Catering Service

A Database Management System

By Rutuja Kaushike (RNK170000) and Radhika Kulkarni (RXK180002)

As a semester project for Database Management System (CS6360.003)

Data Requirements

Services offered by Catering System:

Numerous services are offered by the department which can broadly be classified into two categories –

1. Services for the customer
    a) Choose items based on the type
    b) Create menu of all the selected items
    c) Apply discount coupons
    d) Enter the event details
    e) Checkout

2. Services for the Manager of Catering
    a) Manage staff based on the event details
    b) Add and remove staff members
    c) Enter salary for staff members

MENU

The items of the menu are defined in the database system. The user can create his own menu by choosing the items and their quantity. Thus, it is possible to have a customized menu for all the customers. Each food item has description in order to sort it based on the cuisine.

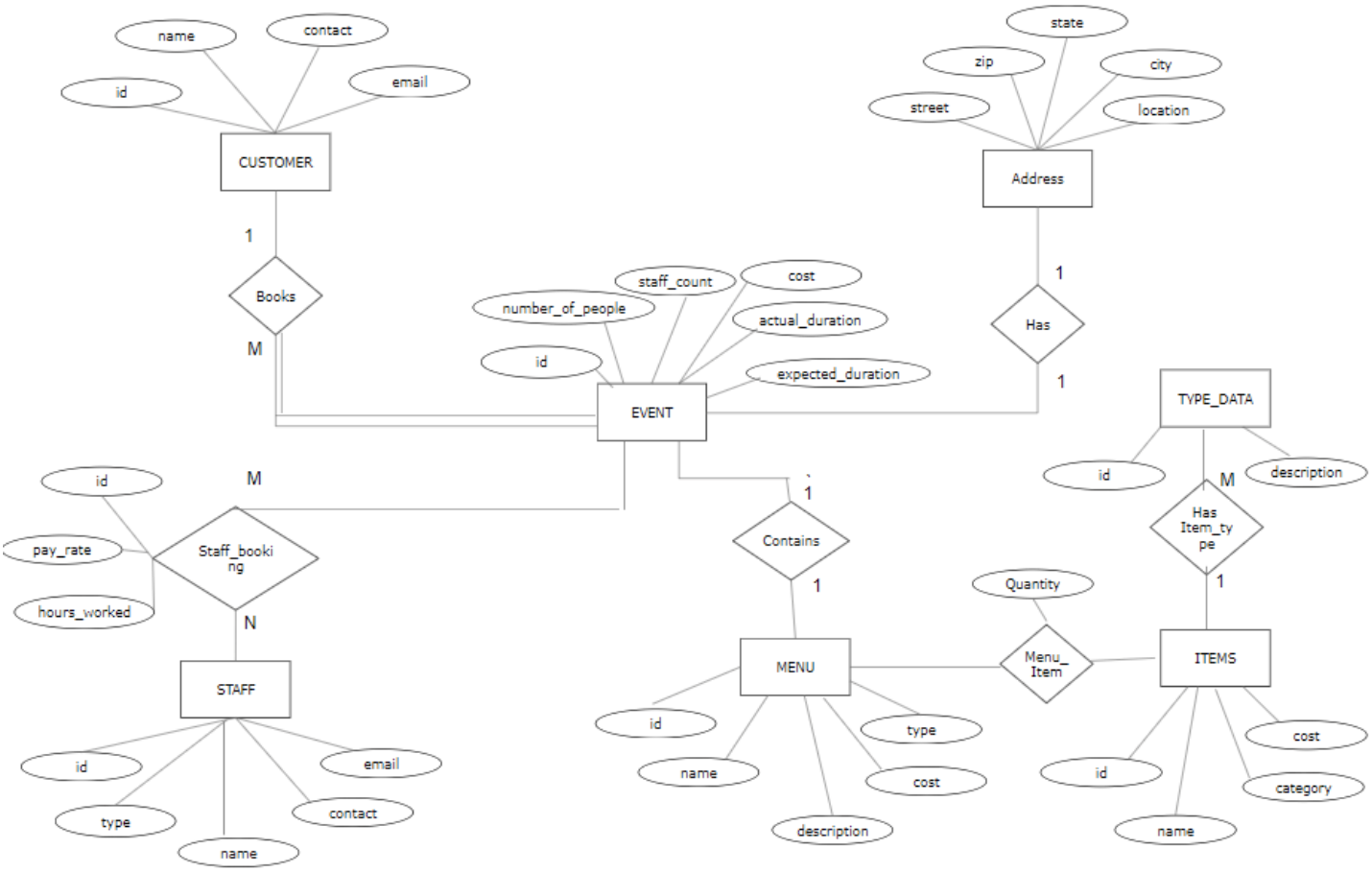
EVENT

A customer can create events having the respective details. He will have to specify the address of the event. He can also specify the number of people who will be attending the event.

STAFF

Catering Service manager has the responsibility to assign staff members to particular events. He can also enter wages of each staff member. It is expected that the manager assigns staff considering the number of people attending the event.

