# Cardinality

Cardinality's official, non-database dictionary definition is mathematical: the number of values in a set. When applied to databases, the meaning is a bit different: the number of distinct values in a table column relative to the number of rows in the table. Repeated values in the column don't count. In database management, cardinality plays an important role. Here cardinality represents the number of times an entity of an entity set participates in a relationship set. Or we can say that the cardinality of a relationship is the number of tuples (rows) in a relationship.

With this definition of database cardinality in mind, it can mean two things in practice. For our purposes, one matters a lot more than the other. Let's explore the simple definition first and then dig into why cardinality matters for query performance.

**What Is Cardinality in Data Modeling?**

The first meaning of cardinality is when you're designing the database—what's called data modeling. In this sense, cardinality means whether a relationship is one-to-one, many-to-one, or many-to-many. So, you're really talking about the *relationship* cardinality.

Types of cardinality in between tables are:

- one-to-one
- one-to-many
- many-to-one
- many-to-many

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

In a one-to-one relationship, each record in the first table is related to exactly one record in the second table, and vice versa.

*Example:*

Consider two tables:

- **People**: Contains columns such as "ID" and "Name".
- **Driving Licenses**: Contains columns like "LicenseID" and "PersonID".

In this scenario, each person has one unique driving license, and each driving license is assigned to one person. Therefore, the "ID" in the People table corresponds to one "PersonID" in the DrivingLicenses table.

**One-to-Many (1**

**)**

In a one-to-many relationship, a record in the first table can be related to one or more records in the second table, but each record in the second table is related to only one record in the first table.

**Example:**

Consider two tables:

- **Parents**: Contains columns such as "ParentID" and "Name".
- **Children**: Contains columns like "ChildID" and "ParentID".

Here, each parent can have multiple children, but each child is related to only one parent. Thus, the "ParentID" in the Parents table can link to multiple "ParentID"s in the Children table, but each child has only one "ParentID".

**Many-to-One (N:1)**

This is the inverse of the one-to-many relationship. In a many-to-one relationship, many records in the first table can be related to one record in the second table, but each record in the second table is related to one or more records in the first table.

**Example:**

Consider two tables:

- **Employees**: Contains columns such as "EmployeeID" and "DepartmentID".
- **Departments**: Contains columns like "DepartmentID" and "DepartmentName".

In this case, many employees can belong to one department, but each department can have many employees. Therefore, the "EmployeeID" in the Employees table can link to only one "DepartmentID", while each "DepartmentID" in the Departments table can relate to multiple employees.

**Many-to-Many (N**

**)**

In a many-to-many relationship, multiple records in the first table can be related to multiple records in the second table, and vice versa. This relationship is often managed through a junction table.

**Example:**

Consider three tables:

- **Students**: Contains columns such as "StudentID" and "Name".
- **Courses**: Contains columns like "CourseID" and "CourseName".
- **Enrollments**: A junction table with columns like "StudentID" and "CourseID".

Here, each student can enroll in multiple courses, and each course can have multiple students enrolled. The "StudentID" in the Students table links to multiple "StudentID"s in the Enrollments table, and the "CourseID" in the Courses table links to multiple "CourseID"s in the Enrollments table. The Enrollments table facilitates the many-to-many relationship between students and courses.

**Summary**

- **One-to-One (1:1)**: A single record in one table is related to a single record in another table.
- **One-to-Many (1**

**)**: A single record in one table is related to multiple records in another table.

- **Many-to-One (N:1)**: Multiple records in one table are related to a single record in another table.
- **Many-to-Many (N**

**)**: Multiple records in one table are related to multiple records in another table, typically managed through a junction table.

**High and Low Database Cardinality Definition**

The more important definition of cardinality for query performance is *data* cardinality. This is all about how many distinct values are in a column.

We usually don't talk about cardinality as a *number*, though. It's more common to simply talk about "high" and "low" cardinality. **A lot of distinct values is high cardinality; a lot of repeated values is low cardinality.**

**Cardinality in Database Example**

Picture a product description table in an e-commerce database:

| ProductID | Category | Name |
|---|---|---|
| 788830 | Headphones | Bose QuietComfort |
| 313096 | Bluetooth Speakers | Polk BOOM |
| 814976 | Bluetooth Speakers | JBL Clip |
| 105945 | Headphones | Audio-Technica ATH |
| 666721 | Bluetooth Speakers | Jamo DS3 |

The **ProductID** column is going to have high cardinality because it's probably the primary key of the table, so it's totally unique. If there's a thousand rows in the table, there'll be a thousand different ProductID values.

The **Category** column will have a lot of repetition, and it'll have low or medium cardinality: maybe 50 or 100 different Category values.

**Name** probably has high cardinality, unless there's more to this table than meets the eye (such as multiple rows for different product colors and other variations).

**Importance of cardinality in Data Science**

**Data Cleaning and Preparation**:

- **Handling High Cardinality**: Columns with high cardinality, such as user IDs or timestamps, may require special handling to avoid excessive memory usage and computational complexity. Techniques like hashing or dimensionality reduction can be employed to manage these columns efficiently.
- **Handling Low Cardinality**: Columns with low cardinality, such as binary or categorical data (e.g., gender, country), are easier to manage but might still require appropriate encoding methods to be useful in machine learning models.

**Importance in Machine Learning**

1. **Model Selection**:
    - **Algorithm Suitability**: Some machine learning algorithms handle high cardinality features better than others. For example, tree-based methods like decision trees or random forests can manage high cardinality categorical features without requiring extensive preprocessing, while linear models might struggle.
    - **Scalability and Performance**: High cardinality can impact the scalability and performance of machine learning models. Efficiently handling such features can lead to more scalable solutions and faster training times.
2. **Dimensionality Reduction**:
    - **Techniques like PCA and t-SNE**: High cardinality can lead to high-dimensional data, which can be challenging to process. Dimensionality reduction techniques such as Principal Component Analysis (PCA) or t-SNE can help reduce the number of features while retaining important information, making the data more manageable for machine learning models.
3. **Managing Multicollinearity**:
    - **Detecting and Mitigating Multicollinearity**: High cardinality in categorical variables can lead to multicollinearity in models, especially in linear regression. Identifying and addressing multicollinearity is crucial for creating stable and interpretable models.
4. **Evaluation and Interpretation**:
    - **Model Evaluation Metrics**: Understanding cardinality helps in selecting appropriate metrics for model evaluation. For example, in a classification problem with high cardinality categorical features, precision and recall might be more informative than accuracy.
    - **Interpretability**: Handling cardinality properly ensures that the resulting models are interpretable. For instance, models that include one-hot encoded low cardinality features are generally easier to interpret than those with complex encodings of high cardinality features.

**Practical Applications**

1. **Recommendation Systems**: High cardinality features like user IDs and item IDs are common in recommendation systems. Proper handling of these features is crucial for building effective recommendation models.
2. **Fraud Detection**: In fraud detection, transaction IDs or user IDs often exhibit high cardinality. Efficiently processing these features can improve the accuracy of fraud detection models.
3. **Customer Segmentation**: Low cardinality features such as demographic information are frequently used in customer segmentation. Appropriate handling ensures accurate and meaningful segmentation.

## Conclusion

Understanding and managing cardinality is essential in data science and machine learning. It impacts various stages of the data pipeline, from data cleaning and feature engineering to model selection and evaluation. Proper handling of cardinality leads to more efficient, scalable, and interpretable models, ultimately enhancing the quality and reliability of data-driven insights.