query processing break up a query such that it can be processed in parallel by different processors. These partitions must be defined before the query by the database designer. An example of instructing Oracle to execute a query using parallel processing is below:

ALTER TABLE ORDER_T PARALLEL 3; *(Hoffer, Venkataraman, & Topi, 2011)*

Each table needs to be finely tuned to best support parallelism and can undergo multiple changes to find the best performance. Schumacher reported, “on a test in which the time to perform a query was cut in half with parallel processing compared to using a normal table scan. Because an index is a table, indexes can also be given the parallel structure, so that scans of an index are also faster.” Schumacher also reported an example of parallel processing reducing the time of creating an index from seven minutes to five seconds (Hoffer, Venkataraman, & Topi, 2011). Parallel processing not only improves the time of table scans but also can be used on joining tables, grouping query results, sorting, deleting, updating, and insertion. Oracle requires the number or virtual parallel database servers to be defined beforehand. Once the servers are defined, the processor will decide what is the best use of parallel processing for that specific command.

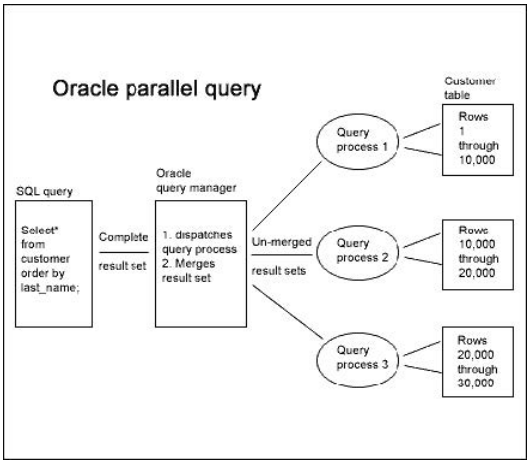


Figure 12.6.1.1: Oracle Parallel Query. (Burleson Consulting, 2014)

Often the designer creating the query has information to better optimize the query that the query module in the DBMS does not. In most relational DBMs, the optimizer’s plan for processing the query can be known by the designer before actually running the query. This is done through the command EXPLAIN or EXPLAIN PLAIN, which display all the information about the optimizer’s plans to process the query. The query optimizer makes its decision on how to process the query by looking at data from each table such as average row length or the number or rows. You can submit multiple EXPLAIN commands with a query written in different ways to see if the optimizer predicts different performance. That then allows you to find the best performance and submit that for actual processing. With some DBMs you can force the optimizer to take different steps or use resources other than what the

optimizer thinks is the best performance. The clause /**/ can be used to override what the query determines is the best way to process the query.

12.7: Concise Summary

Physical database design translates the logical data model into a set of SQL statements that define the database. For relational database systems, it is relatively easy to translate from a logical data model into a physical database (“Physical Database Design,” n.d.). The design process is a collaborative one where preliminary information is collected to determine technical properties of the database, such as response times and data security. In the database design process, fields are heavily emphasized to determine the corresponding attributes and logical data model. Often, a database design is concerned with data partioning of large tables, which are partitioned into smaller, individual tables, allowing for more efficient processing of data and database performance overall.

The database design process is also concerned with file organization of physical files. File organization is responsible for managing records of a file on a secondary storage device. Physical file organization is typically divided into three types of file organization: Sequential, Indexed, and Hashed file organization. In sequential file organization, records on an arbitrary file are stored in sequence onto a secondary storage device based on primary key values. In indexed file organization, records may be stored sequentially or non-sequentially and an index is created to assist with locating individual files. In hashed file organization, a hash algorithm or function is created when the address of a record within a file is to be determined or computed. Efficient structures are a necessary component of database design and are achieved through the use of indexes for database models. The indexes are comprised of primary keys and are convenient in different scenarios with the trade-off of potentially decreased database performance.

Databases are an integral aspect for the successful operations of businesses, and other fields that depend on optimal technology to process information. Physical database design is a process that requires careful fundamental planning in order to realize optimization in technology.

