# Tinder System Design

## Layers

### Components

* Storage profile (Images – 5 images per user)
* Recommended match (No of active users)
* Note match
* Direct message

## Storage Profile

How to store images and deal with it. We can store images are **File** or **Blob.**

What database guarantee to store as Blob

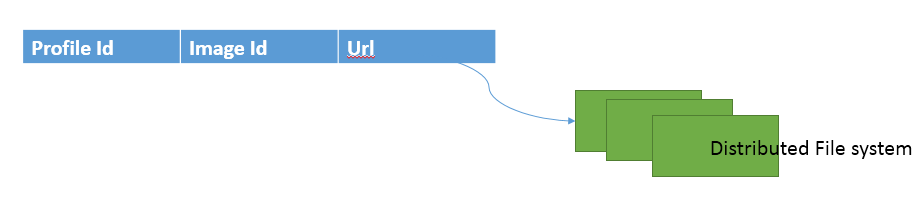
1. Mutability
2. Transaction - ACID
3. Indexes
4. Access control

For the case of images we don’t need mutability every time it requires to update we will create a new image.

For image we may not need transaction properties, we won’t be doing atomic operations on images.

Indexes are good for searching, binary search, for Blob it is useless because we won’t be searching for the content of the blob

We can get access control mechanism for file system



**File system:**

1. Cheaper
2. Faster (Storing large object separately - we can do the same in DB using vertical partitioning)
3. File is static we can easily build CDN for fast access.

## Direct message

For sending message we can use **peer – peer** communication protocol **XMPP**

HTTP is a client – server protocol where client needs to **Poll** the server for messages in a frequent interval which is very inefficient, where we need a protocol which will push the notification or messages.

With every connection id we need to know which user is using this connection. A set of connection is on Gateway service, each of the user needs to talk to other user, we needs to find out where does these user belongs means which connection that user is listening to. It also can be done by Gateway service but to decouple the system we can put the responsibly for maintain connection to another service **Sessions service.**

## Noting recommendation

There can be a **Matcher Service** which will keep the match details and communicate with **Session service** to send message with.

## Recommending people to you

The biggest problem is here to find out who are the people near to you based on the age group, gender we are interested in from million active users.

The **Profile Service** will have all the details with columns **age, gender, location**

We can’t put indexing on all the columns even if we put when we query it will pick up one of the column and search on that (ex. Gender) but we need from that gender again filter on some sort of other range (age group) and then in a particular location. It complete depends on the query optimizer how does it take it. We need to optimize on multiple parameter.

So in this case we can use **No-sql** database like Casandra which are really efficient on these kind of cases where we will replicate the data and based on data we can build the table.

We can use the same concept on relational database and that requires **Shading – Horizontal partitioning.**

### Shading – Horizontal partitioning

In horizontal partitioning we take some property of data and based on some range we direct the data to different location.

Ex: **Name**

**(A – J) – Node 36**

**(K – Z) – Node 54**

Partitioning the data is very useful and one of the way we partition the data is shading. Based on value here we go to some of the node.

For failover scenario we can have **Master – Slave** architecture per partition or multiple master and multiple slaves, so if any of fails other comes up.

Here in either based on the location or a chunk of the location we can move it to a location. So we are discussing about sharding.

## Architecture Diagram

