数据仓库建模

鲲鹏创新实践课：鲲鹏应用数据分析与管理实战

前言

- 数据仓库是为了便于多维分析和多角度展现而将数据按特定的模式进行存储所建立起来的数据库，它的数据基于OLTP源系统。
- 数据仓库能够对多个异构的数据源进行有效集成，集成后按照主题进行重组，存放大量历史数据，用于支持决策，面向分析型数据处理，它不同于企业现有的操作型数据库。
- 本章主要讲述数据仓库的相关概念及其相关建模流程，并介绍了分布式数据库中数据库对象的管理。

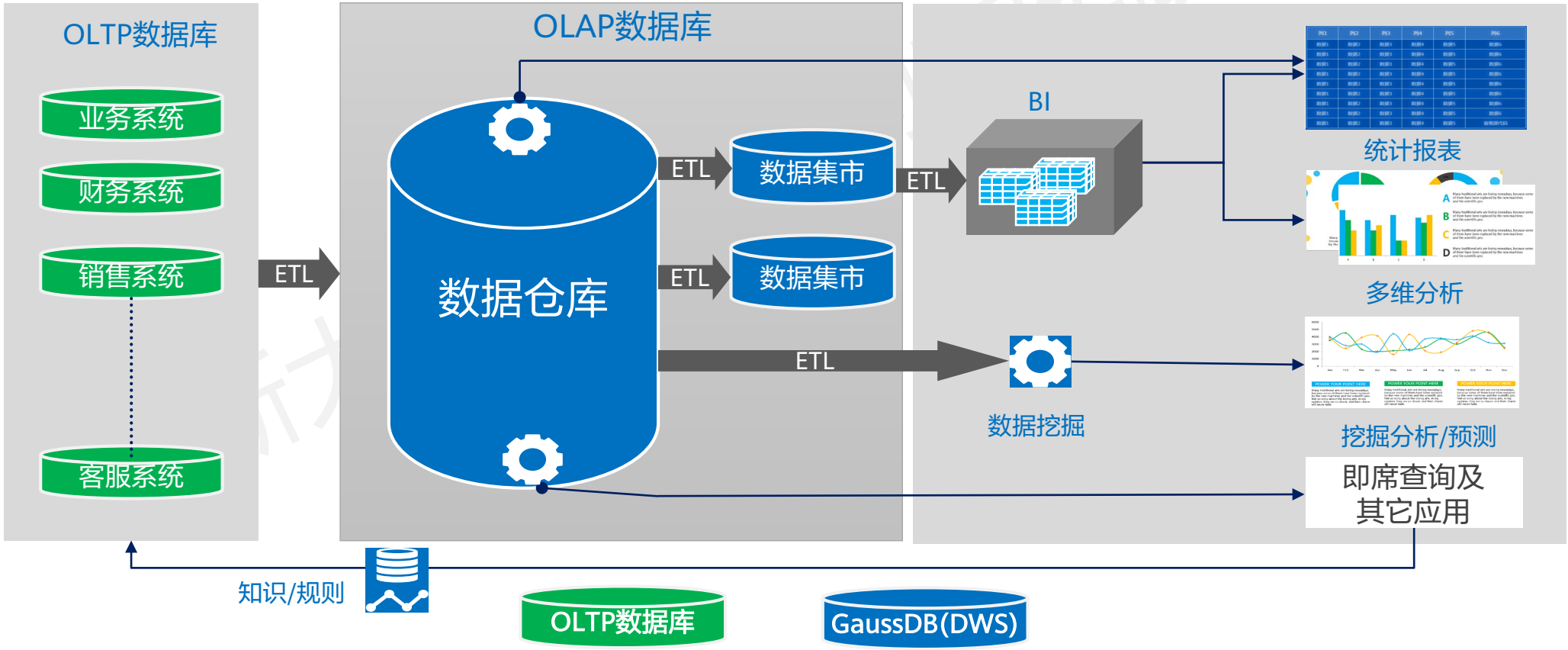
目标

- 学完本课程后，您将能够：
  - 掌握数据仓库、维度建模相关概念；
  - 了解经典的数据仓库架构；
  - 掌握维度建模流程；
  - 掌握分布式数据库中数据库对象的管理。

