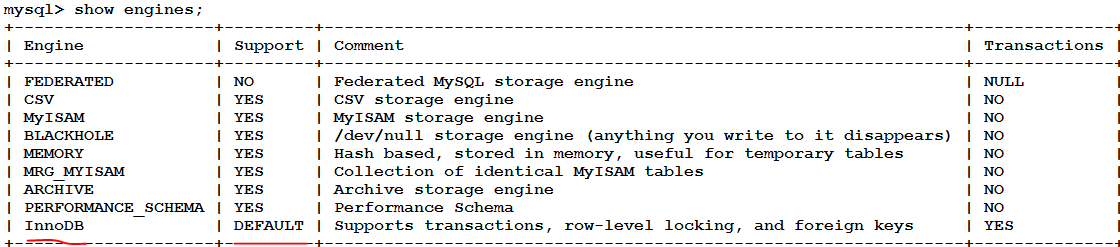
**The InnoDB Storage Engine**

InnoDB is a transaction -safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash- recovery capabilities to protect user data.

InnoDB row- level locking (without escalation to coarser granularity locks) and Oracle-style consistent nonlocking reads increase multi-user concurrency and performance.

To maintain data integrity, InnoDB also supports FOREIGN KEY referential- integrity constraints. You can freely mix InnoDB tables with tables from other MySQL storage engines, even within the same statement.

You can see with the help of this statement mysql> show engines;



The InnoDB storage engine maintains its own buffer pool for caching data and indexes in main memory. InnoDB stores its tables and indexes in a tablespace, which may consist of several files (or raw disk partitions).

This is different from, for example, MyISAM tables where each table is stored using separate files. InnoDB tables can be very large even on operating systems where file size is limited to 2GB.

By Default Innodb from MySQL Version 5.5

**Configuring InnoDB**

The first decisions to make about InnoDB configuration involve how to lay out InnoDB data files, and how much memory to allocate for the InnoDB storage engine. You record these choices either by recording them in a configuration file that MySQL reads at startup, or by specifying them as command-line options in a startup script.

Two important disk-based resources managed by the InnoDB storage engine are its tablespace data files and its log files. If you specify no InnoDB configuration options, **MySQL creates an auto-extending 10MB data file named ibdata1 and two 5MB log files named ib\_logfile0 and ib\_logfile1 in the MySQL data directory**. To get good performance, explicitly provide InnoDB parameters.

**Considerations for Storage Devices:**

Database performance improves if the data is not all placed on the same physical disk. Putting log files on a different disk from data is very often beneficial for performance. The example illustrates how to do this. It places the two data files on different disks and places the log files on the third disk. InnoDB fills the tablespace beginning with the first data file. You can also use raw disk partitions (raw devices) as InnoDB data files, which may speed up I/O.

**InnoDB Table and Index Structures**

MySQL stores its data dictionary information for tables in .frm files in database directories. This is for all MySQL storage engines, but every InnoDB table also has its own entry in the InnoDB internal data dictionary inside the tablespace. When MySQL drops a table or a database, it has to delete one or more .frm files as well as the corresponding entries inside the InnoDB data dictionary.

Consequently, you cannot move InnoDB tables between databases simply by moving the .frm files.

**Managing InnoDB Data and Log Files**

To set up the InnoDB tablespace files, use the innodb\_data\_file\_path option in the [mysqld] section of the my.cnf option file. On Windows, you can use my.ini instead. The value of innodb\_data\_file\_path should be a list of one or more data file specifications.

If you name more than one data file, separate them by semicolon (“;”) characters:

innodb\_data\_file\_path=datafile\_spec1[;datafile\_spec2]...

For example, the following setting explicitly creates a tablespace having the same characteristics as the default:

[mysqld] **innodb\_data\_file\_path=ibdata1:50M:autoextend**

This setting configures a single 10MB data file named ibdata1 that is auto-extending. No location for the file is given, so by default, InnoDB creates it in the MySQL data directory. Sizes are specified using K, M, or G suffix letters to indicate units of KB, MB, or GB.

