Documentazione Tecnica

Oak_Sourcing

Gabriele Zarcone
892865

Introduzione

Oak_sourcing è un'applicazione web che implementa una piattaforma per lavorare secondo il modello del lavoro collettivo attraverso la creazione e l'esecuzione di task.

I task sono domande a risposta multipla e fanno parte ognuno di campagne di lavoro create da utenti **requester**. Lo svolgimento del task è affidato a utenti **worker** che scelgono una delle risposte tra quelle disponibili per ogni task che viene loro assegnata.

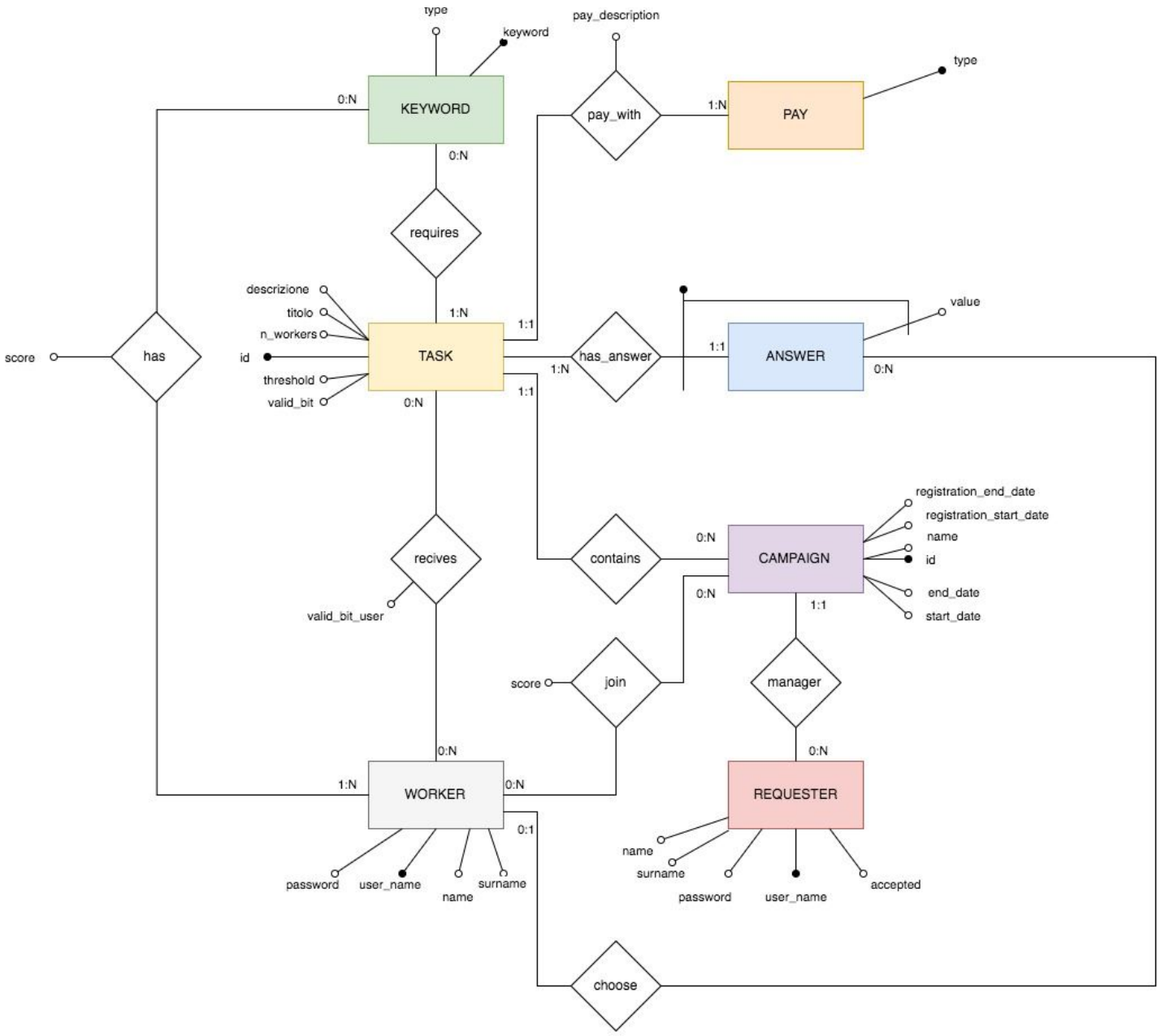
La particolarità del lavoro collettivo consiste nel fatto che la risposta corretta di una task viene decisa tramite un criterio di maggioranza analizzando tutte le risposte dei diversi utenti che hanno eseguito quella task.

