1. **Indexing for Faster Queries**

- Unique IDs (`programID`, `concertID`) are used as indexes.
- Speeds up data retrieval and eliminates full-table scans.

2. **Data Normalization**
   - JSON is split into structured tables (`concerts.txt`, `works.txt`, etc.).
   - Reduces redundancy and improves query efficiency.
3. **Streaming Processing**
   - Processes data line-by-line instead of loading full JSON.
   - Reduces memory usage and improves scalability.
4. **Set-Based Deduplication**
   - Uses Python sets to avoid duplicate inserts.
   - Ensures fast and optimized joins.
5. **Optimized Query Execution Order**
   - Extracts `concerts` first, then `works`, then `soloists`.
   - Minimizes slow joins and dependencies.
6. **Indexing Text-Based Lookups**
   - Standardizes text fields (`lower()`, `replace(" ", "_")`) for faster searches.
   - Allows binary search on sorted data.

Future Enhancements

- **Database Indexing**: B-Trees, Hash Indexes for speed, BitMap.

- **Partitioning Large Data**: Split by year or venue. -

- **Query Caching**: Store frequent queries for faster response.

Conclusion:
These techniques significantly improve data retrieval speed, memory efficiency, and scalability.