Introduction To Database

CT042-3-1-IDB

0005 - Relational Model (1)

Topic & Structure of The Lesson

- Relational Table
- Relational Model Components
- Data Integrity Rules
- Data Dictionary

Learning Outcomes

At the end of this topic, You should be able to

- Explain the relational model's basic components
- Explain how relations are organized in tables
- Explain how data redundancy is handled in the relational database model

Key Terms You Must Be Able To Use

If you have mastered this topic, **you should be able to use the following terms correctly in your assignments and exams**:

- Primary Key
- Foreign Key
- Candidate Key
- Attribute
- Domain

A Logical View of Data

1. In The Context Of Database Systems

The relational model represents a significant evolution in how data is organized, managed, and accessed compared to earlier models like hierarchical and network models. Let's break down the key concepts and provide examples to clarify these ideas.

1A. Data, Metadata, and the Role of the DBMS

- **Data** refers to the actual information stored in the database, such as customer names, product details, and sales transactions.
- **Metadata** is data about the data. It describes the structure of the database, such as the tables, columns, data types, and relationships between tables.

The **Database Management System (DBMS)** is the software that manages the database. It handles tasks like:

- **Storing and retrieving data**: Ensuring data is stored efficiently and can be retrieved quickly.
- **Managing access**: Controlling who can view or modify the data.
- **Maintaining the database structure**: Ensuring that the metadata stays consistent as data is added, removed, or updated.

Example: In a library database:

- **Data**: Book titles, authors, ISBNs, borrower details.
- **Metadata**: The structure of the tables (e.g., a "Books" table with columns for Title, Author, ISBN).

1B. Limitations of File Systems and the Need for DBMS

Traditional file systems, where data is stored in files, have several limitations:

- **Data redundancy and inconsistency**: The same data might be duplicated in multiple files, leading to inconsistencies.
- **Difficulty in accessing data**: To find specific data, programs had to be written to navigate through files, which could be complex and error-prone.
- **Lack of security and data integrity**: File systems lacked robust mechanisms to control access or ensure data validity.

The DBMS overcomes these limitations by providing a unified platform to manage data efficiently.

Example: In a company using file systems, employee information might be stored in separate files for payroll, benefits, and contact details. If an employee changes their address, it must be updated in multiple places, increasing the risk of errors. A DBMS would centralize this information, ensuring consistency.

1C. Complexity of Hierarchical and Network Models

- **Hierarchical Model**: Organizes data in a tree-like structure where each record has a single parent. This can be restrictive because it doesn't handle many-to-many relationships well.

- **Network Model**: Allows more complex relationships by allowing records to have multiple parents, but it can become complicated to design and navigate.

These models focus more on how data is physically stored, which can make the database design process cumbersome and inflexible.

Example: In a hierarchical model, if you have a database of departments and employees, each employee might only be associated with one department, making it difficult to model situations where employees work in multiple departments.

1D. The Relational Model's Simplicity

The **Relational Model** abstracts away the physical storage details and allows the database designer to focus on the logical relationships between data. In short, the relational model enables you to view data logically rather than physically. Data is stored in tables (also known as relations), where each table consists of rows (records) and columns (fields). Although the use of a table, quite unlike that of a file, has the advantages:

Key Advantages:

- **Data Independence**: Changes to the data structure do not require changes to the applications that use the data.
- **Simplified Design**: The relational model's logical simplicity makes it easier to design databases that accurately reflect the real-world entities they represent.

Example: In a relational database, you might have a "Students" table and a "Courses" table. A third table, "Enrollments," could link students to courses, allowing a many-to-many relationship without complex hierarchical or network structures.

1E. Logical vs. Physical View

The relational model emphasizes a logical view of data. This means that users and applications interact with data at a conceptual level, without needing to worry about how it's physically stored on disk.

Example: When querying a database to find all students enrolled in a particular course, the user interacts with the data through SQL queries (a logical interaction), without needing to know how the data is organized on the storage medium.

1F. The Role of Tables in the Relational Model

In the relational model, **tables** are the primary way data is organized. Each table represents an entity (like "Employees" or "Products"), and each row in the table represents a single instance of that entity.

Example:

- **Employees Table**:
  - **Columns**: EmployeeID, Name, Department, Salary.
  - **Rows**: Each row represents a different employee.

- **Departments Table**:
  - **Columns**: DepartmentID, DepartmentName, ManagerID.
  - **Rows**: Each row represents a different department.

Summary

The relational model revolutionized database design by shifting the focus from the physical arrangement of data to its logical structure. This abstraction simplifies the design process and makes databases more flexible and easier to use, much like how an automatic transmission in a car simplifies driving by handling the complex mechanics of gear shifting.

2. Tables and Their Characteristics in Relational Databases

In a relational database, data is organized in structures known as tables. These tables are crucial in facilitating the logical view of data, allowing for easy management, retrieval, and manipulation.

Here's a detailed breakdown of what tables are, their characteristics, and how they relate to other components in a database system.

2A. What is a Table?

A table is a two-dimensional structure composed of rows and columns, where:

- **Rows (Tuples):** Each row represents a single instance of an entity. For example, in a table called STUDENT, each row would represent a different student.
- **Columns (Attributes):** Each column represents a different attribute of the entity. For instance, in the STUDENT table, columns might include Student_ID, Name, Date_of_Birth, etc.

Although tables are stored in databases, they are more than just physical storage units; they are logical constructs that help users understand and manipulate data.

2B. Characteristics of Relational Tables

i. Two-Dimensional Structure:

- A table is perceived as a two-dimensional grid consisting of rows and columns. This grid-like structure makes it intuitive to work with and understand data.

Example:

- A COURSE table might look like this:

Course_ID	Course_Name	Credits
101	Database	3
102	Networks	4
103	AI	3

2B. Characteristics of Relational Tables

ii. Rows Represent Entity Occurrences:

- Each row in a table represents a specific occurrence of an entity within the entity set.

Example:

- In the STUDENT table, each row represents a specific student with unique details.

Student_ID	Name	Date_of_Birth
001	Alice	2000-05-10
002	Bob	1999-07-21
003	Charlie	2001-02-17

iii. Columns Represent Attributes:

- Each column in a table represents an attribute, which is a characteristic of the entity. Columns have distinct names.

Example:

- The COURSE table has columns like Course_ID, Course_Name, and Credits, each representing an attribute of the COURSE entity.

iv. Intersection of Row and Column Represents a Data Value:

- The intersection point between a row and a column in the table holds a **single data value**.

Example:

- In the STUDENT table, the intersection of the Student_ID column and the first row holds the value 001, representing Alice's student ID.

2B. Characteristics of Relational Tables

v. Column Values Conform to a Data Format:

- All the values within a particular column must conform to the same data type and format.

Example:

- In the STUDENT table, the Date_of_Birth column would typically store dates, ensuring that all entries in this column are of the Date type.

vi. Columns Have Specific Attribute Domains:

- Each column has an attribute domain, which is the set of possible values that can be stored in that column or range of the value you stored.

Example:

- The Credits column in the COURSE table might have a domain of integer values between 0 and 5, meaning only numbers within this range are valid.

vii. Order of Rows and Columns is Immaterial:

- The sequence in which rows and columns are stored or displayed does not affect the meaning of the data.

