元数据管理+Azkaban调度自动化方案详细设计

一、方案背景

a：现阶段我们数仓的架构层数少，贴源层和数据明细层是增量导入和标准ETL，处理逻辑比较单一，大部分都是可实现自动化的重复工作；

b：目前贴源层表处于手动维护的状态，且业务初期，各业务系统的数据表变化比较频繁，手动维护成本太高，且容易出错；

c：现阶段三大业务共1000+张表，后期接入更多的表后将难以管理。所以迫切需要将自动化操作纳入我们的数仓管理系统中；

二、方案目的

a：实现自动监控和同步MySQL元数据信息到Hive数仓，保证每天的数据都是最新的，提升数据质量，降低出错率，提升效率；

b：实现批量自动化的MySQL增量导入到HDFS的Parquet文件任务；

c：实现批量自动化的Hive贴源层加载增量数据，完成贴源层到数据明细层的ETL清洗；

d：实现批量自动化的Hive数据明细层拉链表开发；

e：实现Azkaban调度自动化，自动添加新任务和删除旧任务，自动重试；

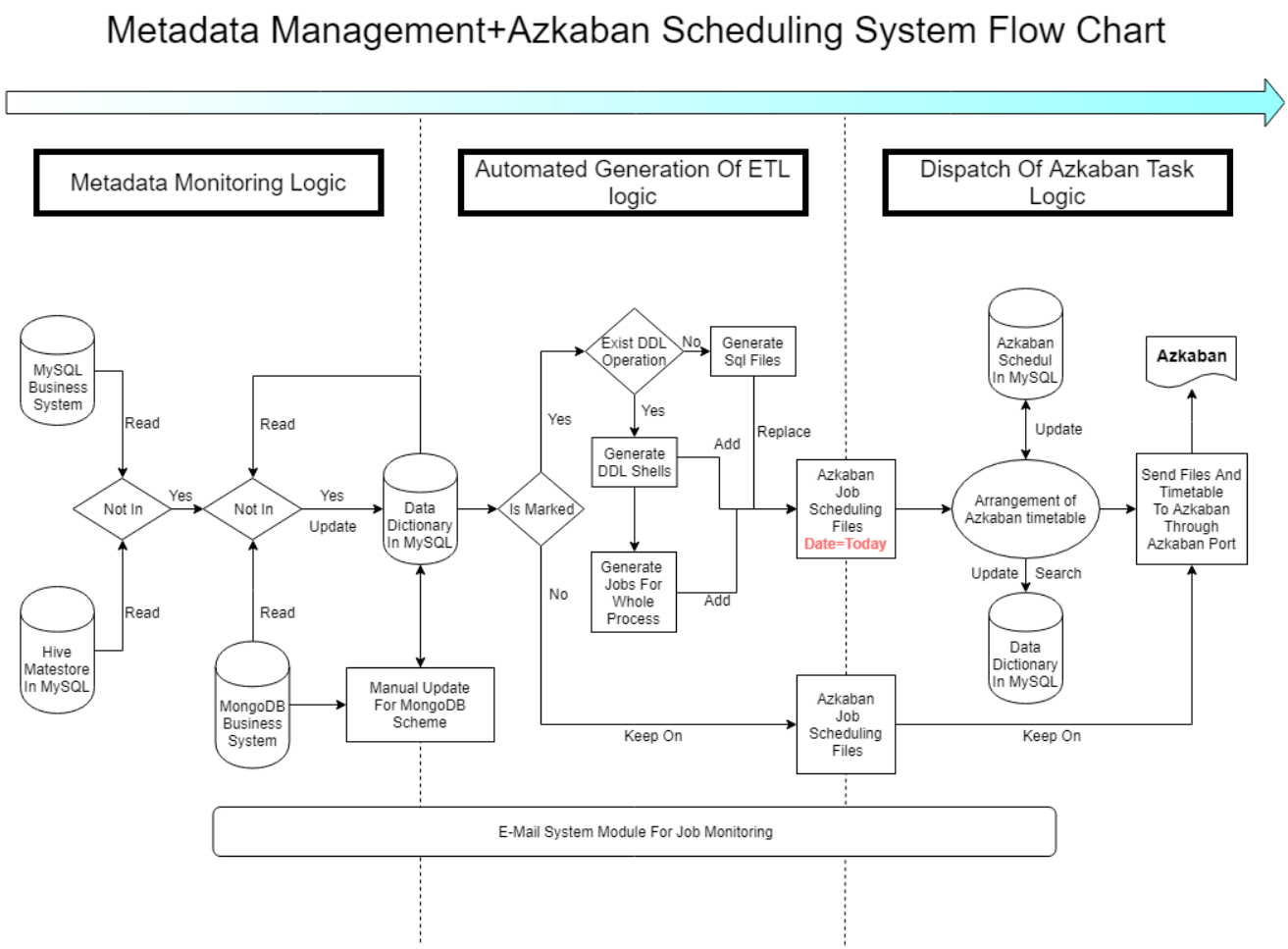
f：实现绝大部分自动化和极少的人工维护，实现外部人工调控和内部代码自动化的有机结合，数仓维护更加定制化；

