University of British Columbia, Department of Computer Science

CPSC 304

Cover Page for Final Deliverable

Date: November 18th 2018

Group Members:

Name	Student Number	CS Userid	Tutorial Section	Email Address
Gina Hong	44446152	i0l0b	T1E (W 4-5)	1ginahong@gmail.com
Nicholas Chin	54101167	p8d1b	T1A (W 12-1)	nicholaschin20@gmail.com
Aleksei Feklisov	28039162	y8v0b	T1H (F 12-1)	feklisoff@gmail.com
Maximilian Was-Damji	21094164	n6g1b	T1H (F 12-1)	maximilian@keemail.me

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above.
In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

MarketDB

About MarketDB

MarketDB (or DBMart) is a web application for users to buy and sell products. The current list of functionalities allows Users to create an account, add product posts, edit and delete posts, search for products in their price range, and see previous comments.

Additionally, based on a user's previous likes, our platform will be able to 'match' certain advertisements to that user.

This project was implemented using Node.js (Express) for backend, and MySQL as the server. Because of Express' "render" functionality, we were able to use .ejs for the frontend.

Notable Changes since Previous Phase

- ER Diagram change where User, Seller, Buyer are identical. Hence, we have deleted the Seller and Buyer table from both the schema and the marketdb.sql file.
- We trimmed down much of the queries we mentioned in the Formal Specification stage, b/c:
    - It became apparent that setting up the backend(learning node.js), and having a reasonable frontend would not be as simple as we had thought.
- We ended up with 11 "main" queries + 2 views.
    - Section **"Functionality of Final Application"** explains what functions we have implemented.
- Because queries are embedded throughout the js files in the routes folder, we have listed the SQL queries in the **"SQL Queries Used"** section.

- **Here's the link to our github repo.**

Screenshot of Sample Output

- See "screenshots" folder to see the output screens!

How to run our App:

- See the README.md on our github repo. After installing the node dependencies listed on the package.json file (npm install ...), we did nodemon app.js to run it on localhost:5000

Tables:

Advertisement[adid: Int(8), AdImage: Varchar(500), AdLink: Varchar(500)]

Primary Key: adid

FDs: adid → AdImage, AdLink

User(uid: Varchar(30), first_name: Varchar(20), last_name: Varchar(20), Password:Varchar(30), BirthDate: Date)

Primary key: uid

FDs: uid → first_name, last_name, Password, BirthDate

Product_Posts(postid: Int, **uid:** Varchar(30), Product_Description: Varchar(1000), Product_Name: Varchar(30), Price: Real, sold: Boolean default 0)

Primary Key: postid

Foreign Key: uid references User

FDs: postid → uid, Product_Description, Product_Name, Price, sold

Transaction_Buys(transactionid: Int, Card Exp: Char(4), Card No: Char(12), Card Name: Varchar(50), **postid**: Int, **uid**: Varchar(30))

Primary key: transactionid

Foreign Key: postid references Product_Post, uid references User

FDs: transactionid → Card_Exp, Card_No, Card_Name, postid, uid

Comment_authors(commentid: Int, **postid**: Int**,** **uid**: Varchar(30), Commenttxt: String, CommentDate: Date, Edited: Boolean)

Primary Key: commentid, postid, uid

Foreign Key: postid references Post, uid references User

FDs: commentid, postid, uid → Commenttxt, CommentDate, Edited

Product_Photo(photoid: Int, **postid**: Int, photo_link: varchar(500))

Primary Key: photoid, postid

Foreign Key: postid references Post.

FDs: photoid, postid → photo_link

Tag(tag_name: Varchar(20))

Primary Key: tag_name

ad_has_tag(**adid**: Int(8), **tag_name**: Varchar(20))

