Database Normalization

-----------------------------------------------------------------

3. Normalization

What is normalization?

"Normalization is the process of organizing the columns and tables of a relational database to minimize data redundancy and improve data integrity.

//Normalization is a systematic process of organizing the columns and tables of a relational database to minimize data redundancy and improve data integrity. Its main goal is to break down a large table into smaller, well-structured tables and define relationships between them, ensuring that data is stored logically and efficiently.

নরমালাইজেশন হলো রিলেশনাল ডেটাবেসের কলাম এবং টেবিলগুলিকে সংগঠিত করার একটি পদ্ধতিগত প্রক্রিয়া, যার প্রধান উদ্দেশ্য হল ডেটা রিডানডেন্সি (redundancy) কমানো এবং ডেটা ইন্টিগ্রিটি (integrity) উন্নত করা। এর মূল লক্ষ্য হলো একটি বড় টেবিলকে ছোট, সুসংগঠিত টেবিলগুলিতে বিভক্ত করা এবং তাদের মধ্যে সম্পর্ক স্থাপন করা, যাতে ডেটা যুক্তিসঙ্গতভাবে এবং দক্ষতার সাথে সংরক্ষিত হয়।

Why is normalization important?

Normalization is crucial for several reasons that contribute to a healthy and efficient database:

নরমালাইজেশন বেশ কয়েকটি কারণে গুরুত্বপূর্ণ যা একটি সুস্থ ও দক্ষ ডেটাবেসের জন্য অপরিহার্য:

Reduces Data Redundancy: By eliminating duplicate data, it saves storage space and prevents inconsistencies that can arise when the same data is stored in multiple places.

ডেটা রিডানডেন্সি হ্রাস করে: নকল ডেটা দূর করার মাধ্যমে এটি স্টোরেজ স্পেস বাঁচায় এবং অসঙ্গতি প্রতিরোধ করে, যা একই ডেটা একাধিক স্থানে সংরক্ষিত থাকলে ঘটতে পারে।

Improves Data Integrity: Ensures that data is consistent and accurate across the database. When data changes, it only needs to be updated in one place.

ডেটা ইন্টিগ্রিটি উন্নত করে: ডেটাবেস জুড়ে ডেটা সামঞ্জস্যপূর্ণ এবং সঠিক তা নিশ্চিত করে। যখন ডেটা পরিবর্তিত হয়, তখন এটি কেবল একটি স্থানে আপডেট করার প্রয়োজন হয়।

Enhances Data Consistency: Prevents update, insertion, and deletion anomalies, which are problems that occur when data is not properly organized.

ডেটা সামঞ্জস্যতা বাড়ায়: আপডেট, ইনসার্শন এবং ডিলিশন অ্যানোমালি (anomalies) প্রতিরোধ করে, যা ডেটা সঠিকভাবে সংগঠিত না হলে ঘটে।

Simplifies Queries: While it might seem counterintuitive due to more joins, a normalized schema often makes it easier to write clear and specific queries because each piece of information has a designated, logical home.

কোয়েরি সহজ করে: যদিও এটি আরও বেশি জয়ের (joins) কারণে অদ্ভুত মনে হতে পারে, একটি নরমালাইজড স্কিমা প্রায়শই পরিষ্কার এবং নির্দিষ্ট কোয়েরি লেখা সহজ করে তোলে কারণ প্রতিটি তথ্যের একটি নির্দিষ্ট, যৌক্তিক স্থান থাকে।

Better Database Design: Promotes a cleaner, more logical, and more scalable database structure.

উন্নত ডেটাবেস ডিজাইন: একটি পরিষ্কার, আরও যৌক্তিক এবং আরও স্কেলযোগ্য ডেটাবেস কাঠামোকে উৎসাহিত করে।

Real-life Example (বাস্তব জীবনের উদাহরণ):

Imagine a school database. If you store student details (name, address, phone) directly in a Courses table for every course a student takes, you'll repeat the student's name and address multiple times. If the student moves, you'd have to update their address in many places, risking errors. Normalization would put student details in a separate Students table, and the Courses table would just have a StudentID to link to it.

একটি স্কুলের ডেটাবেস কল্পনা করুন। যদি আপনি একজন শিক্ষার্থীর প্রতিটি কোর্সের জন্য শিক্ষার্থীর বিবরণ (নাম, ঠিকানা, ফোন) সরাসরি একটি Courses টেবিলে সংরক্ষণ করেন, তাহলে শিক্ষার্থীর নাম এবং ঠিকানা একাধিকবার পুনরাবৃত্তি হবে। যদি শিক্ষার্থী স্থান পরিবর্তন করে, তাহলে আপনাকে তাদের ঠিকানা একাধিক স্থানে আপডেট করতে হবে, যার ফলে ত্রুটির ঝুঁকি থাকে। নরমালাইজেশন শিক্ষার্থীর বিবরণ একটি পৃথক Students টেবিলে রাখবে, এবং Courses টেবিলে কেবল একটি StudentID থাকবে যা এর সাথে লিঙ্ক করবে।

Types of normal forms (1NF, 2NF, 3NF, BCNF)?

Normalization is defined through a series of "normal forms." Each normal form builds upon the previous one, adding stricter rules to reduce redundancy and improve data integrity. The most commonly discussed normal forms are:

নরমালাইজেশন "নরমাল ফর্ম" এর একটি সিরিজের মাধ্যমে সংজ্ঞায়িত হয়। প্রতিটি নরমাল ফর্ম পূর্ববর্তী ফর্মের উপর ভিত্তি করে তৈরি হয়, যা রিডানডেন্সি কমাতে এবং ডেটা ইন্টিগ্রিটি উন্নত করতে কঠোর নিয়ম যোগ করে। সবচেয়ে বেশি আলোচিত নরমাল ফর্মগুলি হলো:

First Normal Form (1NF)

Second Normal Form (2NF)

Third Normal Form (3NF)

Boyce-Codd Normal Form (BCNF) (also known as 3.5NF, a stricter version of 3NF)

There are higher normal forms (4NF, 5NF, etc.), but 3NF and BCNF are generally considered sufficient for most business applications.

উচ্চতর নরমাল ফর্ম (4NF, 5NF, ইত্যাদি) আছে, তবে 3NF এবং BCNF সাধারণত বেশিরভাগ ব্যবসায়িক অ্যাপ্লিকেশনের জন্য যথেষ্ট বলে বিবেচিত হয়।

1NF with example?

A table is in First Normal Form (1NF) if it meets the following criteria:

একটি টেবিল প্রথম নরমাল ফর্ম (1NF) এ থাকে যদি এটি নিম্নলিখিত মানদণ্ডগুলি পূরণ করে:

Atomic Values: Each column must contain only atomic (indivisible) values. This means no repeating groups or multiple values in a single cell.

অ্যাটমিক ভ্যালু: প্রতিটি কলামে শুধুমাত্র অ্যাটমিক (অবিভাজ্য) মান থাকতে হবে। এর অর্থ একটি একক সেলে পুনরাবৃত্ত গ্রুপ বা একাধিক মান থাকবে না।

Unique Rows: Each row must be unique, meaning there's a primary key to uniquely identify each record.

অদ্বিতীয় সারি: প্রতিটি সারি অদ্বিতীয় হতে হবে, যার অর্থ প্রতিটি রেকর্ডকে অদ্বিতীয়ভাবে সনাক্ত করার জন্য একটি প্রাইমারি কী থাকতে হবে।

Single Column per Attribute: Each column should represent a single attribute and not contain multiple pieces of information.

প্রতি অ্যাট্রিবিউটে একক কলাম: প্রতিটি কলাম একটি একক অ্যাট্রিবিউটকে প্রতিনিধিত্ব করবে এবং একাধিক তথ্য ধারণ করবে না।

Example (উদাহরণ):

Consider a table storing student enrollment details in a non-1NF form:

একটি স্টুডেন্ট এনরোলমেন্টের বিবরণ ধারণকারী টেবিল যা 1NF ফর্মে নেই:

Not in 1NF:Student\_Enrollment Table:

| StudentID | StudentName | Course | Instructor |

| :---------- | :------------ | :----------------- | :----------------- |

| 101 | Alice | Math, Physics | Mr. A, Dr. B |

| 102 | Bob | Chemistry | Dr. C |

Explanation of why it's not in 1NF (কেন এটি 1NF এ নেই তার ব্যাখ্যা):

The Course and Instructor columns contain multiple values (e.g., "Math, Physics" in one cell). This violates the atomic values rule.

Course এবং Instructor কলামগুলিতে একাধিক মান রয়েছে (যেমন, একটি সেলে "Math, Physics")। এটি অ্যাটমিক ভ্যালু নিয়ম লঙ্ঘন করে।

Converting to 1NF: To achieve 1NF, we need to separate the repeating groups into individual rows, ensuring each cell has a single, atomic value.

1NF এ রূপান্তর: 1NF অর্জন করতে, আমাদের পুনরাবৃত্ত গ্রুপগুলিকে পৃথক সারিতে বিভক্ত করতে হবে, প্রতিটি সেলে একটি একক, অ্যাটমিক মান নিশ্চিত করতে হবে।

Student\_Enrollment\_1NF Table:

| StudentID | StudentName | Course | Instructor |

| :---------- | :------------ | :-------- | :----------- |

| 101 | Alice | Math | Mr. A |

| 101 | Alice | Physics | Dr. B |

| 102 | Bob | Chemistry | Dr. C |

Explanation (ব্যাখ্যা):

Now, each column contains atomic values. To uniquely identify each row, (StudentID, Course) would form a composite primary key.

এখন, প্রতিটি কলামে অ্যাটমিক মান রয়েছে। প্রতিটি সারিকে অদ্বিতীয়ভাবে সনাক্ত করতে, (StudentID, Course) একটি কম্পোজিট প্রাইমারি কী তৈরি করবে।

2NF with example?

A table is in Second Normal Form (2NF) if it meets the following criteria:

একটি টেবিল দ্বিতীয় নরমাল ফর্ম (2NF) এ থাকে যদি এটি নিম্নলিখিত মানদণ্ডগুলি পূরণ করে:

It is in 1NF.

এটি 1NF এ আছে।

No Partial Dependency: All non-key attributes must be fully functionally dependent on the entire primary key. This means no non-key attribute can be dependent on only a part of a composite primary key.

কোনো আংশিক নির্ভরতা নেই (No Partial Dependency): সমস্ত নন-কী অ্যাট্রিবিউটকে সম্পূর্ণ প্রাইমারি কী-এর উপর সম্পূর্ণরূপে ফাংশনালি নির্ভরশীল হতে হবে। এর অর্থ হলো কোনো নন-কী অ্যাট্রিবিউট একটি কম্পোজিট প্রাইমারি কী-এর শুধুমাত্র একটি অংশের উপর নির্ভরশীল হতে পারবে না।

Example (উদাহরণ):

Consider the Student\_Enrollment\_1NF table from the previous example, which is in 1NF:

পূর্ববর্তী উদাহরণ থেকে Student\_Enrollment\_1NF টেবিলটি বিবেচনা করুন, যা 1NF এ রয়েছে:

Student\_Enrollment\_1NF Table:

| StudentID (Part of PK) | Course (Part of PK) | StudentName | Instructor |

| :----------------------- | :-------------------- | :------------ | :----------- |

| 101 | Math | Alice | Mr. A |

| 101 | Physics | Alice | Dr. B |

| 102 | Chemistry | Bob | Dr. C |

Composite Primary Key: (StudentID, Course)

Explanation of why it's not in 2NF (কেন এটি 2NF এ নেই তার ব্যাখ্যা):

StudentName depends only on StudentID (a part of the composite primary key), not on the full key (StudentID, Course). This is a partial dependency. If Alice's name changes, it should change only based on her StudentID, not on which course record we are looking at.

Instructor depends only on Course (another part of the composite primary key), not on the full key. This is also a partial dependency.

StudentName শুধুমাত্র StudentID (কম্পোজিট প্রাইমারি কী-এর একটি অংশ) এর উপর নির্ভর করে, সম্পূর্ণ কী (StudentID, Course) এর উপর নয়। এটি একটি আংশিক নির্ভরতা (partial dependency)। যদি এলিসের নাম পরিবর্তিত হয়, তাহলে এটি শুধুমাত্র তার StudentID এর উপর ভিত্তি করে পরিবর্তিত হওয়া উচিত, আমরা কোন কোর্স রেকর্ড দেখছি তার উপর নয়।

Instructor শুধুমাত্র Course (কম্পোজিট প্রাইমারি কী-এর আরেকটি অংশ) এর উপর নির্ভর করে, সম্পূর্ণ কী এর উপর নয়। এটিও একটি আংশিক নির্ভরতা (partial dependency)।

Converting to 2NF: To achieve 2NF, we need to remove partial dependencies by creating new tables for the dependent attributes.

2NF এ রূপান্তর: 2NF অর্জন করতে, আমাদের নির্ভরশীল অ্যাট্রিবিউটগুলির জন্য নতুন টেবিল তৈরি করে আংশিক নির্ভরতা দূর করতে হবে।

Students Table:

| StudentID (PK) | StudentName |

| :--------------- | :------------ |

| 101 | Alice |

| 102 | Bob |

Courses Table:

| Course (PK) | Instructor |

| :------------ | :----------- |

| Math | Mr. A |

| Physics | Dr. B |

| Chemistry | Dr. C |

Enrollments Table (Junction Table):

| StudentID (PK, FK) | Course (PK, FK) |

| :------------------- | :---------------- |

| 101 | Math |

| 101 | Physics |

| 102 | Chemistry |

Explanation (ব্যাখ্যা):

Now, StudentName is fully dependent on StudentID in the Students table, and Instructor is fully dependent on Course in the Courses table. The Enrollments table links students to courses using their respective primary keys, with its composite primary key (StudentID, Course) having no partial dependencies.

এখন, StudentName Students টেবিলের StudentID এর উপর সম্পূর্ণরূপে নির্ভরশীল, এবং Instructor Courses টেবিলের Course এর উপর সম্পূর্ণরূপে নির্ভরশীল। Enrollments টেবিল তাদের নিজ নিজ প্রাইমারি কী ব্যবহার করে ছাত্রছাত্রীদের কোর্সের সাথে সংযুক্ত করে, যার কম্পোজিট প্রাইমারি কী (StudentID, Course) এর কোনো আংশিক নির্ভরতা নেই।

3NF with example?

A table is in Third Normal Form (3NF) if it meets the following criteria:

একটি টেবিল তৃতীয় নরমাল ফর্ম (3NF) এ থাকে যদি এটি নিম্নলিখিত মানদণ্ডগুলি পূরণ করে:

It is in 2NF.

এটি 2NF এ আছে।

No Transitive Dependency: There are no transitive dependencies. A transitive dependency occurs when a non-key attribute is dependent on another non-key attribute. (i.e., A -> B and B -> C, then A -> C is a transitive dependency, and C should be moved to a separate table dependent on B).

কোনো ট্রানজিটিভ নির্ভরতা নেই (No Transitive Dependency): কোনো ট্রানজিটিভ নির্ভরতা থাকবে না। একটি ট্রানজিটিভ নির্ভরতা তখন ঘটে যখন একটি নন-কী অ্যাট্রিবিউট অন্য একটি নন-কী অ্যাট্রিবিউটের উপর নির্ভরশীল হয়। (যেমন, A -> B এবং B -> C হলে, A -> C একটি ট্রানজিটিভ নির্ভরতা, এবং C কে একটি পৃথক টেবিলে স্থানান্তরিত করা উচিত যা B এর উপর নির্ভরশীল।)

Example (উদাহরণ):

Consider a Books table:

একটি Books টেবিল বিবেচনা করুন:

Not in 3NF:Books Table:

| BookID (PK) | BookName | AuthorID | AuthorName | AuthorBio |

| :------------ | :------------ | :--------- | :----------- | :---------- |

| B001 | The Great Book| A001 | John Doe | Award Winner|

| B002 | New Adventures| A001 | John Doe | Award Winner|

| B003 | Space Odyssey | A002 | Jane Smith | Sci-Fi Fan |

Explanation of why it's not in 3NF (কেন এটি 3NF এ নেই তার ব্যাখ্যা):

BookID is the primary key.

AuthorName and AuthorBio are non-key attributes.

We observe a transitive dependency: BookID determines AuthorID, and AuthorID determines AuthorName and AuthorBio. So, BookID indirectly determines AuthorName and AuthorBio through AuthorID. This means AuthorName and AuthorBio are dependent on a non-key attribute (AuthorID), which violates 3NF.

BookID হল প্রাইমারি কী।

AuthorName এবং AuthorBio হল নন-কী অ্যাট্রিবিউট।

আমরা একটি ট্রানজিটিভ নির্ভরতা (transitive dependency) লক্ষ্য করি: BookID AuthorID কে নির্ধারণ করে, এবং AuthorID AuthorName এবং AuthorBio কে নির্ধারণ করে। সুতরাং, BookID পরোক্ষভাবে AuthorID এর মাধ্যমে AuthorName এবং AuthorBio কে নির্ধারণ করে। এর অর্থ হলো AuthorName এবং AuthorBio একটি নন-কী অ্যাট্রিবিউট (AuthorID) এর উপর নির্ভরশীল, যা 3NF লঙ্ঘন করে।

Converting to 3NF: To achieve 3NF, we need to remove transitive dependencies by creating new tables for the transitively dependent attributes.

3NF এ রূপান্তর: 3NF অর্জন করতে, আমাদের ট্রানজিটিভভাবে নির্ভরশীল অ্যাট্রিবিউটগুলির জন্য নতুন টেবিল তৈরি করে ট্রানজিটিভ নির্ভরতা দূর করতে হবে।

Books\_3NF Table:

| BookID (PK) | BookName | AuthorID (FK) |

| :------------ | :------------ | :-------------- |

| B001 | The Great Book| A001 |

| B002 | New Adventures| A001 |

| B003 | Space Odyssey | A002 |

Authors Table:

| AuthorID (PK) | AuthorName | AuthorBio |

| :-------------- | :----------- | :---------- |

| A001 | John Doe | Award Winner|

| A002 | Jane Smith | Sci-Fi Fan |

Explanation (ব্যাখ্যা):

Now, AuthorName and AuthorBio are directly dependent on AuthorID in the Authors table, and AuthorID is a foreign key in the Books\_3NF table. There are no transitive dependencies in either table.

এখন, AuthorName এবং AuthorBio Authors টেবিলে AuthorID এর উপর সরাসরি নির্ভরশীল, এবং AuthorID Books\_3NF টেবিলে একটি ফরেন কী। কোনো টেবিলে কোনো ট্রানজিটিভ নির্ভরতা নেই।

What is BCNF?

Boyce-Codd Normal Form (BCNF) is a stricter version of 3NF. A table is in BCNF if and only if:

বয়েস-কড নরমাল ফর্ম (BCNF) হলো 3NF এর একটি কঠোর সংস্করণ। একটি টেবিল BCNF এ থাকে যদি এবং শুধুমাত্র যদি:

It is in 3NF.

এটি 3NF এ আছে।

For every non-trivial functional dependency X -> Y, X must be a super key of the relation. This means that for any dependency, the determinant (the left side of the arrow, X) must be a candidate key (or a super key that contains a candidate key).

প্রতিটি নন-ট্রাইভিয়াল ফাংশনাল ডিপেন্ডেন্সি X -> Y এর জন্য, X অবশ্যই সম্পর্কের একটি সুপার কী (super key) হতে হবে। এর অর্থ হলো যেকোনো নির্ভরতার জন্য, নির্ণায়ক (তীরের বাম দিক, X) অবশ্যই একটি ক্যান্ডিডেট কী (বা একটি সুপার কী যা একটি ক্যান্ডিডেট কী ধারণ করে) হতে হবে।

BCNF addresses a specific type of anomaly that 3NF might miss, especially in tables with overlapping candidate keys. It ensures that every determinant is a candidate key.

BCNF এক নির্দিষ্ট ধরণের অসঙ্গতি সমাধান করে যা 3NF মিস করতে পারে, বিশেষত ওভারল্যাপিং ক্যান্ডিডেট কী সহ টেবিলগুলিতে। এটি নিশ্চিত করে যে প্রতিটি নির্ণায়ক একটি ক্যান্ডিডেট কী।

Example (উদাহরণ):

This is a more complex scenario, often involving multiple candidate keys.

এটি একটি আরও জটিল পরিস্থিতি, যা প্রায়শই একাধিক ক্যান্ডিডেট কী জড়িত থাকে।

Consider a Student\_Course\_Instructor table:

একটি Student\_Course\_Instructor টেবিল বিবেচনা করুন:

Student (C. Key 1) Course (C. Key 2) Instructor

Alice Database Mr. Smith

Alice OS Mr. Jones

Bob Database Mr. Smith

Carol Database Mr. Smith

Export to Sheets

Assumptions (অনুমান):

A student can take multiple courses.

একজন শিক্ষার্থী একাধিক কোর্স নিতে পারে।

An instructor can teach multiple courses.

একজন প্রশিক্ষক একাধিক কোর্স শেখাতে পারে।

A course can have multiple students.

একটি কোর্সে একাধিক শিক্ষার্থী থাকতে পারে।

Crucially: Each Course is taught by only one Instructor. (i.e., Course -> Instructor is a functional dependency).

গুরুত্বপূর্ণ: প্রতিটি Course শুধুমাত্র একজন Instructor দ্বারা শেখানো হয়। (যেমন, Course -> Instructor একটি ফাংশনাল ডিপেন্ডেন্সি)।

The primary key is (Student, Course). This table is in 3NF because there are no partial or transitive dependencies from the primary key.

প্রাইমারি কী হলো (Student, Course)। এই টেবিলটি 3NF এ আছে কারণ প্রাইমারি কী থেকে কোনো আংশিক বা ট্রানজিটিভ নির্ভরতা নেই।

Explanation of why it's not in BCNF (কেন এটি BCNF এ নেই তার ব্যাখ্যা):

The functional dependency Course -> Instructor exists.

ফাংশনাল ডিপেন্ডেন্সি Course -> Instructor বিদ্যমান।

However, Course (the determinant) is not a super key for the entire table. It's only part of the composite primary key (Student, Course). Since Course determines Instructor, but Course itself is not a candidate key for the table, it violates BCNF.

এই ক্ষেত্রে, Course (নির্ণায়ক) সম্পূর্ণ টেবিলের জন্য একটি সুপার কী নয়। এটি শুধুমাত্র কম্পোজিট প্রাইমারি কী (Student, Course) এর অংশ। যেহেতু Course Instructor কে নির্ধারণ করে, কিন্তু Course নিজেই টেবিলের জন্য একটি ক্যান্ডিডেট কী নয়, তাই এটি BCNF লঙ্ঘন করে।

Converting to BCNF: Separate the dependent attributes into new tables so that every determinant is a super key.