Modeling of Requirements as ER-Diagram:



Mapping of ERD in Relational Schema:

1. CUSTOMER

Id	Name	Contact	Email

- Primary Key: Id

2. STAFF

Id	Name	Contact	Email	Type

- Primary Key: Id

3. EVENT

Id	People_Count	Staff_Count	Exp_Duration	Actual_Duration	Cust_id	Address_id	Menu_id	Date	Event_cost

- Primary Key: Id
- Foreign Key: Foreign Key (Cust_id) references CUSTOMER (Id), Foreign Key (Address_id) references ADDRESS (Id), Foreign Key (Menu_id) references MENU (Id)

4. ADDRESS

Id	Location	City	State	Zip	Street

- Primary Key: Id

5. MENU

Id	Name	Cost	Description

- Primary Key: Id

6. ITEM

Id	Name	Type	Category	Cost

- Primary Key: Id

7. TYPE

Type_id	Description

- Primary Key: Type_id

8. ITEM_TYPE

Item_id	Type_id

- Primary Key: Item_id, Type_id
- Foreign Key: Foreign Key (Item_id) references ITEM (Id), Foreign Key (Type_id) references TYPE (Id)

9. MENU_ITEM

Menu_id	Item_id	Qty	Event_id

- Primary Key: Menu_Id, Item_id
- Foreign Key: Foreign Key (Menu_id) references MENU(Id), Foreign Key (Item_id) references ITEM (Id), Foreign Key (Event_id) references EVENT (Id)

10. STAFF_BOOKING

Staff_id	Event_id	Pay_rate	Hrs_watched

- Primary Key: Staff_id, Event_id
- Foreign Key: Foreign Key (Staff_id) references STAFF(Id), Foreign Key (Event_id) references EVENT(Id)

SQL Statements to create Relations in Database and add Constraints:

```
CREATE TABLE CUSTOMER(
ID INT NOT NULL,
NAME VARCHAR(100) NOT NULL,
CONTACT VARCHAR(10) NOT NULL,
EMAIL VARCHAR(50),
PRIMARY KEY(ID)
);

CREATE TABLE STAFF(
ID INT NOT NULL,
NAME VARCHAR(100) NOT NULL,
CONTACT VARCHAR(10) NOT NULL,
EMAIL VARCHAR(50),
TYPE VARCHAR(20) NOT NULL,
PRIMARY KEY(ID)
);

CREATE TABLE ADDRESS(
ID INT NOT NULL,
APARTMENT VARCHAR(100),
STREET VARCHAR(100),
CITY VARCHAR(100),
ZIP INT NOT NULL,
PRIMARY KEY(ID)
);

CREATE TABLE MENU(
ID INT NOT NULL,
NAME VARCHAR(100) NOT NULL,
COST NUMBER NOT NULL,
DESCRIPTION VARCHAR(100),
PRIMARY KEY(ID)
);

CREATE TABLE ITEM(
```

```
ID INT NOT NULL,
NAME VARCHAR(100) NOT NULL,
TYPE VARCHAR(20) NOT NULL,
CATEGORY VARCHAR(20) NOT NULL,
COST NUMBER NOT NULL,
PRIMARY KEY(ID)
);

CREATE TABLE MENU_ITEM(
MENU_ID NOT NULL,
ITEM_ID NOT NULL,
EVENT_ID NOT NULL,
QUANTITY INT DEFAULT 0,
FOREIGN KEY(MENU_ID) REFERENCES MENU(ID),
FOREIGN KEY(ITEM_ID) REFERENCES ITEM(ID),
FOREIGN KEY(EVENT_ID) REFERENCES EVENT(ID)
);

CREATE TABLE TYPE_DATA(
ID INT NOT NULL,
DESCRIPTION VARCHAR(100),
PRIMARY KEY(ID)
);

CREATE TABLE ITEM_TYPE(
ITEM_ID INT NOT NULL,
TYPE_ID INT NOT NULL,
FOREIGN KEY(ITEM_ID) REFERENCES ITEM(ID),
FOREIGN KEY(TYPE_ID) REFERENCES TYPE_DATA(ID)
);

CREATE TABLE EVENT(
ID INT NOT NULL,
PEOPLE_COUNT INT NOT NULL,
STAFF_COUNT INT NOT NULL,
CUSTOMER_ID INT NOT NULL,
ADDRESS_ID INT NOT NULL,
```

```
MENU_ID INT NOT NULL,
EXPECTED_DURATION INT NOT NULL,
ACTUAL_DURATION INT,
EVENT_DATE DATE NOT NULL,
PRIMARY KEY(ID),
FOREIGN KEY(CUSTOMER_ID) REFERENCES CUSTOMER(ID),
FOREIGN KEY(ADDRESS_ID) REFERENCES ADDRESS(ID),
FOREIGN KEY(MENU_ID) REFERENCES MENU(ID)
);
ALTER TABLE EVENT ADD (EVENT_COST NUMBER DEFAULT 0.0);

CREATE TABLE STAFF_BOOKING(
STAFF_ID INT NOT NULL,
EVENT_ID INT NOT NULL,
PAY_RATE NUMBER DEFAULT 10,
HOURS_WORKED NUMBER DEFAULT 0,
FOREIGN KEY(STAFF_ID) REFERENCES STAFF(ID),
FOREIGN KEY(EVENT_ID) REFERENCES EVENT(ID)
);

CREATE TABLE EVENT_COST AS
SELECT EVENT.ID AS EID, SUM(MENU.COST * MENU_ITEM.QUANTITY) AS ECOST
FROM EVENT,MENU,MENU_ITEM
WHERE MENU_ITEM.MENU_ID=MENU.ID AND EVENT.ID= MENU_ITEM.EVENT_ID
GROUP BY EVENT.ID;

-- DROP TABLE QUERIES IF REQUIRED
DROP TABLE CUSTOMER;
DROP TABLE STAFF;
DROP TABLE ADDRESS;
DROP TABLE MENU;
DROP TABLE ITEM;
DROP TABLE MENU_ITEM;
DROP TABLE TYPE_DATA;
DROP TABLE ITEM_TYPE;
DROP TABLE EVENT;
DROP TABLE STAFF_BOOKING;
```

Normalization of a Relational Schema:

The following functional dependencies exist in a relational schema -

1. CUSTOMER {Id -> Name, Contact, Email}
2. STAFF { Id -> Name, Contact, Email, Type}
3. EVENT {Id -> People_Count, Staff_Count, Exp_Duration, Actual_Duration, Cust_id, Address_id, Menu_id, Date, event_cost }
4. ADDRESS {Id -> Location, City, State, Zip, Street}
5. MENU {Id ->Name Cost, Description}
6. ITEM {Id -> Name, Type, Category, Cost}
7. TYPE {Type_id -> Description}
8. MENU_ITEM {Menu_id, Item_id, Event_id -> Qty}
9. STAFF_BOOKING {Staff_id, Event_id ->Pay_rate, Hrs_watched}