Primary Key: adid, tag_name

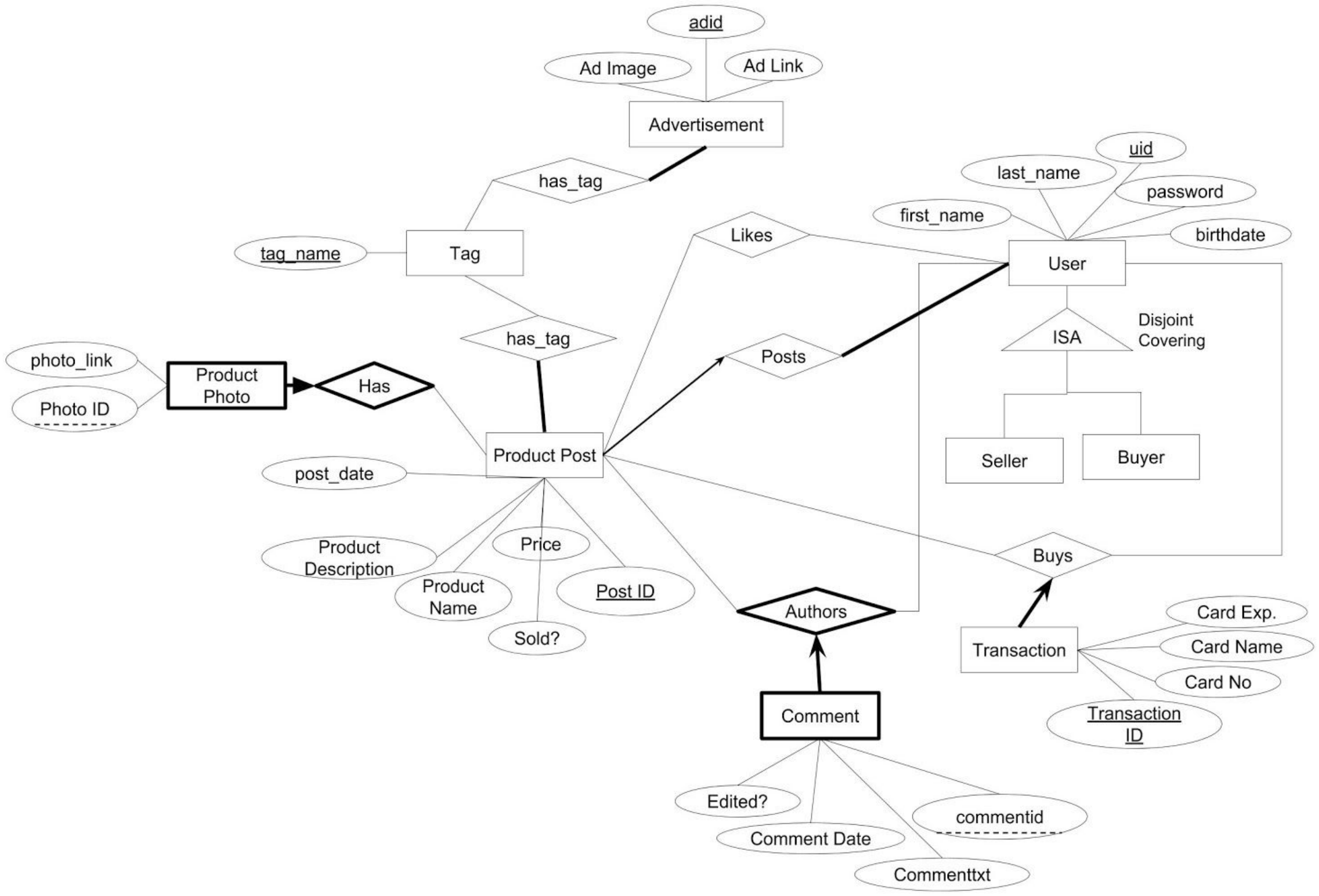
Foreign Key: adid references Advertisement, tag_name references Tag

post_has_tag(<u>**postid**</u>: Int, <u>**tag_name**</u>: Varchar(20))
    Primary Key: postid, tag_name
    Foreign Key: adid references Advertisement, tag_name references Tag

user_likes(<u>**uid**</u>: Int, <u>**postid**</u>: Int)
    Primary Key: uid, postid
    Foreign Key: uid references User, postid references Product_posts

Our table is normalized to BCNF because every FD where X → b, has an X that is a superkey of that relationship.

ER Diagram:



MarketDB Entity Relationship Diagram

ER diagram changed where User, Seller, Buyer are identical. Hence we only make a table for the Parent: User

Platform

Node.js Express.js backend. EJS frontend (because of render in Express, no need for jquery). MySQL for the database.

