Explanation of My SQL Code

1. Populating the `dim_users` table

```
INSERT INTO dim_users (user_id, email, name, is_customer)
SELECT id, email, name, is_customer FROM users;
```

- Here, I created the **User Dimension table** (`dim_users`) by directly pulling user attributes (`id`, `email`, `name`, `is_customer`) from the `users` source table.
- This allows normalized user information to be separated from the main fact table for better performance and scalability.

---

2. Populating the `dim_conversation_parts` table

```
INSERT INTO dim_conversation_parts (part_id, conversation_id,
part_type, created_at)
SELECT id, conversation_id, part_type, created_at FROM
conversation_parts;
```

- Similarly, I populated the **Conversation Parts Dimension table** (`dim_conversation_parts`) by extracting relevant fields from the `conversation_parts` table.
- This dimension captures additional metadata about each part of the conversation, which can be useful for analytics and reporting.

---

3. Populating the `consolidated_messages` fact table

Step 3.1 — Insert conversation starts (initial messages)

```
INSERT INTO consolidated_messages (...)
SELECT ...
FROM conversation_start cs
JOIN users u ON cs.conv_dataset_email = u.email
WHERE u.is_customer = 1
```

- I first inserted records from the `conversation_start` table where the **starter is a customer** (`is_customer = 1`).
- I manually assigned `message_type = 'open'` because conversation starts are the opening messages.
- The `user_id` was mapped from the `users` table based on the email address of the conversation starter.

Step 3.2 — Insert conversation parts (replies, updates, etc.)

```
UNION ALL
SELECT ...
FROM conversation_parts cp
JOIN conversation_start cs ON cp.conversation_id = cs.id
JOIN users u ON cs.conv_dataset_email = u.email
WHERE u.is_customer = 1;
```

- Then, I inserted records from the `conversation_parts` table.
- I only included parts of conversations **where the conversation was started by a customer** (based on the join condition).
- Here, the `message_type` was directly taken from the `part_type` column.
- `user_id` again maps to the customer who initiated the conversation.

---

4. Handling special cases — Starter is not a customer

```
INSERT INTO consolidated_messages (...)
SELECT ...
FROM conversation_start cs
JOIN conversation_parts cp ON cs.id = cp.conversation_id
JOIN users u ON cp.conv_dataset_email = u.email
WHERE u.is_customer = 1
AND cs.id NOT IN (SELECT conversation_id FROM consolidated_messages);
```

- Sometimes, a conversation may be initiated by an **agent or system** (non-customer).
- In such cases, I ensured the conversation is still captured by identifying the customer participant from the conversation parts.
- I added a **filter** to **avoid duplicate insertion** of conversations that were already handled earlier.

---

5. Sorting the `consolidated_messages` table

```
CREATE TABLE temp_consolidated_messages AS
SELECT * FROM consolidated_messages
ORDER BY conversation_id, created_at;

DROP TABLE consolidated_messages;
ALTER TABLE temp_consolidated_messages RENAME TO
consolidated_messages;
```

- SQLite doesn't support native table reordering.
- Therefore, I recreated the table by **ordering the messages by**:
    - `conversation_id`
    - `created_at`
- This ensures that all the messages flow chronologically within each conversation thread.
- Finally, I replaced the original table with the ordered version for cleaner data querying.