目录

典型的企业OLTP和OLAP数据库

- 联机事务处理(OLTP)：存储/查询业务应用中活动的数据以支撑日常的业务活动；
- 联机分析处理(OLAP)：存储历史数据以支撑复杂的分析操作，侧重决策支持。



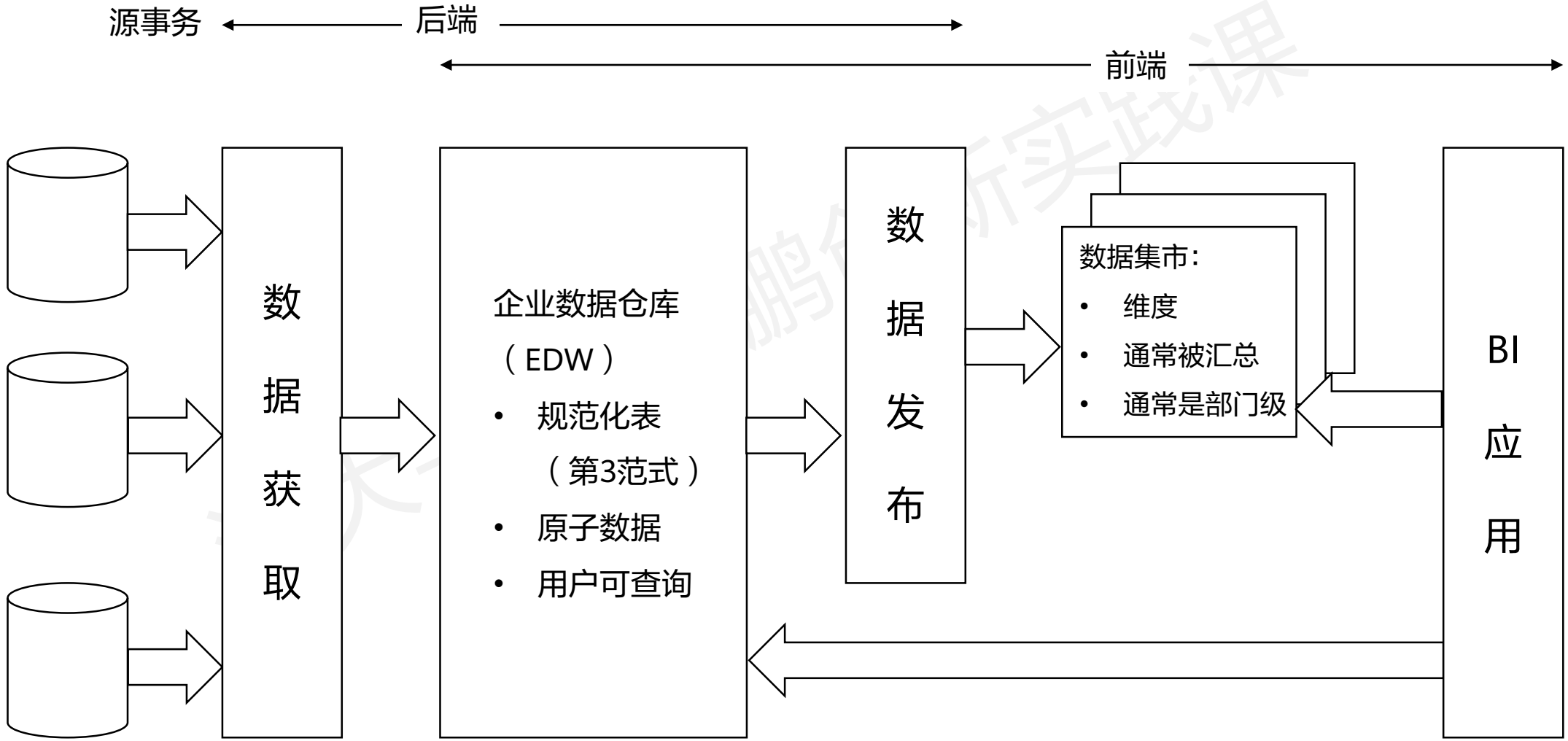
OLTP和OLAP对比

数据处理类型	OLTP	OLAP
分析粒度	细节的	细节的，综合的或提炼的
时效性	在存取瞬间是准确的	代表过去的数据
可更新性	可更新	不更新
操作可预知性	操作需求事先可知道	操作需求事先可能不知道
实时性	对性能要求高，响应毫秒级、秒级	对性能要求相对宽松，响应分钟级、小时级
数据量	一个时刻操作一条或几条记录，数据量小	一个时刻操作一集合，数据量大
驱动方式	事务驱动	分析驱动
应用类型	面向应用	面向分析
应用场景	支持日常运营	支持管理需求
典型应用	银行核心系统、秒杀系统	ACRM、风险管理

