PROGRAMMING LAB-I PRACTICALS

Sr. No.	Name Of Experiment
1	DBMS using connections(Client-Data sever, two tier) Oracle/MySQL (ODBC/JDBC), SQL prompt to create data base tables insert, update data values, delete table, use table, select queries with/without where clause. ,demonstrate use of stored procedure / function (create procedure at the data side and make use of it on the client side)
2	DBMS using connections (Client-application server-Data sever, three tiers) Oracle/MySQL (ODBC/JDBC), SQL Joints, prompt.
3	Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View , Index using Client-Data sever(two tier)
4	Write a program in Python/C++ to read display the i-node information for a given text file, image file.
5	Write an IPC program using pipe. Process A accepts a character string and Process B inverses the string. Pipe is used to establish communication between A and B processes using Python or C++.
6	Use Python for Socket Programming to connect two or more PCs to share a text file.
1	Design at least 10 SQL queries for suitable database application using SQL DML statements: Insert, Select, Update, Delete Clauses using distinct, count, aggregation on Client-Data sever(three tier)
2	Implement database with suitable example using MongoDB and implement all basic operations and administration commands using two tiers architecture.
3	Write a program in Python/C++ to test that computer is booted with Legacy Boot ROMBIOS or UEFI.
4	Write a program in C++ to create a RAMDRIVE and associate an acyclic directory structure to it. Use this RAMDRIVE to store input, out files to run a calculator program.
5	Write a Python/Java/C+ program to verify the operating system name and version of Mobile devices.
6	Aggregation and indexing with suitable example using MongoDB
7	Map reduces operation with suitable example using MongoDB.
8	Indexing and querying with MongoDB using suitable example.
9	Connectivity with MongoDB using any Java application.

Subject Teachers:

Mr Mayur S Patil
Mr A H More
Mr Mahesh A Pavaskar

Head of Department

Prof. Uma Nagaraj

Assignments Group - A

Assignment No 1

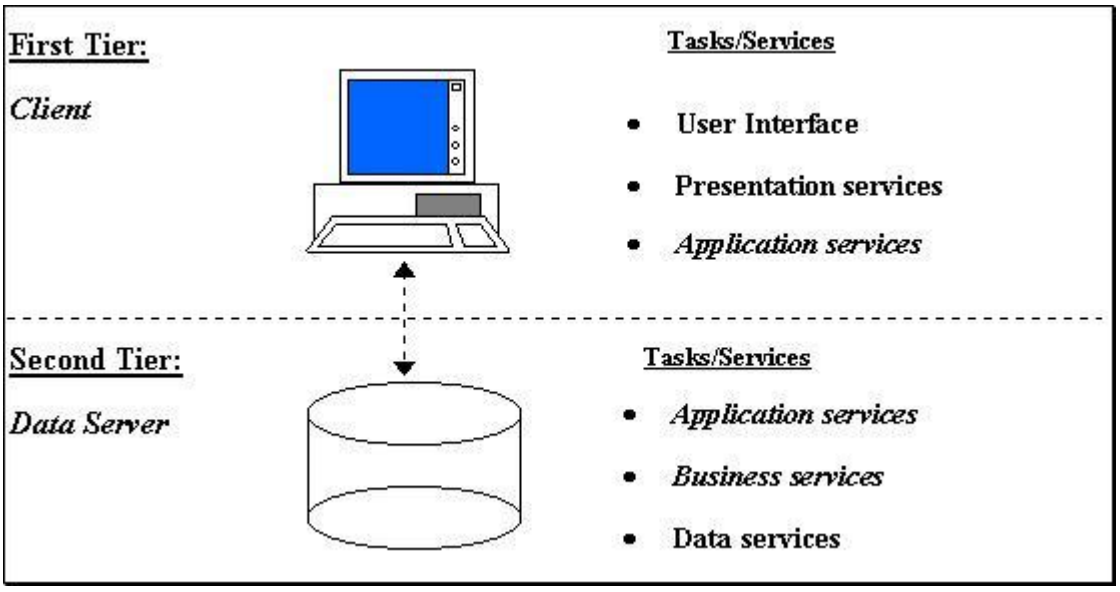
Title: DBMS using connections(Client-Data sever, two tier) Oracle/MySQL (ODBC/JDBC), SQL prompt to create data base tables insert, update data values, delete table, use table, select queries with/without where clause. , demonstrate use of stored procedure / function (create procedure at the data side and make use of it on the client side)

Requirements:

1. Computer System with Linux/Open Source Operating System.
2. Mysql Server
3. JDK
4. JDBC Connector
5. Eclipse

Theory:

Two Tier Architecture:



This assignment provides a tutorial introduction to MySQL by showing how to use the mysql client program to create and use a simple database. mysql (sometimes referred to as the \terminal monitor" or just \monitor") is an interactive program that allows you to connect to a MySQL server, run queries, and view the results. mysql may also be used in batch mode: you

place your queries in a _le beforehand, then tell mysql to execute the contents of the _le. Both ways of using mysql are covered here.

To see a list of options provided by mysql, invoke it with the --help option:
shell> mysql --help

This assignment assumes that mysql is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator.

(If you are the administrator, you will need to consult other sections of this manual.)

Required Steps using JDBC:

T here are following steps required to create a new Database using JDBC application:

Import the pac kag es . Requires that you include the packag es containing the JDBC classes needed for database prog ramming . Most often, using *import java.sql.** will suffice.

Register the JDBC driver . Requires that you initialize a driver so you can open a communications channel with the database.

Open a connec tion . Requires using the *DriverManager.getConnection()* method to create a Connection object, which represents a physical connection with datbase server.

T o create a new database, you need not to g ive any database name while preparing database URL as mentioned in the below example.

Execute a query . Requires using an object of type Statement for building and submitting an SQL statement to the database.

Clean up the environment . Requires explicitly closing all database resources versus relying on the JVM's garbag e collection.

Conclusion:

Thus, we implemented DBMS queries using connections(Client-Data sever, two tier) Oracle/MySQL (ODBC/JDBC), SQLprompt to create data base tables insert, update data values, delete table, use table, select queries with/without where clause.

Assignment No 2

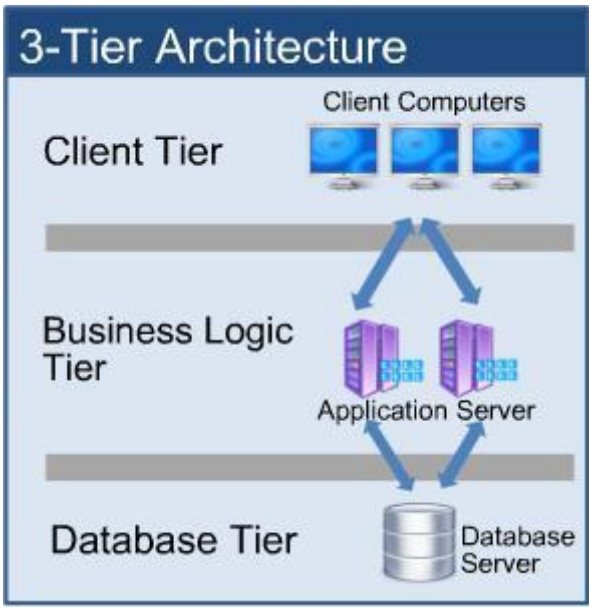
Title: DBMS using connections (Client-application server-Data sever, three tiers) Oracle/MySQL (ODBC/JDBC), SQL Joints, prompt.

Requirements:

1. Computer System with Linux/Open Source Operating System.
2. Mysql Server
3. JDK
4. JDBC Connector
5. Eclipse
6. Web server(Apache Tomcat 6.0) OR Glassfish
7. HTML/JSP

Theory:

Three Tier Architecture:



Joins:

The ability of relational „join" operator is an important feature of relational systems. A join makes it possible to select data from more than table by means of a single statement. This joining of tables may be done in a many ways.

The purpose of a join concept is to combine data spread across tables. A join is actually performed by the "where" clause which combines specified rows of tables.

Syntax: select columns from table1, table2 where logical expression;

Types of Joins

1. Simple Join
2. Self Join
3. Outer Join
4. Inner Join

Types of JOIN:

- **Inner Join:** Also known as equi join.

  Statements generally compare two columns from two columns with the equivalence operator =. This type of join can be used in situations where selecting only those rows that have values in common in the columns specified in the ON clause, is required.

  Syntax:

  SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1> INNER JOIN <tablename2>ON <tablename1>.<columnname> = <tablename2>.<columnname> WHERE <condition> ORDER BY <columnname1>;

  SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1>, <tablename2> WHERE <tablename1>.<columnname> = <tablename2>.<columnname> AND <condition> ORDER BY <columnname1>;

- **Outer Join**

  Outer joins are similar to inner joins, but give a little bit more flexibility when selecting data from related tables. This type of joins can be used in situations where it is desired, to select all rows from the table on left (or right, or both) regardless of whether the other table has values in common & (usually) enter NULL where data is missing.

  Syntax:

  SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1> LEFT OUTER JOIN <tablename2>ON <tablename1>.<columnname> = <tablename2>.<columnname> WHERE <condition> ORDER BY <columnname1>;

- **Right outer Join**

  List the employee details with contact details (if any using right outer join. Since the RIGHT JOIN returns all the rows from the second table even if there are no matches in the first table.

  Syntax:

  (ANSI style)

  SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1> RIGHT OUTER JOIN <tablename2>ON <tablename1>.<columnname> = <tablename2>.<columnname> WHERE <condition> ORDER BY <columnname1>;

  (Theta style)

  SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1>, <tablename2> WHERE <tablename1>.<columnname> = <tablename2>.<columnname> AND <condition> ORDER BY <columnname1>;

- **Cross Join**

  A cross join returns what known as a Cartesian product. This means that the join combines every row from the left table with every row in the right table. As can be imagined, sometimes this join produces a mess, but under the right circumstances, it can be very useful. This type of join can be used in situation where it is desired, to select all possible combinations of rows & columns from both tables. The kind of join is usually not preferred as it may run for a very long time & produce a huge result set that may not be useful.

  Syntax:

  SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1>, <tablename2>;

- **Self Join**

  In some situation, it is necessary to join to itself, as though joining 2 separate tables. This is referred to as self join

  Syntax:

  SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1> INNER JOIN <tablename1>ON <tablename1>.<columnname> = <tablename2>.<columnname> ;

  (Theta style)

SELECT<columnname1>, <columnname2> <columnNameN> FROM <tablename1>, <tablename2> WHERE <tablename1>.<columnname> = <tablename1>.<columnname> ;

- Aggregate function: - Count, Average, Sum, Min, Max
- etc. Set-valued sub queries:- in, any, all
- Clause: - Group by clause, Distinct clause, Ordered by clause.

In combination with the clause with check option any update or insertion of a row into the view is rejected if the new/modified row does not meet the view definition, i.e., these rows would not be selected based on the select statement. A with check option can be named using the constraint clause.

A view can be deleted using the command delete <view-name>.

