CPSC 304 Project Cover Page

Milestone #: 4

Date: 29 November 2024

Group Number: 26 .

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Kelvin Hang How Mah	57129686	r5i5o	kelvinmah02@gmail.com
Rioto Oka	54645734	q8b5g	okarioto@gmail.com
Aadesh Mehra	39288733	a5f9m	aadeshm03@gmail.com

Group 26

Repository Link:
https://github.students.cs.ubc.ca/CPSC304-2024W-T1/project_a5f9m_q8b5g_r5i5o

Project Overview
Auction Database Management System

Our project is an auction system that allows users from the auction company side to make changes and/or update databases containing information regarding auction sessions, auctioneers, items, delivery vehicles, storage facilities, appraisers, and customers. The database tracks the logistics of moving auctioned items from storage to auction houses as well as the status of items including the bids they have and condition. The database also provides a mechanism for sale and auction management and customer transactions, where the company can review who has bought an item or sold an item. The user can review the items to be sold along with their current selling status, enhancing efficiency and transparency in the auction process.
For our final schema, the structure of the relationships remained the same except for our Items entity, where we changed it to a partial overlapping. Partial was chosen since it made the most sense that items could exist without being in inventory or for sale. To prevent repetitive deletion to fit the disjoint constraints of inventory items and for_sale items, we decided it made more sense that items could exist in both tables (overlapping). This makes it easier for the user so when items are up for sale they don't constantly have to be removed first. We also used denormalized tables in our database for easier querying of complex query requests.

INSERT Query:
```
`INSERT INTO Items_Sells (itemID, category, startDate, creditCardNumber)
VALUES (:itemID, :category, TRUNC(TO_DATE(:startDate,'YYYY-MM-DD')),
:creditCardNumber)`;                               (LINE 174 in appService.js)
```

UPDATE Query:
```
`UPDATE AuctionSession_Operates                       (Line 386 in appService.js)
        SET ${setClauseString}
        WHERE event_date = TO_DATE(:auctiondate, 'YYYY-MM-DD')
        AND time = :time
        AND address = :address`
```

DELETE Query:
```
`DELETE FROM Items_Sells WHERE itemID = :itemID;    (LINE 202 in appService.js)
```

SELECTION Query:
```
`SELECT * FROM DeliveryVehicle                       (Line 333 in appService.js)
 WHERE licensePlate = '${plate}' ${plateNload} payloadCapacity ${loadcomp}
${payload} ${loadNmake} makeModel = '${make}'`
```

PROJECTION Query:
```
`SELECT ${selectedColumns} FROM Auctioneer`         (Line 373 in appService.js)
```

Group 26

JOIN Query:
```
`SELECT appraiserName, itemID                         (Line 280 in appService.js)
 FROM Appraiser a, Appraises ap
 WHERE a.appraiserID = ap.appraiserID
 AND a.appraiserID = :appraiserID`,
 [appraiserID]`,
```

AGGREGATION GROUP BY Query:
```
`SELECT itemID, MAX(amount) AS maxBid                 (Line 296 in appService.js)
 FROM Bids_Receive_Places
 WHERE itemID = :itemID
 GROUP BY itemID`,
 [itemID],
```

***This query returns the largest bid placed on the selected itemID.**

AGGREGATION HAVING Query:
```
`SELECT address, phoneNumber                       (Line 347 in appService.js)
 FROM AuctionSession_Operates
 GROUP BY address, phoneNumber HAVING min(numItems) >= ${numberItems}`
```

***This query returns the address and phoneNumber of the auction session(s) with at least X number of items sold.**

NESTED GROUP BY Query:
```
'SELECT A1.event_date, AVG(A1.commissionPercent)   (Line 360 in appService.js)
FROM AuctionSession_Operates A1
GROUP BY A1.event_date
HAVING AVG(A1.commissionPercent) > ( SELECT AVG(A2.commissionPercent)
FROM  AuctionSession_Operates A2)'
```

***This query returns the days with a higher commission percent than average.**

DIVISION Query:
```
`SELECT DISTINCT licensePlate                      (Line 312 in appService.js)
 FROM CollectsFrom C
 WHERE NOT EXISTS
      ((SELECT address FROM StorageFacility)
       MINUS
      (SELECT CF.address FROM CollectsFrom CF WHERE CF.licensePlate =
C.licensePlate))`
```

***This query returns the DeliveryVehicle that has been to every StorageFacility.**