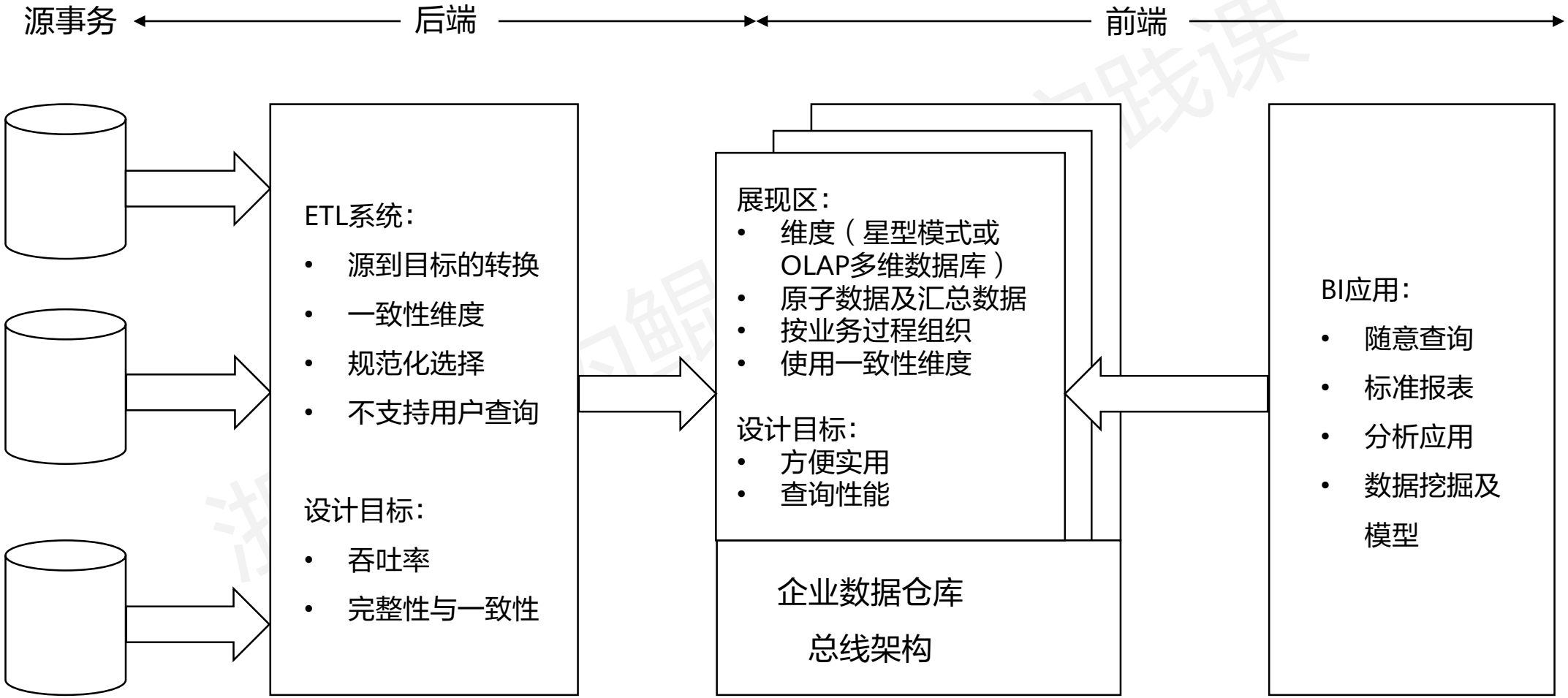
相关概念

- 数据仓库 (DW: Data Warehouse)
  - 1990s：数据仓库是一个面向主题的(Subject Oriented)、集成的(Integrated)、数据不易丢失的(Non-Volatile)、反应历史变化(Time Variant)的数据集合，用于支持管理决策(Decision Making Support))。——Bill Inmon（数据仓库之父）
- 商业智能 (BI: Business Intelligence)
  - 又称商业智慧或商务智能，指用现代数据仓库技术、线上分析处理技术、数据挖掘和数据展现技术进行数据分析以实现商业价值。
- ETL（Extract-Transform-Load）
  - 用来描述将数据从来源端经过萃取（extract）、转置（transform）、加载（load）至目的端的过程。ETL一词较常用在数据仓库，但其对象并不限于数据仓库。

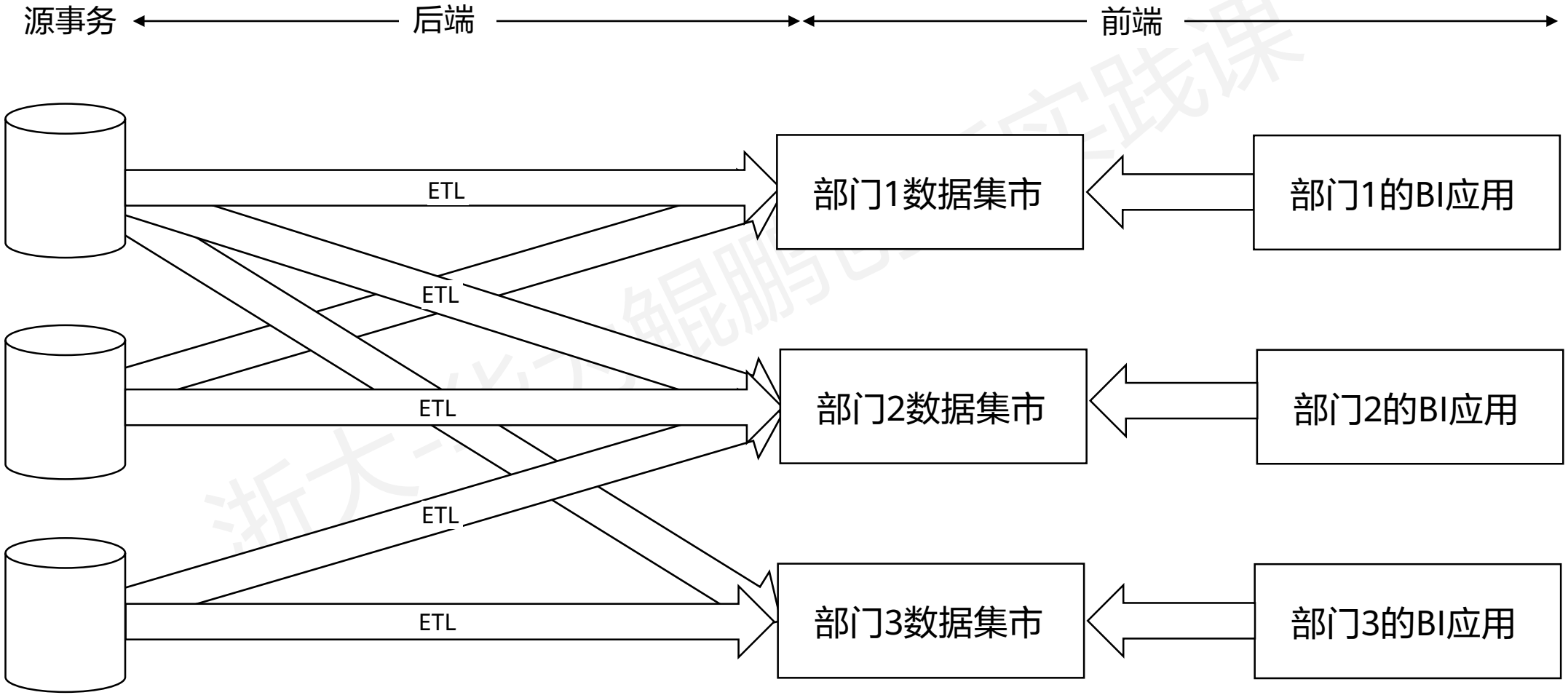
数据仓库架构 - 辐射状企业信息工厂Inmon架构



数据仓库架构 - Kimball的DW/BI架构



数据仓库架构 - 独立数据集市架构



思考题

1. 请大家对刚刚提到的几种架构做一个总结并谈谈对其应用场景的理解。

目录

维度建模 (1)

