CS4380/7380 Database Management Systems –I
Final Project Final Report

Project title: Aureate Artifact

Group 4

Team members

Last name	Fist name	E-mail	Major
Chettri	Nishant	nc5ff@mail.missouri.edu	CS
Neupane	Roshan	rlnzq8@mail.missouri.edu	CS
Besalke	Charles	ceb445@mail.missouri.edu	CS
Li	Yu	ylrbc@mail.missouri.edu	CS

1. Introduction

1.1. Surendra's Tibetan Thangka Treasure

The client is a local store, situated in Kathmandu, Nepal between the hustle and bustle of Thamel. The store was started by Surendra Rajbhandary, about 40 years ago by the name Surendra's Tibetan Thangka Treasure. The name however, hasn't changed since then. Currently, there is only a single store, which sells various types of artifacts. These artifacts are bought by natives and tourists alike. There are about 5-6 employees currently working at the store.

The client is looking to expand their business internationally and are looking for a management solution for their entire system. The name that they have selected for the customer website is 'Aureate Artifacts'. Clients would like to incorporate an online portal where they can manage the system as well as have an online portal through which they can sell the artifacts internationally. Along with that, having a point-of-sale system, directly connected to the database would make their job of managing these records much easier.

The artifacts generally are divided into three different types. They have Decorative Weapons which locally are called "Khukuri"s. The other two types of artifacts are Thangkas, religious paintings on scrolls, and Singing Bowls. All the types of artifact hold huge traditional symbolism. These artifacts come with different designs, engravings and sizes making them unique and of a great concern to the art-loving people- locally and internationally.

Data collection was done in order to properly design the database. Because the clients are planning to extend the service globally, the database design would have to incorporate additional tables and relationships, which are currently non-existent. For instance, contact with courier services, online web access, etc.

1.2. Data Collection

To collect the existing data, we scanned through logbooks and sales receipts and identified the necessary attributes, entities and their relationships. This allowed us to create a proper database design. As mentioned before, since the existing system requires additional entities and relationships, the data collection was an evolutionary process. Regular online conferencing with the client gave rise to new information and were updated regularly in our database design.

Since online records do not exist for the client, to populate our database, we will have to create dummy data as a starting point. After successful testing, data will be transferred from the logbooks to the database.

1.3. Application Domain

The website was developed using PHP along with HTML, CSS and Javascript libraries. The project is divided into 2 main parts, the customer website and the client website. The use of the application is explained in detail in the "User Manual" section of this document. Sample usernames and password to use each for each of these sections are given below.

- Customer side User [Uniform Permission]

  Customer-side username: ray@abc.com

  Customer-side password: apple

- Client Side Users
  - Permission level: Super admin

  Client-side username: alaiho

  Client-side password: nothing
  - Permission level: Admin

Client-side username: akuopalla

Client-side password: else

- Permission level: Manager

Client-side username: lulrich

Client-side password: matters

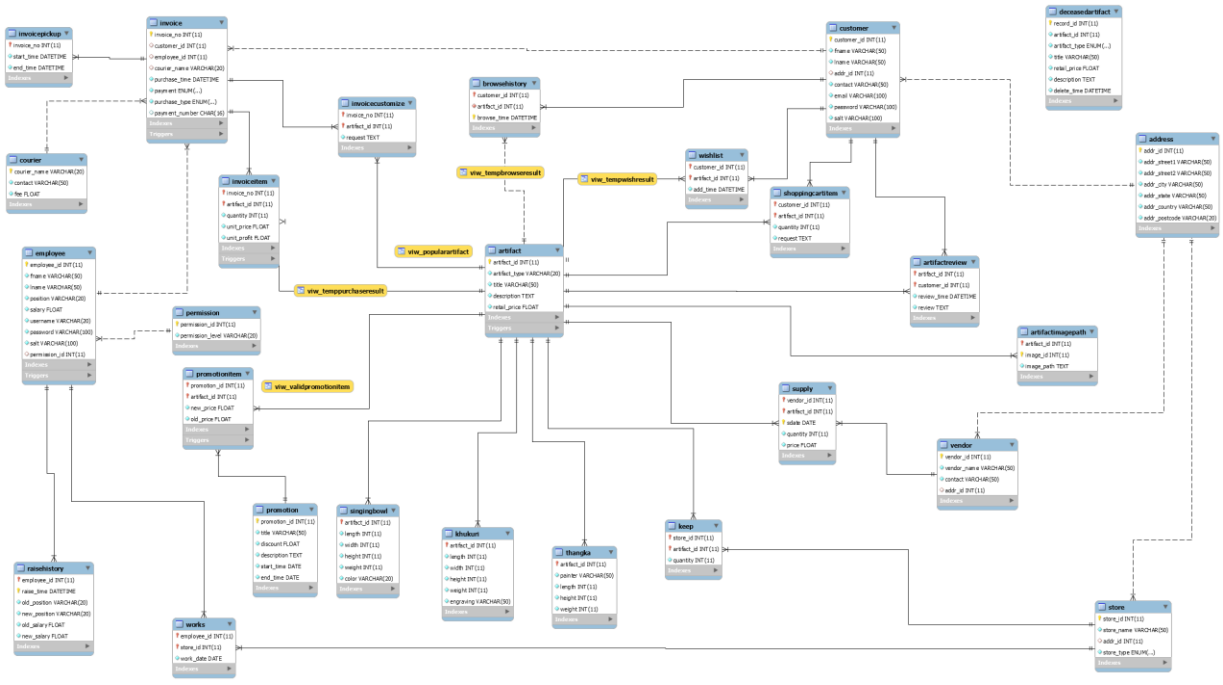
- Permission level: Sales

Client-side username: jhetfield

Client-side username: nothing

2. E-R Diagram

The ERD shown below represents the database schema using the crow's foot notation. It includes 25 tables with 4 hidden views, which have been used on the front end to display the reports. An enlarged version of the diagram can be found at the end of this report.



The DDLs for table creation and their description is listed below.

2.1. Address

The address table is used for storing the addresses for customers, employees, warehouses and anything that requires address information. It was created into its own table to allow multiple entities to share an address while reducing redundancies.

```
CREATE TABLE `Address` (
  `addr_id` int(11) NOT NULL DEFAULT '0',
  `addr_street1` varchar(50) NOT NULL,
  `addr_street2` varchar(50) NOT NULL,
  `addr_city` varchar(50) NOT NULL,
  `addr_state` varchar(50) NOT NULL,
```

```
  `addr_country` varchar(50) NOT NULL,
  `addr_postcode` varchar(20) NOT NULL,
  PRIMARY KEY (`addr_id`),
  KEY `IDX_Addresscity` (`addr_city`,`addr_country`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.2. Artifact

The artifact superclass stores the major common information about each artifact along with the type of artifact to make querying easier.

```
CREATE TABLE `Artifact` (
  `artifact_id` int(11) NOT NULL DEFAULT '0',
  `artifact_type` varchar(20) NOT NULL,
  `title` varchar(50) NOT NULL,
  `description` text NOT NULL,
  `retail_price` float NOT NULL,
  PRIMARY KEY (`artifact_id`),
  KEY `IDX_Artifactitile` (`title`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.3. ArtifactImagePath

The artifact image path uses the artifact id as primary key and stores the image URLs for each artifact.

```
CREATE TABLE `ArtifactImagePath` (
  `artifact_id` int(11) NOT NULL DEFAULT '0',
  `image_id` int(11) NOT NULL DEFAULT '0',
  `image_path` text NOT NULL,
  PRIMARY KEY (`artifact_id`,`image_id`),
  CONSTRAINT `ArtifactImagePath_ibfk_1` FOREIGN KEY (`artifact_id`)
REFERENCES `Artifact` (`artifact_id`) ON DELETE CASCADE ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.4. ArtifactReview

The artifact review table similarly stores the artifact id as primary key and stores the reviews given by customers on each artifact.

```
CREATE TABLE `ArtifactReview` (
  `artifact_id` int(11) NOT NULL DEFAULT '0',
  `customer_id` int(11) NOT NULL DEFAULT '0',
  `review_time` datetime NOT NULL,
  `review` text NOT NULL,
  PRIMARY KEY (`artifact_id`,`customer_id`),
  KEY `customer_id` (`customer_id`),
  CONSTRAINT `ArtifactReview_ibfk_1` FOREIGN KEY (`artifact_id`)
REFERENCES `Artifact` (`artifact_id`) ON DELETE CASCADE ON UPDATE
CASCADE,
  CONSTRAINT `ArtifactReview_ibfk_2` FOREIGN KEY (`customer_id`)
REFERENCES `Customer` (`customer_id`) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.5. BrowseHistory

The browse history table stores the information for each artifact being browsed by the customer along with the time of browse.

```
CREATE TABLE `BrowseHistory` (
```

```
  `customer_id` int(11) NOT NULL DEFAULT '0',
  `artifact_id` int(11) NOT NULL,
  `browse_time` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  PRIMARY KEY (`customer_id`,`browse_time`),
  KEY `artifact_id` (`artifact_id`),
  CONSTRAINT `BrowseHistory_ibfk_1` FOREIGN KEY (`customer_id`)
REFERENCES `Customer` (`customer_id`) ON DELETE CASCADE ON UPDATE
CASCADE,
  CONSTRAINT `BrowseHistory_ibfk_2` FOREIGN KEY (`artifact_id`)
REFERENCES `Artifact` (`artifact_id`) ON DELETE CASCADE ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.6. Courier

The courier table stores the information about the couriers, which have a relationship with the company.

```
CREATE TABLE `Courier` (
  `courier_name` varchar(20) NOT NULL DEFAULT '',
  `contact` varchar(50) NOT NULL,
  `fee` float NOT NULL,
  PRIMARY KEY (`courier_name`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.7. Customer

The customer table stores information of all the registered customers. It also uses the address id as a foreign key constraint.

```
CREATE TABLE `Customer` (
  `customer_id` int(11) NOT NULL DEFAULT '0',
  `fname` varchar(50) NOT NULL,
  `lname` varchar(50) NOT NULL,
  `addr_id` int(11) DEFAULT NULL,
  `contact` varchar(50) NOT NULL,
  `email` varchar(100) NOT NULL,
  `password` varchar(100) NOT NULL,
  `salt` varchar(100) NOT NULL,
  PRIMARY KEY (`customer_id`),
  UNIQUE KEY `IDX_Customeremail` (`email`),
  KEY `addr_id` (`addr_id`),
  KEY `IDX_Customerlname` (`lname`),
  CONSTRAINT `Customer_ibfk_1` FOREIGN KEY (`addr_id`) REFERENCES
`Address` (`addr_id`) ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.8. DeceasedArtifact

Any artifact deleted from the system is inserted into the following table by a trigger.

```
CREATE TABLE `DeceasedArtifact` (
  `record_id` int(11) NOT NULL DEFAULT '0',
  `artifact_id` int(11) NOT NULL,
  `artifact_type` enum('Thangka','Khukuri','SingingBowl','Other')
NOT NULL,
  `title` varchar(50) NOT NULL,
  `retail_price` float NOT NULL,
  `description` text NOT NULL,
  `delete_time` datetime NOT NULL,
```

```
  PRIMARY KEY (`record_id`),
  KEY `IDX_DeceasedArtifacttitle` (`title`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.9. Employee

The employee table stores information about each employee and uses id from the permission table as a foreign key constraint to assign permissions on the database.

```
CREATE TABLE `Employee` (
  `employee_id` int(11) NOT NULL DEFAULT '0',
  `fname` varchar(50) NOT NULL,
  `lname` varchar(50) NOT NULL,
  `position` varchar(20) NOT NULL,
  `salary` float NOT NULL,
  `username` varchar(20) NOT NULL,
  `password` varchar(100) NOT NULL,
  `salt` varchar(100) NOT NULL,
  `permission_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`employee_id`),
  UNIQUE KEY `IDX_Employeeusername` (`username`),
  KEY `permission_id` (`permission_id`),
  KEY `IDX_Employeefname` (`fname`),
  CONSTRAINT `Employee_ibfk_1` FOREIGN KEY (`permission_id`)
REFERENCES `Permission` (`permission_id`) ON DELETE SET NULL ON
UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.10. Invoice

The invoice table stores the high level invoice details for each customer. Any employee associated with the sale's id is also stored here as a foreign key constraint along with the courier name.

```
CREATE TABLE `Invoice` (
  `invoice_no` int(11) NOT NULL DEFAULT '0',
  `customer_id` int(11) DEFAULT NULL,
  `employee_id` int(11) DEFAULT NULL,
  `courier_name` varchar(20) DEFAULT NULL,
  `purchase_time` datetime NOT NULL,
  `payment` enum('Credit','Debit','Cash','Check') NOT NULL,
  `purchase_type` enum('Local','Online') NOT NULL,
  `payment_number` char(16) DEFAULT '0000000000000000',
  PRIMARY KEY (`invoice_no`),
  KEY `customer_id` (`customer_id`),
  KEY `employee_id` (`employee_id`),
  KEY `courier_name` (`courier_name`),
  KEY `IDX_Invoicetime` (`purchase_time`),
  CONSTRAINT `Invoice_ibfk_1` FOREIGN KEY (`customer_id`)
REFERENCES `Customer` (`customer_id`) ON DELETE SET NULL ON UPDATE
CASCADE,
  CONSTRAINT `Invoice_ibfk_2` FOREIGN KEY (`employee_id`)
REFERENCES `Employee` (`employee_id`) ON DELETE SET NULL ON UPDATE
CASCADE,
  CONSTRAINT `Invoice_ibfk_3` FOREIGN KEY (`courier_name`)
REFERENCES `Courier` (`courier_name`) ON DELETE SET NULL ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.11. InvoiceCustomize

Any customization made by the customer when ordering their items are stored in this table along with the customer and artifact id.

```
CREATE TABLE `InvoiceCustomize` (
  `invoice_no` int(11) NOT NULL DEFAULT '0',
  `artifact_id` int(11) NOT NULL DEFAULT '0',
  `request` text NOT NULL,
  PRIMARY KEY (`invoice_no`,`artifact_id`),
  KEY `artifact_id` (`artifact_id`),
  CONSTRAINT `InvoiceCustomize_ibfk_1` FOREIGN KEY (`invoice_no`)
REFERENCES `Invoice` (`invoice_no`) ON DELETE CASCADE ON UPDATE
CASCADE,
  CONSTRAINT `InvoiceCustomize_ibfk_2` FOREIGN KEY (`artifact_id`)
REFERENCES `Artifact` (`artifact_id`) ON DELETE CASCADE ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.12. InvoiceItem

The invoice item table stores the details of the purchase. It uses the invoice table as the foreign and primary key along with the artifact ids in each record.

```
CREATE TABLE `InvoiceItem` (
  `invoice_no` int(11) NOT NULL DEFAULT '0',
  `artifact_id` int(11) NOT NULL DEFAULT '0',
  `quantity` int(11) NOT NULL,
  `unit_price` float NOT NULL,
  `unit_profit` float NOT NULL,
  PRIMARY KEY (`invoice_no`,`artifact_id`),
  KEY `artifact_id` (`artifact_id`),
  CONSTRAINT `InvoiceItem_ibfk_1` FOREIGN KEY (`invoice_no`)
REFERENCES `Invoice` (`invoice_no`) ON DELETE CASCADE ON UPDATE
CASCADE,
  CONSTRAINT `InvoiceItem_ibfk_2` FOREIGN KEY (`artifact_id`)
REFERENCES `Artifact` (`artifact_id`) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.13. InvoicePickup

The invoice pickup table is triggered when a customer chooses to pick up the order from the store itself. This creates a record entry with the start and end time duration being 1 week from the time the order is placed. This is to allow the company to send a reminder to any customer whose orders have not yet been picked up even after a week.

```
CREATE TABLE `InvoicePickup` (
  `invoice_no` int(11) NOT NULL DEFAULT '0',
  `start_time` datetime NOT NULL,
  `end_time` datetime NOT NULL,
  PRIMARY KEY (`invoice_no`),
  CONSTRAINT `InvoicePickup_ibfk_1` FOREIGN KEY (`invoice_no`)
REFERENCES `Invoice` (`invoice_no`) ON DELETE CASCADE ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.14. Keep

The keep table stores the information regarding the stock of artifacts being kept at each store or warehouse.