A tablespace containing a fixed-size 50MB data file named ibdata1 and a 50MB auto-extending file named ibdata2 in the data directory can be configured like this:

[mysqld] innodb\_data\_file\_path=ibdata1:12M;ibdata2:12M:autoextend

The **autoextend** and **max** attributes can be used only for the last data file in the innodb\_data\_file \_path line. If you specify the autoextend option for the last data file, InnoDB extends the data file if it runs out of free space in the tablespace.

The increment is 8 MB at a time by default, To modify the increment, change the innodb\_autoextend\_increment system variable.

Mysql> show variables like 'innodb\_autoextend\_increment';

InnoDB is not aware of the file system maximum file size, so be cautious on file systems where the maximum file size is a small value such as 2GB. To specify a maximum size for an auto-extending data file, use the max attribute following the autoextend attribute. Use the max attribute only in cases where constraining disk usage is of critical importance, because exceeding the maximum size causes a fatal error, possibly including a crash. The following configuration permits ibdata1 to grow up to a limit of 500MB.

[mysqld]

innodb\_data\_file\_path=ibdata1:10M:autoextend:m ax:500M

For example:- mysql> show variables like 'innodb\_data\_file\_path';

InnoDB creates tablespace files in the MySQL data directory by default. To specify a location explicitly, use the innodb \_data\_home\_dir option. For example, to use two files named ibdata1 and ibdata2 but create them in the /ibdata directory, configure InnoDB like this:

For example:- mysql> show variables like 'innodb\_data\_home\_dir';

[mysqld]

innodb\_data\_home\_dir = /var/lib/mysql/sqldata

innodb\_data\_file\_path=ibdata1:50M;ibdata2:50M:autoextend

$sudo chown -R mysql:mysql /var/lib/mysql/ibdata

In this example you can see the redo log information and location

mysql>show variables like 'innodb\_log\_group\_home\_dir';

mysql> show variables like 'innodb\_log\_group\_home\_dir';

+---------------------------+-----------------------+

| Variable\_name | Value |

+---------------------------+-----------------------+

| innodb\_log\_group\_home\_dir | /var/lib/mysql/ibdata |

+---------------------------+-----------------------+

1 row in set (0.01 sec)

mysql> select @@global.innodb\_log\_files\_in\_group;

+------------------------------------+

| @@global.innodb\_log\_files\_in\_group |

+------------------------------------+

| 2 |

+------------------------------------+

Mysql> show variables like 'innodb\_log\_files\_in\_group';

**Note:** InnoDB does not create directories, so make sure that the /ibdata directory exists before you start the server.Even log file directories that you have to configure. Use the Unix or windows mkdir command to create any necessary directories. Make sure that the MySQL server has the proper access rights to create files in the data directory. The Mysql server must have access rights in any directory where it needs to create data files or log files.

InnoDB forms the directory path for each data file by textually concatenating the value of innodb\_data\_home\_dir to the data file name, adding a path name separator (slash or backslash) between values if necessary. If the innodb\_data\_home \_dir option is not mentioned in my.cnf at all, the default value is the “dot” directory ./, which means the MySQL data directory. (The MySQL server changes its current working directory to its data directory when it begins executing.)

If you specify innodb\_data\_home\_dir as an empty field, you can specify absolute paths for the data files listed in the innodb\_data\_file\_path value. Below you can find

[mysqld]

innodb\_data\_home\_dir =

innodb\_data\_file\_path=/ibdata/ibdata1:50M;/ibdata/ibdata2:50M:autoextend

When you add a new file to the tablespace configuration, make sure that it does not exist. InnoDB will create and initialize the file when you restart the server.

**Increase the redo log size**

[mysqld]

**innodb\_log\_buffer\_size = 30M**

**Determining the Maximum Memory Allocation for InnoDB:**

On 32-bit GNU/Linux x86, be careful not to set memory usage too high. glibc may permit the process heap to grow over thread stacks, which crashes your server. It is a risk if the value of the following expression is close to or exceeds 2GB.

**If using just InnoDB, set innodb\_buffer\_pool\_size to 70% of \_available\_ RAM.**

mysql> show variables like 'innodb\_buffer\_pool\_size';

mysql> show variables like 'innodb\_log\_buffer\_size';

**innodb\_buffer\_pool\_size**

**+key\_buffer\_size**

**+max\_connections\*(sort\_buffer\_size+read\_buffer\_size+binlog\_cache\_size)**

**+max\_connections\*2MB**

**Turning Off InnoDB:**

If you do not want to use InnoDB tables, start the server with the --innodb=OFF or --skip-innodb option to disable the InnoDB storage engine. In this case, the server will not start if the default storage engine is set to InnoDB.

