Part 2: Functional Dependencies and Their Impact

Below is an overview of the functional dependencies for each table in the MovieMusicStore database along with a discussion on how these dependencies impact query performance and normalization decisions.

---

1. Customer

Functional Dependencies:

- **customer_id** → first_name, last_name, phone, email, join_date, status
- **email** → customer_id, first_name, last_name, join_date, status

Impact:

- **Normalization:**
  The Customer table is in Third Normal Form (3NF) because every non-key attribute is fully functionally dependent on the primary key (customer_id), and there are no transitive dependencies.
- **Query Performance:**
  - The UNIQUE constraint on email supports fast lookups.
  - The primary key on customer_id enables efficient joins with related tables (e.g., Order).

---

2. Order

Functional Dependencies:

- **order_id** → customer_id, order_date, order_status, total_amount

Impact:

- **Normalization:**
  Order is normalized (3NF) since all attributes depend on order_id.
- **Query Performance:**
  - The foreign key customer_id is indexed, speeding up joins with the Customer table.
  - Primary key indexing on order_id ensures fast access.

---

3. Payment

Functional Dependencies:

- **payment_id** → order_id, payment_date, payment_method, amount

Impact:

- **Normalization:**
  Payment is in 3NF.
- **Query Performance:**
  - The foreign key on order_id allows efficient joins with the Order table.
  - Proper indexing ensures that retrieving payments for an order is fast.

---

4. OrderItem

Functional Dependencies:

- **order_item_id** → order_id, product_type, product_id, quantity, price_each
- (Alternatively, the composite key of (order_id, product_id) could uniquely identify an item, depending on design.)

Impact:

- **Normalization:**
  OrderItem is normalized, as every attribute is fully dependent on its key.
- **Query Performance:**
  - The dependency on order_id ensures efficient retrieval of all items for a given order.
  - Indexes on the composite key (if used) speed up join operations with Order.

---

5. Movie

Functional Dependencies:

- **movie_id** → title, release_date, rating, genre, runtime_minutes, stock_count

Impact:

- **Normalization:**
  Movie is in 3NF.
- **Query Performance:**
  - The primary key (movie_id) supports efficient lookups.
  - However, if `genre` needs to store multiple values, further normalization (e.g., a separate Genre table) might be required, which could improve query flexibility but add join complexity.

---

6. Actor

Functional Dependencies:

- **actor_id** → first_name, last_name, birth_date

Impact:

- **Normalization:**
  Actor is fully normalized with no redundant data.
- **Query Performance:**
  - Fast lookups and efficient joins using actor_id as the primary key.

---

7. Movie_Actor

Functional Dependencies:

- **(movie_id, actor_id)** → role_name

Impact:

- **Normalization:**
  As an associative (junction) table, Movie_Actor is in 3NF.
- **Query Performance:**
  - The composite key enables quick retrieval of all actors for a movie and vice versa.
  - Efficient joins with both Movie and Actor tables due to indexing on the composite key.

---

8. MusicAlbum

Functional Dependencies:

- **album_id** → title, release_date, genre, stock_count

Impact:

- **Normalization:**
  MusicAlbum is in 3NF.
- **Query Performance:**
  - The primary key (album_id) ensures rapid lookups.
  - Similar considerations for `genre` apply as with the Movie table.

---

9. Artist

Functional Dependencies:

- **artist_id** → artist_name, start_year, end_year, active_status

Impact:

- **Normalization:**
  Artist is normalized.
- **Query Performance:**
  - Efficient joins using the primary key artist_id.
  - Indexing supports fast queries in association with Album_Artist.

---

10. Album_Artist

Functional Dependencies:

- **(album_id, artist_id)** → *(no additional attributes)*

Impact:

- **Normalization:**
  Album_Artist is a pure junction table and is in 3NF.
- **Query Performance:**
  - Supports efficient many-to-many relationships between MusicAlbum and Artist.
  - The composite key allows fast joins in queries linking albums to their artists.

Overall Impact on Query Performance and Normalization

- **Normalization Benefits:**
  By ensuring each table adheres to at least 3NF, the design minimizes redundancy, reduces update anomalies, and ensures data consistency.
- **Indexes:**
  Primary keys and foreign keys (derived from these functional dependencies) naturally create indexes, which are crucial for fast join operations and quick data retrieval.
- **Query Complexity:**
  Normalization can sometimes require more complex queries (with additional joins), but in this design, the benefits of data integrity and reduced redundancy outweigh the potential performance cost—especially when proper indexing is applied.
- **Denormalization Considerations:**
  For reporting or performance tuning in a large-scale system, you might consider denormalized views or summary tables. However, for this project, the fully normalized design ensures maintainability and consistency.

This analysis of functional dependencies and their impact helps guide optimization decisions for both query performance and database design.