12.8: Extended Resources

1. This is a video lecture from the University of Washington by Grey Hay, about the physical database design methodology. This video goes into extreme detail about physical database design from the ground up and how the methodology is implemented. https://www.youtube.com/watch?v=S98_8HalY5Q
2. This video talks about the oracle database security in a broad approach, mainly focused in Europe. The video discusses important security topics from current database security laws, benefits, history, risks and many more topics. https://www.youtube.com/watch?v=GXF3T4g2tJg
3. This video by Kimberly Tripp discusses why physical database design matters. Kimberly discusses the importance of good design and also how poor design and can lead to major performance issues. https://www.youtube.com/watch?v=H-jPsp2QlT0
4. Lightstone, S., Nadeau, T., & Teorey, T. J. (2007). Physical Database Design : The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More. Morgan Kaufmann.
5. Erickson, J., & Siau, K. (2009). Advanced Principles for Improving Database Design, Systems Modeling and Software Development. IGI Global.
6. Carmel-Gilfilen, C. (2013). Bridging security and good design: Understanding perceptions of expert and novice shoplifters. Security Journal, 26(1), 80–105. https://doi.org/10.1057/sj.2011.34
7. Burtescu, E. (2009). Database Security - Attacks and Control Methods. Journal of Applied Quantitative Methods, 4(4), 449–454.

12.9: References

CloudGirl, & CloudGirl. (2017, March 26). Data Partitioning: Vertical Partitioning, Horizontal Partitioning, and Hybrid Partitioning Project Update #3. Retrieved from http://cloudgirl.tech/data-partition...-partitioning/

DigitalOcean. (2019, October 28). Understanding Database Sharding. Retrieved from https://www.digitalocean.com/communi...abase-sharding

Database VLDB and Partitioning Guide. (2016, July 20). Retrieved from https://docs.oracle.com/database/121...htm#VLDBG00225

Lesov, P. (2010). Database Security: A Historical Perspective. ArXiv, abs/1004.4022.

Quick Base. (n.d.). A Timeline of Database History. Retrieved from https://www.quickbase.com/articles/t...tabase-history

Watt, Adrienne. (n.d.). Chapter 13 Database Development Process. Retrieved from https://opentextbc.ca/dbdesign01/cha...pment-process/

What is Entity Relationship Diagram? (ERD). (2020). Retrieved from https://www.visual-paradigm.com/guid...nship-diagram/

Hoffer, A. J., Venkataraman, R., Topi, H. (2011). *Modern Database Management 10e*[E-Reader Version]. Retrieved from https://www.amazon.com/Modern-Databa.../dp/0136088392

Burleson Consulting. (2014, April 23). Oracle parallel query tips. Retrieved March 28, 2020, from http://www.dba-oracle.com/art_opq.htm

Chung, C. (n.d.). Introduction to Hashing and its uses. 2BrightSparks. https://www.2brightsparks.com/resour...-its-uses.html

Physical Database Design. (n.d.). Retrieved from http://ewebarchitecture.com/web-data...atabase-design

CHAPTER OVERVIEW

13: Data Warehouse

13.1: Introduction and Background

A data warehouse an element of computing systems considered a core part of many businesses. They are mainly used for data analysis. A data warehouse combines information from multiple different places electronically into one single all-inclusive database. For example, companies may combine sales from the cash registers in a store, online sales, and company-to-company orders all together as one database. Data warehouses are made for this purpose.

A data warehouse is a collection of data that is non-updateable, time-variant, integrated, and subject-oriented that is used in the support of decision-making management processes and business intelligence. The meaning of each of these terms is important for defining what a data warehouse is. The term non-updateable means the information in the data warehouse can be refreshed and loaded from operational systems, but cannot be updated by end users. The term time-variant means the information in the data warehouse contains a dimension of time so that they can be used to study changes and trends. The term integrated refers to the data in the data warehouse are characterized using consistent formats, encoding structures, related characteristics, and naming conventions gathered from many internal or external systems of record. The term subject-oriented means the data warehouse is organized around key subjects, or higher-level entities, of the enterprise. These major subjects may include students, patients, products, or time. The data warehouse is not a mere consolidation of all of the operational databases of an organization. Because of a data warehouse's focus on business intelligence, time-variant data, and external data, a data warehouse is distinct from a simple database. Data warehousing is a process where organizations build and maintain data assets. Making successful a data warehouse requires proven practices of data warehousing, strong organizational commitment, sound project management, and making the right technology decisions.

Putting all this information into one place makes it a lot easier for people to look at and evaluate that information. It also makes the jobs of data miners and data analysts much easier, since they can find everything all in one place. It can even help companies realize higher profits and make sales a lot easier.

Data warehouses have many other advantages that come from combining data from several locations into one central database. Data warehouses prevent database isolation sometimes caused by running very long queries. Even if source transaction systems fail, the data warehouse will still maintain data history. They also much more consistently provide data to the end user, helping to prevent mistakes. A data warehouse can also reorganize all of the information found to make it much easier to read for analysts and to help them more clearly understand the content. Reorganizing the information properly can also increase the performance for queries very noticeably. It can even make the queries much easier to construct in many instances.

