Simplify streaming application development

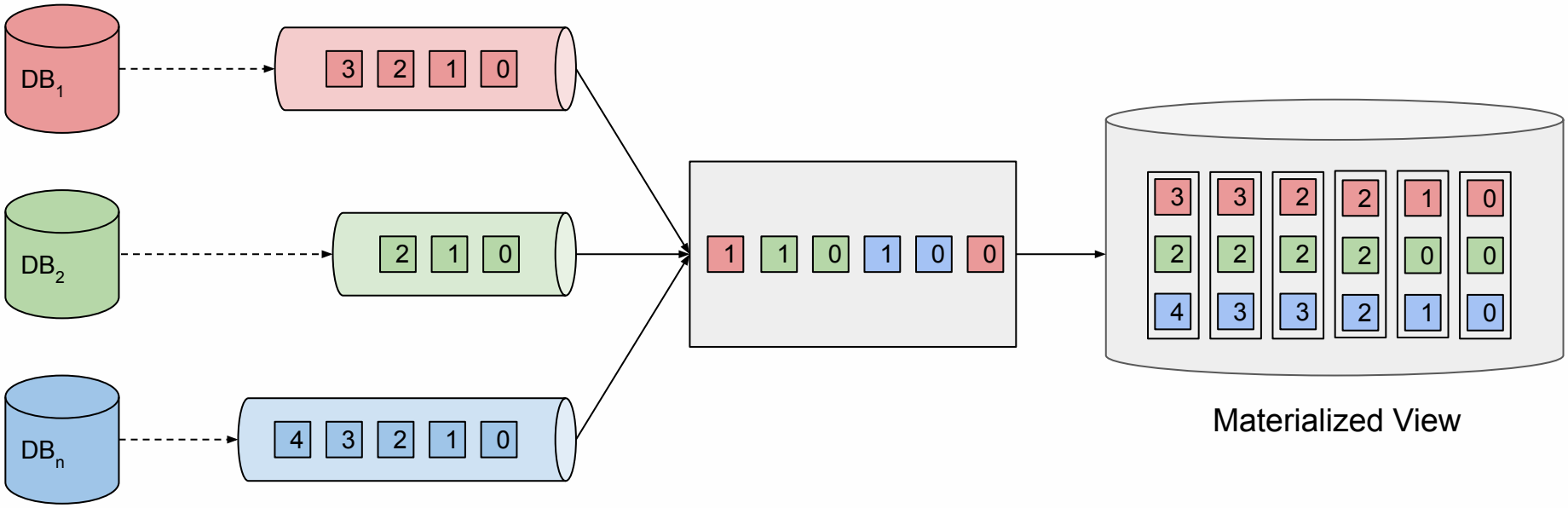
A declarative approach at Airbnb

Xiangmin Liang / 09/05/2024 / StaffPlus New York

The needs of streaming applications are rising

But its development can be complicated

A Popular Streaming Use Case
-- Real-time Materialized View



Database

Change Logs

Stream of changes affecting a single materialized view

Read-optimized Store

```
public class UserSource {
    private Long id;
}
```

```
public class ReviewSource {
    private Long id;
}
```

```
public class ListingSource {
    private Long id;
    private String roomType;
    private Boolean hasAvailability;
    ...
}
```

Model Source

```
public class UserToListingJoin(
  UserSource user,
)
```

```
public class ReviewToUserJoin(
  UserSource user,
  ReviewSource review
) {
    return JoinCondition.builder()
      .join(review)
      .on(review
          .getField(R.USER_ID)
          .eq(user.getField(U.ID)))
}
```

Manage Source Relations

```
public class ReviewByListingStitchModel {
    private Long listingId;
    private String roomType;
    private List<ListingReview> reviews;
    ...
    public static class ListingReview {
      private Long reviewId;
      private ReviewUser reviewer;
      ...
    }
    public static class ReviewUser {
      ...
    }
}
```

Model Output

```
public class ReviewByListingTransformation {
    ...
    public OperatorChain getOperationChain() {
      return OperatorChain
        .<ReviewByListingModel>create()
        .filter( v -> Objects.nonNull(v.id()))
        .filter(
          v -> !Objects.equals("HOST", v.role()))
        .map(DOCUMENT_MAPPER)
        .build();
  }
}
```

Implement Transformation

```
public class ReviewByListingPipelineBase {
  public static Pipeline create(...) {
    JoinCondition1 = userToListingJoin = ...
    JoinCondition2 = reviewToUserJoin = ...
    return pipeline
        .enrich(listing,UserSource.ID)
        .join(userToListingJoin)
        .join(reviewToUserJoin)
        .groupBy(listing)
        .stitch(ListingByUserModel.class);
  }
}
```

Construct Pipeline

```
public class ReviewByListingSourceRepository {
  private DataSourceConfig config;
  private UserSource user;
  private ListingSource listing;
  private ReviewSource review;

  public ReviewByListingSourceRepository(config) {
    this.config = config;
    this.user =
      new UserSource(config.getUserStoreConfig());
    ...
  }
}
```

Manage Source Configs

It can actually be as easy as

Writing a GraphQL Query

And the rest will be taken care of automatically

```
{
  listing {
    id @materializedViewId
    roomType
    hasAvailability
    ...
    review
    @collection(type: "SET")
    @staticFilter(field: "isHidden", operator: EQUALS, value: false)
    @staticFilter(field: "revieweeRole", operator: EQUALS, value: "guest")
    {
        id
        revieweeRole
        isHidden
        ...
        reviewerDetails: user
        @staticFilter(field: "country", operator: EQUALS, value: "US")
        {
            id
            firstName
            lastName
        }
    }
  }
}
```

A Declarative Approach with GraphQL

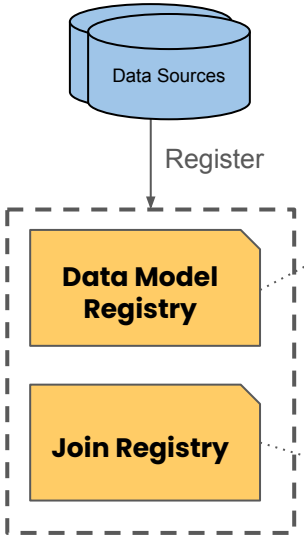
Why GraphQL?

- Widely used within the product teams in Airbnb
- Very similar experience to define a streaming pipeline as writing GraphQL queries to fetch data
- Unified schema-driven development

GraphQL Schema

Data source registries in GraphQL Schema

- GraphQL Object type for data sources
- GraphQL extend type for join relations




```
type ReviewModel
 @kvStore
 @materializedViewSource(
   table: "TABLE_NAME",
   cdcTopic: "KAFKA_TOPIC",
   kafkaBusConfig: "KAFKA_BUS"
 )
{
  fields  ...
}
```

```
extend type ReviewModel {
  user: UserModel @joinCondition(
    sourceField: "reviewUserId",
    joinField: "id"
  )
}
```

GraphQL Query

```
{
  user {
    id @materializedViewId
    firstName
    lastName
    ...
    review
    @collection(type: "SET")
    @staticFilter(field: "isHidden", operator: EQUALS, value: false)
    @staticFilter(field: "revieweeRole", operator: EQUALS, value: "guest")
    {
        id
        revieweeId
        revieweeRole
        isHidden
        ...
    }
  }
}
```

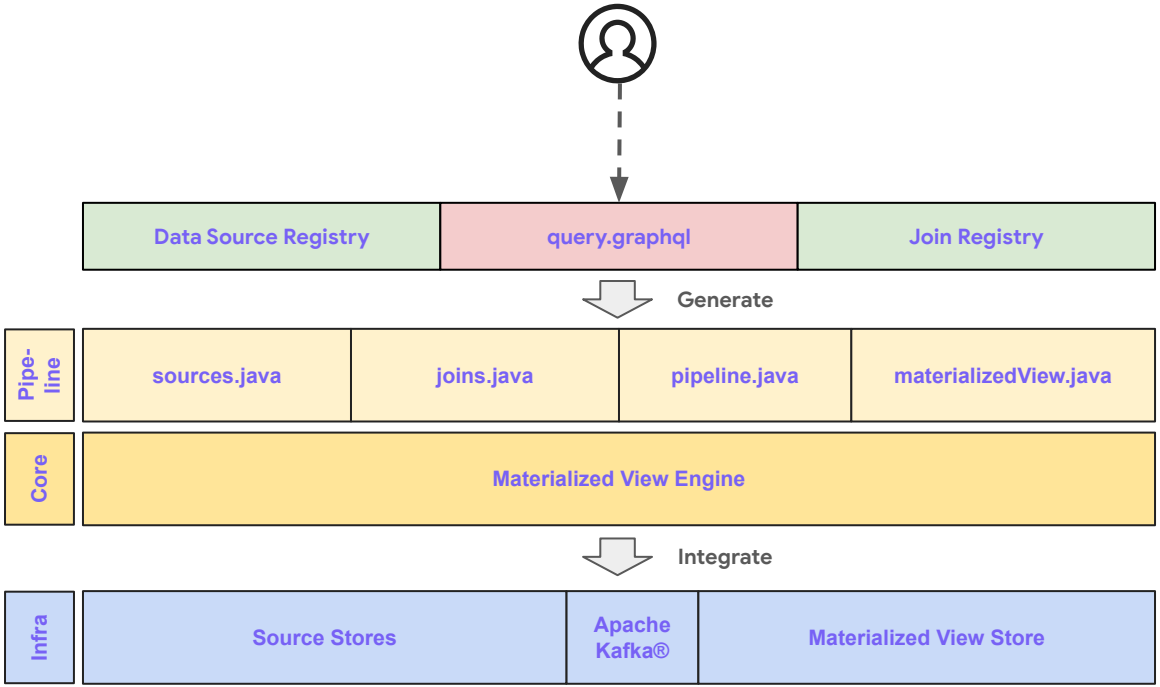
Customer interface to define the pipeline

Simple GraphQL syntax

Rich directives to support various source operations

Pipeline Generator

Model generator to automatically generate pojos necessary for the Materialized View Engine to integrate



Adoptions

20+ Teams using this approach

50%+ Code replaced by GraphQL query files

30+ Materialized views serve in production

Conclusion

- Find the balance between flexibility and ease of use
- Abstract away the challenges and complexities of building streaming applications from the business logic

Thank you.