Matthias Fast, Amos Dinh, Jannik Völker
4750990, 5504890, 5370226
21.06.2023

Documentation IceCreamEmpire

0 Project Overview

Please find the video of the project overview in the folder `video/`.

1 Requirements Specification

Introduction

The system is designed for the company IceCreamEmpire. The company wants to plan and track the daily tours to sell ice cream. It must assign vehicles and vendors to a tour through a neighborhood. It also wants to provide some information about the ice cream, where it is stored and what the sales numbers are. The system consists of the following entities, attributes, and relationships.

Tour

Each tour has a start-datetime and an end-datetime, as well as a unique ID. It is a one-time event and not a recurring event. Each tour is individually planned. There is one of each of the following assigned to a tour: ice cream vendor, vehicle, neighborhood.

Ice cream vendor

An ice cream vendor has a forename, a last name, a salary, and a unique ID. Vendors can sell ice cream on different tours.

Neighborhood

A neighborhood has a unique ID. It has a name, a distance to the headquarter and an area covered (square km). A neighborhood can be served ice cream on multiple tours.

Vehicle

Each Vehicle is identified by its unique ID. It has a vehicle type and storage capacity. One vehicle may be used for many tours. It may store different ice cream flavors with specific amounts.

Flavor

Each ice cream flavor is saved with its name and a unique ID. Each flavor has a base price per scoop.

Content

Each flavor can have a unique content (if the information is known) consisting of amount of calories and the basis (e.g. milk, water, hazelnut milk) from which the attribute if the flavor is vegan can be

derived. The content can be identified by corresponding flavor ID.

Order

If someone orders ice cream on a tour, an order linked to that tour is created. An order is identified by a unique ID and has a datetime and payment type.

Order detail

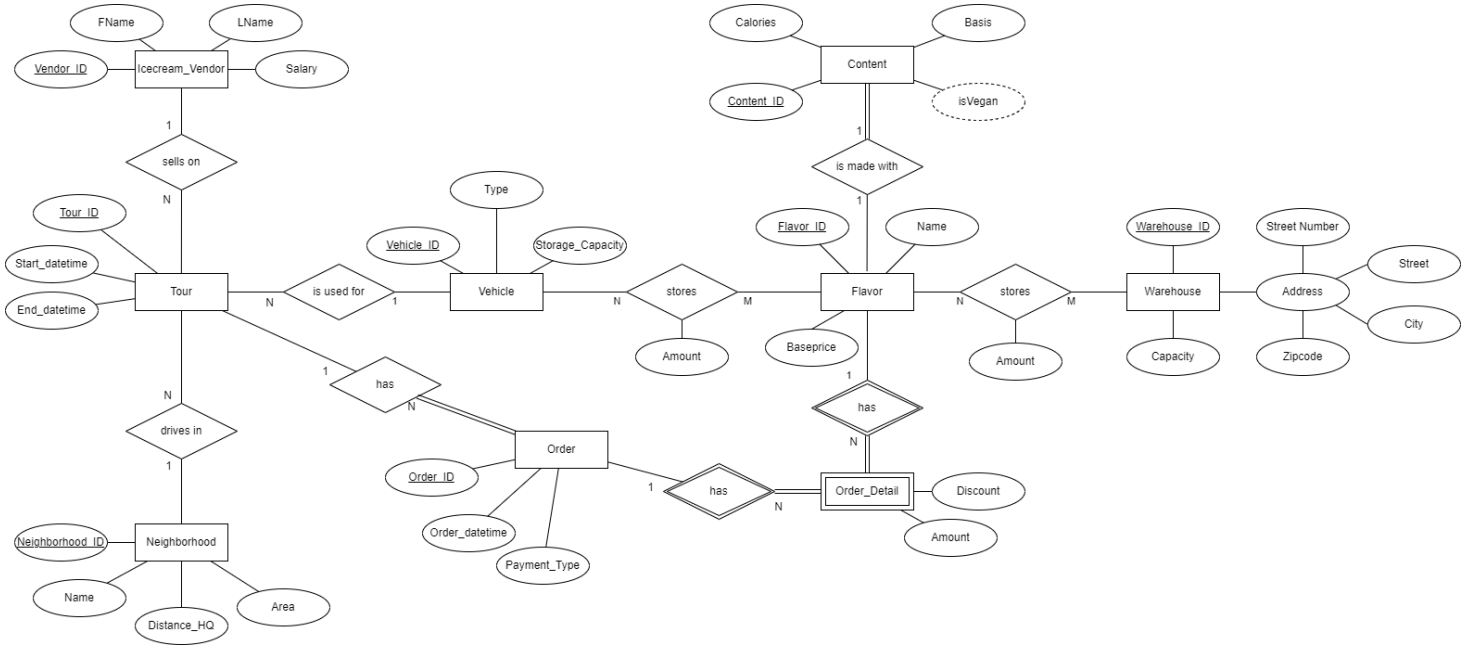
Each order consists of multiple order details. The order details can be differentiated by the corresponding flavor ID and order ID. A detail includes the amount of an ordered flavor. A vendor has the option to give a discount on the ordered flavor (of the order detail) e.g., for a child that has its birthday on that day. The discount is provided in percent and based on the base price of the flavor.

