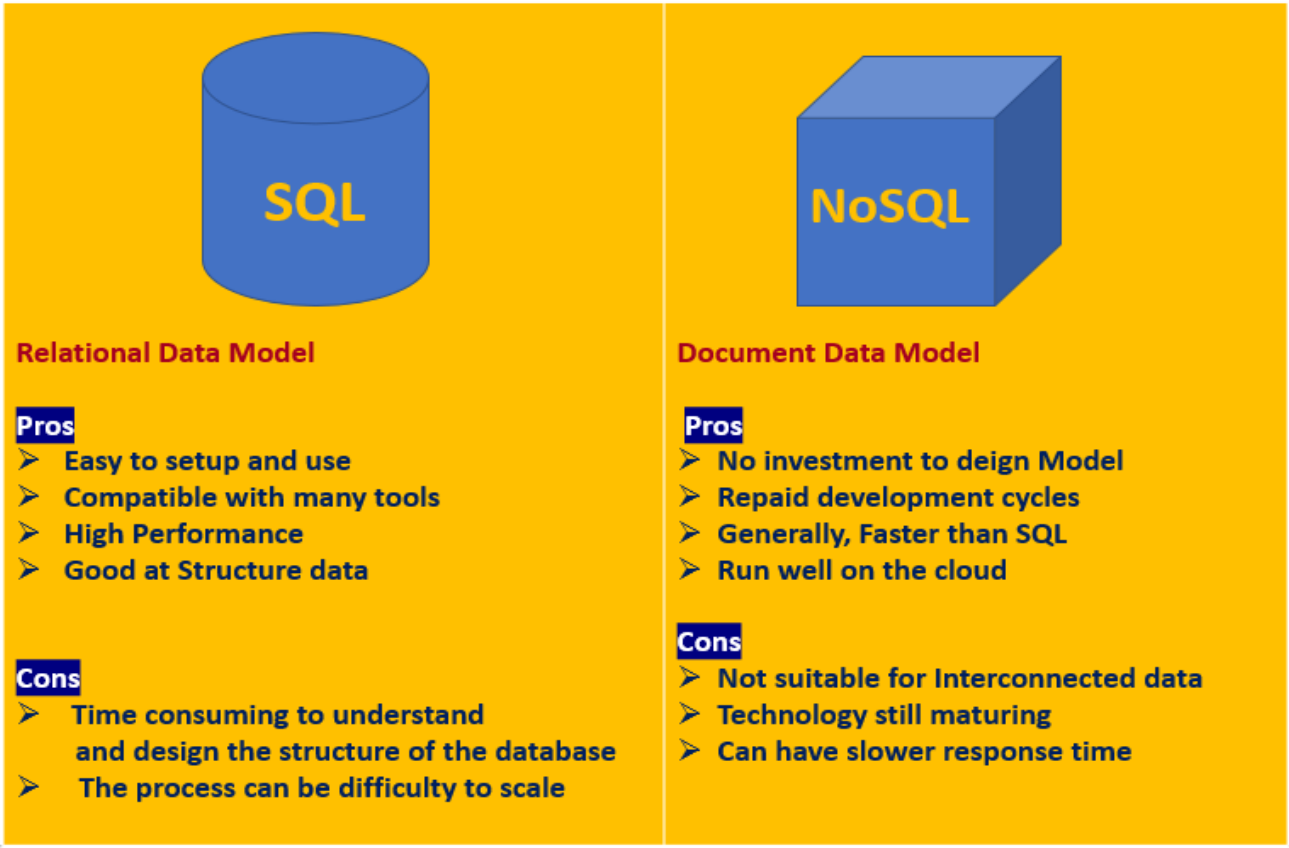
Module_4 - SQL, NoSQL Database Security & Physical Design

4.1.1 - Introduction

This module provides an overview of SQL and NoSQL data security and physical design, with a focus on the relational data model. Physical database design involves the conversion of logical data models into physical data models. NoSQL database systems primarily adopt the document-oriented paradigm, which introduces security concerns for Database Administrators (DBAs), although there are other types utilized as well. NoSQL databases enable efficient querying of media and documents while minimizing the risk of unauthorized access. However, this advantage comes with the need for increased processing power. In contrast, SQL data is more structured and imposes less overhead on storage and CPU resources. It's important to note that NoSQL databases can drive additional requirements.

Despite the convenience of easy data access offered by NoSQL, the security aspect is often overlooked. As a result, developers focus on providing robust queries not only for accessing files but also for securing the physical design and software implementation. This is accomplished through commonly used Application Program Interfaces (APIs) rather than relying solely on the SQL language.



As it was state earlier module that NoSQL Databases are open source and cheap applications when it compares to RDBMS (Relational Database Management Systems), which is fixed schema means the data can be inserted in a suitable format that will support to obtain the primary key and foreign key to align the needed data in the table. As far as security aspect of SQL is much faster and safer than NoSQL due to the complex queries in terms of data consistency, integrity, flexibility and increase efficiencies. The primary security risks that we might encounter by No SQL database are to

not to be able to protect the data on encrypted storage, and authorized exposure of data and its backups to insure the communication over networks are sage and uncheckable.

If the type of database technology to be used is known, an experienced database designer will concurrently perform physical database design alongside conceptual data modeling. Conceptual data modeling involves comprehending the organization and gathering the necessary requirements. On the contrary, physical database design focuses on constructing robust database structures that accurately represent the requirements using technical language. While there exist other data models, we prioritize highlighting the relational data model in this chapter for two reasons. Firstly, the relational data model is extensively employed in modern database applications. Secondly, certain principles of logical database design for the relational model are also applicable to other logical models.

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.

We 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

Comparison between Logical and Physical Design:

4.1.2 - 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 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.

4.2.1 - 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 considered. 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.

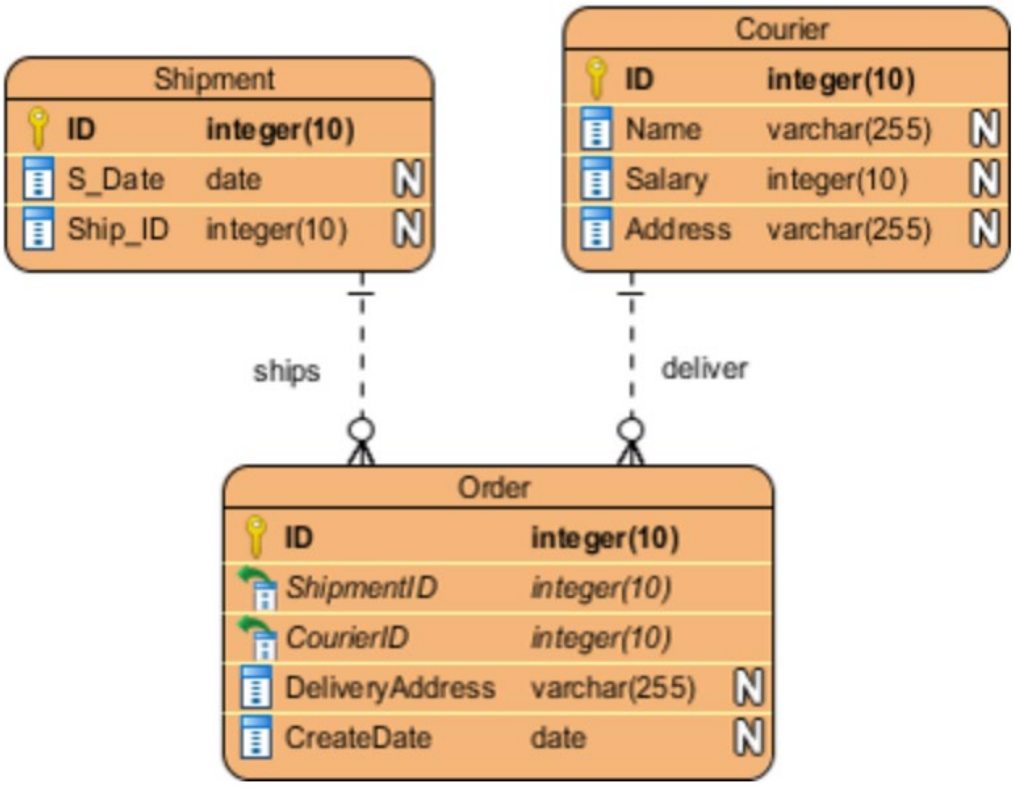


Figure 1. E-R Diagram, by Visual Paradigm, 2020, https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/. Copyright 2020 by Visual Paradigm

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 Table 1.

String	Numeric	Date/Time
CHAR	SMALLINT	DATE
VARCHAR	INTERGER	TIME
CLOB	DOUBLE	TIMESTAMP

Table 1. decomposed into varying lengths of data.

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.

4.2.2 - 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.

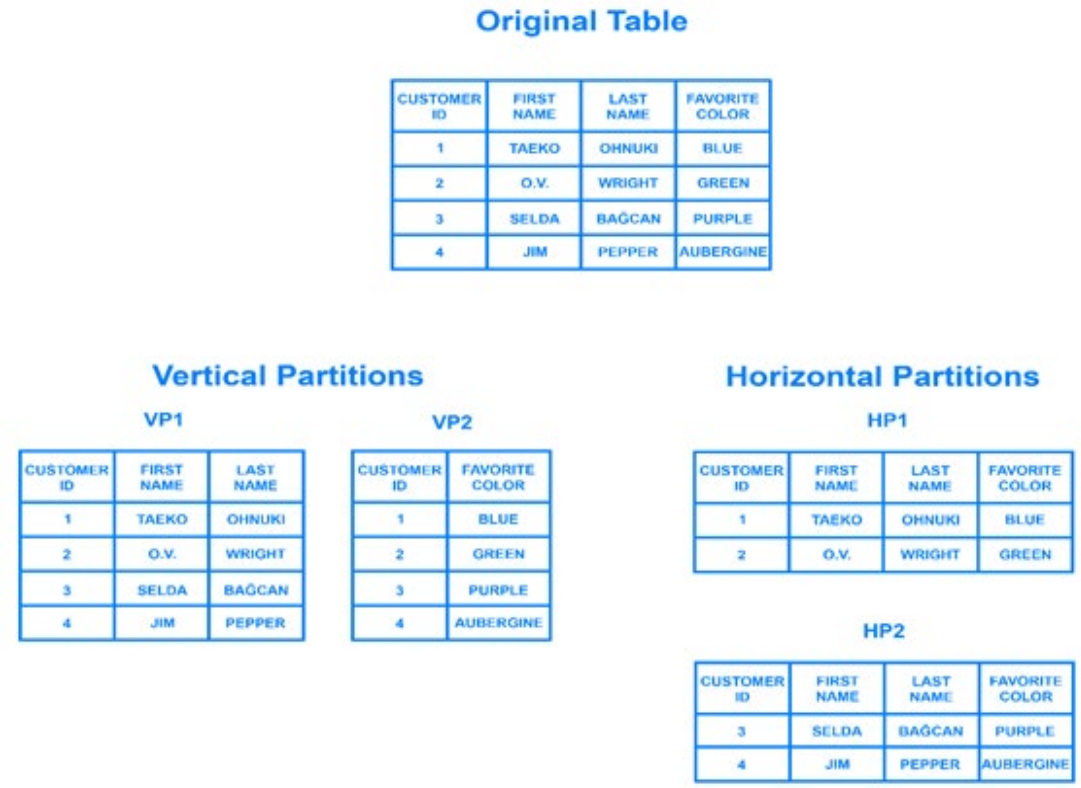


Figure 2. Sharding, by Digital Ocean, 2019, https://www.digitalocean.com/community/tutorials/understanding-database-sharding. Copyright 2020 by Digital Ocean

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.

4.3.1 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 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 several 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 several 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 inconsequentially, 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 Product Finish 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.

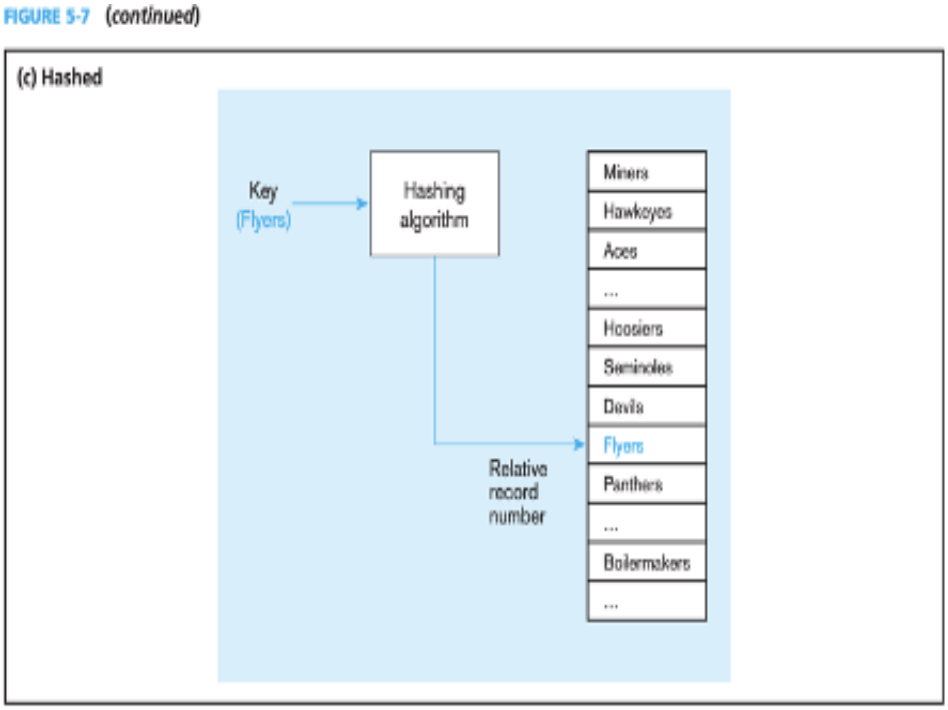


Fig 1.1 Modern Database Management (Venkataraman, R., Topi, H. (2011))

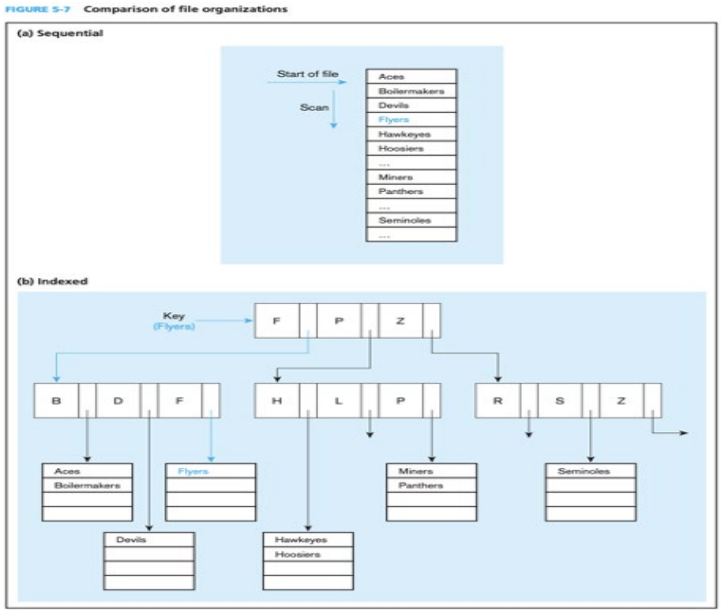


Fig 1.3 Modern Database Management 10th edition.((Venkataraman, R., Topi, H. 2011))

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 1.4.

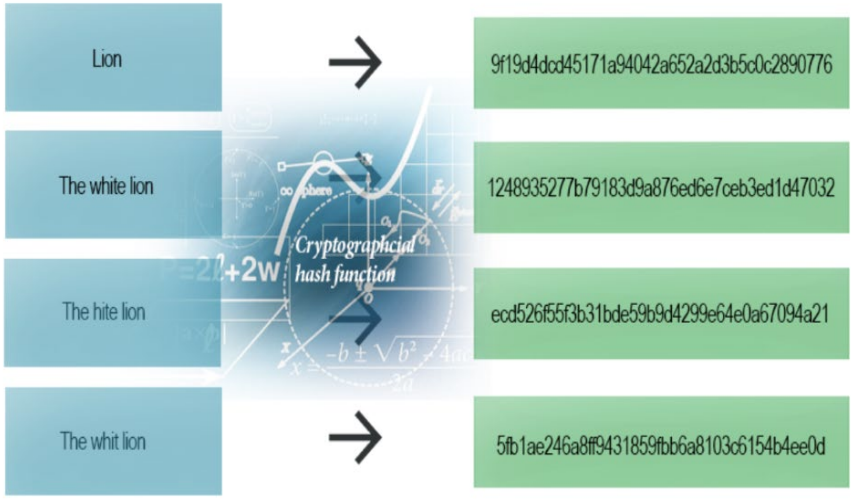


Figure 1.4. Hashing Algorithm, by jscrambler, 2020, https://blog.jscrambler.com/hashing-algorithms/. Copyright 2020 by jscrambler

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).

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 1.5.

```
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:

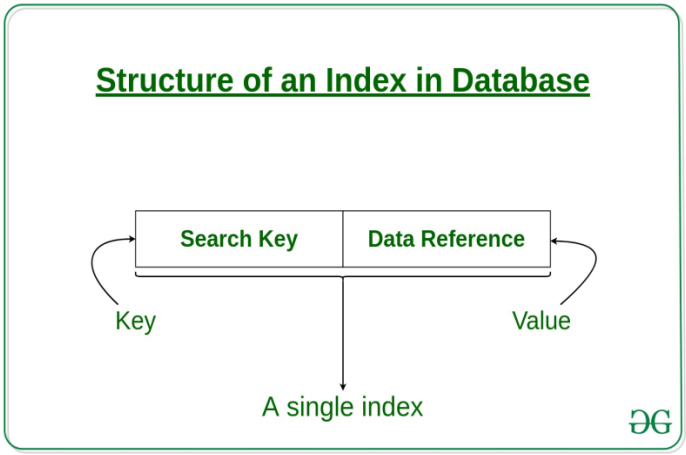
```
CREATE TABLE Customer_T(
   CustomerID                    VARCHAR2(25) NOT NULL,
   CustomerAddress               VARCHAR2(15)
   )
   CLUSTER Ordering (CustomerID);
CREATE TABLE Order_T (
   OrderID                  VARCHAR2(20) NOT NULL,
   CustomerID               VARCHAR2(25) NOT NULL,
   OrderDate                DATE
   )
   CLUSTER Ordering (CustomerID);
```

Figure 1.5. 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.

4.4.1 - Translate a database model into efficient structures.

Most 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.



Indexing in Databases, by GeeksforGeeks, 2020, https://www.geeksforgeeks.org/indexing-in-databases-set-1/
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.

4.4.1 Designing a Database for Optimal Query Performance

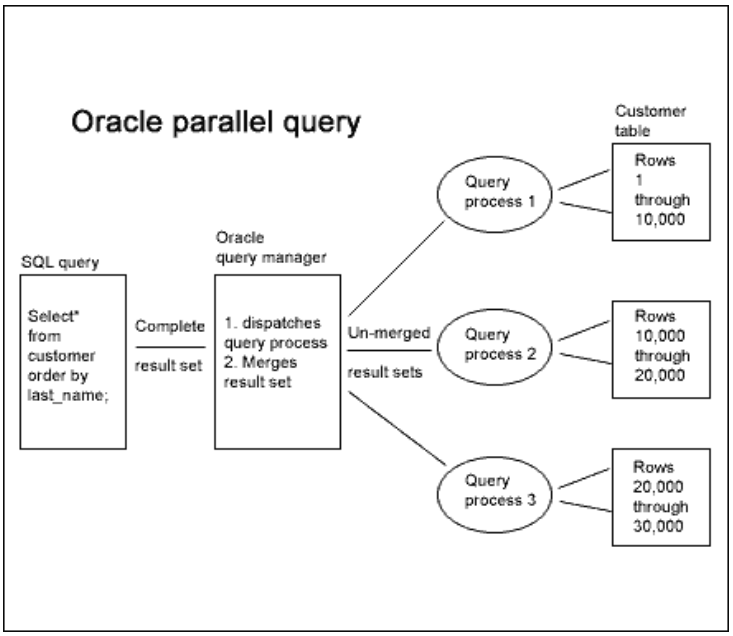
Databases are used every day, which makes it important to optimize the performance of database processing. Database processing can include adding, deleting, and modifying a database along with methods of retrieving data. Since databases have more traffic retrieving data from the database than maintenance, 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.



(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.

4.5 - 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 to realize optimization in technology.

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.

---

8. SQL Vs NoSQL Exact Differences And Know When To Use NoSQL And SQL

   This site will discuss about the comprehend the meaning of SQL and NoSQL mean, then we would be able to move ahead with their comparison easily.

   What is difference between the SQL and NoSQL and when is the best scenarios be able to use these software.

References

CloudGirl, & CloudGirl. (2017, March 26). Data Partitioning: Vertical Partitioning, Horizontal Partitioning, and Hybrid Partitioning Project Update #3. Retrieved from http://cloudgirl.tech/data-partitioning-vertical-horizontal-hybrid-partitioning/

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/resources/articles/introduction-to-hashing-and-its-uses.html

DigitalOcean. (2019, October 28). Understanding Database Sharding. Retrieved from https://www.digitalocean.com/community/tutorials/understanding-database-sharding

Database VLDB and Partitioning Guide. (2016, July 20). Retrieved from https://docs.oracle.com/database/121/VLDBG/GUID-BE424ACC-F746-4CA8-973C-F578CF98FF10.htm#VLDBG00225

Hoffer, A. J., Venkataraman, R., Topi, H. (2011). *Modern Database Management 10e*[E-Reader Version]. Retrieved from https://www.amazon.com/Modern-Database-Management-Jeffrey-Hoffer/dp/0136088392

Lesov, P. (2010). Database Security: A Historical Perspective. ArXiv, abs/1004.4022.

Physical Database Design. (n.d.). Retrieved from http://ewebarchitecture.com/web-databases/physical-database-design

Quick Base. (n.d.). A Timeline of Database History. Retrieved from https://www.quickbase.com/articles/timeline-of-database-history

Sicari, S., Rizzardi, A., Coen-Porisini, A., Seuciryt &* Proivacy issues and Challenges in NoSQL Databases., Elsevier Publisher, Computer Networks, Volume 206, April 7, 2022.

Watt, Adrienne. (n.d.). Chapter 13 Database Development Process. Retrieved from https://opentextbc.ca/dbdesign01/chapter/chapter-13-database-development-process/

What is Entity Relationship Diagram? (ERD). (2020). Retrieved from https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/