Functionality of Final Application

Category	General idea	detailed
2-INSERT (REQ 1)	A. Add Post (N)	A user is able to add a post with a product he wants to sell. The postid is auto-generated user must type in a valid username/password combination..
	B. Add User (A)	A user creates an account. Insert new user info in to User table
	F. Add Tag (N)	A user adds a tag to their product post. ("tag" field when adding post)
3-DELETE (REQ 1)	A. Delete Post (N)	With the right username/password combination, you can delete a post.
4-UPDATE (REQ 1)	A. Edit Post (N)	User can make changes to a post. (change description, product name)
	B. Update account info (A)	User can update their account info (Not user_ID though)
5-JOIN(3) (REQ 1)	A. linking users w/ advertisements they may be interested in. (G)	Query which user is interested in which advertisement + tag. JOINs advertisement, user, ad_has_tag, and user_interested(view)..
6-JOIN(2) (REQ 1)	B. Comment with User first name, last name (M)	Displays user name with comment. JOIN user w/ comment_authors.
7-GROUP BY (REQ 1)	A. Show product_post information (number of likes each post has).	This is the "main_table" that you see on the main page. (groupby, join(3), view, union). One column shows number of likes.
8-GEN (SELECT)	User Verification in delete post (G)	Select query that searches for the user inputted uid and password values. Used in post deletion, where if the result of this query is empty, post cannot be deleted.
9-GEN (SELECT)	Current_posts in price range (G)	Select query that searches for current_posts that are within a user specified price range.

10-GEN	Select from the extra SQLs in Insert (B and F).	
11-VIEW (REQ 1)	A. Current product post (G)	Create view current_posts where posts are not marked as sold. (used to SELECT posts in certain price point).
	B. User_interested (G)	Create view that joins user, post_has_tag, and user_likes. It shows which tags a user is interested in based on their likes history. Used for the Join(3) query (linking users with advertisements).
12-DIVISION	A.Get post that all users liked	

SQL Queries Used (Not an exhaustive list, but shows the main SQL)

Category	**General idea**	**SQL embedded in Application**
CREATE	SQL file for creating the tables(no views) are in the /db folder’s marketdb.sql	
Populate Database	SQL file for populating the database and creating the views are in the /db folder’s populate_marketdb.sql	
2-INSERT	A. Add Post	“**INSERT INTO** product_posts (postid, uid, product_description, product_name, post_date, price) VALUES ('" + newpostid + "', '" + post_by_id + "', '" + product_description + "', '" + product_name + "', '" + post_date + "', '" + price + "');"
	B. Add User	"**INSERT INTO** \`user\` VALUES ('" + uid + "', '" + first_name + "', '" + last_name + "', '" + password + "', '" + birthdate + "')"

	F. Add Tag	“**INSERT INTO** post_has_tag(postid, tag_name)
		VALUES (
		      (SELECT postid
		      FROM product_posts
		      WHERE postid = '" + newpostid + "'),
		 '" + tag + "');”

3-DELETE	A. Delete Post	"**DELETE** FROM product_posts WHERE postid = '" + inputpostid + "';"’

4-UPDATE	A. Edit Post	"**UPDATE** product_posts
		SET product_name = '" + product_name + "', \`product_description\` = '" + product_description + "', \`price\` = '" + price + "'
		WHERE postid = '" + post_id + "';"

	B. Update account	"**UPDATE \`user\`**
		SET \`first_name\` = '" + first_name + "', \`last_name\` = '" + last_name + "', \`password\` = '" + password + "', \`birthdate\` = '" + birthdate + "'
		WHERE \`uid\`='" + uid + "'"

5-JOIN(3)	A. linking users w/ advertisements	“SELECT DISTINCT u.uid, CONCAT(u.first_name, ', ', u.last_name) AS 'Name', a.adid, a.adimage, a.adlink, atag.tag_name
		FROM advertisement a, user u, ad_has_tag atag, user_interested ui
		WHERE (a.adid = atag.adid) and (u.uid = ui.uid) and (ui.tag_name = atag.tag_name)
		
		UNION ALL
		
		SELECT u.uid, CONCAT(u.first_name, ', ', u.last_name) AS 'Name', 'No' AS 'a.adid', 'information', 'on User' AS 'a.adlink', '' AS 'atag.tag_name'
		FROM user u
		WHERE u.uid NOT IN
		     (SELECT u.uid
		     FROM advertisement a, user u, ad_has_tag at, user_interested ui
		     WHERE (a.adid = at.adid) and (u.uid = ui.uid) and (ui.tag_name = at.tag_name))
		ORDER BY uid;"
		
		//Return table of users and their recommended ads.
		//If the user does not exist in user_interested, then columns should say “No information on User”.

