

1) *Introduction*

I have chosen for my application to implement an online recruiting system for use by the CWRU Varsity Swimming Team. Traditionally recruiting information has been kept on postcards that are mailed to high school swimmers and returned to the coach. Clearly there is the capability to modernize this system with a database driven web application. Such an application would provide a centralized place where information could be stored, as well as facilitate easy matching of recruits and recruiters. Potential recruits would also be able to enter their information online and have it immediately accessible in the system.

2) *Application Requirements Specifications*

This application will have three main functions: allow potential recruits to enter their information into the database, allow current swimmers (recruiters) to view this recruit information and contact them, and enable easy matching between recruits and ideal recruiters. There will be three types of users to the system: recruits, recruiters, and coaches (administrators).

Requirements by user type

Recruits: Enter information into the system, modify information they have already entered

Recruiters: Search the database for recruits, view their own “best match” recruits, add recruits, send messages to each other and the coaches, and record when they contact a recruit

Coaches (administrators): All abilities of the recruiters, modify recruit information, add new recruiters, and have a wider variety of “quick searches” to enable easy matching of recruits to recruiters

Security is an important consideration for this application. Although there are no credit card numbers being exchanged or anything like that, it is still important for the sake of confidentiality that recruit information is only available to authorized individuals, the coaches and recruiters. The application will need to implement some sort of login system to ensure that data remains secure.

3) *Database Requirements Specifications*

The database for this application has the task of storing all the data that the application makes available. All information about recruits and recruiters, such as contact information, strokes they swim, and academic information, will be stored in the database. It will also be used for user authentication, as each user will have a password for security purposes. Also maintained will be a list of high schools that either recruits come from or recruiters went to. This will help maintain consistency in searching.

The database will handle many types of transactions. Whenever the application is started, it will be necessary to authenticate the user, so the database must be queried to determine access. Each user's main page will likely contain a list of messages for that user, so another query will have to retrieve the messages for the user. Searching the database will be done with several stored procedure type queries utilizing .NET's SqlCommand class. More advanced searching will need to be done with dynamic SQL generated on the fly. The database will also see transactions to add and modify recruit and recruiter information.

4) *E-R Database Model*

See Appendix A for the E-R model diagram.

Entities

RECRUIT(RecruitID: int, Name: Composite, Address: string, Phone: string, Email: string, GradYear: int, Gender: char, SSN: string, Strokes: string, Academics: Composite)

USER(UserID: int, Email: string, Phone: string, Gender: char, Password: string, Strokes: string, Hometown: string)

SWIM(Stroke: string, Distance: int, Meet: string, Time: real, Date: date)

HIGH_SCHOOL(HighSchoolID: int, Name: string, Address: string, Phone: string)

COUNSELOR(Name: string, Phone: string, Email: string)

MAJOR(MajorCode: string, Description: string)

MESSAGE(Message: string, URL: string)

Relationships (cardinality is listed here, because the diagram became too cluttered and hard to read when I put it in there)

COUNSELS: relationship between RECRUIT (cardinality 1, partial participation) and COUNSELOR (cardinality 1, total participation). Every recruit doesn't necessarily

have a counselor (this is optional information), but every counselor in the database is associated with a recruit.

SWAM: relationship between RECRUIT (cardinality 1, partial participation) and SWIM (cardinality N, total participation). Every recruit doesn't have to enter swim information, but all swim information recorded is associated with a recruit

WORKS_AT: relationship between COUNSELOR(cardinality N, full participation) and HIGH_SCHOOL(cardinality 1, partial participation). All counselors in the database are associated with a high school, but not all high schools have counselors associated with them.

ATTENDS: relationship between RECRUIT(cardinality N, total participation) and HIGH_SCHOOL(cardinality 1, partial participation). All recruits in the database have a high school associated with them, but not all high schools necessarily have a recruit associated with them.

ATTENDED: relationship between USER(cardinality N, partial participation) and HIGH_SCHOOL(cardinality 1, partial participation). Users do not have to specify a high school, and every high school does not necessarily have a use associated with it.

