Computer and Systems Engineering [CSE]

Parallel and Distributed Systems [CSE352s]

Assignment (3)

Team Members

Name	ID
Mostafa Hamada Hassan Ahmed	2100587
Zeyad Magdy Roushdy	2100393
Mina Nasser Shehata	2100370
Ahmed Ashraf Ahmed	2100476
Mina Antony Samy Fahmy	2101022
Mark Wagdy Wadie	2100822
Mohamed Montasser Ibrahim	2100416
George Joseph Basilious Tawadrous	2100261
Youssef Hatem Ahmed	2101052

Contents

List of Figures

1) Database Schema Explanation for Distributed Marketplace System

In this distributed marketplace system, the database schema is structured using a combination of horizontal, vertical, and hybrid fragmentation techniques to improve scalability, performance, and distribution of data across multiple servers. Here's a breakdown of the schema design and the fragmentation strategies applied:

1.1) Horizontal Fragmentation

Horizontal fragmentation divides the tables into subsets of rows based on certain conditions. In this case, we partition the tables based on user activity, country, item publication status, and transaction status. Each subset is stored on separate servers, which helps distribute the load and ensures faster query processing.

```sql
-- Active users
CREATE TABLE auth_user_active AS
SELECT * FROM auth_user WHERE is_active = 1;

-- Inactive users
CREATE TABLE auth_user_inactive AS
SELECT * FROM auth_user WHERE is_active = 0;

-- Profiles from US
CREATE TABLE accounts_profile_us AS
SELECT * FROM accounts_profile WHERE country = 'US';

-- Profiles from other countries
CREATE TABLE accounts_profile_other AS
SELECT * FROM accounts_profile WHERE country ≠ 'US';

-- Published items
CREATE TABLE Item_item_published AS
SELECT * FROM Item_item WHERE Item_published = 1;

-- Unpublished items
CREATE TABLE Item_item_unpublished AS
SELECT * FROM Item_item WHERE Item_published = 0;

```

```sql
-- Completed transactions
CREATE TABLE Market_transaction_completed AS
SELECT * FROM Market_transaction WHERE transaction_status = 'completed';

-- Pending transactions
CREATE TABLE Market_transaction_pending AS
SELECT * FROM Market_transaction WHERE transaction_status = 'pending';
```

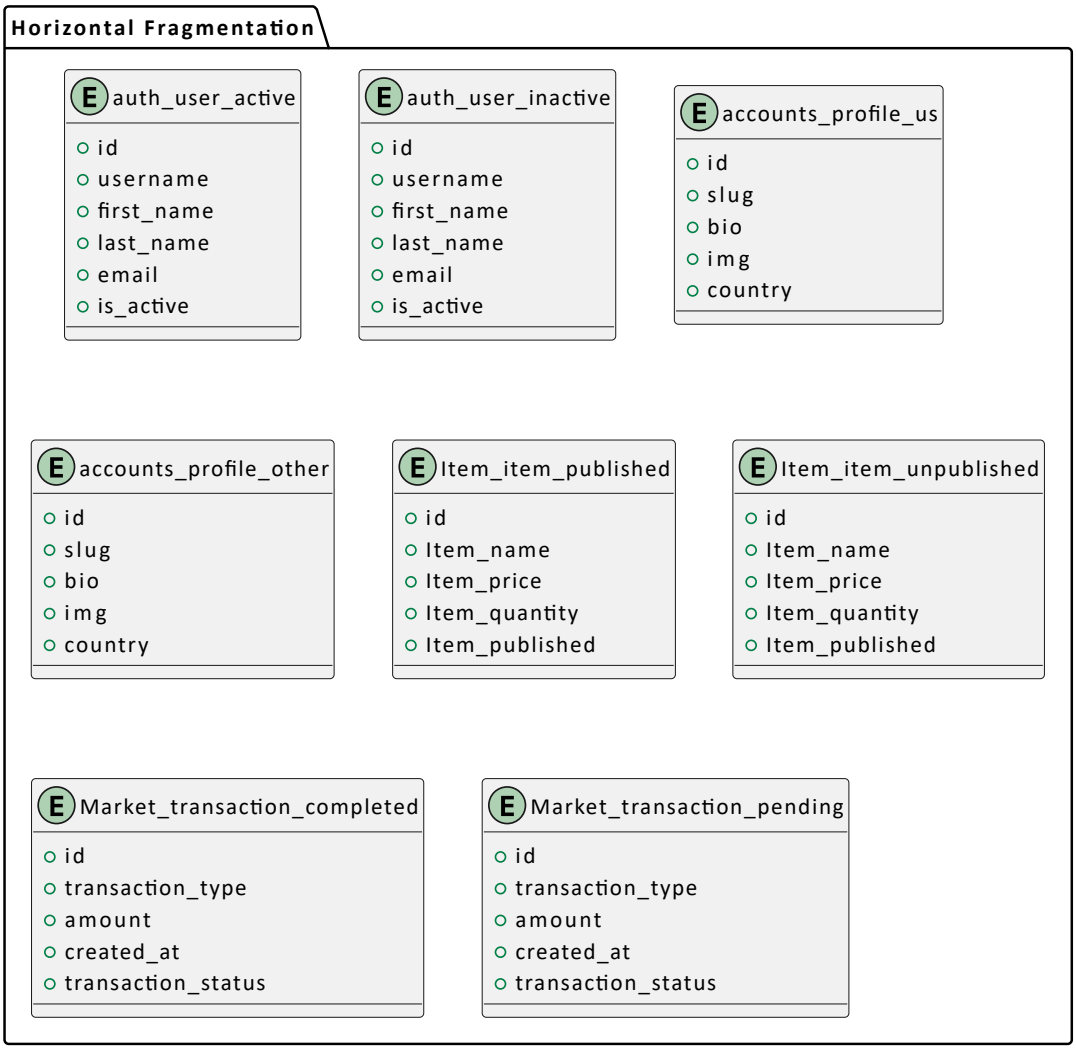

Figure 1: Horizontal Fragmentation

- Active Users (`auth_user_active`): This table contains only the rows where users are marked as active (`is_active = 1`). This separation makes querying active users more efficient since it is a frequently accessed subset.

- Inactive Users (`auth_user_inactive`): Similar to the active users table, this stores all users marked as inactive (`is_active = 0`).
- Profiles from US (`accounts_profile_us`): This table stores profiles where the country is the US (`country = 'US'`). It helps localize user profiles by region and ensures optimized access for users from the US.
- Profiles from Other Countries (`accounts_profile_other`): All profiles from countries other than the US are placed here (`country ≠ 'US'`), ensuring region-based distribution and reducing cross-server queries.
- Published Items (`Item_item_published`): Contains only the items that are published (`Item_published = 1`). This allows for optimized querying of items that are available for sale or viewing.
- Unpublished Items (`Item_item_unpublished`): Contains unpublished items (`Item_published = 0`). It allows for efficient processing of items that are not yet available to users.
- Completed Transactions (`Market_transaction_completed`): This table stores completed transactions (`transaction_status = 'completed'`), which allows efficient access to finalized transaction data.
- Pending Transactions (`Market_transaction_pending`): This table stores pending transactions (`transaction_status = 'pending'`), enabling faster processing of ongoing transactions.

1.2) Vertical Fragmentation

Vertical fragmentation involves splitting a table into columns rather than rows, ensuring that related columns are grouped together for faster retrieval and reducing the amount of data transferred between servers. This fragmentation optimizes access to specific subsets of columns for certain operations, improving efficiency.

```sql
-- Personal details
CREATE TABLE auth_user_identity AS
SELECT id, username, first_name, last_name, email FROM auth_user;

-- Account details
```

```sql
CREATE TABLE auth_user_security AS
SELECT id, password, is_active, is_staff, is_superuser, last_login,
date_joined FROM auth_user;

-- Public info
CREATE TABLE accounts_profile_public AS
SELECT id, slug, bio, img, country FROM accounts_profile;

-- Private info
CREATE TABLE accounts_profile_private AS
SELECT id, address, birth_date, joindate, balance, user_id FROM
accounts_profile;

-- Item summary
CREATE TABLE Item_item_main AS
SELECT id, Item_name, Item_price, Item_quantity, Item_published FROM
Item_item;

-- Item details
CREATE TABLE Item_item_details AS
SELECT id, Item_description, Item_createdat, Item_slug,
Item_category_id, Item_img, Item_owner_id FROM Item_item;

-- Transaction meta
CREATE TABLE Market_transaction_info AS
SELECT id, transaction_type, amount, created_at, transaction_status
FROM Market_transaction;

-- Approval and linkage
CREATE TABLE Market_transaction_approval AS
SELECT id, from_approve, to_approve, admin_approve, user_from_id,
user_to_id, items_id FROM Market_transaction;
```

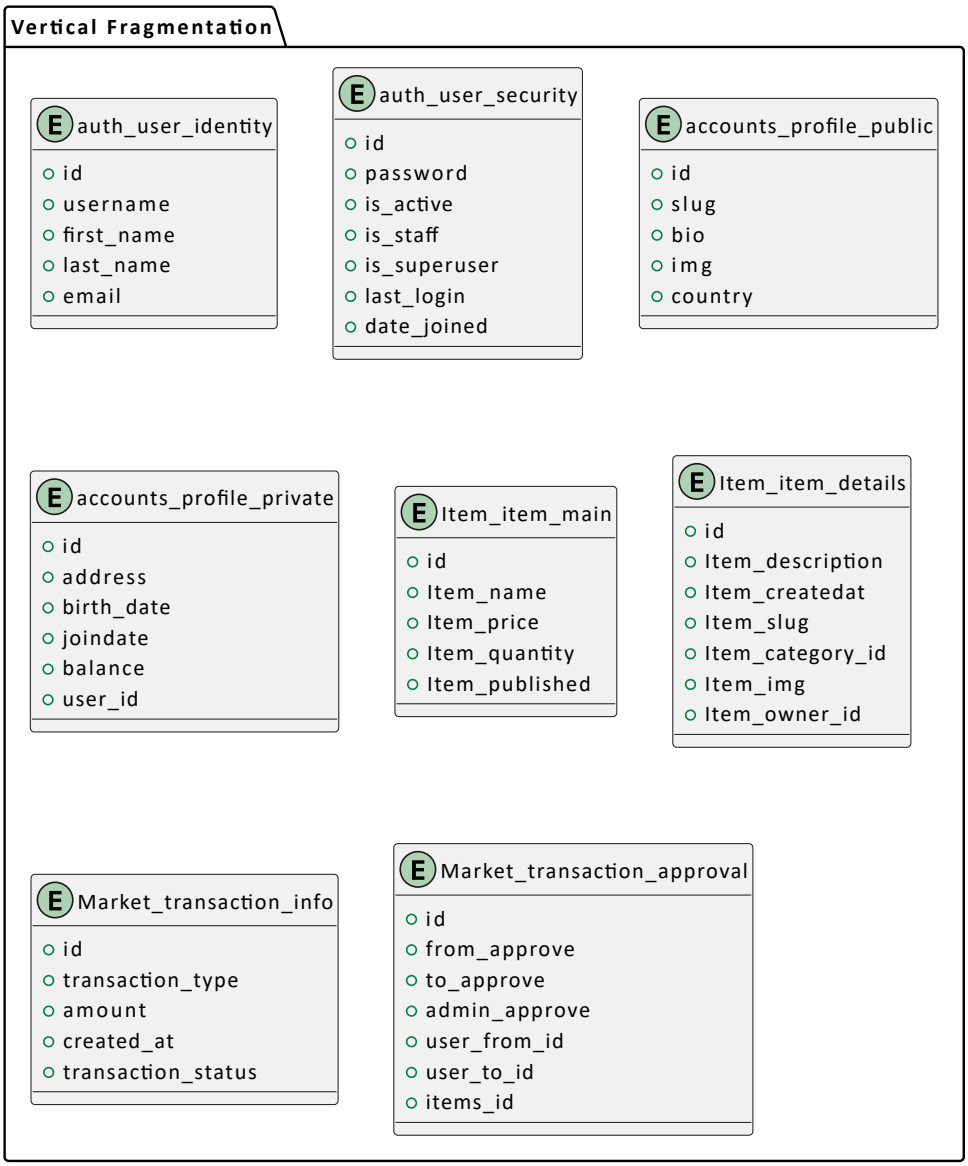

Figure 2: Vertical Fragmentation

- User Identity (`auth_user_identity`): Contains personal details such as username, first name, last name, and email. This table allows access to user identity information without loading security-related columns.
- User Security (`auth_user_security`): This table includes sensitive information like password, activity status (`is_active`, `is_staff`), and login data, such as last login and date joined. Separating security-related data helps secure user access and simplifies authentication operations.

- Public Info (`accounts_profile_public`): Contains public information like profile bio, image, and country. These details are often required for displaying public profiles and can be accessed separately from private information.
- Private Info (`accounts_profile_private`): Contains sensitive data such as address, birth date, balance, and the user's account ID. This separation ensures that private information is only accessed when necessary, improving privacy.
- Item Summary (`Item_item_main`): Contains high-level details like item name, price, quantity, and publication status. This table provides an overview of items without retrieving detailed descriptions or metadata.
- Item Details (`Item_item_details`): Contains detailed information about items, including descriptions, creation date, slug, category ID, images, and owner ID. It can be queried separately when more granular item details are needed.
- Transaction Meta (`Market_transaction_info`): Stores metadata about transactions, such as transaction type, amount, and status. This allows efficient access to transaction summaries without fetching full approval data.
- Transaction Approval (`Market_transaction_approval`): Contains approval-related data for transactions, including whether approvals were given by specific users or administrators. This helps in managing approval workflows.

1.3) Hybrid Fragmentation

Hybrid fragmentation combines both horizontal and vertical fragmentation to optimize data storage and access based on usage patterns. It allows more control over the partitioning of both rows and columns in a way that suits specific application needs.

```sql
-- Published items
CREATE TABLE Item_item_published_main AS
SELECT id, Item_name, Item_price, Item_quantity FROM Item_item WHERE
Item_published = 1;

CREATE TABLE Item_item_published_detail AS
```

```sql
SELECT id, Item_description, Item_createdat, Item_slug, Item_category_id, Item_img, Item_owner_id
FROM Item_item WHERE Item_published = 1;

-- Unpublished items
CREATE TABLE Item_item_unpublished_main AS
SELECT id, Item_name, Item_price, Item_quantity FROM Item_item WHERE Item_published = 0;

CREATE TABLE Item_item_unpublished_detail AS
SELECT id, Item_description, Item_createdat, Item_slug, Item_category_id, Item_img, Item_owner_id
FROM Item_item WHERE Item_published = 0;

-- Completed transactions
CREATE TABLE Market_transaction_completed_info AS
SELECT id, transaction_type, amount, created_at, transaction_status
FROM Market_transaction WHERE transaction_status = 'completed';

CREATE TABLE Market_transaction_completed_approval AS
SELECT id, from_approve, to_approve, admin_approve, user_from_id, user_to_id, items_id
FROM Market_transaction WHERE transaction_status = 'completed';

-- Pending transactions
CREATE TABLE Market_transaction_pending_info AS
SELECT id, transaction_type, amount, created_at, transaction_status
FROM Market_transaction WHERE transaction_status = 'pending';

CREATE TABLE Market_transaction_pending_approval AS
SELECT id, from_approve, to_approve, admin_approve, user_from_id, user_to_id, items_id
FROM Market_transaction WHERE transaction_status = 'pending';
```