The above dependencies cause the database schema to be in 3NF

Triggers and Procedures

1. Triggers

a. Trigger to log the changes made in STAFF Table

```
CREATE TABLE STAFF_AUDIT(

ID INT NOT NULL,

NAME VARCHAR(100) NOT NULL,

CONTACT VARCHAR(10),

EMAIL VARCHAR(50),

TYPE VARCHAR(20),

NEW_NAME VARCHAR(100) NOT NULL,

NEW_CONTACT VARCHAR(10),

NEW_EMAIL VARCHAR(50),

NEW_TYPE VARCHAR(20),

OPERATION VARCHAR2(20) CHECK( operation IN('INSERT','UPDATE','DELETE')) NOT
NULL,

UPDATEDTIME TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);

DROP TABLE STAFF_AUDIT;


CREATE OR REPLACE TRIGGER STAFF_AUDIT

AFTER INSERT OR DELETE OR UPDATE ON STAFF

FOR EACH ROW

BEGIN

IF INSERTING THEN

INSERT INTO STAFF_AUDIT VALUES

(:OLD.ID,                                                                    NULL,
NULL,NULL,NULL,:NEW.NAME,:NEW.CONTACT,:NEW.EMAIL,:NEW.TYPE,'INSERT',SYSDATE);

ELSIF UPDATING THEN

INSERT INTO STAFF_AUDIT VALUES

(:OLD.ID,:OLD.NAME,:OLD.CONTACT,:OLD.EMAIL,:OLD.TYPE,:NEW.NAME,:NEW.CONTACT,:
NEW.EMAIL,:NEW.TYPE,'UPDATE',SYSDATE);

ELSIF DELETING THEN

INSERT INTO STAFF_AUDIT VALUES

(:OLD.ID,:OLD.NAME,:OLD.CONTACT,:OLD.EMAIL,:OLD.TYPE,NULL,NULL,NULL,NULL,'DEL
ETE',SYSDATE);

END IF;

END;
```

b. Trigger to log the changes made in CUSTOMER Table

```
CREATE TABLE CUSTOMER_AUDIT(
AUDIT_ID INT NOT NULL,
OLD_NAME VARCHAR(100) NOT NULL,
OLD_CONTACT VARCHAR(10),
OLD_EMAIL VARCHAR(50),
NEW_NAME VARCHAR(100) NOT NULL,
NEW_CONTACT VARCHAR(10),
NEW_EMAIL VARCHAR(50),
OPERATION VARCHAR2(10) CHECK( operation IN('INSERT','UPDATE','DELETE')) NOT NULL,
UPDATEDTIME TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE OR REPLACE TRIGGER CUSTOMER_AUDIT
AFTER INSERT OR DELETE OR UPDATE ON CUSTOMER
FOR EACH ROW
BEGIN
IF INSERTING THEN
INSERT INTO CUSTOMER_AUDIT VALUES
(:OLD.ID, NULL, NULL,NULL,:NEW.NAME,:NEW.CONTACT,:NEW.EMAIL,'INSERT',SYSDATE);
ELSIF UPDATING THEN
INSERT INTO CUSTOMER_AUDIT VALUES
(:OLD.ID,:OLD.NAME,:OLD.CONTACT,:OLD.EMAIL,:NEW.NAME,:NEW.CONTACT,:NEW.EMAIL,
'UPDATE',SYSDATE);
ELSIF DELETING THEN
INSERT INTO CUSTOMER_AUDIT VALUES
(:OLD.ID,:OLD.NAME,:OLD.CONTACT,:OLD.EMAIL,NULL,NULL,NULL,'DELETE',SYSDATE);
END IF;
END;
```

