Learning to Generate Unique Identifiers (UIDs) in Google Sheets

Authored by

Mohammed looti

November 11, 2025

Generating **unique identifiers** (UIDs) for individual records is a foundational requirement when manipulating large datasets in spreadsheet environments like **Google Sheets**. These identifiers are not merely cosmetic labels; they serve as critical primary keys, ensuring absolute data integrity, streamlining complex lookup operations, and facilitating reliable cross-referencing between different analytical views or tables. The robust application of UIDs is paramount for maintaining order and efficiency in any substantial data project.

Despite the central role UIDs play in rigorous **data management** practices, Google Sheets does not provide a single, integrated function for automatically assigning sequential, categorical, or globally unique keys. To overcome this limitation, users must leverage sophisticated combinations of powerful, built-in functions. By mastering these nested formulas, we can reliably and reproducibly generate UIDs based either on the content of the data (grouping categories) or the physical sequence of the row (tracking instances).

This comprehensive guide details two distinct, highly effective methodologies for generating UIDs, addressing two different analytical imperatives: first, assigning a consistent, standardized ID to all identical entries for categorical normalization; and second, creating a truly unique, row-specific identifier crucial for auditing and tracking individual records.

The Crucial Role of Unique Identifiers in Data Integrity

In analytical and operational environments, relying solely on volatile row numbers or ambiguous, non-standardized text entries for data cross-referencing is highly susceptible to error and instability. A meticulously designed unique identifier resolves this fundamental challenge by supplying a standardized, fixed, and unambiguous reference point for every record within the dataset. When managing operational data--such as tracking sales transactions, inventory movements, or, as demonstrated in our examples, recording team appearances--it is often necessary to classify and group identical textual entries under a single, efficient numeric code.

This systematic technique is exceptionally valuable when preparing source data for processing activities, including the construction of pivot tables, the design of dynamic dashboards, or integration into external database systems. In these downstream applications, numeric keys offer dramatically improved efficiency for processing speed and storage optimization compared to long, repetitive text strings. Crucially, the ability to automate the assignment of these keys using dynamic formulas ensures that as the dataset expands and new data is introduced, the identification system remains perfectly consistent and inherently scalable without demanding continuous manual intervention.

We will delve into how complex, nested logical structures, utilizing key functions such as **MATCH** and **VLOOKUP**, can intelligently assess whether an entry has previously been classified and assigned an ID. This logical structure is essential for guaranteeing immediate consistency across

all repeated instances of a given category, thereby standardizing the entire data column.

Method 1: Generating Consistent Numeric IDs for Categorical Grouping

The first analytical scenario addresses the frequent need to assign the exact same unique numeric ID to every occurrence of a specific item or category name. For instance, if the team "Mavs" is listed ten separate times within a column, all ten corresponding rows must be assigned the identical ID, such as "1". This method is indispensable for the critical process of normalizing categorical variables into fixed, numeric factor codes, which is a prerequisite for many statistical models.

To clearly demonstrate this procedure, we will work with a sample list of basketball team names in Google Sheets. Our primary objective is to populate Column B with a sequential, unique numeric identifier that corresponds consistently to the team name found in Column A:

	A	B	C
1	**Team**		
2	Mavs		
3	Lakers		
4	Mavs		
5	Hawks		
6	Hawks		
7	Warriors		
8	Mavs		
9	Lakers		
10	Nets		
11	Spurs		
12	Lakers		
13			
14			
15			
16			
17			

Because the subsequent formula relies on dynamic checking and assignment logic, we must manually establish the initial unique identifier for the very first entry in the list. For the purposes of this illustration, we will assign the value **1** to the first team listed, placing this value directly into cell B2. This initialization step provides the necessary foundational reference point, allowing the

subsequent formula logic to accurately expand and build upon this starting numerical sequence.

	A	B	C	D
1	**Team**	**Unique ID**		
2	Mavs	1		
3	Lakers			
4	Mavs			
5	Hawks			
6	Hawks			
7	Warriors			
8	Mavs			
9	Lakers			
10	Nets			
11	Spurs			
12	Lakers			
13				
14				
15				
16				

Step-by-Step Implementation of the Dynamic Categorical ID Formula

The operational core of this method resides in a highly sophisticated nested formula designed to execute a three-part logical assessment: (1) Does the current team name already exist in the preceding data? (2) If the name exists, immediately retrieve its previously assigned ID. (3) If the name is entirely new, dynamically calculate and assign the next available, highest ID number. This powerful structure ensures efficiency and non-duplication.

To implement this precise logical flow, the following comprehensive formula is entered into cell **B3**. It is essential to pay close attention to the meticulous utilization of absolute ($A$2) and relative (A2) referencing, which is crucial for defining the correct expanding lookup range as the formula is applied down the column:

=IF(ISNA(MATCH(A3,A2:$A$2,0)),MAX(B2:$B$2)+1,VLOOKUP(A3,A2:$B$2,2,FALSE))

This robust, nested function operates through a sequence of logical checks to ensure accurate assignment:

The **MATCH(A3,A2:$A$2,0)** function initiates the process by attempting to locate the current team name (A3) within the range of all team names processed prior to the current row (A2:$A$2). The absolute anchoring of $A$2 guarantees that the lookup range dynamically expands with each row the formula is dragged to.

The **ISNA()** function wraps the **MATCH** result, checking specifically if an error (#N/A) is returned. This error condition occurs exclusively when the team name in A3 has *not* yet been encountered or assigned an ID in the list above the current row.

If **ISNA()** evaluates to TRUE (meaning a new category is identified), the formula executes the TRUE part of the **IF** statement: **MAX(B2:$B$2)+1**. This segment finds the highest existing ID number within the assigned range (B2:$B$2) and increments it by 1, successfully generating the necessary new **unique identifier** for the new category.

If **ISNA()** evaluates to FALSE (meaning the name has been previously assigned an ID), the formula executes the FALSE part of the **IF** statement: **VLOOKUP(A3,A2:$B$2,2,FALSE)**. This powerful mechanism efficiently retrieves the corresponding, already-assigned ID number for that specific team name from the established two-column range.

Once the formula has been accurately entered into cell B3, the implementation is completed by simply clicking and dragging the formula down to the remainder of Column B. The intelligent design utilizing expanding ranges ensures that every record is systematically checked against all preceding records, maintaining perfect consistency throughout the dataset.

B3 =IF(ISNA(MATCH(A3,$A$2:A2,0)),MAX($B$2:B2)+1,VLOOKUP(A3,A$2:$B2,2,FALSE))

	A	B	C	D	E	F
1	**Team**	**Unique ID**				
2	Mavs	1				
3	Lakers	2				
4	Mavs	1				
5	Hawks	3				
6	Hawks	3				
7	Warriors	4				
8	Mavs	1				
9	Lakers	2				
10	Nets	5				
11	Spurs	6				
12	Lakers	2				
13						
14						
15						

The resulting values in Column B now reliably contain a consistent unique ID for each team name,

regardless of the frequency of its appearance. This methodology successfully ensures that the crucial relational link between the category name and its derived numeric identifier is preserved throughout the data structure. For visual confirmation, observing the completed Column B shows the clear grouping:

Every instance of the team name "Mavs" has been standardized with an ID value of **1**.
Every instance of the team name "Lakers" has been standardized with an ID value of **2**.
Every instance of the team name "Hawks" has been standardized with an ID value of **3**.

Method 2: Creating Row-Specific Sequential IDs (The Instance Tracking Approach)

While the Categorical ID method (Method 1) is optimally suited for grouping and normalization, a different requirement frequently arises: generating a truly **unique identifier** for every single row, even when the primary data column (Column A) contains numerous duplicate entries. This specific type of UID is indispensable for robust auditing purposes, generating distinct transaction codes, or creating highly specific tracking numbers where the instance of the event matters more than the category itself.

This instance-tracking approach is typically executed by concatenating the original data value (the category) with a sequential count of its appearances. The resulting complex identifier achieves global uniqueness because it combines the category (e.g., "Mavs") with its specific instance number (e.g., the 1st Mavs, the 2nd Mavs, the 3rd Mavs, and so on). This concatenation provides both uniqueness and immediate human readability.

We can efficiently implement this solution using the highly versatile **COUNTIF** function in conjunction with the standard concatenation operator (&). The following formula, entered directly into cell B2, offers a compact and exceptionally effective mechanism for indexing each row sequentially based on the repeated occurrences of the value found in Column A:

=A2&"-"&COUNTIF($A$2:A2,A2)*1

Dissecting the COUNTIF Formula for Unique Row Indexing

The elegance of this formula lies in its strategic use of referencing to construct a running count that increments exclusively for the specific data value being evaluated in the current row. A detailed breakdown of the formula components reveals its indexing mechanism:

A2&"-"&...: This segment establishes the output format. It takes the value from cell A2 (the category name) and merges it (concatenates) with a simple hyphen ("-"), followed by the calculated

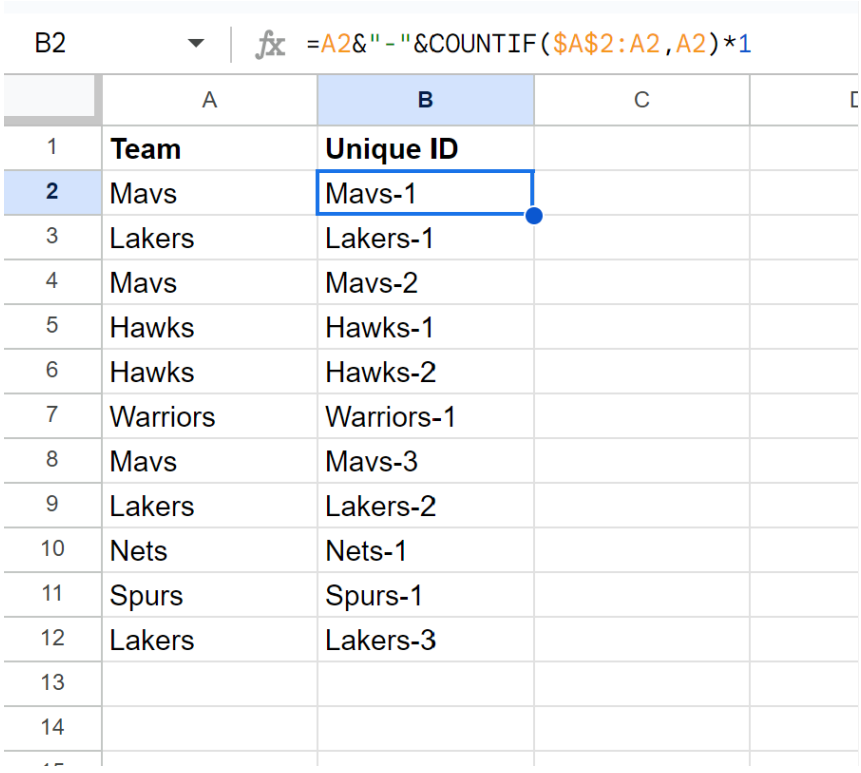
result from the subsequent COUNTIF function.

COUNTIF($A$2:A2,A2): This component forms the absolute core of the indexing mechanism. The definition of the range $A$2:A2 is mathematically critical: the starting point ($A$2) is fixed using an absolute reference, while the endpoint (A2) is a relative reference. This design ensures that as the formula is dragged down the sheet, the range dynamically expands (e.g., changing to $A$2:A3, $A$2:A4, and so forth).

The **COUNTIF** function then performs its duty by counting how many times the value in A2 (the criterion) has appeared within the currently expanding range defined by $A$2:A2. If "Mavs" is first found in row 2, the count returns 1. If "Mavs" is subsequently encountered in row 5, the COUNTIF checks the range $A$2:A5 and correctly returns 2, as it represents the second specific instance of "Mavs".

***1**: The optional step of multiplying the COUNTIF result by 1 serves to explicitly ensure that the function's output is treated purely as a numeric value before concatenation, leading to a cleaner string output, although **Google Sheets** is often capable of implicit type handling.

By applying this formula consistently down Column B, we successfully generate a highly detailed, row-specific identifier for every record in the list:



	A	B	C	D
1	**Team**	**Unique ID**		
2	Mavs	Mavs-1		
3	Lakers	Lakers-1		
4	Mavs	Mavs-2		
5	Hawks	Hawks-1		
6	Hawks	Hawks-2		
7	Warriors	Warriors-1		
8	Mavs	Mavs-3		
9	Lakers	Lakers-2		
10	Nets	Nets-1		
11	Spurs	Spurs-1		
12	Lakers	Lakers-3		
13				
14				

The resulting composite identifiers, such as "Mavs-1," "Mavs-2," and "Lakers-1," are now guaranteed to be unique across all individual row records. This instance-tracking approach is exceptionally powerful for systems requiring the tracking of specific events within a broad category,

making it invaluable for detailed transaction logs, inventory control, or any scenario where the exact order or specific occurrence of an event is a critical piece of metadata.

Advanced Scalability and Strategic Considerations

While both Method 1 and Method 2 offer robust and distinct solutions for generating **unique identifiers**, a thorough understanding of the implications of each approach is essential for successful advanced **data analysis** and manipulation.

When employing Method 1 (Categorical ID), it is important to acknowledge that the assignment of IDs is intrinsically dependent on the initial sequential order of the data. If the dataset is subsequently sorted, the existing IDs remain accurate and consistent for their respective categories. If, however, new, unique categories are inserted into the middle of the already processed list, the formula will correctly assign the next sequential ID based on the current **MAX** value found above it, ensuring that data integrity is maintained even with incremental updates.

For processing exceptionally large datasets, particularly those that surpass 10,000 rows, relying on formulas dragged down thousands of cells can introduce significant calculation overhead and potential performance degradation. In these high-volume scenarios, complex array formulas utilizing functions like **ARRAYFORMULA** or the strategic implementation of streamlined helper columns become necessary optimization techniques. Nevertheless, for the vast majority of standard business applications and moderate datasets, the detailed methods provided here offer highly reliable and easily deployable solutions.

Ultimately, the choice between generating a simple numeric UID (Method 1) and a descriptive concatenated string UID (Method 2) should be dictated entirely by the requirements of the downstream process. Numeric keys are the superior choice when the primary objective is relational lookups, pivot table construction, and statistical computations, whereas concatenated keys provide immediate human readability and offer self-descriptive indexing, simplifying manual checks and validation processes.

Additional Resources for Comprehensive Spreadsheet Mastery

Mastering the dynamic generation and sophisticated manipulation of **unique identifiers** is just one specialized component of achieving advanced spreadsheet proficiency. To continue enhancing your capabilities in data processing and analytical reporting within **Google Sheets**, we highly recommend exploring tutorials focused on complementary topics. These skills are fundamental for any professional tasked with managing complex data models, constructing automated reporting systems, or performing detailed data validation.

The following tutorials explain how to perform other common tasks in Google Sheets and

complement the robust techniques discussed in this guide: