The Primary Keys

COS 221 Practical Assignment 5

u24569608 - Mr. Connor Bell
u23536030 - Mr. Dewald Colesky
u24634434 - Mr. Adriano Jorge
u05084360 - Miss. Zoë Joubert
u23544563 - Miss. Inge Keyser
u23608821 - Miss. Megan Lai
u24594475 - Mr. Johan Nel

Task 1: Research

1. What Consumers Buy Online

Clothing and shoes are the most popular online purchases worldwide. In the U.S., 57% of shoppers bought clothes online in the past year, while 47% bought footwear. Electronics are also a big category, with 40% of U.S. consumers making purchases in this area. In Germany, fashion leads at 66%, followed by electronics (50%), and groceries—which saw a huge jump from just 7% in 2020 to 25% in 2024. Personal care products also grew significantly, doubling from 18% to 35% in the same period. Globally, 34% of shoppers buy something online at least once a week, showing how common digital shopping has become [1][2][4].

2. How Shoppers Compare Prices

Price comparison tools help consumers find the best deals by showing prices from different retailers in one place. Popular options include Google Shopping, Shopzilla, and Camelcamelcamel. For businesses, tools like 42Signals track competitor prices in real time, monitor stock levels, and even detect when sellers break pricing agreements. Research shows these tools make shoppers more likely to switch stores for a better price—especially those who are budget-conscious. Newer systems even use web scraping to provide instant price updates across major online stores [5][6][7][11].

3. Top Online Shopping Categories

Fashion is still the biggest e-commerce category globally, followed by electronics, books/media, and cosmetics. In the U.S., household appliances (27%) and furniture/home goods (19%) are also popular. Fast-growing niches include sports gear, toys, and DIY/garden supplies. Interestingly, over half of online shoppers (52%) buy from international retailers, proving that e-commerce has no borders [1][4].

4. Why User Experience (UX) Matters

A smooth, easy-to-use website directly impacts sales—better UX means more conversions, higher spending, and fewer abandoned carts. Studies show that poor design can drive away up to 80% of potential customers, especially if the checkout process is too complicated. Key features that improve UX include:

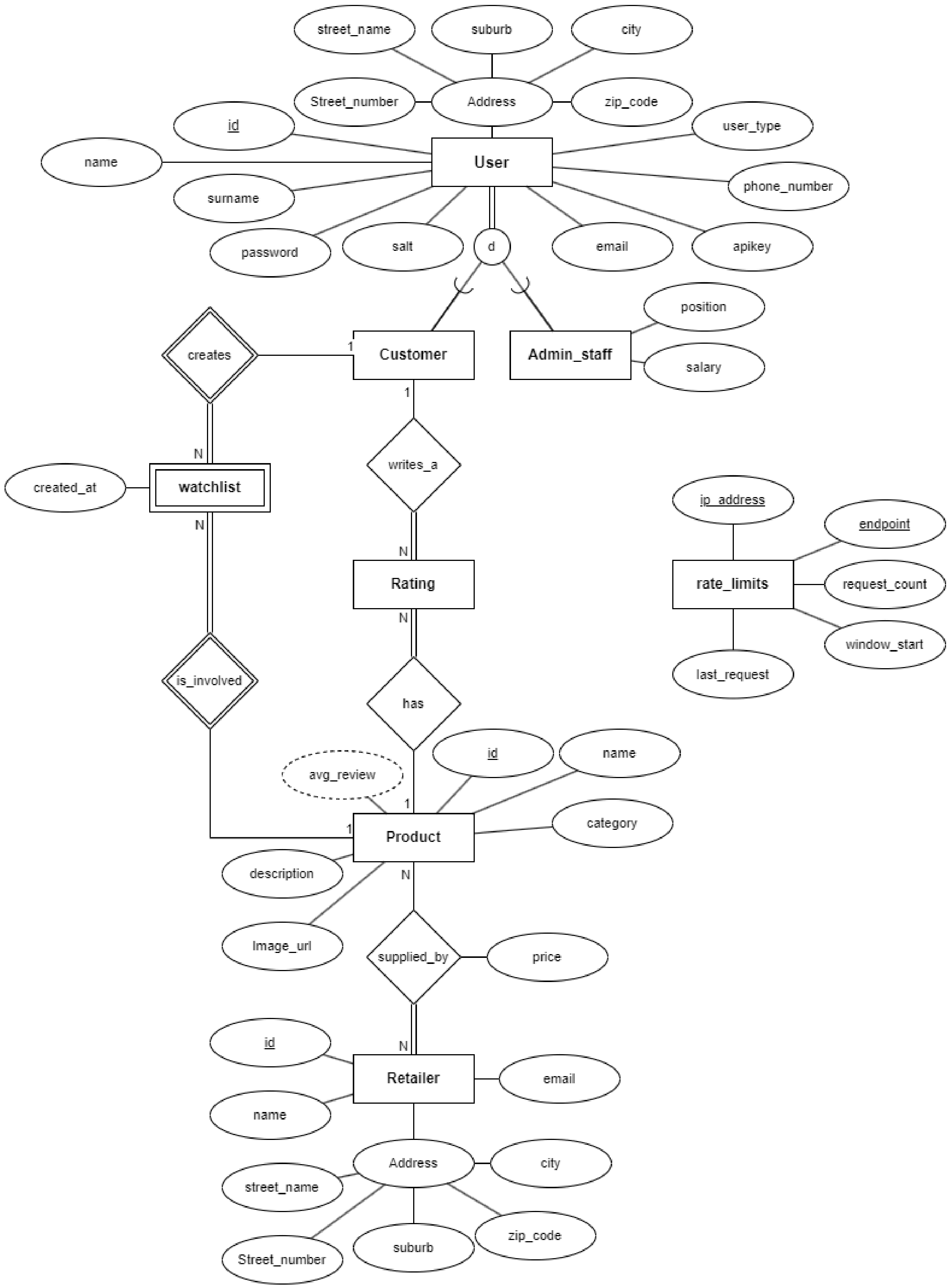
- Simple navigation and search
- Quick, hassle-free checkout

- Clear product reviews and ratings
- Mobile-friendly design Companies that invest in UX gain a long-term advantage because shoppers now expect fast, intuitive online experiences [7][8][9][10].

References:

[1] Sellers Commerce, “51 eCommerce Statistics In 2025 (Global and U.S. Data),” Apr. 15, 2025.
[2] C. Stacey, “Tech-wary Germans get hooked with online shopping habit,” Reuters, Nov. 25, 2024.
[3] A. Rogers and J. Hitchcock, “20 Trending Products and Things to Sell Online (2025),” Shopify, Mar. 25, 2025.
[4] “Most Popular Online Shopping Categories,” Mobiloud, 2025.
[5] “23 Price Comparison Apps, Tools, and Websites (2025),” Shopify, 2025.
[6] “The Art of Price Comparison: Tools and Tips for Online Retailers,” 42Signals, 2023.
[7] A. Smith et al., “The Influence of Price Comparison Websites on Online Switching Behavior,” PMC, 2020.
[8] “The Importance Of User Experience In eCommerce Design,” Wizzy.ai, 2023.
[9] J. Doe, “Clunky Websites Are Costing Companies Money, Scaring Away Shoppers,” Investopedia, 2024.
[10] “Benefits of UX design in E-commerce development,” Capturly, 2023.
[11] F. Chen et al., “Research on Real-time E-commerce Price Comparison System Using Python Web Scraping Technology,” ResearchGate, 2024.

Task 2: (E)ER-Diagram


street_name
suburb
city
Street_number
Address
zip_code
id
user_type
name
User
phone_number
surname
password
salt
d
email
apikey
position
creates
1
Customer
Admin_staff
salary
1
writes_a
N
created_at
watchlist
N
N
ip_address
endpoint
Rating
rate_limits
request_count
N
window_start
last_request
is_involved
has
id
name
avg_review
1
1
Product
category
description
N
Image_url
supplied_by
price
id
N
Retailer
email
name
Address
city
street_name
zip_code
Street_number
suburb

Comments:

- User is a strong entity with a disjoint specialization based on user_type. It has total participation: every User must be either a Customer or Admin_staff. Disjointness means that a user can be only one of the two, never both. A person in our system must have a role (either admin or customer), and cannot hold both roles simultaneously. This enforces clear access control logic (e.g., customers view and watchlist products, admins manage products, add retailers, etc.).

- avg_review is a derived attribute of Product. It is calculated from Rating entries (score) that are associated with this product.

- Watchlist is a weak entity: It cannot exist without both a Customer and a Product. It has no unique attribute to identify it alone. It is identified by a composite key (customer_id and product_id).

- Rating is a strong entity: It has its own primary key (id), and while it references both Customer and Product, it can exist independently.

- The rate_limits entity is a standalone system-level control table used to enforce rate-limiting mechanisms on the API. It is not part of the business (e.g., users, products, orders), but is essential to prevent API abuse. endpoint and ip_address form a composite primary key, uniquely identifying each tracked client-endpoint pair. This entity has no relationships to any other entities - its role is entirely operational/systemic.

Relationships:

- Customer creates Watchlist: One customer creates many watchlist entries. A customer might not create any watchlist entries. Every watchlist entry must be created by a customer. Customers can save products to their watchlist, but aren't required to

- Product is_involved Watchlist: One product can be in many watchlists. A product might not be in any watchlists. Every watchlist entry must reference a product. Products can be watched by multiple customers

- Customer writes_a Rating: One customer writes many ratings. A customer might not write any ratings. Every rating must be written by a customer. Customers can review products, but aren't required to

- Product has Rating: One product has many ratings. A product might not have any ratings yet. Every rating must be for a product. Products accumulate reviews over time

- Product supplied_by Retailer: Many products are supplied by many retailers. A product might not be supplied by any retailer. A retailer might not supply any products. Retailers carry inventory of products at specific prices. The price for this product as sol by this retailer is stored in a "price" attribute.

Task 3: (E)ER-diagram to Relational Mapping

Step 1: Mapping of regular entity types Regular entities are mapped in this step.

rate_limits

ip_address	endpoint	request_count	window_start	last_request

User

id	name	surname	user_type	email	password	salt	phone_number	apikey	street_number	street_name	suburb	city	zip_code