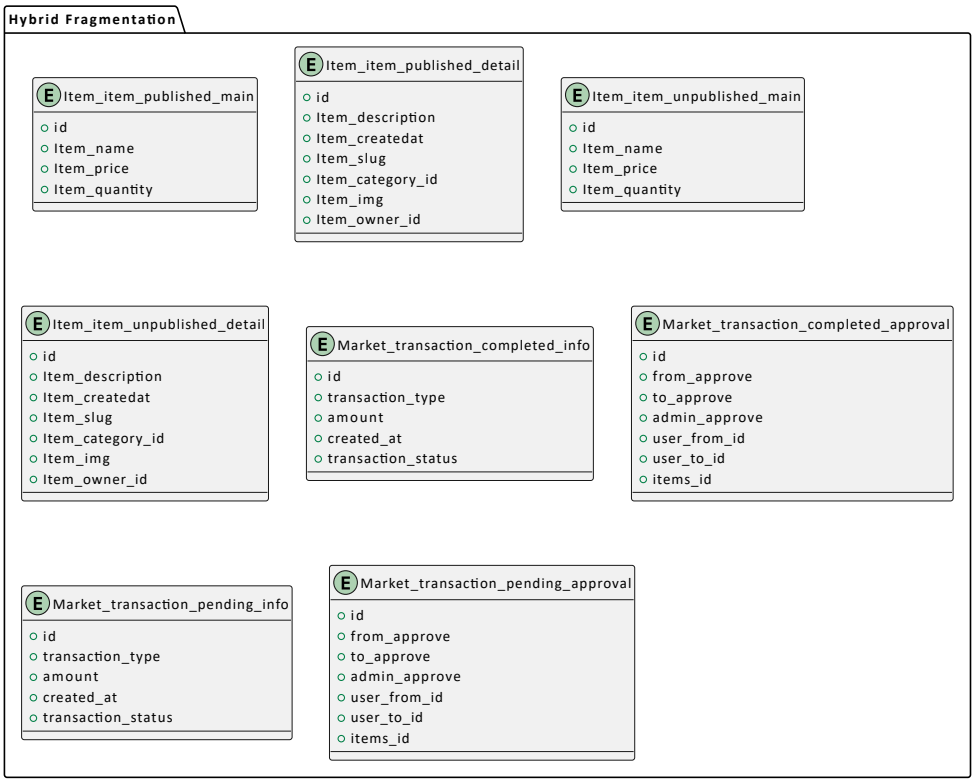

Figure 3: Hybrid Fragmentation

- Published Items (Main and Detail):
  - `Item_item_published_main`: Contains summarized data about published items (name, price, quantity).
  - `Item_item_published_detail`: Contains detailed information about published items (description, creation date, images).

These two tables split the published items data into two parts, allowing for faster retrieval of item summaries and detailed information separately, thus reducing unnecessary data transfer.

- Unpublished Items (Main and Detail):
  - `Item_item_unpublished_main`: Contains summarized data about unpublished items (name, price, quantity).
  - `Item_item_unpublished_detail`: Contains detailed information about unpublished items (description, creation date, images).

Similarly to the published items, unpublished items are fragmented into two parts, optimizing both the storage and query performance for unpublished items.

- Completed Transactions (Meta and Approval):

- Market_transaction_completed_info: Contains transaction metadata for completed transactions (type, amount, status).
- Market_transaction_completed_approval: Contains approval-related information for completed transactions (approval status, user IDs).

Separating transaction meta data from approval details allows for more efficient querying of transaction data versus approval processing.

- Pending Transactions (Meta and Approval):
  - `Market_transaction_pending_info`: Contains transaction metadata for pending transactions (type, amount, status).
  - `Market_transaction_pending_approval`: Contains approval-related information for pending transactions (approval status, user IDs).

Similarly, splitting pending transaction data into meta and approval subsets allows efficient processing for both types of operations.