BCNF এ রূপান্তর: নির্ভরশীল অ্যাট্রিবিউটগুলিকে নতুন টেবিলে বিভক্ত করুন যাতে প্রতিটি নির্ণায়ক একটি সুপার কী হয়।

Enrollments Table:

| Student (PK, FK) | Course (PK, FK) |

| :----------------- | :---------------- |

| Alice | Database |

| Alice | OS |

| Bob | Database |

| Carol | Database |

Course\_Instructor Table:

| Course (PK) | Instructor |

| :------------ | :----------- |

| Database | Mr. Smith |

| OS | Mr. Jones |

Explanation (ব্যাখ্যা):

Now, in Course\_Instructor table, Course is the primary key (and thus a super key) and determines Instructor. In the Enrollments table, the primary key (Student, Course) is a super key, and there are no other determinants. Both tables are in BCNF.

এখন, Course\_Instructor টেবিলে, Course হল প্রাইমারি কী (এবং তাই একটি সুপার কী) এবং Instructor কে নির্ধারণ করে। Enrollments টেবিলে, প্রাইমারি কী (Student, Course) একটি সুপার কী, এবং অন্য কোনো নির্ণায়ক নেই। উভয় টেবিলই BCNF এ আছে।

What is functional dependency?

A functional dependency X -> Y exists if the value of attribute set X uniquely determines the value of attribute set Y. In simpler terms, if you know X, you can always find out Y. X is called the determinant, and Y is called the dependent.

একটি ফাংশনাল ডিপেন্ডেন্সি X -> Y বিদ্যমান থাকে যদি অ্যাট্রিবিউট সেট X এর মান অ্যাট্রিবিউট সেট Y এর মানকে অদ্বিতীয়ভাবে নির্ধারণ করে। সহজ কথায়, যদি আপনি X জানেন, তাহলে আপনি সর্বদা Y খুঁজে বের করতে পারবেন। X কে নির্ণায়ক (determinant) এবং Y কে নির্ভরশীল (dependent) বলা হয়।

Notation (চিহ্ন): X -> Y

Real-life example (বাস্তব জীবনের উদাহরণ):

In a Students table:

একটি Students টেবিলে:

StudentID StudentName StudentAddress StudentPhone

S001 Alice 123 Main St 555-1234

S002 Bob 456 Oak Ave 555-5678

Export to Sheets

StudentID -> StudentName (If you know the StudentID, you know the StudentName).

StudentID -> StudentName (যদি আপনি StudentID জানেন, তাহলে আপনি StudentName জানেন)।

StudentID -> StudentAddress

StudentID -> StudentAddress

StudentID -> StudentPhone

StudentID -> StudentPhone

In fact, StudentID -> (StudentName, StudentAddress, StudentPhone) because StudentID determines all other attributes in the row.

আসলে, StudentID -> (StudentName, StudentAddress, StudentPhone) কারণ StudentID সারির অন্যান্য সমস্ত অ্যাট্রিবিউট নির্ধারণ করে।

What is transitive dependency?

A transitive dependency occurs in a table when a non-key attribute is indirectly dependent on the primary key through another non-key attribute. In other words, if A, B, and C are attributes in a table such that:

একটি ট্রানজিটিভ ডিপেন্ডেন্সি একটি টেবিলে ঘটে যখন একটি নন-কী অ্যাট্রিবিউট অন্য একটি নন-কী অ্যাট্রিবিউটের মাধ্যমে প্রাইমারি কী এর উপর পরোক্ষভাবে নির্ভরশীল হয়। অন্য কথায়, যদি A, B, এবং C একটি টেবিলের অ্যাট্রিবিউট হয় যেমন:

A -> B (A determines B)

B -> C (B determines C)

B is not part of the primary key, and C is not B or part of B.

Then A -> C is a transitive dependency. This violates 3NF.

A -> B (A, B কে নির্ধারণ করে)

B -> C (B, C কে নির্ধারণ করে)

B প্রাইমারি কী এর অংশ নয়, এবং C B বা B এর অংশ নয়।

তাহলে A -> C একটি ট্রানজিটিভ ডিপেন্ডেন্সি। এটি 3NF লঙ্ঘন করে।

Real-life example (বাস্তব জীবনের উদাহরণ):

From our 3NF example:

আমাদের 3NF উদাহরণ থেকে:

Books Table:

| BookID (PK) | BookName | AuthorID | AuthorName | AuthorBio |

| :------------ | :------------ | :--------- | :----------- | :---------- |

| B001 | The Great Book| A001 | John Doe | Award Winner|

Here:

এখানে:

BookID is the primary key (A).

AuthorID is a non-key attribute (B).

AuthorName and AuthorBio are non-key attributes (C).

BookID হল প্রাইমারি কী (A)।

AuthorID হল একটি নন-কী অ্যাট্রিবিউট (B)।

AuthorName এবং AuthorBio হল নন-কী অ্যাট্রিবিউট (C)।

We have:

আমাদের আছে:

BookID -> AuthorID (A determines B)

AuthorID -> AuthorName and AuthorID -> AuthorBio (B determines C)

So, BookID -> AuthorName and BookID -> AuthorBio are transitive dependencies because AuthorName and AuthorBio are indirectly dependent on BookID through AuthorID.

সুতরাং, BookID -> AuthorName এবং BookID -> AuthorBio হল ট্রানজিটিভ ডিপেন্ডেন্সি কারণ AuthorName এবং AuthorBio AuthorID এর মাধ্যমে BookID এর উপর পরোক্ষভাবে নির্ভরশীল।

What is partial dependency?

A partial dependency occurs when a non-key attribute in a table is functionally dependent on only a part of a composite primary key, rather than on the entire composite primary key. This violates 2NF.

