OBJECT DESCRIPTIONS

**SEQUENCES:**

We used sequences to automatically generate unique identifier values for each new record, ensuring data integrity and eliminating duplicate keys. Our sequences, named billing\_id\_gen, contract\_id\_gen, customer\_id\_gen, delmgr\_id\_gen, delper\_id\_gen, delreq\_id\_gen, delivery\_id\_gen, eval\_id\_gen, leads\_id\_gen, ndstf\_id\_gen, noncinv\_id\_gen, pymt\_id\_gen, sales\_id\_gen, wtrcoolers\_id\_gen, cool\_rent\_id\_gen, and stop\_id\_gen are created with a starting value of 1 and increment by 1, caching 10 values for performance improvement.

**VIEWS:**

1. customer\_contracts
   1. This view joins the customer organization table with the contracts table using the customer\_id to present a consolidated snapshot of each customer’s contracts. It displays key contract details such as start and end dates, estimated frequency, and estimated monthly cost, making it easy to review and monitor customer agreements.
2. delivery\_and\_driver\_info
   1. This view combines delivery personnel details with corresponding delivery records by matching the delivery\_person\_id, offering a unified view of driver performance and delivery outcomes. It includes essential information such as delivery dates, driver comments, and customer feedback, which helps in tracking operational efficiency and service quality.

**PROCEDURES:**

1. remove\_old\_evaluations
   1. This procedure automatically deletes evaluation records from the EVAL table for delivery personnel who haven't made a delivery in over three years, based on the latest delivery date from the DLVRY table. It uses a subquery to identify those delivery persons whose last recorded delivery occurred more than 36 months ago, and then removes all corresponding evaluations while outputting the number of records deleted.
2. update\_late\_fees
   1. This procedure calculates and updates late fees on unpaid bills by determining the sum of non-cooler inventory prices associated with each sale and applying a 10% rate, further adjusted by the number of months the bill is overdue. It updates only those bills with a status of 'Unpaid' that are past their due dates, ensuring that late fee charges are accurately reflected in the billing records.

**FUNCTIONS:**

1. get\_delivery\_person\_stats
   1. This function gathers and returns key performance statistics for a given delivery person for a specified month and year. It calculates total revenue generated from non-cooler item sales, counts the total number of deliveries and sales, and tallies the number of leads for that period, returning these figures via a SYS\_REFCURSOR. If any errors occur during execution, the function returns a cursor with error indicator values.
2. rented\_cooler
   1. This function checks whether a specified customer currently rents a water cooler by first retrieving the customer's organization name and then counting any associated rental records from the cool\_rent table. It returns a descriptive message indicating the rental status of the water cooler for that customer. In cases where the customer is not found or another error occurs, the function returns an appropriate error message.

**PACKAGES:**

1. contract\_actions
   1. This package, contract\_actions, encapsulates procedures for managing customer contracts at Dark Springs, covering the entire lifecycle of a contract. The find\_expiring\_contracts procedure queries and displays contracts that are due to expire within the next 30 days, enabling proactive outreach to customers for renewals. The renew\_contract procedure retrieves details of an active contract for a given customer, deactivates it, and then creates a new contract with an extended term, ensuring continuous service. The create\_new\_contract procedure enforces a rule that prevents duplicate active contracts by checking for existing agreements before inserting a new record, while the terminate\_contracts procedure cleanly deactivates any active contracts for a customer, effectively archiving them. Together, these procedures streamline contract management and enforce business rules, making contract maintenance efficient and reliable.
2. sales\_actions
   1. The sales\_actions package centralizes operations related to processing and updating sales transactions. The create\_new\_sale procedure initiates a new sale by generating a unique sale ID, inserting a corresponding record into the sales table, and then associating available non-cooler inventory items with that sale, while ensuring that the requested quantities are available in inventory. The update\_sale procedure enables modifications to an existing sale by swapping out an old product for a new one, updating the inventory accordingly, and then marking the sale as approved. Together, these procedures streamline sales processing, enforce inventory integrity, and provide robust error handling with transactional control, making it easier to maintain accurate and reliable sales records.