Conclusion:-

Thus, we have implemented DBMS queries using connections(Client-application server-Data sever, three tier) Oracle/MySQL (ODBC/JDBC), SQL Joints, prompt.

Assignment No 3

Title: Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, and Index using Client-Data sever (two tiers)

Requirements:

1. Computer System with Linux/Open Source Operating System.
2. Mysql Server
3. JDK
4. JDBC Connector
5. Eclipse

Theory:

Data Definition Language

DDL(Data Definition Language) statements are used to create, delete, or change the objects of a database.
Typically a database administrator is responsible for using DDL statements or production databases in a
large database system. The commands used are:

- Create - It is used to create a table.
- Alter - This command is used to add a new column, modify the existing column definition and to
include or drop integrity constraint.
- Drop - It will delete the table structure provided the table shousld be empty.
- Truncate - If there is no further use of records stored in a table and the structure has to be retained,
then the records alone can be deleted.
- Describe - This is used to view the structure of the table.

SQL: Structured Query Language

SQL: (pronounced SEQUEL) is the standard language to access relational databases. SQL is an abbreviation for Structured Query Language.

1. **Data types**
   1.1. **Number (p, s):** Number having precision p and scale s.
   1.2. **Char (size):** Fixed length character data of length size bytes. This should be used for fixed length data. Such as codes A100, B102...

1.3. **Varchar2 (size):** Variable length character string having maximum length *size* bytes. You must specify size
1.4. **Long:** Character data of variable length (A bigger version the VARCHAR2 data type)
1.5. **Date:** Valid date range
1.6. **Int:** A finite subset of integers. The full form, integer, is equivalent.
1.7. **Rear, Double Precision:** Floating-point & double-precision floating point numbers.
1.8. **Float (n):** A floating point number, with precision of at least n digits.

Data Definition in SQL

Creating Tables

Syntax:-

Create table<table name>
(colume_name 1 datatype size(),
colume_name 2 datatype size(),
.
.
colume_name n datatype size());

Constraints

Both the Create table & Alter Table SQL can be used to write SQL sentences that attach constraints. Basically constraints are of three types

1) Domain

- **Not Null:** Not null constraint can be applied at column level only. We can define these constraints at the time of table creation.

Syntax:
CREATE TABLE <tableName>
(<ColumnName> datatype(size) NOT NULL,
<ColumnName> datatype(size),….
);

After the table creation

ALTER TABLE <tableName>
Modify (<ColumnName> datatype(size) NOT NULL);

Primary Key: A primary key is one or more column(s) in a table used to uniquely identify each row in the table. A table can have only one primary key. Cannot be left blank Data must be UNIQUE. Not allows null values**.** Not allows duplicate values. Unique index is created automatically if there is a primary key.

Primary key constraint defined at column level

Syntax:
CREATE TABLE <TableName>

(<ColumnName1> <DataType>(<Size>)PRIMARY KEY,<columnname2 <datatype(<size>),…..);

Primary key constraint defined at Table level Syntax:

CREATE TABLE <TableName>
(<ColumnName1> <DataType> (<Size>) ,…,
PRIMARY KEY(<ColumnName1> <ColumnName2>));

Foreign key: Foreign key represents relationships between tables. A foreign key is a column (or group of columns) whose values are derived from primary key or unique key of some other table. Foreign key constraint defined at column level

Syntax:

<columnName> <DataType> (<size>) REFERENCES
<TableName>[(<ColumnName>)] [ON DELETE CASCADE]

After the table creation

ALTER TABLE <child_tablename> ADD CONSTRAINT <constraint_name> FOREIGN KEY (<columnname in child_table>) REFERENCES <parent table name>;

FOREIGN KEY constraint at table level

FOREIGN KEY constraint defined with ON DELETE CASCADE

FOREIGN KEY (<ColumnName>[,<columnname>]) REFERENCES <TableName> [(<ColumnName>, <ColumnName>) ON DELETE CASCADE

Foreign Key constraint defined with ON Delete Set Null

FOREIGN KEY(<ColumnName>[,<columnname>]) REFERENCES <TableName> [(<ColumnName>, <ColumnName>) ON DELETE SET NULL

Restrictions for creating a table:

- Table names and column names must begin with a letter. Table
- names and column names can be 1 to 30 characters long.
- Table names must contain only the characters A-Z,a-z,0-9,underscore_,$ and #
- Table name should not be same as the name of another database object.
- Table name must not be an ORACLE reserved word.
- Column names should not be duplicate within a table definition.

3.6 Alteration of Table:-

Syntax:-

Case1:-

Alter table <table_name>
Add(colume_name 1 datatype size(),
colume_name 2 datatype size(),
.
.
colume_name n datatype size());

Case2:-
Alter table <table_name>
Modify (colume_name 1 datatype
size(), colume_name 2 datatype size(),
.
.
colume_name n datatype size());

Dropping a column from a table

Syntax:
ALTER TABLE <Tablename> DROP COLUMN <ColumnName ;

Drop table command
Syntax:
Drop table <table_name>

Drop table command removes the definitions of an oracle table. When you drop a table, the database loses all the data in the table and all the indexes associated with it.
E.g. drop table student;

Truncate table command
Syntax:-
Truncate table<table_name>

The truncate table statement is used to remove all rows from a table and to release the storage space used by the table.

E.g. truncate table student;

Rename table command
Syntax:-
Rename<oldtable_name> to<newtable_name>

Rename statement is used to rename a table, view, sequence or synonym.

E.g. rename student to stud;

3 Database objects:-

Views: - View is a logical representation of subsets of data from one or more tables. A view takes the output of a query and treats it as a table therefore view can be called as stored query or a virtual table. The tables upon which a view is based are called base tables. In Oracle the SQL command to create a view (virtual table) has the form.

Syntax:
create [or replace] view <view-name> [(<column(s)>)] as
<select-statement> [with check option [constraint <name>]];

The optional clause or replace re-creates the view if it already exists. <column(s)> names the columns of the view. If <column(s)> is not specified in the view definition, the columns of the view get the same names as the attributes listed in the select statement (if possible).

A view can be used in the same way as a table, that is, rows can be retrieved from a view(also respective rows are not physically stored, but derived on basis of the select statement in the view Definition), or rows can even be modified. A view is evaluated again each time it is accessed. In Oracle SQL no insert, update, or delete medications on views are allowed that use one of the following constructs in the View Definition:

- Joins
- Aggregate function
- Set-valued sub queries
- Group by clause or distinct clause, Ordered by.

A view can be deleted using the command delete <view-name>.
Removing a view

Syntax:- Drop view <view_name>
E.g. Drop view stud

Index: - An index is a schema object that can speed up retrieval of rows by using pointer. An index provides direct & fast access to rows in a table. Index can be created explicitly or automatically.

Automatically: - A unique index is created automatically when you define a primary key or unique key constraint in a table definition.

Manually: - users can create non unique indexes or columns to speed up access time to the rows.

Syntax:
Create index<index_name>
On table (column[, column]…);

ic.table_name="emp";

Removing an Index

Syntax:-
Drop index <index_name>;

eg. Drop index emp_name_idx;

Note:
1) We cannot modify indexes.
2) To change an index, we must drop it and the re-create it.

Conclusion:- Thus we have design and developed SQL DDL statements which demonstrate the use of SQL objects

such as Table, View , Index using Client-Data sever(two tier).

Assignment No 4

Title: Write a program in Python/C++ to read display the i-node information for a given text file, image file.

Objective:

- To develop Operating Systems programming and administrative skills

Requirements:

1. **Hardware:-** Minimum single core machine

2. **Software: -** Fedora 17 or above [recommended], Python 2.7.x [preferably].

Theory:

The essential part of the operating system functionality as well as basics of its lies in the data transfer which has dealt with basic unit of higher level operations i.e. files. As we know files are the basic entities for transferring the data in and out of the system. For that we have to understand internal representation of files. It indicates how kernel identifies particular aspects of files in operating system. Here comes in picture inode i.e. index node. As the name indicates, it is having whole information about the file except physical file name. Inode has two locations of its copies named as on-disk copy of inode and in-core copy of inode. So, the attributes get covered of inode are explained as follows:

Inodes exist in a static form on disk, and the kernel reads them into an in-core inode to manipulate them. (On-)Disk inodes consist of the following fields:

•. File owner identifier. Ownership is divided between an individual owner and a "group" owner and defines the set of users who have access rights to a file. The superuser has access rights to all files in the system.

• File type. Files may be of type regular, directory, character or block special, or FIFO (pipes).

• File access permissions. The system protects files a/c to three classes: the owner and the group owner of the file and other users; each class his access rights to read, write and execute the file which can be set individually. Because directories cannot be executed, execution permission for a directory gives the right to search the directory or a filename.

• File access times. Giving the time the file was last modified, when it was last accessed and when the inode was last modified.

• Number of links to the file, representing the number of names the file has in the directory hierarchy.

• Table of contents for the disk addresses of data in a file. Although users treat the data in a file as a logical stream of bytes, the kernel saves the data in discontiguous disk blocks. The inode identifies the disk block that contains the file's data.

• File size. Data in a file is addressable by the number of bytes from the beginning of the file, starting from byte offset 0, and the file size is 1 greater than the highest byte offset of data in the file.

The in-core copy of the inode contains the following fields in addition to the fields of the disk inode:

• The status of the in-core inode, indicating whether

- The inode is locked,
- A process is waiting for the inode to become unlocked,
- The in-core representation of the inode differs from the disk copy as a result of a change to the data in the inode,
- The in-core representation of the file differs from the disk copy as a result of a change to the file data,
- File is a mount point.

• The logical device number of the file system that contains the file.

• The inode number. Since inodes are stored in a linear array on disk the kernel identifies the number of a disk inode by its position in the array.

• Pointers to other in-core inodes. The kernel links inodes on hash queues and on a free list in the same way that it links buffers on buffer hash queues and on the buffer free list. A hash queue is identified according to the inode's logs device number and inode number. The kernel can contain at most one in-core copy of a disk inode, but inodes can be simultaneously on a hash queue and on the free list.

• A reference count, indicating the number of instances of the file that are active (such as when opened)

The following description presents the ideology how to write inode-specific information in python. The python 2.x series is having by default ***os*** package which includes following attributes:

os.stat(*path*)

Perform the equivalent of a stat() system call on the given path. The return value is an object whose attributes correspond to the members of the stat structure, namely:

st_mode - protection bits,

st_ino - inode number,

st_dev - device,

st_nlink - number of hard links,

st_uid - user id of owner,

st_gid - group id of owner,

st_size - size of file, in bytes,

st_atime - time of most recent access,

st_mtime - time of most recent content modification,

st_ctime - platform dependent; time of most recent metadata change on Unix, or the time of creation on Windows)

On some Unix systems (such as Linux), the following attributes may also be available:

st_blocks - number of 512-byte blocks allocated for file

st_blksize - filesystem blocksize for efficient file system I/O

st_rdev - type of device if an inode device

st_flags - user defined flags for file

Conclusion: - We have learned about how to get inode detailed about a file in the operating system using Python.