- 第三范式
  - 规范化的3NF将数据划分为多个不同的实体，每个实体构成一个关系表, 主要是为消除冗余。
  - 范式化建模能够提高数据一致性，满足比较高的质量要求，但把结构拆的很分散，使用数据的时候要把各实体中的数据关联起来，才能给最终用户展现一个完整的统计结果或者报表。
  - 在实际项目中，数据仓库的集市层数据基本上不会采用第三范式，而是采用维度建模。
  - 对BI查询来说， 规范化模型太复杂。

注意：维度模型包含的信息与规范化模型包含的信息相同，但将数据以一种用户可理解的、满足查询性能要求的、灵活多变的方式进行了包装。

维度建模 (2)

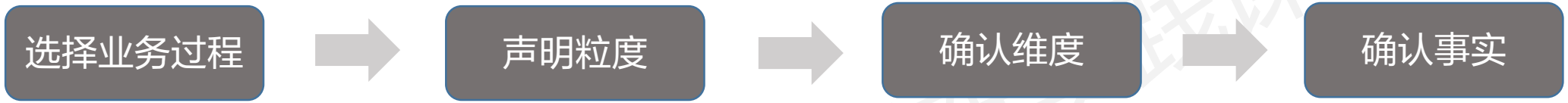
- 维度建模强调两个需求：
  - 以商业用户可理解的方式发布数据。
  - 提供高效的查询性能。
- 站在业务需求的角度进行模型构建，紧紧围绕着业务模型，可以直观地反映出业务模型中的业务问题。
- 各个维作了大量的预处理，如按照维进行预先的统计、分类、排序等。通过这些预处理，能够极大的提升数据仓库的处理能力。
- 维度模型的核心：事实表和维度表。
- 常以星型模型、雪花模型、星座模型等形式进行组织。

事实表和维度表

- 事实表是维度模型的基本表，存放有大量的业务性能度量值以及指向各个维的键值。
- 维度表包含与业务过程度量事件有关的文本环境。存放该维的元数据，即维的描述信息，包括维的层次及成员类别等。
- 星型模型：事实表和维度表连接之后类似星星结构。

维度建模设计流程及案例背景说明

- 维度建模流程：



- Litemall是一个简单的商城系统，基于现有的开源项目开发，重新实现一个完整的前后端项目。
- Litemall中一共包含34张表，实现会员管理、商城管理、商品管理、推广管理、系统管理、配置管理、统计报表功能。
- 用户可以通过网页参与团购活动，团购成功并付款后商家发货，未发货之前可以申请退货，收货之后给商品添加评价等，相关的数据存放在华为云数据库RDS中。
- 对于Litemall， 管理方面主要关注对订单、 销售产品、用户的组织和管理工作，目的是实现商城利润最大化。

维度建模设计 - 选择业务过程 (1)

- 步骤1：选择业务过程

业务过程是什么

- 业务过程是组织完成的**操作型活动**，通常使用“行为动词”来表示。例如，获得订单、买家付款、处理保险索赔、学生课程注册等。
- 在选取业务阶段，数据模型设计者需要具有全局和发展的视角，应该理解整体业务流程的基础上，从全局角度选取业务处理。
- 业务过程事件建立或获取性能度量，并转换为事实表中的事实。在选择了业务过程以后，相应的事实表类型也就确定了。

维度建模设计 - 选择业务过程 (2)

- 针对本次Litemall案例，对整体业务进行分析，我们希望能够更好地理解Litemall商城的经营情况，包括商城GMV，商品销量排行，区域销量排行，客户购买行为等等。因此首先，我们重点关注订单系统。
- 在订单处理上，完整的业务逻辑包含：客户下单 → 生成订单 → 客户付款 →（客户申请退款 → 管理员操作 → 退款完成）→ 订单发货 → 物流配送 → 客户收货。

维度建模设计 - 声明粒度 (1)

- 步骤2：声明粒度

粒度如何理解

- 粒度用于确定某一事实表中的行表示什么。
- 回答“如何描述事实表的单个行?”这一问题。
- 在从给定的业务过程获取数据时，**原子粒度是最低级别的粒度**。强烈建议从关注原子级别粒度数据开始设计，因为原子粒度数据能够承受无法预期的用户查询。
- 针对不同的事实表粒度，要建立不同的物理表，在同一事实表中不要混用多种不同的粒度。