Warehouse

There are multiple warehouses. Each warehouse has an address (street number, street, city, zip code) and a capacity. It is identified by a unique ID. Each warehouse may store multiple flavors. Individual flavors may be stored in multiple warehouses. The amount of each stored flavor per warehouse is also saved in the system.

2 ER Diagram

The figure below represents the Entity-Relationship Diagram, which is a blueprint for the IceCreamEmpire database. In the ER-Diagram only the primary keys are underlined (not all candidate keys) to make it more readable.



3.1 Normalization

In this section the normalization is discussed. The Relations are required to be in 3NF. A relation is in 3NF, if it satisfies 2NF (and in turn 1NF) and no non-prime attributes are transitively dependent on any candidate key.
All relations in the database for IceCreamEmpire satisfy:

- 1NF since there are no multivalued attributes nor nested relations
- 2NF since there are no non-prime attributes which are only partially dependent on any candidate key
- 3NF

For example, violation of 2NF is avoided in `OrderDetails` by not specifying the price in `OrderDetails`, which would be functionally dependent on the partial key `Flavor_ID` because it is given in `Order` as `Baseprice`.
There is no violation of 3NF in relation `Warehouses`. `Capacity` is not transitively dependent on `warehouse_id` because the multivalued attribute `Address` itself is a candidate key, therefore there is no problem.

Contents

	flavor_id [PK] integer	calories integer	basis character varying	isvegan boolean
1	1	200	milk	false
2	2	250	milk	false
3	3	225	water	true

Flavors

	flavor_id [PK] integer	name character varying	base_price_per_scoop numeric (10,2)
1	1	Vanilla	1.00
2	2	Chocolate	1.50
3	3	Strawberry	1.25

IceCreamVendors

	vendor_id [PK] integer	forename character varying	lastname character varying	salary numeric (10,2)
1	1	John	Doe	3000.00
2	2	Jane	Doe	3500.00
3	3	Bob	Smith	3200.00

Neighborhoods

	neighborhood_id [PK] integer	name character varying	distance_to_headquarter_km numeric (10,3)	area_sqkm numeric (10,2)
1	1	Downtown	5.000	10.00
2	2	Uptown	10.000	15.00
3	3	Midtown	7.500	12.50

Orderdetails

	order_id [PK] integer	flavor_id [PK] integer	amount integer	discount integer
1	1	1	2	0
2	1	2	1	30
3	2	3	3	0
4	2	1	4	0

Orders

	order_id [PK] integer	tours_id integer	order_datetime timestamp without time zone	payment_type character varying
1	1	1	2023-05-10 12:30:00	cash
2	2	1	2023-05-10 13:00:00	credit
3	3	2	2023-05-11 13:30:00	cash

Tours

	tours_id [PK] integer	start_datetime timestamp without time zone	end_datetime timestamp without time zone	vendor_id integer	vehicle_id integer	neighborhood_id integer
1	1	2023-05-10 12:00:00	2023-05-10 16:00:00	1	1	1
2	2	2023-05-11 13:00:00	2023-05-11 17:00:00	2	2	2

Vehicles

	vehicle_id [PK] integer	type character varying	storage_capacity integer
1	1	Truck	1000
2	2	Van	800
3	3	Car	600

VehiclesStoreFlavours

	vehicle_id [PK] integer	flavor_id [PK] integer	amount integer
1	1	1	150
2	1	2	250
3	2	1	100
4	2	2	200
5	3	3	300

Warehouses

	warehouse_id [PK] integer	streetnumber character varying	street character varying	zipcode character varying	city character varying	capacity numeric (10,2)
1	1	123	Main Street	Palo Alto	14362	1000.00
2	2	456	Elm Street	Palo Alto	14362	1500.00
3	3	789	Oak Street	Palo Alto	14362	1200.00

WarehoursesStoreFlavours

	warehouse_id [PK] integer	flavor_id [PK] integer	amount integer
1	1	1	100
2	1	2	200
3	2	3	300
4	3	1	150
5	3	2	250

3.2 DDL and insert Statements, Views

DDL Statements for creation, the data insert statements as well as the defined views, the procedure and secondary indices may be found in the `src/db/init.sql` file.

The queries and their description may be found in the `src/db/queries.sql` file.

4 Application

The application uses `docker` to make it portable. For the database `postgresql` is used. The application layer uses `SQLAlchemy` to access the database. `sqlalchemy.ext.automap.automap_base` is used in some instances to map the `postgresql` relations to python. The frontend uses the python package `streamlit`. On every user input the whole code defined in the frontend is rerun (stateless). Data which should outlast this refresh can be stored in a session state.

Folder & File Structure

`doc/`	Documentation and ER diagram
`src/db/`	DDL & Insert SQL files
`src/frontend/`	Frontend
`src/frontend/app/app.py`	Main app
`src/frontend/app/crudorder.py`	Order and Orderdetail CRUD
`src/frontend/app/crudtour.py`	Tour CRUD
`src/frontend/app/classes/queries.py`	SQLAlchemy connection

From the frontend, the relation extension can be viewed.
In the dashboard, total sales per vendor (refreshed on button click, using the materialized view from 3.), revenue per flavor, Vendor salaries, and the current stock per flavor can be viewed. CRUD operations are available for Tours, Orders and respective Orderdetails. More information is provided in the video.

5 Starting the application

To start the application, run from inside the zip-folder	`docker-compose up`
The frontend can then be accessed with:	`localhost`