Assignment No 5

Title: Write an IPC program using pipe. Process A accepts a character string and Process B inverses the string. Pipe is used to establish communication between A and B processes using Python or C++.

Objective:
To develop Operating System programming skills.

Requirements:

1. **Hardware:-** Server and Client.

2. **Software: -** Fedora 17 or above, Python 2.7.x, g++ & gcc 4.6.x and above.

Theory:

This program is used to perform IPC i.e. Inter process Communication. The inter process communication mainly divides into three types:

a. Messaging
b. Shared memory
c. Semaphores

When process spawn *threads*—tasks that run in parallel within the program—they can naturally communicate by changing and inspecting names and objects in shared global memory. This includes both accessible variables and attributes. The care must be taken to use locks to synchronize access to shared items that can be updated concurrently.

Things aren't quite as simple when scripts start child processes and independent programs that do not share memory in general. If we limit the kinds of communications that can happen between programs, many options are available, most of which we've already seen in this and the prior chapters. For example, the following simple mechanisms can all be interpreted as cross-program communication devices:

- Simple files
- Command-line arguments
- Program exit status codes
- Shell environment variables
- Standard stream redirections

Beyond this set, there are other tools in the Python library for performing Inter-Process Communication (IPC). This includes sockets, shared memory, signals, anonymous and named pipes, and more. Some vary in portability, and all vary in complexity and utility. For instance:

- *Signals* allow programs to send simple notification events to other programs.
- *Anonymous pipes* allow threads and related processes that share file descriptors to pass data, but generally rely on the Unix-like forking model for processes, which is not universally portable.
- *Named pipes* are mapped to the system's filesystem—they allow completely unrelated programs to converse, but are not available in Python on all platforms.
- *Sockets* map to system-wide port numbers—they similarly let us transfer data between arbitrary programs running on the same computer, but also between programs located on remote networked machines, and offer a more portable option.

While some of these can be used as communication devices by threads, too, their full power becomes more evident when leveraged by separate processes which do not share memory at large.

Pipes, a cross-program communication device, are implemented by your operating system. Pipes are unidirectional channels that work something like a shared memory buffer, but with an interface resembling a simple file on each of two ends. In typical use, one program writes data on one end of the pipe, and another reads that data on the other end. Each program sees only its end of the pipes and processes it using normal Python file calls.

Pipes are much more within the operating system, though. For instance, calls to read a pipe will normally *block* the caller until data becomes available (i.e., is sent by the program on the other end) instead of returning an end-of-file indicator. Moreover, read calls on a pipe always return

the oldest data written to the pipe, resulting in a *first-in-first-out* model—the first data written is the first to be read. Because of such properties, pipes are also a way to synchronize the execution of independent programs.

Pipes come in two flavors—*anonymous* and *named*. Named pipes (often called fifos) are represented by a file on your computer. Because named pipes are really external files, the communicating processes need not be related at all; in fact, they can be independently started programs.

By contrast, anonymous pipes exist only within processes and are typically used in conjunction with process *forks* as a way to link parent and spawned child processes within an application. Parent and child converse over shared pipe file descriptors, which are inherited by spawned processes. Because threads run in the same process and share all global memory in general, anonymous pipes apply to them as well.

A syscall fork to makes a copy of the calling process. After forking, the original parent process and its child copy speak through the two ends of a pipe created prior to the fork. The pipe call returns a tuple of two *file descriptors*—the low-level file identifiers representing the input and output sides of the pipe. Because forked child processes get *copies* of their parents' file descriptors, writing to the pipe's output descriptor in the child sends data back to the parent on the pipe created before the child was spawned.

Notice how the parent received a bytes string through the pipe. Raw pipes normally deal in binary byte strings when their descriptors are used directly this way with the descriptor-based file tools. That's why we also have to manually encode to bytes when writing in the child - the string formatting operation is not available on bytes. As the next section shows, it's also possible to wrap a pipe descriptor in a text-mode file object but that object simply performs encoding and decoding automatically on transfers; it's still bytes in the pipe.

Conclusion: We have learned about how the interprocess communication works on one machines using pipe () system call in C++.

Assignment No 6

Title: Use Python for Socket Programming to connect two or more PCs to share a text file.

Objective:

- To develop Operating Systems administration skill with multi-core programming

Requirements:

1. **Hardware:-** Two machines
2. **Software: -** Fedora 17 and above, Python 2.7.x.

Theory:

In this case, we are dealing with socket based communication which includes communicating with processes which are dealing with following system calls:
The socket mechanism contains several system calls. The socket syscall establishes the end point of a communications link.

sd = socket (format, type, protocol);

Format specifies the communications domain (the UNIX system domain or the Internet domain), type indicates the type of communication over the socket (virtual circait or datagram), and protocol indicates a particular protocol to control the communication. Processes use the socket descriptor sd in other syscalls. The /close system call closes sockets.

The bind system call associates a name with the socket descriptor:

bind (sd, address, length) ;

sd is the socket descriptor and address points to a structure that specifies an identifier specific to the communications domain and protocol specified in the socket system call. Length is the length of the address structure; without this parameter, the kernel would

not know how long the address is because it can vary across domains and protocols. For example, an address in the UNIX system domain is a file name. Server processes bind addresses to sockets and "advertise" their names to identify themselves to client processes.

The connect system call requests that the kernel make a connection to an existing socket:

connect(sd, address, length);

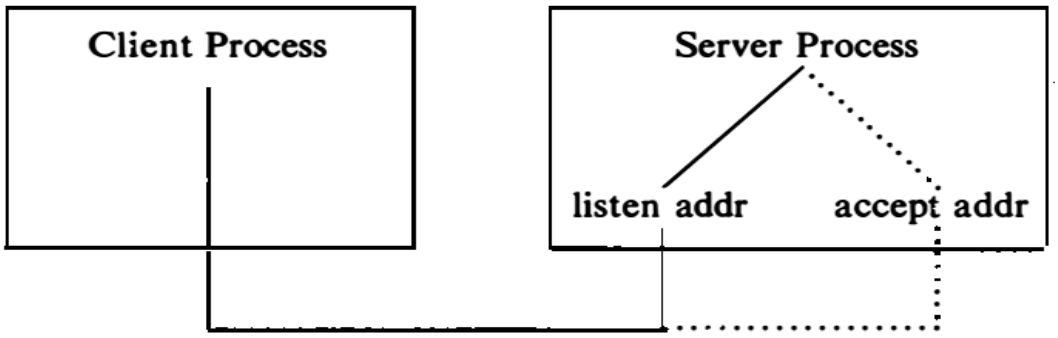
The semantics of the parameters are the same as for bind , but address is the address of the target socket that will form the other end of the communications line. Both sockets must use the same communications domain and protocol and the kernel arranges that the communications links are set up correctly. If type of the socket is a datagram, the connect call informs the kernel of the address to be used on subsequent send calls over the socket; no connections are made at the time of the call.

When a server process arranges to accept connections over a virtual circuit, the kernel must queue incoming requests until it can service them. The listen system call specifies the maximum queue length:
listen (sd, qlength}
where sd is the socket descriptor and qlength is the maximum number of outstanding requests.
The accept call receives incoming requests for a connection to a server process:

nsd = accept (sd, address, addrlen) ;

where sd is the socket descriptor, address points to a .. user data array that the kernel fills with the return address of the connecting client, and addrlen indicates the size of the user array. When accept returns, the kernel overwrites the contents of addrlen with a number that indicates the amount of space taken up by the address. Accept returns a new socket descriptor nsd , different from the socket descriptor sd .

A server can continue listening to the advertised socket while communicating with a client process over a separate communications channel



A Server accepting a call

Conclusion: We have learned about how machines could communicate using the socket mechanisms in Python.

Assignments Group B

Group B

Assignment No 1

Title: Design at least 10 SQL queries for suitable database application using SQL DML statements: Insert, Select, Update, Delete Clauses using distinct, count, aggregation on Client-Data sever (three tiers)

Requirements:

1. Computer System with Linux/Open Source Operating System.
2. Mysql Server
3. JDK
4. JDBC Connector
5. Eclipse
6. Web Server (Apache Tomcat 6.0)

Theory:

1. **Insert Command or Insertions:**
   The simplest way to insert a tuple into a table is to use the insert statement

   Syntax:-
   insert into <table> [(<column i, . . . , column j>)] values (<value i, . . . , value j>);

   For each of the listed columns, a corresponding (matching) value must be specified. Thus an insertion does not necessarily have to follow the order of the attributes as specified in the create table statement. If a column is omitted, the value null is inserted instead. If no column list is given, however, for each column as defined in the create table statement a value must be given.

   Syntax:-
   insert into <table> [(<column i, . . . , column j>)] <query> Case1:- Inserting values for selected columns

   Syntax:-
   Insert into student (name,roll,class,branch,enroll_no) values („abc‟,12,‟BE‟,‟computer‟,213123556);

   Case2:- Inserting values for all columns

Syntax:-
Insert into student values („abc‟,12,‟BE‟,‟computer‟,213123556);

Case3:- Inserting more than one record

Syntax:-
Insert into student values („&name‟, roll,‟class‟,‟&branch‟,enroll_no)

2. Selection of Tuple:

We can select (some) attributes of all tuples from a table. If one is interested in tuples that satisfy certain conditions, the where clause is used. In a where clause simple conditions based on comparison operators can be combined using the logical connectives and, or, and not to form complex conditions. Conditions may also include pattern matching operations and even sub queries.

- Select command :- It is used to retrieve the information stored in the database

Syntax:-
Select [distinct] <column(s)>
From <table>
[where <condition>]
[order by <column(s) [asc|desc]>]

OR

Select column_name1…..,column_name n from <table_name>

OR

Select * from <table_name>

- Selecting distinct rows: To prevent the selection of duplicate rows, we include distinct clause in the select Command. For example,

Syntax:-
select distinct name from student

- Filtering table data: While viewing data from a table, it is rare that all the data from the table will be required each time. The retrieval of specific columns from a table.

Syntax:-
Select column_name1, column_name2,column_name n from <table_name>;

- To retrieve selected rows and all columns from a table : ORACLE provides the option of using a „where‟ clause in an SQL sentence to apply a filter on the rows in the table. When a where clause is added it select only those rows that satisfy the specified condition, will be displayed.

Syntax:-

Select column_name1, column_name2,column_name n from <table_name>
Where <condition>;

Further comparison operators are:

- IN and NOT IN Predicates : In case a value needs to be compare to a list of values, then the IN and NOT IN predicate is used. We can check a single value against multiple values by using the IN & NOT IN predicate.

  Syntax: IN predicate
  Select column_name1, column_name2, ..,column_name n from < table name>
  where column_name in ('valuel' ,'value2', ..,'valuen');

- Set Conditions: <column> [not] in (<list of values>)

  Syntax: NOT IN
  Select column_namel, ,column_name n From < table name>
  Where column name not in (,,value 1 ',……,"value n ');

.

Guidelines:

1) The IN & NOT IN operators can be used with any datatype.
2) If characters or dates are used in the list, they must be enclosed in single quotation marks (' ').

Select column_name1,..,column_name n from < table name>

Where column_name between <lowerbound> and <upperbound>;

You can display rows based on a range of values using BETWEEN operators the range that you specify contains a lower bound and upper bound.

- Domain conditions: <column> [not] between <lower bound> and <upper bound>

  Syntax: NOT BETWEEN
  select column_name 1 , ..,column_name n from < table name> where column_name not between <lowerbound> and <upperbound>;

3. **String Operations:**

   In order to compare an attribute with a string, it is required to surround the string by apostrophes, e.g., where LOCATION = "DALLAS". A powerful operator for pattern matching is the like operator. Together with this operator, two special characters are used: the percent sign % (also called wild card), and the underline, also called position marker. For example, if one is interested in all tuples of the table DEPT that contain two C in the name of the department, the condition would be where DNAME like "%C%C%".

   The percent sign means that any (sub)string is allowed there, even the empty string. In contrast, the underline stands for exactly one character. Thus the condition where

DNAME like "%C C%" would require that exactly one character appears between the two Cs. To test for inequality, the not clause is used.

- **Like & Not Like Predicates:**
  Oracle provides a LIKE predicate, which allows comparison of one string with another string value which is not identical. This is achieved by using wildcard characters.
  - Wide card characters are:
    - Percent sign (%):- It matches any string.
    - Underscore (_):- It matches any single character.

  For char data types

  Syntax: - LIKE using % (Percentage)
  Select column_name 1, column _name2,. . . , column_name n from < table name> Where column~name like 'value%'

  OR

  Syntax: Like using '_' (underscore)
  Select column_name 1... Column_name n from < table name> where column name like '_ value%' ;

The above example will display all rows where the address starts with any character but ends with 'une'.

Further string operations are:

- Upper (<string>) takes a string and converts any letters in it to uppercase, e.g. DNAME= upper (DNAME) (The name of a department must consist only of upper case letters.)

- Lower (<string>) converts any letter to lowercase,

- initcap (<string>) converts the initial letter of every word in <string> to uppercase.

- Length (<string>) returns the length of the string.

- substr (<string>, n [, m]) clips out a m character piece of <string>, starting at position n. If m is not specified, the end of the string is assumed.

4. **Aggregate Functions:**
   Aggregate functions are statistical functions such as count, min, max etc. They are used to compute a single value from a set of attribute values of a column: The SUM and AVG must be a collection of numbers, but the other operators can operate on collections of nonnumeric data types such as string.

- AVG:-It is used to find out the average value.

Syntax: - avg (column_name)

- SUM:-It is used to find out the total or sum.
  Syntax: - sum (column_name)

- MIN:-It is used to find out the minimum value.
  Syntax: - min (column_name)

- MAX:-It is used to find out the average value. COUNT:-It is used to find out the total number of record or rows present in the relation or table.
  Syntax: - max (column_name)

- Count: It is used to eliminate the duplicate and null values in the specified column.
  Syntax: - count (Distinct column_name)

5. **Ordering operation:** using ORDER BY: The order by cause is used to arrange the records in ascending or descending order

   Syntax:-

   Select<expr> from <table_name>
   [where condition(s)]
   [order by{column,expr}[asc|desc]];

- Asc:-orders the rows in ascending order.by default order are asc.
- Desc:-orders the rows in descending order.

6. **Update Command:**
   The update command is used to change or modify data values in a table

   Syntax:-
   Update <table_name>
   Set column_name 1 = expression1,
   column_name 2 = expression2,
   .
   .
   column_name n = expression
   n where <search_condition>;

   An update statement without a where clause results in changing respective attributes of all tuples in the specified table. Typically, however, only a (small) portion of the table requires an update.

7. **Delete Command:**
   Delete command is used for deleting data (rows) from the table
   Syntax:-

Delete from <table_name>

- Removal of specified rows:- It is used to remove specific rows from the table

  Syntax:-
  Delete from <table_name>
  Where <search_condition>

To delete a student whose name is „abc"

Conclusion: we have seen the mysql three tier architecture

Assignment No 2

Title: Implement database with suitable example using MongoDB and implement all basic operations and administration commands using two tiers architecture.

Objective:

- To develop Database programming skill with multi-core programming
- To develop use data storage devices and related programming and Management skills

Requirements:

1. **Hardware:-** Server and Client,

2. **Software: -** MongoDB, Java/Python.

Theory:

1. **Databases:**

In addition to grouping documents by collection, MongoDB groups collections into databases. A single instance of MongoDB can host several databases, each grouping together zero or more collections. A database has its own permissions, and each database is stored in separate files on disk. A good rule of thumb is to store all data for a single application in the same database. Separate databases are useful when storing data for several application or users on the same MongoDB server. Like collections, databases are identified by name. Database names can be any UTF-8 string, with the following restrictions:

1. The empty string ("") is not a valid database name.
2. A database name cannot contain any of these characters: /, \, ., ", *, <, >, :, |, ?, $, (a single space), or \0 (the null character). Basically, stick with alphanumeric ASCII.
3. Database names are case-sensitive, even on non-case-sensitive file systems. To keep things simple, try to just use lowercase characters.
4. Database names are limited to a maximum of 64 bytes.

There are also several reserved database names, which you can access but which have special semantics. These are as follows:

- **Admin:** This is the "root" database, in terms of authentication. If a user is added to the admin database, the user automatically inherits permissions for all databases. There are also certain server-wide commands that can be run only from the admin database, such as listing all of the databases or shutting down the server.

- **Local:** This database will never be replicated and can be used to store any collections that should be local to a single server

2. MongoDB Create Database:

2.1. **The Use Command:** MongoDB **use DATABASE_NAME** is used to create database. The command will create a new database; if it doesn't exist otherwise it will return the existing database.

Syntax:

Use DATABASE_NAME

Example: If you want to create a database with name **<mydb>**, then **use database** statement would be as follows:

```
>use mydb switched
to db mydb
```

To check your currently selected database use the command **db**

```
>db
Mydb
```

If you want to check your databases list, then use the command **show dbs**.

```
>show dbs
local  0.78125GB
test 0.23012GB
```

Your created database (mydb) is not present in list. To display database you need to insert at least one document into it.

```
>db.movie.insert({"name":"tutorials
point"}) >show dbs
local    0.78125GB
mydb  0.23012GB
test 0.23012GB
```

In mongodb default database is test. If you didn't create any database then collections will be stored in test database.

3. MongoDB Drop Database:

3.1. **The Drop Database:** MongoDB **db.dropDatabase ()** command is used to drop an existing database.

Syntax:

Basic syntax of **dropDatabase ()** command is as follows:

```
db.dropDatabase()
```

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database

Example:

First, check the list available databases by using the command **show dbs** >show dbs

local 0.78125GB
mydb 0.23012GB
test 0.23012GB

If you want to delete new database **<mydb>**, then **dropDatabase()** command would be as follows:

>use mydb switched
to db mydb
>db.dropDatabase()
>{ "dropped" : "mydb", "ok" : 1 }

Now check list of databases
>show dbs

local 0.78125GB
test 0.23012GB

4. **Basic Operations with the Shell:** We can use the four basic operations, create, read, update, and delete (CRUD) to manipulate and view data in the shell.

4.1. **Create:** The insert function adds a document to a collection. For example, suppose we want to store a blog post. First, we‟ll create a local variable called post that is a JavaScript object representing our document. It will have the keys "title", "content", and "date". (the date that it was published):

```
> post = {"title" : "My Blog Post",
... "content" : "Here's my blog post.",
... "date" : new Date()}
{
"title" : "My Blog Post", "content" :
"Here's my blog post.",
"date" : ISODate("2012-08-24T21:12:09.982Z")
}
```

This object is a valid MongoDB document, so we can save it to the blog collection using the insert method:
> db.blog.insert(post)

4.2. Read: Find and findOne can be used to query a collection. If we just want to see one document from a collection, we can use findOne.

```
> db.blog.findOne()
{
"_id" : ObjectId("5037ee4a1084eb3ffeef7228"),
"title" : "My Blog Post",
"content" : "Here's my blog post.",
"date" : ISODate("2012-08-24T21:12:09.982Z")
```

}

Find and findOne can also be passed criteria in the form of a query document. This will restrict the documents matched by the query. The shell will automatically display up to 20 documents matching a find, but more can be fetched. See Chapter 4 for more information on querying.

4.3. Update: If we would like to modify our post, we can use update. Update takes (at least) two parameters: the first is the criteria to find which document to update, and the second is the new document. Suppose we decide to enable comments on the blog post we created earlier. We‟ll need to add an array of comments as the value for a new key in our document.

The first step is to modify the variable post and add a "comments" key:

```
> post.comments = [] [ ]
```

Then we perform the update, replacing the post titled “My Blog Post” with our new version of the document:

```
> db.blog.update({title : "My Blog Post"}, post)
```

Now the document has a "comments" key. If we call find again, we can see the new key:

```
> db.blog.find()
{
"_id" : ObjectId("5037ee4a1084eb3ffeef7228"), "title" :
"My Blog Post",
"content" : "Here's my blog post.",
"date" : ISODate("2012-08-24T21:12:09.982Z"),
"comments" : [ ]
}
```

4.4. **Delete:** Remove permanently deletes documents from the database. Called with no parameters, it removes all documents from a collection. It can also take a document specifying criteria for removal. For example, this would remove the post we just created:

```
> db.blog.remove({title : "My Blog Post"})
```

Conclusion: In this way we have seen MongoDB commands for operation.

Assignment No 3

Title: Write a program in Python/C++ to test that computer is booted with Legacy Boot ROMBIOS or UEFI.

Objective:

- To develop Database programming skill with multi-core programming
- To develop use data storage devices and related programming and Management skills

Requirements:

1. **Hardware:-** Server and Client,
2. **Software: -** Python/C++

Theory:

UEFI

The original motivation for EFI came during early development of the first Intel–HP Itanium systems in the mid-1990s. BIOS limitations (such as 16-bit processor mode, 1 MB addressable space and PC AT hardware) had become too restrictive for the larger server platforms Itanium was targeting.[7] The effort to address these concerns began in 1998 and was initially called *Intel Boot Initiative*;[8] it was later renamed to EFI.[9][10]

In July 2005, Intel ceased its development of the EFI specification at version 1.10, and contributed it to the Unified EFI Forum, which has evolved the specification as the Unified Extensible Firmware Interface (UEFI). The original EFI specification remains owned by Intel, which exclusively provides licenses for EFI-based products, but the UEFI specification is owned by the Forum.[7][11]

Version 2.1 of the UEFI (*Unified Extensible Firmware Interface*) specification was released on 7 January 2007. It added cryptography, network authentication and the User Interface Architecture (Human Interface Infrastructure in UEFI). The latest UEFI specification, version 2.5, was approved in April 2015.