6-JOIN(2)	B. Comment with User first name, last name	"SELECT
		     CONCAT(user.first_name, ', ', user.last_name)
		          AS name,
		     DATE_FORMAT(comment.commentdate, '%Y-%m-%d')
		          AS commentdate,
		     comment.commenttxt AS commenttxt
		FROM product_posts product,
		     comment_authors comment, user
		WHERE user.uid = comment.uid
		     AND comment.postid = '" + req.params.postid + "'
		     AND product.postid = '" + req.params.postid + "'",

7-GROUP BY	A. Show product_post information and the	(This query NOT embedded in app. This one is in the populate_marketdb.sql file in /db folder) CREATE OR REPLACE VIEW main_table AS

	number of likes each post has.	SELECT p.postid, u.uid, CONCAT(u.first_name, ', ', u.last_name) AS 'name', p.product_name,
		CONCAT(SUBSTRING(p.product_description, 1, 30), '...') AS 'detail',
		DATE_FORMAT(p.post_date, "%M %d %Y") AS 'date', p.price, p.sold, CASE WHEN p.sold='1' THEN 'true' ELSE 'false' END AS psold,
		COUNT(*) AS 'likes'
		FROM user u, product_posts p, user_likes ul
		WHERE (u.uid = p.uid) and (p.postid = ul.postid)
		GROUP BY u.uid, u.first_name, u.last_name, p.postid, p.product_name, p.product_description, p.post_date, p.price
		UNION ALL
		SELECT p.postid, u.uid, CONCAT(u.first_name, ', ', u.last_name) AS 'name', p.product_name,
		CONCAT(SUBSTRING(p.product_description, 1, 30), '...') AS 'detail',
		DATE_FORMAT(p.post_date, "%M %d %Y") AS 'date', p.price, p.sold, CASE WHEN p.sold='1' THEN 'true' ELSE 'false' END AS psold,
		0 AS 'likes'
		FROM user u, product_posts p, user_likes ul
		WHERE (u.uid = p.uid) and (p.postid NOT IN (SELECT postid FROM user_likes))
		GROUP BY u.uid, u.first_name, u.last_name, p.postid, p.product_name, p.product_description, p.post_date, p.price;

8-GEN (SELECT)	User Verification in delete post	**"SELECT p.postid, u.uid, u.password** FROM product_posts p, user u WHERE p.uid = u.uid and p.uid = '" + userid + "' and u.password = '" + pass + "' and p.postid = '" + inputpostid + "'";

9-GEN (SELECT)	Current_posts in price range	**"SELECT ***
		FROM `current_posts`
		WHERE (price >= '" + min + "') AND (price <= '" + max + "')";

10-GEN	Select from extras in Insert (B and F).	
11-VIEW	A. Current product post	(This query is NOT embedded in app. In the populate_marketdb.sql file in /db folder) **CREATE OR REPLACE VIEW current_posts** AS SELECT postid, name, product_name, detail, date, price, likes FROM main_table WHERE sold='0';

	B. User_ interested	(This query is NOT embedded in app. In the populate_marketdb.sql file in /db folder)
		
		CREATE OR REPLACE VIEW user_interested AS
		SELECT DISTINCT u.uid, pt.tag_name
		FROM user u, post_has_tag pt, user_likes ul
		WHERE (u.uid = ul.uid) and (pt.postid = ul.postid);

12- DIVISION	A.Get post that all users liked	SELECT ul.postid, p.product_name, p.product_description, p.price
		FROM user_likes ul, product_posts p
		WHERE ul.postid = p.postid
		GROUP BY ul.postid
		HAVING count(ul.uid) =
		      (SELECT COUNT(*) FROM user);

Division of Labour

Name	**Tasks**	
Gina	Join(3) A, Group by A, 8/9-Gen,View A/B	Document/Populate DB
Nick	Insert A/F, Delete A, Update A	Queries/Testing
Max	Join(2) B	Queries/Testing
Alex	Insert B, Update B, Division	Queries/Testing

** Everybody embedded their SQL queries into the app.