**TRIGGERS:**

1. referral\_bonus\_trigger
   1. This trigger fires after an update on the LEADS table when a lead's approval status changes from not approved to approved ('Yes'). It then deducts a $25 referral bonus from the corresponding customer's total balance in the CUST\_ORG table, effectively rewarding the customer for a successful referral.
2. collect\_deposit\_trigger
   1. This trigger executes immediately after a new record is inserted into the COOL\_RENT table, indicating that a customer has rented a water cooler. It updates the customer's total balance in the CUST\_ORG table by adding $25 to reflect the deposit collected for the rental.
3. return\_deposit\_trigger
   1. This trigger is activated after a record is deleted from the COOL\_RENT table, which typically occurs when a rental is terminated or a cooler is returned. It subtracts $25 from the customer's total balance in the CUST\_ORG table, effectively processing the return of the deposit.

**SCHEDULED JOB:**

1. monthly\_billing
   1. This scheduled job, monthly\_billing, runs on the first day of every month and automates the process of calculating and updating monthly billing information for customer organizations. It calls the Calc\_Send\_Monthly\_Billing procedure, which aggregates sales and non-cooler inventory costs to determine each customer's total billing cost and updates their total balance in the customer table accordingly. After that, the job sets the billing status from 'Active' to 'Awaiting Payment' to prompt follow-up on outstanding balances. This automation ensures that billing data is current and that customers are consistently notified of their payment obligations.

**ROLES:**

1. DELIVERY\_MANAGER\_EVAL: This role has the authority to view and modify the EVAL Table. This is to enable the delivery manager to add and view the evaluation scores and comments for the delivery personnel.
2. CHRO: This role enables the Cheif HR Officer to only view the EVAL table. It allows the employee to view how the delivery personnel are performing and make decisions about how to increase and enhance their productivity.

**ALTERNATE INDEX:** Creating an alternate index for delivery employees enhances query efficiency by optimizing searches based on common access patterns. Currently, the DEL\_MGR table has DELIVERY\_MANAGER\_ID as its primary key, and the DEL\_PER table has DELIVERY\_PERSON\_ID as its primary key, with a foreign key reference to DELIVERY\_MANAGER\_ID. A frequent query pattern may involve searching for a delivery employee—either a manager or personnel—by their name or retrieving all personnel under a specific manager. Without an alternate index, these queries require a full table scan, which becomes inefficient as data grows. By introducing an alternate index on commonly searched attributes, such as first and last names or the DELIVERY\_MANAGER\_ID in DEL\_PER, the database can quickly locate relevant records without scanning the entire dataset.

**DENORMALIZATION:**

Denormalization improves efficiency by reducing the need for frequent joins, which can be costly in terms of performance. In the current structure, delivery is the most operationally intensive process as it involves a lot of stakeholders and employees. In our database, retrieving both a delivery person’s details and their manager’s name requires joining the DEL\_PER and DEL\_MGR tables. Every time this query runs, the database must match records, which becomes inefficient as data volume increases. By creating a denormalized table DEL\_PER\_MGR, we precompute and store the manager’s name along with the delivery person's details, eliminating the need for a join at query time. This significantly enhances read performance, reduces database load, and speeds up reporting and dashboards, making it ideal for applications where fetching data is more frequent than updating relationships. While it introduces some redundancy, the trade-off is worthwhile in read-heavy scenarios, as it ensures faster access to critical information.

**REPORTS:**

1. Unpaid Bills: This report finds the customers who have not paid the bill. This will allow the business to send reminder messages to them to get them to pay their bill amounts

SQL Query:

SELECT co.name, co.customer\_id,b.late\_fee, b.billing\_id

FROM bills b LEFT JOIN cust\_org co ON co.customer\_id = b.customer\_id

WHERE b.status = 'Awaiting Payment'

1. High Performing Delivery Person: This gives the information about the top performing delivery personnel, which might allow the management to incentivize these people with bonuses and benefits to boost their morale and motivate others to do better.

SQL Query:

SELECT dp.first\_name || ' ' || dp.last\_name name, dp.delivery\_manager\_id, e.total\_score, e.comments

FROM del\_per dp JOIN eval e ON dp.delivery\_person\_id = e.delivery\_person\_id

WHERE total\_score >= 8