Although there are many positives to data warehouses, they have their cons. They can often make certain employers have to do even more work. This would include gathering data from other workers and customers to help maintain the warehouse. Another common issue with data warehousing is the cost-to-benefit ratio. This is a huge project for businesses, and it could take up thousands of hours to establish. Sometimes it just isn't used enough to make up for the enormous implementation costs. This is before even mentioning the huge amount of money it takes to maintain the data warehouse and keep it up and going. You also have to update it and do a lot of maintenance for it as your business grows which can be a pain at certain times. The price for a simple one-terabyte data warehouse with around one hundred thousand queries per month can cost as much as four hundred and sixty-eight thousand dollars a year. For big companies, this might not be a huge deal, but this is definitely not affordable for smaller businesses. Businesses must carefully consider whether they truly need a data warehouse.

The privacy and security of your data for a data warehouse is also only as good as your cloud services, so for smaller businesses, data may not be safe. If a corporation's warehouse is accessed and leaks customer information, it could be terrible for the company. It would give out possibly customers' credit card information and damage the company's reputation. Although there are many things that can go wrong with a data warehouse, if you know how to use one correctly the positives outweigh the negatives.

There are many ways to create a data warehouse and several different ways to upload all the information quickly. First, consider all of the data from each individual source: for example, all data from the cash registers, online sales, and company sales. Next, remove all of the redundant data, so there is no repetition to corrupt the information provided. Lastly, reorganize all of the data obtained into a consistent format so that it can all be queried. A query is a request for information from a specific table inside of a database.

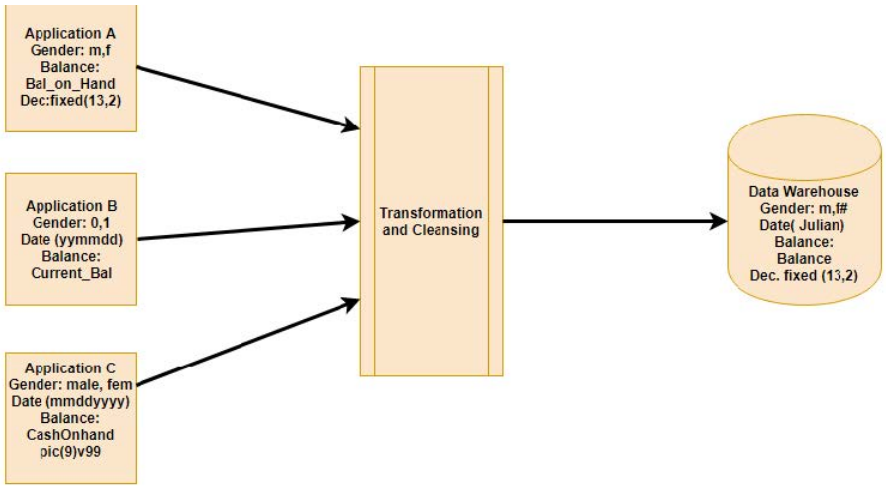


Figure 13.1.1: Sample Data Warehouse (North, 2021)

There are three main types of data warehouses. First is the Enterprise Data Warehouse, which is a central database. It has decision support services. Its approach is unified for keeping the data in the warehouse organized, which helps present the data easier. Users can also sort the data more easily according to category.

Next, we have the Operational Data Store. The Operational Data Store (ODS) is nothing but data being stored for when there is no organization needed. ODS is refreshed in real time. It is highly recommended to use ODS over the other two methods for more simple things like storing employees' hours or pay to keep things simpler.

The last form of data warehouse is a Data Mart. Data Marts are basically just a subset of the actual data from the data warehouse. It was made precisely for sales and financing businesses. If the data mart is self-regulating, then data can be composed straight from the founding source. Basically, any data warehouse that is mostly used for money-related purposes should be created as a data mart.

There are multiple different components of a data warehouse. First, the Load Manager is also known as the "front component" of the warehouse. It is called the front component because it works with the main extraction of data to load it all into the data warehouse. Next we have the Warehouse Manager, which is used to manage the data inside of the warehouse. It helps to investigate the data inside the warehouse to make sure it stays consistent throughout. The Query Manager is another key component to the warehouse, its main job being to manage the queries for the warehouse. Since it manages the queries of the warehouse it is also known by some as the "backend component." Lastly, we have the end-user access tools, which are assembled into five different categories. The first category is responsible for reporting the warehouse data to management. The second category is used to store the tools for the queries. The third category is for tools that help application development. The fourth category is for EIS tools. Lastly, the fifth category is for OLAP and data mining tools.