STUDIES: relationship between USER(cardinality N, partial participation) and MAJOR(cardinality N, partial participation)

MIGHT_STUDY: relationship between RECRUIT(cardinality N, partial participation) and MAJOR(cardinality N, partial participation)

SENT_TO: relationship between MESSAGE(cardinality N, partial participation) and USER (cardinality N, partial participation)

There are several integrity constraints that this database must be able to handle. From the SQLServer standpoint, it is necessary to ensure that foreign keys actually exist in the location where they are a primary key. This is handled automatically by SQLServer and does not require any additional code. In several cases (phone numbers, social security numbers) it is necessary to have the data in a specific format. For this I intend to maintain data integrity using code I have previously developed for other applications that format social security numbers and phone numbers before insertion/update.

5) *Relational Database Model*

To transform the E-R model into a relational model, I attempted to change the entities into tables, and model the relationships with primary key/foreign key relationships.

Tables

RECRUITS(RecruitID: int, FirstName: string, LastName: string, Email: string, Address: string, City: string, State: string, ZIP: int, Phone: string, HighSchoolID: int, GradYear: int, Gender: char, Notes: string, SSN: string, SAT: int, ACT: int, GPA: real, ClassRank: int, Back: Boolean, Breast: Boolean, Fly: Boolean, FreeSprint: Boolean, FreeMid: Boolean, FreeDistance: Boolean, IM: Boolean, Diving: Boolean, AdmissionsContacted: Boolean, Applied: Boolean, Accepted: Boolean, Deposit: Boolean, Confirmed: Boolean)

USERS(UserID: int, FirstName: string, LastName: string, Email: string, Phone: string, UserType: string, Hometown: string, State: string, HighSchoolID: int, Back: bool, Breast: bool, Fly: bool, FreeSprint: bool, FreeMid: bool, FreeDistance: bool, IM: bool, Diving: bool)

SWIMS(RecruitID: int, Date: date, Distance: int, Stroke: string, Time: real, Meet: string)

MAJORS(MajorID: string, Description: string)

HIGHSCHOOLS(HighSchoolID: int, Name: string, City: string, State: string, Address: string, Phone: string)

COUNSELORS(RecruitID: int, HighSchoolID: int, FirstName: string, LastName: string, Phone: string, Email: string)

RECRUITSMAJORS(RecruitID: int, MajorID: string)

USERSMAJORS(UserID: int, MajorID: string)

CONTACTS(RecruitID: int, UserID: int, Date: date, Method: string, Notes: string)

MESSAGES(UserID: int, Date: date, Message: string, URL: string)

Relationship constraints (written in pseudo-SQL)

RECRUITS foreign key HighSchoolID REFERENCES

HIGHSCHOOLS(HighSchoolID)

USERS foreign key HighSchoolID REFERENCES HIGHSCHOOLS(HighSchoolID)

COUNSELORS foreign key RecruitID REFERENCES RECRUITS(RecruitID)

COUNSELORS foreign key HighSchoolID REFERENCES

HIGHSCHOOLS(HighSchoolID)

RECRUITSMAJORS foreign key RecruitID REFERENCES RECRUITS(RecruitID)

RECRUITSMAJORS foreign key MajorID REFERENCES MAJORS(MajorID)

USERSMAJORS foreign key UserID REFERENCES USERS(UserID)

USERSMAJORS foreign key MajorID REFERENCES MAJORS(MajorID)

CONTACTS foreign key RecruitID REFERENCES RECRUITS(RecruitID)

CONTACTS foreign key UserID REFERENCES USERS(UserID)

MESSAGES foreign key UserID REFERENCES USERS(UserID)

The integrity constraints that were previously discussed for the E-R model are handled by the primary key/foreign key relationship in the relational schema.

6) *Sample Queries in SQL, Relational Algebra, and Tuple Relational Calculus*

Note that several of these queries (1-8) are very simple. These will be very frequently used search queries, and can be used either to select only ID's, or to select all information.

Examples of this are shown for the first query, but the principle can be used for all of the first

8. These queries are listed in order of increasing complexity (evaluation time).

- 1) Find all recruits that attended high school X.

SELECT RecruitID FROM Recruits WHERE HighSchoolID = X

OR

SELECT * FROM Recruits WHERE HighSchoolID = X

RA: $\sigma_{HSid=X}(RECRUITS)$ OR $\Pi_{rid}(\sigma_{HSid=X}(RECRUITS))$

TRC: $\{r | (\exists r)(RECRUITS(r) \wedge r[HSid] = X)\}$ OR

$\{z | (\exists r)(RECRUITS(r) \wedge r[HSid] = X \wedge z[RecruitID] = r[RecruitID])\}$

- 2) Find all recruits that live in state X.

