1: Cassandra

Application = Simple Facebook Profiles

- Facebook sells the service of self-expression via a customizable profile that keeps a structured format.
- The data represents peoples' experiences (photos), thoughts (status updates) and preferences (likes)

Why Cassandra is the best fit

- Cassandra is most suitable for content management. Facebook profiles have a lot of content that is in the form of structured data, such as photo albums and status updates, which are almost like a mini-blogging system. More reasons to use Cassandra include:
- There is a vast population of users with Facebook profiles, so the fact that Cassandra handles data across many servers will be very beneficial. This decentralization feature is also related to scalability, so when new machines are added, service can increase linearly.
- Arelational DBMS is not the best option because it will be slower in terms of read-write time, and not as flexible for storing and managing profile content.
- Using HBase or Mongo would be a very similar experience, but Cassandra's column family and keyspace models are more suited for structured document storage. Hbase would encourage heavy use of Hadoop, which may not be wanted, and Mongo would probably be less scalable and slower.

Column families:

- Notes: the bold words are keys. All values have a timestamp that is not shown for simplicity.

users				
taggr	name	password	email	location
	Tagg Ridler	batman11	tagg@yahoo.com	Boulder,CO
mileyc	name	password	location	
	Miley Cyrus	cantstop	Hollywood, CA	
baracko	name	password	email	
	Barack Obama	potus	bo@whitehouse.gov	

friends with		
taggr	mileyc	
mileyc	baracko	taggr
baracko	mileyc	

status updates			
taggr	body	tags	photo
	I love Miley!	mileyc	39hh083
mileyc	body		
	Where am I?		
baracko	body	photo	
	Bye healthcare!	09guq	

CQL queries

```
INSERT INTO users (name, password, location)
VALUES ("Lana Del Rey", "tropico", "Cony Island, NY")

SELECT COUNT (*)
FROM users
WHERE location = "Hollywood, CA"

CREATE COLUMNFAMILY Users (
        KEY uuid PRIMARY KEY,
```

```
        name text,
        password text,
        email text,
        location text );
CREATE COLUMNFAMILY FriendsWith (
        KEY uuid PRIMARY KEY,
        Friends text );
CREATE COLUMNFAMILY StatusUpdates (
        KEY uuid PRIMARY KEY,
        body text,
        tags text,
        photo blob );
```

2: XML language

Language Definition

- This model is general enough to represent any system that manages profiles with a picture, posts that can contain pictures, and friends or connections to other users with profiles.
- This model supports the `newPost` operation which allows a user to create a new post, and the `suggestedFriends` operation which suggests new connections for users based on the friends of their friends.

```
<?xml version="1.0"?>
<profile>
        <name></name>
        <profilePhoto></profilePhoto>
        <newPost>
                <source></source>
                <tagBox></tagBox>
                <inputText></inputText>
                <inputPhoto></inputPhoto>
        </newPost>
        <posts>
                <post>
                        <time></time>
                        <location></location>
                        <tags>
                                <tag></tag>
                        </tags>
                        <body>
                                <photo></photo>
                                <text></text>
                        </body>
                </post>
        </posts>
        <friends>
                <friend>
                        <name></name>
                        <profilePhoto></profilePhoto>
                        <link></link>
                </friend>
        </friends>
        <suggestedFriends>
                <source></source>
                <inputFriends>
                        <friendID></friendID>
                </inputFriends>
                <userID></userID>
        </suggestedFriends>
</profile>
```

XML Schema

- This does not include types for attributes of operations.
- Photos are represented as a URI (string)

```xml
<?xml version="1.0"? encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="name" type="xs:string"/>
        <xs:element name="photo" type="xs:string"/>
        <xs:element name="source" type="xs:string"/>
        <xs:element name="tag" type="xs:string"/>
        <xs:element name="time" type="xs:dateTime"/>
        <xs:element name="location" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
        <xs:element name="link" type="xs:string"/>
        <xs:element name="userID" type="xs:int"/>
        <xs:element name ="friendID" type="xs:int"/>

        <xs:element name="tags">
                <xs:complexType>
                        <xs:list ref="tag"/>
                </xs:complexType>
        </xs:element>

        <xs:element name="post">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="time"/>
                                <xs:element ref="location"/>
                                <xs:element ref="tags"/>
                                <xs:element ref="photo"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

        <xs:element name="friends">
                <xs:complexType>
                        <xs:sequence>
                                <xs:list ref="friendID"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>

</xs:schema>
```

3: OLAP and Relational DB

Relational Schema

users
userID **PK**
name
email
password
location
time

friends
userID **PK**
friendID **PK FK**

posts
userID **PK FK**
body
time

- Functional Dependencies: all values in each table are functionally determined by the primary key.
- Multivalued Dependencies: the friends table. The presence of a row (userID, friendID) requires another row (userID, friendID) that is the reverse of the first row.