```
CREATE TABLE `Keep` (
  `store_id` int(11) NOT NULL DEFAULT '0',
  `artifact_id` int(11) NOT NULL DEFAULT '0',
  `quantity` int(11) NOT NULL,
  PRIMARY KEY (`store_id`,`artifact_id`),
  KEY `artifact_id` (`artifact_id`),
  CONSTRAINT `Keep_ibfk_1` FOREIGN KEY (`store_id`) REFERENCES
`Store` (`store_id`) ON UPDATE CASCADE,
  CONSTRAINT `Keep_ibfk_2` FOREIGN KEY (`artifact_id`) REFERENCES
`Artifact` (`artifact_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.15. Khukuri

The khukuri table stores the details about the Khukuri artifacts

```
CREATE TABLE `Khukuri` (
  `artifact_id` int(11) NOT NULL DEFAULT '0',
  `length` int(11) NOT NULL,
  `width` int(11) NOT NULL,
  `height` int(11) NOT NULL,
  `weight` int(11) NOT NULL,
  `engraving` varchar(50) NOT NULL,
  PRIMARY KEY (`artifact_id`),
  CONSTRAINT `Khukuri_ibfk_1` FOREIGN KEY (`artifact_id`)
REFERENCES `Artifact` (`artifact_id`) ON DELETE CASCADE ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.16. Permission

The permission table is used for creating various permission levels

```
CREATE TABLE `Permission` (
  `permission_id` int(11) NOT NULL DEFAULT '0',
  `permission_level` varchar(20) NOT NULL,
  PRIMARY KEY (`permission_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.17. Promotion

The promotion table stores the various promotional offers on artifacts provided by the company. Indexes on promotion time are created to help make queries for promotional items easier.

```
CREATE TABLE `Promotion` (
  `promotion_id` int(11) NOT NULL DEFAULT '0',
  `title` varchar(50) NOT NULL,
  `discount` float NOT NULL,
  `description` text NOT NULL,
  `start_time` date NOT NULL,
  `end_time` date NOT NULL,
  PRIMARY KEY (`promotion_id`),
  KEY `IDX_Promotiontime` (`start_time`,`end_time`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.18. PromotionItem

The promotion item table stores the list of artifacts, which have had promotional offers placed on them along with a record of their old and new prices.

```
CREATE TABLE `PromotionItem` (
  `promotion_id` int(11) NOT NULL DEFAULT '0',
  `artifact_id` int(11) NOT NULL DEFAULT '0',
  `new_price` float NOT NULL,
  `old_price` float NOT NULL,
  PRIMARY KEY (`promotion_id`,`artifact_id`),
  KEY `artifact_id` (`artifact_id`),
  CONSTRAINT `PromotionItem_ibfk_1` FOREIGN KEY (`promotion_id`)
REFERENCES `Promotion` (`promotion_id`) ON DELETE CASCADE ON UPDATE
CASCADE,
  CONSTRAINT `PromotionItem_ibfk_2` FOREIGN KEY (`artifact_id`)
REFERENCES `Artifact` (`artifact_id`) ON DELETE CASCADE ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.19. RaiseHistory

Information about any raise given to the employees are stored here.

```
CREATE TABLE `RaiseHistory` (
  `employee_id` int(11) NOT NULL DEFAULT '0',
  `raise_time` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  `old_position` varchar(20) NOT NULL,
  `new_position` varchar(20) NOT NULL,
  `old_salary` float NOT NULL,
  `new_salary` float NOT NULL,
  PRIMARY KEY (`employee_id`,`raise_time`),
  CONSTRAINT `RaiseHistory_ibfk_1` FOREIGN KEY (`employee_id`)
REFERENCES `Employee` (`employee_id`) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.20. ShoppingCartItem

Shopping cart item stores information temporarily for each customer when an artifact is added to the cart by them. The information for each customer is cleared when they confirm their purchase, which moves every item from the table to the invoice and invoice items.

```
CREATE TABLE `ShoppingCartItem` (
  `customer_id` int(11) NOT NULL DEFAULT '0',
  `artifact_id` int(11) NOT NULL DEFAULT '0',
  `quantity` int(11) NOT NULL,
  `request` text NOT NULL,
  PRIMARY KEY (`customer_id`,`artifact_id`),
  KEY `artifact_id` (`artifact_id`),
  CONSTRAINT `ShoppingCartItem_ibfk_1` FOREIGN KEY (`customer_id`)
REFERENCES `Customer` (`customer_id`) ON DELETE CASCADE ON UPDATE
CASCADE,
  CONSTRAINT `ShoppingCartItem_ibfk_2` FOREIGN KEY (`artifact_id`)
REFERENCES `Artifact` (`artifact_id`) ON DELETE CASCADE ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.21. SingingBowl

Singing bowls is a subclass of artifacts and stores information only about singing bowls.

```
CREATE TABLE `SingingBowl` (
  `artifact_id` int(11) NOT NULL DEFAULT '0',
  `length` int(11) NOT NULL,
  `width` int(11) NOT NULL,
  `height` int(11) NOT NULL,
  `weight` int(11) NOT NULL,
  `color` varchar(20) NOT NULL,
  PRIMARY KEY (`artifact_id`),
  CONSTRAINT `SingingBowl_ibfk_1` FOREIGN KEY (`artifact_id`)
REFERENCES `Artifact` (`artifact_id`) ON DELETE CASCADE ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.22. Store

Store table keeps a record of all the stores owned by the company, including the type of stores and uses address table as a foreign key constraints to store the address.

```
CREATE TABLE `Store` (
  `store_id` int(11) NOT NULL DEFAULT '0',
  `store_name` varchar(50) NOT NULL,
  `addr_id` int(11) DEFAULT NULL,
  `store_type` enum('Shop','Warehouse') NOT NULL,
  PRIMARY KEY (`store_id`),
  KEY `addr_id` (`addr_id`),
  KEY `IDX_Storename` (`store_name`),
  CONSTRAINT `Store_ibfk_1` FOREIGN KEY (`addr_id`) REFERENCES
`Address` (`addr_id`) ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.23. Supply

The supply table stores how many artifacts have been sold to the company by which vendor along with sale quantity per artifact and the date and quantity of item sold by the vendor.

```
CREATE TABLE `Supply` (
  `vendor_id` int(11) NOT NULL DEFAULT '0',
  `artifact_id` int(11) NOT NULL DEFAULT '0',
  `sdate` date NOT NULL DEFAULT '0000-00-00',
  `quantity` int(11) NOT NULL,
  `price` float NOT NULL,
  PRIMARY KEY (`vendor_id`,`artifact_id`,`sdate`),
  KEY `artifact_id` (`artifact_id`),
  KEY `IDX_Supplydate` (`sdate`),
  CONSTRAINT `Supply_ibfk_1` FOREIGN KEY (`vendor_id`) REFERENCES
`Vendor` (`vendor_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `Supply_ibfk_2` FOREIGN KEY (`artifact_id`) REFERENCES
`Artifact` (`artifact_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.24. Thangka

The thangka table stores information about the thangka artifact and like the other subclasses of artifact, uses the artifact id as its foreign and primary key.

```
CREATE TABLE `Thangka` (
  `artifact_id` int(11) NOT NULL DEFAULT '0',
  `painter` varchar(50) NOT NULL,
  `length` int(11) NOT NULL,
```

```
  `height` int(11) NOT NULL,
  `weight` int(11) NOT NULL,
  PRIMARY KEY (`artifact_id`),
  CONSTRAINT `Thangka_ibfk_1` FOREIGN KEY (`artifact_id`)
REFERENCES `Artifact` (`artifact_id`) ON DELETE CASCADE ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2.25. Vendor

The vendor table stores information about the vendors and their addresses using address id.

```
CREATE TABLE `Vendor` (
  `vendor_id` int(11) NOT NULL DEFAULT '0',
  `vendor_name` varchar(50) NOT NULL,
  `contact` varchar(50) NOT NULL,
  `addr_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`vendor_id`),
  KEY `addr_id` (`addr_id`),
  KEY `IDX_Vendorname` (`vendor_name`),
  CONSTRAINT `Vendor_ibfk_1` FOREIGN KEY (`addr_id`) REFERENCES
`Address` (`addr_id`) ON DELETE SET NULL ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

3. Queries

Listed below are useful queries that for our database design and application. The description of their function and application is provided before each SQL statement.

3.1. To obtain max ID from a table

This query was particularly helpful while creating new record IDs (primary keys) for multiple tables.

```
SELECT max(addr_id) as max FROM Address;
```

3.2. To list all the types of Khukuri/Singing Bowl/Thangka

Khukuri has been used in this instance.

```
SELECT A.title, K.*
FROM Khukuri K, Artifact A
WHERE K.artifact_id=A.artifact_id;
```

3.3. To check the items and the profits made from them

This query was useful in populating the items with the total number of items sold and the profit made from them.

```
SELECT artifact_id, SUM(unit_profit), SUM(quantity)
FROM InvoiceItem
GROUP BY artifact_id;
```

3.4. To list the employees with the profits they have made

This query lists out the names of employees and the total sales they have made so far.

```
SELECT E.employee_id, E.fname, SUM(II.unit_price) as Total_sales
FROM Employee E, Invoice I, InvoiceItem II
WHERE E.employee_id=I.employee_id AND I.invoice_no=II.invoice_no
GROUP BY E.employee_id;
```

3.5. To check for the profits made from the items from different vendors

This query was useful in obtained the profits earned from the various suppliers. It has been used to generate the analytical reports as well.

```
SELECT S.vendor_id, SUM(I.unit_profit) as profit
FROM Supply S, InvoiceItem I
WHERE S.artifact_id=I.artifact_id
GROUP BY S.vendor_id;
```

3.6. To enlist the Employees with their payroll

This query is useful in obtaining the monthly salary of each employee and presenting them to the person in-charge of distributing each employee's salary.

```
SELECT fname, lname, salary, salary/12 as monthly_pay
FROM Employee;
```

3.7. To check the profits made through items within a specified date

This query is useful in identifying how many artifacts have been sold within a certain duration and the total profit earned from each of the artifacts.

```
SELECT II.artifact_id, SUM(II.unit_profit), SUM(II.quantity)
FROM InvoiceItem II, Invoice I
WHERE I.invoice_no=II.invoice_no
    AND I.purchase_time BETWEEN "2016-01-01" AND "2016-05-01"
GROUP BY artifact_id;
```

3.8. To calculate the profits from customer

The following produces the total number of customers per country and the total profit earned from each country. This was particularly useful in creating the Sales by Location report.

```
SELECT a.addr_country, COUNT(c. customer_id) AS Number_of_Customer,
SUM(II.unit_profit) AS Profit
FROM Address a, Customer c, Invoice I, InvoiceItem II
WHERE a.addr_id=c.addr_id
    AND c.customer_id=I.customer_id
    AND II.invoice_no=I.invoice_no
GROUP BY a.addr_country;
```

3.9. To check the profits made specified by the item type

The profits made by each type of artifacts are obtained in this query along with the total number of artifacts sold per type. This is also useful in generating the Sales by Artifact Type report.

```
SELECT A.artifact_type, SUM(I.quantity), SUM(I.unit_profit)
FROM InvoiceItem I, Artifact A
WHERE I.artifact_id=A.artifact_id
GROUP BY A.artifact_type;
```

3.10. To identify the country that has the highest total sales online

This query is used when there is the need of knowing the demand of the artifacts based on the country. To scale the items made available country wise.

```
FROM Address A, Customer C, Invoice I, InvoiceItem II
WHERE A.addr_id=C.addr_id AND C.customer_id=I.customer_id
    AND I.invoice_no=II.invoice_no
GROUP BY A.addr_country
HAVING SUM(II.unit_price)=(
```

```
    SELECT MAX(sale_country)
    FROM (SELECT SUM(I2.unit_price) AS sale_country
        FROM Address A2, Customer C2, Invoice I1, InvoiceItem I2
        WHERE A2.addr_id=C2.addr_id
            AND C2.customer_id=I1.customer_id
            AND I1.invoice_no = I2.invoice_no
        GROUP BY A2.addr_country) AS tbl_sale);
```

3.11. List the vendors that supply all the decorations with certain specification (weight>50 and height<4)

Whenever there is need of knowing which vendor supply artifact with a certain dimension, so that the client can easily make the orders given the specifications.

```
SELECT V.vendor_name
FROM Vendor V
WHERE NOT EXISTS (
    SELECT K.artifact_id
    FROM Khukuri K
    WHERE K.weight<=50
        OR K.height>=4
        AND K.artifact_id NOT IN (
            SELECT S.artifact_id
            FROM Supply S
            WHERE S.vendor_id=V.vendor_id));
```

3.12. Most used courier service

This query is useful to know the choices made by the customers to get their items delivered. They obviously considered the things such as cheapness and effectiveness of the courier service. To have a better bond with the service providers.

```
SELECT courier_name
FROM Invoice I
WHERE courier_name!='PickUp'
GROUP BY courier_name
HAVING COUNT(I.invoice_no)=(
    SELECT MAX(cnt)
    FROM (SELECT COUNT(DISTINCT invoice_no) AS cnt
        FROM Invoice
        WHERE courier_name!='PickUp'
        GROUP BY courier_name) AS tbl_cnt);
```

3.13. Identify the leading vendor (yields highest total sales so far)

This query is useful to realize which vendor is supplying the artifacts that are of great interests to the customers.

```
SELECT V.vendor_name, V.vendor_id
FROM Vendor V, Supply S, InvoiceItem I
WHERE V.vendor_id=S.vendor_id AND I.artifact_id=S.artifact_id
GROUP BY S.vendor_id
HAVING SUM(I.unit_price)=(
    SELECT MAX(t_sale)
    FROM (SELECT SUM(I2.unit_price) AS t_sale
        FROM Supply S2, InvoiceItem I2
        WHERE S2.artifact_id=I2.artifact_id
        GROUP BY S2.vendor_id) AS tbl_sale);
```

3.14. To check the type of payment and the courier service used by a particular customer

This query is useful whenever there is a need of reviewing the customer's moves. Upon requested by the customer or even by the client, to affirm the courier and/or the payment type a particular customer has used.

```
SELECT C.fname, C.lname, C.customer_id, I.courier_name, I.payment,
I.purchase_time
FROM Customer C, Invoice I
WHERE I.customer_id=C.customer_id AND C.customer_id="98";
```

3.15. Get all the Invoice Items given an Invoice Number

To check the quantity of artifacts purchased by a particular customer at once.

```
SELECT A.title, II.*
FROM Invoice I, Invoiceitem II, Artifact A
WHERE I.customer_id=121
    AND I.invoice_no = 3
    AND I.invoice_no = II.invoice_no
    AND A.artifact_id = II.artifact_id;
```

3.16. Get all the items bought by a particular customer

For tracking down what are the preferences of a particular customer so that we can show related ones once they login, we need this type of listings.

```
SELECT C.fname, C.lname, C.customer_id, A.title
FROM Customer C, Invoice I, InvoiceItem II, Artifact A
WHERE II.invoice_no=I.invoice_no
    AND I.customer_id=C.customer_id
    AND II.artifact_id=A.artifact_id
    AND I.customer_id=121;
```

3.17. Update the title, description and the price of an artifact

This query can be useful to change the price, the name and the description used for a particular artifact.

```
UPDATE Artifact
SET title='Long tipped',
    description='This type of weapon was used for…',
    retail_price=5000
WHERE artifact_id=5;
```

3.18. Insert a new artifact

This type of query can be used to add new artifacts.

```
INSERT INTO Artifact (artifact_id, title, description,
retail_price, artifact_type)
VALUES (45, 'Long tipped', 'This type of weapon was used
for…',5000, 'Khukuri');
```

3.19. Get the invoice number from a receipt with only a few details visible/comprehensible with a provided customer detail

This type of queries are of great importance when a customer/client wishes to make verify a certain purchase from a receipt which is vague.

```
SELECT I.*, C.customer_id
FROM Invoice I, Customer C
WHERE I.customer_id=C.customer_id
```

```
    AND invoice_no LIKE '1&&23';
```

3.20. Remove the items from Shopping Cart

To remove the items that are present in the shopping cart once they are moved into invoice i.e. bought.