Steps:- vi /etc/mysql/my.cnf

[mysqld]

####Edited by Sarrath MySQL DBA [www.dbacentre.com](http://www.dbacentre.com/) ###

skip-innodb

default-storage-engine=MyISAM

:wq

Save and quit..

mysql restart your instance, once restarted very with below commands.

SHOW ENGINES;

SELECT TABLE\_NAME, ENGINE FROM information\_schema.TABLES

WHERE TABLE\_SCHEMA = 'dbacentre';

**Using Per-Table Tablespaces:**

You can store each InnoDB table and its indexes in its own file. This feature is called “multiple tablespaces” because in effect each table has its own tablespace.

To enable multiple tablespaces, start the server with the --innodb\_file\_per\_table option. For example, add a line to the [mysqld] section of my.cnf.

Mysql> show variables like 'innodb\_file\_per\_table';

SELECT FILE\_NAME,TABLESPACE\_NAME,TABLE\_SCHEMA FROM INFORMATION\_SCHEMA.FILES;

set global innodb\_file\_per\_table=off;

add this line to the [mysqld] section of my.cnf:

[mysqld]

innodb\_file\_per\_table=1

( or )

You can also set innodb\_file\_per\_table dynamically, while the server is running:

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

With multiple tablespaces enabled, InnoDB stores each newly created table into its own tbl\_name.ibd file in the database directory where the table belongs. This is similar to what the MyISAM storage engine does, but MyISAM divides the table into a tbl\_name.MYD data file and an tbl\_name.MYI index file.

For InnoDB, the data and the indexes are stored together in the .ibd file. The tbl\_name.frm file is still created as usual.

You cannot freely move .ibd files between database directories as you can with MyISAM table files. This is because the table definition that is stored in the InnoDB shared tablespace includes the database name, and because InnoDB must preserve the consistency of transaction IDs and log sequence numbers.

If you remove the innodb\_file\_per\_table line from my.cnf and restart the server, InnoDB creates tables inside the shared tablespace files again.

The -- innodb\_file\_ per\_table option affects only table creation, not access to existing tables. If you start the server with this option, new tables are created using .ibd files, but you can still access tables that exist in the shared tablespace. If you start the server without this option, new tables are created in the shared tablespace, but you can still access any tables that were created using multiple tablespaces.

**Note:** InnoDB always needs the shared tablespace because it puts its internal data dictionary and undo logs there.The .ibd files are not sufficient for InnoDB to operate.

To move an .ibd file and the associated table from one database to another, use a RENAME TABLE statement:

mysql> RENAME TABLE database1.tbl\_name TO database2.tbl\_name;

* **Dealing with InnoDB Initialization Problems:- (** Restrictions on InnoDB )

If InnoDB prints an operating system error during a file operation, usually the problem has one of the following causes:

* You did not create the InnoDB data file directory or the InnoDB log directory.
* mysqld does not have access rights to create files in those directories.
* mysqld cannot read the proper my.cnf or my.ini option file, and consequently does not see the options that you specified.
* The disk is full or a disk quota is exceeded.
* You have created a subdirectory whose name is equal to a data file that you specified, so the name cannot be used as a file name.
* There is a syntax error in the innodb\_data\_home\_dir or innodb\_data\_file\_path value.

If something goes wrong when InnoDB attempts to initialize its tablespace or its log files, delete all files created by InnoDB. This means all ibdata files and all ib\_logfile files. In case you have already created some InnoDB tables, delete the corresponding .frm files for these tables (and any .ibd files if you are using multiple tablespaces) from the MySQL database directories as well. Then you can try the InnoDB database creation again. It is best to start the MySQL server from a command prompt so that you see what is happening.

**InnoDB Startup Options and System Variables**

**innodb**

Enables the InnoDB storage engine, if the server was compiled with InnoDB support. **innodb-status-file**

Controls whether InnoDB creates a file named innodb\_status.<pid> in the MySQL data directory. If enabled, InnoDB periodically writes the output of SHOW ENGINE INNODB STATUS to this file.

By default, the file is not created. To create it, start mysqld with the --innodb-status-file=1 option. The file is deleted during normal shutdown.

**skip-innodb**

Disable the InnoDB storage engine.

**innodb\_additional\_mem\_**

--- Version 5.5

mysql> show variables like '%innodb\_additional\_mem%';

**innodb\_fast\_shutdown**

The InnoDB shutdown mode. By default, the value is 1, which causes a “fast” shutdown (the normal type of shutdown).

If the value is 0, InnoDB does a full purge and an insert buffer merge before a shutdown. These operations can take minutes, or even hours in extreme cases. If the value is 1, InnoDB skips these operations at shutdown.

If the value is 2, InnoDB will just flush its logs and then shut down cold, as if MySQL had crashed; no committed transaction will be lost, but crash recovery will be done at the next startup.

mysql> show variables like 'innodb\_fast\_shutdown';

**innodb\_lock\_wait\_timeout**

The timeout in seconds an InnoDB transaction may wait for a lock before being rolled back. The default is 50 seconds. A transaction that tries to access a row that is locked by another InnoDB transaction will hang for at most this many seconds before issuing the following error:

ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction

mysql> show variables like 'innodb\_lock\_wait\_timeout';

When a lock wait timeout occurs, the current statement is not executed. The current transaction is not rolled back. (To have the entire transaction roll back, start the server with the --innodb\_rollback\_on\_timeout option, available as of MySQL 5.1.15.

**innodb\_rollback\_on\_timeout**

In MySQL 5.1, InnoDB rolls back only the last statement on a transaction timeout by default. If -- innodb\_rollback\_ on\_timeout is specified, a transaction timeout causes InnoDB to abort and roll back the entire transaction. This variable was added in MySQL 5.1.15.

**innodb\_log\_buffer\_size**

The size in bytes of the buffer that InnoDB uses to write to the log files on disk. The default value is 1MB. Sensible values range from 1MB to 8MB. A large log buffer enables large transactions to run without a need to write the log to disk before the transactions commit. Thus, if you have big transactions, making the log buffer larger saves disk I/O.

mysql> show variables like 'innodb\_log\_buffer\_size';

if you want to increase or decrease the value, you can change in my.cnf file.

**innodb\_log\_file\_size**

The size in bytes of each log files in a log group. The combined size of log files must be less than 4GB. The default value is 5MB. Sensible values range from 1MB to 1/N-th of the size of the buffer pool, where N is the number of log files in the group. The larger the value, the less checkpoint flush activity is needed in the buffer pool, saving disk I/O. But larger log files also mean that recovery is slower in case of a crash.

mysql> show variables like 'innodb\_log\_file\_size';

**innodb\_log\_files\_in\_group**

The number of log files in the log group. InnoDB writes to the files in a circular fashion. The default (and recommended) value is 2.

mysql> show variables like 'innodb\_log\_files\_in\_group';

**innodb\_log\_group\_home\_dir**

The directory path to the InnoDB log files. If you do not specify any InnoDB log variables, the default is to create two 5MB files names ib\_logfile0 and ib\_logfile1 in the MySQL data directory.

mysql> show variables like 'innodb\_log\_group\_home\_dir';

**innodb\_open\_files**

This variable is relevant only if you use multiple tablespaces in InnoDB. It specifies the maximum number of

.ibd files that InnoDB can keep open at one time. The minimum value is 10. The default value is 300. This variable is available as of MySQL 4.1.1.

The file descriptors used for .ibd files are for InnoDB only. They are independent of those specified by the -- open-files-limit server option, and do not affect the operation of the table cache.

**innodb-safe-binlog**

If this option is given, then after a crash recovery by InnoDB, mysqld truncates the binary log after the last not-rolled-back transaction in the log. The option also causes InnoDB to print an error if the binary log is smaller or shorter than it should be.

**foreign\_key\_checks**

If set to 1 (the default), foreign key constraints for InnoDB tables are checked. If set to 0, they are ignored. Disabling foreign key checking can be useful for reloading InnoDB tables in an order different from that required by their parent/child relationships.

Setting foreign\_key\_ checks to 0 also affects data definition statements: DROP SCHEMA drops a schema even if it contains tables that have foreign keys that are referred to by tables outside the schema, and DROP TABLE drops tables that have foreign keys that are referred to by other tables.

**Note:** Setting foreign\_key\_checks to 1 does not trigger a scan of the existing table data. Therefore, rows added tothe table while foreign\_key\_checks = 0 will not be verified for consistency.

mysql> **show variables like 'foreign\_key\_checks';**

**innodb\_commit\_concurrency**

The number of threads that can commit at the same time. A value of 0 (the default) permits any number of transactions to commit simultaneously.

As of MySQL 5.1.36, the value of innodb\_commit\_concurrency cannot be changed at runtime from zero to nonzero or vice versa. The value can still be changed from one nonzero value to another.

**innodb\_force\_recovery**

The crash recovery mode. Possible values are from 0 to 6. This variable should be set greater than 0 only in an emergency situation when you want to dump your tables from a corrupt database! As a safety measure, InnoDB prevents any changes to its data when this variable is greater than 0

**innodb\_stats\_on\_metadata**

When this variable is enabled (which is the default, as before the variable was created), InnoDB updates statistics during metadata statements such as SHOW TABLE STATUS or SHOW INDEX, or when accessing the INFORMATION\_SCHEMA tables TABLES or STATISTICS. (These updates are similar to what happens for ANALYZE TABLE.) When disabled, InnoDB does not updates statistics during these operations. Disabling this variable can improve access speed for schemas that have a large number of tables or indexes. It can also improve the stability of execution plans for queries that involve InnoDB tables.

**Creating and Using InnoDB Tables:**

To create an InnoDB table, specify an ENGINE = InnoDB option in the CREATE TABLE statement:

mysql> CREATE TABLE customers (a INT, b CHAR (20), INDEX (a)) ENGINE=InnoDB;

The statement creates a table and an index on column a in the InnoDB tablespace that consists of the data files that you specified in my.cnf. In addition, MySQL creates a file customers.frm in the test directory under the MySQL database directory. Internally, InnoDB adds an entry for the table to its own data dictionary. The entry includes the database name. For example, if test is the database in which the customers table is created, the entry is for 'test/customers'. This means you can create a table of the same name customers in some other database, and the table names do not collide inside InnoDB.

You can query the amount of free space in the InnoDB tablespace by issuing a SHOW TABLE STATUS statement for any InnoDB table. The amount of free space in the tablespace appears in the Data\_ free section in the output of **SHOW TABLE STATUS** (or the Comment section prior to MySQL 5.1.24). For example:

mysql> SHOW TABLE STATUS FROM test LIKE 'customers'

**How to Use Transactions in InnoDB with Different APIs:**

By default, each client that connects to the MySQL server begins with autocommit mode enabled, which automatically commits every SQL statement as you execute it. To use multiple-statement transactions, you can switch autocommit off with the SQL statement SET autocommit = 0 and end each transaction with either COMMIT or ROLLBACK. If you want to leave autocommit on, you can begin your transactions within START TRANSACTION and end them with COMMIT or ROLLBACK. The following example shows two transactions. The first is committed; the second is rolled back.

shell> mysql test

mysql> CREATE TABLE customer (a INT, b CHAR (20), INDEX (a)) ->

ENGINE=InnoDB;

Query OK, 0 rows affected (0.00 sec)

mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO customer VALUES (10, 'Dbaclass');

Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;

Query OK, 0 rows affected (0.00 sec)

mysql> SET autocommit=0;

Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO customer VALUES (15, 'John');

Query OK, 1 row affected (0.00 sec)

mysql> ROLLBACK;

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT \* FROM customer;

| a | b | +--- | 10 | Dbaclass | +-----+

1 row in set (0.00 sec)

**Converting Tables from Other Storage Engines to InnoDB:**

To convert a non-InnoDB table to use InnoDB use ALTER TABLE:

mysql> ALTER TABLE t1 ENGINE=InnoDB;

**Note:** Do not convert MySQL system tables in the mysql database (such as user or host) to the InnoDB type. Thisis an unsupported operation. The system tables must always be of the MyISAM type.

**InnoDB Error Codes:**

1005 (ER\_CANT\_CREATE\_TABLE)

Cannot create table. If the error message refers to error 150, table creation failed because a foreign key constraint was not correctly formed. If the error message refers to error –1, table creation probably failed because the table includes a column name that matched the name of an internal InnoDB table.

1016 (ER\_CANT\_OPEN\_FILE)

Cannot find the InnoDB table from the InnoDB data files, although the .frm file for the table exists.

1114 (ER\_RECORD\_FILE\_FULL)

InnoDB has run out of free space in the tablespace. Reconfigure the tablespace to add a new data

file.

1205 (ER\_LOCK\_WAIT\_TIMEOUT)

Lock wait timeout expired. Transaction was rolled back.

1216 (ER\_NO\_REFERENCED\_ROW)

You are trying to add a row but there is no parent row, and a foreign key constraint fails. Add the parent row first.

1217 (ER\_ROW\_IS\_REFERENCED)

You are trying to delete a parent row that has children, and a foreign key constraint fails. Delete the children first.

Exercises:-

1. You cannot remove a data file from the tablespace. To decrease the size of your tablespace, use this follow the steps:
2. Use mysqldump to dump all your InnoDB tables.
3. Stop the server.
4. Remove all the existing tablespace files, including the ibdata and ib\_log files. If you want to keep a backup copy of the information, then copy all the ib\* files to another location before the removing the files in your MySQL installation.
5. Remove any .frm files for InnoDB tables.
6. Configure a new tablespace.
7. Restart the server.
8. Import the dump files.
9. If you want to change the number or the size of your InnoDB log files, use the following instructions. The procedure to use depends on the value of innodb\_fast\_shutdown:

If innodb\_fast\_shutdown is not set to 2: Stop the MySQL server and make sure that it shuts down without errors (to ensure that there is no information for outstanding transactions in the log). Copy the old log files into a safe place in case something went wrong during the shutdown and you need them to recover the tablespace. Delete the old log files from the log file directory, edit my.cnf to change the log file configuration, and start the MySQL server again. mysqld sees that no InnoDB log files exist at startup and creates new ones.

If innodb\_fast\_shutdown is set to 2: Set innodb\_fast\_shutdown to 1:

mysql> SET GLOBAL innodb\_fast\_shutdown = 1;

**InnoDB Performance Tuning and Troubleshooting**

− *innodb\_buffer\_pool\_size* specifies the size of the buffer pool. If your buffer pool is small and you have sufficient

memory, making the pool larger can improve performance by reducing the amount of disk I/O needed as queries access InnoDB tables.

− Beware of big rollbacks of mass inserts: InnoDB uses the insert buffer to save disk I/O in inserts, but no such mechanism is used in a corresponding rollback. A disk-bound rollback can take 30 times as long to perform as the corresponding insert. Killing the database process does not help because the rollback starts again on server startup. The only way to get rid of a runaway rollback is to increase the buffer pool so that the rollback becomes CPU-bound and runs fast, or to use a special procedure.

− Beware also of other big disk-bound operations.

Use DROP TABLE and CREATE TABLE to empty a table, not DELETE FROM tbl\_name.

− Place tablespaces in different locations.

− Make your log files big, even as big as the buffer pool. When InnoDB has written the log files full, it must write the modified contents of the buffer pool to disk in a checkpoint. Small log files cause many unnecessary disk writes. The disadvantage of big log files is that the recovery time is longer.

− When importing data into InnoDB, make sure that MySQL does not have autocommit mode enabled because that requires a log flush to disk for every insert. To disable autocommit during your import operation, surround it with

SET autocommit and COMMIT statements:

SET autocommit=0;

... SQL import statements ...

COMMIT;

− If you have UNIQUE constraints on secondary keys, you can speed up table imports by temporarily turning off the uniqueness checks during the import session:

SET unique\_checks=0;

... SQL import statements ...

SET unique\_checks=1;

− If you have FOREIGN KEY constraints in your tables, you can speed up table imports by turning the foreign key

checks off for the duration of the import session:

SET foreign\_key\_checks=0;

... SQL import statements ...

SET foreign\_key\_checks=1;

− Use the multiple-row INSERT syntax to reduce communication overhead between the client and the server

if you need to insert many rows:

mysql> INSERT INTO yourtable VALUES (1,2), (5,5), ...;

This tip is valid for inserts into any table, not just InnoDB tables.