Prodotti software e linguaggi utilizzati

Per l'implementazione dell'applicazione web sono stati utilizzati PostgreSQL come DBMS, plpgSQL come linguaggio procedurale e PHP come linguaggio per lo sviluppo dell'interfaccia web. Per la gestione del database si è usato il tool grafico Postico, mentre come web server ci si è affidati ad Apache.

1 - Schema concettuale ER della base di dati

Schema ER



Descrizione dello schema

Task

i task sono la componente principale del sito. Hanno un titolo e una descrizione e sono domande a risposta multipla. Ogni task fa parte di una campagna creata da un utente requester(relazione *contains*). Le risposte date dagli utenti worker ad ogni task vengono collezionate e al superamento di un certo numero di risposte (n_workers) se la percentuale di worker che hanno dato una certa risposta supera una certa soglia (threshold) la suddetta risposta viene riconosciuta come quella corretta per quella task. Se al superamento del numero di risposte atteso una task avrà una risposta corretta, allora quella task ha un

risultato valido e il *valid_bit* verrà aggiornato a true, altrimenti il risultato non sarà valido e si aggiornerà a false.
Ad ogni task inoltre sono associate delle keyword che descrivono le abilità richieste per poter rispondere a quel task.

Answer

l'entità answer rappresenta una singola risposta di una task. Viene rappresentata nel diagramma come una entità debole poichè ogni risposta è strettamente collegata alla task di cui fa parte. Di conseguenza non può esistere una risposta senza che questa sia collegata ad una task.

Worker

è la prima tipologia di utenti del sito. Esso è il responsabile dell'esecuzione delle task che gli vengono assegnati dall'algoritmo di scheduling. Ogni worker associa a se stesso delle keyword che rappresentano le sue abilità o attitudini e indica al sistema il suo grado di conoscenza di ogni keyword(relazione has e attributo score). Il sito assegna al worker (relazione recives) una per una tutte le task che fanno parte delle campagne a cui il worker è iscritto (relazione join) in base alle keyword scelte dal worker e al loro livello. Una volta che il worker riceve una trask può scegliere una delle risposte di quella task e la sua scelta viene inserita nella tabella choose, se la scelta fatta dal worker si rivelerà quella fatta dalla maggioranza allora la la sua risposta sarà valida e verrà aggiornato il valid_bit_user e lo score del worker nella campagna di cui fa parte la task.

Requester

la seconda tipologia di utente sono i Requester. Questi hanno il compito di creare le campagne e i task a cui gli utenti Worker rispondono. Vengono accettati da un utente amministratore del sistema che può aggiornare a true l'attributo *accepted* di tutti i requester accettati.

Campaign

Una campagna contiene le task create dal requester per quella determinata campagna. Hanno una data di apertura (start_date) e di scadenza (end_date) definite anch'esse dal requester. Per poter eseguire le task di una determinata campagna i worker dovranno iscriversi nel periodo definito dal requester per l'iscrizione (registration_start_date, registration_end_date) .

