Real World SQL Insights

1. Foreign key cascading

- **ON DELETE CASCADE:** The ON DELETE CASCADE, a referential action in Postgres, allows database developers to automatically delete related rows in child tables when a parent row is deleted from the parent table. This feature makes sure that dependent rows are deleted along with their related rows, which helps you preserve referential integrity in the database. (Gyamfi, n.d.)

Implement the DELETE CASCADE action

```
        CREATE TABLE parent_table(
          id SERIAL PRIMARY KEY,
          ...
        );

        CREATE TABLE child_table(
          id SERIAL PRIMARY KEY,
          parent_id INT,
          FOREIGN_KEY(parent_id)
            REFERENCES parent_table(id)
            ON DELETE CASCADE
);
```

- **ON UPDATE CASCADE:** The ON UPDATE CASCADE option is similar to ON DELETE CASCADE but applies when the primary key in the parent table is updated. All corresponding foreign key values in the child table are automatically updated to match the new primary key value. (prasad, 2024)

Implement the UPDATE CASCADE action

CREATE TABLE parent_table (
  parent_id INT **PRIMARY** KEY
);
CREATE TABLE child_table (
  child_id INT **PRIMARY** KEY,
  parent_id INT,
  FOREIGN KEY (parent_id) **REFERENCES**
parent_table(parent_id) **ON UPDATE** CASCADE
); (geeksforgeeks, 2024)

- **The potential risks or consequences of using cascading deletes:**
    - i. **Accidental Data Loss:** Deleting a parent record automatically deletes all related child records, which can result in unintended data loss.
    - ii. **Complexity:** Understanding how deletes impact related tables can be confusing, especially in large databases.
    - iii. **Performance Issues:** Deleting multiple related records can slow down the database, especially if there are many related tables.
    - iv. **Maintenance Challenges:** Adjusting cascading rules later can be complicated and error-prone.
    - v. **Transaction Conflicts:** Multiple cascading paths can cause conflicts and deadlocks.
    - vi. **Data Integrity Risks:** Important historical data can be lost permanently without notice.
    - vii. **Debugging Difficulty:** Tracking what records are deleted becomes harder, making debugging more complex.

(Beckett, 2020)

2. DQL in Action

- **How Systems Like Amazon, Netflix, and Banking Apps Use SELECT Queries:**

Amazon

1. **Product Search & Recommendations:** When users search for products, Amazon uses SELECT queries to fetch relevant product details, prices, and reviews from its vast product database.

2. **Order History:** Retrieves a user's past orders to display purchase history and track shipments.

3. **Inventory Management:** Monitors stock levels by selecting current inventory counts to prevent overselling.

Netflix

1. **Content Recommendations:** Uses SELECT to fetch personalized movie and show suggestions based on viewing history and preferences.

2. **User Watch History:** Retrieves the list of shows and movies a user has watched to allow resuming playback and providing recommendations.

3. **Trending Content:** Selects data on popular shows and movies to highlight trending content to users.

Banking Apps
1. **Account Balance & Transaction History:** When users check their account balance or view recent transactions, the app uses SELECT queries to retrieve this information from the banking database. 2. **Loan Details:** Fetches information about active loans, including amounts, interest rates, and due dates. 3. **Alerts & Notifications:** Selects data to trigger alerts for activities like large withdrawals or deposits.

- **What kind of data do they pull? How often?**

1. **Amazon:** Product details, user reviews, inventory levels, and order statuses are pulled in real-time as users interact with the platform.

2. **Netflix:** User preferences, watch history, and content metadata are retrieved continuously to provide seamless streaming experiences.

3. **Banking Apps:** Account balances, transaction records, and loan information are accessed on-demand whenever a user initiates a request through the app.

- **Where does DQL (Data Query Language) fit in daily operations?**

    1. **Real-Time Data Retrieval:** Ensures users receive up-to-date information instantly.

    2. **Personalization:** Fetches user-specific data to tailor experiences, such as recommendations or account details.

    3. **Operational Efficiency:** Supports backend processes like inventory checks, content management, and financial record-keeping.

- **In Spotify**, SELECT queries retrieve a user's listening history and preferences to generate personalized playlists and song recommendations, enhancing the user experience through tailored music suggestions.

3. How Scrolling Works Behind the Scenes

1. Is Scrolling Powered by SELECT Queries?

Yes. As you scroll, the app dynamically fetches new data from the database using SELECT queries. For instance:

```
SELECT * FROM products ORDER BY popularity DESC LIMIT 20 OFFSET 40;
```

This query retrieves the next set of products (e.g., items 41–60) to display as you continue scrolling.

---

2. Fetching More Data on Scroll

To load additional data when you scroll:

- **LIMIT and OFFSET:** Specify how many records to fetch and from where.

  ```
  SELECT * FROM items ORDER BY created_at DESC LIMIT 10 OFFSET
  30;
  ```

- **Cursor-Based Pagination:** Use a unique identifier (like id) to fetch records after a certain point, which can be more efficient.

  ```
  SELECT * FROM items WHERE id > last_seen_id ORDER BY id ASC
  LIMIT 10;
  ```

This method is particularly useful when dealing with real-time data or avoiding performance issues associated with large OFFSET values.

---

3. Pagination and Lazy Loading in SQL-Backed Apps

- **Pagination:** Divides content into discrete pages. Users navigate through pages, each fetching a specific subset of data.
- **Infinite Scrolling:** Automatically loads more content as the user scrolls down, providing a seamless browsing experience.
- **Lazy Loading:** Delays the loading of data until it's needed, reducing initial load time and conserving resources.

These techniques enhance user experience by ensuring that only necessary data is loaded, improving performance and responsiveness.

In Spotify, as you scroll through your music library or playlists, SELECT queries retrieve song information in batches. This approach ensures quick load times and a smooth user experience by fetching only the necessary data as needed.