```
DELETE FROM ShoppingCartItem WHERE customer_id = "121";
```

3.21. Get the customer information for those customer who have shopped artifacts of price more than NRs. 20000 in one go

This type of queries are useful where there are some sale/discount schemes for which customers who make a purchase of amount of a specified threshold.

```
SELECT I.invoice_no, I.customer_id, C.fname, C.lname,
SUM(II.unit_price*II.quantity)
AS Total_Amount
FROM Customer C, Invoice I, InvoiceItem II
WHERE I.invoice_no=II.invoice_no
    AND I.customer_id=C.customer_id
    AND I.invoice_no=2;
```

3.22. Add a new column

This type of queries are important if there is in need of an unforeseen attribute for a certain entity.

```
ALTER TABLE store
ADD COLUMN store_name varchar(200)
AFTER store_id;
```

4. Analytics

The analytics are useful to represent the data in a graphical as well as tabular format, so that the client will be able to analyze the information stored in the database. These reports will be used by them in order to improve their sales and profits.

The following subsections show the sample results as examples of the available analytics functions. One of the example is showing the results of Popular Artifacts. The graphical and tabular representation of this example were produced by the system utilizing the following views and queries.

4.1. Views

The following queries have been used to create views for popular artifact display. The result is used for easier analytics function implementation as well as more efficient queries for list generation.

To create temporary browse history view:

```
CREATE
DEFINER='root'@'localhost'
VIEW VIW_tempBrowseResult (artifact_id, num_browsed)
AS SELECT artifact_id, COUNT(*) AS num_browsed
    FROM BrowseHistory
    GROUP BY artifact_id
    HAVING COUNT(*)>3;
```

To create temporary purchase history view:

```
CREATE
DEFINER='root'@'localhost'
VIEW VIW_tempPurchaseResult (artifact_id, num_purchase)
AS SELECT artifact_id, SUM(quantity) AS num_purchase
```

```
    FROM InvoiceItem
    GROUP BY artifact_id
    HAVING SUM(quantity)>3;
```

To create temporary wishlist history view:

```
CREATE
DEFINER='root'@'localhost'
VIEW VIW_tempWishResult (artifact_id, num_wishlist)
AS SELECT artifact_id, COUNT(*) AS num_wishlist
    FROM WishList
    GROUP BY artifact_id
    HAVING COUNT(*)>3;
```

Merges all the above views in a single view to show popular artifact view:

```
CREATE
DEFINER='root'@'localhost'
VIEW VIW_PopularArtifact (artifact_id, artifact_title,
num_purchase, num_wishlist, num_browsed)
AS SELECT VIW_tempBrowseResult.artifact_id, Artifact.title,
VIW_tempPurchaseResult.num_purchase,
VIW_tempWishResult.num_wishlist, VIW_tempBrowseResult.num_browsed
FROM VIW_tempBrowseResult LEFT JOIN VIW_tempPurchaseResult ON
(VIW_tempBrowseResult.artifact_id=VIW_tempPurchaseResult.artifact_i
d)
    LEFT JOIN VIW_tempWishResult ON
(VIW_tempBrowseResult.artifact_id=VIW_tempWishResult.artifact_id)
    LEFT JOIN Artifact ON
(VIW_tempBrowseResult.artifact_id=Artifact.artifact_id);
```

4.2. Queries for Analytical Functions

4.2.1. Sales by day

For the purpose of getting accounts maintained and reviewed daily, we can have this query utilized.

```
SELECT DATE(Invoice.purchase_time) AS date,
    SUM(InvoiceItem.unit_price*InvoiceItem.quantity) AS total_sale,
    SUM(InvoiceItem.unit_profit*InvoiceItem.quantity) AS
total_profit,
    SUM(InvoiceItem.quantity) AS total_quantity
FROM InvoiceItem, Invoice
WHERE Invoice.invoice_no=InvoiceItem.invoice_no
    AND Invoice.purchase_time BETWEEN '2016-01-01' AND '2016-05-01'
GROUP BY DATE(Invoice.purchase_time);
```

4.2.2. Sales by hour

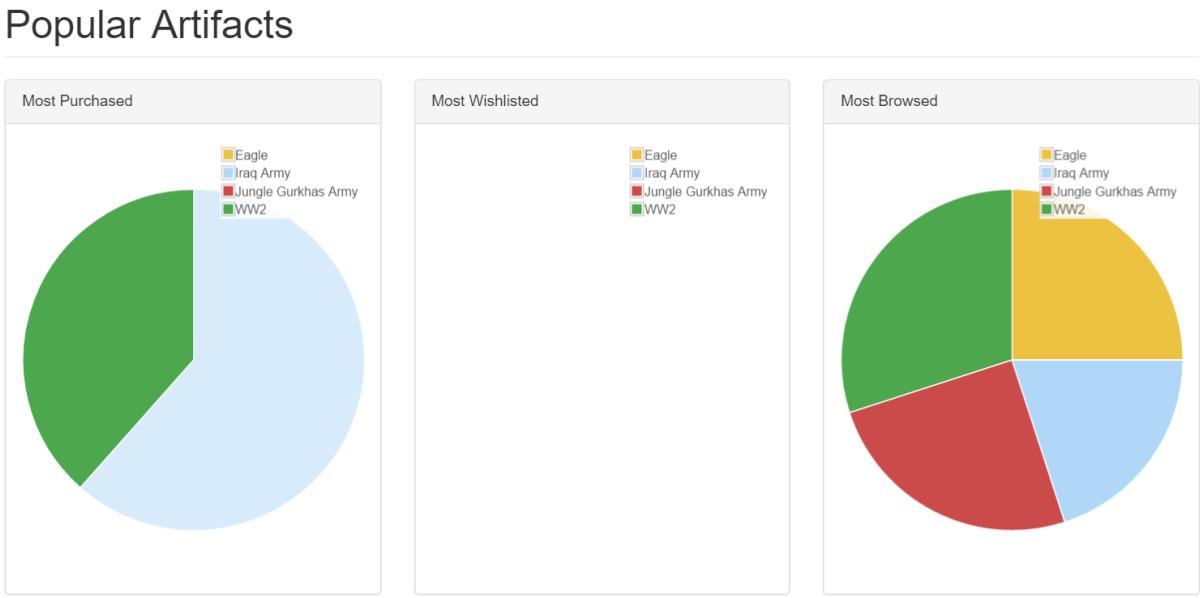
This holds the same importance but then is checked by hours.

```
SELECT HOUR(Invoice.purchase_time) AS hour,
    SUM(InvoiceItem.unit_price*InvoiceItem.quantity) AS total_sale,
    SUM(InvoiceItem.unit_profit*InvoiceItem.quantity) AS
total_profit,
    SUM(InvoiceItem.quantity) AS total_quantity
FROM InvoiceItem, Invoice
WHERE Invoice.invoice_no=InvoiceItem.invoice_no
    AND Invoice.purchase_time BETWEEN '2016-01-01' AND '2016-05-01'
```

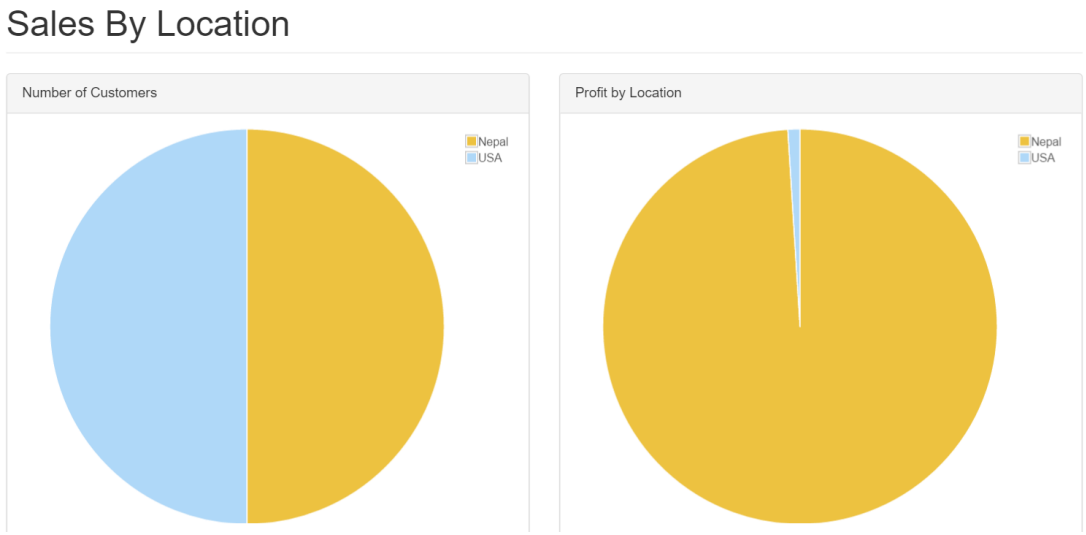
```
GROUP BY DATE(Invoice.purchase_time);
```

4.3. Pie Charts

- The representation of popular artifacts using Pie Charts:

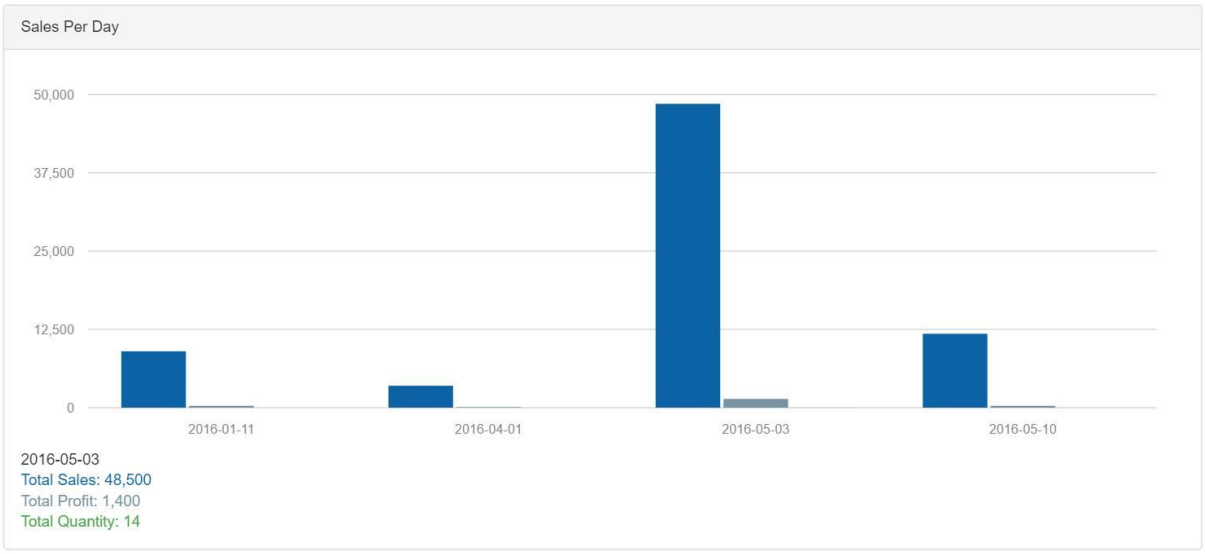


- The representation of sales by location using pie charts:

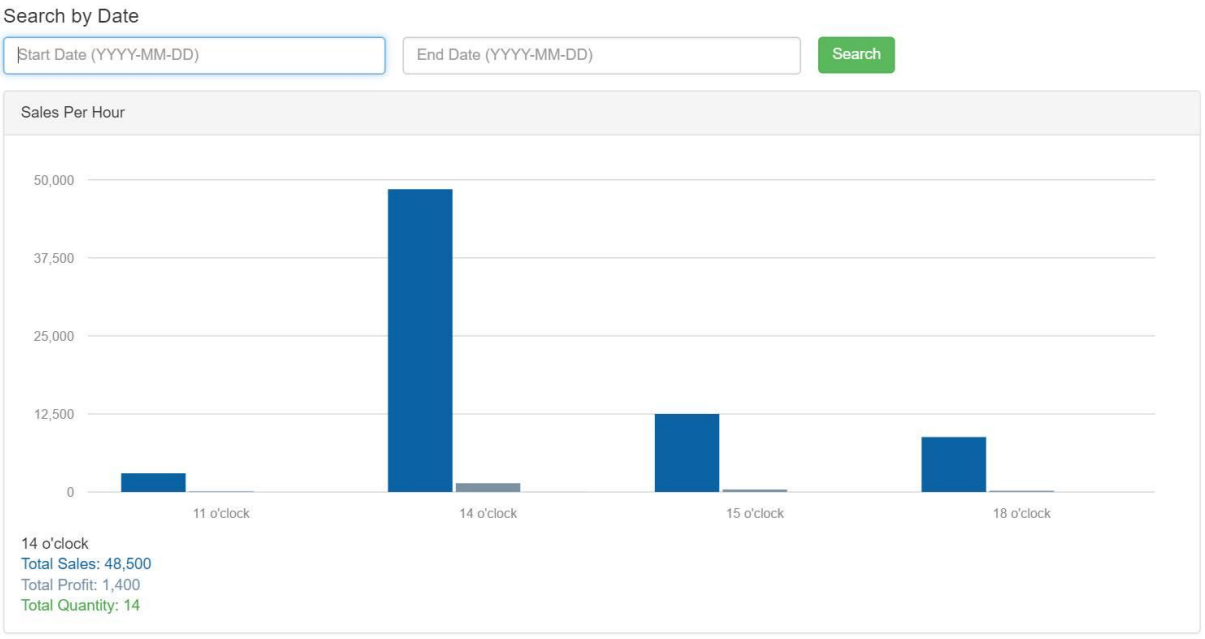


4.4. Bar Charts

- Bar chart representation of the sales per day:



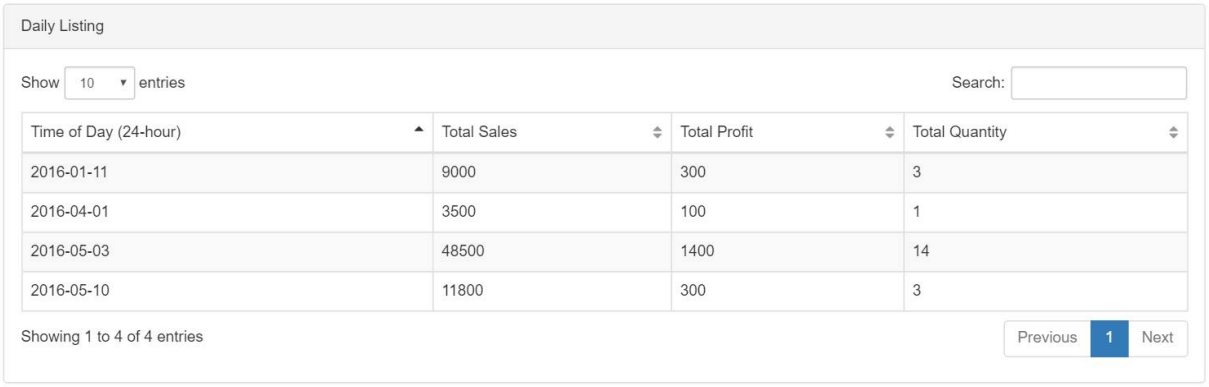
- Bar chart representation of sales per hour with the interface to select the start and end time:



4.5. Tables

- The table listing all the artifacts with the quantity sold and the profit earned:

- The table to check the total sales made and the total profit on a daily basis:


Daily Listing

Show 10 entries Search:

Time of Day (24-hour)	Total Sales	Total Profit	Total Quantity
2016-01-11	9000	300	3
2016-04-01	3500	100	1
2016-05-03	48500	1400	14
2016-05-10	11800	300	3

Showing 1 to 4 of 4 entries Previous 1 Next

5. Normalization

After the first complete design of the database, we identified the that the Purchase and Promotion tables are the major tables which were not in BCNF. For this report, normalization processes of Purchase and Promotion is presented as example.

5.1. Purchase

The original Purchase relation has the following attributes: invoice_no, artifact_id, customer_id, employee_id, courier_name, purchase_time, quantity, unit_price, payment, purchase_type, payment_number, unit_profit. By denoting the attributes using capital letters A to L following the same order, we have $R_{Purchase}$=ABCDEFGHIJKL. According to our client, we have the following functional dependencies: $FD_{Purchase}$={AB→GHL, A→CDEFIJK}. The key for $R_{Purchase}$ is {AB}.

Clearly the functional dependency A→CDEFIJK violates the criteria for BCNF because A is only part of key. This FD also makes $R_{Purchase}$ disqualified for 3NF since CDEFIJ is not part of a key. A being part of the key AB further makes $R_{Purchase}$ not in 2NF and only in 1NF.

$R_{Purchase}$ is decomposed into two relations $R_{Invoice}$=ACDEFIJK and $R_{InvoiceItem}$ = ABGHL. This decomposition is loss-less. Both dependencies are directly preserved in the two decomposed relations, so it is also a dependency-preserving decomposition. The result is the following two tables:

- Invoice: invoice_no, customer_id, employee_id, courier_name, purchase_time, payment, purchase_type, payment_number
- InvoiceItem: invoice_no, artifact_id, quantity, unit_price, unit_profit

5.2. Promotion

The normalization of Promotions relation followed the same principle. The original Promotions relation has attributes: promotion_id, artifact_id, title, discount, description, start_time, end_time, new_price, old_price. Also representing the attributes with capital letters A to I we have $R_{Promotions}$=ABCDEFGHI. In the current situation, each promotion event is designed to have a unified title and discount value. So the functional dependencies are: $FD_{Promotions}$={AB→HI, A→CDEFG}. The key for $R_{Promotions}$ is {AB}.

Similar to the original Purchase relation, the functional dependency A→CDEFG, which has a partial key in the left-hand side and no partial key on the right-hand side, makes $R_{Promotions}$ violates the criteria for BCNF, 3NF and 2NF. So $R_{Promotions}$ is in 1NF as well.

Decomposing $R_{Promotions}$ with respect to A→CDEFG results two relations $R_{Promotion}$=ACDEFG and $R_{PromotionItem}$=ABHI. This decomposition is also a loss-less and dependency-preserving decomposition. The decomposed tables are:

- Promotion: promotion_id, title, discount, description, start_time, end_time
- PromotionItem: promotion_id, artifact_id, new_price, old_price

5.3. Denormalization

It was argued if the original Purchase table should be denormalized after normalization when we continued developing the function for sales report. Because joining the two decomposed tables cannot be avoided for the function, denormalizing these two tables will make queries easier. However considering the original Purchase table had 12 attributes, we settled for the design with normalized tables. Future development could favor denormalization if performance was greatly hindered.

6. Index Selection

According to the usage habit of the client, it is unnatural and counterintuitive to use ID directly. We have provided search and filtering functionalities for tasks such as locating an invoice or employee record or creating a sales report. The most heavily used functions were collected after consulting our client. These functions can be categorized into time-based and text-based with the target object. Indices were created following this category.

Currently all the indices are stored using B-Tree, the only permissible index type for the default single server based engine InnoDB. Although some queries may prefer Hash index type, it is not available for our application under MySQL framework. The vulnerability of MEMORY engine against crashes makes it less ideal for our application. The distributed nature of NDB leaves only InnoDB and MyISAM engine, both of which supports only B-Tree.

However, most of the searches and report generation are using ranged queries. Inability to use Hash index is not considered a great loss.

6.1. Time-based

The time-based functions are most frequently used in generating reports at the back-end of the website. It could also be potentially useful when locating an invoice or checking for a specific promotion. Additional indices for time were created to speed up such searches.

```
CREATE INDEX IDX_Invoicetime ON Invoice (purchase_time);
CREATE INDEX IDX_Supplydate ON Supply (sdate);
CREATE INDEX IDX_Promotiontime ON Promotion (start_time, end_time);
```

6.2. Text-based

Indices on text can reduce the time it takes to locate a certain artifact or a customer when using names as search criteria. It is also extremely beneficial for the Address table to have these indices since all values in the tables are plain text. These type of indices could potentially benefit from the FULLTEXT index type. However it is not applicable to the InnoDB engine.

```
CREATE INDEX IDX_Addresscity ON Address (addr_city, addr_country);
CREATE INDEX IDX_Artifactitile ON Artifact (title);
CREATE INDEX IDX_Customerlname ON Customer (lname);
CREATE INDEX IDX_DeceasedArtifacttitle ON DeceasedArtifact (title);
CREATE INDEX IDX_Employeefname ON Employee (fname);
CREATE INDEX IDX_Storename ON Store (store_name);
CREATE INDEX IDX_Vendorname ON Vendor (vendor_name);
```

6.3. UNIQUE

In addition to the primary keys in each table, two unique keys were specified to enforce data integrity. For the current application, each registered customer has to have a valid and unique email address, and each employee has to create a unique username. The verification of email format is easier to implement on the webpage. Unique constraint is naturally supported by MySQL and thus the following unique indices were added.

```
CREATE UNIQUE INDEX IDX_Customeremail ON Customer (email);
```

```
CREATE UNIQUE INDEX IDX_Employeeusername ON Employee (username);
```

7. Optimization and Tuning

Various optimization techniques were applied to overcome some inefficiencies identified during development. Each of them is explained in the subsections.

7.1. Additional Attributes

In order to increase querying performance, some additional attributes were added to the tables at a small cost of storage space. This optimization is performed to the operations involve frequent arithmetic or logic expressions. Two examples are given below.

The first example is adding 'artifact_type' attribute in the table Artifact. The original design with only subclasses but no 'artifact_type' attribute is sufficient to distinguish the type of an artifact. However the process for determining the type requires a series of conditional selection. Having an additional attribute for type avoids the potential problem of a linear increase in query time to the number of types.

Another example is 'unit_profit' attribute in InvoiceItem. This attribute is a derived attribute calculated by finding the difference between the actual invoice price and the averaged the supplying price of an artifact. This query involves aggregation and arithmetic operations, both of which hinders the performance. Instead of performing this calculation thousands of times every time a sales report is being generated, an added attribute, which is calculated by a trigger on insertion of new entry to InvoiceItem, greatly improves query time at a small cost of storage space.

7.2. Vertical Decompositions

In addition to the decomposition discussed in the Normalization chapter, some tables were vertically decomposed even though they are in BCNF. The processed was performed to the attributes which are not frequently used.

One example is the Address table. The address is used in Customer, Store and Vendor tables. There are 6 attributes of potentially long strings for a standard address and the format is identical for all three tables. Another phenomenon shared by the three tables is the address is rarely retrieved or searched. To reduce this overhead we created stand-alone entity for address. The system will only query from the Address tables when it is absolutely need to. This normalization will reduce the load time for querying these three tables.

Another example for vertical decomposition is the InvoiceCustomize table. If it is incorporated in InvoiceItem table, there will be no violation of BCNF. However only a small percentage of order will have special customization requests. It is more efficient for both storage and performance to form the special request into a weak entity of InvoiceItem.

7.3. NULL Value Prevention

During optimization processes, the NULL values are prevented by specifying NOT NULL constraints for almost all attributes when creating tables. One reason for avoiding NULL values in the tables is to increase query efficiency. More importantly, NULL value could lead to erroneous results for aggregation and arithmetic operations, which are critical for sales report. For an application where aggregation is fairly frequently used, it is important to eliminate NULL values to ensure result correctness.

7.4. Further Optimization

One possible optimization could be beneficial is horizontal decomposing the Invoice and InvoiceItem table. It is not clear how many sales record our client is willing to transfer to the new system at the moment. If it the number of old record exceeds a certain amount, horizontally decomposing the two tables using a threshold on date will make the query more efficient.

8. Security Setting

8.1. Database Access Control

We have created restrictions on the different portions of the site so that the various kinds of users of the website all have different permissions. The access control is not implemented at the database level, but similar ideas of role based discretionary access control is implemented for the website.

The public users who visit the site only have the ability to view artifacts and promotions. The registered customers have the same access as public users, but they also have the ability to view their purchase histories and invoices, as well as add things to their wishlists and carts.

By creating various permission levels in the website for the employees, we have been able to achieve discretionary access control security measures for the database.

The Sales users have the ability to read product information and have the ability to create invoices, as well as correct current customer invoices (in case a customer needs a refund or needs to have something added or removed from their invoice. Management can access everything, and can update product that is for sale on the website.

Admins and Super Admins have the ability to access, update, delete, and add all data, so that they can clean data that has been contaminated and can do any job that needs to be done on the database system. The Admins are able to also manage employees, which is a feature exclusive to Admins and Super Admins.

Depending on the requirements of the clients, the permissions for the reports will need to be changed in the future. Since the views have been created in order to produce the reports, any user not allowed to access these view, will not have access to them.

The web-based database access control has the following advantages:

- Prevent SQL injection by using bound parameters when executing SQL statements via PHP, which prepares the SQL and keeps unintended commands from being run
- Prevent Cross-Site-Scripting by enforcing who can insert data into the database
- The ability to enforce a least-access control model by enforcing who can edit data

Two major benefit will be discussed in the following subsection.

8.2. Other Security Settings

In addition to the database security, we have also implemented website security in terms of the following:

8.2.1. Salted-Hashed Passwords

Similarly, we have opted for a salted-hashed password storage solution. In this solution, we take the password that the user provides, add a random value to it and execute a hashing function. The result along with the random value is stored in the database. This way, we can recreate the operations that was done on the password input by the user by applying the same operations, and compare it to our resulted password, and if the entered password matches the stored password, we can authenticate the user.

```
$db_pass = $res['password'];
$salt = $res['salt'];
$saltedPassword = $password . $salt;
$convertedPassword = hash("sha256", $saltedPassword);
if ($convertedPassword == $db_pass){
    $match = 1;
}
```

The benefits of handling passwords in this manner in a database is that in the event of a data breach, the passwords are more difficult to crack, and in the time it would take for a malicious individual to crack

each password, we could inform all of the users of the website to both change their password, and change their passwords on other sites if they use the password in other places. This also ensures that a DBA is unable to see user's credentials, and prevents malicious DBAs from stealing user credentials.

8.2.2. Prevention of SQL Injection

Another security measure applied to the system is prevention of SQL injection from the website. To enforce this measure, bound parameters are being used.

```
public function getImageByID($artifact_id){
    $dbcon= new connect();
    $qry= $dbcon->db1->prepare("Select * from artifactimagepath
where artifact_id = :artifact_id");
    $qry->bindParam(":artifact_id",$artifact_id, PDO::PARAM_INT);
    $qry->execute();
    return $qry;
}
```

By using the bindParam function, we are able to make sure the function recognizes the type of value being sent to the database and that it corresponds to the type of data expected. Along with this, any special characters, which are used to escape SQL and inject malicious codes are prevented from doing so.

9. Other Topics

We have implemented triggers for different purposes. Also there are other features of the system

9.1. Triggers

Triggers were added for auto-completion and auto-calculation in some attributes when inserting new data into InvoiceItem, PromotionItem. Another three triggers were implemented for automatically adding entries into DeceasedArtifact, RaiseHistory and InvoicePickup on the event of delete, update and insert other tables respectively. SQL for the triggers are provided below.

9.1.1. Profit calculation for InvoiceItem

This trigger is used to calculate the profit of a sold artifact when InvoiceItem is inserted with new data.

```
DELIMITER $$
CREATE
DEFINER='root'@'localhost'
TRIGGER TGG_calIIProfit
BEFORE INSERT ON InvoiceItem
FOR EACH ROW
BEGIN
    DECLARE supply_price FLOAT;
    SET supply_price=(SELECT AVG(price) FROM Supply WHERE
artifact_id=NEW.artifact_id);
    SET NEW.unit_profit=NEW.unit_price-supply_price;
END $$
DELIMITER ;
```

9.1.2. Sale price calculation for PromotionItem

This trigger automatically calculates the sale price when an artifact is put on promotion.

```
DELIMITER $$
CREATE
DEFINER='root'@'localhost'
```

```
TRIGGER TGG_calPromotionItemPrice
BEFORE INSERT ON PromotionItem
FOR EACH ROW
BEGIN
    DECLARE temp_price FLOAT;
    SET temp_price=(SELECT retail_price FROM Artifact WHERE
artifact_id=NEW.artifact_id);
    SET NEW.old_price=temp_price,
        NEW.new_price=temp_price*(SELECT discount FROM Promotion
WHERE promotion_id=NEW.promotion_id);
END $$
DELIMITER ;
```

9.1.3. Raise history recording for employee

This trigger automatically adds position or salary change when the Employee table gets updated.

```
DELIMITER $$
CREATE
DEFINER='root'@'localhost'
TRIGGER TGG_addRaiseHistory
BEFORE UPDATE ON Employee
FOR EACH ROW
BEGIN
    INSERT INTO RaiseHistory
    SET employee_id=OLD.employee_id,
        raise_time=NOW(),
        old_position=OLD.position,
        new_position=NEW.position,
        old_salary=OLD.salary,
        new_salary=NEW.salary;
END $$
DELIMITER ;
```

9.1.4. Pickup information recording for Invoice

This trigger automatically adds new tuple into the InvoicePickup table when ‘Pickup’ is selected when checking out.

```
DELIMITER $$
CREATE
DEFINER='root'@'localhost'
TRIGGER TGG_addInvoicePickup
AFTER INSERT ON Invoice
FOR EACH ROW
BEGIN
    IF (NEW.courier_name='Pickup') THEN
        INSERT INTO InvoicePickup
        SET invoice_no=NEW.invoice_no,
            start_time=DATE(NEW.purchase_time),
            end_time=DATE(NEW.purchase_time)+7;
    END IF;
END $$
DELIMITER ;
```

9.1.5. Deceased artifact recording from Artifact

This trigger is triggered when an artifact is deleted from the Artifact table to keep a record for future reference.