Next, we discuss the architecture of a data warehouse. A simple data warehouse design is where all the different forms of data are stored in the central repository of the date warehouse. The repository receives information from each source and then is used for data analysis and mining. Next we have simple architecture design with a staging area included. All of the non-required data must be prepared before going to the warehouse, because only the necessary data is wanted. The staging part of the warehouse prepares data before it actually goes into the data warehouse. The next form is called hub-and-spoke warehousing. It works similarly to a staging area, but instead adds a data mart after the central repository to further customize the data warehouse, which may be needed in certain types of business applications. After the data is sorted, cleaned, and ready to use, it is moved into the responding data mart. Lastly, sandboxes are very private and protected data stores, which make them very efficient and allow new ways of analyzing the data sets without having to follow the basic rules of the data warehouse.

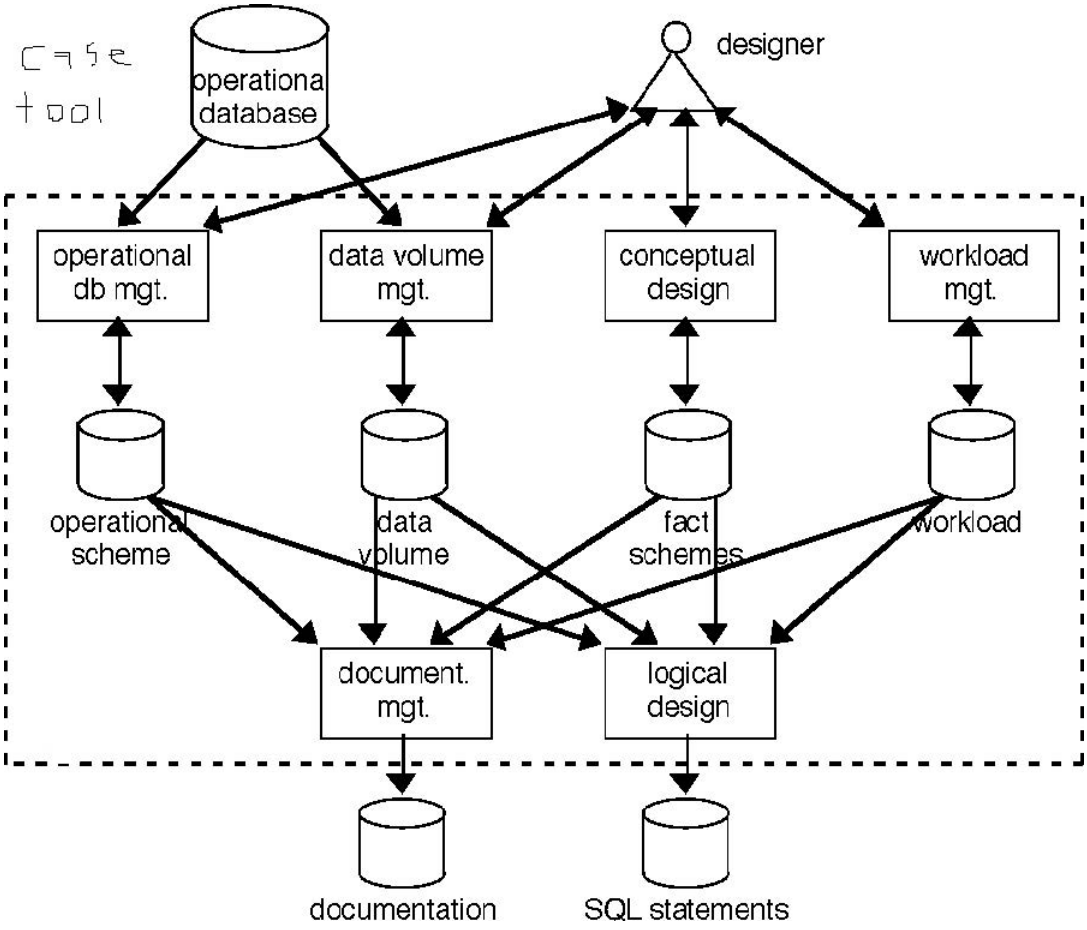


Figure 13.1.2: Database Diagram (North, 2021)

Many tools and utilities are required to effectively use a data warehouse. These can be very useful when navigating the warehouse. Data Extraction is used to get data from several different outside sources and present them. Data Cleaning tools can also be very useful for finding the errors inside of the data set, even correcting them for the user. This can also be used to get rid of duplicate data. Data Transformation tools are used for converting the format type of the data sets. For example, if the date is in a legacy format, it can easily be transformed to warehouse standards. Data Loading is used mostly for sorting and checking reliability of data. The refresh tool is just for simply updating the sources into the warehouse format.

