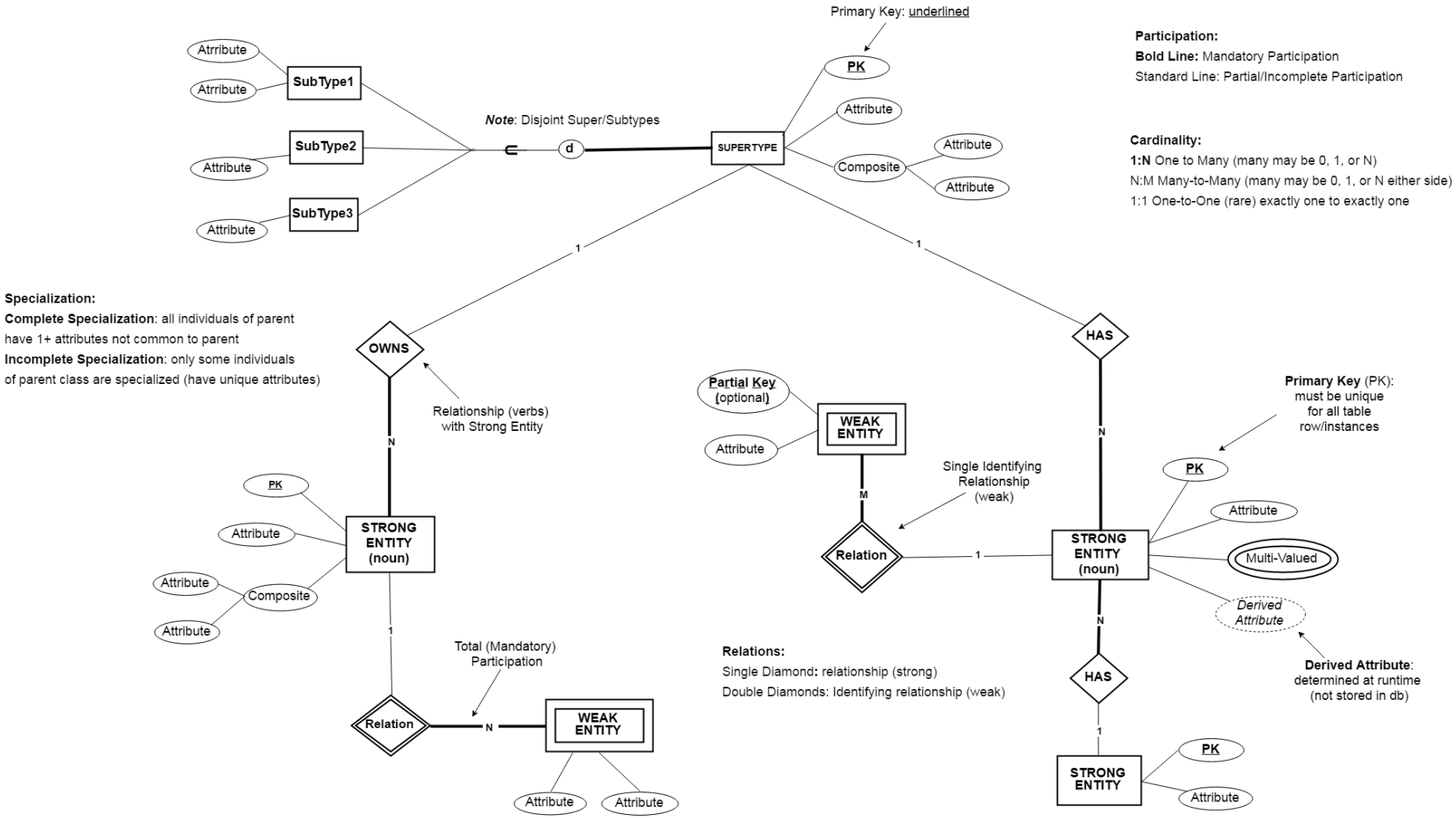
Phase 1 EER | CS 6400 - Fall 2017 | **Team 001**

