ORM Data Sync Process: MySQL → SQLite

Scenario

Your team maintains Sakila (operational, MySQL). You need a lightweight analytics database (SQLite) fed by an ORM-only data sync process. No raw SQL for reads/writes (allowed only if your ORM requires it).

Objectives

- Model Sakila entities in an ORM and read from MySQL.
- Design an analytics schema in SQLite and write via ORM.
- Implement initial full load + incremental updates using timestamps.

Target analytics schema (minimum)

Use ORM models to create these tables in SQLite. Each table below includes a short description and an example row.

Dimensions

dim_date

Stores calendar and derived date attributes for time-based aggregation.

date_key	date	year	quarter	month	day_of_month	day_of_week	is_weekend
20060214	2006-02-14	2006	1	2	14	2	0

dim_film

Represents individual films with descriptive attributes.

film_key	film_id	title	rating	length	language	release_year	last_update
25801	258	CHICAGO NORTH	PG	107	English	2005	2006-02-15

dim_actor

Contains actor details used for film-level analysis.

actor_key	actor_id	first_name	last_name	last_update
50101	101	JENNIFER	DAVIS	2006-02-15

dim_category

Stores film categories or genres.

category_key	category_id	name	last_update
30101	10	DRAMA	2006-02-15

`dim_store`

Holds store-level location details to analyze performance by geography.

store_key	store_id	city	country	last_update
1001	1	Lethbridge	Canada	2006-02-15

`dim_customer`

Captures customer attributes to support customer-based analytics.

customer_key	customer_id	first_name	last_name	active	city	country	last_update
34101	341	BRENDA	BROWN	1	Dallas	USA	2006-02-15

Bridges (for many-to-many)

`bridge_film_actor`

Links films and actors to support analysis of actor participation.

film_key	actor_key
25801	50101

`bridge_film_category`

Links films and categories to support genre-level insights.

film_key	category_key
25801	30101

Facts

`fact_rental`

Records each rental event. Enables analysis of rental behavior, duration, and store activity.

fact_rental_key	rental_id	date_key_rented	date_key_returned	film_key	store_key	customer_key	staff_id	rental_duration_days
50001	16050	20060214	20060219	25801	1001	34101	2	5

`fact_payment`

Tracks payment transactions and supports revenue analysis.

fact_payment_key	payment_id	date_key_paid	customer_key	store_key	staff_id	amount
80001	17503	20060219	34101	1001	2	6.99

Notes:

- When data changes in the source (for example, a customer’s city or a film’s rating is updated), just overwrite the existing record in the corresponding dimension table instead of keeping old versions. This keeps each dimension table always showing the latest information from Sakila.
- Generate `date_key` (e.g., `YYYYMMDD`) from timestamps in rentals/payments.

- The natural key (*_id) — the original identifier from Sakila (e.g., film_id = 258)
- The surrogate key (*_key) — a new, auto-generated ID you assign in SQLite (e.g., film_key = 25801).

Required Data Sync Process Features

1) Connections & models

- Define ORM models for the subset of Sakila you need: `film, language, actor, category, film_actor, inventory, rental, payment, store, staff, customer, address, city, country`.
- Define ORM models for all analytics tables above in SQLite.
- Use two ORM sessions/contexts: source (MySQL) and target (SQLite).

2) Initial full load

- Load all rows from the source Sakila tables into the corresponding SQLite analytics tables using your ORM. Make sure to include all necessary data for dimensions, bridges, and facts so that the analytics database contains a complete copy for reporting. If necessary, use ORM transactions so that if anything fails, the database stays consistent.

3) Incremental updates (rerunnable job)

- Use Sakila's `last_update` field or timestamps such as `rental_date`, `return_date`, or `payment_date` to detect which records have changed since the last sync. Treat these as last updated markers.
- Create and maintain a `sync_state` table in SQLite to record the most recent update time for each table you process.
- Each time you run your data sync job:
  - Read the last updated marker for each table from `sync_state`.
  - Query only the rows from Sakila that were added or changed since that time.
  - Update existing records in SQLite if the source data has changed.
  - Insert any new rows that are not yet in SQLite.
  - Finally, update the `sync_state` table to save the new last updated marker.
- This approach allows your sync process to run repeatedly, keeping the SQLite analytics database synchronized with the latest Sakila data without reloading everything each time.

4) Performance & correctness

- Create useful indexes in SQLite.
- Add data validation: after each sync run, compare record counts and key totals (for example, total number of rentals or total payment amounts per store) between the

source and target. Any large difference should raise a warning or error so you can verify data consistency.

5) CLI or script interface

Provide the following commands. Each should return or print human-readable message on failure or success.

- `Init` — Set up the analytics database
  Initializes the SQLite database and creates all analytics tables using ORM models. It verifies connections to MySQL and prepares any necessary structures like `dim_date` and `sync_state`.

- `Full-load` — Load all source data
  Performs a complete import from Sakila into SQLite. It reads all data from MySQL tables, transforms them as needed, and loads them into the SQLite analytics tables. This is typically run once at the start or when rebuilding the analytics database.

- `Incremental` — Load only new or changed data
  Performs an update based on what has changed in Sakila since the last sync. Uses timestamps (`last_update`, `rental_date`, etc.) to find new or modified rows and updates SQLite accordingly. It is meant to run regularly to keep the analytics database up to date.

- `Validate` — Verify data consistency
  Compares counts and totals between MySQL and SQLite over a selected period (default: last 30 days). It ensures there are no missing rows, duplicates, or inconsistencies. Use this to confirm that data is correctly synchronized.

6) Tests

Each test should verify a key function of the CLI commands. Keep tests short and focused.

1. Init command — Confirms database and tables are created successfully.
2. Full-load command — Verifies all data from Sakila is loaded into SQLite.
3. Incremental command (new data) — Checks that new records added in Sakila appear correctly in SQLite.
4. Incremental command (updates) — Ensures existing rows are updated when source data changes.
5. Validate command — Confirms data consistency between MySQL and SQLite.

Deliverables

Submit a repository that clearly demonstrates your ORM data sync process:

- Readme — Setup instructions, how to run each CLI command, any environment variables, and schema diagrams.
- ORM models — For both MySQL (Sakila) and SQLite analytics database.
- Sync process code — Implementation of all four CLI commands with logging, transactions, and error handling.

Submit a Word document that clearly demonstrates your testing:

- Sample output — Screenshots and logs of all five tests