c. Trigger to log the changes made in EVENT Table

```
-- AUDIT TABLE FOR EVENT TABLE
CREATE TABLE EVENT_AUDIT(
ID INT NOT NULL,
OLD_PEOPLE_COUNT INT NOT NULL,
OLD_STAFF_COUNT INT NOT NULL,
```

```
OLD_CUSTOMER_ID INT NOT NULL,
OLD_ADDRESS_ID INT NOT NULL,
OLD_MENU_ID INT NOT NULL,
OLD_EXPECTED_DURATION INT,
OLD_ACTUAL_DURATION INT,
OLD_EVENT_DATE DATE,
OLD_EVENT_COST NUMBER,
NEW_PEOPLE_COUNT INT NOT NULL,
NEW_STAFF_COUNT INT NOT NULL,
NEW_CUSTOMER_ID INT NOT NULL,
NEW_ADDRESS_ID INT NOT NULL,
NEW_MENU_ID INT NOT NULL,
NEW_EXPECTED_DURATION INT,
NEW_ACTUAL_DURATION INT,
NEW_EVENT_DATE DATE,
NEW_EVENT_COST NUMBER,
OPERATION VARCHAR2(10) CHECK( operation IN('INSERT','UPDATE','DELETE')) NOT
NULL,
UPDATEDTIME TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);


CREATE OR REPLACE TRIGGER EVENT_AUDIT
AFTER INSERT OR DELETE OR UPDATE ON EVENT
FOR EACH ROW
BEGIN
IF INSERTING THEN
INSERT INTO EVENT_AUDIT VALUES
(:OLD.ID, NULL, NULL,NULL,NULL, NULL,NULL,NULL, NULL,NULL,
:NEW.PEOPLE_COUNT,:NEW.STAFF_COUNT,:NEW.CUSTOMER_ID,:NEW.ADDRESS_ID,:NEW.MENU
_ID,:NEW.EXPECTED_DURATION,:NEW.ACTUAL_DURATION,:NEW.EVENT_DATE,:NEW.EVENT_CO
ST,'INSERT',SYSDATE);
ELSIF UPDATING THEN
INSERT INTO EVENT_AUDIT VALUES
(:OLD.ID,:OLD.PEOPLE_COUNT,:OLD.STAFF_COUNT,:OLD.CUSTOMER_ID,:OLD.ADDRESS_ID,
:OLD.MENU_ID,:OLD.EXPECTED_DURATION,:OLD.ACTUAL_DURATION,:OLD.EVENT_DATE,:OLD
:EVENT_COST,
:NEW.PEOPLE_COUNT,:NEW.STAFF_COUNT,:NEW.CUSTOMER_ID,:NEW.ADDRESS_ID,:NEW.MENU
_ID,:NEW.EXPECTED_DURATION,:NEW.ACTUAL_DURATION,:NEW.EVENT_DATE,:NEW.EVENT_CO
ST'UPDATE',SYSDATE);
```

```
ELSIF DELETING THEN

INSERT INTO EVENT_AUDIT VALUES

(:OLD.ID,:OLD.PEOPLE_COUNT,:OLD.STAFF_COUNT,:OLD.CUSTOMER_ID,:OLD.ADDRESS_ID,
:OLD.MENU_ID,:OLD.EXPECTED_DURATION,:OLD.ACTUAL_DURATION,:OLD.EVENT_DATE,:OLD
.EVENT_COST,

NULL, NULL,NULL,NULL, NULL,NULL,NULL, NULL,NULL,'DELETE',SYSDATE);

END IF;

END;
```

2. Procedures

a. Procedure to set the total cost of food in an event

```
CREATE OR REPLACE PROCEDURE setCost

IS

thiseventcost EVENT_COST%rowtype;

CURSOR Cost_Update IS

SELECT * FROM EVENT_COST

FOR UPDATE;


BEGIN

OPEN Cost_Update;

LOOP

FETCH Cost_Update INTO thiseventcost;

EXIT WHEN (Cost_Update%NOTFOUND);

UPDATE EVENT SET event.COST=thiseventcost.ecost

WHERE event.ID=thiseventcost.eid;

END LOOP;

CLOSE Cost_Update;

END;
```

b. Procedure to calculate discount given to a customer for an event

```
CREATE OR REPLACE PROCEDURE calculateDiscount

(PEOPLE IN EVENT.PEOPLE_COUNT%TYPE,EVETNID IN EVENT.ID%TYPE)IS

THISDISCOUNT EVENT.COST%TYPE;

BEGIN

LOOP

IF PEOPLE>100 THEN

THISDISCOUNT := 0.05;
```

```
ELSIF PEOPLE>500 THEN

THISDISCOUNT :=0.10;

ELSIF PEOPLE>1000 THEN

THISDISCOUNT :=0.15;

END IF;

UPDATE EVENT SET EVENT.COST = EVENT.COST-THISDISCOUNT WHERE EVENT.ID=EVETNID;

END LOOP;

END;
```