```
DELIMITER $$
CREATE
DEFINER='root'@'localhost'
TRIGGER TGG_archiveArtifact
BEFORE DELETE ON Artifact
FOR EACH ROW
BEGIN
    DECLARE original_type ENUM('Thangka', 'Khukuri', 'SingingBowl',
'Other');
    DECLARE temp_id INT;
    IF (EXISTS (SELECT * FROM Thangka WHERE
artifact_id=OLD.artifact_id)) THEN
        SET original_type='Thangka';
    ELSEIF (EXISTS (SELECT * FROM Khukuri WHERE
artifact_id=OLD.artifact_id)) THEN
        SET original_type='Khukuri';
    ELSEIF (EXISTS (SELECT * FROM SingingBowl WHERE
artifact_id=OLD.artifact_id)) THEN
        SET original_type='SingingBowl';
    ELSE
        SET original_type='Other';
    END IF;
    SET temp_id=(SELECT MAX(record_id) FROM DeceasedArtifact)+1;
    IF (temp_id IS NULL) THEN
        SET temp_id=1;
    END IF;
    INSERT INTO DeceasedArtifact
    SET record_id=temp_id,
        artifact_id=OLD.artifact_id,
        artifact_type=original_type,
        title=OLD.title,
        retail_price=OLD.retail_price,
        description=OLD.description,
        delete_time=NOW();
END $$
DELIMITER ;
```

9.2. Background Features in the Website

There are certain features, which have been implemented on the website, which are not as apparent. Some of them are as follows:

- When artifacts are confirmed for purchase and payments are made, the artifacts from the cart are deleted in the database as they are only used temporarily and moved to the invoice and invoice item tables, with the invoice item table storing the item listings and the invoice table storing the other required information.
- On the Details page, every time a user views an artifact, an entry will be created in the browsehistory table, where the customer_id and the artifact_id will be stored. This is done to keep track of the artifacts, which have been browsed the most times. This allows the client to understand their most interesting artifacts, which have been viewed by the clients.

10. User's Manual

10.1. Tools and Installation

To create the database and develop the website the following tools were used:

10.1.1. Database and Server Tools

MySQL Workbench GUI was used to execute most of the queries during development. Similarly, FileZilla was used to manage files on the remote server. Since the website was developed using PHP, the XAMPP server was used which supplies us with the Apache server and its own web interface for database management.

10.1.2. Development Tools

Development of the website was done using Notepad++ and Atom. The development of the website was done using PHP using the XAMPP server. Along with PHP, HTML, CSS, and JavaScript and its libraries were used.

10.1.3. Installation

To run the application locally, the user will first have to install the XAMPP server for the website [https://www.apachefriends.org/download.html]. After installation, they will need to run the XAMPP control panel and start MySQL and Apache servers. The application will need to be placed in the user's C:/xampp/htdocs folder. On a web browser, the following URLs will open the respective portion of the application.

Customer website: localhost/aureate-artifacts

Client website: localhost/aureate-artifacts/admin/pages/index.php

The URL for the online version of the customer site is:

http://artifacts.centralus.cloudapp.azure.com/

The URL for the online version of the client side is:

http://artifacts.centralus.cloudapp.azure.com/admin/index.php

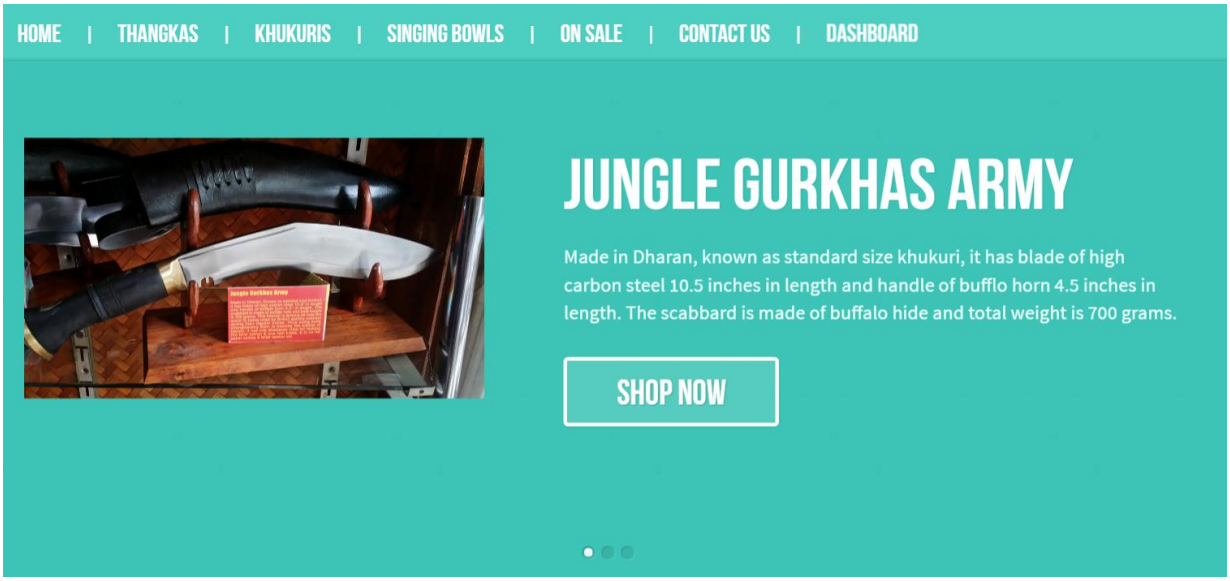
10.2. Customer Website

The customer website is as the name states, directed towards the customer, where the customer will be able to create their user profiles, view the artifacts and its details, add them to the wish-list, and make any purchases. Along with this, the customer will be able to view their purchase history and make changes to their user profile.