Advantages

The interface defined by the EFI specification includes data tables that contain platform information, and boot and runtime services that are available to the OS loader and OS. UEFI firmware provides several technical advantages over a traditional BIOS system:[12]

- Ability to boot from large disks (over 2 TB) with a GUID Partition Table (GPT)[13][b]
- CPU-independent architecture

- CPU-independent drivers
- Flexible pre-OS environment, including network capability
- Modular design

Compatibility

Processor compatibility

As of version 2.5, processor bindings exist for Itanium, x86, x86-64, ARM (AArch32) and ARM64 (AArch64).[14] Only little-endian processors can be supported.[15]

Standard PC BIOS is limited to a 16-bit processor mode and 1 MB of addressable memory space, resulting from the design based on the IBM 5150 that used a 16-bit Intel 8088 processor.[7][16] In comparison, the processor mode in a UEFI environment can be either 32-bit (x86-32, AArch32) or 64-bit (x86-64, Itanium, and AArch64).[7][17] 64-bit UEFI firmware implementations support long mode, which allows applications in the preboot execution environment to use 64-bit addressing to get direct access to all of the machine's memory.[18]

UEFI requires the firmware and operating system loader (or kernel) to be size-matched; for example, a 64-bit UEFI firmware implementation can load only a 64-bit operating system boot loader or kernel. After the system transitions from "Boot Services" to "Runtime Services", the operating system kernel takes over. At this point, the kernel can change processor modes if it desires, but this bars usage of the runtime services (unless the kernel switches back again).[19]:sections 2.3.2 and 2.3.4 As of version 3.15, Linux kernel supports booting of 64-bit kernels on 32-bit UEFI firmware implementations running on x86-64 CPUs, with *UEFI handover* support from a UEFI boot loader as the requirement.[20] UEFI handover protocol deduplicates the UEFI initialization code between the kernel and UEFI boot loaders, leaving the initialization to be performed only by the Linux kernel's *UEFI boot stub*.[21][22]

Disk device compatibility

See also: GPT § Operating systems support and Protective MBR

In addition to the standard PC disk partition scheme that uses a master boot record (MBR), UEFI also works with a new partitioning scheme called GUID Partition Table (GPT), which is free from many of the limitations of MBR. In particular, the MBR limits on the number and size of disk partitions (up to four primary partitions per disk, and up to 2 TB (2×2^{40} bytes) per disk) are relaxed.[23] More specifically, GPT allows for a maximum disk and partition size of 8 ZB (8×2^{70} bytes).[23][24]

Linux

See also: EFI System partition § Linux

Support for GPT in Linux is enabled by turning on the option `CONFIG_EFI_PARTITION` (EFI GUID Partition Support) during kernel configuration.[25] This option allows Linux to recognize and use GPT disks after the system firmware passes control over the system to Linux.

For reverse compatibility, Linux can use GPT disks in BIOS-based systems for both data storage and booting, as both GRUB 2 and Linux are GPT-aware. Such a setup is usually referred to as *BIOS-GPT*.[26] As GPT incorporates the protective MBR, a BIOS-based computer can boot from a GPT disk using GPT-aware boot loader stored in the protective MBR's bootstrap code area.[24] In case of GRUB, such a configuration requires a BIOS Boot partition for GRUB to embed its second-stage code due to absence of the post-MBR gap in GPT partitioned disks (which is taken over by the GPT's *Primary Header* and *Primary Partition Table*). Commonly 1 MiB in size, this partition's Globally Unique Identifier (GUID) in GPT scheme is `21686148-6449-6E6F-744E-656564454649` and is used by GRUB only in BIOS-GPT setups. From the GRUB's perspective, no such partition type exists in case of MBR partitioning. This partition is not required if the system is UEFI-based because no embedding of the second-stage code is needed in that case.[13][24][26]

UEFI systems can access GPT disks and boot directly from them, which allows Linux to use UEFI boot methods. Booting Linux from GPT disks on UEFI systems involves creation of an EFI System partition (ESP), which contains UEFI applications such as bootloaders, operating system kernels, and utility software.[27][28][29] Such a setup is usually referred to as *UEFI-GPT*, while ESP is recommended to be at least 512 MiB in size and formatted with a FAT32 filesystem for maximum compatibility.[24][26][30]

For backward compatibility, most UEFI implementations also support booting from MBR-partitioned disks, through the Compatibility Support Module (CSM) that provides legacy BIOS compatibility.[31] In that case, booting Linux on UEFI systems is the same as on legacy BIOS-based systems.

Microsoft Windows

The 64-bit versions of Microsoft Windows Vista[c] and later, 32-bit versions of Windows 8 and later, and the Itanium versions of Windows XP and Server 2003 can boot from disks with a partition size larger than 2 TB

BIOS

BIOS (basic input/output system) is the program a personal computer's microprocessor uses to get the computer system started after you turn it on. It also manages data flow between the computer's operating system and attached devices such as the hard disk, video adapter, keyboard, mouse and printer.

BIOS is an integral part of your computer and comes with it when you bring it home. (In contrast, the operating system can either be pre-installed by the manufacturer or vendor or installed by the user.) BIOS is a program that is made accessible to the microprocessor on an erasable programmable read-only memory (EPROM) chip. When you turn on your computer, the microprocessor passes control to the BIOS program, which is always located at the same place on EPROM.

When BIOS boots up (starts up) your computer, it first determines whether all of the attachments are in place and operational and then it loads the operating system (or key parts of it) into your computer's random access memory (RAM) from your hard disk or diskette drive.

With BIOS, your operating system and its applications are freed from having to understand exact details (such as hardware addresses) about the attached input/output devices. When device details change, only the BIOS program needs to be changed. Sometimes this change can be made during your system setup. In any case, neither your operating system or any applications you use need to be changed.

Although BIOS is theoretically always the intermediary between the microprocessor and I/O device control information and data flow, in some cases, BIOS can arrange for data to flow directly to memory from devices (such as video cards) that require faster data flow to be effective.

Conclusion: We have seen details of UEFI and BIOS.

Assignment No 4

Title: Write a program in C++ to create a RAMDRIVE and associate an acyclic directory structure to it. Use this RAMDRIVE to store input, out files to run a calculator program.

Objective:

- To develop Database programming skill with multi-core programming
- To develop use data storage devices and related programming and Management skills

Requirements:

1. **Hardware:-** Server and Client,

2. **Software: -**

Theory:

1. Create Input File in /tmp/ramdisk for input
   name it as input.txt

2. Write a Program for calculator in C

code is as follow

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 int a,b,res;
 FILE *fp;
 fp=fopen("/tmp/ramdisk/input.txt","r");
 fscanf(fp,"%d",&a);
 fscanf(fp,"%d",&b);
 fclose(fp);
  fp=fopen("/tmp/ramdisk/output.txt","w");
 fprintf(fp,"\n Addition is :%d ",(a+b));
return 0;
}
```

save it as calci.c and compile using command
   gcc calci.c -o calculator

3. Write program for RAMDISK in C language

```
   #include<stdio.h>
#include<stdlib.h>
int main()
{
  printf("\n RAMDISK creation for 512M size");
  system("rmdir /tmp/ramdisk");
  system("mkdir /tmp/ramdisk");
  system("chmod 777 /tmp/ramdisk");
  system("mount -t tmpfs -o size=512M tmpfs /tmp/ramdisk/");
  printf("\n RAMDISK created");
  system("df -h|grep ram");
  system("cp input.txt /tmp/ramdisk");
  system("cp calculator /tmp/ramdisk");
  return 0;
}
```

save it as ramdisk.c and compile as

gcc ramdisk.c -o ramdisk

execute output of above command as
ramdisk <Enter>

4. Calculator code can be executed using command
/tmp/ramdisk/calculator

5. Output is saved in output.txt file. Check it from /tmp/ramdisk/output.txt

Conclusion: In this way we have seen use of RAM disk to store calculator program.

Assignment No 5

Title: Write a Python/Java/C+ program to verify the operating system name and version of Mobile devices.

Objective:

- To develop Database programming skill with multi-core programming
- To develop use data storage devices and related programming and Management skills

Requirements:

1. **Hardware:-** Server and Client,

2. **Software: -** Python, Java.

Theory:

System Properties

In Properties, we examined the way an application can use `Properties` objects to maintain its configuration. The Java platform itself uses a `Properties` object to maintain its own configuration. The `System` class maintains a `Properties` object that describes the configuration of the current working environment. System properties include information about the current user, the current version of the Java runtime, and the character used to separate components of a file path name.

The following table describes some of the most important system properties

Key	Meaning
`"file.separator"`	Character that separates components of a file path. This is `"/"` on UNIX and `"\"` on Windows.
`"java.class.path"`	Path used to find directories and JAR archives containing class files. Elements of the class path are separated by a platform-specific character specified in the `path.separator` property.
`"java.home"`	Installation directory for Java Runtime Environment (JRE)
`"java.vendor"`	JRE vendor name

`"java.vendor.url"`	JRE vendor URL
`"java.version"`	JRE version number
`"line.separator"`	Sequence used by operating system to separate lines in text files
`"os.arch"`	Operating system architecture
`"os.name"`	Operating system name
`"os.version"`	Operating system version
`"path.separator"`	Path separator character used in `java.class.path`
`"user.dir"`	User working directory
`"user.home"`	User home directory
`"user.name"`	User account name

Reading System Properties

The `System` class has two methods used to read system properties: `getProperty` and `getProperties`. The `System` class has two different versions of `getProperty`. Both retrieve the value of the property named in the argument list. The simpler of the two `getProperty` methods takes a single argument, a property key For example, to get the value of `path.separator`, use the following statement:

```
System.getProperty("path.separator");
```

The `getProperty` method returns a string containing the value of the property. If the property does not exist, this version of `getProperty` returns null.
The other version of `getProperty` requires two `String` arguments: the first argument is the key to look up and the second argument is a default value to return if the key cannot be found or if it has no value. For example, the following invocation of `getProperty` looks up the `System` property called `subliminal.message`. This is not a valid system property, so instead of returning null, this method returns the default value provided as a second argument: `"Buy StayPuft Marshmallows!"`

```
System.getProperty("subliminal.message", "Buy StayPuft Marshmallows!");
```

The last method provided by the `System` class to access property values is the `getProperties` method, which returns a `Properties` object. This object contains a complete set of system property definitions.

Writing System Properties

To modify the existing set of system properties, use `System.setProperties`. This method takes a `Properties` object that has been initialized to contain the properties to be set. This method replaces the entire set of system properties with the new set represented by the `Properties` object.

The next example, `PropertiesTest`, creates a `Properties` object and initializes it from `myProperties.txt`.

```
subliminal.message=Buy StayPuft Marshmallows!
```

`PropertiesTest` then uses `System.setProperties` to install the new `Properties` objects as the current set of system properties.