Rating

id	score	description	updated_at

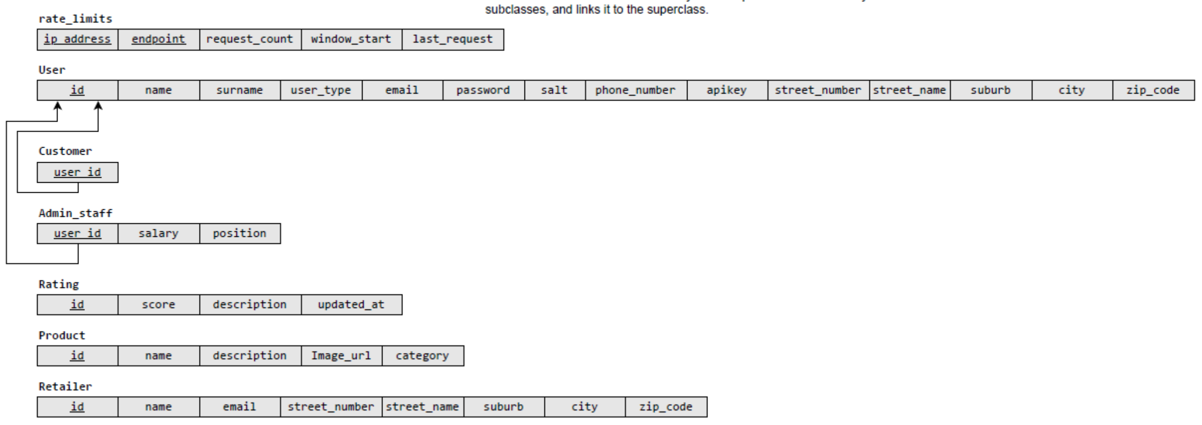
Product

id	name	description	Image_url	category

Retailer

id	name	email	street_number	street_name	suburb	city	zip_code

Step 8: Mapping specialisation and generalisation We need to do this step early because we need to use CUSTOMER in the coming steps. User is totally disjoint specialised into two types: CUSTOMER and ADMIN_STAFF. We use approach 8A in the slides of L14: That is, create a relation for each subclass (customer and admin_staff) with the key of the superclass (user) and the attributes of this subclass. The key of the superclass will be the key of the subclasses, and links it to the superclass.



rate_limits

ip_address	endpoint	request_count	window_start	last_request

User

id	name	surname	user_type	email	password	salt	phone_number	apikey	street_number	street_name	suburb	city	zip_code

Customer

user_id

Admin_staff

user_id	salary	position

Rating

id	score	description	updated_at

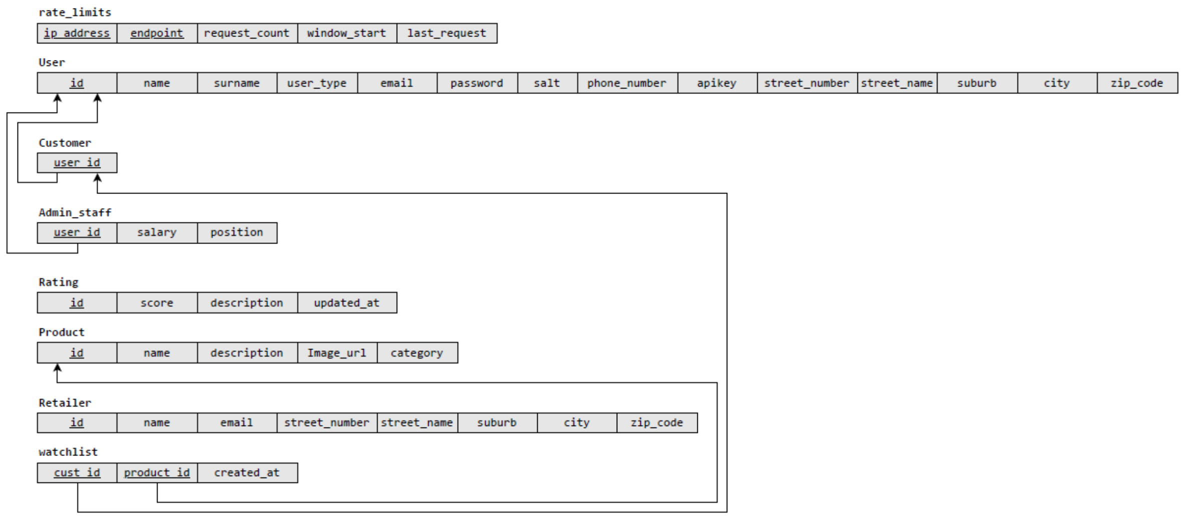
Product

id	name	description	Image_url	category

Retailer

id	name	email	street_number	street_name	suburb	city	zip_code

Step 2: Mapping of weak entity types We map weak entities in this step. WATCHLIST is the only weak entity in our EERD, since it depends on both a CUSTOMER and a PRODUCT to exist: Without them, the watchlist entry cannot exist. RATING is not a weak entity since ratings can be updated over time, ratings may need to be deleted, editied or moderated independetly of the user or product that it is linked to.



rate_limits

ip_address	endpoint	request_count	window_start	last_request

User

id	name	surname	user_type	email	password	salt	phone_number	apikey	street_number	street_name	suburb	city	zip_code

Customer

user_id

Admin_staff

user_id	salary	position

Rating

id	score	description	updated_at

Product

id	name	description	Image_url	category

Retailer

id	name	email	street_number	street_name	suburb	city	zip_code

watchlist

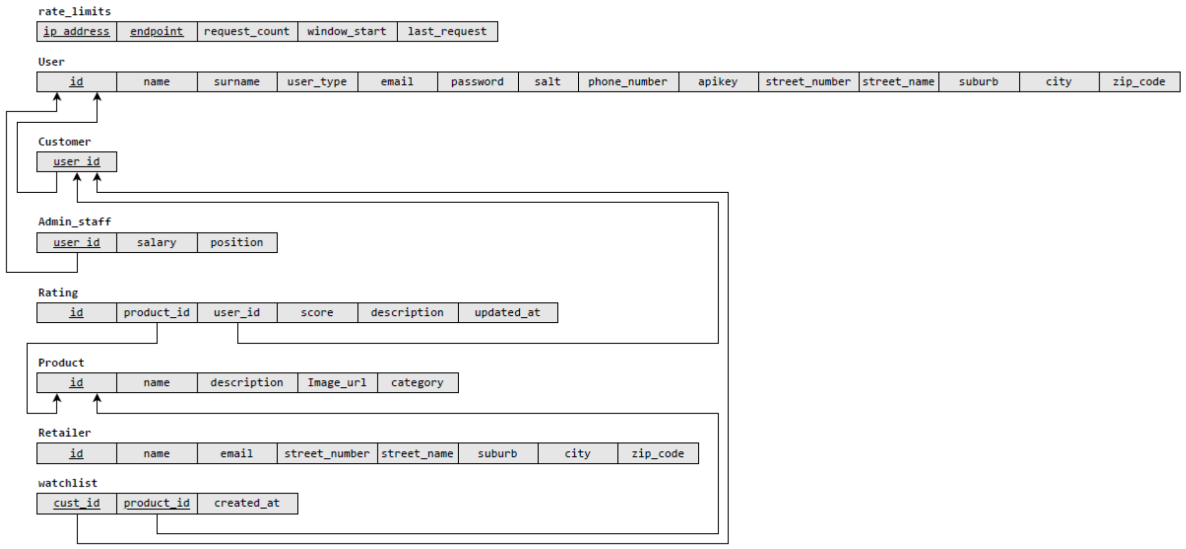
cust_id	product_id	created_at

Step 3: Mapping of Binary 1:1 relationships

We do not have binary 1:1 relationships in our EERD. We do nothing in this step

Step 4: Mapping of Binary 1:M relationships

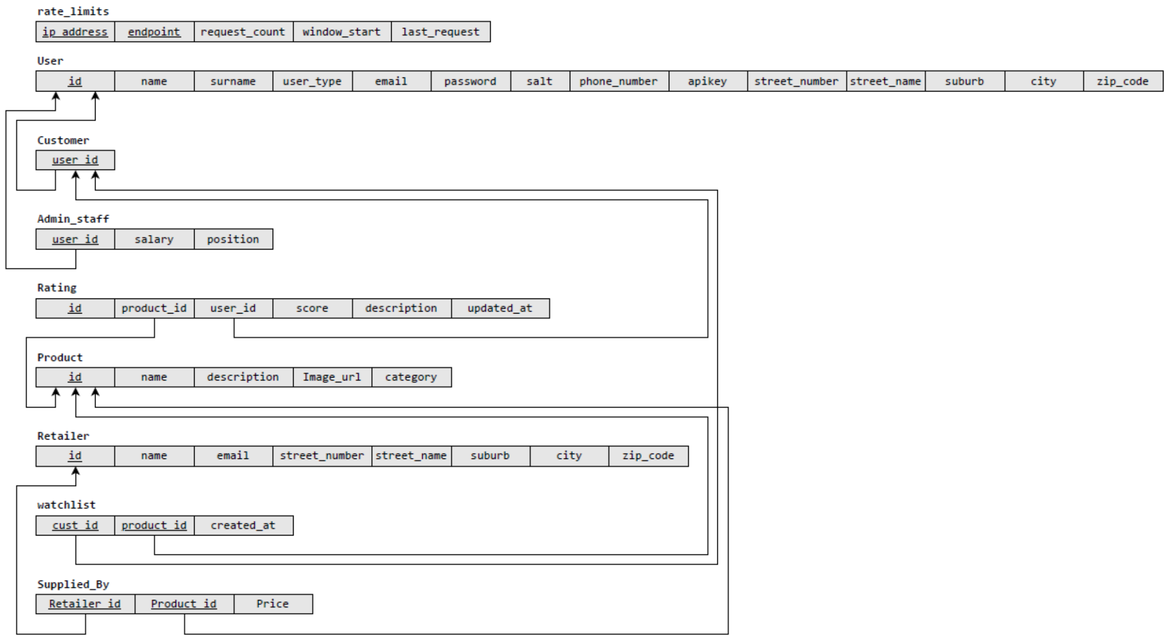
We have four 1:M relationships in our EERD: CUSTOMER creates WATCHLIST, PRODUCT is_invloded WATCHLIST (both of these are already mapped in Step 2), CUSTOMER writes_a RATING and PRODUCT has RATING. We use Approach 1 in the slides of L14



rate_limits
ip_address | endpoint | request_count | window_start | last_request
User
id | name | surname | user_type | email | password | salt | phone_number | apikey | street_number | street_name | suburb | city | zip_code
Customer
user_id
Admin_staff
user_id | salary | position
Rating
id | product_id | user_id | score | description | updated_at
Product
id | name | description | Image_url | category
Retailer
id | name | email | street_number | street_name | suburb | city | zip_code
watchlist
cust_id | product_id | created_at

Step 4: Mapping of Binary N:M relationships

We have one binary N:M relationship in our EERD: PRODUCT supplied_by RETAILER. We create a new relation called SUPPLIED_BY to support this and we add a price attribute since it is an attribute that belongs to the supplied_by relationship.



rate_limits
ip_address | endpoint | request_count | window_start | last_request
User
id | name | surname | user_type | email | password | salt | phone_number | apikey | street_number | street_name | suburb | city | zip_code
Customer
user_id
Admin_staff
user_id | salary | position
Rating
id | product_id | user_id | score | description | updated_at
Product
id | name | description | Image_url | category
Retailer
id | name | email | street_number | street_name | suburb | city | zip_code
watchlist
cust_id | product_id | created_at
Supplied_By
Retailer_id | Product_id | Price

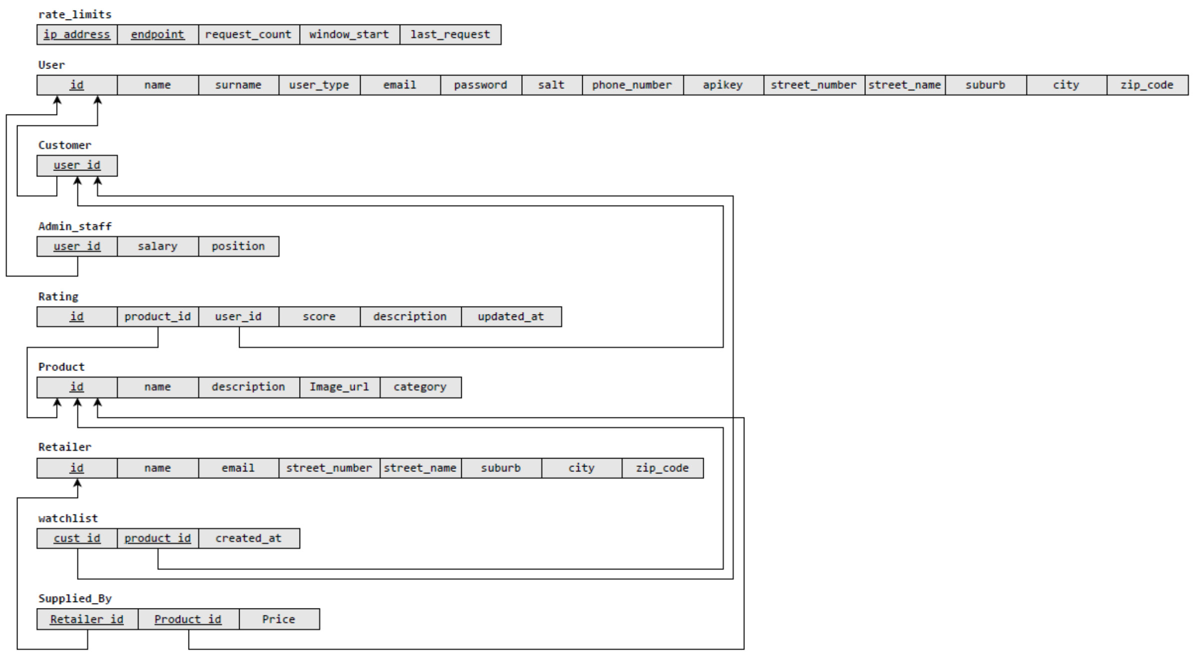
Step 6: Mapping of multivalued attributes We do not have multivalued attributes in our EERD. We do nothing in this step.

Step 7: Mapping of N-ary relationships We do not have N-ary relationships in our EERD. We do nothing in this step.

Step 8: Mapping specialisation and generalisation Step 8 was already done directly after step 1 to ensure that CUSTOMER exists before we map relationships. We do not have other specialisations in our EERD.

Step 9: Mapping unions We do not have unions in our EERD. We do nothing in this step.

FINAL RELATIONAL MAPPING


rate_limits
ip_address | endpoint | request_count | window_start | last_request
User
id | name | surname | user_type | email | password | salt | phone_number | apikey | street_number | street_name | suburb | city | zip_code
Customer
user_id
Admin_staff
user_id | salary | position
Rating
id | product_id | user_id | score | description | updated_at
Product
id | name | description | Image_url | category
Retailer
id | name | email | street_number | street_name | suburb | city | zip_code
watchlist
cust_id | product_id | created_at
Supplied_By
Retailer_id | Product_id | Price

Task 4: Relational Schema

Task 5: Web-based Application

Task 6: Data

Task 7: Analyse and Optimise

Task 8: Development

Task 9: Demo

Connor Bell:

Dewald Colesky:

Adriano Jorge:

Zoë Joubert:

Inge Keyser:

Megan Lai:

Johan Nel:

API implementation and documentation

Task 10: Bonus Task - Push the Boundaries

2. Security First - Secure Your App

We use password hashing implementing salt strings together with hashing to secured passwords.

Login attempts are rate-limiting:
How Rate Limiting Works

Database-Backed:

Rate limiting is implemented using a dedicated rate_limits table in the database. This allows for accurate and persistent tracking of request counts and time windows to ensure that rate limits are enforced correctly.

Per-IP and Per-Endpoint:

Limits are tracked for each unique combination of IP address and the endpoint that this IP is accessing. Each endpoint may have a different rate limit.

Time Window:

The time window for rate limiting is typically 60 seconds (1 minute), but may vary per endpoint.

Limits:

The number of allowed requests per endpoint per minute varies based on how computationally expensive the endpoint is.

For example:

Register and Login: 5 requests per minute per IP
getReviewStats (all stats): 3 requests per minute per IP
getReviewStats (single stat): 10 requests per minute per IP

Most other endpoints allow for 10–20 requests per minute per IP, depending on the endpoint's complexity.

How it works:

On each request, the API checks the rate_limits table for your IP and the endpoint.
If you have not exceeded the allowed number of requests in the current time window, your request proceeds and your request count is incremented.
If you exceed the limit, the API immediately returns a 429 error and does not process your request.
When the time window expires, your request count is reset.

Error Response Example

If you exceed the rate limit, you will receive a response like this:

```
{
  "status": "error",
  "timestamp": 1748000000000,
  "code": 429,
  "message": "Rate limit exceeded. Please try again later."
}
```