একটি আংশিক নির্ভরতা (partial dependency) একটি টেবিলে তখন ঘটে যখন একটি নন-কী অ্যাট্রিবিউট একটি কম্পোজিট প্রাইমারি কী-এর শুধুমাত্র একটি অংশের উপর ফাংশনালি নির্ভরশীল হয়, পুরো কম্পোজিট প্রাইমারি কী-এর উপর নয়। এটি 2NF লঙ্ঘন করে।

Real-life example (বাস্তব জীবনের উদাহরণ):

From our 2NF example:

আমাদের 2NF উদাহরণ থেকে:

Student\_Enrollment\_1NF Table:

| StudentID (Part of PK) | Course (Part of PK) | StudentName | Instructor |

| :----------------------- | :-------------------- | :------------ | :----------- |

| 101 | Math | Alice | Mr. A |

Composite Primary Key: (StudentID, Course)

Here:

এখানে:

StudentName is a non-key attribute.

StudentName একটি নন-কী অ্যাট্রিবিউট।

StudentName is dependent on StudentID. (StudentID -> StudentName)

StudentName StudentID এর উপর নির্ভরশীল। (StudentID -> StudentName)

StudentID is only a part of the composite primary key (StudentID, Course).

StudentID হল কম্পোজিট প্রাইমারি কী (StudentID, Course) এর শুধুমাত্র একটি অংশ।

This means StudentName has a partial dependency on the primary key, as it only depends on StudentID and not the full (StudentID, Course) combination. Similarly, Instructor has a partial dependency on Course.

এর অর্থ হলো StudentName এর প্রাইমারি কী এর উপর একটি আংশিক নির্ভরতা আছে, কারণ এটি শুধুমাত্র StudentID এর উপর নির্ভর করে এবং সম্পূর্ণ (StudentID, Course) সমন্বয়ের উপর নয়। একইভাবে, Instructor এর Course এর উপর একটি আংশিক নির্ভরতা আছে।

What is denormalization?

Denormalization is the process of intentionally introducing redundancy into a database by combining data from multiple tables into a single table, or by duplicating columns. This is typically done to improve query performance, especially for read-heavy applications or data warehousing, by reducing the number of joins required to retrieve data.

ডিনরমালাইজেশন হলো একাধিক টেবিল থেকে ডেটা একত্রিত করে একটি একক টেবিলে বা কলামগুলি নকল করে একটি ডেটাবেসে ইচ্ছাকৃতভাবে রিডানডেন্সি যুক্ত করার প্রক্রিয়া। এটি সাধারণত কোয়েরি পারফরম্যান্স উন্নত করতে করা হয়, বিশেষত রিড-হেভি অ্যাপ্লিকেশন বা ডেটা ওয়্যারহাউসিংয়ের জন্য, ডেটা পুনরুদ্ধার করতে প্রয়োজনীয় জয়েনগুলির সংখ্যা হ্রাস করে।

Key points (গুরুত্বপূর্ণ বিষয়):

It's a controlled form of redundancy.

এটি রিডানডেন্সির একটি নিয়ন্ত্রিত রূপ।

It sacrifices some data integrity and consistency (potentially) for speed.

এটি গতির জন্য কিছু ডেটা ইন্টিগ্রিটি এবং সামঞ্জস্যতা (সম্ভাব্যভাবে) ত্যাগ করে।

Often used after a database has been fully normalized, when performance bottlenecks are identified.

একটি ডেটাবেস সম্পূর্ণরূপে নরমালাইজড হওয়ার পরে, যখন পারফরম্যান্সের সমস্যাগুলি চিহ্নিত করা হয় তখন এটি প্রায়শই ব্যবহৃত হয়।

Real-life example (বাস্তব জীবনের উদাহরণ):

Imagine an e-commerce website where you frequently display product names and their categories together.

একটি ই-কমার্স ওয়েবসাইট কল্পনা করুন যেখানে আপনি প্রায়শই পণ্যের নাম এবং তাদের ক্যাটাগরি একসাথে প্রদর্শন করেন।

Normalized structure:Products table: (ProductID, ProductName, CategoryID, Price)Categories table: (CategoryID, CategoryName)

To get ProductName and CategoryName, you'd need a JOIN.ProductName এবং CategoryName পেতে, আপনার একটি JOIN এর প্রয়োজন হবে।

Denormalized structure (for a specific report/view):

You might add CategoryName directly into the Products table for faster reporting or display:

আপনি দ্রুত রিপোর্টিং বা প্রদর্শনের জন্য CategoryName সরাসরি Products টেবিলে যোগ করতে পারেন:

Products\_Denormalized table: (ProductID, ProductName, CategoryID, CategoryName, Price)

Now, retrieving product name and category name doesn't require a join, making queries faster. However, if CategoryName changes, you'd need to update it in both the Categories table and the Products\_Denormalized table, introducing redundancy and potential for inconsistency if not managed carefully.

এখন, পণ্যের নাম এবং ক্যাটাগরির নাম পুনরুদ্ধার করতে কোনো জয়ের প্রয়োজন হয় না, যা কোয়েরিগুলিকে দ্রুত করে তোলে। তবে, যদি CategoryName পরিবর্তিত হয়, তাহলে আপনাকে এটি Categories টেবিল এবং Products\_Denormalized টেবিল উভয় স্থানে আপডেট করতে হবে, যা রিডানডেন্সি এবং সম্ভাব্য অসঙ্গতির কারণ হতে পারে যদি সাবধানে পরিচালিত না হয়।

Difference between normalization and denormalization?

Feature Normalization Denormalization

Goal Minimize data redundancy, improve data integrity. Improve query performance by introducing redundancy.

Process Breaking down large tables into smaller, related tables. Combining tables or duplicating data from related tables.

Redundancy Reduces redundancy. Introduces redundancy (controlled).

Data Integrity Enhances data integrity and consistency. Can compromise data integrity (if not managed carefully).

Update/Insert/Delete Anomalies Eliminates anomalies. Can reintroduce anomalies (managed by application logic).

Joins Increases the need for joins in queries. Reduces the need for joins in queries.

Write Performance Generally better due to less data to write/update in fewer places. Can be worse due to more data to write/update in multiple places.

Read Performance Can be slower for complex queries due to more joins. Generally faster for specific read-heavy queries.

Use Cases OLTP (Online Transaction Processing) systems, transactional databases. OLAP (Online Analytical Processing) systems, data warehousing, reporting, read-heavy applications.

Export to Sheets

সাধারণ ভাষায় পার্থক্য:

নরমালাইজেশন: ডেটাকে পরিষ্কার-পরিচ্ছন্ন করে, প্রতিটি তথ্যকে তার নির্দিষ্ট জায়গায় রাখে। এতে ডেটা ডুপ্লিকেট হয় না এবং ভুল হওয়ার সম্ভাবনা কমে। কিন্তু ডেটা খুঁজে বের করতে মাঝে মাঝে অনেক টেবিল একসাথে জুড়তে (join) হয়, যা একটু সময়সাপেক্ষ।

ডিনরমালাইজেশন: নরমালাইজেশনের উল্টো। এখানে ইচ্ছাকৃতভাবে কিছু ডেটা ডুপ্লিকেট করা হয়, যাতে ডেটা খোঁজার সময় দ্রুত ফলাফল পাওয়া যায়। এটা বিশেষ করে যখন প্রচুর ডেটা পড়তে (read) হয়, তখন কাজে লাগে। কিন্তু এতে ডেটা ভুল হওয়ার ঝুঁকি বেড়ে যায় কারণ একই তথ্য একাধিক জায়গায় থাকে, এবং সব জায়গায় আপডেট না করলে অসঙ্গতি দেখা দিতে পারে।

Advantages of normalization?

নরমালাইজেশনের সুবিধা:

Reduced Data Redundancy (ডেটা রিডানডেন্সি হ্রাস):

Explanation (ব্যাখ্যা): Duplicate data is minimized, saving storage space and ensuring each piece of information is stored in one place.

নকল ডেটা কমানো হয়, স্টোরেজ স্পেস বাঁচায় এবং প্রতিটি তথ্য এক জায়গায় সংরক্ষিত হয় তা নিশ্চিত করে।

Example (উদাহরণ): Storing customer addresses only once in a Customers table, instead of repeating it for every order they place.

গ্রাহকের ঠিকানা শুধুমাত্র Customers টেবিলে একবার সংরক্ষণ করা, তারা যে প্রতিটি অর্ডার দেয় তার জন্য এটি পুনরাবৃত্তি না করে।

Improved Data Integrity and Consistency (উন্নত ডেটা ইন্টিগ্রিটি এবং সামঞ্জস্যতা):

Explanation (ব্যাখ্যা): Data changes are only made in one place, preventing inconsistencies and ensuring accuracy across the database.

ডেটা পরিবর্তন শুধুমাত্র এক জায়গায় করা হয়, অসঙ্গতি প্রতিরোধ করে এবং ডেটাবেস জুড়ে নির্ভুলতা নিশ্চিত করে।

Example (উদাহরণ): If a product's price changes, you update it in the Products table once, and every order referencing that product will automatically reflect the correct price (assuming order items store product ID and not the price directly).

যদি একটি পণ্যের মূল্য পরিবর্তিত হয়, তাহলে আপনি এটি Products টেবিলে একবার আপডেট করেন, এবং সেই পণ্যের উল্লেখ করা প্রতিটি অর্ডার স্বয়ংক্রিয়ভাবে সঠিক মূল্য প্রতিফলিত করবে (যদি অর্ডার আইটেমগুলি পণ্যের আইডি সংরক্ষণ করে এবং সরাসরি মূল্য নয়)।

Elimination of Anomalies (অসঙ্গতি দূরীকরণ):

Explanation (ব্যাখ্যা): Prevents update anomalies (changing data in one place but not another), insertion anomalies (cannot insert data without complete information), and deletion anomalies (deleting one piece of data accidentally deletes another related piece).

আপডেট অসঙ্গতি (এক জায়গায় ডেটা পরিবর্তন কিন্তু অন্য জায়গায় নয়), ইনসার্শন অসঙ্গতি (সম্পূর্ণ তথ্য ছাড়া ডেটা ইনসার্ট করতে না পারা), এবং ডিলিশন অসঙ্গতি (একটি ডেটা মুছে ফেললে দুর্ঘটনাক্রমে অন্য সম্পর্কিত ডেটা মুছে ফেলা) প্রতিরোধ করে।

Example (উদাহরণ): In a non-normalized table, if you delete the last record of a student, you might accidentally delete the only record of a course they were enrolled in. Normalization separates these, so deleting a student doesn't delete course information.

একটি নন-নরমালাইজড টেবিলে, যদি আপনি একজন শিক্ষার্থীর শেষ রেকর্ড মুছে ফেলেন, তাহলে আপনি দুর্ঘটনাক্রমে একটি কোর্সের একমাত্র রেকর্ড মুছে ফেলতে পারেন যা তারা নথিভুক্ত ছিল। নরমালাইজেশন এগুলিকে আলাদা করে, তাই একজন শিক্ষার্থীকে মুছে ফেললে কোর্সের তথ্য মুছে যায় না।

Easier Data Maintenance (ডেটা রক্ষণাবেক্ষণ সহজ):

Explanation (ব্যাখ্যা): With data organized logically, it's simpler to manage, update, and validate.

ডেটা যৌক্তিকভাবে সংগঠিত হলে, এটি পরিচালনা, আপডেট এবং যাচাই করা সহজ হয়।

Better Querying (উন্নত কোয়েরিং):

Explanation (ব্যাখ্যা): Although joins are necessary, queries tend to be more precise and less ambiguous when data is in its most atomic form.

যদিও জয়ের প্রয়োজন হয়, ডেটা তার সবচেয়ে অ্যাটমিক ফর্মে থাকলে কোয়েরিগুলি আরও সুনির্দিষ্ট এবং কম অস্পষ্ট হয়।

Disadvantages of normalization?

নরমালাইজেশনের অসুবিধা:

Increased Complexity for Queries (কোয়েরির জন্য বর্ধিত জটিলতা):

Explanation (ব্যাখ্যা): To retrieve a complete set of related information, you often need to perform multiple JOIN operations across several tables. This can make queries more complex to write and understand.

