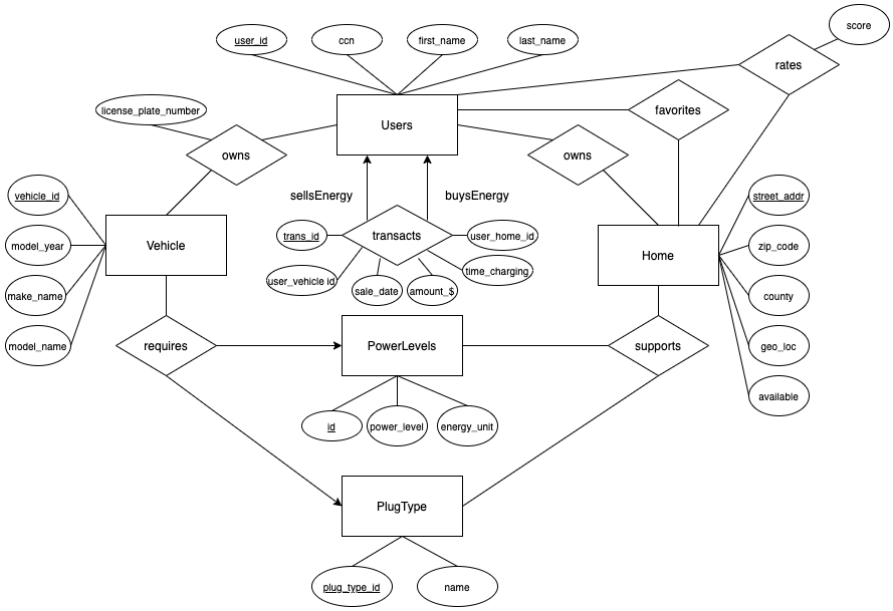
CS 411 Project Template

Team Name	**Illinois Chargers**
Members Names (NetIDs)	Brenden Prieto (bprieto2), Brutus Martin (bm19), Abhay Joshi (abhay2)
Email	bprieto2@illinois.edu, bm19@illinois.edu, abhayj@illinois.edu
Captain	Brenden Prieto (bprieto2)
Project Title	Outlet
Project Summary	Outlet is the gig economy for the electric vehicle charging space. It is a web application that allows homeowners to advertise their homes as a charging station for electric vehicles. Homeowners would interact with the app by sharing their home location as a "charging station" and Drivers would be able to find these closest "charging stations" next to them. Users of our application can register as both a homeowner and driver.
Project Description	With the rising number of electric vehicles (EV) in the market, this application solves two problems: it balances the load of electric vehicle drivers searching for a charging station, and it shrinks the distance-gap between drivers and charging stations. Our application solves the problems of finding EV charging stations to be a lot easier and so that people will not have to rely on private companies who set up charging stations. This will help save a lot of time especially as slow charging vehicles take up stations for long periods of time. PlugShare is a similar application to our project however their application doesn't appear to handle transactions for homeowners. So far, there are two data sets we intend to use: One is a corpus of addresses aggregated by the US Transportation department, and the other is an electric vehicle description table that one of our project member's has access to via his employer, Cox Automotive. We also plan to simulate user data (user id, first name, last name) from https://mockaroo.com/. The transportation website link is: https://www.transportation.gov/gis/national-address-database/national-address-database-0. Functionalities in our application are going to be that users are able to make and receive payments. Users can add as many homes and cars that they want to their profile and will have to provide information about either their homes or cars. Users will also be able see all of their past transactions. Another functionality we will

	have on our application is that when a driver gets to a home they will “check-in” through the app which updates the database of available homes and likewise the driver will “check-out” when they are finished and our database will reflect the change. So overall we will need the database in order to insert new users, update information, deleting user accounts, and in order to search nearest charging stations. Basic: • insert new users, homes, cars • update information about availability, favorites, home ratings • deleting users, homes, cars Advanced Functionalities: • Constraints such as NOT NULL for most of the ids in the table, Primary key for each table and used UNIQUE in table Users for the attribute email_addr • Partition the transaction table by Month in order to improve the response time for SQL queries • Prepared Statement for inserting data into UserVehicle table • Created a Trigger in the Users table that would (right before deleting a user) delete the any houses or cars stored by the userid in the uservehicle and userhome table • Indexing for multiple tables including Home and Vehicle in order to speed up queries • Created a View named tansactions_summary which gives a summary of transactions at any point in time split by states in US. • Stored Procedure (SP) , SP1- getUser which returns all user_id and First_Name , SP2 - getUserEmail , that gives the user email for a given user_id

ER Design	
Development Plan	**Relation schema of your database -** Create Table Users (user_id INTEGER, email_addr VARCHAR(255), First_name VARCHAR(16) , Last_name VARCHAR(16)); Create Table UserVehicle (User_vehicle_id INTEGER, User_id INTEGER, Vehicle_id INTEGER, Lpn VARCHAR(16), Plug_type_id INTEGER); Create Table Vehicle (Vehicle_id INT, Model_year INT, Make_name VARCHAR(16) , Model_name VARCHAR(16)); Create Table UserHome (User_home_id INTEGER, Home_id INTEGER, User_id INTEGER, Available BOOLEAN, Agg_rating FLOAT)

Create Table HomePlugSupport (
Home_plug_id INTEGER,
Home_id INTEGER,
Plug_type_id INTEGER
)

Create Table UserHomeFavorites (
User_home_favorites_id INTEGER,
User_id INTEGER,
User_home_id INTEGER
)

Create Table Home (
Home_id INTEGER,
Street_addr VARCHAR(255),
Zip_code INTEGER,
County VARCHAR(16),
Geo_loc VARCHAR(255)
);

Create Table PlugType (
Plug_type_id INT,
Name VARCHAR(16)
);

Create Table Transactions (
trans_id INT,
User_vehicle_id INT,
User_home_id INT,
Sale_date TIMESTAMP,
sale_price DECIMAL (18,2),
time_charging TIME
);

Choice of database and languages

Front End: Javascript w/ React.js
Backend: MySQL and Node.js for server

Sources of your data -

Cox Automotive for Year, Make, Model names
Mockaroo (https://mockaroo.com/) for generating user names, emails, ids
National Address Database from Transportation.gov for addresses
Mostly simulated data

Labor division among group members -

Abhay - Back-end
Brenden - Front-end
Brutus - Back-end

Project timeline -

- End of Week 7
  - Backend:
    - Create Users, UserVehicles, UserHomes, Vehicles, Homes tables and fill in with sample data
  - Frontend:
    - Complete basic search functions for homes
      - Find user location by IP or allow users to enter address
      - Display nearest homes
- End of week 8
  - Backend:
    - Insert addresses from transportation.gov website into Homes table
    - Create PlugType and HomePlugSupport tables
      - Simulate available plug types for homes and insert records into HomePlugSupport
    - Create UserHomeFavorites table and simulate data
  - Frontend:
    - Complete home selection, transaction complete/receipt modal
- End of Week 9
  - Backend:
    - Simulate transactions and insert records into Transactions table
    - Create UserHomeFavorites table and insert simulated data
  - Frontend:
    - Work on transaction history page
    - Add form validation to user signup page to ensure no duplicate users are created
- End of week 10
  - Backend
    - All tables should have at least some data at this point
    - Start writing queries to tables
  - Frontend
    - Complete “add vehicle” functionality
    - Complete “add home” functionality
- End of week 11
  - Backend
    - Continue to add data to tables
    - Continue to write queries
  - Frontend
    - Add filtering of nearby homes by Favorites
    - Add sorting of nearby homes

- End of week 12
  - Backend
    - All queries to access data should be complete
  - Frontend
    - Complete remove/edit homes and vehicles
- End of week 13
  - Backend
    - All tables should be have all the data at this point
  - Frontend
    - Complete transactions page
- End of week 14
  - Backend
    - Buffer time in case we need it
  - Frontend
    - Start working on My Vehicles to show user activity
- End of week 15
  - Backend
    - Buffer time in case we need it
  - Frontend
    - Complete My Vehicles and My Home page
- End of week 16
  - Project complete

API:

https://api.ipgeolocation.io/getip - In order to get the IP of client browser

https://api.ipgeolocation.io/ipgeo - To get the geolocation of the client IP address

https://us1.locationiq.com/v1/search.php - To get the geolocation of an input address

These API's would allow us to find the nearest available charging stations for car owners by locating the nearest locations.

URL Endpoint / DB calls:

http://outletprototype.web.illinois.edu/api/vehicle/makes?model_year=2018 - By accessing this endpoint and adjusting the model_year parameter you can see the different makes for that particular year.

http://outletprototype.web.illinois.edu/api/vehicle/models?model_year=2020&make_name=Tesla - You can also query out via the model year and make name. In this

	example above we queried out models from 2020 and where car makers was Tesla. **URL:** Primary domain: outlet.prototype.web.illinois.edu
System Demo URL	outlet.prototype.web.illinois.edu
Initial Demo Video	Demo video: https://mediaspace.illinois.edu/media/t/0_l9luwa50 Example username: glycettoqj@webs.com Example password: 1N1JCIOO Just in case you feel like there's not enough "proof" that the database is being updated, you can validate the results yourself by using the api endpoints in the browser. For example, if you open up a separate tab with the endpoint http://outletprototype.web.illinois.edu/api/userCars?user_id=33060 (for the example user above), you can directly query the database for this user's vehicles and it will show in the browser. This means, if you were to make any updates, deletions, or additions to the user vehicles, in the My Vehicles page, this endpoint will show you the effects in the database. Since this is exactly what's being used to show user vehicles in the search page for homes, you don't really need to do this api step manually, but I'm providing it just in case.
Project Files	https://github.com/bprieto12/cs-411-submission-folder
Final Demo Video Link	Demo Video: https://mediaspace.illinois.edu/media/t/0_e7gp76r5 !! Please see bottom of this page for usernames and passwords for you to test out in your review !! Or you can use the username_pwd.txt file in the project file

Advanced Functionality:

1. Searching for nearest homes

Challenge: Finding the homes nearest to the user's input address along with additional info about the home

Solution: To solve this, we use an API to get the latitude and longitude of the user's input address. That is then sent to a stored procedure we created called findNearestHomes that uses a spherical distance formula (called the haversine formula) to find the distance from the input address to all of the addresses in our Homes table. Prior to this step, we also had to find the latitude and longitude of every home in our Home table. After finding all the distances in a subquery, we join this result on two of our views, "hotspots" and "home_review_stats" to get additional info on how often this home has been visited recently and how this home has been rated and the number of times reviewed.

Why this is challenging: This is technically challenging because we had to employ three advanced techniques to make it work: a stored procedure to simplify the query from the server, views to get the additional information, and indexes to make the query fast so the user doesn't have to wait too long

2. Dynamic vehicle dropdowns

Challenge: In the "My Cars" page, create dropdowns that dynamically update based on user input. In the past, the available values in the dropdowns were updated once on page-load, with the distinct values available for MODEL_YEAR, MAKE_NAME, MODEL_NAME, and PLUG_TYPE. While the database did return a failure response to the server if an invalid vehicle was sent, it created a bad user experience because it wasn't clear which vehicles were configurable according to the database. The challenge here isn't just about creating a situation where only configurable vehicles are sent, it's also about creating a good and simple user experience.

Solution: Every time a user changes one of the dropdown fields, a query is sent to find the distinct values and the subsequent dropdown fields will update based on the results. This ensures only valid vehicles can be sent to the database. However, additional user experience elements were implemented to make this more technically challenging (as the programmer), but easier for users. For example, let's say a user had a 2013 Chevrolet Bolt Type 1 Plug, and wanted to update the model year dropdown to 2014 Chevrolet Bolt Type 1 Plug. If the Chevy Bolt Type 1 Plug exists in 2014, when the user selects 2014 in the model year dropdown, the make, model, and plug type dropdowns will be re-selected to avoid users having to select them again. On the other hand, if that vehicle didn't exist in 2014, the make, model, and plug type filters would clear out. However, the code at all times tries to simplify the experience as much as possible, if the Chevy Bolt existed in 2014 but the Type 1 Plug did not, it would reselect the 2014 Chevy Bolt and clear out the Plug Type dropdown.

Why this is challenging: Ensuring only valid vehicles can be sent and dropdowns accommodate exactly what the user expects is very challenging and requires a lot of code. We also added an advanced technique, a prepared statement, to ensure only valid inputs could be added to the database, especially for the license plate number which is a text field.

3. <u>Handling charging at a home and rating the experience</u>

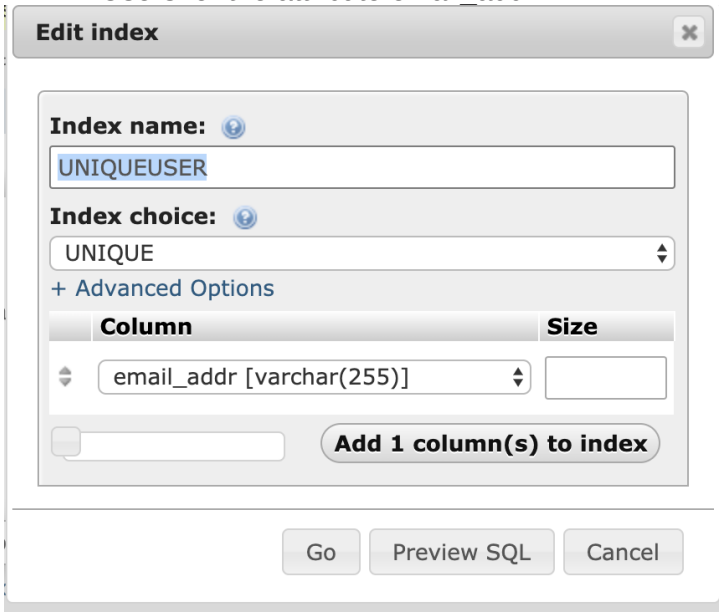
Challenge: Allowing the user to charge at a home for however long they want, while keeping a running total of charging time and cost, and allowing the user to rate the experience at the end.

Solution: Created two modals, one for charging that holds the running time and cost in its state, and another for the receipt and rating, which gets its info from the charging modal. The receipt modal, once the user clicks Finish, then inserts data into the Transactions table with the review of the experience, the total time charging, the cost, the user vehicle, the user home, and the day of the transaction.

Why this is challenging: This was technically challenging mostly from a front-end experience because it required maintaining state properly (starting the clock and cost and making sure they stop and clear appropriately when the user completes the transaction so they can start another fresh if they'd like) and passing it to other UI elements, and making sure the experience was simple and intuitive.

Advanced Techniques:

4. Constraints such as NOT NULL for most of the ids in the table, Primary key for each table and used UNIQUE in table Users for the attribute email_addr

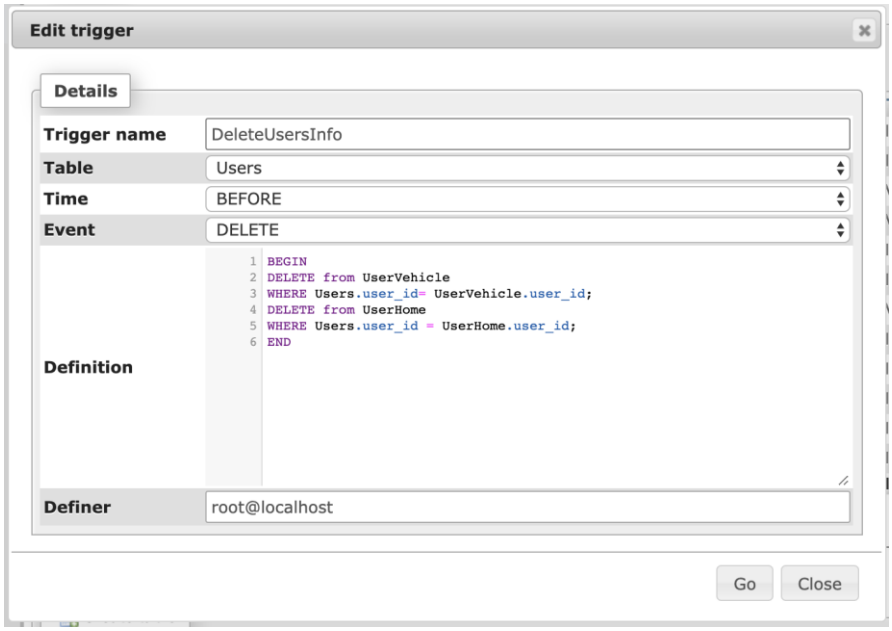


5. Partition the transaction table by Month in order to improve the response time for SQL queries
6. Prepared Statement for inserting data into UserVehicle table

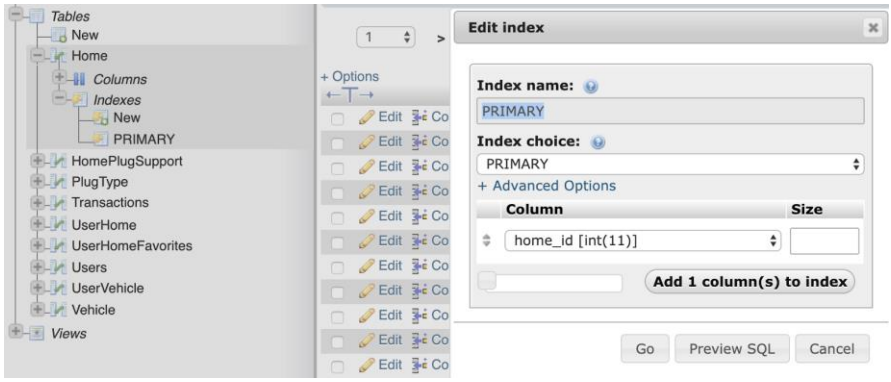
- Line 274 of server.js file:

```
let submit_new_vehicle_query = "insert into UserVehicle values (?,?,?,?,?);";
let prepared_statement = mysql.format(submit_new_vehicle_query, [new_user_vehicle_id, req.query.user_id,req.query.vehicle_id,req.query.Lpn, req.query.isDefault]);
```

7. Created a Trigger in the Users table that would (right before deleting a user) delete the any houses or cars stored by the userid in the uservehicle and userhome table



8. Indexing for multiple tables including Home and Vehicle in order to speed up queries



9. Created three views

Home_review_stats: gets the average review and the number of reviews by home

CREATE ALGORITHM=UNDEFINED DEFINER=`outletprototype`@`localhost` SQL SECURITY DEFINER VIEW `home_review_stats` AS select `t`.`user_home_id` AS `user_home_id`,count(0) AS `num_reviews`,avg(`t`.`rating`) AS `avg_rating` from `Transactions` `t` group by `t`.`user_home_id`

Hotspots: finds the homes that have been visited several times recently. Since we don't have actual user data, we limited this to the last 60 days and number of visits greater than 2.

CREATE ALGORITHM=UNDEFINED DEFINER=`outletprototype`@`localhost` SQL SECURITY DEFINER VIEW `hotspots` AS select `t`.`user_home_id` AS `user_home_id`,`uh`.`home_id` AS `home_id`,count(0) AS `num_visits` from (`Transactions` `t` join `UserHome` `uh` on((`t`.`user_home_id` = `uh`.`user_home_id`))) where (`t`.`sale_date` > (curdate() - 60)) group by `t`.`user_home_id`,`uh`.`home_id` having (`num_visits` > 2)

Transactions_summary: a view for analyzing data on a state by state basis (which can be used for promotional purposes)

CREATE ALGORITHM=UNDEFINED DEFINER=`outletprototype`@`localhost` SQL SECURITY DEFINER VIEW `transactions_summary` AS select `c`.`state` AS `state`,count(distinct `b`.`user_id`) AS `total_home_owners`,count(distinct `b`.`home_id`) AS `total_home`,count(distinct `d`.`Vehicle_id`) AS `total_vehicle_charged`,count(distinct `d`.`user_id`) AS `total_users`,avg(`a`.`time_charging`) AS `avg_charge_time`,avg(`a`.`sale_price`) AS `avg_sale` from (((`Transactions` `a` join `UserHome` `b` on((`a`.`user_home_id` = `b`.`user_home_id`))) join `Home` `c` on((`b`.`home_id` = `c`.`home_id`))) join `UserVehicle` `d` on((`a`.`user_vehicle_id` = `d`.`user_vehicle_id`))) group by `c`.`state`

10. Created two Stored Procedures

1. findNearestHomes: this is the main way we query the database for homes near the user input address. This stored procedure uses the views hotspots and home_review_stats and uses the

haversine formula to calculate distances from users latitude and longitude

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `findNearestHomes`(userLatitude float, userLongitude float, userLimit int)
BEGIN
select a.*, round(distance_meters / 1509, 3) as distance_miles, case when h.user_home_id is null then 0 else 1 end as is_hotspot, hs.avg_rating, hs.num_reviews
		from (
			SELECT h.*, uh.user_home_id, uh.avg_rating,
			6371000 * acos(sin(radians(geo_latitude)) * sin(radians(userLatitude)) + cos(radians(geo_latitude)) * cos(radians(userLatitude)) * cos(radians(userLongitude - geo_longitude))) as distance_meters
			from Home h
				join UserHome uh on (uh.home_id = h.home_id)
		) as a
		left join hotspots h on (h.user_home_id = a.user_home_id)
		left join home_review_stats hs on (hs.user_home_id = a.user_home_id)
		order by distance_miles asc
	limit userLimit;
END
```

In server.js (line 50):

```
app.get("/api/search/homes", (req, res) => {
    if (req.query.latitude && req.query.longitude) {
        let limit = 10;
        if (req.query.show) {
            limit = req.query.show;
        }
        // select a.*, round(distance_meters / 1509, 3) as distance_miles, case when h.user_home_id is null then 0 else 1 end as
        //             from (
        //                 SELECT h.*, uh.user_home_id, uh.avg_rating,
        //                 6371000 * acos(sin(radians(geo_latitude)) * sin(radians(userLatitude)) + cos(radians(geo_latitude)) *
        //                 from Home h
        //                     join UserHome uh on (uh.home_id = h.home_id)
        //             ) as a
        //             left join hotspots h on (h.user_home_id = a.user_home_id)
        //             left join home_review_stats hs on (hs.user_home_id = a.user_home_id)
        //             order by distance_miles asc
        // limit userLimit;

        let query = "call findNearestHomes(" + req.query.latitude + ", " + req.query.longitude + ", " + limit + ")";
        console.log(query);
        con.query(query, (err, rows) => {
            res.json(rows[0]);
        });
    } else {
        res.status(400).json({"message": "a latitude and logitude must be passed to search for homes"});
    }
});
```

2. getUserTransactions: gets the user transactions

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`getUserTransactions`(userId int)
BEGIN
select t.*,
    v.model_year,
    v.make_name,
    v.model_name,
    p.name as plugType,
    h.street_addr,
    h.zipcode,
    h.state
    from Transactions t
        join UserVehicle uv on (uv.user_vehicle_id =
t.user_vehicle_id)
         join Vehicle v on (v.vehicle_id = uv.vehicle_id)
         join PlugType p on (p.plug_type_id = v.plug_type_id)
         join UserHome uh on (uh.user_home_id = t.user_home_id)
         join Home h on (h.home_id = uh.home_id)
     where uv.user_id = userId;
END
```

In server.js file (line 212):

```
app.get("/api/userPurchases/:user_id", (req, res) => {
    // select t.*,
    // v.model_year,
    // v.make_name,
    // v.model_name,
    // p.name as plugType,
    // h.street_addr,
    // h.zipcode,
    // h.state
    // from Transactions t
    //     join UserVehicle uv on (uv.user_vehicle_id = t.user_vehicle_id)
    //      join Vehicle v on (v.vehicle_id = uv.vehicle_id)
    //      join PlugType p on (p.plug_type_id = v.plug_type_id)
    //      join UserHome uh on (uh.user_home_id = t.user_home_id)
    //      join Home h on (h.home_id = uh.home_id)
    //  where uv.user_id =  userId;
    let query = "call getUserTransactions(" + req.params.user_id + ")";

    con.query(query, (err, rows) => {
        if (err) {
            res.status(400).json({"message": "couldn't get transactions for user id: " + req.params.user_id});
        }
        res.json(rows);
    })
});
```

Usernames and Passwords:

email_addr pwd

aabelsoud@hexun.com WJMC0XKK

aabercrombiem3g@booking.com 78K9KJDD

aabernethykgf@prnewswire.com N95NT8JJ

aablewhite104r@yolasite.com O5ZB8LOO

aabrahamsonp7h@gizmodo.com TS84YN88

aabrahamst6j@hc360.com C565L300

aabramovitzqq5@npr.org 1X61KJWW

aabreymjc@dion.ne.jp 8EYUN211

aachrameevl1a@eepurl.com Y4MILIPP

aadamoviczme8@desdev.cn LJTZ63AA

aaddersonszk@symantec.com SN2JSGG

aaggissf1c@technorati.com 4BE1OTLL

aairesllk@baidu.com 404RIGUU

aaldersleywu0@answers.com 3B7NSS

aaleksankinsd1@unesco.org YGLEIVLL

aallcorng4k@wordpress.org 9X0WZO88

aalyoshink1k@hubpages.com KYDZ94QQ

aambresint4f@wiley.com 91LL52DD

aanderlhtb@latimes.com JSO1FQFF

aandreelge@wired.com SU35RPBB

aandrivelm2n@lulu.com JEKL9FSS

aanstissu1z@desdev.cn 1BGS28ZZ

aantczakt4a@phpbb.com 3UIXU2LL

aarmour10zk@diigo.com 3CTXKILL

aarpurpaf@uol.com.br U2HK7GG

aaspitalmw1@nymag.com Z6JWMI77

aavramovicf22@wikispaces.com UP56EEHH

aaxelrmu@paginegialle.it 5FIBC3CC

abadbyoun@paginegialle.it 9U01ZTBB

abartlemanqep@issuu.com MNLK9WW

abasketterkow@51.la 17QBE388

abattanymhi@mapquest.com V0JFUS55

abeadnallixy@jugem.jp L53R1BZZ

abearegh1@bing.com 7LUEZ6GG

abeckleskrt@mediafire.com 52LJ77

abedomevvd@wiley.com TQSU0QPP

abehnkes6w@123-reg.co.uk A3QYNJFF

abelkoxf@weather.com UFIHOI44

abellardx70@ebay.com BJPIUYY

abendawpi@cornell.edu M4Q8PZUU

abendonlol@sciencedirect.com E03RIE11

abernhartq89@cnbc.com UYWOZWQQ

abertramk62@w3.org 5LNVNP66

abeurichly7@is.gd 5EXXLG99

abewsyglu@eepurl.com ZRQ4HSBB

abiasinilpl@booking.com DK9FO6WW

abice11lm@jugem.jp QUALM8RR

abichenoko2@google.de DMHBSVHH

abiddlestonehbs@soup.io 8MI78MOO

abikkerf1n@skyrock.com TJJJ57YY