On the main navigation bar, the user will be able to navigate towards the various main pages of the website, which are as follows:

10.2.1. Homepage

The homepage presents most of the artifacts with the respective images and corresponding price. By clicking on any of these artifacts, the user will be directed to a detailed page where they will be able to see a detailed description of the product.



10.2.2. Detail Page

In this page, the user, along with being able to view further details about the product, will be able to post their reviews and add the product to their checkout cart or their wish list.

JUNGLE GURKHAS ARMY

Made in Dharan, known as standard size khukuri, it has blade of high carbon steel 10.5 inches in length and handle of bufflo horn 4.5 inches in length. The scabbard is made of buffalo hide and total weight is 700 grams. This khukuri is famous all over the world for being used by british gurkha soldiers during their regular duties. Thus using knife strengthening them to be come the soldier of history. It has one sharpener (used for making fire form stone) and one real blade. It is used for paper cutting and letter opener etc.

NRs.3500

Quantity 1

Special Request

CUSTOMER REVIEWS

Enter A Comment

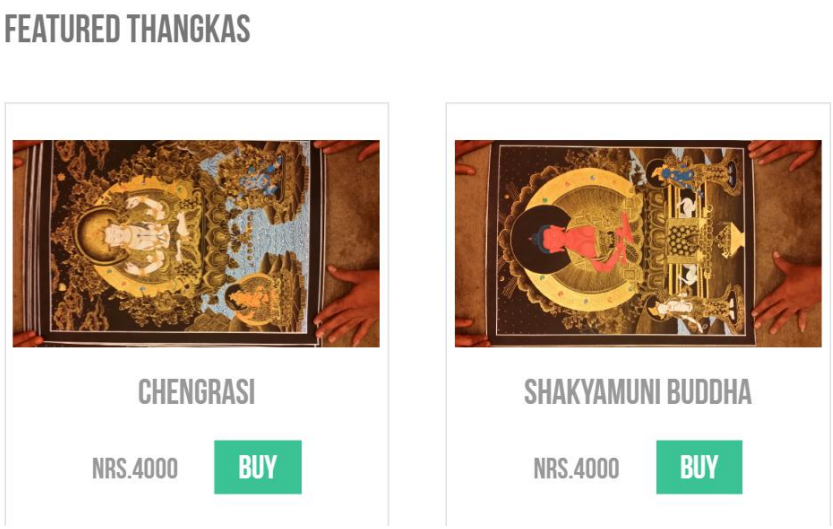
POST COMMENT

Ray Smith (Posted On: 2016-04-01 09:03:04)

Well Engraved

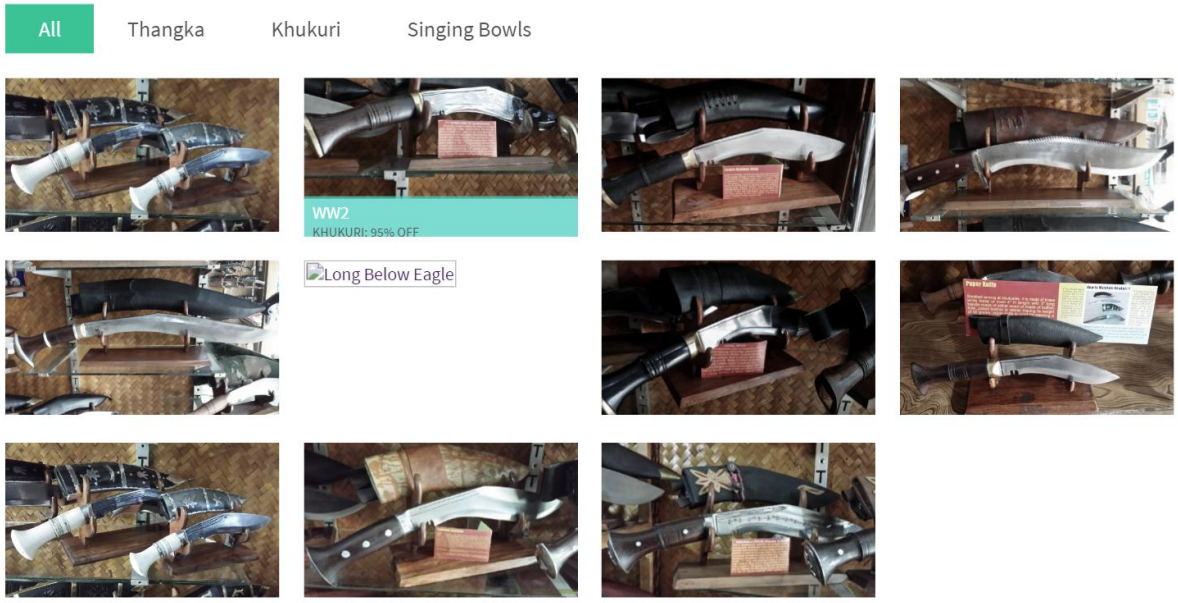
10.2.3. Thangka / Khukuri / Singing Bowl

These pages show their respective artifacts and their images and prices, through which the user can again go to the detail page as mentioned above.



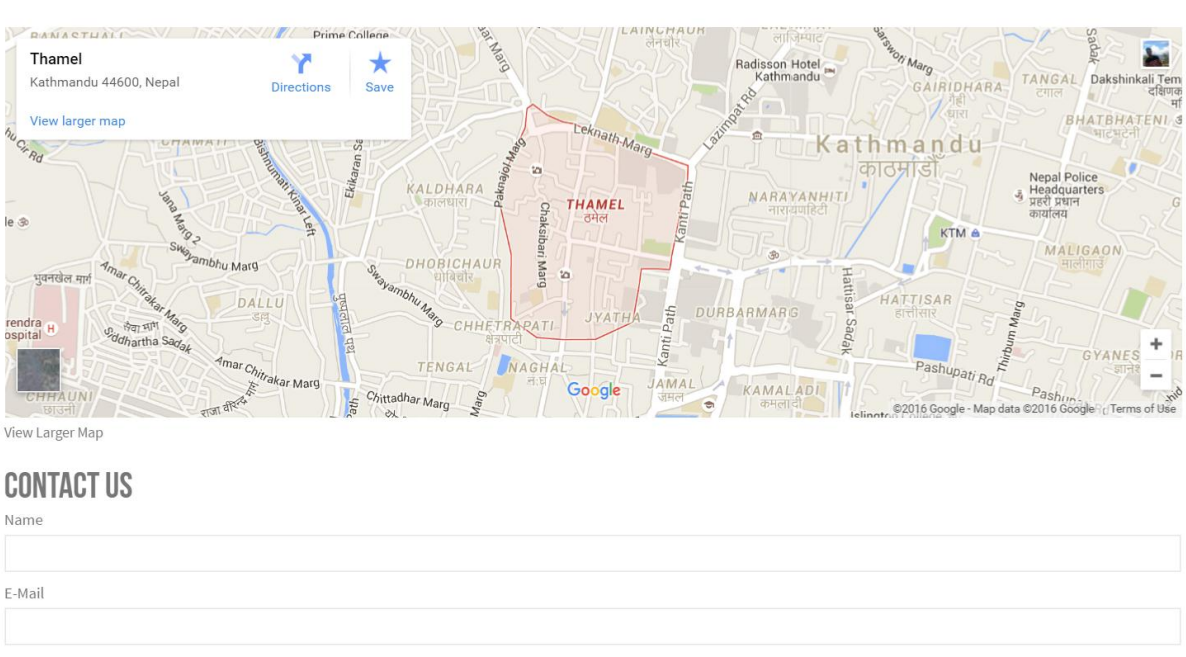
10.2.4. On Sale

These pages present the artifacts, which have promotional offers applied on them. The user will be able to view only the artifacts that are currently going through a promotional offer and have a cheaper price placed on them than the usually do.



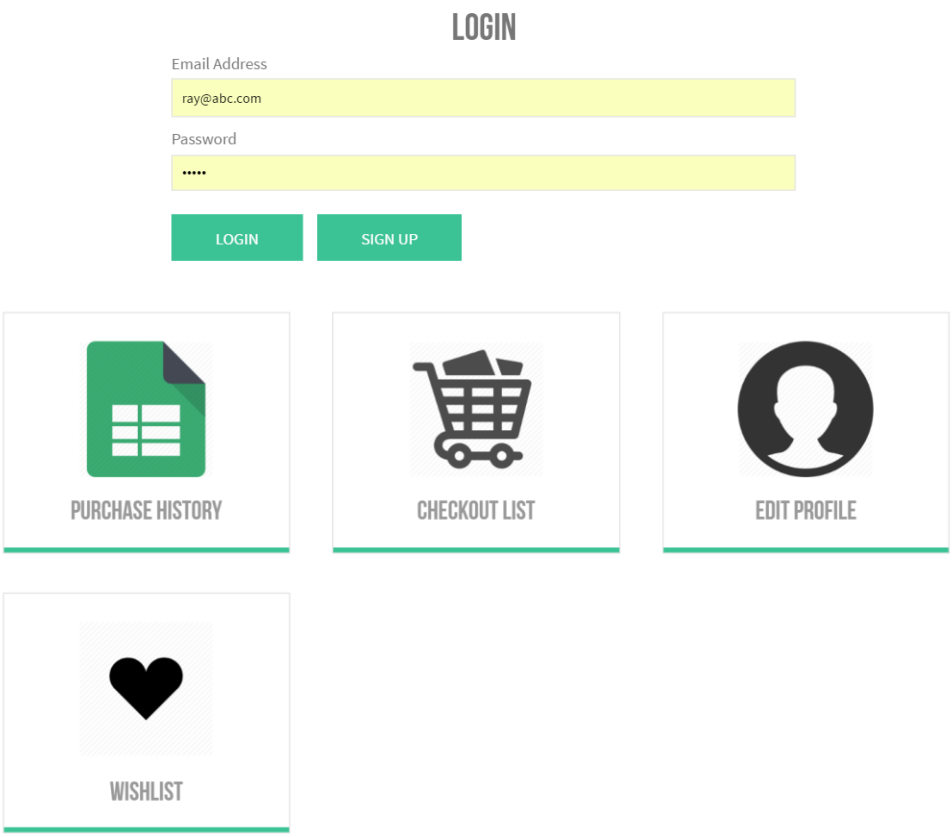
10.2.5. Contact Us

This page is mainly to receive any queries from any customer, who wishes to contact the company with their message. This page also has a Google map with an approximation of the main store.



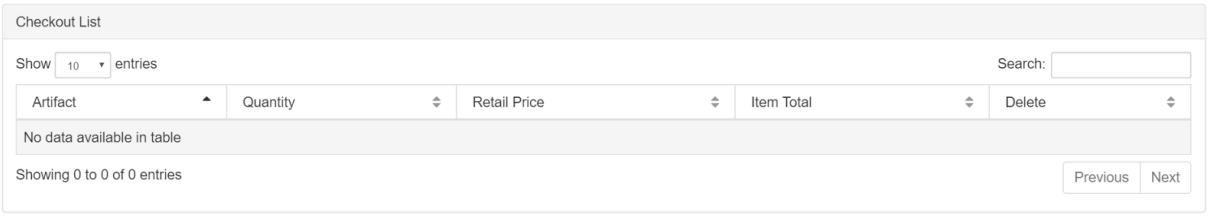
10.2.6. My Account / Dashboard

Depending on whether the user has logged in, the tab on the menu bar will present either a ‘My Account’ or ‘Dashboard’ on it. The My Account page directs the customer to a login page, whereas the ‘Dashboard’ will present the user with a sub-menu with the options to view the following pages:



10.2.7. Checkout

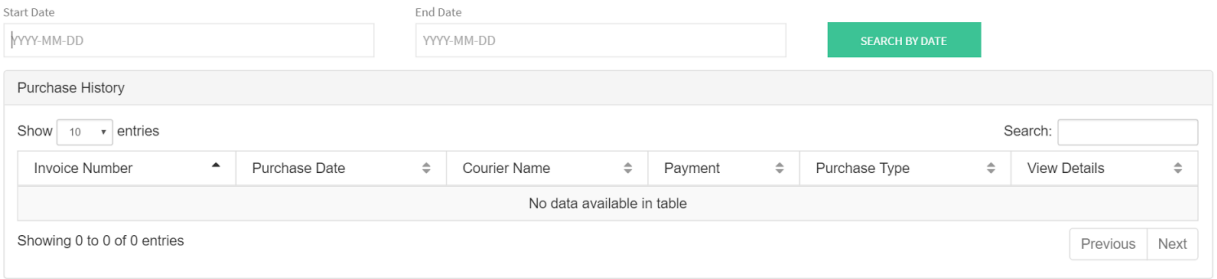
Any product that the user has added to their checkout cart will be presented in this page in a tabular format along with its details.



Total Price: 0

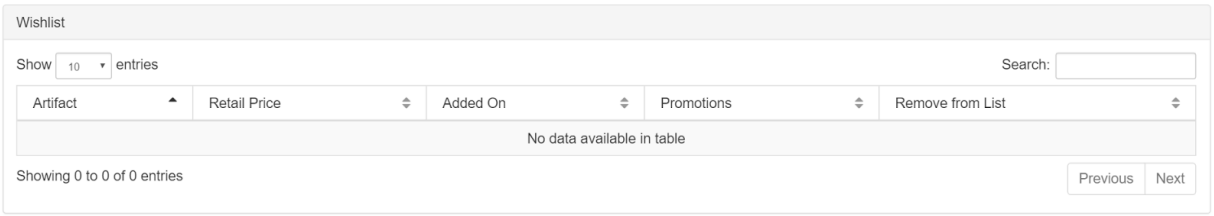
10.2.8. Purchase History

The user will be able to view their previous purchase history in this page and view details about each purchase by moving to a deeper page for each invoice.



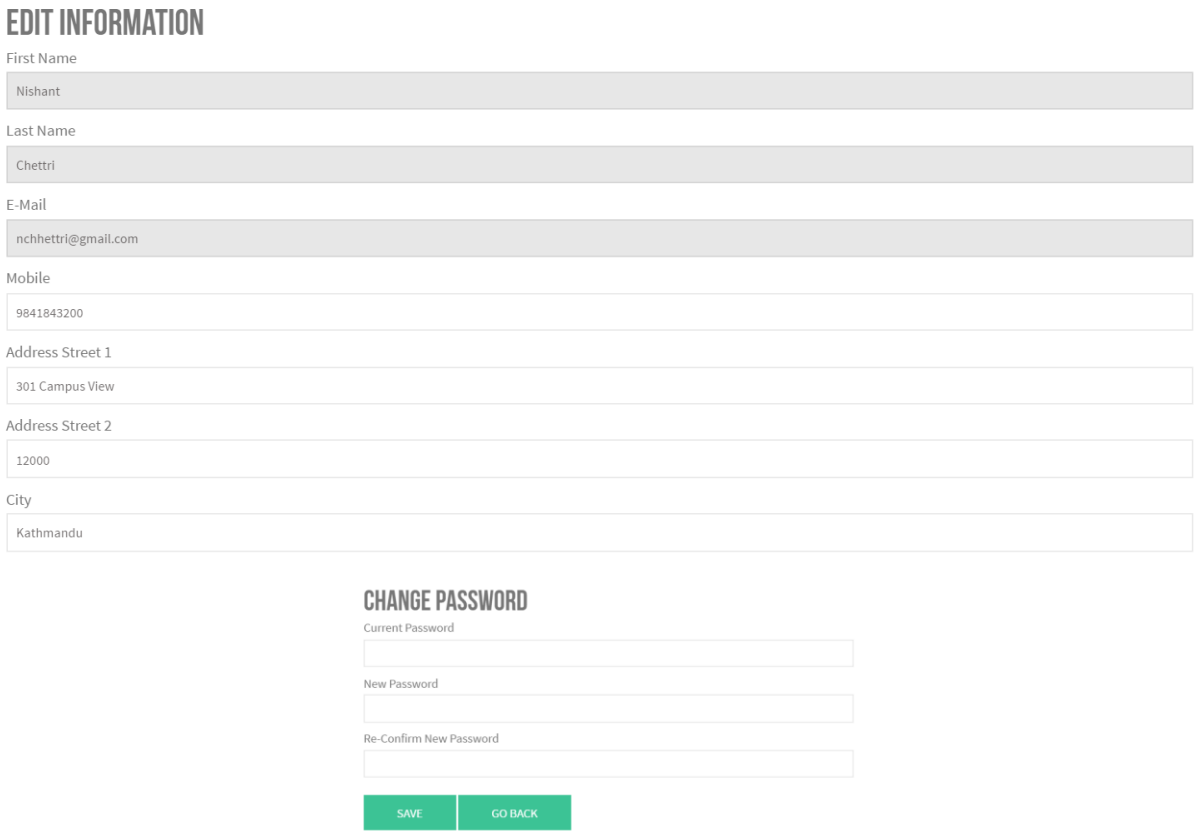
10.2.9. Wishlist

The wishlist page allows the user to keep a record of any artifact that they wish to buy in the future.



10.2.10.Edit Profile

From this page, the user will be able to edit the contact information and view any other information. They will also be able to change their password.



10.2.11.Sign Out

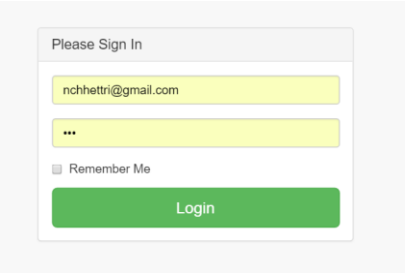
The sign out button at the bottom of the Dashboard tab in the menu bar ends the current user's session redirects the user to the login page.

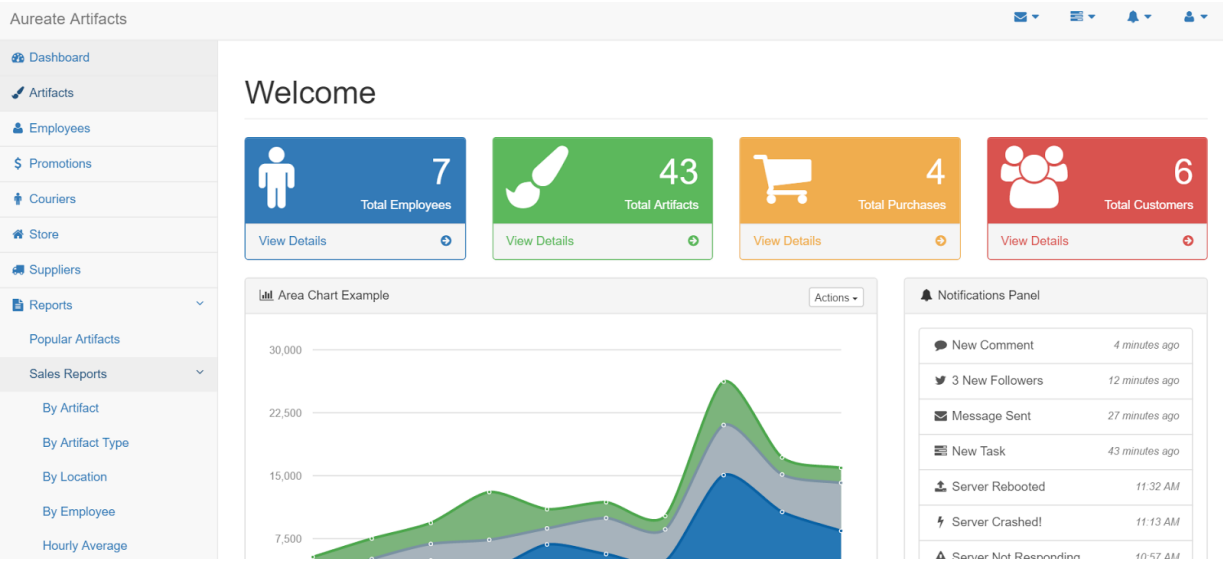
10.3. Client Side (Admin Panel)

The client side of the website is only for the employees of the company. Initially the login page will be shown to the user where only assigned users will be able to login.

After successful login, the user will be directed to the dashboard. Here, the employee will be able to view some statistics about the store. Firstly, they will be able to view the total number of employees in the company, the total number of artifacts available, the total number of purchases and the total number of registered customers.

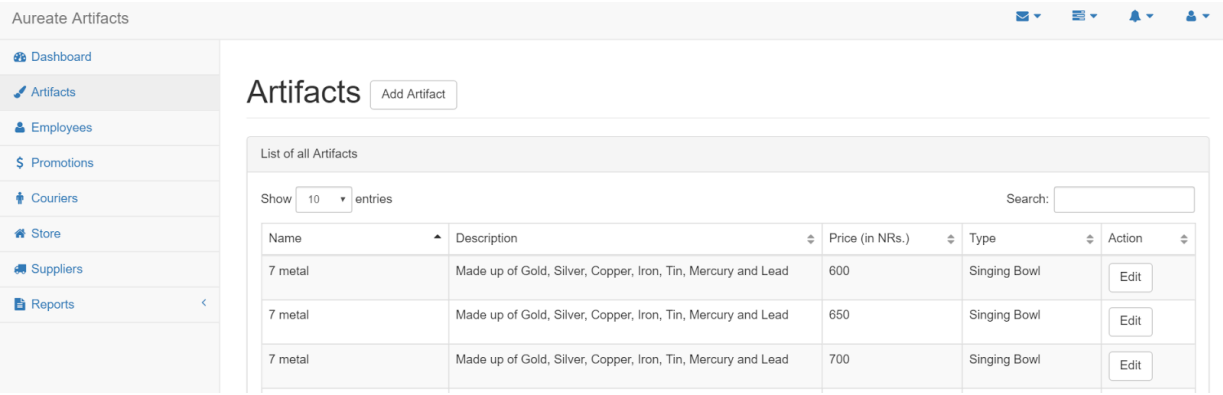
The menu bar presents the user with the following pages and depending on the permission applied to the employee, they will be able to view and make certain changes on the database.

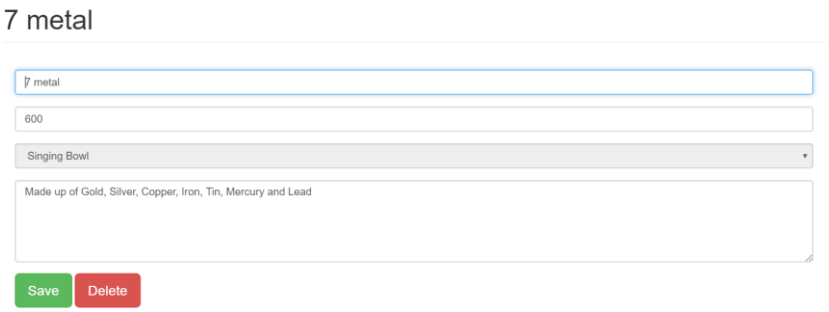




10.3.1. Artifacts

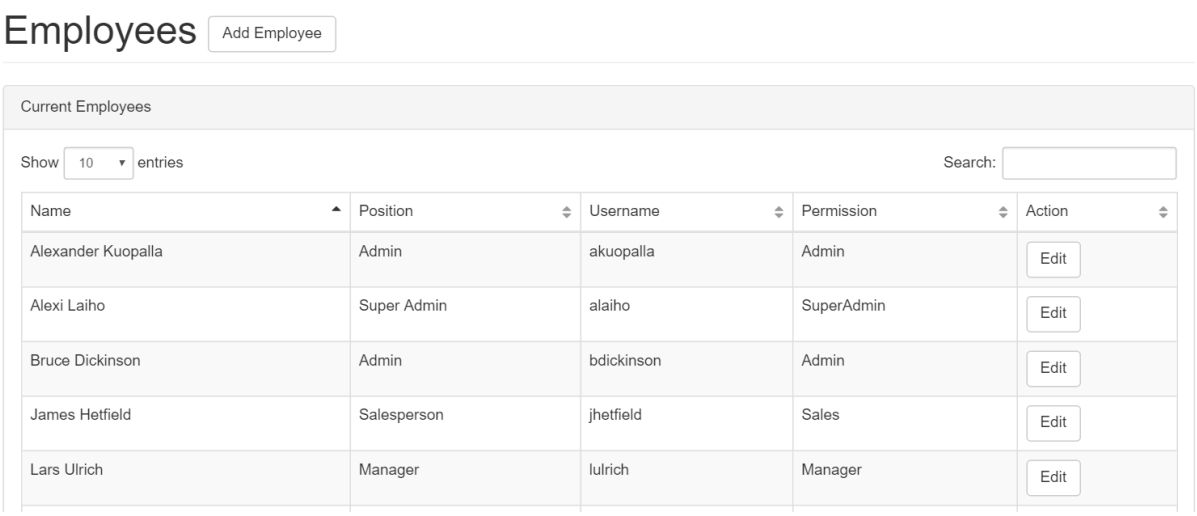
The employees will be able to view a list of artifacts and be able to edit these artifacts' name, price and descriptions.

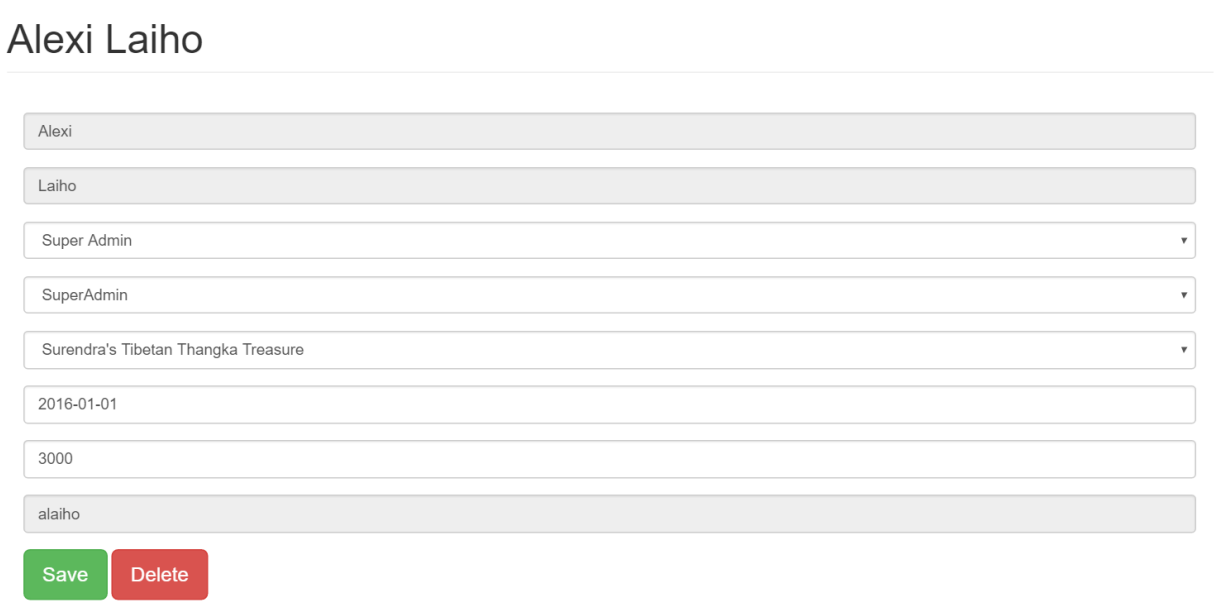




10.3.2. Employees

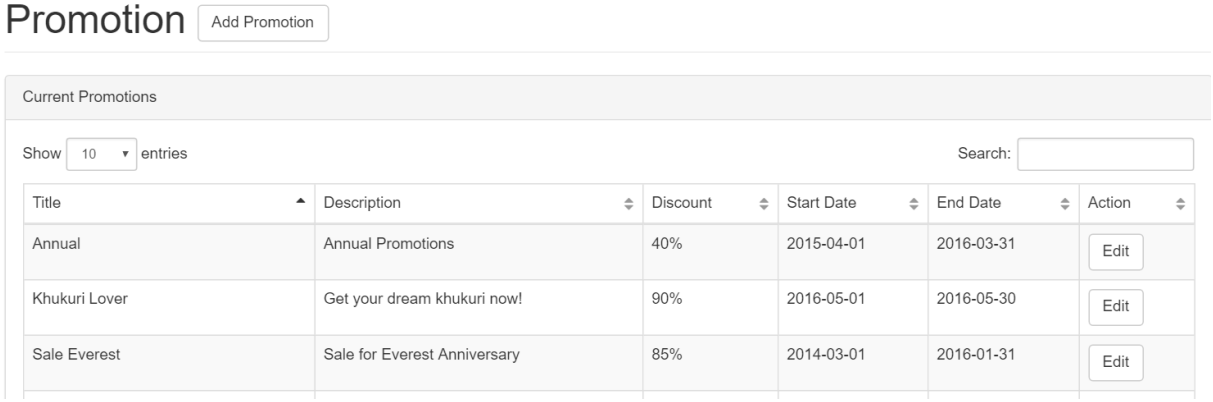
The list of current employees of the company will be viewed from this page. Similar add, edit and delete employee options will be created from this interface. A record of which store the employee works in will also be kept.

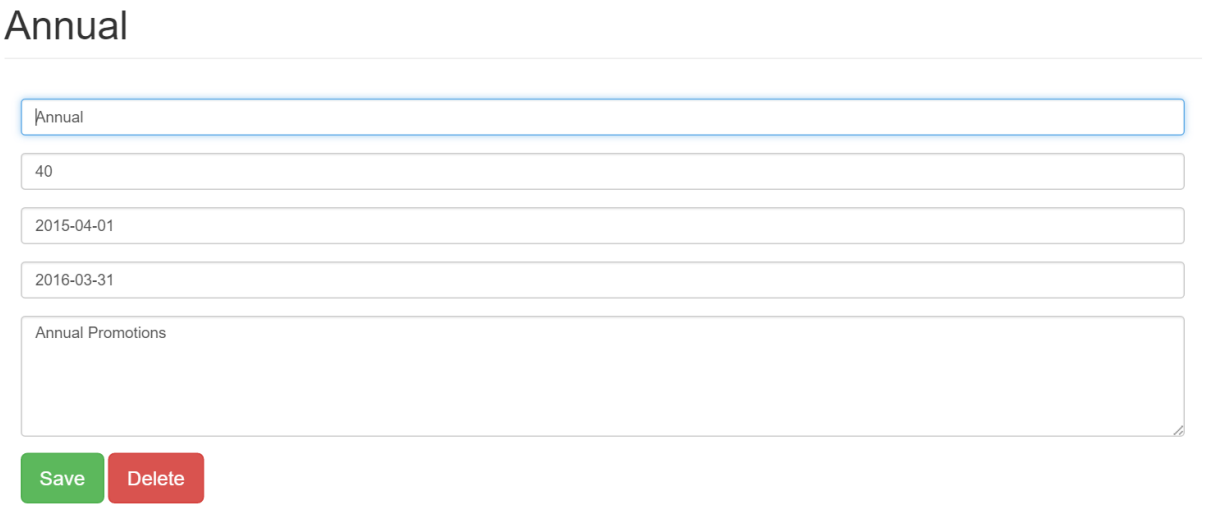




10.3.3. Promotions

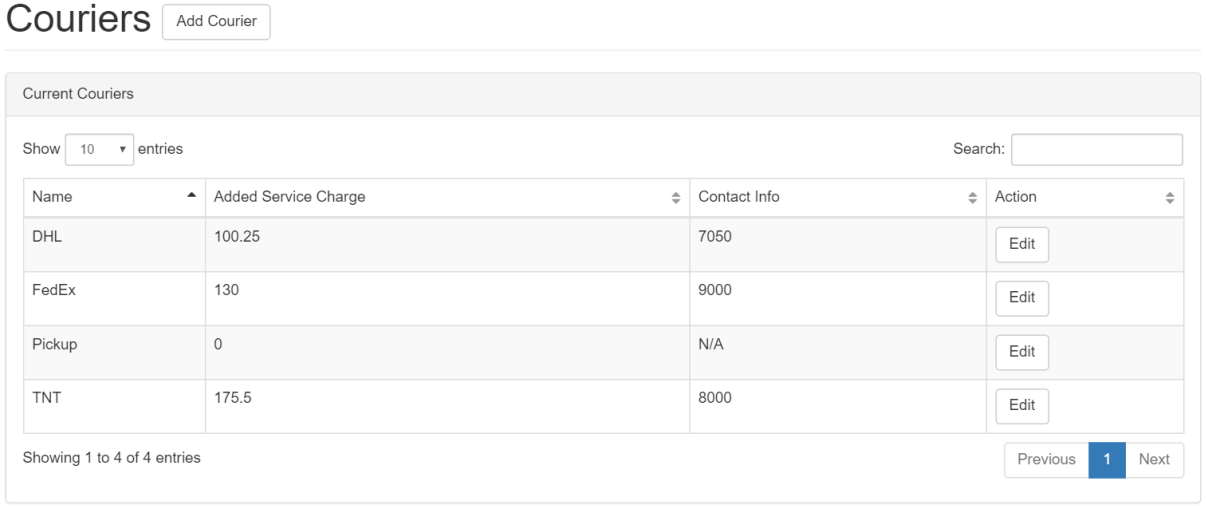
The list of promotions can be viewed from this interface. The user will be able to create new promotions and set the discount amount and the duration of the promotional offer.

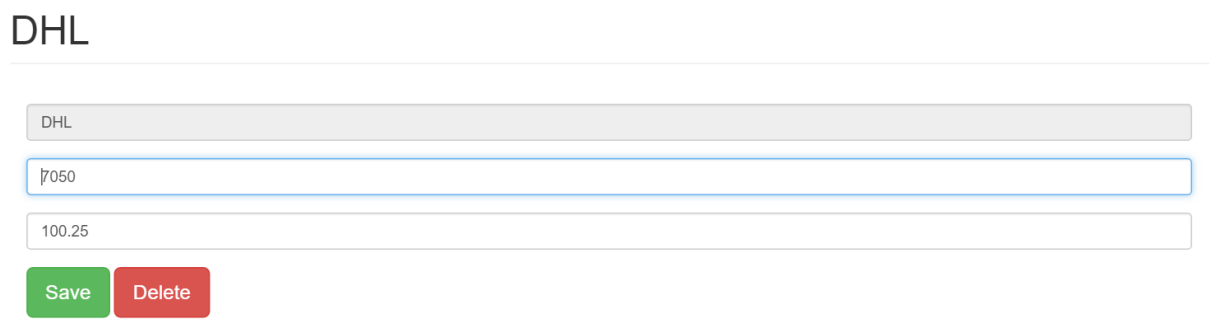




10.3.4. Couriers

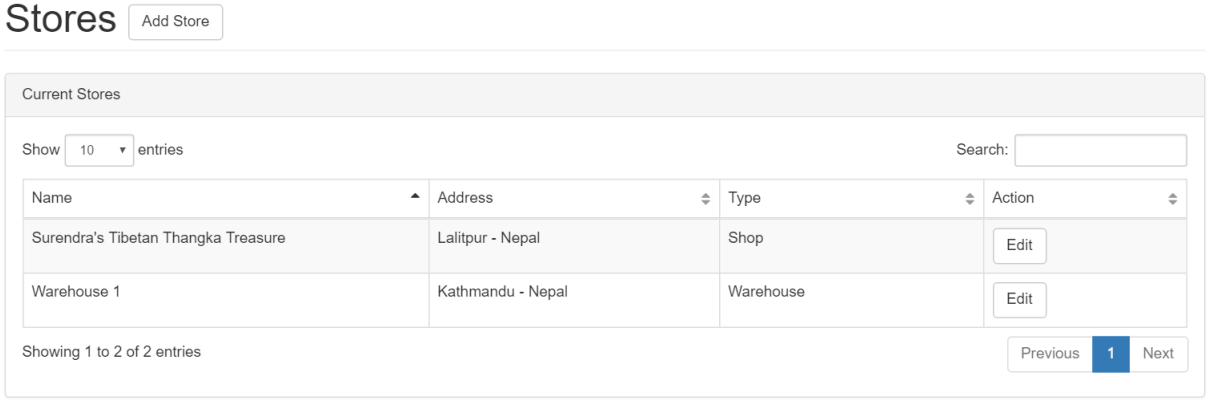
Any couriers, which have deals with the company will be kept here. These couriers will be presented in the customer side.

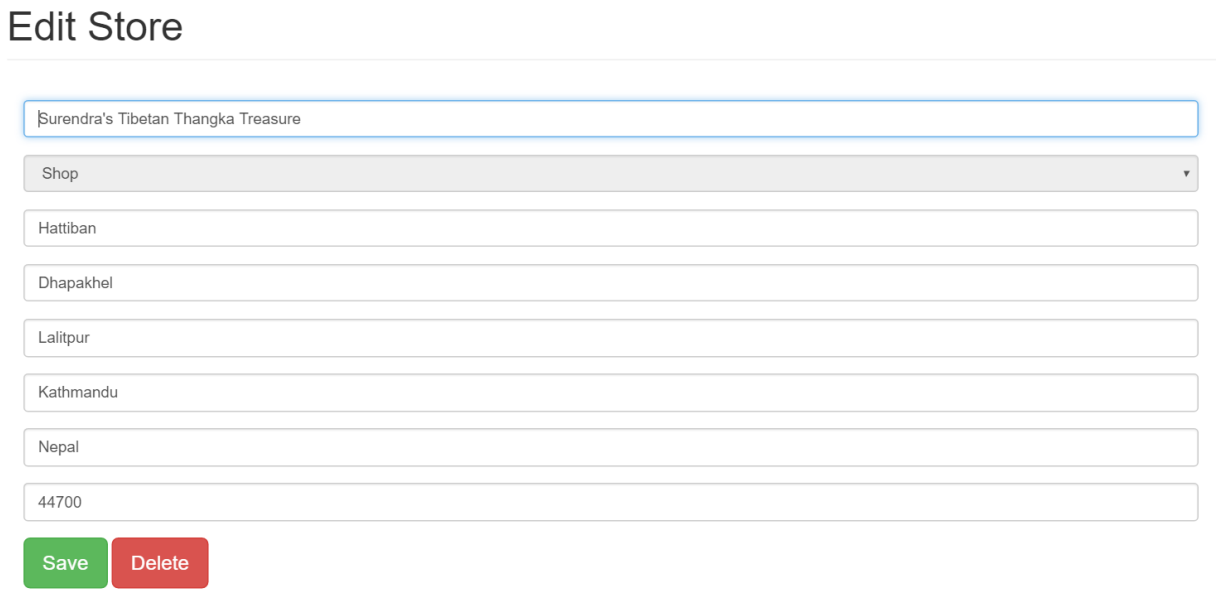




10.3.5. Store

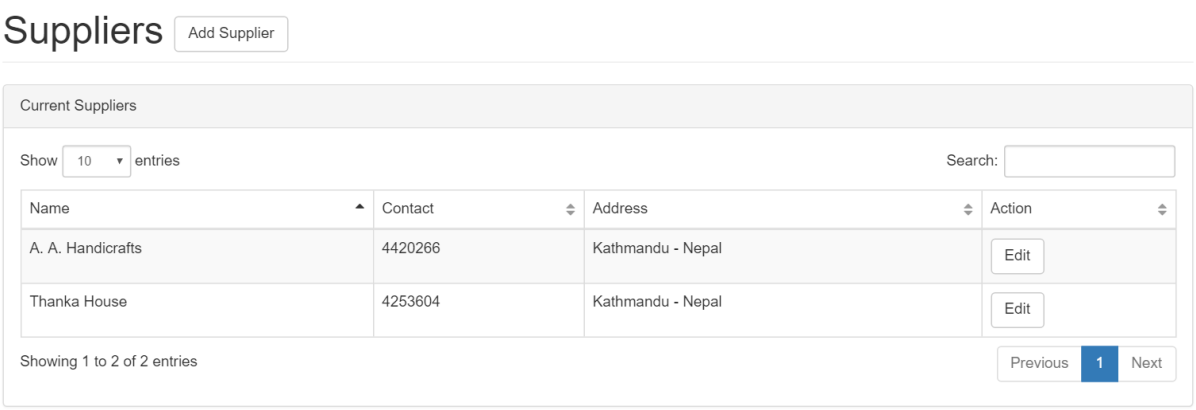
The list of shops and warehouses belonging to the company along with any necessary information pertaining to these stores can be viewed, edited and deleted from here.

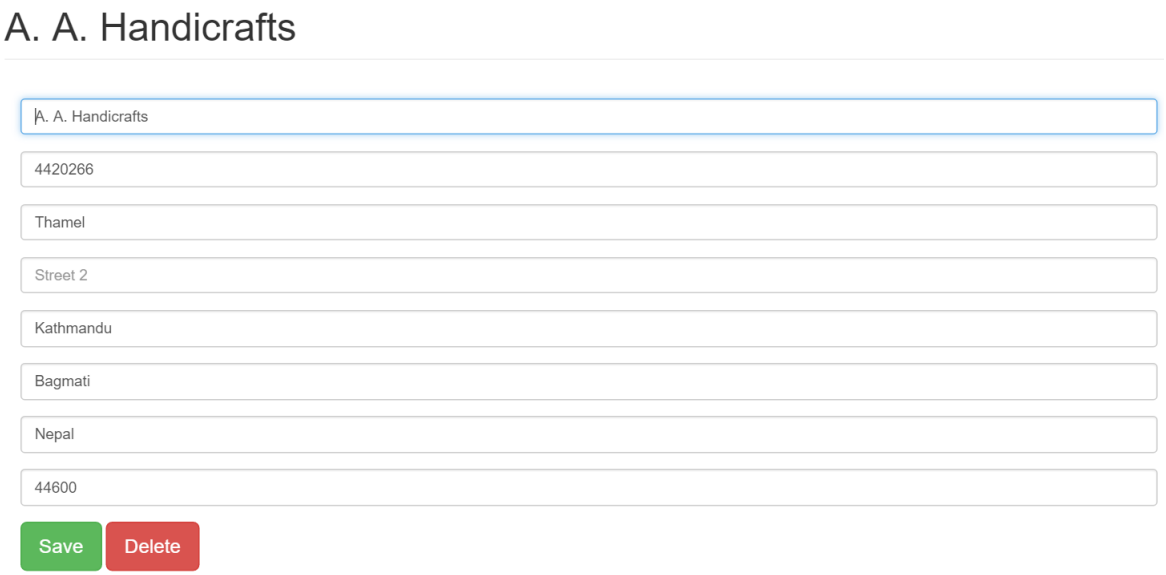




10.3.6. Suppliers

Any suppliers with connection with the company will can be added, viewed, edited and removed from the database using this interface.



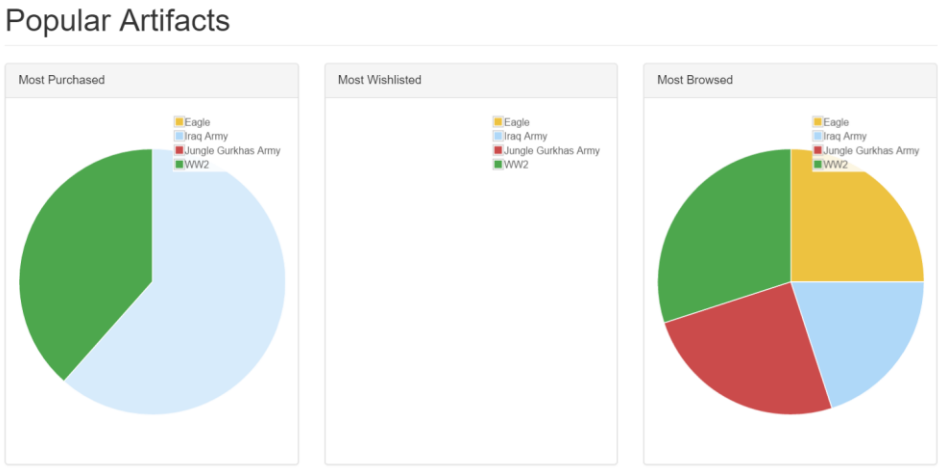


10.4. Reports

The main function of the website is to allow the store to be able to view reports about their sales and profits. The following pages presents some of the main reports that have been identified in order to improve the performance of the company.

10.4.1. Popular Artifacts

This report shows the most popular artifacts sold by the company depending on the most purchased, most wishlisted and most browsed. A pie chart also presents these information.

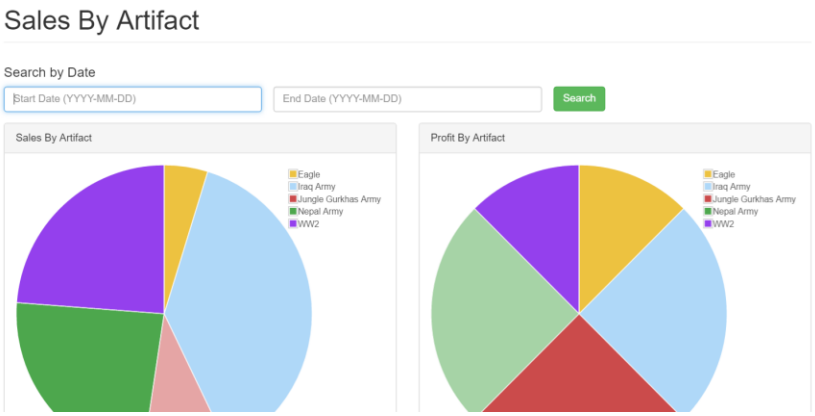


10.4.2. Sales Reports

The sales report presents the total number of artifacts sold and where possible, the profits obtained from selling these artifacts. The users will also be able to filter dates for the reports.

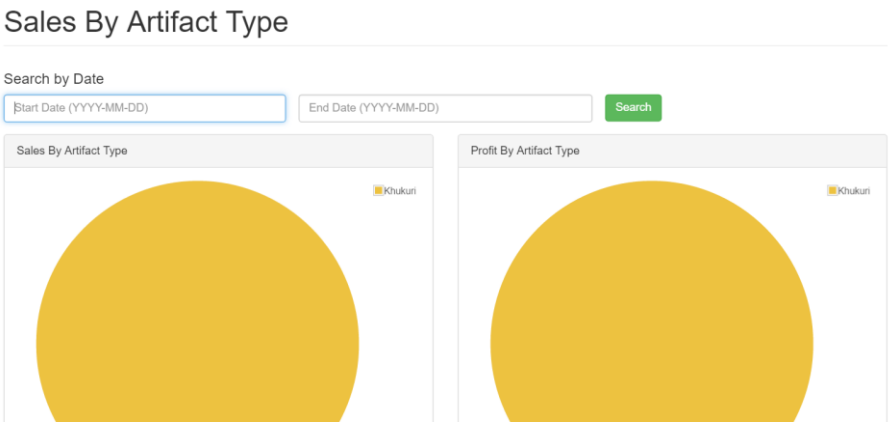
- By Artifact

Here, the company will be able to view how many artifacts were sold and the profits made from each of them.



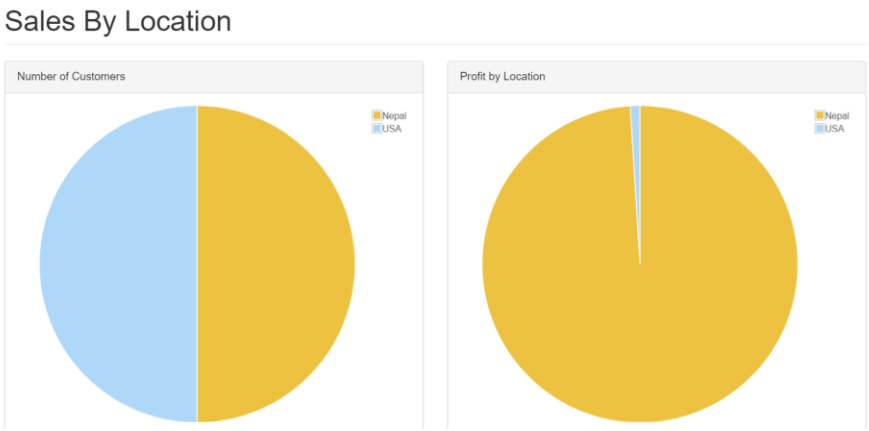
- By Artifact Type

Similarly, the company will be able to view the same reports as previous but the reports will be presented depending on the artifact type, i.e. Thangka, Khukuri and Singing Bowls.



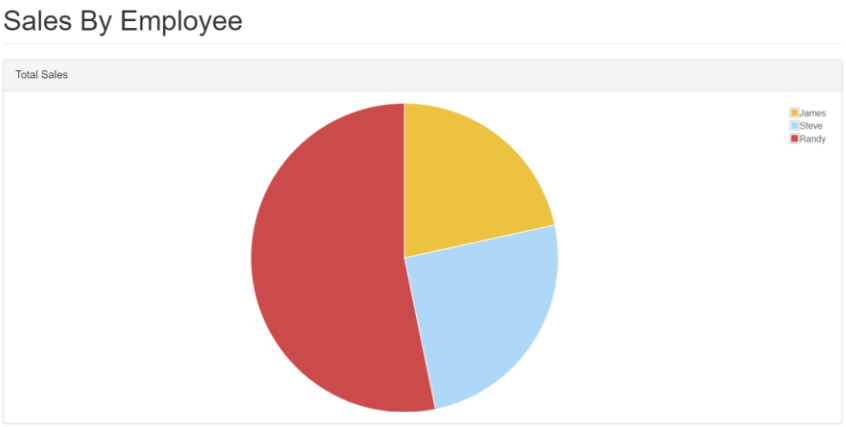
- By Location

Here, the company will be able to view the total number of customers, who have bought artifacts depending on the country of purchase.



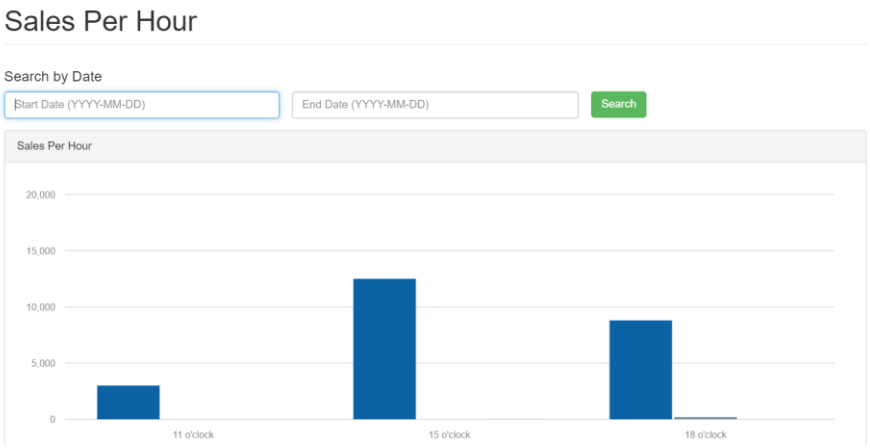
- By Employee

The employee report shows the total number of sales carried out by an employee so far.



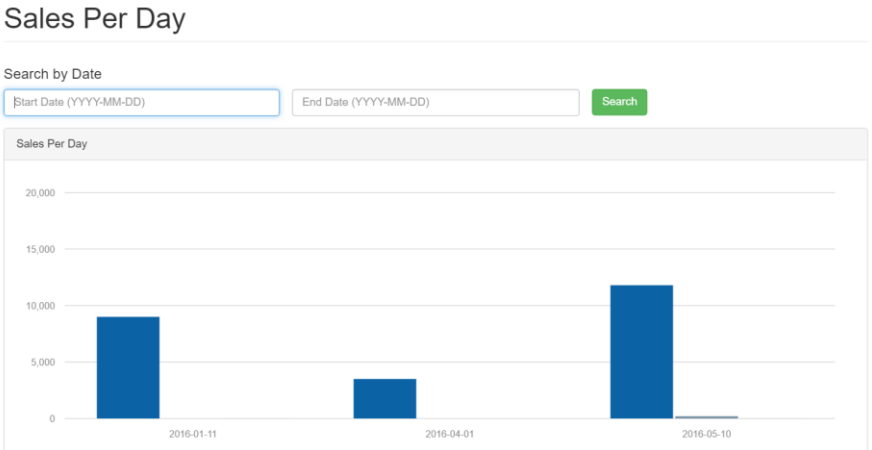
- Hourly Average

This report shows at what time most of the purchases are being carried out.



- Daily Reports

The daily report presents the daily sales and profits for the required dates.

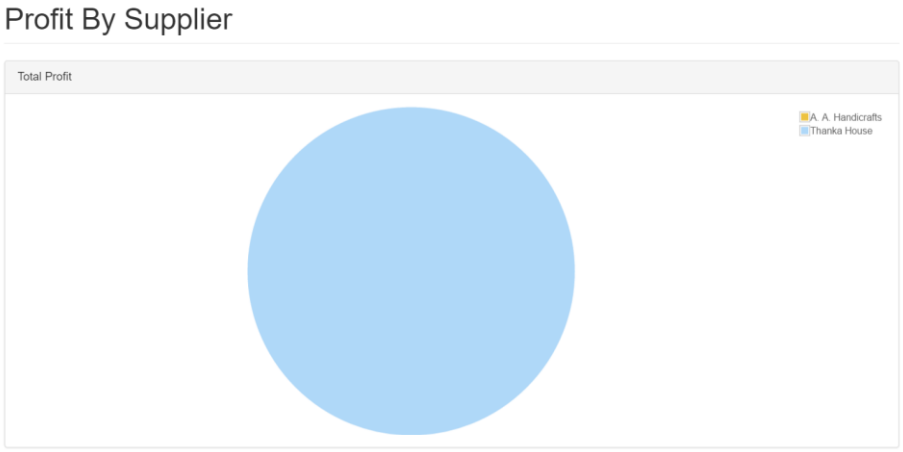


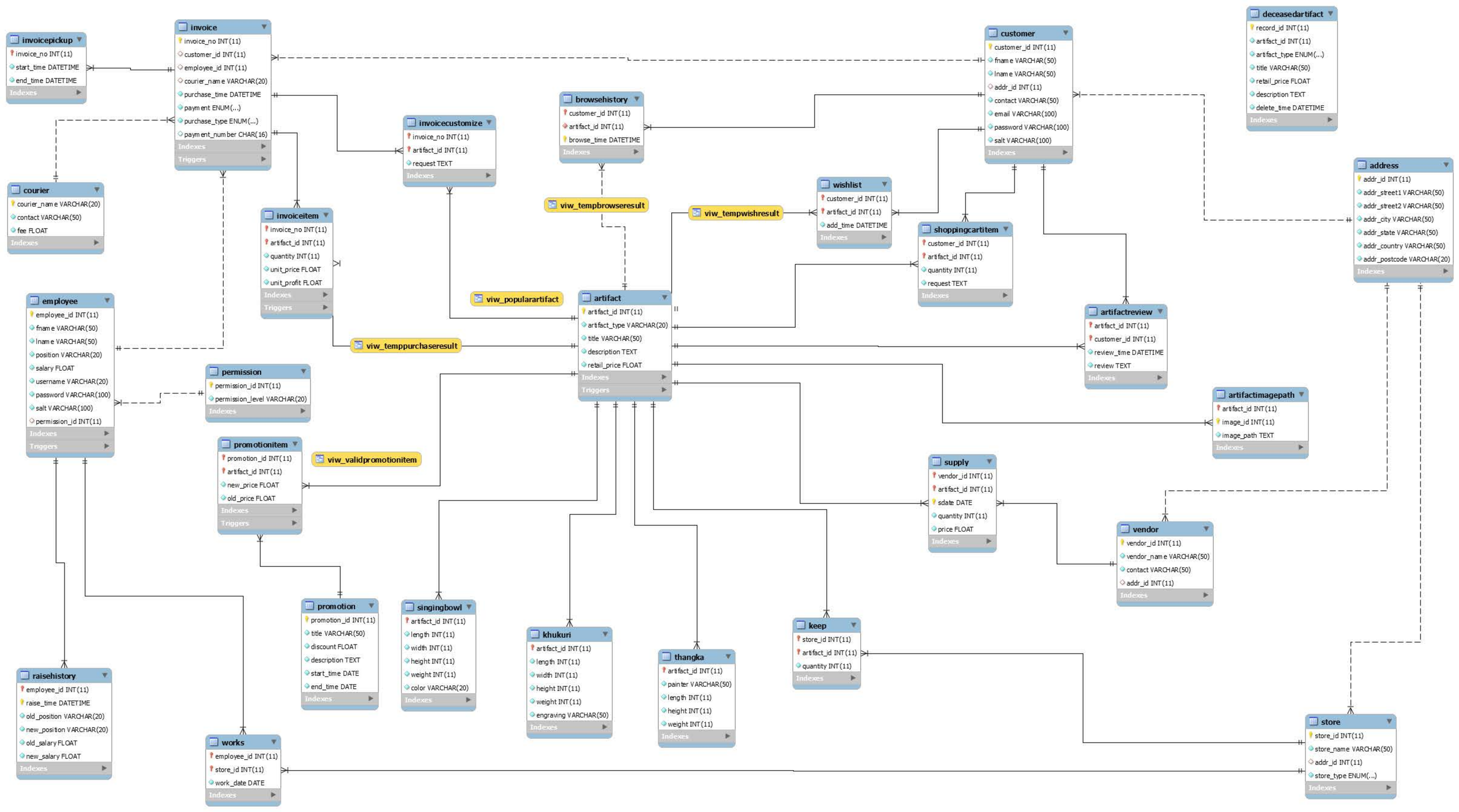
10.4.3. Profits

The total profits obtained from the supplier is presented in the following report.

- By Supplier

The profit report presents the profits obtained from each supplier.




invoicepickup
invoice_no INT(11)
start_time DATETIME
end_time DATETIME
Indexes
invoice
invoice_no INT(11)
customer_id INT(11)
employee_id INT(11)
courier_name VARCHAR(20)
purchase_time DATETIME
payment ENUM(...)
purchase_type ENUM(...)
payment_number CHAR(16)
Indexes
Triggers
customer
customer_id INT(11)
fname VARCHAR(50)
lname VARCHAR(50)
addr_id INT(11)
contact VARCHAR(50)
email VARCHAR(100)
password VARCHAR(100)
salt VARCHAR(100)
Indexes
deceasedartifact
record_id INT(11)
artifact_id INT(11)
artifact_type ENUM(...)
title VARCHAR(50)
retail_price FLOAT
description TEXT
delete_time DATETIME
Indexes
browsehistory
customer_id INT(11)
artifact_id INT(11)
browse_time DATETIME
Indexes
invoicecustomize
invoice_no INT(11)
artifact_id INT(11)
request TEXT
Indexes
address
addr_id INT(11)
addr_street1 VARCHAR(50)
addr_street2 VARCHAR(50)
addr_city VARCHAR(50)
addr_state VARCHAR(50)
addr_country VARCHAR(50)
addr_postcode VARCHAR(20)
Indexes
courier
courier_name VARCHAR(20)
contact VARCHAR(50)
fee FLOAT
Indexes
viw_tempbrowseresult
wishlist
customer_id INT(11)
artifact_id INT(11)
add_time DATETIME
Indexes
viw_tempwishresult
invoiceitem
invoice_no INT(11)
artifact_id INT(11)
quantity INT(11)
unit_price FLOAT
unit_profit FLOAT
Indexes
Triggers
shoppingcartitem
customer_id INT(11)
artifact_id INT(11)
quantity INT(11)
request TEXT
Indexes
viw_popularartifact
artifact
artifact_id INT(11)
artifact_type VARCHAR(20)
title VARCHAR(50)
description TEXT
retail_price FLOAT
Indexes
Triggers
artifactreview
artifact_id INT(11)
customer_id INT(11)
review_time DATETIME
review TEXT
Indexes
employee
employee_id INT(11)
fname VARCHAR(50)
lname VARCHAR(50)
position VARCHAR(20)
salary FLOAT
username VARCHAR(20)
password VARCHAR(100)
salt VARCHAR(100)
permission_id INT(11)
Indexes
Triggers
viw_temppurchaseresult
permission
permission_id INT(11)
permission_level VARCHAR(20)
Indexes
artifactimagepath
artifact_id INT(11)
image_id INT(11)
image_path TEXT
Indexes
promotionitem
promotion_id INT(11)
artifact_id INT(11)
new_price FLOAT
old_price FLOAT
Indexes
Triggers
viw_validpromotionitem
supply
vendor_id INT(11)
artifact_id INT(11)
sdate DATE
quantity INT(11)
price FLOAT
Indexes
vendor
vendor_id INT(11)
vendor_name VARCHAR(50)
contact VARCHAR(50)
addr_id INT(11)
Indexes
promotion
promotion_id INT(11)
title VARCHAR(50)
discount FLOAT
description TEXT
start_time DATE
end_time DATE
Indexes
singingbowl
artifact_id INT(11)
length INT(11)
width INT(11)
height INT(11)
weight INT(11)
color VARCHAR(20)
Indexes
khukuri
artifact_id INT(11)
length INT(11)
width INT(11)
height INT(11)
weight INT(11)
engraving VARCHAR(50)
Indexes
thangka
artifact_id INT(11)
painter VARCHAR(50)
length INT(11)
height INT(11)
weight INT(11)
Indexes
keep
store_id INT(11)
artifact_id INT(11)
quantity INT(11)
Indexes
raisehistory
employee_id INT(11)
raise_time DATETIME
old_position VARCHAR(20)
new_position VARCHAR(20)
old_salary FLOAT
new_salary FLOAT
Indexes
works
employee_id INT(11)
store_id INT(11)
work_date DATE
Indexes
store
store_id INT(11)
store_name VARCHAR(50)
addr_id INT(11)
store_type ENUM(...)
Indexes