OLTP Queries

Create a new user and automatically make them friends with the original user (MySpace model):

```
INSERT INTO users (name, email, password, location)
VALUES ("Kanye West", "ye@thebomb.com", "yeezus", "Chicago, IL");

INSERT INTO friends VALUES (1, 2);
```

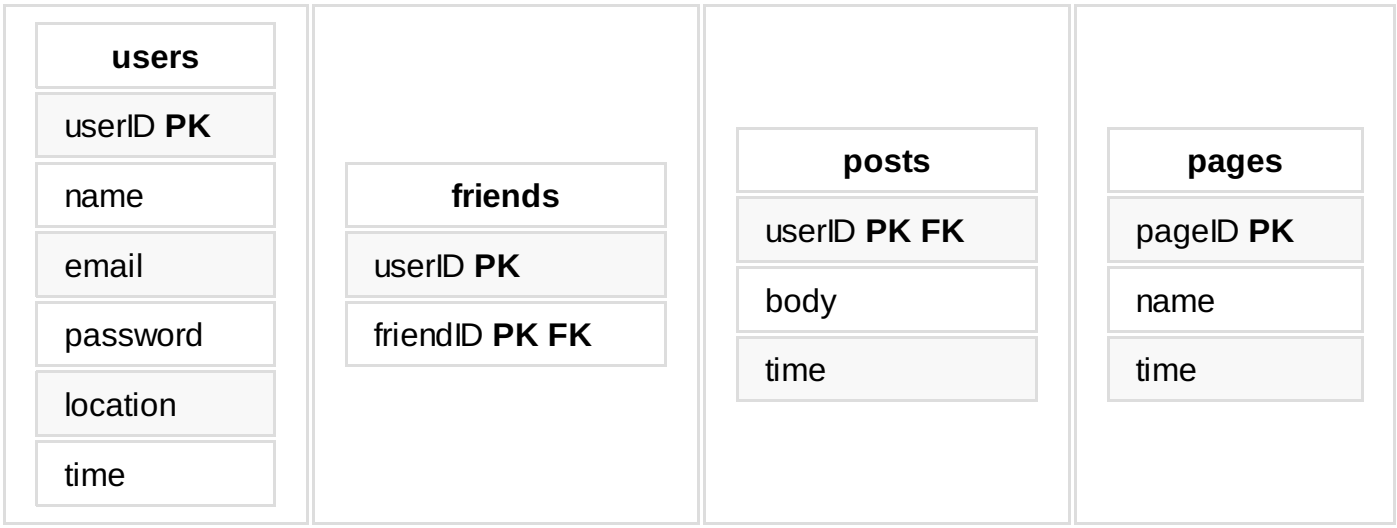
Auser can add the person as a freind who lives in Santa Barbara, CAwith the most recent post:

```
INSERT INTO friends VALUES (1,
(
        SELECT userID
        FROM users u JOIN posts p
                ON u.userID = p.userID
        WHERE u.location = "Santa Barbara, CA"
        ORDER BY p.time LIMIT 1
));
```

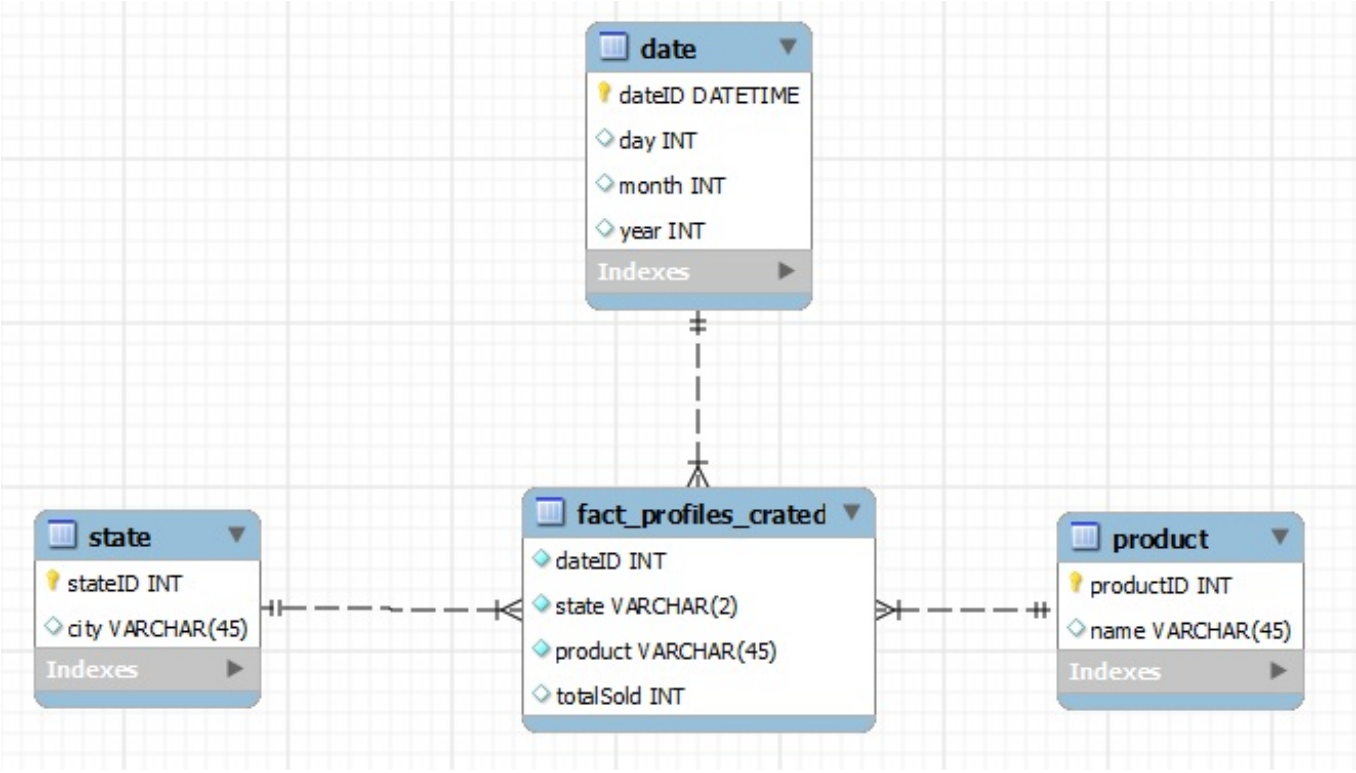
Data Warehousing

Schemas

New relational schema to make this work (I am adding a page table, in reference to facebook pages vs profiles)



Star Schema: for explaination of how it relates to the above schema, see below.



Queries for Trends over Time

- Summarizes "sales" over time by state

```
SELECT dateID, state, SUM(totalSold)
FROM fact_profiles_created
GROUP BY state, dateID
```

- Summarizes "sales" of certain product type over time by state

```
SELECT product, dateID, state, SUM(totalSold)
FROM fact_profiles_created
GROUP BY product, dateID, state
```

- Summarizes "sales" by product type over a particular month in California

```
SELECT product, SUM(totalSold)
FROM fact_profiles_created f NATURAL JOIN date d
WHERE d.month = 8 AND f.state = "CA"
GROUP BY product
```

Explaination of My Data Warehousing

- The sales table in the star schema is aggregating the number of times a new profile or page was created in the application. The state table is storing the information related to the location of the profiles or pages that were created, while the product table is storing whether the thing created was a profile or a page. The date table is keeping track of the timestamp representing the time of creation of a profile or page.
- The aggregation descibed above is done so summary queries can be written to analyze the data relating to the usage of the facebook-like app, and reveal trends over time. This process is very different from online transaction processing (OLTP) because its only purpose is data analysis. The data warehousing system should not modify records in the OLTP system in any way; it should only analyze them. Relatively, the OLTP system should not be used to analyze any of its own data; it should only create, read, update, or delete it.

4: Assertions and Inferences

Modified XML Language:

```
<?xml version="1.0"?>
<profile>
        <home></home>
        <name></name>
        <profilePhoto></profilePhoto>
        <newPost>
                <source></source>
                <tagBox></tagBox>
                <inputText></inputText>
                <inputPhoto></inputPhoto>
        </newPost>
        <totalPosts></totalPosts>
        <posts>
                <post>
                        <time></time>
                        <location></location>
                        <tags>
                                <tag>
                                        <friendID></friendID>
                                        <taggedBy>
                                                <userID></userID>
                                        </taggedBy>
                                </tag>
                        </tags>
                        <body>
                                <photo></photo>
                                <text></text>
                        </body>
                </post>
        </posts>
        <friends>
                <friend>
                        <name></name>
                        <profilePhoto></profilePhoto>
                        <link></link>
                </friend>
        </friends>
        <suggestedFriends>
                <source></source>
                <inputFriends>
                        <friendID></friendID>
                </inputFriends>
                <userID></userID>
        </suggestedFriends>
</profile>
```

RDF Triples / Assertions

- a post by mileyc should be tagged by mileyc
    - `www.myface.com/mileyc/posts/1432324 <taggedBy> www.myface.com/mileyc`
- one of baracko's suggested friends should be taggr
    - `www.myface.com/baracko <suggestedFriends> www.myface.com/taggr`
- any profile's home route should be the home page
    - `www.myface.com/profile/ <home> www.myface.com`
- a new post's time should be the current time when it is created
    - `www.myface.com/lanar/posts/new <time> 1/1/2014`
- a new post's location should be the location of the person who owns the post
    - `www.myface.com/kanyew/posts/new <location> Chicago, IL`
- taggr's profile photo shouuld be a photo he owns and in his 'profile' route
    - `www.myface.com/taggr <profilePhoto> www.myface.com/taggr/photos/profile`
- the name on someone's profile should match their actual identity
    - `www.myface.com/mileyc <name> Miley Cyrus`
- the input photo for a new post should come from someone's hard drive
    - `www.myface.com/lanar/posts/new <inputPhoto> C:\Users\LanaDelRey\Photos\`
- taggr's friends should include mileyc

    - `www.myface.com/taggr <friends> www.myface.com/mileyc`
- the number of posts someone has made should be an integer representing their total number of posts
    - `www.myface.com/mileyc <totalPosts> www.metrics.com/myface/posts/mileyc`

Inferences

- All of the left values in the triples are a route in the app, while all the right values are either a string, local file, or a different route within the app. Thus I can infer that many of the data elements in profiles do not rely on outside web services.
- All of the right values that relate to documents saved to the database are routes within the app. Thus I can infer that many of the documents that make up a profile are contained in the app's document database.
    - A photo that is part of a new post that hasn't been saved yet is a file on someone's hard drive, but a profile photo that has already been saved as a profile document has a route contained with in the app's domain.

5: Distributed Databases

Schemas

Original Profile Schema

users
userID **PK**
name
email
password
location
time

friends
userID **PK**
friendID **PK FK**

posts
userID **PK FK**
body
time

- Functional Dependencies: all values in each table are functionally determined by the primary key.
- Multivalued Dependencies: the friends table. The presence of a row (userID, friendID) requires another row (userID, friendID) that is the reverse of the first row.

Pictures App Schema (like Instagram)

albums
ID **PK**
ownerID **FK**
description
time_created

photos
ID **PK**
albumID **FK**
caption
photo_location
time_taken

users
ID **PK**
name

- Functional Dependencies: all values in each table are functionally determined by the primary key.
- Multivalued Dependencies: none

Q&A App Schema

answers

questions

ID **PK**
questionID **FK**
answererID **FK**
answer_text
time_answered

ID **PK**
askerID **FK**
question_text
category
time_asked

users
ID **PK**
points

- Functional Dependencies: all values in each table are functionally determined by the primary key.
- Multivalued Dependencies: none

New Schema built with Primitives

Please see attached hand-drawn schema

6: Ambient Intelligence

Last summer, I went to Italy and had to ride a lot of trains. I'm going to describe a higher-tech ticketing system than the one they currently have, in the same train stations that I visited.

Description of System

- This system would be located in a train station
- The first part of the system would be touch-screen ticketing kiosks. The user would purchase their ticket from at this kiosk and their thumbprint would be scanned, as a means of storing their identity. The second part of the system would be a thumbprint reader on the actual train, which users would have to use in order to validate their ticket and board the train.
- The system adapts to a diverse population by requiring a standardized set of information to purchase a ticket. Everyone who wants to use this system should have a means of purchasing a train ticket. If a user did not have thumbprints, a backup identification method such as loading their identity into the system from a credit card should be possible, but I'm not going to worry about such a case for this example.
- The hardware involved would be thumbprint readers, credit card scanners, touch-screen computers, and web servers.

Web Connectivity

- Aweb connection would be valuable to the users because the system could then email them reciepts or iteneraries.
- Users could also have the option of buying a ticket on a web-connected, fingerprint-enabled devise such as an iPhone 5S, and not have to go early to the station to use the kiosks.
- Aweb connection would be valuable to the implementers because it could provide information about the customers, which could lead to information helpful for marketing or other analytical purposes.
- Aconnection to the system's servers that host information about routes, available tickets, and purchases is necessary for the system's basic functions.

Appropriate Database

The best database to back this system would be a relational database system such as MySQL. This system's primary purpose is transactions, so the locks and various levels of transaction protocalls that MySQL supports would be necessary. One transaction would be when a user purchases a ticket. This would need to be stored in the database without any uncertanty that the row(s) were written. The other transaction would be when a user checks in to the actual train. This transaction would require less certainty and more speed than purchases, so a lower isolation level should be used. Since there is not much else going on other than transactions, other types of databases would not be as valuable for this system.

7: Comparisons

Database Name	Structure	"Query" Method	Transactions	Focus
Neo4J	Graph	Cypher Query Language	full ACID support	ease for common graph-like data

Mongo	BSON/Document	MongoDB CRUD Operations	Nested Documents	large-scale document storage
MySQL	Tables	SQL	full rollback / ACID support	transactions, routine querying
Cassandra	Column Families	CQL	3rd Party additions required	Big Data, masterless decentralization, scalability
Hbase	Colums	Map-Reduce/Hadoop	mostly ACID support (locks)	scalability, clusters, big data