সম্পর্কিত তথ্যের একটি সম্পূর্ণ সেট পুনরুদ্ধার করতে, আপনাকে প্রায়শই একাধিক টেবিল জুড়ে একাধিক JOIN অপারেশন করতে হয়। এটি কোয়েরি লেখা এবং বোঝা আরও জটিল করে তুলতে পারে।

Example (উদাহরণ): To get a student's name, their course name, and the instructor's name, you might need to join Students, Enrollments, and Courses tables.

একজন শিক্ষার্থীর নাম, তাদের কোর্সের নাম, এবং প্রশিক্ষকের নাম পেতে, আপনাকে Students, Enrollments, এবং Courses টেবিলগুলি জয়েন করতে হতে পারে।

Slower Read Performance for Complex Queries (জটিল কোয়েরির জন্য ধীর পঠন কার্যকারিতা):

Explanation (ব্যাখ্যা): The need for multiple joins can sometimes lead to slower query execution times, especially in data warehousing or reporting scenarios where large amounts of data need to be aggregated from many tables.

একাধিক জয়ের প্রয়োজন কখনও কখনও ধীর কোয়েরি এক্সিকিউশন সময় নিয়ে যেতে পারে, বিশেষত ডেটা ওয়্যারহাউসিং বা রিপোর্টিং পরিস্থিতিতে যেখানে প্রচুর পরিমাণে ডেটা অনেক টেবিল থেকে একত্রিত করতে হয়।

Increased Storage Space (Increased number of tables) (বর্ধিত স্টোরেজ স্পেস - টেবিলের সংখ্যা বৃদ্ধি):

Explanation (ব্যাখ্যা): While it reduces redundant data within columns, normalization typically leads to a higher number of tables and potentially more index overhead, which can sometimes consume more total storage than a highly denormalized structure (though this is often negligible compared to the benefits).

যদিও এটি কলামের মধ্যে অপ্রয়োজনীয় ডেটা হ্রাস করে, নরমালাইজেশন সাধারণত আরও বেশি সংখ্যক টেবিল এবং সম্ভাব্য আরও বেশি ইনডেক্স ওভারহেড তৈরি করে, যা কখনও কখনও একটি অত্যন্ত ডিনরমালাইজড কাঠামোর চেয়ে বেশি মোট স্টোরেজ ব্যবহার করতে পারে (যদিও এটি সুবিধার তুলনায় প্রায়শই নগণ্য)।

More Tables to Manage (পরিচালনার জন্য আরও টেবিল):

Explanation (ব্যাখ্যা): A highly normalized database can have many small tables, which might make it harder for developers or DBAs to understand the schema initially.

একটি উচ্চ নরমালাইজড ডেটাবেসে অনেক ছোট টেবিল থাকতে পারে, যা ডেভেলপার বা ডিবিএদের জন্য প্রাথমিকভাবে স্কিমা বোঝা কঠিন করে তুলতে পারে।

What is redundancy?

Redundancy in a database refers to the duplication or repetition of data within the database. It means storing the same piece of information in multiple places.

ডেটাবেসে রিডানডেন্সি বলতে ডেটাবেসের মধ্যে ডেটার নকল বা পুনরাবৃত্তি বোঝায়। এর অর্থ হলো একই তথ্য একাধিক জায়গায় সংরক্ষণ করা।

Real-life example (বাস্তব জীবনের উদাহরণ):

Imagine a contact list where you store a person's name and phone number.

একটি পরিচিতি তালিকা কল্পনা করুন যেখানে আপনি একজন ব্যক্তির নাম এবং ফোন নম্বর সংরক্ষণ করেন।

Redundant (রিডানডেন্ট):

| ContactID | Name | Phone | Address |

| :---------- | :------- | :--------- | :------------- |

| 1 | Alice | 123-4567 | 1 Main St |

| 2 | Alice | 123-4567 | 1 Main St |

| 3 | Bob | 890-1234 | 2 Elm St |

Here, "Alice" and "123-4567" and "1 Main St" are repeated, which is data redundancy. This can lead to:

এখানে, "Alice" এবং "123-4567" এবং "1 Main St" পুনরাবৃত্তি হয়েছে, যা ডেটা রিডানডেন্সি। এটি নিম্নলিখিতগুলির কারণ হতে পারে:

Wasted Storage Space: You're using more disk space than necessary.

স্টোরেজ স্পেসের অপচয়: আপনার প্রয়োজনের চেয়ে বেশি ডিস্ক স্পেস ব্যবহার হচ্ছে।

Inconsistencies: If Alice's phone number changes, and you update it only in the first row but forget the second, you now have inconsistent data for Alice.

অসঙ্গতি: যদি এলিসের ফোন নম্বর পরিবর্তিত হয়, এবং আপনি শুধুমাত্র প্রথম সারিতে এটি আপডেট করেন কিন্তু দ্বিতীয়টি ভুলে যান, তাহলে আপনার কাছে এখন এলিসের জন্য অসঙ্গতিপূর্ণ ডেটা থাকবে।

Update Anomalies: Difficult to update.

আপডেট অসঙ্গতি: আপডেট করা কঠিন।

Deletion Anomalies: If you delete ContactID 1, you might lose Alice's information, even if ContactID 2 is meant to be a separate entry for some reason.

ডিলিশন অসঙ্গতি: যদি আপনি ContactID 1 মুছে ফেলেন, তাহলে আপনি এলিসের তথ্য হারাতে পারেন, এমনকি যদি ContactID 2 কোনো কারণে একটি পৃথক এন্ট্রি হতে থাকে।

Normalization aims to eliminate or significantly reduce this redundancy.

নরমালাইজেশনের লক্ষ্য হলো এই রিডানডেন্সি দূর করা বা উল্লেখযোগ্যভাবে কমানো।