g：实现了对一般错误、异常有一定的容纳程度，实现每个模块出现失败自动重跑；

h：实现了保证每个模块的独立工作互不干扰，实现统一监控和邮件告警；

三、总体架构

3.1、元数据管理+Azkaban调度自动化方案流程图



3.2、方案开发策略

a：Java实现主要逻辑、Python实现脚本批量生成、JavaScript和Ajax实现Azkaban数据交互、Shell实现服务器进程操作和任务文件的发送拷贝；

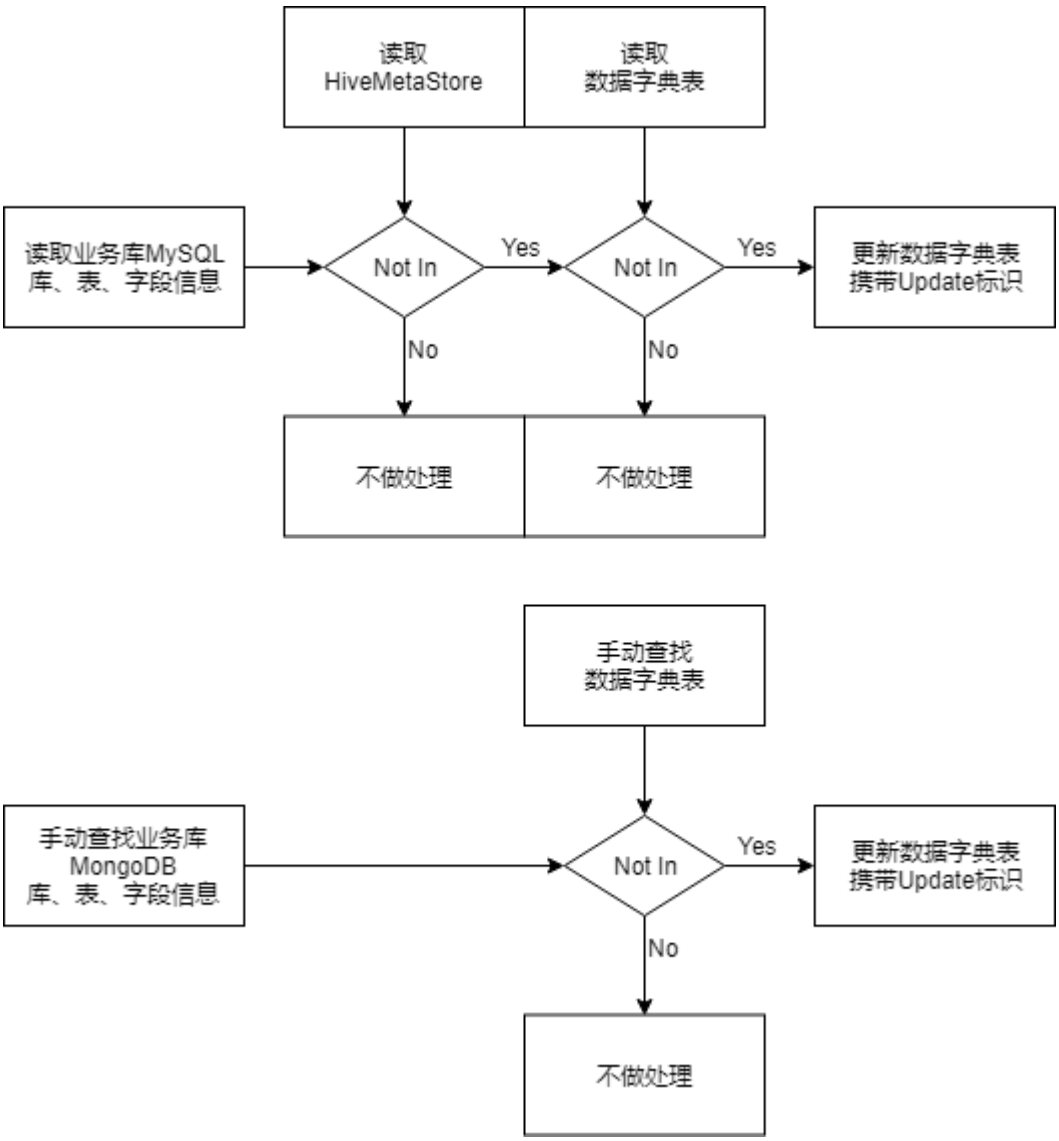
b：将该系统放到每日的Azkaban任务调度中，第一批次执行，由该系统生成后续的Azkaban任务；

c：由监控模块去监控日志的报错同时负责重跑任务，并监控上下游数据的一致性，同时负责记录生成报告发送邮件通知

d：采用迭代开发；

四、元数据监控模块

4.1、元数据自动化管理模块逻辑图



4.2、元数据自动化管理模块逻辑阐述

a：对比MySQL业务系统元数据和Hive元数据信息，将不存在Hive元数据中的库、表、字段（且不存在于数据字典的值，因为存在于数据字典却不在Hive元数据库的值是不需要存数仓的）添加到数据字典中，数据字典存放所有数据字段信息，并打上不同的功能标签；

b：在这个环节，数据字典的作用存储和记录元数据变动的情况，自动化标识Update标签的情况下，同时接受手动标识LoadToHive标签，双重维护，使得数仓开发更加定制化；由于MongoDB不支持查询元数据信息，需要手动维护MongoDB的元数据在数据字典表中，将在以后改进；

c：考虑到将来对接的数据库会有多个，将采用读取外置配置文件的方式，添加业务库信息，提升代码复用性和元数据集中管理；

d：仓库对于业务系统表的改动，采用"你加我就加，你删我不删"的原则，又因为ods层和业务系统的表结构保持高度一致，所以要以源数据表为基准作表和字段的比较；

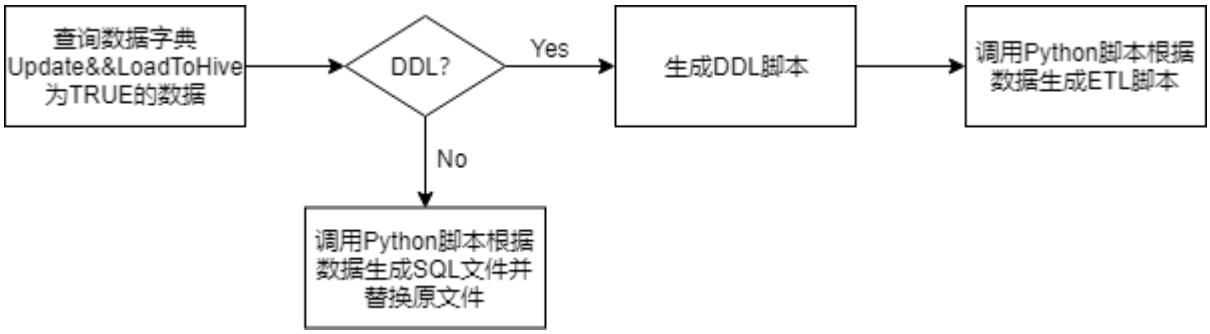
e：相对于既有比对逻辑，元数据信息不再只和Hive Metastore作比对，而是采用了双重比对逻辑，和数据字典做比对，数据字典存储各业务库的元数据。避免不必要更新的库和表对之后的操作产生影响。

f：该模块只做数据字典表的同步，实现了多个数据来源的统一元数据管理，极大的方便了以后对接更多的业务库。

4.3、代码结构

五、自动化生成ETL脚本模块

5.1、自动化生成ETL脚本模块逻辑图



5.2、自动化生成ETL脚本模块逻辑阐述

a：在这个模块中，查询数据字典中标识Update和LoadToHive为TRUE库和表，调用Python脚本，批量生成SqoopImport脚本和MongoExport脚本，批量生成贴源层到明细层的ETL脚本，批量生成数据明细层的拉链表更新脚本；

