Chapter 3: Relational Model Characteristics

# A logical View of Data

## Introduction to Databases

**Database:** A collection of organized data and metadata (data about data) that is managed by a Database Management System (DBMS).

**DBMS:** Software that manages and controls access to the database, sitting between the application and the database itself.

## The Relational Model

**Relational Database Model:** Focuses on the logical representation of data rather than physical storage.

**Table (Relation):** The core component of a relational database. It is a two-dimensional structure composed of rows and columns.

## Logical vs Physical

**Logical View:** Think of data in terms of tables (like spreadsheets) with rows and columns, which simplifies database design.

**Physical Storage:** How data is actually stored on disk, which is managed by the DBMS, not by the designer.

## Key Concepts

**Row (Tuple):** Represents a single record or entity.

**Column (Attribute):** Represents a characteristic or property of the entity.

**Entity Set:** A collection of similar entities (e.g., all students in a university).

**Attribute:** A named column in a table.

**Domain:** The set of permissible values for an attribute. For example a GPA score is between 0 and 4 so therefore the domain is (0,4)

## Properties of a Relation (Table)

1. **Two-dimensional Structure:** Consists of rows and columns. If there are 10 rows and 12 attributes then it is said that the table has a cardinality of 10 and degrees of 12.

2. **Unique Rows:** Each row represents a distinct entity.

3. **No Duplicate Rows**: Each row must be unique.

4. **Named Columns:** Each column must have a distinct name.

5. **Atomic Values:** Each cell (intersection of row and column) must contain a single value.

6. **Uniform Data Format:** All values in a column must conform to the same data type (e.g.,

integers, dates).

7. **Attribute Domain:** Each column has a specific range of valid values.

8. **Order Irrelevance:** The order of rows and columns does not matter to the DBMS.

9. **Primary Key:** Each table must have an attribute or a combination of attributes that uniquely identifies each row.

## Examples and Terms

**Relation:** In a relational database, a table is referred to as a relation.

**Tuple:** Another term for a row in a table.

**Attributes:** Columns in a table.

**Primary Key (PK):** A unique identifier for each row in a table, ensuring that no two rows are identical.

**Degree:** The number of columns in a table.

**Cardinality:** The number of rows in a table.

**Example:**

Consider a simple table named `STUDENT`:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STU\_NUM** | **STU\_LNAME** | **STU\_FNAME** | **STU\_DOB** | **STU\_GPA** |
| 321452 | Ndlovu | Amehlo | 12Feb1999 | 2.84 |
| 324257 | Smithson | Anne | 15Nov2000 | 3.27 |
| 324258 | Le Roux | Dan | 23Aug2000 | 2.26 |

Columns (Attributes): STU\_NUM, STU\_LNAME, STU\_FNAME, STU\_DOB, STU\_GPA

Rows (Tuples): Each row represents a student.

Primary Key: STU\_NUM uniquely identifies each student.

## Summary

* The relational model simplifies database design by focusing on logical data representation through tables.
* Tables consist of rows and columns, where each row is a unique record, and each column is a characteristic of that record.
* The relational model's simplicity makes it easier to understand and design databases compared to older hierarchical and network models.

# Keys

## Keys and Their Role in Databases

A key in a relational database is a set of one or more attributes that uniquely identifies a row (record) within a table. Keys are essential for ensuring data integrity and for establishing relationships between tables. Here, we delve into various types of keys and their roles, using examples from a hypothetical `STUDENT` table to illustrate key concepts.

## Determination

The concept of a key is rooted in the principle of **determination**, where knowing the value of one attribute (or a combination of attributes) allows you to determine the value of another attribute. For example, in a `STUDENT` table:

* If you know the `STU\_NUM` (student number), you can determine the `STU\_LNAME` (student's last name), `STU\_FNAME` (student's first name), `STU\_PHONE` (student's phone number), and other attributes.
* This can be represented as:
* STU\_NUM → STU\_LNAME (STU\_NUM determines STU\_LNAME)
* STU\_NUM → STU\_FNAME, STU\_INIT, STU\_DOB, STU\_TRANSFER

However, the reverse is not true; knowing `STU\_LNAME` does not necessarily determine `STU\_NUM` because multiple students can have the same last name.

## Functional Dependence

Functional dependence describes a relationship where one attribute's value is determined by another's. Attribute B is functionally dependent on attribute A if knowing A allows you to know B. For instance:

* `STU\_PHONE` is functionally dependent on `STU\_NUM` because each student number uniquely determines a phone number.
* However, `STU\_NUM` is not functionally dependent on `STU\_PHONE` if multiple students share the same phone number.

**Functional dependence can be formally defined as:**

* Attribute A determines attribute B (B is functionally dependent on A) if all rows in the table with the same value of A also have the same value of B.

## Types of Keys

1. **Primary Key:**

* A candidate key chosen to uniquely identify rows in a table.
* Must contain unique values and cannot contain null values.
* Example: `STU\_NUM` in the `STUDENT` table.

1. **Superkey:**

* Any set of attributes that uniquely identifies rows.
* Can be a single attribute or a combination.
* Example: `STU\_NUM`, or `STU\_NUM, STU\_LNAME` are both superkeys, but only `STU\_NUM` is a candidate key due to its minimal nature.

1. **Candidate Key:**

* A minimal superkey, meaning it has no redundant attributes.
* Example: `STU\_NUM` alone is a candidate key because it uniquely identifies each student without needing additional attributes.

1. **Composite Key:**

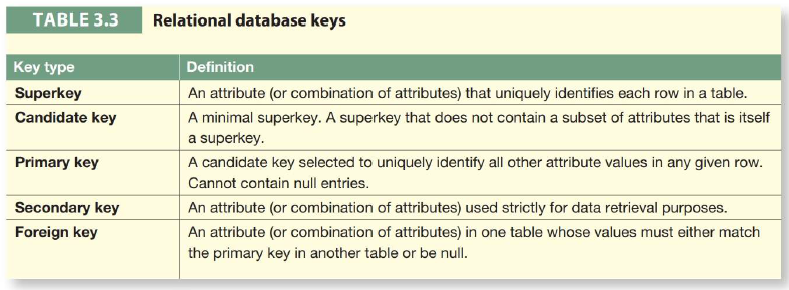
* A key composed of more than one attribute.
* Example: `STU\_LNAME, STU\_FNAME, STU\_INIT, STU\_PHONE` could uniquely identify a student when combined.

1. **Secondary Key:**

* Used for data retrieval purposes and does not necessarily uniquely identify rows.
* Example: `STU\_LNAME` and `STU\_PHONE` could be used to search for students but might not be unique.

1. **Foreign Key:**

* An attribute in one table that matches the primary key in another table, establishing a relationship between the two tables.
* Example: `VEND\_CODE` in a `PRODUCT` table referencing `VEND\_CODE` in a `VENDOR` table.



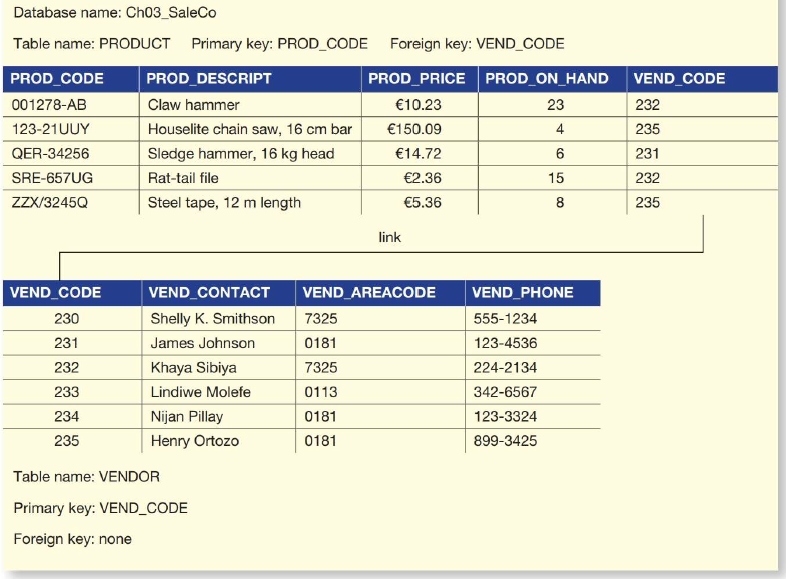
## Entity Integrity and Referential Integrity

**Entity Integrity:**

* Ensures each row in a table is uniquely identifiable via the primary key.
* Primary keys must be unique and non-null.

**Referential Integrity:**

* Ensures that foreign keys in a table correctly refer to primary keys in another table.
* Example: In the `PRODUCT` and `VENDOR` tables, `VEND\_CODE` in `PRODUCT` must match an existing `VEND\_CODE` in `VENDOR`.



## Practical Examples

Consider the following simplified tables:

**STUDENT Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **STU\_NUM** | **STU\_LNAME** | **STU\_FNAME** | **STU\_PHONE** |
| 1 | Smith | John | 555-1234 |
| 2 | Doe | Jane | 555-5678 |

**VENDOR Table:**

|  |  |
| --- | --- |
| **VEND\_CODE** | **VEND\_NAME** |
| 101 | Vendor A |
| 102 | Vendor B |

**PRODUCT Table:**

|  |  |  |
| --- | --- | --- |
| **PROD\_CODE** | **PROD\_NAME** | **VEND\_CODE** |
| 201 | Product X | 101 |
| 202 | Product Y | 102 |

Understanding keys in a relational database is crucial for designing robust and efficient database systems. Keys ensure data integrity, enable precise data retrieval, and define relationships between tables. By mastering the different types of keys and their roles, one can create databases that are both logically sound and practically useful.

# Integrity Rules

Relational database integrity rules are fundamental for ensuring data accuracy and consistency within a relational database. Proper enforcement of these rules is crucial for maintaining reliable and meaningful data. Many Relational Database Management Systems (RDBMSs) can enforce these rules automatically, but it is essential to design applications with these rules in mind to ensure robust database operations. The two primary integrity rules are Entity Integrity and Referential Integrity. Additionally, constraints like NOT NULL and UNIQUE further help maintain data integrity.

## Entity Integrity

**Requirement:**

* All primary key entries must be unique, and no part of a primary key can be null.

**Purpose:**

* This rule ensures that each row in a table can be uniquely identified by its primary key, allowing for precise retrieval and management of data. It also ensures that foreign key values can properly reference primary key values without ambiguity.

**Example:**

* In an invoice system, each invoice must have a unique number. No two invoices can share the same number, and every invoice must have an assigned number (i.e., it cannot be null). This ensures that each invoice is uniquely identified by its invoice number.

## Referential Integrity

**Requirement:**

* A foreign key must have either a null value (if it is not part of its table's primary key) or a value that matches an existing primary key in a related table. Essentially, every non-null foreign key value must reference a valid primary key value in the related table.

**Purpose**:

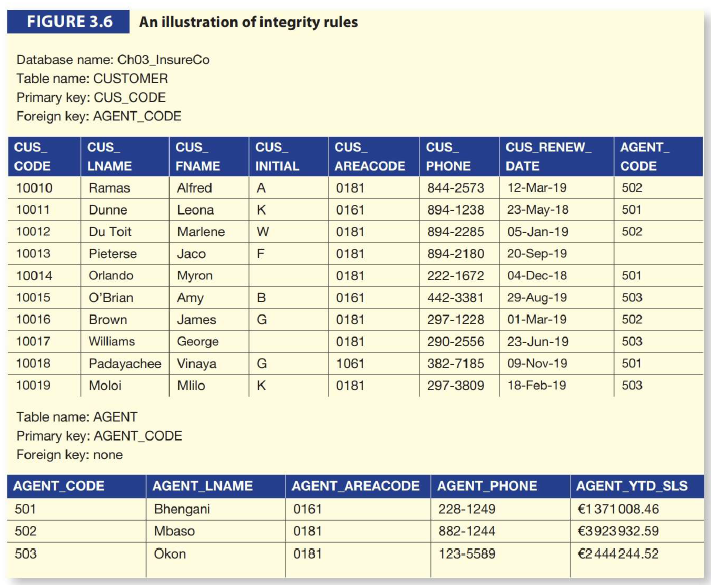
* Referential integrity ensures that relationships between tables remain consistent. For instance, it prevents the deletion of a row in one table if that row's primary key is referenced as a foreign key in another table. This rule ensures that it is impossible to have an invalid reference.

**Example:**

* In a customer database, a customer might not yet have an assigned sales representative. In this case, the foreign key (sales representative number) in the CUSTOMER table can be null. However, it is impossible to have a reference to a non-existent sales representative number, ensuring all foreign key references are valid.

## Illustration of Integrity Rules

Figure 3.6 exemplifies these integrity rules using two tables: CUSTOMER and AGENT.



1. **Entity Integrity:**

* CUSTOMER Table: The primary key is `CUS\_CODE`. Each `CUS\_CODE` value is unique, and there are no null entries, ensuring every customer is uniquely identifiable.
* AGENT Table: The primary key is `AGENT\_CODE`, also ensuring unique and non-null values.

1. **Referential Integrity:**

* The `CUSTOMER` table includes a foreign key `AGENT\_CODE`, linking to the `AGENT` table. For instance, the row for customer `10013` has a null `AGENT\_CODE`, indicating no assigned agent yet. Other entries match valid `AGENT\_CODE` values in the `AGENT` table.

To avoid nulls, some designers use special codes known as **flags**. For example, a code like `-99` could be used to indicate that a customer does not yet have an assigned agent. In such cases, the `AGENT` table would need a dummy entry with `AGENT\_CODE` set to `-99`.

## Additional Constraints

**NOT NULL Constraint:**

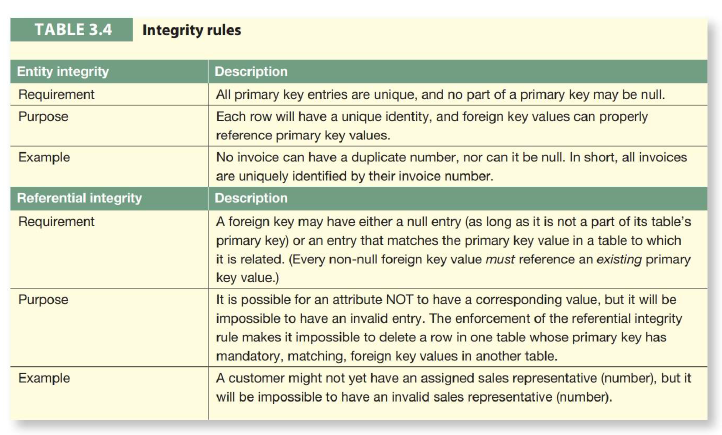
Ensures every row in the table has a value for a specific column. This constraint prevents null values in important columns (Such as the primary column)

**UNIQUE Constraint:**

Ensures no duplicate values exist for a particular column. This constraint helps maintain data uniqueness beyond primary keys.

## Summary

**Table below summarizes the key integrity rules:**



## Handling Nulls and Referential Integrity

Handling nulls effectively is crucial in database design. For instance, nulls may indicate unknown, missing, or inapplicable values. Improper use of nulls can create issues, especially in functions like COUNT, AVERAGE, and SUM. To mitigate this, some systems use flags or dummy values.

In summary, maintaining entity and referential integrity, along with enforcing constraints like NOT NULL and UNIQUE, is vital for a reliable and efficient relational database system. These practices ensure data consistency, accuracy, and meaningful relationships between tables.

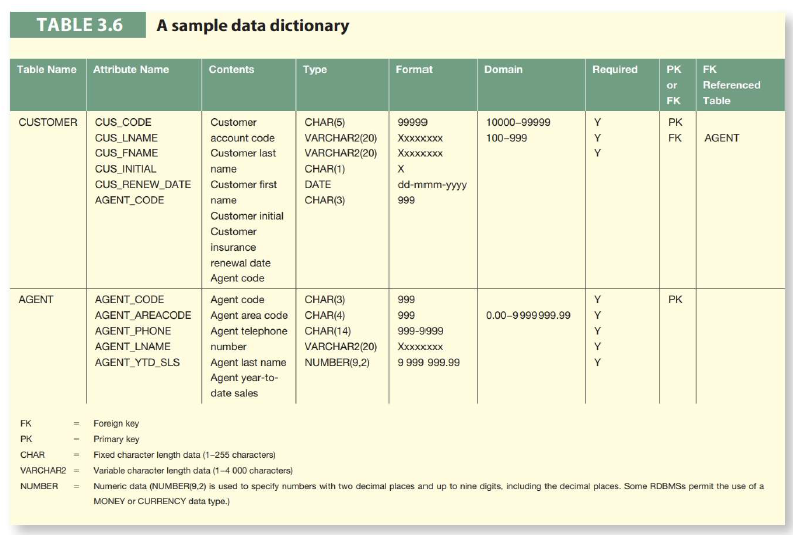
# Data Dictionary and the System Catalogue

The data dictionary provides a detailed accounting of all tables found within the user/designer-created database. Thus, the data dictionary contains atleast all of the attribute names and characteristics for each table in the system. In short, the data dictionary contains metadata (data about data). The data dictionary is a repository that provides detailed accounting of all tables and attributes within a database. It acts as a blueprint for database designers and developers, capturing essential information such as:

* **Attribute Names:** The names of the fields or columns in each table.
* **Attribute Characteristics:** Data types, constraints (e.g., NOT NULL, UNIQUE), default values, etc.

The data dictionary is sometimes referred to as the "database designer's database" because it records design decisions about tables and their structures. It is a vital tool for ensuring consistency and clarity in database design and development.

Using the small database presented in Figure 3.6, you might picture its data dictionary as shown in Table 3.6.



**NOTE 1:**

Telephone area codes are always composed of digits 0-9. Because area codes are not used arithmetically, they are most efficiently stored as character data. Also, the area codes are always composed of a maximum of four digits. Therefore, the area code data type is defined as CHAR(4). On the other hand, names do not conform to a standard length. Therefore, the customer first names are defined as VARCHAR2(20), thus indicating that up to 20 characters may be used to store the names. Character data are shown as left justified.

**NOTE 2:**

The data dictionary in Table 3.6is an example of the human view of the entities, attributes, and relationships. The purpose of this data dictionary is to ensure that all members of database design and implementation teams use the same table and attribute names and characteristics. The DBMSs internally stored data dictionary contains additional information about relationship types, entity and referential integrity checks and enforcement, and index types and components. This additional information is generated during the database implementation stage

## System Catalogue

The **system catalogue**, often synonymous with the data dictionary, is a more comprehensive metadata repository managed by the DBMS. It contains detailed descriptions of all database objects, including:

* Table Names
* Table Creators and Creation Dates
* Column Information: Names, data types, sizes
* Index Information: Filenames, creators
* User and Access Information: Authorized users, access privileges

Since the system catalogue includes all necessary data dictionary information, it is often used interchangeably with the term data dictionary. Modern RDBMSs typically provide a system catalogue, which serves as the authoritative source of metadata. The system catalogue is automatically maintained by the DBMS, ensuring that all database documentation is up-to-date and consistent.

## Homonyms and Synonyms in Databases

The system catalogue helps prevent issues related to homonyms and synonyms:

* **Homonyms:** These are identical attribute names used for different attributes in various tables, which can lead to confusion. For example, using `C\_NAME` for both a customer name in a `CUSTOMER` table and a consultant name in a `CONSULTANT` table. The data dictionary can help avoid this by providing clear and distinct names for each attribute.
* **Synonyms:** These are different names used for the same attribute, which can also cause confusion. For example, using `car` in one table and `auto` in another to refer to the same entity. Synonyms should be avoided to maintain clarity and consistency in the database schema.

## Importance of Data Dictionary and System Catalogue

Importance of Data Dictionary and System Catalogue:

1. **Consistency:** Ensures that all database components adhere to the design specifications.
2. **Documentation:** Provides up-to-date documentation of the database structure, useful for developers and administrators.
3. **Data** **Integrity:** Assists in maintaining data integrity by enforcing rules and constraints.
4. **Query** **Support**: Facilitates querying the database schema to understand its structure and relationships.
5. **Eliminating** **Confusion:** Helps avoid issues related to homonyms and synonyms by providing clear and consistent naming conventions.

In conclusion, the data dictionary and system catalogue are fundamental to effective database design and management, ensuring consistency, integrity, and clarity in the database environment.

# Relationships Within the Relational Database

## One-to-Many (1: \*) Relationships

The one-to-many (1:) relationship is the most common and preferred type in relational database design. This relationship exists when a single entity instance of one type is associated with multiple instances of another type. This structure ensures data normalization and minimizes redundancy.

**Example:**

* Customer and Orders: One customer can place many orders, but each order is placed by only one customer. In this case, the customer entity has a primary key that is referenced as a foreign key in the orders entity.

**Implementation:**

* In a relational database, a one-to-many relationship is implemented by placing the primary key of the "one" side (e.g., Customer) as a foreign key in the "many" side (e.g., Orders).

**Example SQL Code:**

CREATE TABLE Customer (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(50)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

OrderDate DATE,

CustomerID INT,

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)

);

## One-to-One (1:1) Relationships

The one-to-one (1:1) relationship is less common and typically indicates that two entities are closely related but not combined for practical or security reasons. Each instance of an entity relates to one and only one instance of another entity.

**Example:**

* Person and Passport: Each person has one passport, and each passport is issued to one person. This relationship can be used for separating sensitive information or optimizing database performance.

**Implementation:**

* In a relational database, a one-to-one relationship is implemented by ensuring that the primary key of one table is also the primary key or a unique foreign key in another table.

**Example SQL Code:**

CREATE TABLE Person (

PersonID INT PRIMARY KEY,

PersonName VARCHAR(50)

);

CREATE TABLE Passport (

PassportID INT PRIMARY KEY,

PassportNumber VARCHAR(20),

PersonID INT UNIQUE,

FOREIGN KEY (PersonID) REFERENCES Person(PersonID)

);

## Many-to-Many (\*: \*) Relationships

The many-to-many (:) relationship is where multiple instances of one entity are associated with multiple instances of another entity. Direct implementation of many-to-many relationships in relational databases is not possible. Instead, they are broken down into two one-to-many relationships using an associative (junction) table.

**Example:**

* Students and Courses: A student can enroll in many courses, and a course can have many students.

**Implementation:**

* An associative table (e.g., Enrollment) is created to hold foreign keys that reference the primary keys of the entities involved in the many-to-many relationship.

**Example SQL Code:**

CREATE TABLE Students (

StudentID INT PRIMARY KEY,

StudentName VARCHAR(50)

);

CREATE TABLE Courses (

CourseID INT PRIMARY KEY,

CourseName VARCHAR(50)

);

CREATE TABLE Enrollment (

StudentID INT,

CourseID INT,

EnrollmentDate DATE,

PRIMARY KEY (StudentID, CourseID),

FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)

);

## Summary

* One-to-Many (1:) Relationships: This is the preferred relationship type in relational databases. It is implemented by placing the primary key of the "one" entity as a foreign key in the "many" entity.
* One-to-One (1:1) Relationships: Rarely used but useful for splitting closely related data or managing sensitive information. It is implemented by sharing a primary key or using a unique foreign key.
* Many-to-Many (:) Relationships: Cannot be directly implemented and must be converted into two one-to-many relationships with an associative table.

## UML Class Diagram and Multiplicity

In UML class diagrams, relationships are represented as associations among objects, and the multiplicity element can represent many-to-many relationships directly. However, for practical database implementation, such relationships are broken down into multiple one-to-many relationships, often involving an association class to encapsulate the relationship's attributes and behaviors. This will be further explored in Chapter 5 with data modeling techniques like Entity Relationship Diagrams (ERDs).

# Data Redundancy Revisited

## Understanding Data Redundancy

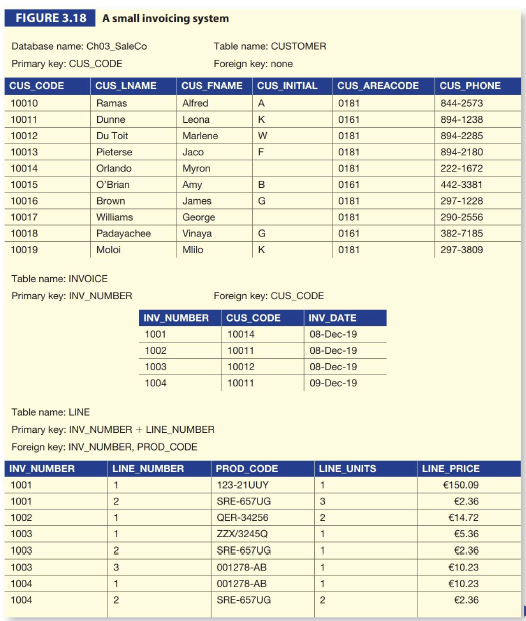
* **Data redundancy** - refers to the unnecessary duplication of data in a database.
* **Data anomalies** - are errors or inconsistencies that result from improper database operations, such as insert, update, or delete actions.
* **Anomalies** can destroy the effectiveness of a database, leading to incorrect data or system errors.

## Foreign Keys and Data Redundancy

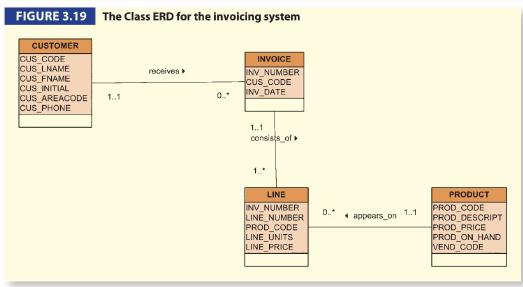
* **Foreign keys** are attributes that establish relationships between tables, minimizing data redundancies by referencing primary keys in other tables.
* While foreign keys may appear multiple times in a table, they are not considered redundant because their purpose is to maintain relational integrity and allow the linking of tables.

## Planned Redundancy

* The goal is to balance **design elegance**, **processing speed**, and **information requirements**.
* Planned redundancies is often necessary to ensure transaction speed and meet information requirements.
* Redundancies are carefully managed to prevent the potential for damage and to maintain the accuracy and reliability of the database.







* **Historical Accuracy:** The product price is stored in both the `PRODUCT` table and the `LINE` table. This redundancy ensures that the price at the time of sale is preserved, even if the product price changes in the future.
* **Data Consistency:** By storing the price at the time of the transaction, past revenue calculations remain accurate and reliable.
* **Practical Considerations:** In some cases, like the automatic generation of line numbers in invoicing software, redundancies exist but do not cause anomalies.

## Summary

* The use of foreign keys **minimizes** data redundancies but does not eliminate them.
* **Planned** **redundancies** are designed to ensure the database meets **performance** and **accuracy** requirements.
* Properly managed, these redundancies do not lead to anomalies and can enhance the practical usability of the database system.

# Indexes

## Concept and Importance

* **Indexes** in a database function similarly to indexes in books or library catalogs. They allow for quick retrieval of data without having to search through every row in a table.
* **Index** **Key**: The reference point used to locate the data.
* **Pointers**: Indicate the actual location of the data in the table.
* Consider the PAINTING table in the Ch03\_Museum database:

|  |  |  |  |
| --- | --- | --- | --- |
| **PAINTING\_ID** | **PAINTER\_NUM** | **TITLE** | **YEAR** |
| 1 | 123 | Mona Lisa | 1503 |
| 2 | 123 | The Last Supper | 1498 |
| 3 | 126 | Starry Night | 1889 |
| 4 | 123 | Lady with an Ermine | 1489 |
| 5 | 126 | Irises | 1889 |

An index on the `PAINTER\_NUM` attribute might look like this:

|  |  |
| --- | --- |
| **PAINTER\_NUM** | **POINTERS** |
| 123 | 1, 2, 4 |
| 126 | 3, 5 |

This index allows quick lookup of paintings by a particular painter.

## Benefits of Indexes

* 1. **Efficient Data Retrieval:** Indexes significantly speed up the process of finding specific rows.
  2. **Ordering** **Data**: Indexes can also help in retrieving data in a sorted order based on specific attributes.
  3. **Composite** **Indexes**: An index can be created on multiple columns, facilitating complex queries. For example, creating an index on `VEND\_CODE` and `PROD\_CODE` in the PRODUCT table allows retrieval of products ordered by vendor and within each vendor by product.

## Automatic Index Creation

* When a primary key is defined, the DBMS automatically creates a unique index on that primary key column(s).
* A unique index ensures that each key corresponds to a single row, which enforces the uniqueness of primary keys.

## Practical Uses

* Search and Sort: Indexes facilitate quick searching and sorting of data. For instance, indexing customer last names can enable fast retrieval of customer data in alphabetical order.
* Join Operations: Indexes can speed up join operations by quickly locating matching rows across tables.

## Creating an Index

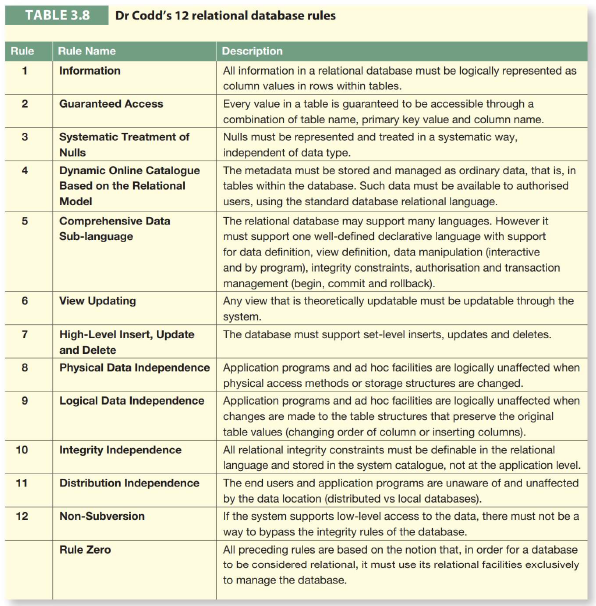
* Creating an index is straightforward with SQL. For instance, to create an index on the `PAINTER\_NUM` column in the PAINTING table, you might use the following SQL command:

CREATE INDEX idx\_painter\_num ON PAINTING (PAINTER\_NUM);

In summary, indexes are essential tools in relational databases for optimizing data retrieval, maintaining data order, and supporting efficient join operations. They are integral to the performance and functionality of a database system.

# 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. Dr Codd’s list, shown in Table 3.8, serves as a frame of reference for what a truly relational database should be. Bear in mind that even the dominant database vendors do not fully support all 12 rules.



# Summary

* Tables are the basic building blocks of a relational database. A grouping of related entities, known as an entity set, is stored in a table. Conceptually speaking, the relational table is composed of intersecting rows (tuples) and columns. Each row represents a single entity, and each column represents the characteristics (attributes) of the entities.
* Keys are central to the use of relational tables. Keys define functional dependencies; that is, other attributes are dependent on the key and can, therefore, be found if the key value is known. A key can be classified as a superkey, a candidate key, a primary key, a secondary key or a foreign key.
* Each table row must have a primary key. The primary key is an attribute or a combination of attributes that uniquely identifies all remaining attributes found in any given row. Because a primary key must be unique, no null values are allowed if entity integrity is to be maintained.
* Although the tables are independent, they can be linked by common attributes. Thus, the primary key of one table can appear as the foreign key in another table to which it is linked. Referential integrity dictates that the foreign key must contain values that match the primary key in the related table or must contain nulls.
* Once you know the relational database basics, you can concentrate on design. Good design begins by identifying appropriate entities and attributes, and the relationships among the entities.
* Those relationships (1:1, 1:\* and \*:\*) can be represented using ERDs. The use of ERDs allows you to create and evaluate simple logical design. The 1:\* relationships are most easily incorporated in a good design; you just have to make sure that the primary key of the 1 is included in the table of the many.

# Review Questions

1. **What is the difference between a database and a table?** A **database** is a structured set of data. It is an organized collection of related information that is managed and stored efficiently. On the other hand, a **table** is a component of a database. It is a structured set of data within the database, consisting of rows (records) and columns (fields).
2. **What does it mean to say that a database displays both entity integrity and referential integrity?** When a database displays both **entity integrity** and **referential integrity**, it means that the database is following certain rules to maintain its consistency and avoid data anomalies. Entity integrity ensures that every record in a table is unique and identifiable, typically enforced by a primary key. Referential integrity ensures that relationships between tables remain consistent, typically enforced by foreign keys.
3. **Why are entity integrity and referential integrity important in a database?** **Entity integrity** and **referential integrity** are important in a database to maintain data accuracy and consistency. Entity integrity ensures that there are no duplicate records within a table and each record can be uniquely identified. Referential integrity ensures that the relationships between tables are maintained, preventing inconsistencies and ensuring that the data in related tables matches.
4. **What can a NULL value represent?** A **NULL** value in a database can represent several things: the attribute does not apply to the record, the attribute’s value is unknown, or the attribute’s value is not available at the moment. It’s important to note that NULL is different from zero or an empty string, which are actual values.
5. **What is the domain of an attribute?** The **domain** of an attribute refers to the set of allowable values that the attribute can take. It defines the type of data that can be stored in the attribute, such as integer, string, date, etc., and any constraints on the values, such as a range of numbers or a specific format for dates.
6. **Entities:** Students & Lecturers