Enhanced Entity-Relationship (EER) Diagram: (format example: crop/fit to one page)



Participation:
Bold Line: Mandatory Participation
Standard Line: Partial/Incomplete Participation

Cardinality:
1:N One to Many (many may be 0, 1, or N)
N:M Many-to-Many (many may be 0, 1, or N either side)
1:1 One-to-One (rare) exactly one to exactly one

Specialization:
Complete Specialization: all individuals of parent have 1+ attributes not common to parent
Incomplete Specialization: only some individuals of parent class are specialized (have unique attributes)

Relations:
Single Diamond**:** relationship (strong)
Double Diamonds: Identifying relationship (weak)

Single Image:format to **fit one-page *landscape***:clearly label attributes, cardinality (1:N), strong vs. weak entities, total(bold) vs. partial participation links, underline keys. Distinguish between double ovals/rectangles/diamonds for attributes, entities, and relations respectively.
Lastly, condense attributes around entities, make sure lines do not cross over other attributes (may unintentionally look like composite attribute).

ENTITY type (**nouns**) and **relationship** type (**verbs**) names are UPPERCASE letters, Attributes with Uppercase first letter
Primary Key (PK) underline, Parital Key (weak) dashed underline

Total/Complete specialization (existence dependency) demands that every entity in the superclass belong to some subclass- represented with a bold line link/connection
Partial/Incomplete specialization(optional all or none) rule allows an entity to not belong to any of subclasses- represented with a **single line** link/connection

ⓓ **Disjoint**:(exclusive either/or but not both) an individual of the parent class may be a member of ***exactly one*** specialized subclass.

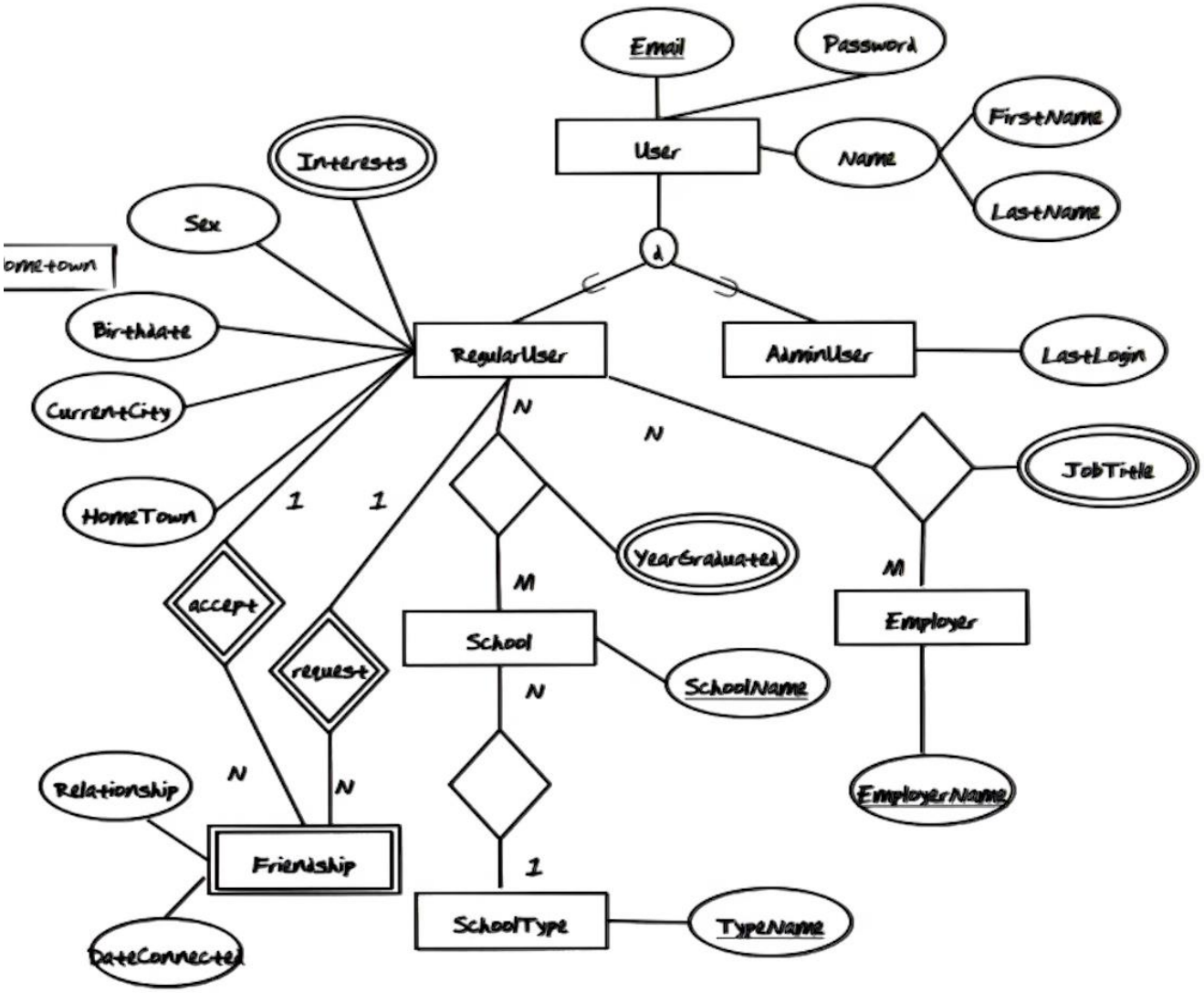
ⓞ **Overlapping**: (both are possible) individual of parent class may be a member of ***more than one*** of specialized subclasses.

∪ **Union**: joining of two super types/classes

Revised: 6/27/2017

Phase 1 EER | CS 6400 - Fall 2017 | **Team 001**

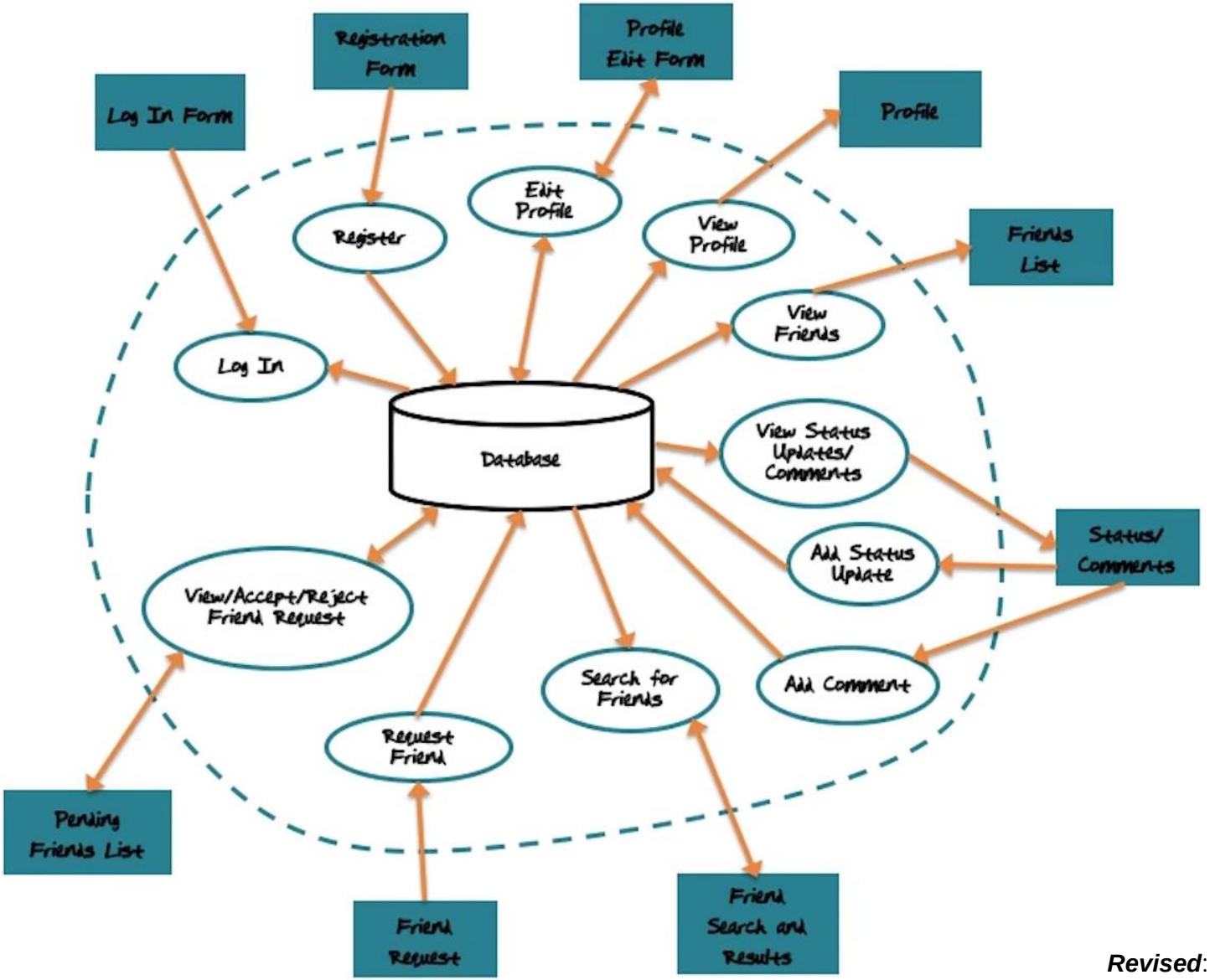
EER Submission details:
Single Image: format to **fit one-page *landscape***, clearly label attributes, cardinality (1:N), strong vs. weak entities, total (bold) vs. partial participation links, underline keys. Distinguish between double ovals/rectangles/diamonds for attributes, entities, and relations respectively. Lastly, condense attributes around entities, make sure lines do not cross over other attributes (may unintentionally look like compound attribute).
Do NOT include Surrogate Keys for Phase 1, these will be added for Phase 2 submission.



Revised: 6/27/2017

Phase 1 IFD | CS 6400 - Fall 2017 | **Team 001**

Single Page Image: format **to fit one-page *landscape***: include forms, directed arrows, dedicated tasks, DB, and system boundary dashed line. All tasks present on the IFD should be on the report in the task decmposition section. Consider the 'forms' to be like webpage UI data entry forms. See notes on whether to use dedicated tasks or combine into a "mother' task. Follow the naming convention for forms/tasks provided in the requirements document.



Revised: 6/27/2017

Phase 1 Report | CS 6400 | **Team 001**

Phase 1 Template:

Please use this template as a guide for what to submit for phase 1 report. Include team number, class, and current term in the header for all pages (show above). Also include a **table of contents** (TOC) as the first page of your '*team###_p1_report.pdf*' with working links to navigate the task document quickly. Use consistent indentation, font sizes, and bullets to model control flow. Lastly, include consistent markup to convey meaning (bold, italic, underline, etc.) with an example shown below:

Table of Contents:

Datatypes, Constraints, Task Decomposition (TD), and Abstract Code (AC) for each task working **navigation links**.

Data Types:

Include data types for all attributes in your EER diagram. These should be simple datatypes such as integer, string, Boolean, and float for single-value attributes, and for multi-value attributes, you should use list, collection, or single-dimension array with the individual element data type properly reflecting the data being stored. You should also **indicate if NULLs** are allowed for the attribute.
Make sure to recognize when to appropriately store certain kinds of data (such as phone numbers, quantities, whole numbers, floating point, set/enums.) In this phase 1, **do <u>not</u> create database tables** or any kind of formal database schema, so **do <u>not</u> include any surrogate keys**, foreign keys, or any data types that represent them. In phase 3, you may optionally implement 'best-practice' security safeguards (password hashing, SQL injection prevention, etc.), but you should not include them for phase 1 or phase 2 submissions.

Business Logic Constraints:

Include applicable *business logic constraints* which cannot be represented sufficiently in the EER diagram.

Task Decomposition:

Include the rules of thumb outlined in the lectures for each task and determine if a mother task is needed (lock types, enabling conditions, frequency, schemas, indices, consistency, and subtasks). All names of tasks present on the IFD should match those found on the TD. Please follow the report and screen names provided in the requirements document, so TAs can easily follow your logic.
All task/document names present on your IFD should match those found on the report.pdf.

Per **Leo Mark**: "*The rules of thumb for task decomposition give you an indication of whether to sub-divide or not. You stop the decomposition when no more decomposition is called for by the rules. You need a mother task if the sub-tasks need to be sequenced, all executed together as an atomic unit*."
Per **Jay Summet**: "*If your task has no sub-tasks you would only have the single oval in the IFD and the abstract code in the report. If your tasks do have subtasks, then you show how it breaks down and show the abstract code for each subtask in the report.*"

Abstract Code:

Your abstract code should handle basic **data validation** (example: user cannot enter string for an int field) and basic **error handling** (example: user types in wrong password what happens?). Consider the IFD 'forms/documents' to be like web-page user interfaces where the user clicks on buttons or types input fields. <u>Walk us through how you get to each task and how you leave each task to get to the next one</u>. Include consistent markup to convey meaning (**bold**, *italic*, underline, etc.). Include the words *'task'*, '*form*', and '*button*' (or similar) after every use in your abstract code so there is no confusion on your team's intent.

Markup something similar to:

<u>Bold Underline</u>: <u>Form</u>	Example: **<u>Main Menu</u>** form
Bold Italics: Buttons	Example: ***Save*** button
Bold*:* Task	Example: **Login** task
Italics: Form Input Fields:	Example: *username*, *price*, *date*, etc.

Please do not put TD and AC side-by-side, TD before AC sections in order are allowed.
For your p1_report.pdf submission, please follow this order:

- Table of Contents (working links to other parts of document)
- Data Types (non-SQL)
- Business Logic Constraints (rules which cannot be easily represented in the EER)
- TD/AC for each Task including oval diagrams for each task on remaining pages of report

Lastly, use this as a guide so the grading process will be faster. Feel free to post a private question: (Team### + Instructors) if you need further clarification for your project.
Thank you in advance,
CS6400 Instructional Team

Data Types:

User

Attribute	Data type	Allow Null
username	String	Not Null
email	String	Not Null
password	String	Not Null
first_name	String	Not Null
middle_name	String	Not Null
last_name	String	Not Null
interests	List<String>	NULL
…	…	…

Friendship

Attribute	Data type	Allow Null
email	String	Not Null
friend_email	String	Not Null
relationship	String	Not Null
date_connected	Date	NULL
…	…	…

Granted the formal definition of your tables will be done during the EER-to-Relational mapping for **phase 2. Until then just *approximate* the table names needed for read/write locks on the TD. (e.g. 'User', 'Tool', etc.)

Please do **NOT** include surrogate keys or foreign keys in **phase 1** (e.g. 'user_id', 'tool_id', etc.), feel free to reference them unique instances for phase 1 as 'tool-number', 'confirmation number', etc.

Business Logic Constraints:

GTOnline User

- Users who are new to GTOnline must register first.
- Users who have an existing GTOnline account will not be able to register.
- Both users must send friend requests to each other and both requests must be accepted.
- …

Login

Task Decomp

Lock Types: Read-only on RegularUser
Number of Locks: Single
Enabling Conditions: None
Frequency: Around 200 logins per day
Consistency (ACID): not critical, order is not critical.
Subtasks: Mother Task is not needed. No decomposition needed.

