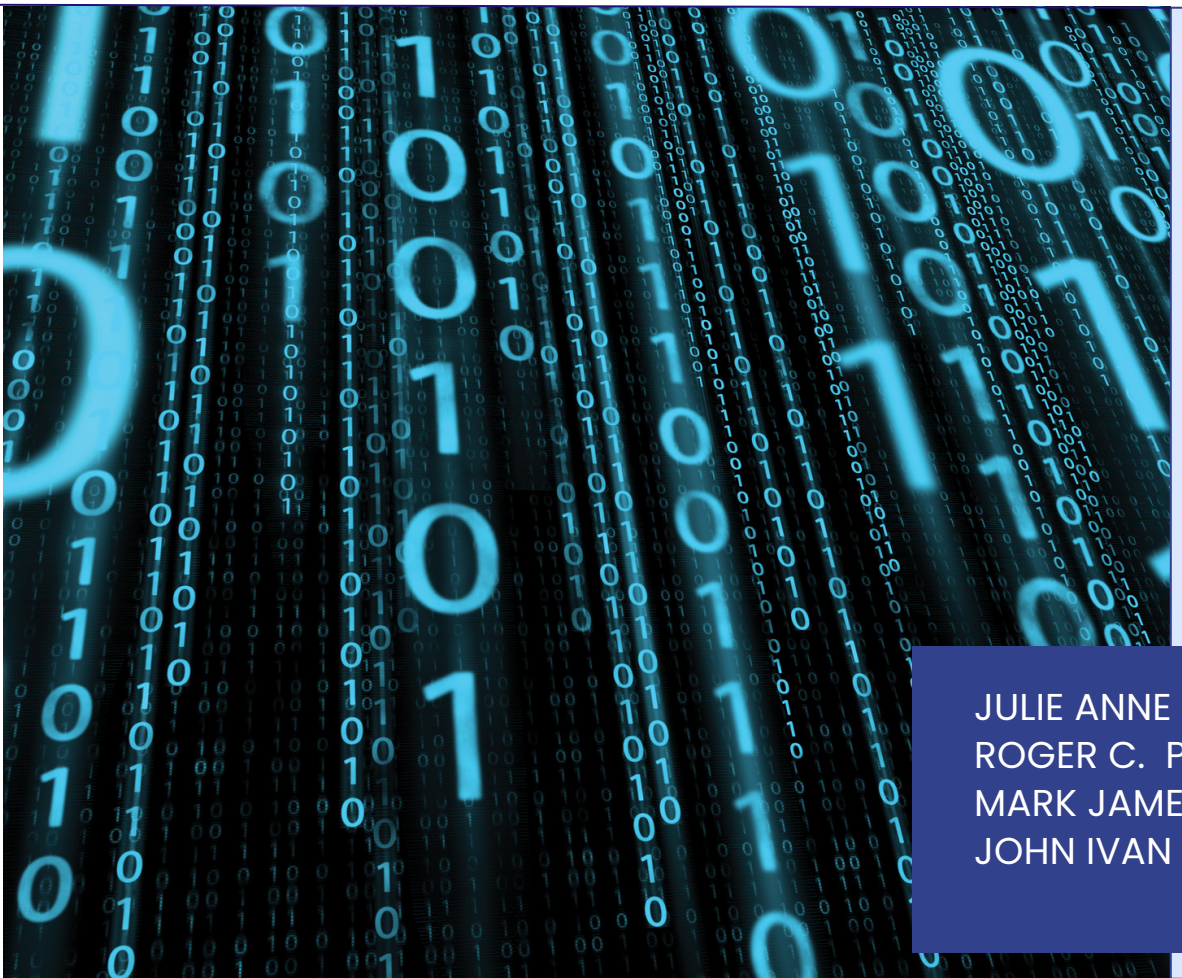
NATIONAL UNIVERSITY
COLLEGE OF COMPUTING AND
INFORMATION TECHNOLOGIES

VER 1.0

Course MANUAL

INFORMATION MANAGEMENT

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



INFORMATION TECHNOLOGY

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

INFORMATION MANAGEMENT

Table of Contents

MODULE

#06

NATIONAL UNIVERSITY

MODULE GOALS

- Database Tables and Normalization
- The Need for Normalization
- The Normalization Process
- Improving the Design
- Surrogate Key Considerations
- Higher-Level Normal Forms
- Normalization and Database Design
- Denormalization
- Data Modelling Checklist

FLEX Course Material

Normalization of Database Tables

WEEK 10 MODULE

Education that works.

LESSON #

#6.1

Database Tables and Normalization

Normalization is the process to eliminate data redundancy and enhance data integrity in the table.

Normalization also helps to organize the data in the database.

It is a multi-step process that sets the data into tabular form and removes the duplicated data from the relational tables.

Normalization

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies.

Normalization rules divides larger tables into smaller tables and links them using relationships. The purpose of Normalization in SQL is to eliminate redundant (repetitive)

The inventor of the relational model Edgar Codd proposed the theory of normalization of data with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form. Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form. data and ensure data is stored logically.

DATABASE NORMAL FORMS

Here is a list of Normal Forms in SQL:

- 1NF (First Normal Form)
- 2NF (Second Normal Form)
- 3NF (Third Normal Form)
- BCNF (Boyce-Codd Normal Form)
- 4NF (Fourth Normal Form)
- 5NF (Fifth Normal Form)
- 6NF (Sixth Normal Form)

LESSON #

#6.2

The Need for Normalization

Normalization helps to reduce redundancy and complexity by examining new data types used in the table.

It is helpful to divide the large database table into smaller tables and link them using relationship.

It avoids duplicate data or no repeating groups into a table.

Normalization

Database **Normalization Example** can be easily understood with the help of a case study.

Assume, a video library maintains a database of movies rented out. Without any normalization in database, all information is stored in one table as shown below.

EXAMPLE:

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean, Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal, Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

Here you see **Movies Rented column has multiple values.**
Now let's move into 1st Normal Forms:

1NF (First Normal Form) Rules

- Each table cell should contain a single value.
- Each record needs to be unique.

1NF Example

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean	Ms.
Janet Jones	First Street Plot No 4	Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal	Mr.
Robert Phil	3rd Street 34	Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

Normalization

Let's move into second normal form 2NF

2NF (Second Normal Form) Rules

- Rule 1- Be in 1NF
- Rule 2- Single Column Primary Key that does not functionally dependant on any subset of candidate key relation
- It is clear that we can't move forward to make our simple database in 2nd Normalization form unless we partition the table above.

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

- We have divided our 1NF table into two tables viz. Table 1 and Table2. Table 1 contains member information. Table 2 contains information on movies rented.
- We have introduced a new column called Membership_id which is the primary key for table 1. Records can be uniquely identified in Table 1 using membership id

Normalization

3NF (Third Normal Form) Rules

- Rule 1- Be in 2NF
- Rule 2- Has no transitive functional dependencies
- To move our 2NF table into 3NF, we again need to again divide our table.

3NF Example

- Below is a 3NF example in SQL database:

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION ID
1	Janet Jones	First Street Plot No 4	2
2	Robert Phil	3rd Street 34	1
3	Robert Phil	5th Avenue	1

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

SALUTATION ID	SALUTATION
1	Mr.
2	Ms.
3	Mrs.
4	Dr.

- We have again divided our tables and created a new table which stores Salutations.
- There are no transitive functional dependencies, and hence our table is in 3NF
- In Table 3 Salutation ID is primary key, and in Table 1 Salutation ID is foreign to primary key in Table 3
- Now our little example is at a level that cannot further be decomposed to attain higher normal form types of normalization in DBMS. In fact, it is already in higher normalization forms. Separate efforts for moving into next levels of normalizing data are normally needed in complex databases.

Normalization

- **BCNF (Boyce-Codd Normal Form)**
- Even when a database is in 3rd Normal Form, still there would be anomalies resulted if it has more than one **Candidate** Key.
- Sometimes is BCNF is also referred as **3.5 Normal Form.**
- **4NF (Fourth Normal Form) Rules**
- If no database table instance contains two or more, independent and multivalued data describing the relevant entity, then it is in 4th Normal Form.
- **5NF (Fifth Normal Form) Rules**
- A table is in 5th Normal Form only if it is in 4NF and it cannot be decomposed into any number of smaller tables without loss of data.
- **6NF (Sixth Normal Form) Proposed**
- 6th Normal Form is not standardized, yet however, it is being discussed by database experts for some time. Hopefully, we would have a clear & standardized definition for 6th Normal Form in the near future…

Advantages and Disadvantages:

- **Advantages of Normalization**
  - Normalization helps to minimize data redundancy.
  - Greater overall database organization.
  - Data consistency within the database.
  - Much more flexible database design.
  - Enforces the concept of relational integrity.

