Here is a tabular representation highlighting the key differences between the **TRUNCATE** and **DELETE** statements in SQL:

| **Feature** | **TRUNCATE** | **DELETE** |
| --- | --- | --- |
| **Operation Type** | DDL (Data Definition Language) | DML (Data Manipulation Language) |
| **Atomicity** | Atomic (Transaction cannot be rolled back) | Non-atomic (Can be part of a transaction that can be rolled back) |
| **Usage on Tables with Foreign Keys** | Limited. Usually requires disabling or dropping foreign keys before truncating and re-enabling them after truncation. | Can be used without disabling foreign keys. |
| **Logging** | Minimal logging, usually faster. | Fully logged, potentially slower due to log generation. |
| **Where Clause** | Cannot use a **WHERE** clause. | Can use a **WHERE** clause to filter specific rows based on conditions. |
| **Data Removal** | Removes all rows from the table. | Removes specific rows based on the **WHERE** clause or deletes all rows if no **WHERE** clause is specified. |
| **System Resource Utilization** | Generally uses fewer system resources. | Can be resource-intensive, especially for large datasets. |
| **Transaction Rollback** | Cannot be rolled back. | Can be rolled back within a transaction. |
| **Identity Column Reset** | Resets identity columns to the seed value (if applicable). | Identity columns retain their current values. |
| **Trigger Execution** | Does not activate DELETE triggers. | Activates DELETE triggers if defined on the table. |

These differences highlight the use cases and considerations for choosing between **TRUNCATE** and **DELETE** based on the specific requirements and constraints of the database operation.