DATABASE II

Indexes I & SimpleDB

Dr. Fatma Meawad
Caroline Sabty

DataBase II

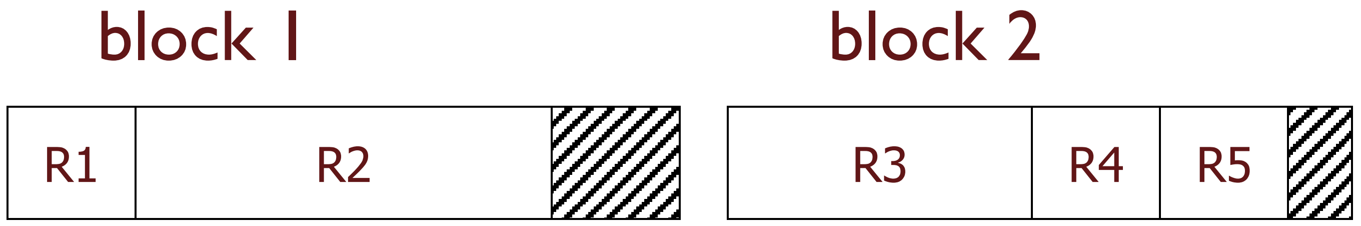
There are two main categories in undergraduate introductory database course:

1- How to use a database system (user oriented)
Topics :database design, relational algebra, SQL
Covered in DB1 with Prof Slim ;)

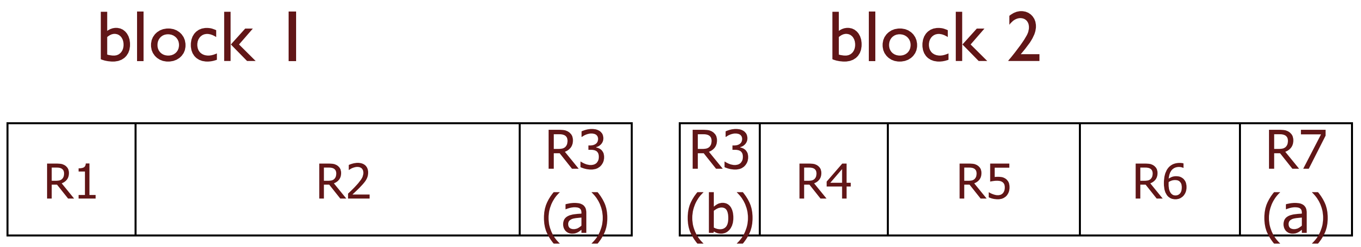
2- How a database system works (system oriented). How Databases are Implemented.
Topics: concurrency control, recovery, indexing, and query processing
Going to be covered in this Course :)

Spanned vs. Unspanned

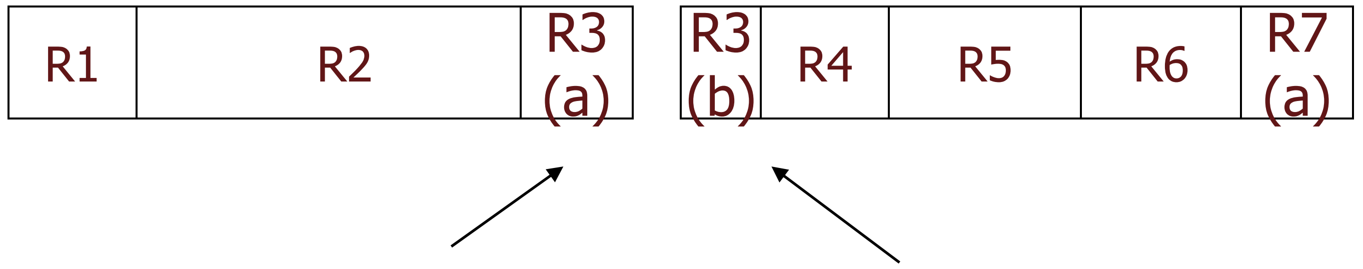
- Unspanned: records must be within one block



- Spanned:



With spanned records



need indication
of partial record
"pointer" to rest

need indication
of continuation
(+ from where?)