```
import java.io.FileInputStream;
import java.util.Properties;

public class PropertiesTest {
    public static void main(String[] args)
        throws Exception {

        // set up new properties object
        // from file "myProperties.txt"
        FileInputStream propFile =
            new FileInputStream( "myProperties.txt");
        Properties p =
            new Properties(System.getProperties());
        p.load(propFile);

        // set the system properties
        System.setProperties(p);
        // display new properties
        System.getProperties().list(System.out);
    }
}
```

Note how `PropertiesTest` creates the `Properties` object, `p`, which is used as the argument to `setProperties`:

```
Properties p = new Properties(System.getProperties());
```

This statement initializes the new properties object, `p`, with the current set of system properties, which in the case of this small application, is the set of properties initialized by the runtime system. Then the application loads additional properties into `p` from the file `myProperties.txt` and sets the system properties to `p`. This has the effect of adding the properties listed in `myProperties.txt` to the set of

properties created by the runtime system at startup. Note that an application can create `p` without any default `Properties` object, like this:

```
Properties p = new Properties();
```

Also note that the value of system properties can be overwritten! For example, if `myProperties.txt` contains the following line, the `java.vendor` system property will be overwritten:

```
java.vendor=Acme Software Company
```

In general, be careful not to overwrite system properties.

The `setProperties` method changes the set of system properties for the current running application. These changes are not persistent. That is, changing the system properties within an application will not affect future invocations of the Java interpreter for this or any other application. The runtime system re-initializes the system properties each time its starts up. If changes to system properties are to be persistent, then the application must write the values to some file before exiting and read them in again upon startup.

Conclusion: We have seen demonstration of program used for system properties.

Assignment No 6

Title: Aggregation and indexing with suitable example using MongoDB

Objective:

- To develop Database programming skill with multi-core programming
- To develop use data storage devices and related programming and Management skills

Requirements:

1. **Hardware:-** Server and Client,

2. **Software: -** MongoDB, Java/Python.

Theory:

Aggregation:

Aggregation Introduction

Aggregations are operations that process data records and return computed results. MongoDB provides a rich set of aggregation operations that examine and perform calculations on the data sets. Running data aggregation on the `mongod` instance simplifies application code and limits resource requirements.

Like queries, aggregation operations in MongoDB use *collections* of documents as an input and return results in the form of one or more documents.

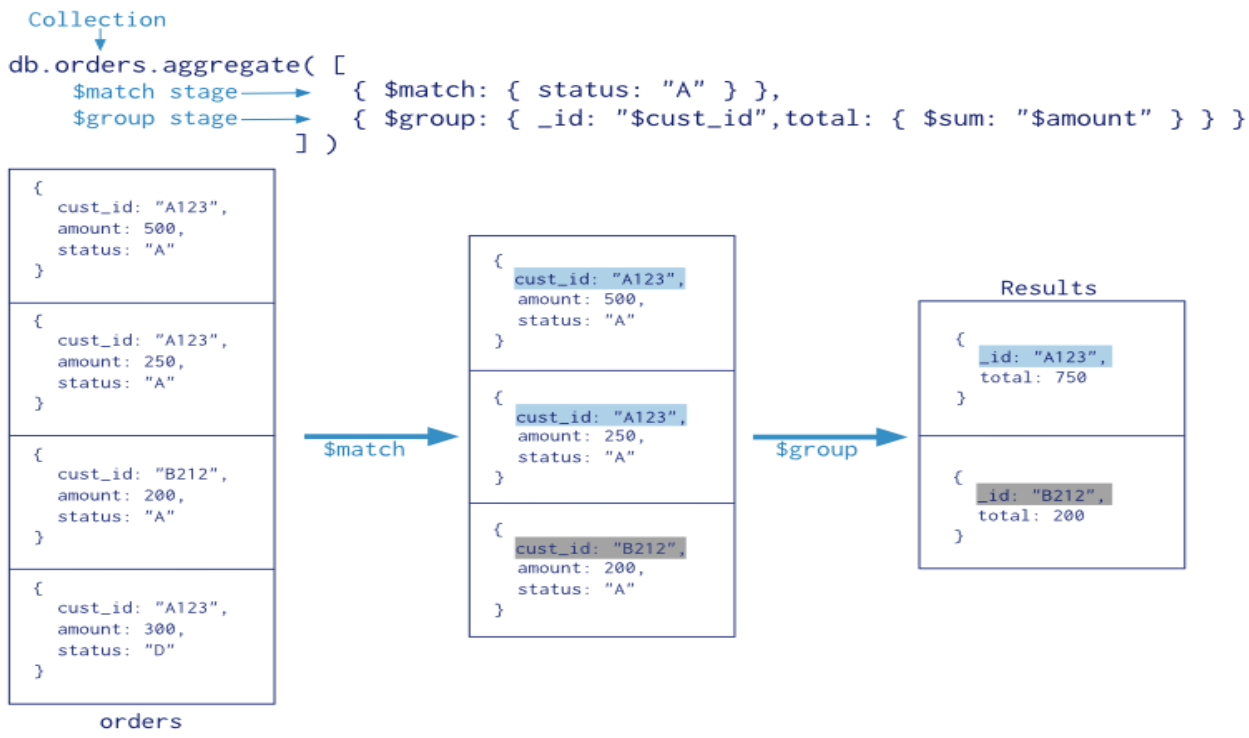
Aggregation Pipelines

MongoDB 2.2.x introduced a new *aggregation framework*, modeled on the concept of data processing pipelines. Documents enter a multi-stage pipeline that transforms the documents into an aggregated result.

The most basic pipeline stages provide *filters* that operate like queries and *document transformations* that modify the form of the output document.

Other pipeline operations provide tools for grouping and sorting documents by specific field or fields as well as tools for aggregating the contents of arrays, including arrays of documents. In addition, pipeline stages can use *operators* for tasks such as calculating the average or concatenating a string.

The pipeline provides efficient data aggregation using native operations within MongoDB, and is the preferred method for data aggregation in MongoDB.



Aggregation refers to a broad class of data manipulation operations that compute a result based on an input *and* a specific procedure. MongoDB provides a number of aggregation operations that perform specific aggregation operations on a set of data.

Although limited in scope, particularly compared to the *aggregation pipeline* and *map-reduce*, these operations provide straightforward semantics for common data processing options.

Count

MongoDB can return a count of the number of documents that match a query. The count command as well as the count() and cursor.count() methods provide access to counts in the mongo shell.

Example

Given a collection named records with *only* the following documents:

```
{ a: 1, b: 0 }
{ a: 1, b: 1 }
{ a: 1, b: 4 }
{ a: 2, b: 2 }
```

The following operation would count all documents in the collection and return the number 4:

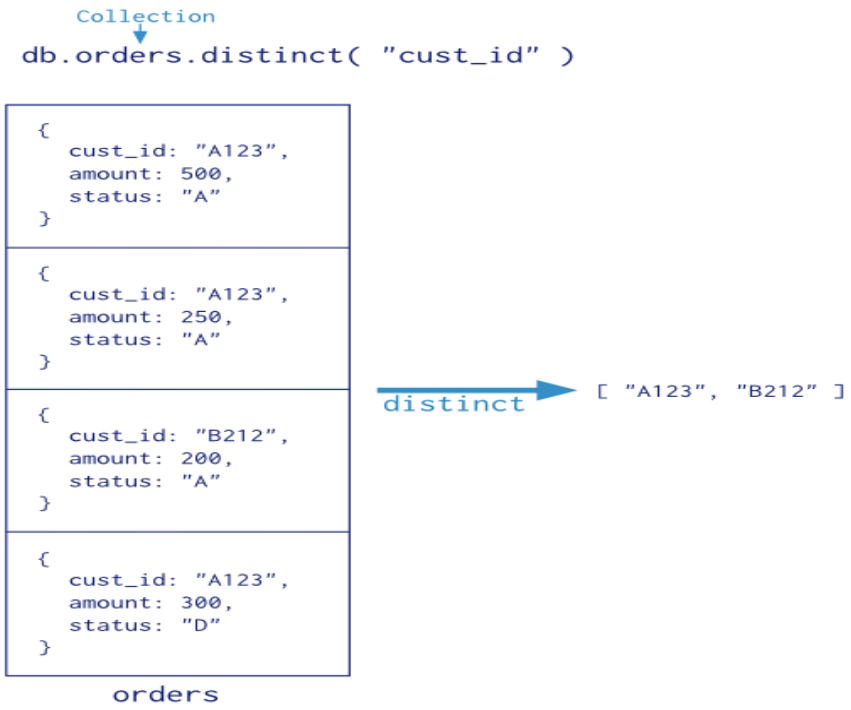
```
db.records.count()
```

The following operation will count only the documents where the value of the field a is 1 and return 3:

```
db.records.count( { a: 1 } )
```

Distinct

The *distinct* operation takes a number of documents that match a query and returns all of the unique values for a field in the matching documents. The distinct command and db.collection.distinct() method provide this operation in the mongo shell. Consider the following examples of a distinct operation:



Example

Given a collection named records with *only* the following documents:

```
{ a: 1, b: 0 }
{ a: 1, b: 1 }
{ a: 1, b: 1 }
{ a: 1, b: 4 }
{ a: 2, b: 2 }
{ a: 2, b: 2 }
```

Consider the following db.collection.distinct() operation which returns the distinct values of the field b:

```
db.records.distinct( "b" )
```

The results of this operation would resemble:

```
[ 0, 1, 4, 2 ]
```

Group

The *group* operation takes a number of documents that match a query, and then collects groups of documents based on the value of a field or fields. It returns an array of documents with computed results for each group of documents.

Access the grouping functionality via the group command or the db.collection.group() method in the mongo shell.

Warning

group does not support data in sharded collections. In addition, the results of the group operation must be no larger than 16 megabytes.

Consider the following group operation:

Example

Given a collection named records with the following documents:

```
{ a: 1, count: 4 }
{ a: 1, count: 2 }
{ a: 1, count: 4 }
{ a: 2, count: 3 }
{ a: 2, count: 1 }
{ a: 1, count: 5 }
{ a: 4, count: 4 }
```

Consider the following group operation which groups documents by the field a, where a is less than 3, and sums the field count for each group:

```
db.records.group( {
   key: { a: 1 },
   cond: { a: { $lt: 3 } },
   reduce: function(cur, result) { result.count += cur.count },
   initial: { count: 0 }
} )
```

The results of this group operation would resemble the following:

```
[
  { a: 1, count: 15 },
  { a: 2, count: 4 }
]
```

Conclusion: In this way we have seen indexing and aggregatio

Assignment No 7

Title: Map reduces operation with suitable example using MongoDB.

Objective:

- To develop Database programming skill with multi-core programming
- To develop use data storage devices and related programming and Management skills

Requirements:

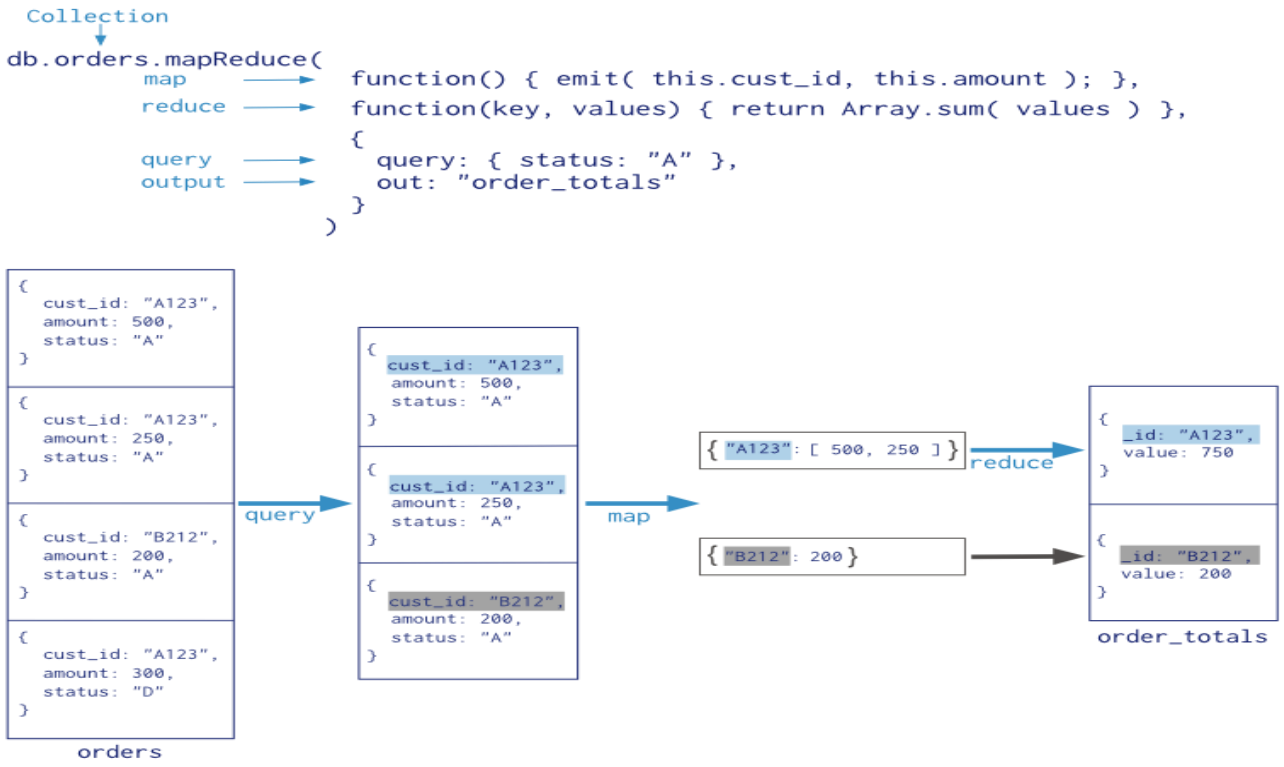
1. **Hardware:-** Server and Client,

2. **Software: -** MongoDB, Java/Python.

Theory:

Map-reduce is a data processing paradigm for condensing large volumes of data into useful *aggregated* results. For map-reduce operations, MongoDB provides the `mapReduce` database command.

Consider the following map-reduce operation:



In this map-reduce operation, MongoDB applies the *map* phase to each input document (i.e. the documents in the collection that match the query condition). The map function emits key-value pairs. For those keys that have multiple values, MongoDB applies the *reduce* phase, which collects and condenses the aggregated data. MongoDB then stores the results in a collection. Optionally, the output of the reduce function may pass through a *finalize* function to further condense or process the results of the aggregation.

All map-reduce functions in MongoDB are JavaScript and run within the `mongod` process. Map-reduce operations take the documents of a single *collection* as the *input* and can perform any arbitrary sorting and limiting before beginning the map stage. `mapReduce` can return the results of a map-reduce operation as a document, or may write the results to collections. The input and the output collections may be sharded.

Map-Reduce JavaScript Functions

In MongoDB, map-reduce operations use custom JavaScript functions to *map*, or associate, values to a key. If a key has multiple values mapped to it, the operation *reduces* the values for the key to a single object.

The use of custom JavaScript functions provide flexibility to map-reduce operations. For instance, when processing a document, the map function can create more than one key and value mapping or no mapping. Map-reduce operations can also use a custom JavaScript function to make final modifications to the results at the end of the map and reduce operation, such as perform additional calculations.

Map-Reduce Behavior

In MongoDB, the map-reduce operation can write results to a collection or return the results inline. If you write map-reduce output to a collection, you can perform subsequent map-reduce operations on the same input collection that merge replace, merge, or reduce new results with previous resultsWhen returning the results of a map reduce operation *inline*, the result documents must be within the `BSON Document Size` limit, which is currently 16 megabytes.
MongoDB supports map-reduce operations on *sharded collections*. Map-reduce operations can also output the results to a sharded collection.

Conclusion: In this way we have learnt Map reduces operation with suitable example using MongoDB

Assignment No 8

Title: Indexing and querying with MongoDB using suitable example.

Objective:

- To develop Database programming skill with multi-core programming
- To develop use data storage devices and related programming and Management skills

Requirements:

1. **Hardware:-** Server and Client,

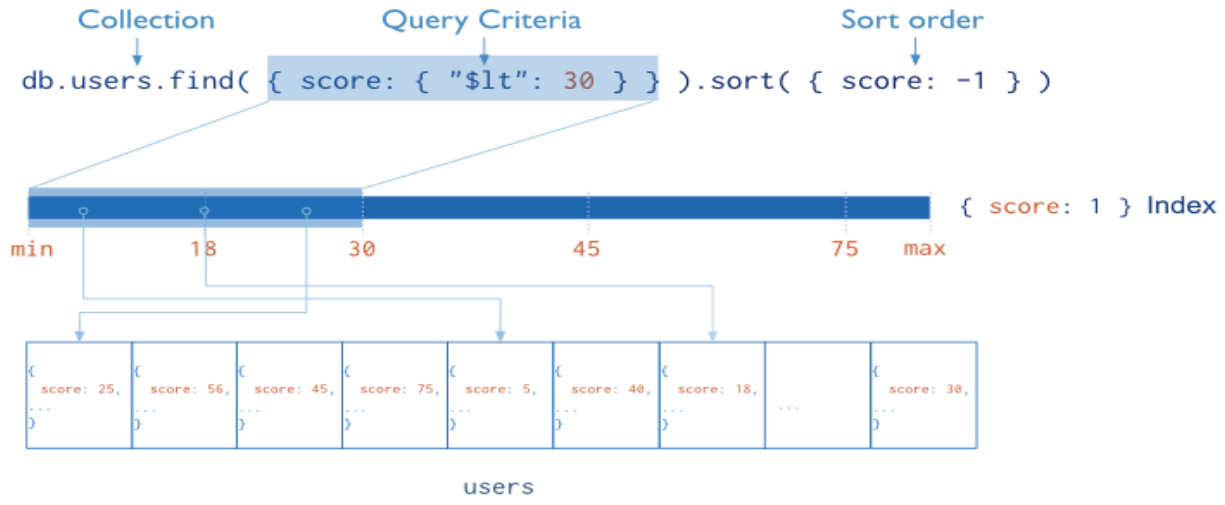
2. **Software: -** MongoDB, Java/Python.

Theory:

Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must perform a *collection scan*, i.e. scan every document in a collection, to select those documents that match the query statement. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect.

Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field. The ordering of the index entries supports efficient equality matches and range-based query operations. In addition, MongoDB can return sorted results by using the ordering in the index.

The following diagram illustrates a query that selects and orders the matching documents using an index:



Fundamentally, indexes in MongoDB are similar to indexes in other database systems. MongoDB defines indexes at the *collection* level and supports indexes on any field or sub-field of the documents in a MongoDB collection.

MongoDB indexes use a B-tree data structure.

Index Types

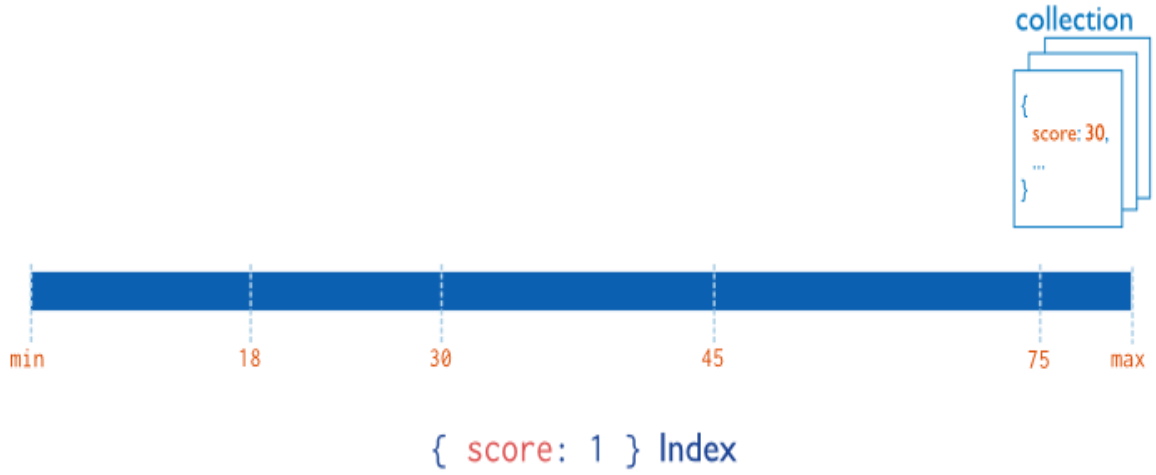
MongoDB provides a number of different index types to support specific types of data and queries.

Default `_id`

All MongoDB collections have an index on the `_id` field that exists by default. If applications do not specify a value for `_id` the driver or the `mongod` will create an `_id` field with an *ObjectId* value. The `_id` index is *unique* and prevents clients from inserting two documents with the same value for the `_id` field.

Single Field

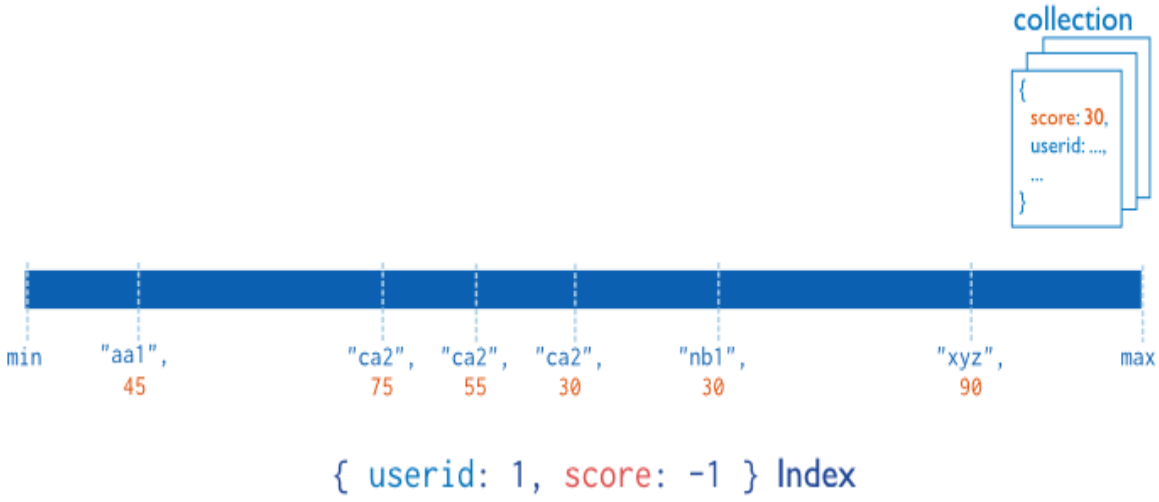
In addition to the MongoDB-defined `_id` index, MongoDB supports the creation of user-defined ascending/descending indexes on a *single field of a document*.



For a single-field index and sort operations, the sort order (i.e. ascending or descending) of the index key does not matter because MongoDB can traverse the index in either direction.

Compound Index

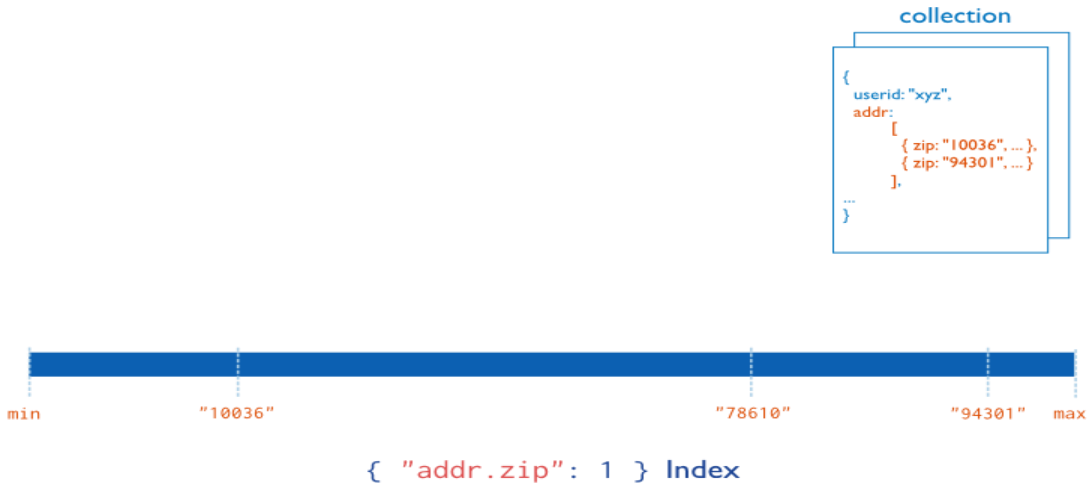
MongoDB also supports user-defined indexes on multiple fields, i.e. *compound indexes*. The order of fields listed in a compound index has significance. For instance, if a compound index consists of `{ userid: 1, score: -1 }`, the index sorts first by `userid` and then, within each `userid` value, sorts by `score`.



For compound indexes and sort operations, the sort order (i.e. ascending or descending) of the index keys can determine whether the index can support a sort operation. See *Sort Order* for more information on the impact of index order on results in compound indexes.

Multikey Index

MongoDB uses *multikey indexes* to index the content stored in arrays. If you index a field that holds an array value, MongoDB creates separate index entries for *every* element of the array. These *multikey indexes* allow queries to select documents that contain arrays by matching on element or elements of the arrays. MongoDB automatically determines whether to create a multikey index if the indexed field contains an array value; you do not need to explicitly specify the multikey type.



Geospatial Index

To support efficient queries of geospatial coordinate data, MongoDB provides two special indexes: *2d indexes* that uses planar geometry when returning results and *2sphere indexes* that use spherical geometry to return results.

Text Indexes

MongoDB provides a `text` index type that supports searching for string content in a collection. These text indexes do not store language-specific *stop* words (e.g. “the”, “a”, “or”) and *stem* the words in a collection to only store root words.

Hashed Indexes

To support *hash based sharding*, MongoDB provides a *hashed index* type, which indexes the hash of the value of a field. These indexes have a more random distribution of values along their range, but *only* support equality matches and cannot support range-based queries.

Index Properties

Unique Indexes

The *unique* property for an index causes MongoDB to reject duplicate values for the indexed field. Other than the unique constraint, unique indexes are functionally interchangeable with other MongoDB indexes.

Sparse Indexes

The *sparse* property of an index ensures that the index only contain entries for documents that have the indexed field. The index skips documents that *do not* have the indexed field. You can combine the

sparse index option with the unique index option to reject documents that have duplicate values for a field but ignore documents that do not have the indexed key.

TTL Indexes

TTL indexes are special indexes that MongoDB can use to automatically remove documents from a collection after a certain amount of time. This is ideal for certain types of information like machine generated event data, logs, and session information that only need to persist in a database for a finite amount of time.

Conclusion: In this way we have learnt indexing and aggregation in MongoDB.

Assignment No 9

Title: Connectivity with MongoDB using any Java application.

Objective:

- To develop Database programming skill with multi-core programming
- To develop use data storage devices and related programming and Management skills

Requirements:

1. **Hardware:-** Server and Client,

2. **Software: -** MongoDB, Java/Python.

Theory:

Since MongoDB is a document database, you might not be surprised to learn that you don't connect to it via traditional SQL/relational DB methods like JDBC. But it's simple all the same:

```
MongoClient mongoClient = new MongoClient(new MongoClientURI("mongodb://localhost:27017"));
```

Where I've put mongodb://localhost:27017, you'll want to put the address of where you've installed MongoDB. There's more detailed information on how to create the correct URI, including how to connect to a Replica Set, in the MongoClientURI documentation.

If you're connecting to a local instance on the default port, you can simply use:

```
MongoClient mongoClient = new MongoClient();
```

Note that this does throw a checked Exception, UnknownHostException. You'll either have to catch this or declare it, depending upon what your policy is for exception handling.
The MongoClient is your route in to MongoDB, from this you'll get your database and collections to work with (more on this later). Your instance of MongoClient (e.g. mongoClient above) will ordinarily be a singleton in your application. However, if you need to connect via different credentials (different user names and passwords) you'll want a MongoClient per set of credentials.
It is important to limit the number of MongoClient instances in your application, hence why we suggest a singleton - the MongoClient is effectively the connection pool, so for every new MongoClient, you are opening a new pool. Using a single MongoClient (and optionally configuring its settings) will allow the driver to correctly manage your connections to the server. This MongoClient singleton is safe to be used by multiple threads.
One final thing you need to be aware of: you want your application to shut down the connections to MongoDB when it finishes running. Always make sure your application or web server calls MongoClient.close() when it shuts down.

Try out connecting to MongoDB by getting the test in Exercise1ConnectingTest to pass.
Where are my tables?
MongoDB doesn't have tables, rows, columns, joins etc. There are some new concepts to learn when you're using it, but nothing too challenging.
While you still have the concept of a database, the documents (which we'll cover in more detail later) are stored in collections, rather than your database being made up of tables of data. But it can be helpful to think of documents like rows and collections like tables in a traditional database. And collections can have indexes like you'd expect.
Selecting Databases and Collections
You're going to want to define which databases and collections you're using in your Java application. If you remember, a few sections ago we used the MongoDB shell to show the databases in your MongoDB instance, and you had an admin and a local.
Creating and getting a database or collection is extremely easy in MongoDB:

```
DB database = mongoClient.getDB("TheDatabaseName");
```

You can replace "TheDatabaseName" with whatever the name of your database is. If the database doesn't already exist, it will be created automatically the first time you insert anything into it, so there's no need for null checks or exception handling on the off-chance the database doesn't exist.
Getting the collection you want from the database is simple too:

```
DBCollection collection = database.getCollection("TheCollectionName");
```

Again, replacing "TheCollectionName" with whatever your collection is called. Something that is, hopefully, becoming clear to you is that MongoDB is different from the traditional relational databases you've used. As I've mentioned, there are collections, rather than tables, and documents, rather than rows and columns.
Documents are much more flexible than a traditional row, as you have a dynamic schema rather than an enforced one. You can evolve the document over time without incurring the cost of schema migrations and tedious update scripts. But I'm getting ahead of myself.

Although documents don't look like the tables, columns and rows you're used to, they should look familiar if you've done anything even remotely JSON-like. Here's an example:

```
person = {
  _id: "jo",
  name: "Jo Bloggs",
  age: 34,
  address: {
    street: "123 Fake St",
    city: "Faketon",
    state: "MA",
    zip: "12345"
  }
  books: [ 27464, 747854, ...]
}
```

There are a few interesting things to note:
1. Like JSON, documents are structures of name/value pairs, and the values can be one of a number of primitive types, including Strings and various number types.

2. It also supports nested documents - in the example above, address is a subdocument inside the person document. Unlike a relational database, where you might store this in a separate table and provide a reference to it, in MongoDB if that data benefits from always being associated with its parent, you can embed it in its parent.

3. You can even store an array of values. The books field in the example above is an array of integers that might represent, for example, IDs of books the person has bought or borrowed.

Conclusion: In this way, we have learned the connecting Java application with MongoDB.

Assignment No 19

Title: Using MongoDB create a database of employee performance, employee attendance on the workstation. Perform statistical analysis for the results of the products produced by employees rated as passed ok, damaged products (5 samples per batch size 1000) and the portion covered in the training and absentee of the employees during training. Use programming language R. (or R-Python/R-Java) or equivalent assignment using R Programming Language for BiGDATA computing.

Objective:

- To develop Database programming skill with multi-core programming
- To develop use data storage devices and related programming and Management skills

Requirements:

1. **Hardware:-** Server and Client,

2. **Software: -** MongoDB, R-Java/R-Python.

Theory:

Mathematical Model:

Conclusion:

Assignment Group C

Assignment No 1

Title: BIG DATA applications using Hadoop

Objective:

- To develop Database programming skill with multi-core programming
- To develop use data storage devices and related programming and Management skills

Requirements:

1. **Hardware:-** Server and Client,

2. **Software: -** Hadoop, Java/Python.

Theory:

Mathematical Model:

Conclusion:

Assignment No 2

Title: BIG DATA applications using Blogs

Objective:

- To develop Database programming skill with multi-core programming
- To develop use data storage devices and related programming and Management skills

Requirements:

1. **Hardware:-** Server and Client,

2. **Software: -** Hadoop, Java/Python.

Theory:

Mathematical Model:

Conclusion:

Assignment No 3

Title: Big Data Predictive Machine Learning

Objective:

- To develop Database programming skill with multi-core programming
- To develop use data storage devices and related programming and Management skills

Requirements:

1. **Hardware:-** Server and Client,

2. **Software: -** Hadoop, Java/Python.

Theory:

Mathematical Model:

Conclusion:

Assignment No 4

Title: Create and test functioning of Windows-8 ReFS (Resilient File System)

Objective:

- To develop Database programming skill with multi-core programming
- To develop use data storage devices and related programming and Management skills

Requirements:

1. **Hardware:-** Server and Client,

2. **Software: -**

Theory:

Mathematical Model:

Conclusion: