# What is mysql.ibd?

The mysql.ibd file is typically created for system tables in the MySQL database if innodb\_file\_per\_table is enabled.

**Path:** C:\ProgramData\MySQL\MySQL Server 8.0\Data\mysql.ibd

## When is mysql.ibd Created?

Enable innodb\_file\_per\_table:

SET GLOBAL innodb\_file\_per\_table = 1;

## Will the Size of mysql.ibd Increase or Decrease After Installation?

### Addition/Deletion of Data in System Tables

* **Increase in Size:**
  + New user accounts are added to the user table.
  + New privileges are added to the db, tables\_priv, columns\_priv, etc.
* **Decrease in Size:**
  + User accounts or privileges are deleted.
  + Other system metadata is removed or cleaned up.

### Table Modification

If there are changes to the structure of system tables (like adding or removing columns), this may also affect the size of the .ibd file.

### Automatic Updates or Internal MySQL Operations

MySQL internally might update its system tables, and this could result in an increase in the .ibd file size.

### Autonomous Growth (Reclaiming Space)

If data is deleted, MySQL does not shrink the .ibd file automatically (even if space becomes free).

## Will the Size Decrease After Data Removal?

While the size may increase due to added data, the .ibd file will not shrink automatically when data is deleted from the system tables. MySQL does not automatically reclaim space within .ibd files, so the file size might stay the same or increase, even if the system tables have less data.

To shrink the .ibd file, you would need to:

* Optimize the table (though it’s typically not done for system tables):
* OPTIMIZE TABLE user; -- Example for a user table in the MySQL database

## What if in my database school db I add 500 tables? Will it affect mysql.ibd or just those 500 tables’ ibd files?

mysql.ibd will not be affected by the addition of tables in schooldb. It is reserved for MySQL system tables like user, db, etc.

The size of mysql.ibd will only change if there are updates or changes to the system tables in the MySQL database (like adding/deleting users, altering privileges, etc.).

## List of System Tables (during the installation of MySQL):

* **user:** Stores MySQL user accounts and their privileges.
* **db:** Contains database-level privileges.
* **tables\_priv:** Stores table-level privileges.
* **proxies\_priv:** Stores proxy user information.
* **columns\_priv:** Stores column-level privileges.
* **host:** Defines the host-based access control for users.
* **server\_cost:** Stores InnoDB cost data.
* **innodb\_index\_stats, innodb\_table\_stats:** InnoDB statistics tables.

## Other .ibd Files:

When innodb\_file\_per\_table is enabled, each user-created table will have its own individual .ibd file for storage. This means that the .ibd file is created for each table you create in a database (e.g., schooldb).

**Path:** C:\ProgramData\MySQL\MySQL Server 8.0\Data\schooldb\stu01.ibd

## What if we don’t enable innodb\_file\_per\_table?

By default, when innodb\_file\_per\_table is disabled, InnoDB stores all its tables’ data in a shared tablespace file (e.g., ibdata1). When enabled, InnoDB stores each table’s data and indexes in its own .ibd file.

**Path:** C:\ProgramData\MySQL\MySQL Server 8.0\Data\ibdata1

## Advantages of Enabling Over Storing in a Shared Tablespace

1. **Improved Performance:** When all tables are stored in a single shared tablespace (e.g., ibdata1), multiple tables contend for the same storage blocks. With innodb\_file\_per\_table, tables are stored separately, reducing contention and potentially improving I/O performance.
2. **Easier Backup and Restoration:** With innodb\_file\_per\_table, each table’s data is stored in its own .ibd file, which makes backing up individual tables more efficient. You can back up or restore a single table without affecting the rest of the database.

* You can easily copy the .ibd files to perform physical backups (using tools like Percona XtraBackup).

1. **Easier Space Management:** With innodb\_file\_per\_table enabled, you can more easily reclaim space from a table using the OPTIMIZE command or by dropping the table, as the space is freed from the .ibd file.

## Alternatives to innodb\_file\_per\_table

### Using a Shared Tablespace (innodb\_file\_per\_table = OFF)

When innodb\_file\_per\_table is disabled, MySQL uses a single shared tablespace for storing all InnoDB data. All tables and indexes are stored in the ibdata1 file, which also includes undo logs, the InnoDB internal data dictionary, and other system-related data.

#### Advantages:

You don’t have to worry about managing a large number of .ibd files.

#### Disadvantages:

* If you delete a table or data, the space is not reclaimed in the ibdata1 file, and the file can keep growing over time.
* Backing up and restoring individual tables becomes harder because all tables share the same storage.

If you are using the shared tablespace (innodb\_file\_per\_table is disabled), system tables like user, db, etc., will be stored in the same ibdata1 file, along with all the other InnoDB tables and internal data structures.

## Live Use Case

* **Created a database:** testdb0
* **Then I created 1 table in it.**
* **Disk size of all individual .ibd files:** 62.5 MB
* **Insert 6800 records in it:** 384 KB
* **Deleted records in its remaining:** 3332
* **After delete records:** 400 KB
* **After optimizing:** 256 KB

### 1. PostgreSQL Storage Architecture

PostgreSQL organizes its database storage into several components:

* **Tablespaces:** A tablespace is a location on the filesystem where database objects like tables and indexes are stored. By default, all data is stored in the pg\_default tablespace unless explicitly assigned to a different one.
* **Database Files:** Each PostgreSQL database is stored in its own subdirectory under the base directory inside the data directory.
* **Relation Files:** Tables, indexes, and other objects are stored in relation files. A single table or index may span multiple files if it exceeds 1 GB (using a segment size of 1 GB).

### 2. Key Files and Directories in PostgreSQL

* **Data Directory ($PGDATA):**
  + Contains all data and metadata.
  + Example: C:\Program Files\PostgreSQL\<version>\data (on Linux) or a custom directory set during initialization.
* **Relation Files:**
  + Each table or index is stored as one or more files in the database’s directory.
  + Filenames are numeric OIDs (Object IDs) rather than table names.
  + Example: 12345 (table), 12345.1 (if segmented).
* **WAL Files (Write-Ahead Logging):**
  + Located in pg\_wal (or pg\_xlog in older versions).
  + Used for crash recovery and replication.
* **System Catalog Tables:**
  + PostgreSQL stores system metadata in catalog tables in the pg\_catalog schema (e.g., pg\_class, pg\_attribute).

### 3. Handling Data Growth and Space Management

#### **a. Tablespace Management**

* Unlike MySQL’s innodb\_file\_per\_table, PostgreSQL allows you to define **tablespaces** for better space and performance management.
* Create a new tablespace:
* CREATE TABLESPACE myspace LOCATION '/path/to/directory';
* Assign tables to a specific tablespace:
* CREATE TABLE mytable (id SERIAL) TABLESPACE myspace;

#### **b. Vacuuming**

* PostgreSQL does not automatically reclaim space when rows are updated or deleted. Instead, it uses **MVCC (Multi-Version Concurrency Control)** to keep old row versions for transactions.
* To reclaim space:
  + **Autovacuum:** Automatically cleans up dead tuples in the background.
  + **Manual VACUUM:**
  + VACUUM table\_name; -- Reclaims storage but does not shrink files
  + **Full Vacuum (Shrinks Files):**
  + VACUUM FULL table\_name;

#### **c. Index Optimization**

* Over time, indexes can become bloated. Use the following to rebuild:
* REINDEX TABLE table\_name;

### 4. Key Differences: MySQL vs PostgreSQL File Management

| Feature | MySQL (InnoDB) | PostgreSQL |
| --- | --- | --- |
| Default Tablespace | Shared or innodb\_file\_per\_table (optional) | Shared (pg\_default), but can use custom |
| Storage File for a Table | .ibd file (or ibdata1 if shared) | Relation files in base (numeric OID files) |
| Space Reclamation | OPTIMIZE TABLE or ALTER TABLE | VACUUM or VACUUM FULL |
| Index Management | Handled automatically, or rebuilt manually | Rebuild using REINDEX |
| Temporary Tablespaces | Custom temporary tablespaces supported | Supported (pg\_temp tablespaces) |

### 5. Reclaiming Space in PostgreSQL

If a table’s file size has grown due to excessive updates/deletes, you can reclaim space:

1. **VACUUM FULL:**
   * Shrinks files by rewriting the table.
   * Locks the table during the process.
   * Example:
   * VACUUM FULL table\_name;
2. **Cluster Table:**
   * Rewrites the table and orders rows based on an index.
   * Also reclaims space.
   * Example:
   * CLUSTER table\_name USING index\_name;
3. **Rebuilding Tables:**
   * Manually create a new table and copy data:
   * CREATE TABLE new\_table AS TABLE old\_table;  
     DROP TABLE old\_table;  
     ALTER TABLE new\_table RENAME TO old\_table;