Abstract Code

- User enters *email* and *password* into input fields.
- If data validation is successful for both *email* and *password* input fields, then:
  - When ***Enter*** button is clicked:
    - If User record is found but *password* is incorrect:
      - Go back to **Login** form, with error message.
    - Else:
      - Go to **View Profile** form.
- Else *email* and *password* input fields are invalid, display **Login** form, with error message.

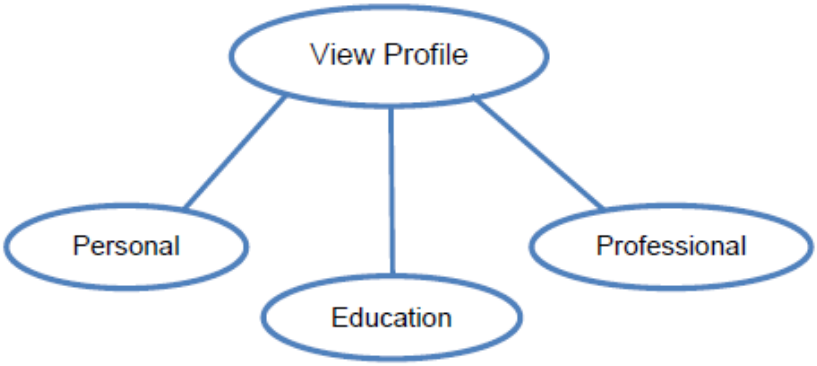
Main Menu / Navigation Bar

Abstract Code

- Show "***View Profile***", "***Edit Profile***", ***"View Friends***" ", ***"Search for Friends***" ", ***"View Requests***" ", ***"View Status Updates***", and "***Log out***" tabs.
- Upon:
  - Click ***View Profile*** button- Jump to the **View Profile** task.
  - Click ***Edit Profile*** button- Jump to the **Edit Profile** task.
  - Click ***View Friends*** button- Jump to the **View Friends** task.
  - Click ***Search for Friends*** button- Jump to the **Search for Friends** task.
  - Click ***View Requests*** button- Jump to the **View Requests** task.
  - Click ***View Status Updates*** button- Jump to the **View *Status Updates*** task.
  - Click ***Log Out*** button- Invalidate login session and go back to the **Login** form.

View Profile

Task Decomp



Lock Types: 3 read-only lookups of Personal, Education, and Professional information for a RegularUser
Number of Locks: Several different schema constructs are needed
Enabling Conditions: All 3 are enabled by a user's login or a friend's lookup
Frequency: Low- All 3 have the same frequency
Consistency (ACID): is not critical, even if the profile is being edited by the user while a friend is looking at it.
Subtasks: All tasks must be done, but can be done in parallel. Mother task is required to coordinate subtasks. Order is not necessary.

Abstract Code

- User clicked on ***View Profile*** button from **Main Menu**:
- Run the **View Profile** task: query for information about the user and their profile where $UserID is the ID of the current user using the system from the HTTP Session/Cookie.
  - Find the current User using the User.email; Display users first and last name;
  - Find the current RegularUser using the User Email; Display RegularUser Gender, Birthdate, Current City, HomeTown.
  - Find and display the current user interests;
  - Find each School for the RegularUser:
    Display School Name and Years Graduated;
    Find School Type;
    Display SchoolType Name;
  - For each Employer for the RegularUser:
    Display Employer Name and Job Title;

  When ready, user selects next action from choices in **Main Menu**

…etc.

***Supplemental* Test Cases:** (not required to include with report.pdf submission)

TC#	*Area*	*Test Case Description*
L000	**Login**	
L001		Login page entry (username, password) credentials
L002		Valid credentials (username, password) accepted
L003		Reject invalid (username, password) when user does not exist
L004		Reject invalid (username, password) when user exists, password invalid
V000	**View Profile**	
V001		User information (username, password, full name, email) stored correctly
V002		User information (username, full name, email) displayed correctly
V003		User password not displayed
…		…

It is recommended teams work through appropriate test cases throughout the project to ensure project requirements are met.