NATIONAL UNIVERSITY
COLLEGE OF COMPUTING AND
INFORMATION TECHNOLOGIES

VER 1.0

Course MANUAL

INFORMATION MANAGEMENT

This course provides students with the theoretical knowledge and practical skills in the utilization of databases and database management systems in the ICT applications. The logical design, physical design and implementation of relational databases are all covered in this course.

INFORMATION TECHNOLOGY

JULIE ANNE ANGELES-CRYSTAL
ROGER C. PRIMO JR.
MARK JAMES G. CAYABYAB
JOHN IVAN C. MAURAT

INFORMATION MANAGEMENT

Table of Contents

01
Topic 1
DATA VS. INFORMATION
INTRODUCTION TO DATABASE
PURPOSE OF DATABASE
DATABASE ARCHITECTURE

02
Topic 2
DATA MODELLING AND DATA MODELS
THE EVOLUTION OF DATA MODELS

03
Topic 3
RELATIONAL DATABASE MODEL
LOGICALVIEW DATA
DATA DICTIONARY

04
Topic 4
ENTITY RELATIONSHIP MODEL
DEVELOPING ER DIAGRAM

05
Topic 5
EXTEBDED ENTITY RELATIONSHIP MODE
ENTITY INTEGRITY

06
Topic 6
DATABASE TABLES NORMALIZATION
THE NEED FOR NORMALIZATION
THE NORMALIZATION PROCESS

07
Topic 7
INTRODUCTION RO SQL
DATA DEFINNITION COMMANDS
DATA MANIPULATION COMMANDS
SELECT QUERIES

MODULE

#03

NATIONAL UNIVERSITY

MODULE GOALS

- Logical View of Data
- Keys
- Integrity Rules
- Relational Set Operators
- Data Dictionary and the System Catalog
- Relationship within the Relational Database
- Data Redundancy Revisited
- Indexes
- Codd's Relational Database Rules.

FLEX Course Material

The Relational Database Model

WEEK 4-5 MODULE

Education that works.

LESSON #

3.1

Logical View of Data

A logical view of data refers to how data is perceived, organized, and structured from a conceptual or abstract standpoint, independent of the physical storage and implementation details.

It provides a high-level representation of the relationships between data elements, entities, and attributes within a system or database.

The logical view focuses on the semantics and meaning of the data rather than the technical aspects of how it is stored or accessed.

Logical View of Data

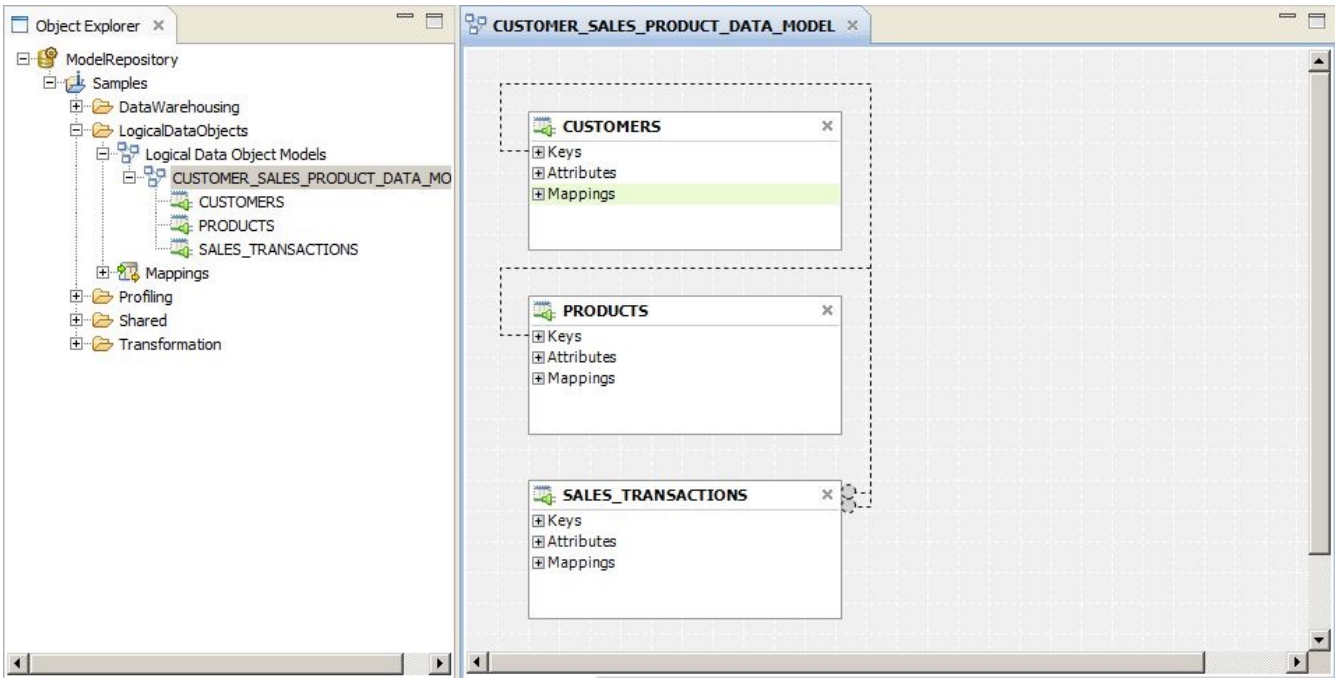
LOGICAL VIEW OF DATA OVERVIEW

- Relational database models structural and data independence enables us to view data logically rather than physically.
- The logical view allows a simpler file concept of data storage.
- The use of logically independent tables is easier to understand.
- Logical simplicity yields simpler and more effective database design methodologies.

- A logical view of data is a representation of data that resides in an enterprise. A logical view of data includes a logical data model, logical data objects, and logical data object mappings.
- With a logical view of data, you can achieve the following goals:
  - Use common data models across an enterprise so that you do not have to redefine data to meet different business needs. It also means if there is a change in data attributes, you can apply this change one time and use one mapping to make this change to all databases that use this data.
  - Find relevant sources of data and present the data in a single view. Data resides in various places in an enterprise, such as relational databases and flat files. You can access all data sources and present the data in one view.
  - Expose logical data as relational tables to promote reuse.

Logical View of Data

SAMPLE OF RELATED LOGICAL DATA OBJECTS



Logical View of Data

CHARACTERISTICS OF A LOGICAL VIEW OF DATA

1.Entity-Relationship Model:

- Entities represent distinct objects, concepts, or things of interest.
- Relationships define how entities are related to each other.

2.Attributes:

- Attributes describe the properties or characteristics of entities.
- Attributes help define the details and qualities associated with each entity.

3.Normalization:

- Logical views often adhere to normalization principles to minimize redundancy and maintain data integrity.
- Normalization involves organizing data in a way that reduces data duplication and dependency.

4.Data Integrity Constraints:

- Logical views include constraints that enforce data integrity rules, such as primary key constraints, foreign key constraints, and unique constraints.

5.Data Abstraction:

- Abstracts away technical details like storage mechanisms, access paths, and indexing.
- Focuses on the essential relationships and attributes that describe the data.

6.Entity-Relationship Diagrams (ERDs):

- ERDs are commonly used to visually represent the logical view of data.
- Entities, relationships, and attributes are depicted in a diagram, providing a clear and understandable representation.

CHARACTERISTICS OF A LOGICAL VIEW OF DATA

7. Logical Data Models:

- Logical data models are constructed to document and represent the logical view of data in a systematic way.
- These models are often created using data modeling techniques such as the Entity-Relationship Model or Unified Modeling Language (UML).

8.Business Rules:

- Business rules, which define the policies and regulations governing the data, are an integral part of the logical view.

- The logical view of data is crucial in the early stages of database design and development.
- It helps stakeholders, including database designers, analysts, and business users, understand the structure and relationships of the data without being concerned with the technical implementation details.
- This abstraction allows for a more comprehensive and meaningful representation of the information that aligns with the requirements and semantics of the business or application domain.

LESSON #

3.2

Keys

Keys play a crucial role in organizing and managing data.

They help establish relationships between tables, ensure data integrity, and facilitate efficient retrieval of information.

Understanding and appropriately implementing keys is fundamental to database design.

They ensure data integrity, facilitate efficient querying, and establish relationships between tables in a relational database system.

Keys

KEY TYPES:

1. Primary Key (PK):

- A primary key uniquely identifies each record in a table.
- It must contain unique values, and no two records in the table can have the same primary key.
- Primary keys are used as a reference in other tables (as foreign keys).
- Example: StudentID in a table of students.

2. Foreign Key (FK):

- A foreign key establishes a link between two tables.
- It refers to the primary key of another table, creating a relationship between the two tables.
- It ensures referential integrity by ensuring that values in the foreign key column match values in the corresponding primary key column.
- Example: CourseID in a table of enrollments, referring to the CourseID primary key in the Courses table.

3. Unique Key:

- Similar to a primary key but can allow one instance of a null value.
- It ensures that all values in the column are unique, except for null values.
- Useful when you want to ensure uniqueness but allow for exceptions where values might be unknown.
- Example: Social Security Number in an employee table (assuming null values are allowed).

KEY TYPES:

4. Super Key:

- A super key is a set of one or more attributes that, taken collectively, can uniquely identify a record.
- It may include more attributes than necessary to form a key.
- Example: (StudentID, Name, DateOfBirth) might be a super key for a table of students.

5. Candidate Key:

- A candidate key is a minimal super key, meaning it is a super key with no unnecessary attributes.
- It is a key that could be chosen as the primary key.
- Example: (Email) might be a candidate key for a table of users.

6. Alternate Key:

- An alternate key is a candidate key that is not selected as the primary key.
- It represents an alternative choice for the primary key.
- Example: If both (Email) and (Username) are candidate keys, and (Email) is chosen as the primary key, then (Username) is an alternate key.

7. Composite Key:

- A composite key is a key that consists of more than one attribute (column).
- Together, the combination of these attributes is unique across the table.
- Example: (DepartmentID, CourseID) might form a composite key in a table of courses, ensuring that no combination of department and course is repeated.

EXAMPLE - KEY TYPES:

There are different types of keys, each serving a specific purpose. Let's explore some common key types with examples:

- **Primary Key:**
  - The primary key uniquely identifies each record in a table.
  - Each table can have only one primary key.
  - Example: In a "Students" table, the student ID could be the primary key.

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Age INT
);
```

- **Foreign Key:**
  - A foreign key is a field in a table that refers to the primary key in another table. It establishes a link between two tables.
  - Example: In a "Courses" table, you might have a foreign key referencing the "StudentID" in the "Students" table.

```
CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(50),
    StudentID INT,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
);
```

EXAMPLE - KEY TYPES: (cont..)

- **Unique Key:**
  - A unique key ensures that all values in a column are unique.
  - Unlike the primary key, a table can have multiple unique keys.
  - Example: In a "Employees" table, the employee email could be a unique key.

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Email VARCHAR(100) UNIQUE
);
```

- **Composite Key:**
  - A composite key is a combination of two or more columns to uniquely identify a record.
  - Example: In a "Orders" table, a combination of "OrderID" and "ProductID" could form a composite key.

```
CREATE TABLE Orders (
    OrderID INT,
    ProductID INT,
    Quantity INT,
    PRIMARY KEY (OrderID, ProductID)
);
```

EXAMPLE - KEY TYPES: (cont..)

- **Candidate Key:**
  - A candidate key is a column or a set of columns that can qualify as a primary key.
  - Example: In a "Library" table, a combination of "ISBN" and "CopyNumber" could be a candidate key.

```
CREATE TABLE Library (
    ISBN VARCHAR(20),
    CopyNumber INT,
    Title VARCHAR(100),
    PRIMARY KEY (ISBN, CopyNumber)
);
```

- **Alternate Key:**
  - An alternate key is a candidate key that is not selected as the primary key.
  - Example: In a "Employees" table, both the employee ID and the employee SSN (Social Security Number) could be alternate keys.

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    SSN VARCHAR(11) UNIQUE,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
```

LESSON #

3.3

Integrity Rules

Integrity rules in the context of databases, especially relational database management systems (RDBMS), refer to a set of constraints that are applied to maintain the accuracy, consistency, and reliability of the data stored in the database.

These rules help ensure that the data remains valid and meaningful throughout its lifecycle.

INTEGRITY RULES

Types of Integrity Rules:

Entity Integrity:

- Ensures that each row (record) in a table is uniquely identified by a primary key, and that the primary key column does not contain null values.
- This is typically enforced by defining a primary key for each table.

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
```

Referential Integrity:

- Ensures the consistency of relationships between tables. It is based on the concept of foreign keys.
- A foreign key in one table must match a primary key in another table (or be null).
- This prevents orphaned records and maintains the integrity of relationships.

```
CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(50),
    StudentID INT,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
);
```

Types of Integrity Rules: (cont..)

Domain Integrity:

- Enforces valid data types and ranges for columns.
- It ensures that data entered into a column complies with the specified data type and constraints

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Age INT CHECK (Age >= 18) -- Example of domain integrity constraint
);
```

Check Integrity:

- Defines specific conditions that must be met for data to be entered or modified.
- These conditions are typically expressed using Boolean expressions.

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    OrderDate DATE,
    TotalAmount DECIMAL(10, 2),
    CHECK (TotalAmount >= 0) -- Example of check integrity constraint
);
```

Types of Integrity Rules: (cont..)

Default Integrity:

- Specifies a default value for a column. If a value is not provided during an insert operation, the default value is used.

```
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(50),
    QuantityInStock INT DEFAULT 0 -- Example of default integrity constraint
);
```

Unique Integrity:

- Ensures that all values in a column (or a combination of columns) are unique.
- This is often used to prevent duplicate entries.

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Email VARCHAR(100) UNIQUE -- Example of unique integrity constraint
);
```

Enforcing integrity rules is crucial for maintaining the quality of data within a database, preventing errors, and ensuring that the relationships between tables are accurate and reliable.

These rules are typically defined during the database schema design and are automatically enforced by the RDBMS.

INTEGRITY RULES

Integrity Rules Constraints

- Integrity Rules are imperative to a good database design. Most RDBMS have these rules automatically, but it is safer to just make sure that the rules are already applied in the design. There are two types of integrity mentioned in integrity rules, entity and reference. Two additional rules that aren't necessarily included in integrity rules but are pertinent to database designs are business rules and domain rules.
- Entity integrity exists when each primary key within a table has a value that is unique. This ensures that each row is uniquely identified by the primary key. One requirement for entity integrity is that a primary key cannot have a null value. The purpose of this integrity is to have each row to have a unique identity, and foreign key values can properly reference primary key values.
- Reference integrity exists when a foreign key contains a value and that value refers to an existing tuple/row in another relation. The purpose of reference integrity is to make it impossible to delete a row in one table whose primary key has mandatory matching foreign key values in another table.
- Business rules are constraints or definitions created by some aspect of a business. They can apply to almost all aspects of a business and are meant to describe the operations of a business. An example of a business rule might be no credit check is to be performed on return customers. This example would change a database design for a car company.

  Domain rules or integrity specify that all columns in a database must be declared upon a defined domain. A domain is a set values of the same value type.

  Other integrity rules include not null and unique constraints. The not null constraint can be placed on a column to ensure that every row in the table has a value for that column. The unique constraint is restriction placed on a column to ensure that no duplicate values exist for that column.

LESSON #

3.4

Relational Set Operators

Relational set operators are fundamental operations used in relational databases to manipulate and combine sets of data.

These operators enable you to perform various operations on tables, such as combining rows, finding common elements, and filtering data.

RELATIONAL SET OPERATORS

COMMON RELATIONAL SET OPERATORS:

Union (UNION):

- The UNION operator combines the result sets of two or more SELECT statements, eliminating duplicate rows.
- The tables or SELECT statements involved must have the same number of columns with similar data types.

```
SELECT column1, column2 FROM table1
UNION
SELECT column1, column2 FROM table2;
---------------------------------------
SELECT FirstName, LastName FROM Students
UNION
SELECT FirstName, LastName FROM Faculty;
```

- It combines the similar columns from two tables into one resultant table. All columns that are participating in the UNION operation should be Union Compatible.
- This operator combines the records from both the tables into one. If there are duplicate values as a result, then it eliminates the duplicate. The resulting records will be from both table and distinct.

RELATIONAL SET OPERATORS

COMMON RELATIONAL SET OPERATORS:

Union All (UNION ALL):

- The UNION ALL operator combines the result sets of two or more SELECT statements, including duplicate rows.
- Unlike the regular UNION operator, UNION ALL does not eliminate duplicate rows; it includes all rows from all SELECT statements.

```
SELECT column1, column2 FROM table1
UNION ALL
SELECT column1, column2 FROM table2;
---------------------------------------
SELECT FirstName, LastName FROM Students
UNION ALL
SELECT FirstName, LastName FROM Faculty;
```

- This operation is also similar to UNION, but it does not eliminate the duplicate records. It shows all the records from both the tables.
- All other features are same as UNION. We can have conditions in the SELECT query. It need not be a simple SELECT query.

Key Differences between UNION and UNION ALL:

- UNION removes duplicate rows, while UNION ALL includes all rows, even if they are duplicates.
- Because UNION ALL does not need to check for and eliminate duplicates, it is generally faster than UNION.
- Use UNION when you want to combine and deduplicate rows, and use UNION ALL when you want to combine all rows, including duplicates.
- It's important to choose between UNION and UNION ALL based on your specific requirements. If you want distinct rows, use UNION. If you want all rows, including duplicates, use UNION ALL.

RELATIONAL SET OPERATORS

COMMON RELATIONAL SET OPERATORS:

Intersection (INTERSECT):

- The INTERSECT operator returns the common rows between the result sets of two SELECT statements.
- Like UNION, the tables or SELECT statements must have the same number of columns with similar data types.

```
SELECT column1, column2 FROM table1
INTERSECT
SELECT column1, column2 FROM table2;

---------------------------------------
SELECT CourseID FROM EnrolledStudents
INTERSECT
SELECT CourseID FROM PassedCourses;
```

- This operator is used to pick the records from both the tables which are common to them.
- In other words it picks only the duplicate records from the tables. Even though it selects duplicate records from the table, each duplicate record will be displayed only once in the result set.
- It should have UNION Compatible columns to run the query with this operator.

RELATIONAL SET OPERATORS

COMMON RELATIONAL SET OPERATORS:

Difference (MINUS):

- The EXCEPT or MINUS operator returns the rows that exist in the first result set but not in the second result set.
- The tables or SELECT statements must have the same number of columns with similar data types.

```
SELECT column1, column2 FROM table1
MINUS
SELECT column1, column2 FROM table2;
----------------------------------------
SELECT FirstName, LastName FROM Students
MINUS
SELECT FirstName, LastName FROM GraduatedStudents;
```

- This operator is used to display the records that are present only in the first table or query, and doesn't present in second table / query. It basically subtracts the first query results from the second.

LESSON #

3.5

Data Dictionary & the System Catalog

Data Dictionary or System Catalog is a vital component of database management.

It provides a structured way to store and retrieve metadata, making it easier to understand, manage, and maintain the database over time.

DISTINCTION BETWEEN THE TERMS:

System Catalog:

- The term "System Catalog" is commonly associated with relational database management systems (RDBMS), such as Oracle, PostgreSQL, MySQL, and Microsoft SQL Server.
- It refers to a specific type of Data Dictionary that is managed by the database system itself.
- The System Catalog contains metadata about the database schema, tables, columns, indexes, constraints, users, and other structural aspects.

Data Dictionary:

- The term "Data Dictionary" is a broader term that can be used to refer to a repository of metadata about the data within a database.
- While the Data Dictionary may encompass the System Catalog in the context of RDBMS, it can also include additional information about data definitions, business rules, data lineage, and other documentation related to the database.

In some cases, the distinction between these terms may not be strictly observed, and they are used interchangeably to describe the repository of metadata about a database.

In summary, while there may be subtle differences in usage and scope, in the context of relational databases, the terms "System Catalog" and "Data Dictionary" are often used synonymously to refer to the repository of metadata that describes the structure and characteristics of the data in a database.

DATA DICTIONARY

A Data Dictionary serves as a central repository for metadata related to the database. It contains information that describes the data and the structure of the database. Key components include:

1. **Table Definitions:** Descriptions of each table, including their names, purposes, and relationships.
2. **Column Definitions:** Information about each column in a table, such as data type, length, and constraints.
3. **Relationships:** Details about how tables are related, including foreign key relationships.
4. **Indexes:** Information about indexes created on tables for performance optimization.
5. **Data Types:** Definitions of data types used in the database.
6. **Constraints:** Rules and restrictions applied to ensure data integrity.

SYSTEM CATALOG

The System Catalog is a term often associated with relational database management systems (RDBMS). It is a specific type of Data Dictionary that is maintained by the database management system itself. The System Catalog contains metadata that is essential for the system to manage and organize data effectively.

Key aspects include:

1. **Table Information:** Names, owners, and other details about tables.
2. **Column Information:** Data types, lengths, and constraints for each column.
3. **Index Information:** Details about indexes and their structures.
4. **User Information:** Details about users and their permissions.
5. **View Information:** Definitions and details about views.
6. **Procedure and Trigger Information:** Metadata related to stored procedures and triggers.

PURPOSE OF DATA DICTIONARY & SYSTEM CATALOG

- **Data Integrity:** Ensure the accuracy and consistency of data by enforcing constraints.
- **Data Organization:** Provide a structured view of the database schema, helping both developers and administrators understand the data model.
- **Data Security:** Store information about user permissions and access controls.
- **Database Administration:** Assist database administrators in managing and maintaining the database.

- A data dictionary is essentially a document or set of documents that outlines the data elements, their definitions, and the characteristics within a database or data source. It serves as a centralized resource for managing and documenting data, and helps ensure that data is consistent, accurate, and complete.
- In a business, a data dictionary can help teams collaborate, analyze, and standardize data effectively. It provides clear definitions and rules for data entry and management, which helps ensure that data is consistent and accurate across different systems and applications.
- The components of a data dictionary include data element names, definitions, data types, and any constraints or rules associated with the data.
- To create a data dictionary, one must identify the data elements and their characteristics, then document them in a standardized format.

DATA DICTIONARY

WHY YOU NEED SYSTEM CATALOG & DATA DICTIONARY

- Data catalogs and data dictionaries are complementary tools used to manage and organize data within a company. A data catalog acts as a central hub for all of a company's data, providing a single point of access for employees to find, understand and use the data they need. It helps to reduce the complexity of managing large amounts of data and enables companies to become more data-driven by providing a holistic view of all the data assets within an organization.
- Without a data dictionary, the data catalog would be incomplete as employees would struggle to understand the data and its relation to the other data and would find it difficult to interpret the information correctly. But with a unified data dictionary that captures knowledge about data stored in databases and data sources in a user-friendly, accessible, and collaborative environment, it becomes easier to understand and use the data, ultimately saving time and enabling better collaboration among employees.

- Helps ensure that data is accurate, consistent, and complete, which improves data quality
- By providing a standardized view of data across an organization, a data dictionary can increase efficiency and reduce errors
- Provides a common language and understanding of data, helping to improve collaboration between teams and departments
- A data dictionary helps ensure that data is compliant with regulatory requirements and internal policies, which can simplify compliance efforts
- You end up with accurate and consistent data, enabling better decision-making and more informed insights

LESSON #

3.6

Relationship within the Relational Database

BASIC TERMINOLOGY

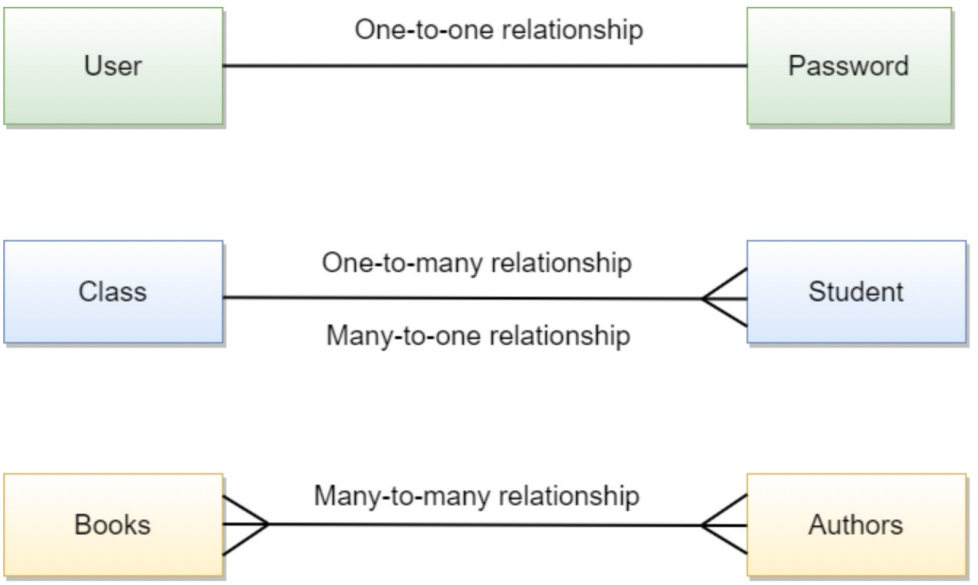
- The term ***relation*** is sometimes used to refer to a table in a relational database. However, it is more often used to describe the relationships that exist between the tables in a relational database.
- A ***relationship*** between two database tables presupposes that one of them has a foreign key that references the primary key of another table.
- ***Database entity***—strictly speaking — is a person, place, thing, object, or any item about which data is stored in the database. However, the term is usually used to refer to the database table as tables are, in fact, the physical implementation of entities.
- An ***entity-relationship diagram***, also known as ERD, ER diagram, or ER model, comprises a graphical representation of how entities relate to each other within a database.
  ER models are widely used in database design as they are fairly abstract and are easy to view and analyze.

RELATIONSHIP

TYPES OF RELATIONSHIP

There are 3 main types of relationship in a database:

- one-to-one
- one-to-many
- many-to-many.



ONE TO ONE RELATIONSHIP

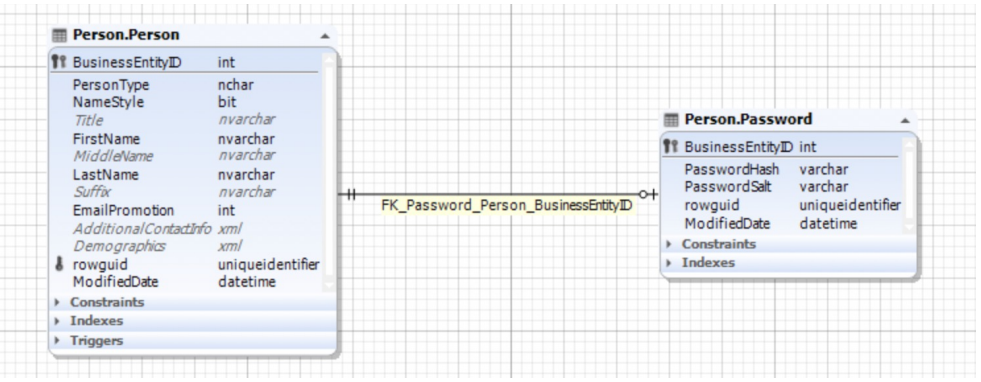
Definition: In a one-to-one relationship, each record in the first table is related to exactly one record in the second table, and vice versa.

Example: Consider a scenario where you have a "Person" table and a "Passport" table. Each person has exactly one passport, and each passport is associated with exactly one person.

```
CREATE TABLE Person (
    PersonID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);

CREATE TABLE Passport (
    PassportID INT PRIMARY KEY,
    PassportNumber VARCHAR(20),
    PersonID INT UNIQUE,
    FOREIGN KEY (PersonID) REFERENCES Person(PersonID)
);
```

In this example, the Passport table has a foreign key (PersonID) that references the primary key of the Person table.



ONE TO MANY RELATIONSHIP

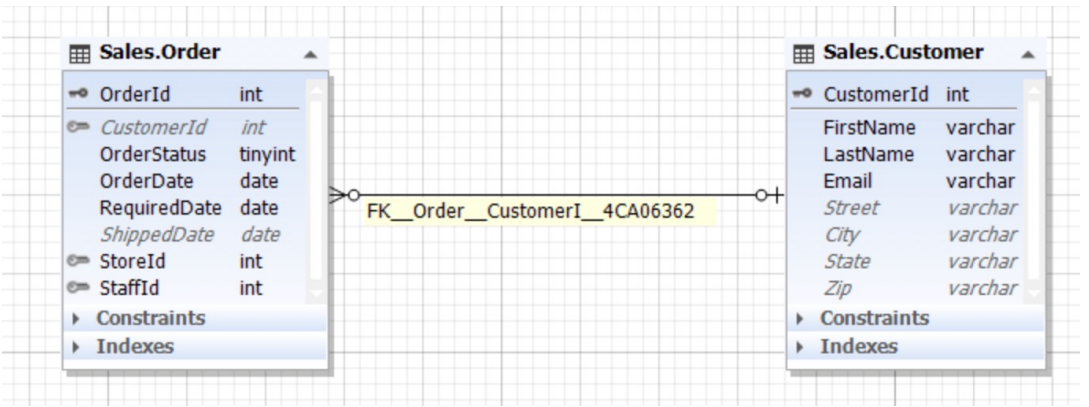
Definition: In a one-to-many relationship, each record in the first table can be related to multiple records in the second table, but each record in the second table is related to only one record in the first table.

Example: Consider a "Department" table and an "Employee" table. Each department can have multiple employees, but each employee belongs to only one department.

```
CREATE TABLE Department (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(50)
);

CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID)
);
```

In this example, the Employee table has a foreign key (DepartmentID) referencing the primary key of the Department table.



MANY TO MANY RELATIONSHIP

Definition: In a many-to-many relationship, each record in the first table can be related to multiple records in the second table, and vice versa.

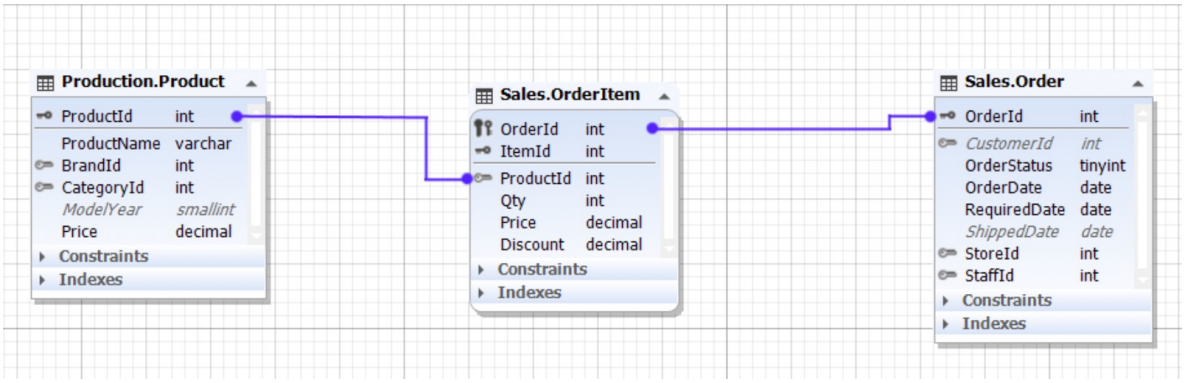
Example: Consider a "Student" table and a "Course" table. Each student can enroll in multiple courses, and each course can have multiple students.

```
CREATE TABLE Student (
   StudentID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50)
);

CREATE TABLE Course (
   CourseID INT PRIMARY KEY,
   CourseName VARCHAR(50)
);

CREATE TABLE Enrollment (
   EnrollmentID INT PRIMARY KEY,
   StudentID INT,
   CourseID INT,
   FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
   FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);
```

In this example, the Enrollment table serves as a junction table connecting the Student and Course tables in a many-to-many relationship.



MANY TO MANY RELATIONSHIP

Definition: In a many-to-many relationship, each record in the first table can be related to multiple records in the second table, and vice versa.

Example: Consider a "Student" table and a "Course" table. Each student can enroll in multiple courses, and each course can have multiple students.

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);

CREATE TABLE Course (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(50)
);

CREATE TABLE Enrollment (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);
```

In this example, the Enrollment table serves as a junction table connecting the Student and Course tables in a many-to-many relationship.

Understanding and appropriately modeling these relationships are crucial for designing a well-structured and normalized database. They help ensure data integrity, reduce redundancy, and enable effective querying of the database.

EXAMPLE:

Explore other examples of one-to-one, one-to-many, and many-to-many relationships in the context of a simple online bookstore scenario.

One-to-One Relationship:

- In this scenario, let's consider a one-to-one relationship between authors and author biographies. Each author has a unique biography.

In this example, the AuthorBiography table has a one-to-one relationship with the Authors table. Each author can have exactly one biography, and each biography is associated with exactly one author

```
-- Authors table
CREATE TABLE Authors (
    AuthorID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);

-- AuthorBiography table (one-to-one relationship with Authors)
CREATE TABLE AuthorBiography (
    BiographyID INT PRIMARY KEY,
    AuthorID INT UNIQUE,
    BirthDate DATE,
    Nationality VARCHAR(50),
    FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)
);
```

EXAMPLE:

One-to-Many Relationship:

- Consider a scenario where each book category can have multiple books, but each book belongs to only one category.

In this example, the Books table represents a one-to-many relationship. Each book is associated with one category, but a category can have multiple books.

```
-- BookCategories table
CREATE TABLE BookCategories (
    CategoryID INT PRIMARY KEY,
    CategoryName VARCHAR(50)
);

-- Books table (one-to-many relationship with BookCategories)
CREATE TABLE Books (
    BookID INT PRIMARY KEY,
    Title VARCHAR(100),
    Author VARCHAR(100),
    CategoryID INT,
    FOREIGN KEY (CategoryID) REFERENCES BookCategories(CategoryID)
);
```

EXAMPLE:

Many-to-Many Relationship:

- Let's consider a many-to-many relationship between customers and orders. Each customer can place multiple orders, and each order can have multiple customers (for instance, in case of joint orders).

In this example, the CustomerOrders table serves as a junction table for the many-to-many relationship between customers and orders. Each record in the CustomerOrders table links a customer to an order, and multiple customers can be associated with multiple orders.

```
-- Customers table
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);

-- Orders table
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    OrderDate DATE
);

-- CustomerOrders table (junction table for many-to-many relationship)
CREATE TABLE CustomerOrders (
    CustomerOrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);
```

LESSON #

3.7

Data Redundancy

Data redundancy refers to the duplication of data in a database system. It occurs when the same piece of information is stored in multiple places within a database.

While some level of redundancy is inevitable and even necessary for certain scenarios, excessive redundancy can lead to several issues, including increased storage requirements, data inconsistency, and difficulties in maintaining and updating the data.

DATA REDUNDANCY

WHAT IS DATA REDUNDANCY?

When there are multiple instances of the same data value, we call this "data redundancy." Data values should only appear within a database as many times as necessary. Ideally, each unique value should only appear once.

Data redundancy can lead to wasted resources and slower query times. As well as that, it can actually cause dangerous inconsistencies and even data loss. Database developers work hard to chase redundancy out of their systems with a process known as "normalization."

How does data redundancy work?

- Data needs to be stored in two or more places for it to be considered redundant. If the primary data becomes corrupted, or if the hard drive the data is on fails, then the extra set of data provides a fail-safe the organization can shift to.

- The redundant data can be either a whole copy of the original data or select pieces of data. Keeping select pieces of data enables an organization to reconstruct lost or damaged data. Hard drives with copies of data are stored in an array, so if something happens to the original data, the array can kick in with little to no downtime. In addition, redundancy measures can be accomplished through backups or RAID systems.

- **What is RAID? RAID (redundant array of independent disks) is a way of storing the same data in different places...*

DATA REDUNDANCY

HOW DOES DATA REDUNDANCY HAPPEN?

- Redundancy can occur within a database that lacks proper structure. For example, consider a database that contains two tables: one for customer information and one for order information.

- When a customer creates an account, the system creates a record in the customer information table. This record includes all of their information, including a home address. Later, the customer orders a product for home delivery. Because it is in a poorly designed database, the order information table also has an address column.

- Here, we see data redundancy: the same address exists twice. It's wasteful and inefficient, but it's also a risk. What happens if the customer logs in and changes their address? The system may only register the change on one table, so you now have two addresses for the customer, one of which is incorrect.

- This kind of problem can happen on a larger scale within an organization. For instance, most businesses will have a Customer Relationship Management system and a separate order fulfillment system. Out of necessity, both platforms will have their own databases. Each database may have information about an entity, such as a customer. Businesses have to keep these entities in sync.

DATA REDUNDANCY

CATEGORIES:

Data redundancy is when an organization stores the same data in multiple places at the same time.

It may occur within many fields in one database or across multiple technological platforms.

Positive data redundancy:

- This is intentional and occurs when an organization creates compressed versions of data to access as a backup.
- Intentional data redundancy promotes uniformity and protects data, and it safeguards data in different places to ensure the company's data remain sustainable.

Wasteful data redundancy:

- This is an unintentional replication of data in a company, which can result from complicated data processes and inefficient coding.
- It can be difficult to assess which data to update or use when unintentional storage of the same data occurs, but an organization can follow certain practices to reduce this problem.

DATA REDUNDANCY

BENEFITS & DRAWBACKS OF DATA REDUNDANCY

- Data redundancy can occur in databases and file-based storage systems.
- A database is a systematic collection of data stored electronically on a computer in which a database management system (DBMS) controls and can manipulate the stored data.
- A file-based storage system is an ordered and nested folder-based method to store and organize data on devices that include a hard drive, flash drive, DVD or cloud-based storage system.
- Depending on the application, data redundancy may have various benefits and drawbacks, including:

BENEFITS OF DATA REDUNDANCY

A company benefits from data redundancy that's intentional and built into a daily data management plan. Purposeful, positive data redundancy also:

- **Creates data backups:** Data redundancy helps protect and reinforce data backups when data disruption occurs through unintended data loss. It rebuilds or replaces missing data and ensures continuity.
- **Improves data protection:** Data redundancy minimizes the effect of a data breach because you can access the data from multiple sources.
- **Provides data access speed:** In a company that has many locations, individuals may access data from redundant sources to enjoy faster access to the same data. Easy access to data is vital for customer-oriented businesses that seek to provide efficient services.
- **Ensures data accuracy:** Hosting multiple data servers for the same data enables a DBMS to examine and evaluate variances and ensure data is consistent and accurate.
- **Expedites data recovery:** Through the support of data backups and data that's easy to access, data redundancy expedites data recovery and minimizes downtime of access to vital data.
- **Utilizes data storage flexibility:** A company can use flexible data storage options to enable data redundancy to support data sharing, which is vital in complex and customer-oriented organizations.

DRAWBACKS OF DATA REDUNDANCY

Unintentional data redundancy can put an organization at risk of ineffective decision-making because it may analyze outdated, biased or irrelevant data.

Here are some drawbacks that wasteful data redundancy may cause:

Increases data inconsistencies

- Data inconsistencies occur when an organization unintentionally stores data in different formats across multiple tables. Incorrect values and missing information can cause discrepancies in the data when intentional live updates are absent and may provide meaningless or unreliable information. A company can eliminate data inconsistencies when it checks audit trails in detail, performs regular archives and implements risk-based validation of the operating system.

Allows for data corruption

- Unplanned storing of multiple copies of the same data increases the chance of data corruption from errors in writing, processing, transferring and reading data across many locations. A company can overcome data corruption by resolving system and application issues in the initial programming stage and by troubleshooting the application and system issues regularly. Daily backups of essential information can be a good strategy to prepare for potential data corruption.

DRAWBACKS OF DATA REDUNDANCY

Increases data maintenance costs

- Multiple unintentional copies of the same data can increase a company's data maintenance costs. Expensive storage fees of a complex and wasteful data system can burden a company that wants to reduce its overhead costs. Unintentional redundant copies of a large amount of data waste storage space and may create confusion if data administrators can't find portions of stored data.

- When a company consolidates data storage with a storage-area network (SAN) and a single data storage pool, it's easier to back up and manage than multiple separate servers and requires less excess capacity. Using open-source software is also a cost-effective way to reduce database maintenance costs and help overcome this drawback.

TIPS FOR REDUCING WASTEFUL DATA REDUNDANCY

It might be difficult to eliminate data redundancy completely, but you can make your data redundancy more efficient.

Here are some tips to help you reduce wasteful data redundancy:

- **Design effective databases**
- A company can design databases engineered to facilitate the identification of data redundancy. To facilitate effective data storage methods, computer programmers often format identical fields in multiple tables and don't store the data in different formats. If a business uses external data, it tries to ensure its data collection follows an accurate, reliable and consistent methodology.
- **Integrate data**
- Data integration is the practice of combining data from internal and external sources. Many organizations mine data for different purposes, and strategic data integration creates accessible, meaningful and valuable information. Valuable data can improve a company's workflow and create a better customer experience.
- **Delete unused data**
- Valuable data is any data that's easily accessible and of high quality. Database managers can identify data they no longer need and delete data they're not using to optimize their databases. When migrating to a new operating system, it's also often vital to delete data that wastes storage space, causes unnecessary storage costs and slows essential functions.

TIPS FOR REDUCING WASTEFUL DATA REDUNDANCY

- **Use data normalization**
- Data normalization refers to data that doesn't have updates, inserts or deletion inconsistencies. When a company normalizes data, it can standardize how it organizes data across storage platforms, which makes it easier to recognize wasteful data redundancies.
- For example, a company may normalize a country with an abbreviation, while another may require complete words.

LESSON #

3.8

Codd's Relational Database Rules

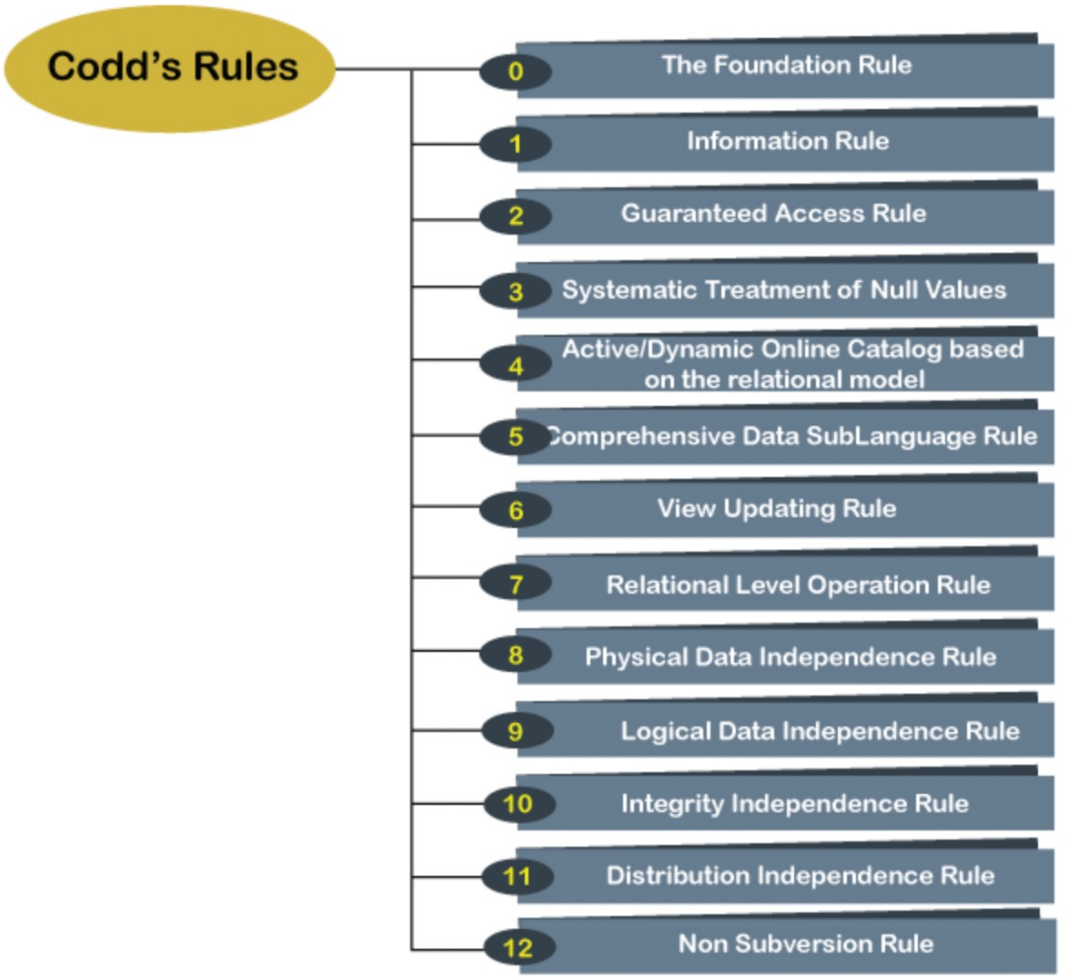
In 1985, Dr. E. F. Codd published a list of 12 rules to define a relational database system.

The reason Dr. Codd published the list was his concern that many vendors were marketing products as "relational" even though those products did not meet minimum relational standards.

CODD'S RULES

- Every database has tables, and constraints cannot be referred to as a rational database system. And if any database has only relational data model, it cannot be a **Relational Database System (RDBMS)**. So, some rules define a database to be the correct RDBMS.
- These rules were developed by **Dr. Edgar F. Codd (E.F. Codd)** in **1985**, who has vast research knowledge on the Relational Model of database Systems. Codd presents his 13 rules for a database to test the concept of DBMS against his relational model, and if a database follows the rule, it is called a **true relational database (RDBMS)**.

CODD'S 12 RULES:



CODD'S 12 RULES

- **Rule 1: Information Rule**
- The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.
- **Rule 2: Guaranteed Access Rule**
- Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.
- **Rule 3: Systematic Treatment of NULL Values**
- The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following – data is missing, data is not known, or data is not applicable.
- **Rule 4: Active Online Catalog**
- The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

CODD'S 12 RULES

- **Rule 5: Comprehensive Data Sub-Language Rule**
- A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.
- **Rule 6: View Updating Rule**
- All the views of a database, which can theoretically be updated, must also be updatable by the system.
- **Rule 7: High-Level Insert, Update, and Delete Rule**
- A database must support high-level insertion, updating, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.
- **Rule 8: Physical Data Independence**
- The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

CODD'S 12 RULES

- **Rule 9: Logical Data Independence**
- The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.
- **Rule 10: Integrity Independence**
- A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.
- **Rule 11: Distribution Independence**
- The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.
- **Rule 12: Non-Subversion Rule**
- If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.