SELECT RecruitID FROM Recruits WHERE State = X

RA: $\sigma_{State=X}(RECRUITS)$

TRC: $\{r | (\exists r)(RECRUITS(r) \wedge r[State] = X)\}$

- 3) Find all recruits that are the specified gender.

SELECT RecruitID FROM Recruits WHERE Gender = X.

RA: $\sigma_{Gender=X}(RECRUITS)$

TRC: $\{r | (\exists r)(RECRUITS(r) \wedge r[Gender] = X)\}$

- 4) Find all recruits that swim the specified stroke.

SELECT RecruitID FROM Recruits WHERE StrokeX = true

RA: $\sigma_{StrokeX=true}(RECRUITS)$

TRC: $\{r | (\exists r)(RECRUITS(r) \wedge r[StrokeX] = true)\}$

- 5) Retrieve all information about user X.

SELECT * FROM Users WHERE UserID = X

RA: $\sigma_{rid=X}(USERS)$

TRC: $\{u | (\exists u)(USERS(u) \wedge u[UserID] = X)\}$

- 6) Retrieve all information about recruit X.

SELECT * FROM Recruits WHERE RecruitID = X

RA: $\sigma_{rid=X}(USERS)$

TRC: $\{u | (\exists u)(USERS(u) \wedge u[UserID] = X)\}$

- 7) Find all recruits that have applied to CWRU.

SELECT RecruitID FROM Recruits WHERE Applied = true

RA: $\sigma_{applied=true}(RECRUITS)$

TRC: $\{r | (\exists r)(RECRUITS(r) \wedge r[Applied] = true)\}$

- 8) Find all recruits from high school graduating class X.

SELECT RecruitID FROM Recruits WHERE GradYear = X

RA: $\sigma_{GradYear=X}(RECRUITS)$

TRC: $\{r | (\exists r)(RECRUITS(r) \wedge r[GradYear] = X)\}$

- 9) Match recruits and recruiters based on home state

SELECT RecruitID, UserID FROM Recruits R, Users U WHERE U.State = R.State

RA: $\Pi_{RecruitID, UserID} (RECRUITS \bowtie_{r.State=u.State} USERS)$

TRC: $\left\{ z \mid (\exists r) \left(RECRUITS(r) \wedge (\exists u) \left(\begin{array}{l} USERS(u) \wedge r[State] = u[State] \\ \wedge z[RecruitID] = r[RecruitID] \\ \wedge z[UserID] = u[UserID] \end{array} \right) \right) \right\}$

- 10) Find all recruits that have not been contacted yet.

SELECT RecruitID FROM Recruits R WHERE NOT EXISTS

(SELECT * FROM Contacts C WHERE C.RecruitID = R.RecruitID)

RA: $\Pi_{rid} (RECRUITS) - \Pi_{rid} (RECRUITS \bowtie CONTACTS)$

TRC: $\left\{ z \mid (\exists m) \left(\begin{array}{l} MAJORRECRUITS(m) \wedge m[MajorID] = X \\ \wedge (\exists r) \left(\begin{array}{l} RECRUIT(r) \\ \wedge r[RecruitID] = m[RecruitID] \\ \wedge z[RecruitID] = r[RecruitID] \end{array} \right) \end{array} \right) \right\}$

- 11) Find all recruits that are interested in the specified major.

SELECT RecruitID FROM RecruitsMajors WHERE MajorID = X

RA: $\Pi_{rid} (\sigma_{mid=X} (RECRUITS \bowtie MAJORRECRUITS))$

TRC: $\left\{ z \mid (\exists m) \left(\begin{array}{l} MAJORRECRUITS(m) \wedge m[MajorID] = X \\ \wedge (\exists r) \left(\begin{array}{l} RECRUITS(r) \\ \wedge r[RecruitID] = m[RecruitID] \\ \wedge z[RecruitID] = r[RecruitID] \end{array} \right) \end{array} \right) \right\}$

Appendix A: E-R Database Model