Spanned vs. unspanned:

Unspanned :

- Simpler
- Waste of space

Spanned essential if

record size > block size

Random Order file

How do we find sid =40 ?

Select * from Student
 where sid=40

sid	name	GPA
20	Susan	3.5
60	James	1.7
70	Peter	2.6
40	Elaine	3.9

Search a random list linear O(n) → expensive
BUT search a sorted list using binary sort O(log n) → better

Indexes

Why you might create an index?

- Index is an auxiliary file
- By defining index you improve database performance by making it faster to find and sort the information stored in a table

- One form of an index is a file of entries **<field value, data pointer >**, which is ordered by field value.

Indexes

The database index is like the index in a book. Rather than search through the entire book for a particular subject, you look at the index because it contains all the different subject titles of the book in a sorted format, and it gives you the page number. Obviously this is much faster than scanning the entire book. Similarly, as a book index contains a page #, a database index contains a pointer to the row containing the value that you are looking for.

Indexes

Any table can have multiple indexes defined on it.

Cost of having a database index?

- Space (the larger your table, the larger your index).
- Add, delete, or update rows should be also done to the index.

Types of Indexes

- Single-level Ordered Indexes
  - Primary Indexes
  - Clustering Indexes
  - Secondary Indexes
- Multi-level Indexes

Dense vs. Spare Index

- Dense Index: has an index entry for every search key value (hence every record) in the data file
- Sparse (non-dense) Index: has an index entries for only some of the search values

Primary Index

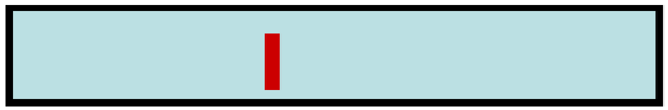
- Similar to a unique index, except that each table can have only one primary key (uniquely identify each row in the table).
- Includes one index entry for each block in the data file. **The data file is ordered on a key field (Primary Index).**
- When a unique index is created, the key data is checked for uniqueness and fails if duplicates are found.
  Notes:
  They can't be dropped unless you drop the whole table.
  Primary indexes created implicitly are defined as follows:
  CREATE UNIQUE INDEX index1 On employee (Id ASC);
  CREATE UNIQUE INDEX index2 On dept (Id ASC);

Single Table Scan

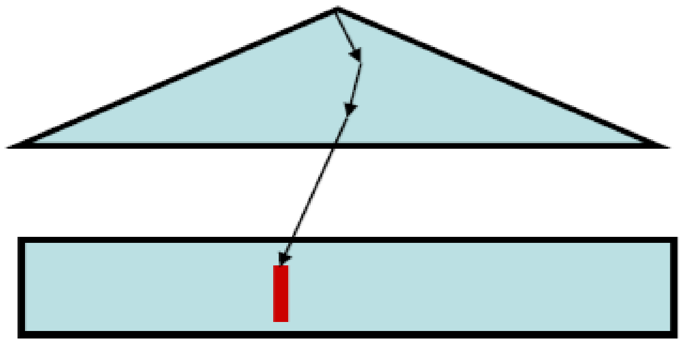
Select *
From Employee
Where Id =102

Create unique Index **index1**
On Employee
(Id Asc)

Table scan



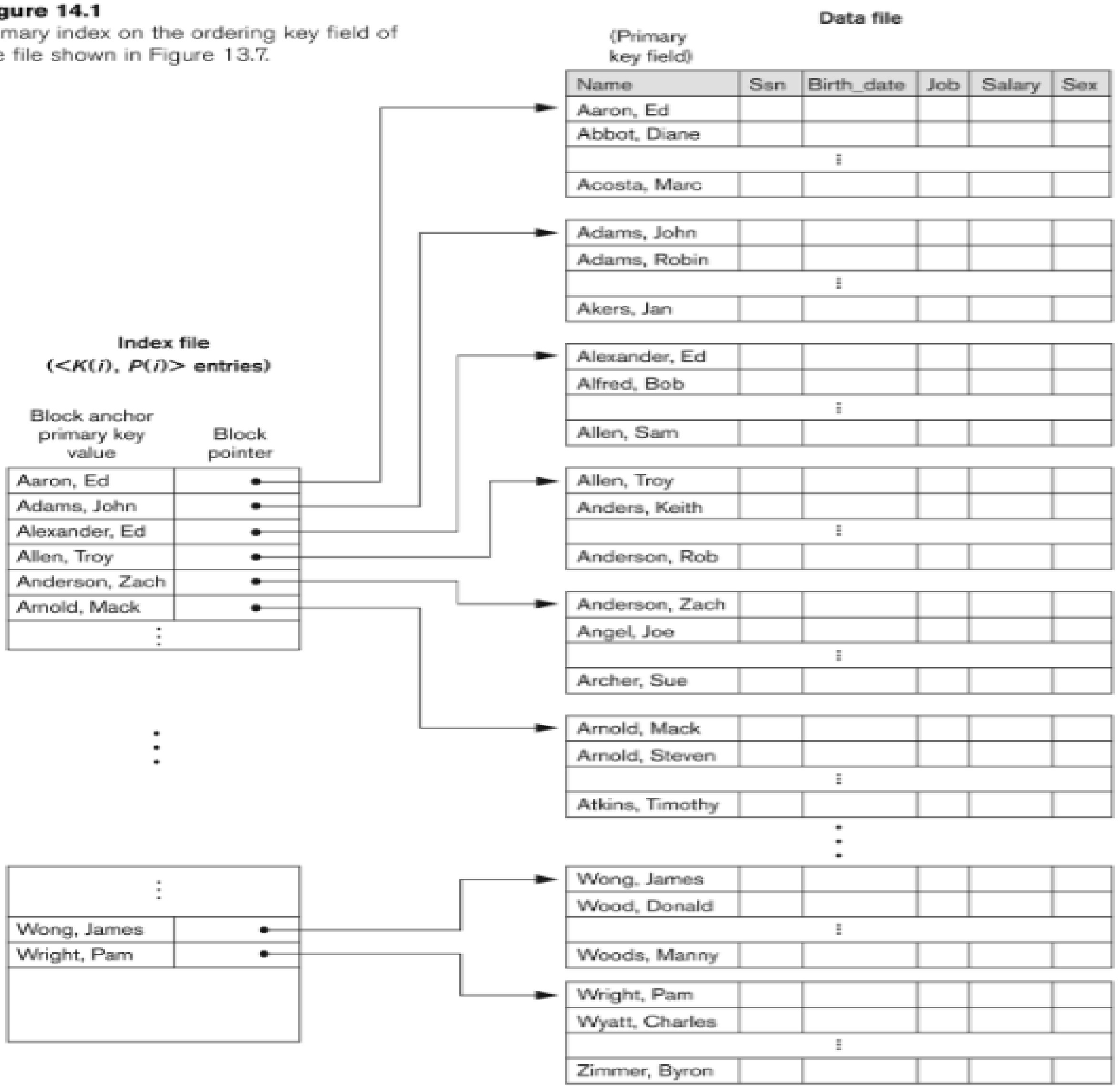
Index scan



Primary Index

Primary Index nondense (sparse) index

Figure 14.1
Primary index on the ordering key field of the file shown in Figure 13.7.



Exercise 1

Consider a disk with block size B=512 bytes.

- A block pointer is P=6 bytes long
- Record pointer is P R=7 bytes long.
- A file has r=30,000 EMPLOYEE records of fixed-length.
- Each record has the following fields:
  - NAME (30 bytes)
  - SSN (9 bytes)
  - DEPARTMENTCODE (9 bytes)
  - ADDRESS (40 bytes)
  - PHONE (9 bytes)
  - BIRTHDATE (8 bytes)
  - SEX (1 byte)
  - JOBCODE (4 bytes)
  - SALARY (4 bytes, real number)

An additional byte is used as a deletion marker.

Exercise 1

(a) Calculate the record size R in bytes.

Record length R = (30 + 9 + 9 + 40 + 9 + 8 + 1 + 4 + 4) + 1 = 115 bytes

(b) Calculate the blocking factor bfr and the number of file blocks b assuming an unspanned organization.

- Blocking factor bfr = floor(B/R) = floor(512/115) = 4 records per bloc
- Number of blocks needed for file = ceiling(r/bfr) = ceiling(30000/4) = 7500

Exercise 1

c) Suppose the file is ordered by the key field SSN and we want to construct a **primary** index on SSN. Calculate:

1. The index blocking factor bfr i (which is also the index fan-out fo)

- Index record size R i = (V SSN + P) = (9 + 6) = 15 bytes
- Index blocking factor bfr i = fo = floor(B/R i) = floor(512/15) = 34

2. The number of first-level index entries and the number of first-level index blocks

- Number of first-level index entries r 1 = number of file blocks b = 7500 entries
- Number of first-level index blocks b 1 = ceiling(r 1 /bfr i) = ceiling(7500/34)= 221 blocks

Databases II: We Learn

- How DB Servers manage:
  - Concurrency control
  - Recovery
  - Indexing
  - Query processing
- Practically How this is Implemented in Simpledb
  - Check the Database Internals
  - Perform some Modifications

What is SimpleDB?

- A mini-database implemented in Java.
- Designed by Edward Sciore for educational purposes.
- Comes in Two Parts:
  - Server Code
  - Client Code:
    - JDBC Interface and Driver Implementation
    - Package for Testing the Database

DB Clients and Servers from your Experience

- Your First DB Connection (in DB I):
  - The Server: SQL Server
  - The Client: Your Code that used sun.jdbc.odbc.JdbcOdbcDriver

to Obtain a DB Connection.

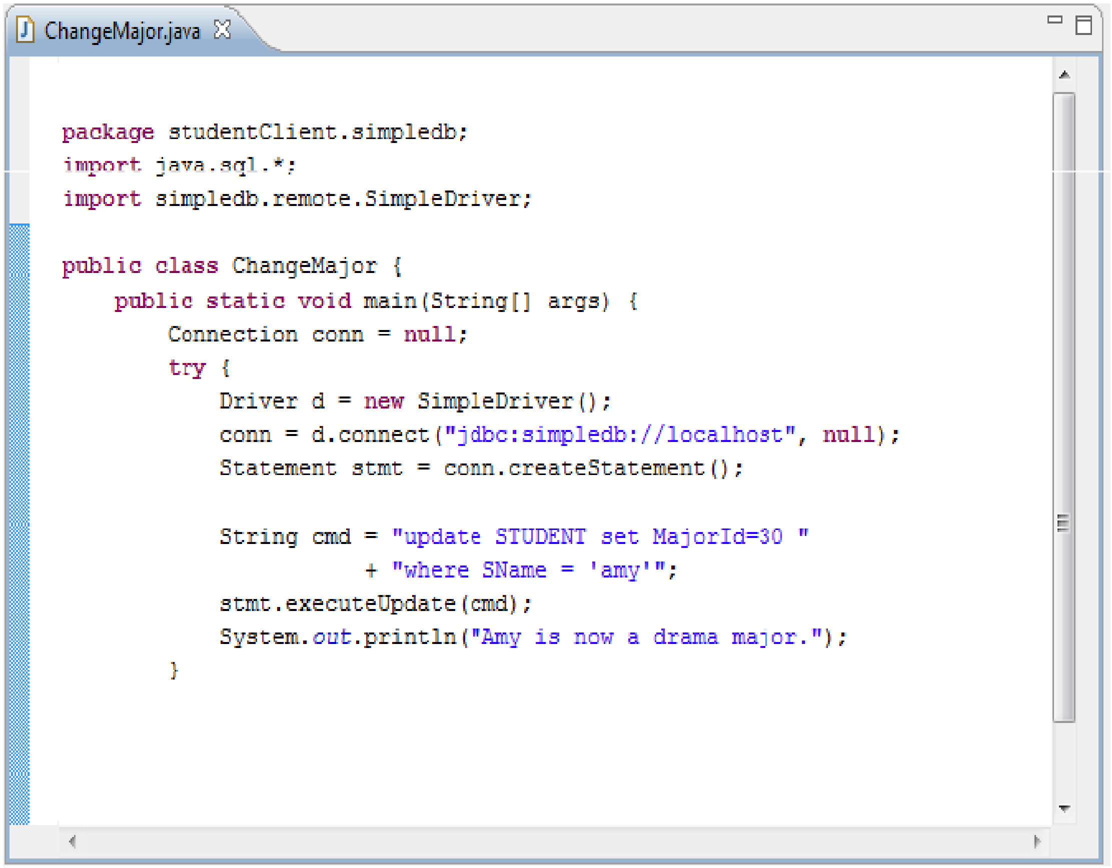
Simpledb: Client Side

Code Samples, in "studentClient" Package, that access the Database using the SimpleDriver.

The SimpleDriver is the Driver used to obtain Connections to SimpleDB

It implements java.sql.Driver

Located under the Package simpledb.remote


ChangeMajor.java

```
package studentClient.simpledb;
import java.sql.*;
import simpledb.remote.SimpleDriver;

public class ChangeMajor {
    public static void main(String[] args) {
        Connection conn = null;
        try {
            Driver d = new SimpleDriver();
            conn = d.connect("jdbc:simpledb://localhost", null);
            Statement stmt = conn.createStatement();

            String cmd = "update STUDENT set MajorId=30 "
                      + "where SName = 'amy'";
            stmt.executeUpdate(cmd);
            System.out.println("Amy is now a drama major.");
        }
```

SimpleDB JDBC interface in Client

JDBC classes manages the transfer of data between a Java client and a DB server.

JDBC package java.sql defines the interfaces:

- Driver
- Connection
- Statement
- ResultSet

The standard JDBC interfaces have a large number of methods.

SimpleDB comes with its own version of these interfaces, with small methods.

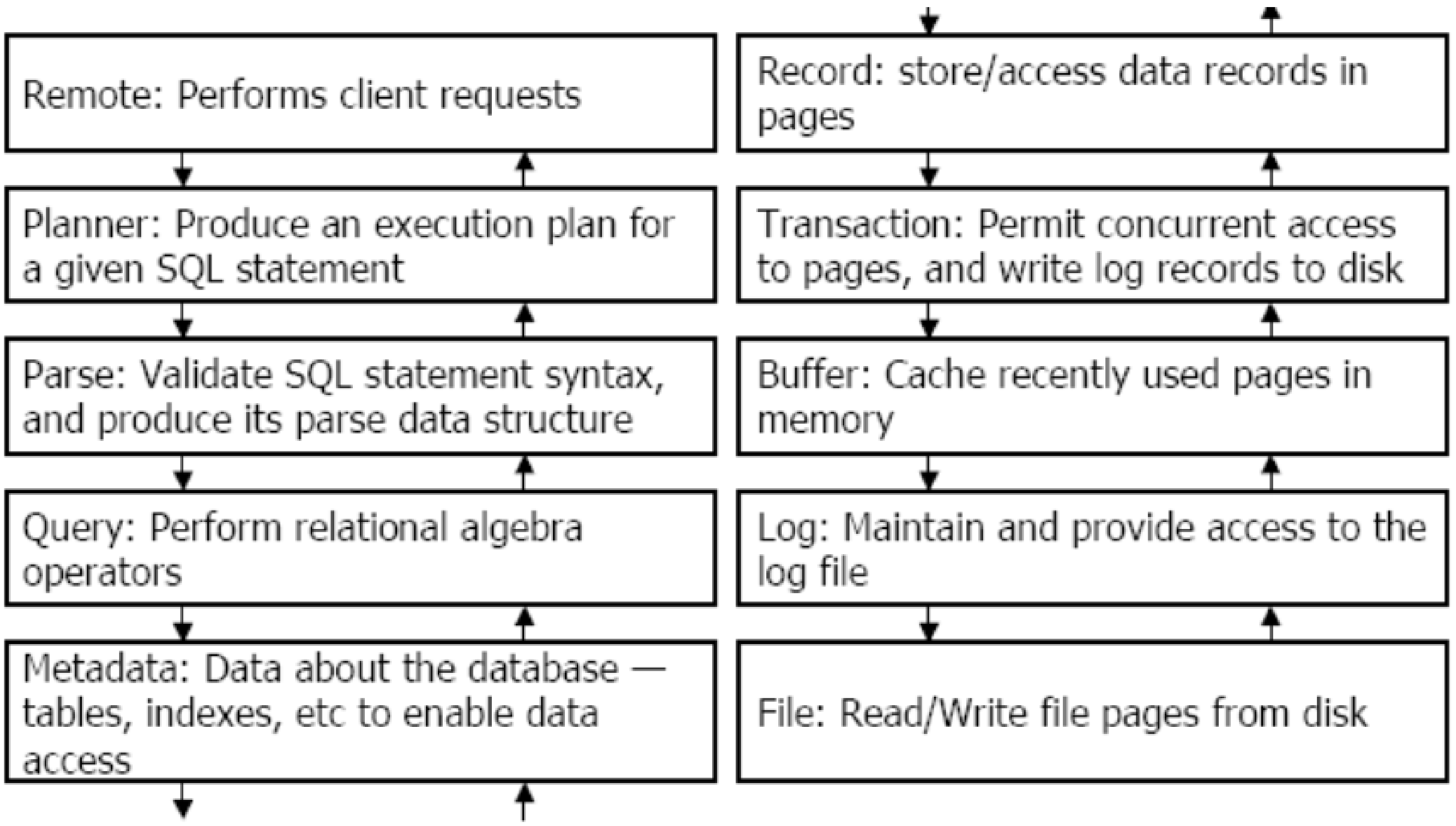
SimpleDB server side

-Comprises most of the SimpleDB code.

-Ten layered components.

-Each component uses the services of the components below it and provides services to the components above it.

SimpleDB server side


Remote: Performs client requests
Planner: Produce an execution plan for a given SQL statement
Parse: Validate SQL statement syntax, and produce its parse data structure
Query: Perform relational algebra operators
Metadata: Data about the database — tables, indexes, etc to enable data access
Record: store/access data records in pages
Transaction: Permit concurrent access to pages, and write log records to disk
Buffer: Cache recently used pages in memory
Log: Maintain and provide access to the log file
File: Read/Write file pages from disk

How the Client finds the Server and Communicates with it?

- SimpleDB handles the Client-Server communications through the Remote Method Invocation (RMI) technology.
- RMI is a Java API that performs the OO equivalent of remote procedure calls. wikipedia

Client Server Communication

RMI Architectural Components

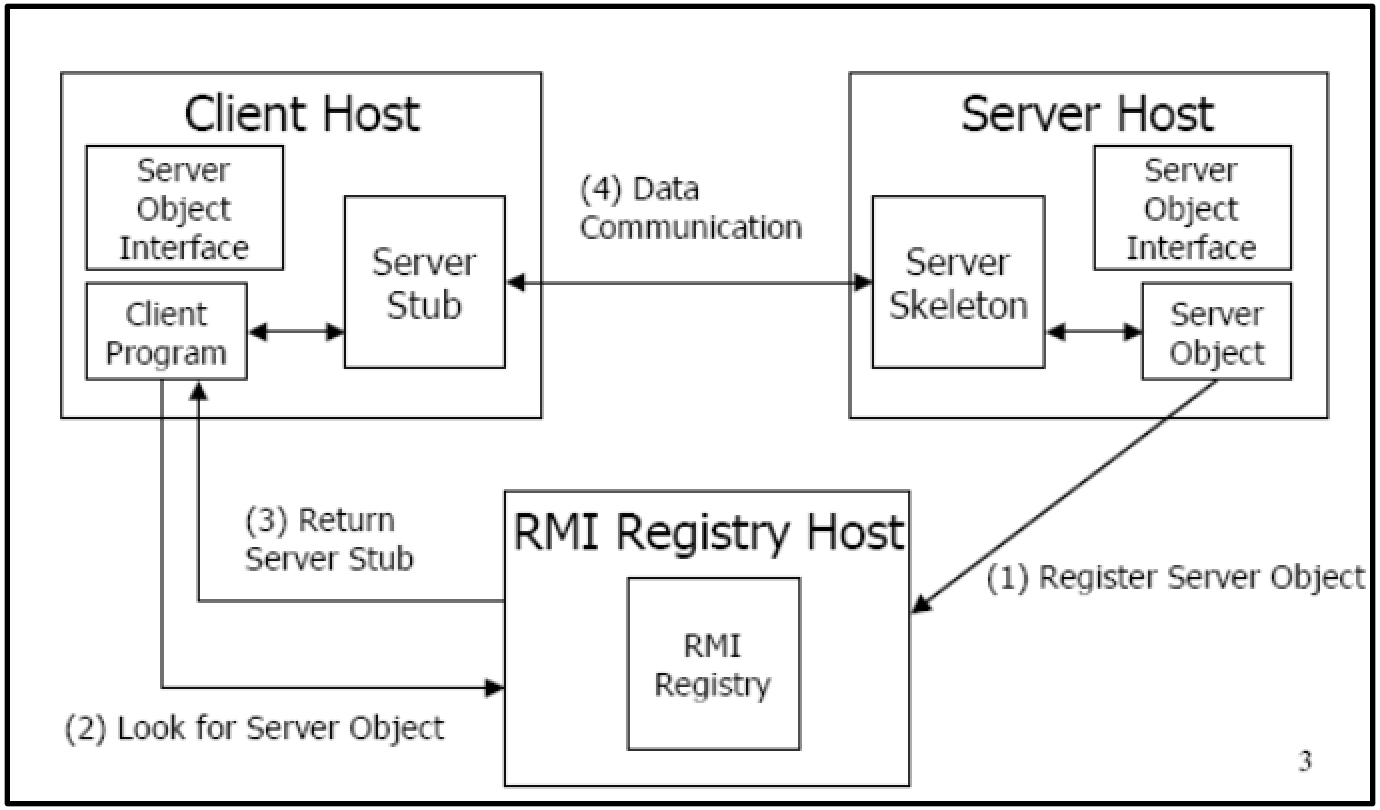
It has a Remote interface

- Extends java.rmi.Remote interface
- Contract between caller of the remote method (RMI client) and remote object (RMI server)

Has two implementing class:

- RMI Stub class who's object live in client.
- Remote implementation class who's object live in the server.

When the client calls the method from a stub object, the method call is sent cross the network to the server and executed there by the corresponding remote implementation object



1.Remote object is created and stores a stub to it in the RMI Registry.
2.It waits for the Stub to contact it.
3.JDBC client wants to connect to the DB system.
4.It obtains the Stub from the Registry to make the connection.

How to get simpleDB ready ?

1- Download SimpleDB from MET Website.

Please Note: This is not the Original version of SimpleDB. The original version is available here: www.cs.bc.edu/~sciore/simpledb/intro.html

2- Open the three projects (SimpleDB Server, Driver and Client) in Eclipse:

- File / Import / General / Existing Projects into Workspace / Next
- Browse to the root directory of each Project.
- Do that for the three projects.

3- Add SimpleDB Driver to the Client Build path:

- RClick the Client project / Build Path / Configure Build Path
- From the Projects Tab, Add the Simpledb Driver Project

Start SimpleDB Server

4- Start the RMI Registry:

- Run / Run Configurations
- Double click Java Application (at the left side)
- Name the new Configuration "RMI_Registry"
- Run the main of RegistryImpl:

  Check "Include system libraries when searching for a main class" / Search/ Select RegistryImpl / Run

An empty Console entitled RMI_Registry should open, and the Terminate button is Activated (the Red Square at the top of the console); this means the RMI Registry is Running.

Use the Database “studentdb”

5- Register SimpleDB Server to the RMI Registry and Use the Database “studentdb”:

Check Startup.java located under simpledb.server Package; it uses args[0] referring to the Database name, so we need to pass it :

- RClick Startup.java / Run As / Run Configurations
- Double Click Java Application
- In the Program arguments under the Arguments Tab, write “studentdb”
- Run

A new Console entitled Starup should open, and the Terminate button is

Activated; this means the SimpleDB Server is Running.

If the database is new, this should be printed:

```
new transaction: 1
creating new database
transaction 1 committed
database server ready
```

Else, instead of creating new database:

```
recovering existing database
```

simpleDB server packages

- *(SimpleDB.server)*
- simpledb.file
- simpledb.buffer
- simpledb.index
  - simpledb.index.btree
  - simpledb.index.hash
  - simpledb.index.planner
  - simpledb.index.query
- simpledb.log
- simpledb.materialize
- simpledb.metadata
- simpledb.multibuffer
- simpledb.opt
- simpledb.parse
- simpledb.planner
- simpledb.query
- simpledb.record
- simpledb.remote
- simpledb.server
- simpledb.tx
  - simpledb.tx.concurrency
  - simpledb.tx.recovery

simpleDB clients (Studentclient.simpleDB)

- **CreateStudentDB:** creates and populates the student database used by the other clients.
- **StudentMajors:** prints a table listing the names of students and their majors.
- **FindMajors:** prints the name and graduation year of all students in certain major (given as an argument).
- **SQLInterpreter:** it executes single SQL statement.
- **ChangeMajor:** changes the student named Amy to be a drama major.

SimpleDB understands only a limited subset of SQL.

Running the client

- Open the file CreateStudentDB in the studentClient.simpledb package and select “Run As” a java application.
- Turn in the output created by running this application.
- Check out the output on the server console an the output on the client console.

•Table STUDENT created.
•STUDENT records inserted.
•Table DEPT created.
•DEPT records inserted.
•Table COURSE created.
•COURSE records inserted.
•Table SECTION created.
•SECTION records inserted.
•Table ENROLL created.
•ENROLL records inserted.

•The new output on the server console is:

•new transaction: 1
•recovering existing database
•transaction 1 committed
•database server ready
•new transaction: 2
•transaction 2 committed

•new transaction: 36
•transaction 36 committed

SQLInterpreter

- Launch the SQLInterpreter and see if you can write an SQL statement to display Sid, Sname form the STUDENT table.

- Turn in the console output of your session with simpledb to display STUDENTtable.

Output:

```
SQL> select SId, SName from STUDENT;
                 sid sname
----------------
1 joe
2 amy
3 max
4 sue
5 bob
6 kim
7 art
8 pat
9 lee
SQL>
```

Thank you =)