Example:

- Whether the COURSE table displays Course_ID first or Course_Name first does not alter the information the table conveys.

viii. Unique Identifier for Each Row:

- Each table must have one or more attributes that uniquely identify each row, known as a primary key.

Example:

- In the STUDENT table, the Student_ID column can serve as a primary key because it uniquely identifies each student.

Student_ID (PK)	Name	Date_of_Birth
001	Alice	2000-05-10
002	Bob	1999-07-21
003	Charlie	2001-02-17

2C. Key Terminology and Clarifications

Relation:

- In relational databases, a table is often referred to as a "**relation**." because the relational model's creator, E. F. Codd, used the two terms as synonyms.
- You can think of a table as a persistent representation of a logical relation—that is, a relation whose contents can be permanently saved for future use. Exp: we stored data in a table for future use such as check the inventory, analyze the data, how many female or male customer...

Attribute Domain:

- This defines the allowed set of values for an attribute. For instance, a GPA attribute might have a domain of numeric values between 0.0 and 4.0.

Terminology Confusion:

- Terms like "record," "field," and "file" are often used interchangeably with "row," "column," and "table" in everyday language, especially by those familiar with file systems.
- However, in the context of databases, it's crucial to understand that a table is a logical construct, not a physical one.

2D. Example in Practice

Consider a LIBRARY database containing the following tables:

- **BOOKS** table:

Book_ID	Title	Author	Year_Published
1001	"Database Systems"	E. F. Codd	1970
1002	"Computer Networks"	A. Tanenbaum	1981
1003	"AI: A Modern Approach"	S. Russell	1995

- **BORROWERS** table:

Borrower_ID	Name	Membership_Date
2001	Alice	2022-01-15
2002	Bob	2023-03-10
2003	Charlie	2023-05-22

- **LOANS** table:

Loan_ID	Borrower_ID	Book_ID	Loan_Date	Return_Date
3001	2001	1001	2024-07-01	2024-07-15
3002	2002	1002	2024-08-05	2024-08-20

In this database:

- The BOOKS table records details about each book in the library.
- The BORROWERS table keeps track of library members.
- The LOANS table records when books are borrowed and returned.

Each table is structured to maintain data integrity, ensure easy data retrieval, and support the relationships between entities (e.g., which borrower has borrowed which book).

Understanding these foundational characteristics of tables is crucial for designing, managing, and interacting with relational databases effectively.

2E. STUDENT Table Attribute Values

Table name: STUDENT **Database name: Ch03_TinyCollege**

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT	STU_DOB	STU_HRS	STU_CLASS	STU_GPA	STU_TRANSFER	DEPT_CODE	STU_PHONE	PROF_NUM
321452	Bowser	William	C	12-Feb-2000	42	So	2.84	No	BIOL	2134	205
324257	Smithson	Anne	K	15-Nov-2001	81	Jr	3.27	Yes	CIS	2256	222
324258	Brewer	Juliette		23-Aug-1999	36	So	2.26	Yes	ACCT	2256	228
324269	Oblonski	Walter	H	16-Sep-1996	66	Jr	3.09	No	CIS	2114	222
324273	Smith	John	D	30-Dec-1988	102	Sr	2.11	Yes	ENGL	2231	199
324274	Katinga	Raphael	P	21-Oct-2000	114	Sr	3.15	No	ACCT	2267	228
324291	Robertson	Gerald	T	08-Apr-1999	120	Sr	3.87	No	EDU	2267	311
324299	Smith	John	B	30-Nov-2000	15	Fr	2.92	No	ACCT	2315	230

STU_NUM = Student number
STU_LNAME = Student last name
STU_FNAME = Student first name
STU_INIT = Student middle initial
STU_DOB = Student date of birth
STU_HRS = Credit hours earned
STU_CLASS = Student classification
STU_GPA = Grade point average
STU_TRANSFER = Student transferred from another institution
DEPT_CODE = Department code
STU_PHONE = 4-digit campus phone extension
PROF_NUM = Number of the professor who is the student's advisor

2E. STUDENT Table Attribute Values

Let's break down the details of the STUDENT table shown in Figure 3.1, following the **characteristics of a relational table**:

1. Two-Dimensional Structure

The STUDENT table is represented as a two-dimensional grid composed of rows and columns:

- **Rows (Tuples):** The table contains 8 rows, each representing a unique student.

- **Columns (Attributes):** There are 12 columns, each representing a specific attribute of the student entity, such as STU_NUM, STU_LNAME, STU_FNAME, etc.

2. Single Entity Occurrences

Each row in the STUDENT table describes a single entity occurrence within the entity set. The entity set, represented by the STUDENT table, includes eight distinct students.

- **Example:** Row 4 represents the student **Walter H. Oblonski**, with details like STU_NUM = 324269, STU_DOB = 16-Sep-1996, and so on.

3. Distinct Column Names

Each column in the STUDENT table represents a distinct attribute, and each has a unique name.

- **Example:** STU_NUM is the student number, STU_LNAME is the last name, STU_FNAME is the first name, etc.

4. Matching Attribute Characteristics

All values in each column conform to the characteristics of the attribute they represent:

- **Numeric:** Attributes like STU_HRS (credit hours earned) and STU_GPA (grade point average) are numeric, meaning they can be used for mathematical operations.

- **Character:** Attributes like STU_CLASS (student classification) and STU_PHONE (campus phone extension) are character-based, meaning they contain text or symbols.

- **Date:** The STU_DOB (student date of birth) column is a date attribute.

- **Logical:** The STU_TRANSFER (indicating if a student transferred from another institution) is a logical attribute with values Yes or No.

5. Attribute Domain

Each column has a specific domain, or a range of permissible values:

- **Example:** The STU_GPA column has a domain from 0 to 4, meaning the GPA values must fall within this range.

6. Order of Rows and Columns

The order of the rows and columns in the STUDENT table does not affect the functionality or interpretation of the data.

- **Example:** Rearranging the columns or rows would not change the meaning or content of the data. For instance, you can swap the STU_LNAME and STU_FNAME columns, but it would still represent the same student details.

7. Primary Key

A primary key is an attribute or a combination of attributes that uniquely identifies each row in the table. In the STUDENT table, STU_NUM is the primary key:

- **Why STU_NUM as the Primary Key?:**
  - It uniquely identifies each student.
  - Columns like STU_LNAME (last name) or even a combination of STU_LNAME and STU_FNAME would not work as a primary key because there can be multiple students with the same name (e.g., two students named John Smith).

Summary of the STUDENT Table

The STUDENT table is a classic example of a relational table with well-defined characteristics:

- It organizes student data in a logical, two-dimensional format.
- It uses columns to represent various attributes of students, each with a distinct name and specific data type.
- It ensures data integrity through the use of a primary key (STU_NUM) and conforms to domain restrictions for each attribute.

Understanding these characteristics is crucial for designing and interacting with relational databases, ensuring that data is stored, retrieved, and managed effectively.

Keys

1. Keys in Relational Database Systems

In relational databases, keys are crucial for ensuring that each record (row) in a table is uniquely identifiable, establishing relationships between tables, and maintaining data integrity. Let's explore the different types of keys in detail with examples.

In relational databases, a key is a value or set of values that uniquely identifies a row or rows in a table. Keys are used to establish relationships between tables and columns, and to ensure data integrity. They are also essential for identifying and retrieving specific records within a table.

The **Keys are important** because to controlled redundancy:

- Makes the relational database work.
- Tables within the database share common attributes that enable the tables to be linked together.
- Multiple occurrences of values in a table are not redundant when they are required to make the relationship work.
- Redundancy exists only when there is unnecessary duplication of attribute values.

1A. Primary Key (PK)

Definition:

- A primary key is an attribute (or a combination of attributes) that uniquely identifies a single record within a table. It **cannot be NULL** and **must contain unique values for each record**.

Role:

- The primary key ensures that each row in the table can be uniquely identified. This is crucial for operations like searching for, updating, or deleting a specific record.

Example:

- Consider the STUDENT table:

Student_ID (PK)	First_Name	Last_Name	DOB	Email
101	John	Doe	2000-01-15	john.doe@gmail.com
102	Jane	Smith	1999-03-22	jane.smith@gmail.com

Here, Student_ID is the primary key because it uniquely identifies each student. No two students can have the same Student_ID.

1B. Superkey

Definition:

- A superkey is a set of one or more attributes that, when taken together, can uniquely identify a record in a table.

Role:

- Superkeys include all combinations of attributes that can uniquely identify rows. This means the primary key is a minimal superkey, but there can be other superkeys that include additional attributes.

Example:

For the STUDENT table, possible superkeys could be:

- {Student_ID}: This is the primary key.
- {Student_ID, Email}: Although Student_ID alone is sufficient, adding Email still uniquely identifies each record, making this a superkey.
- {Student_ID, First_Name, Last_Name}: Again, although Student_ID alone suffices, this combination is still a superkey.

1C. Candidate Key

Definition:

- A candidate key is a minimal superkey—one that can uniquely identify a record but does not contain any unnecessary attributes.

Role:

- Candidate keys are potential primary keys. They are minimal by definition, meaning you cannot remove any attributes from them without losing their ability to uniquely identify records.

Example: In the STUDENT table:

- {Student_ID} is a candidate key.
- {Email} could be another candidate key if the database design ensures that each email is unique to a student.

Only one of these candidate keys will be chosen as the primary key.

1D. Secondary Key

Definition:

- A secondary key is an attribute or a set of attributes that is used strictly for data retrieval purposes. It is not necessarily unique and is not the primary key.

Role:

- Secondary keys are useful for queries that retrieve records based on non-unique attributes.

Example: In the STUDENT table:

- {Last_Name} could be a secondary key. If you often query the database to retrieve students by their last name, indexing on Last_Name would improve query performance.

- {DOB} could be another secondary key if you frequently search for students born on a specific date.

Summary

- **Primary Key (PK)**: Uniquely identifies a record in a table. Example: Student_ID.

- **Superkey**: Any set of attributes that can uniquely identify a record. Example: {Student_ID, Email}.

- **Candidate Key**: A minimal superkey, a candidate for being the primary key. Example: Email.

- **Secondary Key**: Used for data retrieval; not necessarily unique. Example: Last_Name.

Understanding these keys helps in designing databases that are efficient, maintain data integrity, and support complex queries.

STUDENT TABLE

Table name: STUDENT **Database name: Ch03_TinyCollege**

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT	STU_DOB	STU_HRS	STU_CLASS	STU_GPA	STU_TRANSFER	DEPT_CODE	STU_PHONE	PROF_NUM
321452	Bowser	William	C	12-Feb-2000	42	So	2.84	No	BIOL	2134	205
324257	Smithson	Anne	K	15-Nov-2001	81	Jr	3.27	Yes	CIS	2256	222
324258	Brewer	Juliette		23-Aug-1999	36	So	2.26	Yes	ACCT	2256	228
324269	Oblonski	Walter	H	16-Sep-1996	66	Jr	3.09	No	CIS	2114	222
324273	Smith	John	D	30-Dec-1988	102	Sr	2.11	Yes	ENGL	2231	199
324274	Katinga	Raphael	P	21-Oct-2000	114	Sr	3.15	No	ACCT	2267	228
324291	Robertson	Gerald	T	08-Apr-1999	120	Sr	3.87	No	EDU	2267	311
324299	Smith	John	B	30-Nov-2000	15	Fr	2.92	No	ACCT	2315	230

STU_NUM = Student number
STU_LNAME = Student last name
STU_FNAME = Student first name
STU_INIT = Student middle initial
STU_DOB = Student date of birth
STU_HRS = Credit hours earned
STU_CLASS = Student classification
STU_GPA = Grade point average
STU_TRANSFER = Student transferred from another institution
DEPT_CODE = Department code
STU_PHONE = 4-digit campus phone extension
PROF_NUM = Number of the professor who is the student's advisor

2. Database Dependencies

2A. Determination: The Role of a Key

In a database, determination refers to the concept that knowing the value of one attribute (or a set of attributes) allows you to determine the value of another attribute. This concept is critical in understanding how databases function, as it underpins the relationships between different pieces of data.

Example:

- Consider the formula **revenue - cost = profit**. If you know the revenue and cost, you can determine the profit. This is a form of determination. In a database context, this might look like knowing a student's ID number (attribute A) allows you to determine their last name (attribute B).

- If you know the student ID (STU_NUM), you can look up and determine the student's last name (STU_LNAME).

2B. Functional Dependence

Functional dependence occurs when the value of one attribute (or set of attributes) determines the value of another attribute. It is denoted as A → B, meaning attribute B is functionally dependent on attribute A.

- **Example:** In the STUDENT table:
  - STU_NUM → STU_LNAME means that the student number (STU_NUM) determines the student's last name (STU_LNAME). Given a specific student number, you can determine the exact last name of the student.

- Similarly, STU_NUM → (STU_LNAME, STU_FNAME, STU_GPA) means that the student number determines multiple attributes: last name, first name, and GPA.

2C. Determinants and Dependents

Determinant:

- An attribute whose value determines the value of another attribute.

Dependent:

- An attribute whose value is determined by another attribute.

Example:

- In the relationship STU_NUM → STU_LNAME, STU_NUM is the determinant because it determines the value of STU_LNAME, which is the dependent.

2D. Composite Keys and Full Functional Dependence

A composite key is a key that consists of more than one attribute. Full functional dependence occurs when an attribute is functionally dependent on the entire composite key and not just a part of it.

- **Example:**
  - STU_NUM → STU_GPA indicates that the student number alone determines the GPA.
  - (STU_NUM, STU_LNAME) → STU_GPA indicates that the combination of student number and last name determines the GPA. However, in this case, the last name (STU_LNAME) is not necessary because STU_NUM alone can determine STU_GPA. Therefore, this is not a full functional dependency.

- A full functional dependency example would be a situation where the combination of attributes is necessary to determine another attribute. If removing one of the attributes from the determinant breaks the functional dependency, it is a full functional dependence.
  - **Example:** (STU_FNAME, STU_LNAME, STU_INIT, STU_PHONE) → STU_DOB means that all these attributes together determine the student's date of birth. If you remove any one of these attributes, you may not be able to determine the exact date of birth, indicating a full functional dependency.

2E. Application in Database Design

Understanding determination and functional dependencies is crucial in designing efficient databases. It helps in normalizing tables, eliminating redundancy, and ensuring data integrity.

Example of Redundant Dependency:

- Suppose a table has a composite key (STU_NUM, STU_LNAME) and you have a dependency (STU_NUM, STU_LNAME) → STU_GPA.
- If STU_NUM alone can determine STU_GPA, including STU_LNAME in the key introduces redundancy. This is where normalizing the database by breaking down the table or removing unnecessary attributes from the determinant is essential.

Conclusion

In summary, determination and functional dependencies are fundamental concepts in database design that ensure relationships among attributes are well-defined. These principles guide the creation of keys, normalization of tables, and overall efficiency of the database system. Understanding these concepts allows for better database management and design, avoiding data redundancy and maintaining data integrity.

3. Types of Keys

3A. Key Concepts Overview

- In relational databases, a **key** is an attribute or a set of attributes that can uniquely identify a row in a table. Keys are crucial in maintaining the integrity of the database by ensuring that each record is unique and easily retrievable.

3B. Composite Key

A **composite key** is a key that is composed of more than one attribute. Composite keys are necessary when a single attribute is not sufficient to uniquely identify a row in a table, so a combination of attributes is used.

Example:

- Consider a situation where a university stores information about students in courses. If the table stores StudentID, CourseID, and Semester, none of these attributes alone might be able to uniquely identify a record (because a student can enroll in multiple courses, and a course can be taken by multiple students across different semesters).
- However, the combination of StudentID, CourseID, and Semester would uniquely identify a student's enrollment in a specific course during a specific semester. This combination forms a composite key.

  In the context of the STUDENT table:
  - STU_NUM → STU_GPA indicates that the student number (STU_NUM) alone determines the GPA, so STU_NUM is a single key attribute.
  - (STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE) → STU_HRS indicates that a combination of last name, first name, middle initial, and phone number is needed to determine the credit hours (STU_HRS). This combination of attributes is a composite key because it uses multiple attributes to determine a dependent attribute.

3C. Key Attribute

A **key attribute** is an attribute that is part of a key, whether it's a single key or part of a composite key.

- **Example:**
  - In the functional dependency STU_NUM → STU_GPA, STU_NUM is the key attribute.
  - In the composite key (STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE) → STU_HRS, all four attributes (STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE) are key attributes because they all contribute to the key.

3D. Types of Keys in a Relational Model

Understanding different types of keys is fundamental to designing a database schema that effectively manages data.

- **Primary Key:**
  - A primary key is the main key in a table that uniquely identifies each record. It can be a single attribute or a composite key.
  - **Example:** In a STUDENT table, STU_NUM might be used as the primary key because it uniquely identifies each student.

- **Candidate Key:**
  - A candidate key is any key that could serve as a primary key. It is a minimal superkey (a set of attributes that uniquely identifies a record).
  - **Example:** If a table has StudentID and Email as unique identifiers, both can be candidate keys.

- **Superkey:**
  - A superkey is a set of attributes that can uniquely identify a record, though it might contain additional attributes not necessary for uniqueness.
  - **Example:** (STU_NUM, STU_LNAME) could be a superkey, even though STU_NUM alone could be sufficient as a primary key.

- **Foreign Key:**
  - A foreign key is an attribute or a set of attributes in one table that references the primary key in another table. It is used to establish and enforce a link between the data in two tables.
  - **Example:** If a GRADES table has a StudentID that references STU_NUM in the STUDENT table, StudentID is a foreign key.

Conclusion

Understanding the different types of keys—particularly composite keys and key attributes—is vital for effective database design. Keys ensure data integrity and facilitate efficient data retrieval by uniquely identifying records in a table. By mastering the concepts of keys, you can create more robust and reliable database systems.

3E. Explanation of Superkey with Examples

1. Superkey

A **superkey** is an attribute or a combination of attributes that can uniquely identify each row (or record) in a table. This means that a superkey functionally determines every other attribute in the row.

Example:

In the STUDENT table:

- STU_NUM (Student Number) alone can uniquely identify each student, making it a superkey.

- Any combination of attributes that includes STU_NUM, such as (STU_NUM, STU_LNAME) or (STU_NUM, STU_LNAME, STU_INIT), is also a superkey. This is because STU_NUM alone can uniquely identify each row, so adding more attributes doesn't change its ability to do so.

2. Importance of Superkeys

Superkeys are important because they ensure the uniqueness of rows in a table. However, not all keys are superkeys. A superkey must be able to uniquely identify each row in the table, but the reverse is not always true for other types of keys.

Table 3.2 Student Classification

Hours Completed	Classification
Less than 30	Fr
30–59	So
60–89	Jr
90 or more	Sr

3. Gigantic State University Example

The provided Table 3.2 shows the classification of students based on the number of credit hours they have completed:

- **Classification Rules:**
  - Less than 30 hours: Freshman (Fr)
  - 30–59 hours: Sophomore (So)
  - 60–89 hours: Junior (Jr)
  - 90 or more hours: Senior (Sr)

- **Functional Dependency:** Based on the classification rules, you can write the functional dependency as STU_HRS → STU_CLASS, meaning the number of hours completed (STU_HRS) determines the student's classification (STU_CLASS).

However, the reverse is not true:

- **Counter-Example:** The classification (like Jr for Junior) does not determine the exact number of hours completed (STU_HRS). For example, a Junior could have completed 62 hours or 84 hours. Thus, STU_CLASS does not uniquely determine STU_HRS, meaning it's not a functional dependency in the opposite direction.

4. Understanding Superkeys in the Context

In the context of STU_HRS and STU_CLASS, neither is a superkey because neither uniquely identifies a row in the table.

Clarification:

- STU_NUM is a superkey because it can uniquely identify each student record.
- However, the combination of STU_HRS and STU_CLASS is not a superkey because it cannot uniquely identify a student; multiple students can have the same number of hours completed and the same classification.

Conclusion

- A **superkey** is a powerful tool in relational databases, as it ensures that each row is unique. The concept of superkeys is essential in understanding how different attributes in a table relate to one another and how to maintain data integrity.

- By knowing which attributes or combinations of attributes can serve as superkeys, you can better design and normalize database tables to avoid redundancy and ensure accurate data retrieval.

3F. Explanation of Candidate Key with Examples

1. Candidate Key

A **candidate key** is a minimal superkey, meaning it is the smallest set of attributes that can uniquely identify a record in a table. Unlike a superkey, a candidate key does not contain any unnecessary attributes—no subset of a candidate key can uniquely identify a record.

Example:

- In the STUDENT table, STU_NUM (Student Number) is a candidate key because it uniquely identifies each student and does not contain any extraneous attributes.

2. Distinguishing Between Superkey and Candidate Key

While every candidate key is a superkey, not every superkey is a candidate key. The key difference is minimality:

- A **superkey** can have additional attributes that are not necessary for uniqueness.
- A **candidate key** is the minimal form of a superkey without any redundant attributes.

- **Example:**
  - **Superkey:** (STU_NUM, STU_LNAME) is a superkey because it can uniquely identify each student. However, it is not a candidate key because STU_NUM alone is sufficient for uniqueness, making STU_LNAME redundant.

  - **Candidate Key:** STU_NUM alone is a candidate key because removing any attribute from it would no longer uniquely identify the rows in the table.

3. Full Functional Dependency

A candidate key is always based on a full functional dependency. This means that every attribute in the table is functionally dependent on the candidate key.

Example:

Consider the functional dependencies in the STUDENT table:

- STU_NUM → STU_LNAME, STU_FNAME, STU_GPA, etc. indicates that all other attributes are functionally dependent on STU_NUM. Thus, STU_NUM is a candidate key.

However, in the dependency (STU_NUM, STU_LNAME) → STU_GPA, STU_LNAME is not necessary because STU_NUM alone is sufficient to determine STU_GPA. Therefore, (STU_NUM, STU_LNAME) is a superkey but not a candidate key.

4. Multiple Candidate Keys

A table can have more than one candidate key. If multiple sets of attributes can uniquely identify each record, each set is a candidate key.

Example:

- If the STUDENT table also included the students' Social Security numbers as STU_SSN, then STU_SSN would also be a candidate key. Both STU_NUM and STU_SSN are candidate keys because each can uniquely identify a student without any extraneous attributes.

5. Primary Key Selection

Among all candidate keys, one is selected as the **primary key**. The primary key is the main key used by the database to uniquely identify records. The choice of primary key may depend on factors like simplicity, performance, or convention.

Example:

- If both STU_NUM and STU_SSN are candidate keys, the database designer might choose STU_NUM as the primary key because it is shorter and likely easier to work with than a Social Security number.

A Candidate Key can be any column or a combination of columns that can qualify as unique key in database. There can be multiple Candidate Keys in one table. Each Candidate Key can qualify as Primary Key.

Example:

- Passport Number, APU TP Number, Identification Number, Email Address — everyone have a unique number. These 4 can potential be Primary Key but at the end you still have to choose one to be become Primary Key. Any column that is Unique and can be a Primary Key is Candidate Key.

Summary

- **Candidate Key:** Minimal set of attributes that can uniquely identify each row in a table.
- **Superkey:** Set of attributes that can uniquely identify rows but may include additional, unnecessary attributes.
- **Primary Key:** Chosen from the candidate keys to be the main identifier for records.

By understanding these key concepts, you can ensure that your database tables are well-designed, with clear and efficient mechanisms for uniquely identifying records.

3G. How To Select Primary Key

1. Entity Integrity

Entity integrity is a fundamental principle in relational database design that ensures each row in a table is uniquely identifiable. This is crucial for maintaining the accuracy and reliability of data within a database.

- **Requirements for Entity Integrity**:
  - **Uniqueness**: Each row in the table must have a unique value for the primary key. This guarantees that **no two rows are identical in terms of their primary key**.
  - **No Nulls in Primary Key**: The primary key must not contain any null values. A null value signifies the absence of data, which would violate the uniqueness requirement.

- **Example**: In the STUDENT table:
  - If STU_NUM is the primary key, entity integrity ensures that each STU_NUM is unique across all rows. No two students can have the same student number.
  - Additionally, STU_NUM cannot be null. Every student in the table must have a defined student number.

2. What is Null Values

A **null** in a database is the absence of a value in a particular field. It's important to understand that a null is not the same as a blank space, a zero, or any other placeholder. It literally means that the value is unknown, missing, or not applicable.

- **Characteristics of Null**:
  - **Not a Value**: A null does not represent a zero, blank, or any other specific value. It simply indicates that no data is present in the field.
  - **Creation of Null**: A null is created when you move to the next field without entering any data. For example, if you press the Tab key or Enter key without typing anything, a null is created in that field.

- **Example**: Consider a STUDENT table where the middle initial (STU_INIT) of a student is optional:
  - If a student does not have a middle name, the STU_INIT field for that student might be null.
  - Null values are permissible in this field because it is not part of the primary key.

3. Why Nulls Can Be Problematic

Null values pose various challenges in relational databases, primarily because they introduce ambiguity and complicate the interpretation of data.

Problems Associated with Nulls:

- **Ambiguity:** A null can mean different things, such as:
  - The value is unknown.
  - The value is missing but known.
  - The attribute is not applicable condition to the record.

- **Impact on Aggregation Functions:** Nulls can create issues in aggregation operations like COUNT, SUM, or AVERAGE. For example:
  - If you have a column of numbers with some nulls, applying SUM might ignore the nulls, leading to potentially misleading results.

- **Complications in Joins:** When tables are linked (joined) on columns that may contain nulls, the presence of nulls can lead to unexpected or incomplete results.

Example:

- In a student database, if STU_PHONE contains nulls for students who haven't provided a phone number, counting the total number of students with COUNT(STU_PHONE) might give you a lower count than expected because it would ignore the nulls.

4. Entity Integrity and Nulls: Best Practices

While nulls can be unavoidable in certain situations, best practices suggest minimizing their use whenever possible:

- **Avoid Nulls in Primary Keys:** Since a primary key must uniquely identify each row and nulls introduce uncertainty, never allow nulls in any part of the primary key.

- **Minimize Use of Nulls:** Where possible, design tables to minimize the need for nulls by using default values or additional attributes that clarify the absence of data.

- **Clear Documentation:** If nulls must be used, ensure they are well-documented so that the meaning of a null is clear to anyone using the database.

Summary

- **Entity Integrity:** Ensures that every row in a table is uniquely identifiable and that primary keys are unique and non-null.

- **Null Values:** Represent the absence of data and should be used sparingly due to the complications they introduce in data interpretation and manipulation.

- **Best Practices:** Avoid nulls in primary keys, minimize their use in other fields, and provide clear documentation when they are necessary.

By understanding these concepts, you can design relational databases that maintain data integrity while minimizing the complexities associated with null values.

3H. Foreign Key (FK)

1. Foreign Key (FK)

Definition:

- A foreign key is a primary key from one table that is placed into another table to establish a relationship between the two tables. This key creates a common attribute between the tables, which allows for controlled redundancy and links them together.

Example:

- In the **PRODUCT** table (Figure 3.2), VEND_CODE is a foreign key. It references the VEND_CODE primary key in the **VENDOR** table. This means that every product listed in the **PRODUCT** table is associated with a specific vendor from the **VENDOR** table.

- In the **STUDENT** table (Figure 3.1), DEPT_CODE and PROF_NUM are foreign keys. These likely reference the DEPT_CODE primary key in a **DEPARTMENT** table and the PROF_NUM primary key in a **PROFESSOR** table, respectively.

Figure 3.2 An Example of a Simple Relational Database

Table name: PRODUCT **Database name: Ch03_SaleCo**
Primary key: PROD_CODE
Foreign key: VEND_CODE

PROD_CODE	PROD_DESCRIPT	PROD_PRICE	PROD_ON_HAND	VEND_CODE
001278-AB	Claw hammer	12.95	23	232
123-21UUY	Houselite chain saw, 16-in. bar	189.99	4	235
QER-34256	Sledge hammer, 16-lb. head	18.63	6	231
SRE-657UG	Rat-tail file	2.99	15	232
ZZX/3245Q	Steel tape, 12-ft. length	6.79	8	235

link

Table name: VENDOR
Primary key: VEND_CODE
Foreign key: none

VEND_CODE	VEND_CONTACT	VEND_AREACODE	VEND_PHONE
230	Shelly K. Smithson	608	555-1234
231	James Johnson	615	123-4536
232	Annelise Crystall	608	224-2134
233	Candice Wallace	904	342-6567
234	Arthur Jones	615	123-3324
235	Henry Ortozo	615	899-3425

STUDENT TABLE

Table name: STUDENT **Database name: Ch03_TinyCollege**

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT	STU_DOB	STU_HRS	STU_CLASS	STU_GPA	STU_TRANSFER	DEPT_CODE	STU_PHONE	PROF_NUM
321452	Bowser	William	C	12-Feb-2000	42	So	2.84	No	BIOL	2134	205
324257	Smithson	Anne	K	15-Nov-2001	81	Jr	3.27	Yes	CIS	2256	222
324258	Brewer	Juliette		23-Aug-1999	36	So	2.26	Yes	ACCT	2256	228
324269	Oblonski	Walter	H	16-Sep-1996	66	Jr	3.09	No	CIS	2114	222
324273	Smith	John	D	30-Dec-1988	102	Sr	2.11	Yes	ENGL	2231	199
324274	Katinga	Raphael	P	21-Oct-2000	114	Sr	3.15	No	ACCT	2267	228
324291	Robertson	Gerald	T	08-Apr-1999	120	Sr	3.87	No	EDU	2267	311
324299	Smith	John	B	30-Nov-2000	15	Fr	2.92	No	ACCT	2315	230

STU_NUM = Student number
STU_LNAME = Student last name
STU_FNAME = Student first name
STU_INIT = Student middle initial
STU_DOB = Student date of birth
STU_HRS = Credit hours earned
STU_CLASS = Student classification
STU_GPA = Grade point average
STU_TRANSFER = Student transferred from another institution
DEPT_CODE = Department code
STU_PHONE = 4-digit campus phone extension
PROF_NUM = Number of the professor who is the student's advisor

2. Referential Integrity

Definition:

- Referential integrity ensures that a foreign key value in a dependent table must either be null or match a primary key value in the related table. This ensures that the relationship between tables remains consistent and that every reference to another entity instance is valid.

Example:

- In the **PRODUCT** table, every VEND_CODE must either be null or match a VEND_CODE in the **VENDOR** table. This ensures that each product has a valid vendor associated with it.

- In the **STUDENT** table, the PROF_NUM must correspond to an existing professor in the **PROFESSOR** table, ensuring that each student is advised by a valid professor.

3. Secondary Key

Definition:

- A secondary key is used for data retrieval purposes but does not enforce a unique value like a primary key does. It helps in searching and querying the database more efficiently when the primary key is not known or practical to use. **Secondary Key also is that Candidate Key not being to choose as Primary Key.**

Example:

- Imagine a **CUSTOMER** table where the primary key is CUS_NUM (Customer Number). Most customers might not know their customer number, but they can be easily searched using their CUS_LNAME (last name) and CUS_PHONE (phone number). Here, CUS_LNAME and CUS_PHONE together act as a secondary key.

- If you wanted to find a customer based on their city, you could use CUS_CITY as a secondary key, although it might not narrow down the results significantly in large cities.

4. Further Example: Consider a Library Database

- **Tables**:
  - **BOOK**: Contains details of books with BOOK_ID as the primary key.
  - **AUTHOR**: Contains details of authors with AUTH_ID as the primary key.
  - **BOOK_AUTHOR**: Links books and authors with BOOK_ID and AUTH_ID as foreign keys.

- **Foreign Key Example**:
  - In the **BOOK_AUTHOR** table, BOOK_ID is a foreign key referencing the primary key in the **BOOK** table, and AUTH_ID is a foreign key referencing the primary key in the **AUTHOR** table.

- **Referential Integrity**:
  - The **BOOK_AUTHOR** table must ensure that each BOOK_ID corresponds to a valid book in the **BOOK** table and each AUTH_ID corresponds to a valid author in the **AUTHOR** table.

- **Secondary Key Example**:
  - To find books by an author's last name, you might use a secondary key composed of the author's last name (AUTH_LNAME) in the **AUTHOR** table, which could help retrieve all books written by authors with that last name.

These keys help maintain the structure, integrity, and efficiency of relational databases, ensuring that data is both organized and retrievable in meaningful ways.

Integrity Rules

1. Integrity Rules Overview

Integrity rules are crucial to maintaining the accuracy and consistency of data in a relational database. These rules ensure that the data adheres to certain conditions, making the database reliable and trustworthy.

The two primary integrity rules in relational databases are **Entity Integrity (PK)** and **Referential Integrity (FK)**. Let's explore each in detail:

Table name: CUSTOMER **Database name: Ch03_InsureCo**
Primary key: CUS_CODE
Foreign key: AGENT_CODE

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_RENEW_DATE	AGENT_CODE
10010	Ramas	Alfred	A	05-Apr-2024	502
10011	Dunne	Leona	K	16-Jun-2024	501
10012	Smith	Kathy	W	29-Jan-2025	502
10013	Olowski	Paul	F	14-Oct-2024	
10014	Orlando	Myron		28-Dec-2024	501
10015	O'Brian	Amy	B	22-Sep-2024	503
10016	Brown	James	G	25-Mar-2025	502
10017	Williams	George		17-Jul-2024	503
10018	Farriss	Anne	G	03-Dec-2024	501
10019	Smith	Olette	K	14-Mar-2025	503

Table name: AGENT (only five selected fields are shown)
Primary key: AGENT_CODE
Foreign key: none

AGENT_CODE	AGENT_AREACODE	AGENT_PHONE	AGENT_LNAME	AGENT_YTD_SLS
501	713	228-1249	Alby	132735.75
502	615	882-1244	Hahn	138967.35
503	615	123-5589	Okon	127093.45

1A. Entity Integrity

Requirement:

- **Rule:** All primary key entries must be **unique**, and **no** part of a primary key may be **null**.

Purpose: Each row will have a known, unique identity, and foreign key values can properly reference primary key values.

- **Explanation:** The primary key uniquely identifies each row in a table. By ensuring that primary keys are unique and never null, the database guarantees that every record can be uniquely distinguished. This uniqueness allows other tables to reference specific rows reliably using foreign keys.

Example: No invoice can have a duplicate number, nor can it be null; in short, all invoices are uniquely identified by their invoice number.

- **Illustration:** Consider an Invoices table where the InvoiceNumber is the primary key. Each invoice must have a unique InvoiceNumber. For example, you cannot have two invoices with the same number, nor can an invoice exist without a number. This ensures that every invoice is uniquely identifiable.

1B. Referential Integrity

Requirement:

- A foreign key may have either a null entry, as long as it is not a part of its table's primary key, or an entry that matches the primary key value in a table to which it is related (every non-null foreign key value must reference an existing primary key value).

- **Rule:** A foreign key must either have a null entry or match an existing primary key value in the table it relates to.

Purpose:

- The purpose is to ensure that every reference by a foreign key is a valid reference to the related primary key. It is possible for an attribute not to have a corresponding value, but it will be impossible to have an invalid entry; the enforcement of the referential integrity rule makes it impossible to delete a row in one table whose primary key has mandatory matching foreign key values in another table.

- **Explanation:** Referential integrity ensures that relationships between tables remain consistent. A foreign key in one table must reference an existing primary key in another table or be null. This rule prevents invalid entries that could lead to orphaned records or data inconsistencies.

Example:

- A customer might not yet have an assigned sales representative (number), but it will be impossible to have an invalid sales representative (number).

- **Illustration:** Suppose you have a Customers table and a SalesRepresentatives table. The Customers table has a foreign key SalesRepNumber that references the primary key RepNumber in the SalesRepresentatives table. If a customer does not yet have an assigned sales representative, SalesRepNumber may be null. However, if SalesRepNumber is filled, it must match an existing RepNumber in the SalesRepresentatives table. This prevents the scenario where a customer references a nonexistent sales representative.

1C. Features of Figure 3.3

Figure 3.3 An Illustration of Integrity Rules

Table name: CUSTOMER **Database name: Ch03_InsureCo**
Primary key: CUS_CODE
Foreign key: AGENT_CODE

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_RENEW_DATE	AGENT_CODE
10010	Ramas	Alfred	A	05-Apr-2024	502
10011	Dunne	Leona	K	16-Jun-2024	501
10012	Smith	Kathy	W	29-Jan-2025	502
10013	Olowski	Paul	F	14-Oct-2024	
10014	Orlando	Myron		28-Dec-2024	501
10015	O'Brian	Amy	B	22-Sep-2024	503
10016	Brown	James	G	25-Mar-2025	502
10017	Williams	George		17-Jul-2024	503
10018	Farriss	Anne	G	03-Dec-2024	501
10019	Smith	Olette	K	14-Mar-2025	503

Table name: AGENT (only five selected fields are shown)
Primary key: AGENT_CODE
Foreign key: none

AGENT_CODE	AGENT_AREACODE	AGENT_PHONE	AGENT_LNAME	AGENT_YTD_SLS
501	713	228-1249	Alby	132735.75
502	615	882-1244	Hahn	138967.35
503	615	123-5589	Okon	127093.45

Database integrity rules using two tables: CUSTOMER and AGENT. Here's a detailed explanation of the key concepts, using the example from the image.

1. Entity Integrity

Entity integrity ensures that each table in the database has a unique primary key, which means that no two rows can have the same primary key value, and no primary key value can be NULL.

- **In the CUSTOMER table:**
  - The primary key is CUS_CODE. This column must contain unique values, and none of its entries can be NULL. In the provided table, values like 10010, 10011, etc., are examples of unique, non-null primary keys.

- **In the AGENT table:**
  - The primary key is AGENT_CODE. Similar to the CUSTOMER table, this column must also contain unique and non-null values.

2. Referential Integrity

Referential integrity ensures that a foreign key in one table must either be NULL or match an existing primary key value in another table.

- **In the CUSTOMER table:**
  - The AGENT_CODE column is a foreign key, meaning it refers to the AGENT_CODE column in the AGENT table.
  - For example, in the CUSTOMER table, the row with CUS_CODE = 10012 has AGENT_CODE = 502. This means that customer Kathy Smith is associated with the agent whose AGENT_CODE is 502, and there is indeed an agent with this code in the AGENT table.
  - However, for the row with CUS_CODE = 10013 (Paul F. Olowski), the AGENT_CODE is NULL, indicating that no agent has been assigned yet.

3. Use of Flags to Avoid NULLs

Sometimes, instead of leaving a field as NULL, a special code (flag) is used to represent a missing or unassigned value.

- **Example with AGENT_CODE:**
  - Instead of leaving AGENT_CODE as NULL for Paul F. Olowski, a flag value like -99 could be used. This code would indicate that no agent has been assigned yet.
  - If this approach is used, the AGENT table would also have a corresponding row with AGENT_CODE = -99 and dummy values for other fields (like area code, phone number, etc.).
- **Example Entry for AGENT_CODE = -99:**
  - AGENT_CODE: -99
  - AGENT_AREACODE: 000
  - AGENT_PHONE: 000-0000
  - AGENT_LNAME: None
  - AGENT_YTD_SLS: $0.0

4. Other Integrity Rules

NOT NULL Constraint:

- This ensures that a column must have a value. For example, if the CUS_LNAME column in the CUSTOMER table has a NOT NULL constraint, every row must have a value in this column.

UNIQUE Constraint:

- This ensures that all values in a column are distinct. For instance, if AGENT_PHONE in the AGENT table had a UNIQUE constraint, no two agents could have the same phone number.

5. Further Example

Suppose we add another customer:

- CUS_CODE: 10020
- CUS_LNAME: Doe
- CUS_FNAME: Jane
- CUS_INITIAL: J
- CUS_RENEW_DATE: 01-Apr-2025
- AGENT_CODE: -99

This new customer would use the flag -99 for AGENT_CODE, indicating that Jane Doe does not yet have an assigned agent. The AGENT table would already contain the corresponding row for AGENT_CODE = -99, as described above.

Conclusion

The integrity rules—entity integrity, referential integrity, and the use of constraints like NOT NULL and UNIQUE—are crucial for maintaining the accuracy and consistency of data in relational databases. They prevent anomalies such as duplicate records, missing foreign key references, and null values where they should not exist.

The Data Dictionary and the System Catalog

The **Data Dictionary** and the **System Catalog** are crucial components of a database management system (DBMS). They serve as repositories of information that help in managing the database's structure, ensuring data integrity, and facilitating user interaction with the database.

1. Data Dictionary

The **Data Dictionary** is a centralized repository that holds detailed information (metadata) about the database's structure and the objects it contains. It's often referred to as "the database designer's database" because it keeps track of the design decisions made during the database's creation. The data dictionary includes the following:

1A. Key Features of a Data Dictionary:

- **Tables Information:** The dictionary provides a comprehensive listing of all the tables in the database, including their names and the schemas they belong to.
- **Attributes and Their Characteristics:** For each table, the data dictionary contains detailed information about the attributes (columns) within the table. This includes:
  - Attribute names
  - Data types (e.g., INTEGER, VARCHAR, DATE)
  - Sizes (e.g., VARCHAR(50))
  - Default values
  - Nullability (whether the attribute can contain NULL values)
  - Any constraints applied (e.g., PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL)
- **Relationships:** The dictionary also records relationships between tables, such as foreign key constraints that enforce referential integrity.
- **Indexes:** Information about any indexes that exist for faster data retrieval.

1A. Key Features of a Data Dictionary:

- **Views and Synonyms:** The dictionary may also contain information about views (virtual tables created by queries) and synonyms (alternate names for database objects).

- **Users and Permissions:** Details about who has access to what data, including user roles and permissions.

- **Triggers and Stored Procedures:** Information about any triggers (automatic actions that occur in response to certain events on a table) and stored procedures (predefined SQL procedures).

1B. Example of a Data Dictionary Entry:

Let's consider a simple example of a data dictionary entry for a table named Employee:

Attribute	Data Type	Size	Nullable	Default Value	Constraint
Employee_ID	INTEGER	N/A	NO	N/A	PRIMARY KEY, UNIQUE
First_Name	VARCHAR	50	NO	'John'	
Last_Name	VARCHAR	50	NO	'Doe'	
DOB	DATE	N/A	YES	NULL	
Department_ID	INTEGER	N/A	NO	N/A	FOREIGN KEY REFERENCES Department(Department_ID)

This entry describes the Employee table, specifying details about each attribute, such as its data type, size, whether it can be NULL, default values, and any constraints.

2. System Catalog

The **System Catalog** is closely related to the data dictionary but typically includes additional information that is vital to the functioning of the DBMS itself. It can be thought of as the DBMS's own metadata repository, containing more detailed and lower-level information than the data dictionary.

2A. Key Features of the System Catalog:

- **Storage Information:** The system catalog includes details about how data is physically stored in the database, such as information about tablespaces, files, and pages.
- **Performance Statistics:** The system catalog may store performance-related data, such as statistics about table access frequencies, index usage, and query performance.
- **Security Information:** Details about security policies, roles, privileges, and audit logs are often stored in the system catalog.
- **Execution Plans:** The catalog may store information about query execution plans that the DBMS uses to optimize query performance.
- **System Configuration:** The catalog holds configuration parameters that control the behavior of the DBMS, such as memory usage, buffer sizes, and logging options.

2B. Example of System Catalog Information:

Consider a system catalog that tracks index usage:

Table Name	Index Name	Index Type	Access Frequency	Last Accessed
Employee	Employee_ID_Index	B-Tree	1000 times/day	2024-08-28 10:45:00
Department	Dept_ID_Index	Hash	500 times/day	2024-08-28 11:00:00

This entry shows how often the Employee_ID_Index on the Employee table is accessed, the type of index, and the last time it was accessed.

3. Relationship Between Data Dictionary and System Catalog

While both the data dictionary and the system catalog store metadata, they serve different purposes:

- The **Data Dictionary** is more focused on the user and designer perspective, providing a high-level overview of the database's structure and design.

- The **System Catalog** is more technical and is used internally by the DBMS to manage and optimize the database's operation.

Conclusion

The Data Dictionary and System Catalog are integral to the management and operation of a database system. They provide the necessary metadata and operational information that support the database's structure, performance, and security. Understanding these components is essential for database administrators, designers, and anyone involved in database management.

4. A Sample Data Dictionary

Table 3.6 A Sample Data Dictionary

Table Name	Attribute Name	Contents	Type	Format	Range	Required	PK OR FK	FK Referenced Table
CUSTOMER	CUS_CODE	Customer account code	CHAR(5)	99999	10000–99999	Y	PK	
	CUS_LNAME	Customer last name	VARCHAR(20)	Xxxxxxxx		Y		
	CUS_FNAME	Customer first name	VARCHAR(20)	Xxxxxxxx		Y		
	CUS_INITIAL	Customer initial	CHAR(1)	X				
	CUS_RENEW_DATE	Customer insurance renewal date	DATE	dd-mmm-yyyy				
	AGENT_CODE	Agent code	CHAR(3)	999			FK	AGENT
AGENT	AGENT_CODE	Agent code	CHAR(3)	999		Y	PK	
	AGENT_AREACODE	Agent area code	CHAR(3)	999		Y		
	AGENT_PHONE	Agent telephone number	CHAR(8)	999–9999		Y		
	AGENT_LNAME	Agent last name	VARCHAR(20)	Xxxxxxxx		Y		
	AGENT_YTD_SLS	Agent year-to-date sales	NUMBER(9,2)	9,999,999.99				

FK	= Foreign key
PK	= Primary key
CHAR	= Fixed character length data (1- 255 characters)
VARCHAR	= Variable character length data (1- 2,000 characters)
NUMBER	= Numeric data. NUMBER (9,2) is used to specify numbers with up to nine digits, including two digits to the right of the decimal place. Some RDBMSs use NUMERIC or DECIMAL instead of NUMBER, and others permit the use of a MONEY or CURRENCY data type.

The image represents a sample data dictionary, which is an essential component in database management. Here's a detailed explanation of the key concepts, using the example in the image.

4A. Key Elements of a Data Dictionary

- **Table Name**: The name of the table in the database (e.g., CUSTOMER, AGENT).
- **Attribute Name**: The name of each column within the table (e.g., CUS_CODE, CUS_LNAME).
- **Contents**: A brief description of what the attribute represents (e.g., Customer account code).
- **Type**: The data type for each attribute, specifying how the data is stored (e.g., CHAR, VARCHAR, NUMBER).
- **Format**: The specific format for the data, if applicable (e.g., 99999 for a 5-digit number).
- **Range**: The permissible range of values that the attribute can hold (e.g., 10000–99999).
- **Required**: Indicates whether the attribute must have a value (Y for Yes).
- **PK or FK**: Specifies whether the attribute is a Primary Key (PK) or Foreign Key (FK).
- **FK Referenced Table**: If the attribute is a foreign key, this column shows the table it references.

4B. System Catalog

The **system catalog** is an extended form of the data dictionary that includes additional metadata for internal use by the DBMS. This might include:

- Relationship types.
- Indexes and their components.
- Entity and referential integrity checks.
- Information on authorized users, access privileges, table creation dates, etc.

4C. Homonyms and Synonyms

- **Homonyms**: In a database, a homonym refers to using the same name for different attributes in different tables, which should be avoided to reduce confusion. For example, using C_NAME for both a customer name and a consultant name in different tables.

- **Synonyms**: Synonyms occur when different names are used to describe the same attribute. This should also be avoided. For example, using CAR in one table and AUTO in another when they refer to the same concept.

4D. Further Example

- Suppose we have another table, CONSULTANT, and you want to record the consultant's name. Using C_NAME for the consultant's name would create a homonym issue if C_NAME is also used in the CUSTOMER table for the customer's name. To avoid this, you should name the attribute something like CONS_NAME to distinguish it from C_NAME.

- Similarly, if CAR is used in one table and AUTO in another, both referring to the same object, it's better to standardize the attribute name across tables to avoid synonyms and maintain consistency.

Conclusion

The data dictionary and system catalog are vital tools in managing and maintaining database integrity and consistency. They ensure that all database design and implementation team members are aligned on the structure and constraints of the data. By clearly defining attribute types, constraints, and relationships, the data dictionary helps prevent errors and confusion during database development and maintenance.

Quick Review Question

- What is a candidate key?
- What is a primary key?
- What is a foreign key?
- What is entity integrity?
- What is referential integrity?
- What is data dictionary?

Summary of Main Teaching Points

- Tables are basic building blocks of a relational database
- Keys are central to the use of relational tables
- Keys define functional dependencies
  - Candidate key
  - Primary key
  - Secondary key
  - Foreign key