Pay

Ad ogni task è associato un metodo di pagamento per i worker che eseguono quel task. L'entità pay definisce il tipo di metodo possibile che nel nostro caso, per come è stato costruito il sito, sono solamente 4 fissati a priori:

- Contanti
- Coupon
- Oggetto in regalo
- Codice promozionale

La relazione pay_with assegna ad ogni task uno dei 4 tipi di pagamento e l'attributo pay_description indica il valore del pagamento e viene definito dal requester al momento della creazione del task.

Keyword

L'entità keyword rappresenta una singola keyword assegnata ai task o a un requester. Essendoci un'unica entità Keyword sia per i task che per i requester in questo modo si assicura che le stesse keyword che appaiono ai worker saranno le stesse che appaiono per le task.

2- Schema relazionale della base di dati

Schema relazionale

attributo = chiave primaria
attributo* = chiave esterna
{attributo1, attributo2}* = chiave esterna

Entità

- keyword(keyword, type)
- task(id, description, title, n_workers, valid_bit, campaign*, pay_type*, pay_description*)
- answer(task*,value)
- worker(user_name, password, name, surname)
- requester(user_name, password, name, surname, accepted)
- campaign(id, name, registration_start_date, registration_end_date, start_date, end_date, requester*)
- pay(type)

Relazioni

- requires_keyword(task*, keyword*)
- recives_task(task*, worker*, valid_bit_user)
- has_keyword(worker*, keyword*, score)
- join(worker*, campaign*, score)
- choose(worker*, {task, value}*)

Traduzione delle relazioni

- la relazione 1-N **contains** è stata tradotta aggiungendo come chiave esterna l'attributo *campaign* nella tabella task
- la relazione 1-N **manager** è stata tradotta aggiungendo come chiave esterna l'attributo *requester* nella tabella campaign
- la relazione 1-N **pay_with** è stata tradotta aggiungendo come chiave esterna l'attribut *pay_type* e l'attributo *pay_description* nella tabella campaign
- le relazione N-N **requires** è stata tradotta con la tabella **requires_keyword** aggiungendo come chiave primaria la coppia *(task, keyword)* che sono a loro volta due chiavi esterne delle tabelle task e keyword rispettivamente
- le relazione N-N **recives** è stata tradotta con la tabella **recives_task** aggiungendo come chiave primaria la coppia *(task, worker)* che sono a loro volta due chiavi esterne delle tabelle task e worker rispettivamente, più l'attributo valid_bit_user
- le relazione N-N **has** è stata tradotta con la tabella **has_keyword** aggiungendo come chiave primaria la coppia *(worker, keyword)* che sono a loro volta due chiavi esterne delle tabelle worker e keyword rispettivamente
- le relazione N-N **join** è stata tradotta con la tabella **join** aggiungendo come chiave primaria la coppia *(worker, campaign)* che sono a loro volta due chiavi esterne delle tabelle task e keyword rispettivamente, più l'attributo score

- la relazione 1-N **choose** è stata tradotta creando una tabella choose con *worker* come chiave primaria e con attributi *task* e *value.* Worker è chiave esterna della tabella worker e la coppia (task, value) è chiave esterna per la tabella answer.

3- Funzioni realizzate

answer_insert [TRIGGER]

```
CREATE TRIGGER answer_insert after insert on
   crowdsourcing.choose
for each row execute procedure check_majority();
```

Ogni qualvolta un utente worker sceglie una delle opzioni di risposta di un task avviene un inserimento nella tabella *choose*, evento che fa scattare questo trigger.
Il trigger in questione attiva la procedura *check_majority* la quale controlla se per quel task è stato superato il numero di risposte necessario a validare il task.
Nel caso in cui si ha un numero di risposte maggiore o uguale al numero di risposte richiesto la procedura effettua le seguenti operazioni:

- Nel caso in cui c'è una **percentuale superiore alla soglia richiesta** di utenti che hanno scelto una determinata domanda allora la procedura:
    - aggiorna a TRUE il *valid_bit_user* della tabella *recives_task* per tutti gli utenti worker che hanno risposto in accordo con la risposta corretta
    - rende valido il task aggiornando a TRUE il *valid_bit* della tabella *task* del task in questione
- Nel caso in cui **non esista una risposta** che risulti superare la percentuale richiesta allora la procedura
    - aggiorna a FALSE il *valid_bit* della tabella *task* del task in questione

La procedura per valutare il superamento della soglia utilizza la funzione *right_answer* che verrà spiegata più avanti. La funzione *right_answer* restituisce però NULL ogni qualvolta il campo *valid* della tabella *task* è diverso da TRUE, di conseguenza la procedura *check_majority* aggiorna momentaneamente il campo *valid* a TRUE e se la task non supera la soglia richiesta cambia nuovamente il campo a NULL.

worker_score_update [TRIGGER]

```
CREATE TRIGGER worker_score_update AFTER UPDATE of valid_bit
   ON crowdsourcing.task
FOR EACH ROW execute procedure score_update();
```

Questo trigger lavora in coppia con il precedente poiché si attiva ogni volta che il campo *valid_bit* della tabella *task* si aggiorna, ovvero ogni volta che una task raggiunge la soglia di risposte necessarie e vi è una risposta corretta.

Il trigger esegue la procedura *score_update* la quale aggiorna il campo *score* della tabella *joins_campaign* aumentando di uno il valore per tutti gli utenti worker che hanno risposto correttamente al task a cui è stato aggiornato il campo *valid_bit* a TRUE.
I worker che hanno risposto correttamente sono quelli che nella tabella *recives_task* hanno *valid_bit_user* uguale a TRUE.

rigth_answer

```
CREATE OR REPLACE FUNCTION right_answer(task_id INTEGER)
RETURNS TABLE(right_answer VARCHAR(100), num_ans BIGINT)
```

questa funzione restituisce il valore della risposta corretta di una determinata task presa come argomento e il numero di worker che hanno scelto quella risposta. L'output viene calcolato solamente se il task in questione è valido:

- nel caso in cui il task **non** fosse valido allora la funzione ritorna NULL
- se invece il task **fosse valido** vengono selezionate all'interno della tabella *choose* la o le risposte a quella task che hanno avuto più consensi tra i worker che hanno eseguito la task.

Per chiarezza dell'informazione quando si parla di task valido si intende un task che ha ottenuto il numero previsto di risposte e che vi è una o più di queste che ha una percentuale di risposte superiore a quella richiesta.

best_task

```
CREATE OR REPLACE FUNCTION best_task (INTEGER, varchar(20))
RETURNS TABLE(id INTEGER, title VARCHAR(50), description
   VARCHAR(280))
```

La funzione *best_task* è una delle più importanti poiché responsabile di assegnare ad un worker la task migliore di una determinata campagna. Prende come argomenti l'id della campagna e l'username del worker a cui si vuole assegnare il task e restituisce l'id, il titolo e la descrizione della task migliore.
L'algoritmo per la scelta della task prende in considerazione vari elementi:

- in primo luogo vengono selezionate solamente le task che fanno parte dell campagna e si controlla se la campagna è già iniziata e non è ancora scaduta.
- tra queste si prendono solamente le task che hanno il campo *valid_bit* ancora NULL, scartando dunque tutte le task che hanno superato la soglia di worker necessari per la risposta
- si selezionano solamente le task che hanno keyword uguali a quelle possedute dal worker
- poiché ogni worker può rispondere ad ogni task una singola volta, si scartano tutte le task a cui il worker ha già risposto

- il risultato della query viene ordinato in base al grado di conoscenza delle keyword del worker (viene restituita per prima la task che possiede la keyword con score più alto per l'utente)
- e infine viene restituito solamente la prima riga del risultato della query grazie ad un LIMIT 1

top10

```
CREATE OR REPLACE FUNCTION
   top10(crowdsourcing.campaign.id%TYPE)
RETURNS TABLE(worker crowdsourcing.joins_campaign.worker%TYPE,
   score crowdsourcing.joins_campaign.score%TYPE)
```

La funzione *top10* prende come argomento l'id di una campagna e restituisce l'username e lo score in quella campagna dei 10 worker che hanno score più alto.
Effettua il procedimento grazie ad una query che restituisce tutti i worker che partecipano a quella campagna ordinati a seconda del campo *score* e limitando l'output solamente alle prime 10 righe.

Rapporto campagna

Le funzioni che seguono hanno tutte lo scopo di creare le statistiche relative ad una specifica campagna. Si è deciso infatti di usare più funzioni con scopi diversi per arrivare al report di ogni singola campagna.

camp_task

```
CREATE OR REPLACE FUNCTION camp_tasks (INTEGER)
RETURNS TABLE(id INTEGER, description
   crowdsourcing.task.description%TYPE, title
   crowdsourcing.task.title%TYPE, n_workers
   crowdsourcing.task.n_workers%TYPE, threshold
   crowdsourcing.task.threshold%TYPE, valid_bit
   crowdsourcing.task.valid_bit%TYPE, campaign
   crowdsourcing.task.campaign%TYPE, pay_type
   crowdsourcing.task.pay_type%TYPE, pay_description
   crowdsourcing.task.pay_description%TYPE)
```

Questa funzione restituisce la tabella *task* contenente però solamente le tuple appartenenti ad una specifica campagna il cui id viene passato come argomento della funzione. Il sito utilizza questa funzione per ottenere un riassunto di tutte le task di una campagna.

task_result

```
CREATE OR REPLACE FUNCTION task_result (INTEGER)
RETURNS TABLE(task INTEGER, answer VARCHAR(100))
```

La funzione *task_result* restituisce una tabella fatta di tutte le task di una specifica campagna accompagnate ciascuna dalla propria risposta corretta (o da NULL eventualmente)

completed_task

```
CREATE OR REPLACE FUNCTION completed_task(INTEGER)
RETURNS SETOF INTEGER
```

La funzione *completed_task* restituisce una tabella fatta di tutte le task che sono state completate in una specifica campagna

completed_percentage

```
CREATE OR REPLACE FUNCTION completed_percentage(INTEGER)
RETURNS INTEGER
```

La funzione *completed_percentage* restituisce la percentuale di task che sono state completate contando quante task ci sono in totale nella campagna e facendone la percentuale con le task risultato della funzione *completed_task*

Statistiche worker in una campagna

Analogamente a quanto detto per la campagna queste funzioni servono per la creazione di un report con le statistiche di un determinato worker

standing_position

```
CREATE OR REPLACE FUNCTION
   standing_position(crowdsourcing.campaign.id%TYPE,
   crowdsourcing.worker.user_name%TYPE)
RETURNS table(pos BIGINT)
```

Prendendo come argomenti l'id di una campagna e l'username di un worker questa funzione restituisce la posizione in classifica del worker nella campagna in questione.

Per farlo la procedura utilizza una query che conta quanti altri utenti iscritti alla campagna hanno il valore del campo *score* della tabella *joins_campaign* maggiore rispetto all'utente preso in considerazione. Il numero trovato + 1 rappresenta la posizione in classifica del worker per quella campagna.

answered_task_num

```
CREATE OR REPLACE FUNCTION
   answered_task_num(crowdsourcing.worker.user_name%TYPE,
   crowdsourcing.campaign.id%TYPE)
RETURNS table(answered BIGINT)
```

Restituisce il numero di task di una campagna a cui il worker ha dato una risposta selezionando tutte le occorrenze nella tabella *choose* che combaciano con il worker e la campagna suddetti. L'username del worker e l'id della campagna sono passati come argomenti.

correct_task_num

```
CREATE OR REPLACE FUNCTION
   correct_task_num(crowdsourcing.worker.user_name%TYPE,
   crowdsourcing.campaign.id%TYPE)
RETURNS table(correct BIGINT)
```

La funzione *correcr_task_num* restituisce tutti i task di una campagna a cui un worker ha dato risposta corretta. Ovvero tutte le task la cui risposta data dall'utente è risultata essere poi quella corretta una volta che il task viene validato. Funziona in modo analogo alla precedente con la differenza che vengono selezionate solamente le righe della tabella *choose* il cui campo *valid_bit_user* ha valore TRUE.