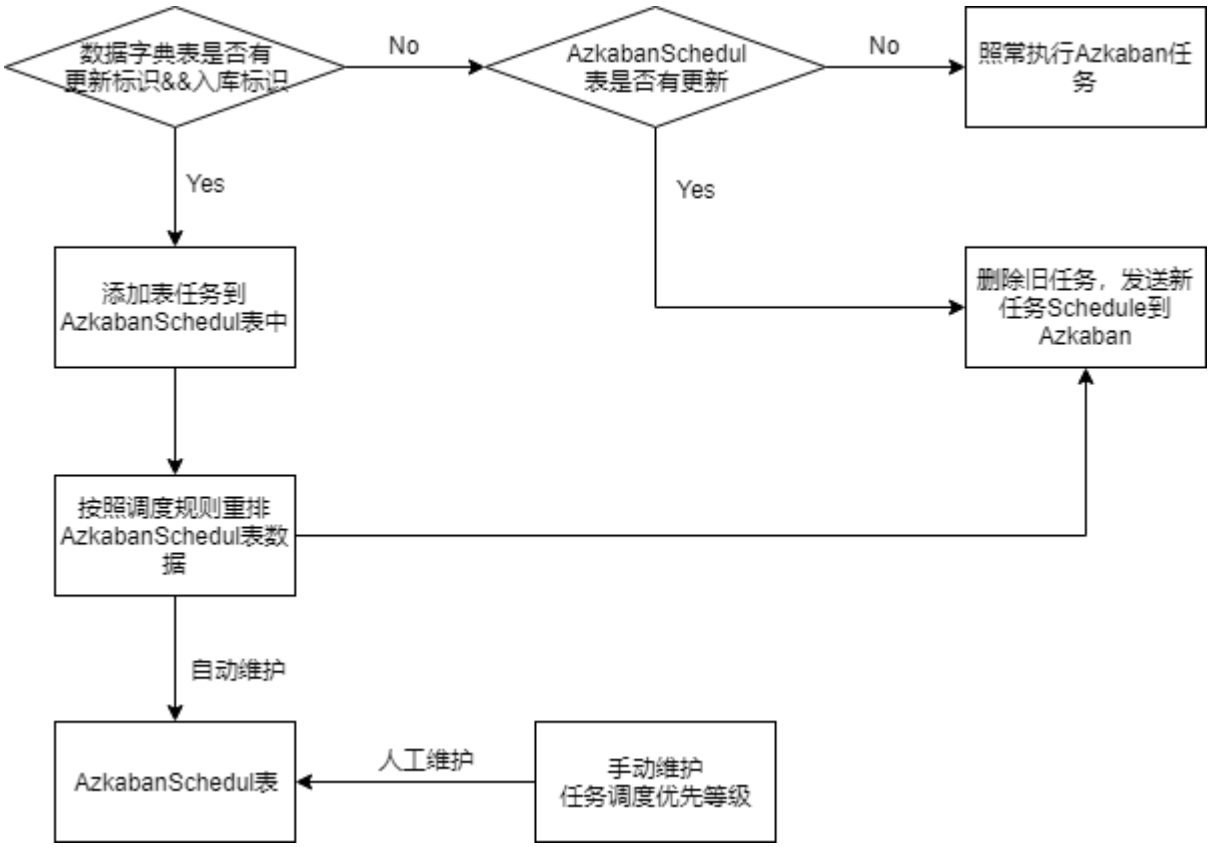
b：将DDL操作和DML操作分离，DDL操作为添加脚本到日期为今天的目录，并生成新表的任务流脚本；DDL操作为更新SQL文件，如果数据字典中没有需要更新的标识，那么我们将保持所有脚本的状态；

c：将生成的文件发送到Azkaban服务器上并更新历史脚本；

5.3、代码结构

六、Azkaban任务分配模块

6.1、Azkaban任务分配模块逻辑图



6.2、Azkaban任务分配模块逻辑阐述

a：对于新增的调度任务，需要添加到原有任务体系中，此时需要重新制定Azkaban的执行计划，所以维护了一张AzkabanSchedul表用于记录调度计划；

b：将数据字典表中需要更新的表添加到AzkabanSchedul表中，根据排序规则，进行重新排序，将任务计划发送给Azkaban，并将数据字典表中的更新标识去掉；

c：设置Azkaban任务执行的重试次数，检测到执行失败后在AzkabanSchedul表中标记任务失败；

d：满足手动定义任务调度的优先等级设定，使得Azkaban调度流程更加规范化，更加智能；

6.3、代码结构

七、数据库设计---DataDictionary表设计

7.1、表需求阐述

数据字典表用于记录所有业务系统的元数据信息，便于我们大数据人员维护和数据数仓入库自动化，需要业务系统的信息和其包含的库、表、表字段、主键、索引、以及每个字段数据类型的信息，另外为了便于数仓入库，我们需要维护Update字段和Load2Hive字段。

7.2、表结构设计

TableSchemeDesign: DataDictionary									
ParentTable: data_dictionary									
Column	id	business_source	db_source	database	table	amount	comment	last_update	load2hive
DataType	int	string	string	string	string	bigint	text	date	boolean(default 1)
Comment	PrimaryKey	BusinessName	DBName	DatabaseName	TableName	TotalRecords	Comment	MarkByCode	MarkByArtificial
ChildTable: data_scheme_detail									
Column	id	database	table	character	datatype	is_column_key	is_create_time	is_update_time	update_time
DataType	int	string	string	string	string	boolean(default	boolean(default	boolean(default	date
Comment	ForeignKey	DatabaseName	TableName	Columns	DataType	Mark	Mark	Mark	UpdateDate
ChildTable: data_update_records									
Column	id	table	increase_num	export_num	update_status	update_date			
DataType	int	string	bigint	bigint	boolean	date			
Comment	ForeignKey	TableName	IncreasedNum	ExportNum	UpdateStatus	UpdateDate			

八、数据库设计---AzkabanSchedul表设计

8.1、表需求设计

Azkaban调度表主要用于规划调度顺序，实现新任务插入后根据Azkaban调度方案进行动态调整，便于大数据人员直观查看任务详情和后期统计，需要任务名、表名、业务名称、数据来源、脚本存档地址、脚本调用方式，任务流各个层级，今日任务是否执行成功，任务流执行断点，便于后期任务重跑，需要根据人工维护的字段是任务调度的优先等级。

8.2、表结构设计

TableSchemeDesign: AzkabanSchedul									
ParentTable: azkaban_schedul									
Column	id	business_source	db_source	database	table	task	comment	increase_num	level
DataType	int	string	string	string	string	string	text	bigint	int(default 0)
Comment	PrimaryKey	BusinessName	DBName	DatabaseName	TableName	TaskName	Comment	IncreasedNum	MarkByArtificial
ChildTable: azkaban_task_detail									
Column	id	task	job	comment	update_date				
DataType	int	string	string	text	date				
Comment	ForeignKey	TaskName	JobName	Comment	UpdateDate				

九、重跑机制（升级模块）

9.1、重跑方式

采用检查日志的方式进行失败模块定位，触发重跑机制；

重跑次数为1，重跑如果仍然失败则执行下一个模块，使用历史表和历史文件执行任务，优先保证大部分数据的更新，对于失败的新表任务则采用人工的方式执行；

9.2、失败判定标准

日志中采用：“时间字段+模块名+模块执行任务状态” 作为判别标准，如果模块执行任务状态出现ERROR，则停止该模块，并触发该模块以及后续模块的重跑机制。

9.3、正常重跑

正常重跑是需要做数据备份或者数据源有问题产生的重跑；不是因为环境异常，也不是由于其他异常导致的；

恢复全部表结构与文件，删除已经入库的数据分区，将其置于前一天的状态，然后重新启动 ”元数据管理+Azkaban调度系统“ 的Azkaban任务；

9.4、失败重跑

9.4.1、元数据同步失败

a：如果日志报错发生在元数据同步模块，那首先需要恢复数据字典表原状态，再重新进行同步；

b：需要维护数据字典的历史表，维护时长暂定一周；

c：元数据同步失败重跑后需要同时启动后续模块的执行；

9.4.2、自动化脚本生成失败

a：自动化脚本生成失败则删除当日发生更新的任务文件；

b：不需要重跑元数据同步模块；

c：自动化脚本生成模块失败重跑后需要同时启动后续模块的执行；

9.4.3、Azkaban任务更新失败

a：日志报错发生在Azkaban任务更新的模块，则需要先恢复AzkabanSchedul表的数据，再重新执行该模块；

b：需要维护AzkabanSchedul表的历史表，维护时长暂定一周；

c：失败重跑后无后续模块；

十、任务监控---邮件通知系统模块

10.1、监控内容

该模块主要为了通知数仓人员元数据变更操作、任务的进度、以及任务失败告警，统一信息通知渠道，后期维护；

监控内容分为X部分：

a：三个模块的日志文件(日志统一存放)；

b：Azkaban日志；

c：数据字典表和AzkabanSchedul表；

d：MySQL采集条数和入库条数；

e：……

10.2、通知机制

a：当模块执行报错，以及重跑报错都将触发邮件告警；

b：Azkaban报错告警；

c：数据字典表和AzkabanSchedul表发生更新通知；

d：MySQL采集条数和入库条数通知；