- **Disadvantages of Normalization**
  - You cannot start building the database before knowing what the user needs.
  - The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
  - It is very time-consuming and difficult to normalize relations of a higher degree.
  - Careless decomposition may lead to a bad database design, leading to serious problems.

LESSON #

#6.3

The Normalization Process Improving the Design

The normalization process in database design is a systematic approach to organizing data to reduce redundancy and improve data integrity.

It involves breaking down complex tables into smaller, well-structured tables, reducing data duplication, and eliminating certain types of dependencies.

The primary goal of normalization is to create a database schema that avoids anomalies and ensures data consistency.

There are several normal forms, each building on the previous one, and the process continues until the database meets the desired level of normalization.

Normalization

Let's explore the normalization process, focusing on the typical normal forms (1NF, 2NF, and 3NF):

- **1. First Normal Form (1NF):**
  - **Remove Duplicate Rows:**
    - Ensure that each table has a primary key, and no duplicate rows exist.
  - **Atomic Values:**
    - Ensure that each attribute contains atomic (indivisible) values.
- **2. Second Normal Form (2NF):**
  - **Full Functional Dependency:**
    - Eliminate partial dependencies by ensuring that non-prime attributes are fully functionally dependent on the primary key.
  - **Separate Tables:**
    - Create separate tables for entities with composite primary keys.
- **3. Third Normal Form (3NF):**
  - **Transitive Dependency:**
    - Eliminate transitive dependencies by ensuring that non-prime attributes are not dependent on other non-prime attributes.
  - **Remove Calculated Data:**
    - Avoid storing calculated data that can be derived from other attributes.

Normalization

- **Benefits of Normalization:**
- **Data Integrity:**
    - Reduces data redundancy, minimizing the risk of inconsistencies.
- **Flexibility and Adaptability:**
    - Allows for easier modifications and additions to the database structure.
- **Efficient Data Retrieval:**
    - Simplifies queries and improves data retrieval efficiency.
- **Consistent Updates:**
    - Updates to the database are more straightforward and less prone to errors.

- **Common Normalization Anomalies:**
- **Insertion Anomaly:**
    - Difficulty in inserting data due to incomplete information.
- **Update Anomaly:**
    - Inconsistencies arise when updating data, leading to discrepancies.
- **Deletion Anomaly:**
    - Unintended loss of data when deleting certain information.

Exercise:

Consider the following unnormalized table, which stores information about students and the courses they are enrolled in:

Unnormalized Table: StudentCourses

StudentID	StudentName	CourseID	CourseName	Instructor	Credits
1	Alice	101	Math 101	Prof. Smith	4
1	Alice	102	English 101	Prof. Johnson	3
2	Bob	101	Math 101	Prof. Smith	4
3	Charlie	102	English 101	Prof. Johnson	3
3	Charlie	103	Science 101	Prof. Davis	4

This table exhibits redundancy and violates the principles of normalization. Let's normalize it step by step.

- **Step 1: First Normal Form (1NF)**
- Ensure that each column has atomic (indivisible) values.

Normalized Table: StudentCourses_1NF

StudentID	StudentName	CourseID	CourseName	Instructor	Credits
1	Alice	101	Math 101	Prof. Smith	4
1	Alice	102	English 101	Prof. Johnson	3
2	Bob	101	Math 101	Prof. Smith	4
3	Charlie	102	English 101	Prof. Johnson	3
3	Charlie	103	Science 101	Prof. Davis	4

Exercise:

- **Step 2: Second Normal Form (2NF)**
- Ensure that all non-key attributes are fully functionally dependent on the primary key.
- **Normalized Tables: Students and Courses**
- **Students**

StudentID	StudentName
1	Alice
2	Bob
3	Charlie

- **Courses**

CourseID	CourseName	Instructor	Credits
101	Math 101	Prof. Smith	4
102	English 101	Prof. Johnson	3
103	Science 101	Prof. Davis	4

- **Enrollment**

StudentID	CourseID
1	101
1	102
2	101
3	102
3	103

Exercise:

- **Step 3: Third Normal Form (3NF)**
- Ensure that there are no transitive dependencies.
- **Normalized Tables: Students, Courses, and Instructors**
- **Instructors**

Instructor	CourseID
Prof. Smith	101
Prof. Johnson	102
Prof. Davis	103

Now, the database is in 3NF, and redundancy has been minimized. Each table has a clear purpose, and relationships are established through foreign keys. This design helps maintain data integrity and supports efficient data retrieval and updates.