Statistiche worker

Le seguenti funzioni realizzano delle statistiche relative al singolo worker indipendentemente da una campagna

answered_task_num

```
CREATE OR REPLACE FUNCTION
   answered_task_num(crowdsourcing.worker.user_name%TYPE)
RETURNS table(answered BIGINT)
```

Questa funzione è un overloading della precedente *answered_task_num*. Le due funzioni infatti hanno lo stesso nome ma differente numero di argomenti. Questa in particolare

prende come argomento solamente l'username del worker e calcola quindi il numero di task a cui il worker ha dato una risposta considerando tutte le campagne a cui è iscritto.

correct_task_num

```
CREATE OR REPLACE FUNCTION
   correct_task_num(crowdsourcing.worker.user_name%TYPE)
RETURNS table(correct BIGINT)
```

Il medesimo discorso fatto per la funzione precedente vale anche per questa. Si tratta infatti di un overloading di *correct_task_num* che prende però come unico argomento lo username di un worker. La funzione restituisce il numero di task a cui l'utente ha risposto correttamente, indipendentemente dalla campagna.

4 - Informazioni aggiuntive

Gestione delle keyword associate a task e worker

Come scelta progettuale in fase di definizione dello schema ER si è deciso di utilizzare un'unica entità *keyword* per rappresentare le parole chiave di entrambi i task e i worker. In questa maniera esiste una sola lista condivisa di keyword disponibili e consente all'applicazione web di eseguire il confronto facendo un confronto sintattico tra le keyword che appartengono al worker e quelle richieste dall'utente. Vengono assegnate al worker solamente le task con keyword che hanno un'esatta corrispondenza con le keyword dell'utente.
Il confronto però non tiene in considerazione del tipo di keyword; l'utente può scegliere se considera la sua keyword come una conoscenza o una attitudine, ma il confronto non utilizza questa informazione, controlla solamente che ci sia corrispondenza tra i nomi.

Definizione del profilo di un lavoratore

Il profilo del lavoratore viene definito dal lavoratore stesso al momento della registrazione tramite una form che richiede:

- username
- password
- Nome
- Cognome

Il profilo è modificabile dal worker in ogni momento nell'apposita pagina.
L'utente viene creato senza alcuna skill poiché è lui stesso che, tramite l'apposita pagina aggiunge le keyword che preferisce. Al momento dell'aggiunta di una nuova keyword viene chiesto inoltre al lavoratore di inserire un punteggio da 0 a 10 relativo alla conoscenza della skill che si sta aggiungendo.
In questo modo nel profilo di ogni lavoratore ci sono solamente le keyword da lui scelte e non altre.

Algoritmo di scheduling utilizzato per assegnare i task ai lavoratori

L'algoritmo di scheduling per l'assegnamento dei task viene implementato grazie alla funzione best_task che seleziona le task che:

- fanno parte di campagne con data di inizio e di scadenza valide
- hanno il campo *valid_bit* ancora NULL, scartando dunque tutte le task che hanno superato la soglia di worker necessari per la risposta
- hanno almeno una keyword uguale a quelle possedute dal worker
- poiché ogni worker può rispondere ad ogni task una singola volta, si scartano tutte le task a cui il worker ha già risposto

il risultato della query utilizzata dalla funzione viene ordinato in base al grado di conoscenza delle keyword del worker (viene restituita per prima la task che possiede la keyword con score più alto per l'utente). Infine viene restituito solamente la prima riga del risultato della query grazie ad un LIMIT 1.