There are three main ways to test a data warehouse. The first method is called Unit Testing and consists of testing each individual part of the warehouse in turn. This test is usually done by the developer. Next is Integration Testing, in which different parts of the application are formed into one, then tested compared to the number of inputs put in. This method is mainly used to make sure all of the parts work together after combining them. Lastly, System Testing tests all the data in the entire data warehouse together. This the final check to make sure that everything is working together as a final product. This kind of testing is the testing done by the whole team.

Most organizations have a plethora of data but cannot access it effectively and efficiently. There are multiple reasons why there is a knowledge gap between so many organizations. The first reason is because organizations have a fragmented way in which they created their information systems and databases. Because these organizations have time constraints and resource constraints, they have been developing information systems one at a time. This creates inconsistent and uncoordinated databases. Databases are usually based on different purchased applications, software platforms, and hardware that may have been brought in from a variety of organizational acquisitions, mergers, and reorganizations. From these circumstances, it can be very difficult or nearly impossible for managers to find and use accurate information that has to be synthesized across different systems.

Most systems are created to help operational processing, in which there is little to no thought is given to information and analytical tools needed to make decisions. Operation processing, also known as transactional processing, takes, stores, and changes data to help daily operations of an organization. It usually focuses on optimizing access to a small set of data that relates to a specific transaction—for example, a customer, an order, and associated product data. Information processing is an analysis of data or other

forms of data to help decision making. Large swaths of data are needed to derive information, such as sales of total products, across several years, from all sales regions. Systems that are mostly developed internally or bought from outside vendors were designed to help operational processing, and very little thought is given to information processing.

The next trend is that multiple systems are not synchronized. It may be difficult, maybe impossible, to make separate databases consistent. Even if the metadata are controlled and created by only one data administrator, the values of data for the same attributes may not agree. That is because of a difference in update cycles and of the separate places where the same data is captured for all systems. Therefore, to get a single view of the organization, the data from separate systems has to be periodically separated and synchronized into one additional database.

Many organizations use some form of a balanced scorecard-metrics, which show organization results in customer satisfaction, product quality, human, financial, and other terms simultaneously. To make sure that this multidimensional view of the organization shows consistent results, a warehouse of data is necessary. When questions rise on the balanced scorecard, analytical software that the data warehouse may be used to "slice and dice," "drill down," visualize, and in many other ways mine business intelligence. The next trend is customer

relationship management. This is when organizations in all sections are realizing there is value in having a total picture of interactions with customers across all of the touch points. For example, for a bank, the touch points are ATMs, online banking, electronic fund transfers, tellers, loans, and investment portfolio management, all supported by consolidated operational systems. Therefore, without having a data warehouse, a teller does not know to try to cross-sell a customer one of the bank's mutual funds if the large, atypical automatic deposit transaction shows up on the teller's screen. To have a total picture of the activity by a given customer requires a consolidation of data from many different operational systems.

Another trend is supplier relationship management. The process of managing the supply chain has become critical in raising product quality and reducing costs for many organizations. Organizations desire to make strategic supplier partnerships that are based on a total picture of activities and suppliers, from billing, to the meeting of delivery dates, to support, to pricing, to quality control. Data about these different activities may be locked inside separate operational systems: for example, accounts payable, shipping and receiving, maintenance, and product scheduling. ERP systems also have improved this situation by bringing much of this data into a single database. However, many ERP systems tend to be designed to optimize operational, instead of informational or analytical, processing.

There is a need to separate informational and operational systems. An operational system must be a system which is used to run a business in actual time, based on current data. Some examples of operational systems are sales order, patient registration systems, and reservation systems. Operational systems have to process large volumes of relatively simple write/read transactions and provide fast responses. Operational systems can also be called systems of record. Informational systems were designed to support decision making based on historical prediction and point-in-time data. Also, they were designed for data-mining applications or complex queries. Some examples of informational systems would be customer segmentation, systems for sales trend analysis, and human resources planning.

The main differences between informational and operational systems are defined by six key characteristics. These characteristics are primary purpose, type of data, primary users, scope of usage, design goal, and volume. For the first characteristic, the primary purpose of an operational system is to run the business on current basis, but for informational systems, the primary purpose is to support managerial decision making. The second characteristic is type of data. The type of data that operational systems use is a current representation of the state of the business. The type of data that informational systems use is historical point-in-time, snapshots, and predictions. The third key characteristic is the primary user. The primary users of operational systems are administrators, clerks, and salespersons. The primary users of informational systems are managers, customers, and business analysts. The fourth key characteristic of the differences is the scope of usage. The scope of usage for operational systems is planned, narrow, and simple updates and queries. The scope of usage for informational systems is ad hoc, broad, and complex queries and analysis. The fifth key characteristic the design goal. The design goal of operational systems is that performance is throughput and the availability. The design goal of informational systems is the ease of flexible access and use. The sixth key characteristic of the difference is volume. The volume of operational systems consists of multiple constant updates and queries on one or a few table rows. The volume of informational systems consists of a periodic batch of updates and queries requiring many or all rows.

This shows that the two types of processing have vastly different characteristics in almost every category of comparison. What is particularly noticeable is that they have very different communities of users, because operational systems are used by salespersons, administrators, clerks, and others who need to process business transactions, while informational systems are used by executives,

business analysts, managers, and increasingly by customers who are in search of status information or those who are decision makers.

The need to separate informational systems and operational systems is based on three key factors. The first factor is that a data warehouse tries to centralize data that is scattered throughout different operational systems and makes them easily available for decision support applications. The second factor is that a properly designed data warehouse can add value to data by increasing their quality and consistency. The third factor is that a separated data warehouse negates much of the contention for resources that would result when informational applications are competing with operational processing.

The success of data warehousing is not guaranteed. Data warehousing projects fail about forty percent of the time. This is because warehousing data is complex and requires cooperation and synchronization across an organization. For example, when errors become discovered in data that is loaded into the warehouse, the correct place to fix the errors will be in the source systems so bad data is not loaded again later. But the errors might be acceptable in the source system or even not considered errors by the unit of business for the source system. Many professional organizations sponsor yearly award programs to show the best data warehousing practices. The Data Warehousing Institute Awards is one of the most prestigious yearly award programs. There are many impressive winners from this award program. One of these winners is Continental Airlines, which won the award for the best enterprise data warehouse. The continental data warehouse had an actual-time architecture and had automated data transformations. What this did was simplify consolidated data from various source systems. A cross-unit-business system steering committee developed standard data definitions, also known as metadata. The uses of and changes to the warehouse have to be justified by profitability and revenue projections.

Ease of access is a necessity for companies that use collected data for their business. Many businesses function by using user data to create value for themselves or for other companies. Companies such as Google use user search data to tailor ads to the user's interest. The ease of use provided by data warehouse interfaces allows for companies to perform their tasks more efficiently than if they had to access the data by hand. They do this by using an algorithm that queries their database through the interface and then analyzes the user search history to show ads to the user that might provoke a response. Some companies just collect the data and store it, selling it to advertising companies who use the data to send relevant information to users who fit certain criteria. These are just a couple of examples of what is possible with data warehouse queries and analysis.

The architecture of a data warehouse is broken down into three levels; Data Sources, Warehouse, and users. The user-level of the data warehouse is where all of the analysis, reporting, and data mining happens. This is the level where most businesses operate, searching for and using the data for their intended purposes. The warehouse level is where the metadata, summary data, and raw data are stored. This allows for usage from the user level and for data deposits from the data sources level. This is the level that receives the most traffic because it handles the movement of data and the resorting after the reception and removal of data. The data sources level is where the data is generated, either through digital or real-world collection, and then stored in the warehouse level.

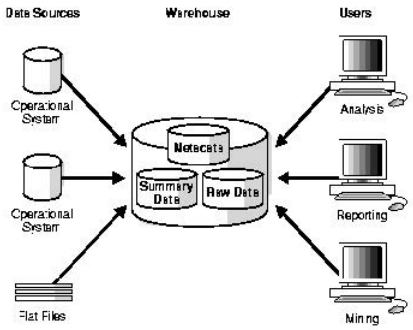


Figure 13.1.3: Example of a Data Warehouse. (North, 2021)

Each piece of data is categorized based on various criteria, depending on the type of data. For example, if the data came from a financial transaction, it would be sorted based on date, amount, sender, receiver, and status. The data is then put with data that has the same criteria, typically sorted by date. Once the data is stored it can be easily retrieved and aggregated into a list of data that all matches a list of requirements for analysis. This allows for the data to be easily accessed and sorted.

These warehouses hold information that has become important to the function of most companies, schools, banks, militaries, and governments. Data warehouses store information about equipment purchases, student class lists, transaction history, payment history, payroll, inventory listings, and employee information. Without efficient, organized ways to store all this information, companies and organizations wouldn't be able to function. Stores wouldn't be able to keep track of their stock so that they can

remain stocked. Companies wouldn’t be able to manage and maintain payroll for their employees, keep track of what equipment is at what location, or what businesses placed orders for how much. Data storage, management, and organization have grown in importance to the point where most of modern society is dependent on the storage and reproduction of data.

13.2: Concise Summary

In today's life, data is everywhere and a part of everything. Data is generated whenever you make a purchase, browse the internet, watch a video, visit a web page, or see an ad. That data is collected by your internet service provider, the browser you use, and the web pages you visit. Once all the data is collected, it is stored in data warehouses. These data warehouses manage and maintain the data, storing it in a format that is easily accessible and searchable. The ease of access makes the data easier to use for analysis and comparison.

Data warehousing is an extremely important component of modern business, forming the technical backbone of any given company. Data warehouses are used to store, manage, and access the data that the companies use. Data warehouses process many different types of data, varying from purchase data to internet ad data. Data warehouses operate with the same general structure. The data sources put the data into the warehouse system, and the users access the warehouse through the user level. The warehouse system allows for both the data sources and the users to access the data, helping to resolve some of the information gap between user and source. Keeping track of all this data requires a particular setup in order to be efficient.

Data warehouses are created in a very particular way so that users and data sources can interact with it at the same time. The physical structure of data warehouses consists of a series of computers connected together in such a way so that they share storage space, allowing them to distribute the data for easier access and storage. This system of distributed storage allows for the users to access the data at the same time as the data sources, in addition to having multiple users able to access the data simultaneously. These data warehouses also make it easier to combine multiple data sets into one place, sorting the data sets into a format that is easily searchable and accessible.

While data warehouses are useful systems, allowing users to collate data and search it to be processed, implementing a data warehouse is a very big project. Creating a data warehouse requires a lot of time, money, and resources in order to do properly. A company will have to pay for the computer hardware and networking as well as having to pay employees or contractors to set up the hardware. Once the hardware is set up, the company has to pay their employees to set up the system before it can be used. This results in a time span where they are paying for a system that is not providing any return until it is complete. Depending on the size of the warehouse, this can result in loss of profit versus the cost of set up.

Data warehouses are an exercise in cost-benefit analysis in most situations, usually ending up being more beneficial in the long run. The utility of a system that automatically sorts the data stored within based on input criteria is quite valuable. Once the data warehouse is set up, it ends up resolving a lot of problems that come from not having the sorted collection of data. It allows for employees across the company, and across the country, to access the same data and collaborate in the data analysis. The collaboration ends up saving the company time and effort, eventually resulting in more work done for less effort. All in all, data warehouses expand the availability of data and make data manipulation significantly easier. Data warehouses are a necessity for large companies that is costly to set up in the short term but is well worth the effort in the long term.

13.3: Extended Resources

1. Data Warehouse Tutorial For Beginners | Data Warehouse Concepts | Data Warehousing | Edureka. Data Warehousing & BI Training: https://www.edureka.co/data-warehousi... This Data Warehouse Tutorial For Beginners will give you an introduction to data warehousing and business intelligence. You will be able to understand basic data warehouse concepts with examples. The following topics have been covered in this tutorial:
    - What Is The Need For BI?
    - What Is Data Warehousing?
    - Key Terminologies Related To DWH Architecture:
    - OLTP Vs OLAP b. ETL c. Data Mart d. Metadata
    - DWH Architecture 5. Demo: Creating A DWH
    - https://www.youtube.com/watch?v=J326LIUrZM8
2. Data Warehouse Concepts | Data Warehouse Tutorial | Data Warehouse Architecture | Edureka

    Data Warehousing & BI Training: https://www.edureka.co/data-warehousi...

    This tutorial on data warehouse concepts will tell you everything you need to know in performing data warehousing and business intelligence. The various data warehouse concepts explained in this video are:

    - What Is Data Warehousing?
    - Data Warehousing Concepts:
    - OLAP (On-Line Analytical Processing)
    - Types Of OLAP Cubes
    - Dimensions, Facts & Measures
    - Data Warehouse Schema
    - https://www.youtube.com/watch?v=CHYPF7jxlik
3. What is a Data Warehouse - Explained with real life example | data warehouse vs database (2020)

    About this video - In this video, we will understand what a data warehouse is using a very simple really life example. A data warehouse is nothing more than a storeroom of your house. Data warehouse stored all your enterprise data into a centralized location and hence called EDW (Enterprise Data Warehouse). So, in this video we will cover

    - What is a Warehouse?
    - Real-life example to explain datawarehouse ?
    - Various data warehousing stages
    - Characteristics of a datawarehouse
    - https://www.youtube.com/watch?v=jmwGNhUXn_o
4. Microsoft Azure Analytics - Azure Synapse Analytics is a limitless analytics service that brings together data integration, enterprise data warehousing, and big data analytics. It gives you the freedom to query data on your terms, using either serverless or dedicated resources—at scale. Azure Synapse brings these worlds together with a unified experience to ingest, explore, prepare, manage, and serve data for immediate BI and machine learning needs.
    - https://azure.microsoft.com/en-us/se...pse-analytics/
5. What is Data Warehouse-as-a-Service?
    - Data Warehouse-as-a-Service (DWaaS) is a modern solution to address the data management challenges of today's companies. Data is critical to how modern companies operate, from providing actionable analytics and insights to fueling digitally transformed business processes.
    - Companies generate tremendous amounts of data each day, but to translate this resource into value, a company needs a place to aggregate, store, organize and analyze the data – that is a data warehouse. As one might imagine, data warehouses can be quite large and costly to build and maintain. Data Warehouse-as-a-Service addresses this challenge by providing the full-featured capabilities companies need, without much of the administrative overhead.
6. A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management s decision-making process.– W. H. Inmon

- https://www2.cs.sfu.ca/CourseCentral...ousing%202.pdf

7. Data Warehouse: From Architecture to Implementation - Data warehousing is one of the hottest topics in the computing industry today. For business executives, it promises significant competitive advantage for their companies, while information systems managers see it as the way to overcome the traditional roadblocks to providing business information for managers and other end users. With the publication of this book comes the most comprehensive, practical guide to designing, building, and implementing a data warehouse on the market today. Barry Devlin - one of the world's leading experts on data warehousing - is also one of the first practitioners in this area. In this book, he distills the insights and experiences gained over 10 years of designing and building data warehouses.
    - https://dl.acm.org/doi/book/10.5555/548222
8. Building the data warehouse – Author: Stephen R. Gardner Publication: Communications of the ACM September 1998
    - https://dl.acm.org/doi/abs/10.1145/285070.285080

13.4: References

Hoffer, J. A., Ramesh, V., & Topi, H. (2011). *Modern database management*. Upper Saddle River, NJ: Pearson.

Burnside, K. (2017, November 21). The Disadvantages of a Data Warehouse. Retrieved from https://smallbusiness.chron.com/disa...use-73584.html

Data Warehousing - Concepts. (n.d.). Retrieved from https://www.tutorialspoint.com/dwh/dwh_data_warehousing.htm

Data Warehousing Concepts. (n.d.). Retrieved from https://docs.oracle.com/cd/B10500_01...20/concept.htm

Data Warehousing. (n.d.). Retrieved from https://investinganswers.com/dictionary/d/data- warehousing

Data warehouse. (2020, February 7). Retrieved from https://en.wikipedia.org/wiki/Data_warehouse#Benefits

2017, September 27. The True Cost of Building a Data Warehouse. Retrieved from https://www.cooladata.com/cost-of-building-a-data-warehouse/

Data Warehouse Architecture, Concepts and Components. (n.d.). Retrieved from https://www.guru99.com/data-warehouse-architecture.html

Golfarelli, M., & Rizzi, S. (1970, January 1). Figure 1 from WAND: A CASE Tool for Data Warehouse Design: Semantic Scholar. Retrieved from https://www.semanticscholar.org/pape...ata-Warehouse-Design-Golfarelli-Rizzi/4eef34cdd2505cb5bcb553496ab654fa0cd1c475/figure/0

Pearlman, S. (n.d.). What is a Data Warehouse? - Talend. Retrieved from https://www.talend.com/resources/what-is-data-warehouse/

Walls, D. (2019, January 15). 7 Steps to Data Warehousing. Retrieved from https://www.itprotoday.com/sql-server/7-steps-data-warehousing

What is a Data Warehouse? (n.d.). Retrieved from https://www.oracle.com/database/what-is-a-data-warehouse/

What I Data Warehousing? Types, Definition & Example. (n.d.). Retrieved from https://www.guru99.com/data-warehousing.html#4

CHAPTER OVERVIEW

14: Virtual Desktop and Implementing SQL Queries

14.1: Introduction and Background

14.2: Connect to a virtual desktop

14.3: Constructing a Database

14.4: Implementing SQL Queries

14.5: Concise Summary

14.6: Extended Resources

14.7: References

Index

Glossary

Sample Word 1 | Sample Definition 1

Detailed Licensing

Overview

Title: Introduction to Database Systems

Webpages: 128

All licenses found:

- Undeclared: 100% (128 pages)

By Page