Exercise:

- Let's consider another unnormalized table, and we'll go through the normalization process step by step.
- **Unnormalized Table: EmployeeSkills**

EmployeeID	EmployeeName	SkillID	SkillName	SkillCategory	Department
1	John	101	Programming	IT	Development
1	John	102	Database	IT	Development
2	Alice	101	Programming	IT	Testing
3	Bob	102	Database	HR	HR
3	Bob	103	Communication	HR	HR
4	Carol	101	Programming	IT	Development

- **Step 1: First Normal Form (1NF)**
- Ensure that each column has atomic (indivisible) values.
- **Normalized Table: EmployeeSkills_1NF**

EmployeeID	EmployeeName	SkillID	SkillName	SkillCategory	Department
1	John	101	Programming	IT	Development
1	John	102	Database	IT	Development
2	Alice	101	Programming	IT	Testing
3	Bob	102	Database	HR	HR
3	Bob	103	Communication	HR	HR
4	Carol	101	Programming	IT	Development

*** Do the 2NF and 3NF**

Exercise:

- Suppose we have a table that stores information about customers and the products they have purchased:
- **Unnormalized Table: CustomerProducts**

CustomerID	CustomerName	ProductID	ProductName	Category	Price
1	Alice	101	Laptop	Electronics	1200
1	Alice	102	Smartphone	Electronics	800
2	Bob	101	Laptop	Electronics	1200
3	Charlie	103	Headphones	Electronics	150
3	Charlie	104	Coffee Maker	Appliances	80

*** Do the 1NF, 2NF and 3NF**

- **Step 1: First Normal Form (1NF)**
  - Ensure that each column has atomic (indivisible) values.
- **Step 2: Second Normal Form (2NF)**
  - Ensure that all non-key attributes are fully functionally dependent on the primary key.
- **Step 3: Third Normal Form (3NF)**
  - Ensure that there are no transitive dependencies.

LESSON #

#6.4

Surrogate Key Considerations

A surrogate key is defined as a unique identifier for some record or object in a table.

It is similar to a primary key, but with a significant difference: it is not derived from the table data – the object generates this key itself.

Next, the surrogate key does not have any business value or semantic meaning – it only serves for data analysis.

It is not part of the application, and it is invisible to the user.

SURROGATE KEY

Briefly, we can define the following surrogate key characteristics:

- It holds a unique value for all records
- It is generated automatically
- It can't be modified by the user or the application
- It can be used only in the CRUD (Create, Read, Update, Delete) operations

The most common format for the surrogate key is a sequential number that identifies a table row.

But why is the surrogate key used in SQL – is the primary key not enough?

There are cases (and it happens frequently) when it is impossible to use a unique primary key on the table. A simple example is a table with duplicate values in different columns.

For instance, we have a table containing information about books written by two authors who wrote some novels solo and co-authored other books.

Author	Title	Year	ISBN	Publisher
Douglas Preston, Lincoln Child	Mount Dragon	1996	0-7653-5996-0	Tor Books
Douglas Preston, Lincoln Child	Thunderhead	1999	0-446-60837-8	Grand Central Publishing
Douglas Preston	Tyrannosaur Canyon	2005	0-7653-1104-6	Forge Books
Douglas Preston	Impact	2010	978-0-7653-1768-1	Forge Books

SURROGATE KEY

To identify a row uniquely, we need a key based on the unique combination of the novel's title and the writers' names (it can have one author or two co-authors). The surrogate key will be the solution.

Surrogate keys also help us resolve more issues, such as the following ones:

- To apply primary keys to data collected from various sources
- To maintain primary keys for a longer time
- To preserve the primary key value under different conditions

The surrogate key in SQL acts as the primary key when the primary key is not applicable. Ideally, every table row should have both the primary and surrogate keys: the primary key identifies a row in the database, and the surrogate key identifies a separate entity.

Surrogate key vs. natural key

You can use both the natural and surrogate keys in databases. To apply them correctly, we'll define them and understand how they differ.

As we already know, a surrogate key is an automatically generated unique identifier that does not have any business or contextual meaning. In the below table, you can see a separate column containing these identifiers:

	Author	Title	Year	ISBN	Publisher
1	Douglas Preston, Lincoln Child	Mount Dragon	1996	0-7653-5996-0	Tor Books
2	Douglas Preston	Tyrannosaur Canyon	2005	0-7653-1104-6	Forge Books
3	Douglas Preston	Impact	2010	978-0-7653-1768-1	Forge Books

SURROGATE KEY

- A natural key is a key that has a contextual or business meaning. It relates to one or several columns already existing in the particular table.
- One more difference between a surrogate key and a natural key is that a natural key needs to be created manually, whether partially or completely.
- Every published book has an International Standard Book Number (ISBN) – a unique identifier of the specific publication. It is a prime example of a natural key.

Author	Title	Year	ISBN	Publisher
Douglas Preston, Lincoln Child	Mount Dragon	1996	0-7653-5996-0	Tor Books
Douglas Preston	Tyrannosaur Canyon	2005	0-7653-1104-6	Forge Books
Douglas Preston	Impact	2010	978-0-7653-1768-1	Forge Books

- It is worth noticing more differences between the surrogate and natural keys:

Surrogate key	**Natural key**
Can't be used as a search key	Can be used as a search key
Is always unique	Can include duplicate values
Applies uniform rules to all records	Does not apply uniform rules for each record
Does not change with time	Can be changed according to requirements
Requires an additional column in a table	Does not require an additional column (already exists)

Some business scenarios require you to combine natural and surrogate keys. For instance, you can use the surrogate key as a primary key, while the natural key will serve as a foreign key. It won't affect user experience.

The pros and cons of the surrogate key

- Database specialists like surrogate keys for many reasons. Still, they aren't a panacea, and you need to consider both pros and cons to see whether you need them in your particular case.

The advantages of using surrogate keys

- Uniqueness. We have mentioned this parameter several times, but it is worth stressing that a surrogate key is always unique. Thus, if you migrate data, you can be sure there won't be any duplicate keys – the system will generate new, unique keys for your inserted data.
- Unlimited number. As the most common SQL surrogate key formats are sequential numbers of random strings, the system can generate any number of such unique values and combinations.
- Consistency. It is another crucial factor we stressed earlier. Natural keys can change because of changes in data or business requirements, but surrogate keys never change. It is always the same reference to the particular table row. Thus, if you use the surrogate key as a primary key, you can be sure it will be stable.
- Less code. As the surrogate key is always the same, you can reuse it whenever needed. Besides, the surrogate key consists most frequently of integers, which makes it more compact. When used in an SQL JOIN clause, the surrogate key makes the query less expensive.
- Better performance. Surrogate keys are smaller than natural keys. The key consistency allows easier table integration and handling. Queries are faster, and operations require less disk IO – the usage of surrogate keys is a decent query optimization technique.

The disadvantages of using surrogate keys

- A surrogate key requires an extra column in a table. Therefore, the system consumes extra disk space and IO to store and handle surrogate keys. You will also need an additional index for the surrogate key column.
- Surrogate keys violate normalization (the third normal form) as the surrogate key value has no relation to the actual table data.
- For the same reason, surrogate keys require more table JOINs in a query.
- Surrogate keys may complicate the distinction between the test and production data. As surrogate keys are values generated automatically without meaningful references, one may not tell test data from actual business data easily.
- Finally, surrogate keys require a specific implementation technique for every particular DBMS.
- Many work scenarios suggest using surrogate keys. However, every case is different. Even experienced developers should consider and evaluate all options to pick the most suitable one.
- However, it won't demand in-depth SQL knowledge to implement a surrogate key.

SURROGATE KEY

Example of using a surrogate key in a SQL database. In this example, we'll create two tables: one with a natural key and another with a surrogate key.

Natural Key Example:

```
-- Create a table with a natural key
CREATE TABLE Employees_Natural (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Department VARCHAR(50)
);

-- Insert some data
INSERT INTO Employees_Natural (EmployeeID, FirstName, LastName, Department)
VALUES
    (1, 'Alice', 'Smith', 'IT'),
    (2, 'Bob', 'Johnson', 'HR'),
    (3, 'Charlie', 'Davis', 'Marketing');
```

In this example, the EmployeeID is a natural key, and it uniquely identifies each employee. However, in some scenarios, natural keys might have drawbacks, such as changes in the employee name or department, which could require updates to multiple records.

Surrogate Key Example:

```
-- Create a table with a surrogate key
CREATE TABLE Employees_Surrogate (
    EmployeeID INT PRIMARY KEY,
    EmployeeCode INT UNIQUE,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Department VARCHAR(50)
);

-- Insert some data with auto-incrementing surrogate key
INSERT INTO Employees_Surrogate (EmployeeCode, FirstName, LastName, Department)
VALUES
    (101, 'Alice', 'Smith', 'IT'),
    (102, 'Bob', 'Johnson', 'HR'),
    (103, 'Charlie', 'Davis', 'Marketing');
```

In this example, we introduce a surrogate key (EmployeeID) with an auto-incrementing feature. This key is not based on any business meaning; it's just a system-generated identifier.

SURROGATE KEY

Benefits of Surrogate Key:

1. **Stability:** Even if the employee's name or department changes, the surrogate key remains stable.
2. **Data Independence:** The surrogate key is not dependent on business-related information, providing better data independence.
3. **Simplifies Joins:** Joining tables based on the surrogate key is often simpler and more efficient.
4. **Indexing:** Surrogate keys are often used for indexing, which can enhance query performance.
5. **Security:** Auto-incrementing integers can be more secure than using business-related information.

It's important to note that the decision to use a surrogate key or a natural key depends on the specific requirements of the system and the characteristics of the data. In some cases, a combination of both approaches might be appropriate.

LESSON #

#6.5

Higher-Level Normal Forms

Beyond the commonly discussed normal forms like 1NF, 2NF, and 3NF, there are higher-level normal forms designed to address more complex data modeling scenarios.

Two additional normal forms often considered are Boyce-Codd Normal Form (BCNF) and Fourth Normal Form (4NF).

Normalization

1. Boyce-Codd Normal Form (BCNF):

- **Definition:**
    - A table is in BCNF if, for every non-trivial functional dependency $X \rightarrow Y$, X is a superkey.
- **Key Points:**
    - BCNF is an extension of 3NF that addresses certain anomalies involving non-prime attributes.
    - It deals specifically with tables where there are multiple candidate keys.
    - A table is automatically in BCNF if it contains only one candidate key.

- **Example:** Consider a table with attributes {StudentID, CourseID, Instructor} and functional dependencies {StudentID, CourseID} → Instructor and {CourseID} → Instructor. In this case, {StudentID, CourseID} is a superkey, and the table is in BCNF.

2. Fourth Normal Form (4NF):

- **Definition:**
    - A table is in 4NF if, for every non-trivial multivalued dependency $X \twoheadrightarrow Y$, X is a superkey.
- **Key Points:**
    - 4NF addresses situations where a table has multiple independent multi-valued dependencies.
    - It is a refinement beyond BCNF to handle cases where there are several independent sets of multi-valued dependencies.
- **Example:** Consider a table with attributes {EmployeeID, ProjectID, Task, HoursWorked} and the multivalued dependency {EmployeeID, ProjectID} \twoheadrightarrow \{Task, HoursWorked\}. In this case, {EmployeeID, ProjectID} is a superkey, and the table is in 4NF.

Considerations:

- **Higher Normal Forms vs. Practicality:**
    - Achieving higher normal forms may lead to more complex database designs. It's crucial to balance the benefits of normalization against the practical aspects of query performance and system complexity.
- **Performance Implications:**
    - Higher normal forms may result in additional tables, joins, and complexities that could impact query performance. Pragmatic decisions may be necessary based on the specific use cases and performance requirements.
- **Normalization Trade-offs:**
    - Database designers often need to make trade-offs between achieving higher normal forms and the practicality of implementation. Striking the right balance depends on the nature of the data and the application's requirements.
- It's essential to carefully consider the specific characteristics of the data and the requirements of the system when deciding to normalize beyond 3NF. While higher normal forms provide additional assurances against certain types of anomalies, they also introduce complexities that need to be managed. The normalization process should align with the goals of data integrity, consistency, and system performance.

LESSON #

#6.6

Normalization and Database Design

Normalization is a crucial concept in database design that involves organizing data in a systematic way to reduce redundancy and dependency.

The normalization process aims to create a well-structured and efficient relational database that minimizes data anomalies and ensures data integrity.

Normalization

- **Benefits of Normalization in Database Design:**

1. Data Integrity:

- Reduces data redundancy and ensures that data is consistent and accurate.

2. Consistency:

- Helps maintain consistency across the database by avoiding anomalies.

3. Efficient Storage:

- Optimizes storage space by eliminating duplicate data.

4. Ease of Updates:

- Simplifies data updates and reduces the likelihood of update anomalies.

5. Simplified Queries:

- Facilitates simpler and more straightforward SQL queries.

6. Flexibility:

- Allows for easier modification and expansion of the database structure.

7. Data Independence:

- Reduces dependencies between tables, promoting data independence.

Normalization

Considerations in Database Design:

1.Nature of the Data:

- The nature of the data and the relationships between entities influence the normalization process. For some datasets, achieving higher normal forms might be more beneficial, while for others, 3NF might be sufficient.

2.Application Requirements:

- The requirements of the application or system using the database influence the design decisions. Performance considerations, ease of maintenance, and the complexity of queries should be taken into account.

3.Trade-offs:

- Designers often need to make trade-offs between achieving higher normal forms and the practicality of implementation. Striking the right balance depends on the specific use cases and performance requirements.

4.Practicality vs. Purity:

- While normalization principles provide a guide, practical considerations may lead to deviations from strict normalization rules. The goal is to achieve a balance between purity and practicality.

In summary, normalization is a crucial process in database design that helps ensure data integrity, reduce redundancy, and optimize the efficiency of data storage and retrieval. Designers should carefully consider the specific characteristics of the data and the requirements of the system when applying normalization principles.

LESSON #

#6.7

Denormalization

Denormalization is a database design technique where one or more normal forms are deliberately violated in order to achieve specific goals related to performance, simplicity, or ease of use.

While normalization focuses on reducing redundancy and ensuring data consistency, denormalization intentionally introduces redundancy to meet certain business or performance requirements.

Denormalization

Goals of Denormalization:

1.Improved Query Performance:

- Denormalization can lead to simplified and faster query execution, especially for complex joins or aggregations, as it reduces the need for multiple table lookups.

2.Simplified Query Logic:

- By combining data from multiple tables into a single table, query logic can be simplified, making it easier for developers and analysts to work with the database.

3.Enhanced Read Performance:

- For read-heavy applications, denormalization can lead to improved read performance, as data retrieval becomes more straightforward.

4.Reduced Joins:

- Denormalization can reduce the number of joins required for certain queries, which can be particularly beneficial in systems where join operations are resource-intensive.

5.Caching and Materialized Views:

- Denormalized structures can be used to create caching mechanisms or materialized views, precomputing and storing results for frequently executed queries.

6.Simplified Application Code:

- Denormalization can simplify the application code by reducing the need for complex joins and allowing for more straightforward data retrieval.

Denormalization

Techniques of Denormalization:

1.Flattening Tables:

- Combining related tables into a single table to reduce the need for joins. For example, merging a customer table and an order table into a single denormalized table.

2.Redundant Storage:

- Storing redundant copies of data in different tables to eliminate the need for certain joins. This can be useful for scenarios where data retrieval speed is critical.

3.Aggregated Data:

- Precomputing and storing aggregated data (sums, averages, etc.) to avoid performing these calculations during query execution.

4.Materialized Views:

- Creating materialized views that store the results of specific queries, reducing the need to recompute the results each time the query is executed.

Denormalization

Techniques of Denormalization:

1.Flattening Tables:

- Combining related tables into a single table to reduce the need for joins. For example, merging a customer table and an order table into a single denormalized table.

2.Redundant Storage:

- Storing redundant copies of data in different tables to eliminate the need for certain joins. This can be useful for scenarios where data retrieval speed is critical.

3.Aggregated Data:

- Precomputing and storing aggregated data (sums, averages, etc.) to avoid performing these calculations during query execution.

4.Materialized Views:

- Creating materialized views that store the results of specific queries, reducing the need to recompute the results each time the query is executed.

Denormalization

A simple example to illustrate denormalization. Suppose we have a normalized database with two tables: Customers and Orders. The normalized structure is designed to minimize redundancy:

Normalized Tables:

1. **Customers Table:**

CustomerID	CustomerName	Email
1	Alice	alice@email.com
2	Bob	bob@email.com
3	Charlie	charlie@email.com

2. **Orders Table:**

OrderID	CustomerID	OrderDate	TotalAmount
101	1	2023-01-15	120.50
102	2	2023-01-16	75.20
103	3	2023-01-16	200.00

In this normalized structure, CustomerID in the Orders table is a foreign key referencing the Customers table.

Now, let's consider a denormalized structure where we combine information from both tables into a single table:

Denormalized Table:

- **Denormalized Orders Table:**

OrderID	CustomerName	Email	OrderDate	TotalAmount
101	Alice	alice@email.com	2023-01-15	120.50
102	Bob	bob@email.com	2023-01-16	75.20
103	Charlie	charlie@email.com	2023-01-16	200.00

In this denormalized structure, we have combined information from both the Customers and Orders tables into a single table. The CustomerName and Email columns, originally in the Customers table, are duplicated for each order in the denormalized table. This duplication simplifies queries related to orders but comes at the cost of redundancy.

LESSON #

#6.8

Data Modelling Checklist

A data modeling checklist is a comprehensive set of considerations and best practices that guide the process of designing and implementing a database

Data Modelling Checklist

- Below is a checklist that covers key aspects of data modeling. Note that the specifics may vary based on the database management system (DBMS) and the requirements of the particular project.

Preliminary Considerations:

1. Understand Requirements:

- Clearly understand the business requirements and objectives for the database.

2. Define Scope:

- Clearly define the scope of the database, including the entities, relationships, and functionalities it will support.

3. Identify Stakeholders:

- Identify and involve key stakeholders (e.g., business users, developers, administrators) in the data modeling process.

Conceptual Data Modeling:

4. Identify Entities:

- Identify the main entities that need to be represented in the database.

5. Define Relationships:

- Define relationships between entities, including cardinality and participation constraints.

6. Refine Attributes:

- Identify and refine attributes for each entity, ensuring they are atomic and well-defined.

7. Normalization:

- Consider normalization to reduce redundancy and dependency.

Data Modelling Checklist

Logical Data Modeling:

8.Choose Data Types:

- Choose appropriate data types for each attribute based on the nature of the data.

9.Define Primary Keys:

- Define primary keys for each entity.

10.Establish Foreign Keys:

- Establish foreign key relationships between related tables.

11.Indexing:

- Identify fields that should be indexed to optimize query performance.

12.Check Constraints:

- Implement check constraints to enforce data integrity.

Physical Data Modeling:

13.Partitioning:

- Consider partitioning strategies for large tables to enhance performance.

14.Define Storage Parameters:

- Define storage parameters such as tablespaces, filegroups, and file placement.

15.Consideration for Clusters:

- Evaluate the need for clustering tables based on usage patterns.

16.Optimize Joins:

- Optimize tables and indexes to minimize the cost of joins.

17.Denormalization:

- Evaluate the need for denormalization for performance gains.

Documentation:

18.Entity-Relationship Diagram (ERD):

- Create an ERD to visually represent entities, relationships, and attributes.

19.Data Dictionary:

- Develop a data dictionary documenting definitions, relationships, and constraints.

Testing and Validation:

20.Data Integrity Checks:

- Implement data integrity checks and validate them through testing.

21.Performance Testing:

- Conduct performance testing to ensure the database meets performance requirements.

Security and Access Control:

22.Access Control:

- Implement access control mechanisms to restrict unauthorized access.

23.Encryption:

- Consider encryption for sensitive data and communication.

Maintenance and Scalability:

24.Backup and Recovery:

- Establish a robust backup and recovery strategy.

25.Data Archiving:

- Consider data archiving strategies to manage historical data.

26.Scalability Planning:

- Plan for scalability by considering future growth and changes.

- **Collaboration and Documentation:**

27.Collaboration:

- Encourage collaboration between database designers, developers, and other stakeholders.

28.Documentation:

- Maintain up-to-date documentation reflecting changes and updates.

- **Review and Quality Assurance:**

29.Peer Review:

- Conduct peer reviews of the data model to identify potential issues.

30.Quality Assurance:

- Implement a quality assurance process to ensure adherence to standards.

This checklist provides a broad overview of considerations for data modeling. Depending on the specific project, industry, and compliance requirements, additional considerations may be necessary. Regularly revisiting and updating the checklist based on lessons learned from project experiences is also beneficial.