维度建模设计 - 声明粒度 (2)

- 原子数据能够提供最佳的分析灵活性。
- 当然，也可以定义汇总粒度来表示对原子数据的聚集。然而，一旦选择了级别较高的粒度，就限制了建立更细节的维度的可能性。粒度较高的模型无法实现用户下钻细节的需求。
- 针对本次Litemall案例，我们希望通过原子数据来进行不同层面不同角度的分析，希望能够保留最底层最细节的数据来对业务情况进行洞察。最细粒度的数据为订单中某个商品下某种货品的交易信息。

维度建模设计 - 确认维度 (1)

- 步骤3：确认维度

维度有什么用

- 提供围绕某一业务过程事件所涉及的“谁、什么、何处、何时、为什么、如何”等背景。即：维度包含与业务过程度量事件有关的文本环境。
- 维度表是业务分析的入口和描述性标识。例如，想要从“地区”、“用户”、“渠道”等不同角度来衡量整体的销售情况，那么就需要有相应的维度表来对地区信息、用户信息、渠道信息进行记录。
- 维度表通常有多列，或者说包含多个属性，但包含较少的行（通常少于100万行)。

维度建模设计 - 确认维度 (2)

- 针对本次Litemall案例，主要的维度表包括：
  - 日期
  - 时间
  - 商品
  - 用户
  - 地址

维度建模设计 - 确认维度 (3)

- 日期维度 (1)
  - 日期维度是一种特殊的维度，因为它几乎出现在所有的维度模型中。
  - 可以提前建立日期维度表。可以在表中按行表示10年或20年的不同日期，因此可以涵盖历史的日期，也可以包含未来的几年。
  - 日期维度表中的每列由行表示的特定日期定义。weekday列包含天的名称，如周一。使用该列可建立用于比较周一与周日业务的报表。日期数字是日历月列从每月1号开始，根据不同的月份以28、29、30、31日结束。该列用于比较每个月的相同一天的情况。
  - 利用SQL提供的日期语义按照月或年过滤，从而可避免非常昂贵的连接操作。

维度建模设计 - 确认维度 (4)

- 日期维度 (2)

日期维度
date_Key (PK)
full_date
day_of_week
day_num_in_month
day_num_overall
day_name
day_abbrev
weekday_flag
week_num_in_year
week_num_overall
week_begin_date
week_begin_date_key

如果包含了具体的时间信息，那维度表会发生什么变化?

日期维度
month
month_num_overall
month_name
month_abbrev
quarter
year
yearmo
fiscal_month
fiscal_quarter
fiscal_year
last_day_in_month_flag
same_day_year_ago_date

维度建模设计 - 确认维度 (5)

- 时间维度
  - 一个简单的时间维度如表所示，也可以增加更多的属性信息，且这些属性后续能够在分析中发挥价值。

时间维度
Time_key
Hour
Minute
Second
T_am_pm
...

维度建模设计 - 确认维度 (6)

- 商品维度 (1)
  - 商品维度主要包含每个商品的描述性属性。
  - 商品层次是属性的主要分组之一。单个的货品上卷到商品，商品上卷到类别。每一不同层次都存在多对一关系。
  - 商品维度表中的大多数属性并不是商品层次的组成部分，例如单位类型属性的值可能包括瓶、包、盒或听等。例如在某类目下面查找所有以件为单位的货品。

维度建模设计 - 确认维度 (7)

- 商品维度 (2)

商品维度	商品维度	商品维度	商品维度
goods_key(PK)	keywords	detail	category_sort_order
goods_id	brief	product_id	category_add_date
goods_sn	is_on_sale	specifications	category_add_time
goods_name	sort_order	gp_add_date	cateogy_update_date
category_id	pic_url	gp_add_time	category_update_time
category_name	share_url	gp_update_date	category_deleted
category_keywords	is_new	gp_update_time	goods_add_date
category_desc	is_hot	gp_deleted	goods_add_time
category_pid	unit	category_icon_url	goods_update_date
brand_id	counter_price	category_pic_url	goods_update_time
gallery	retail_price	category_level	goods_deleted

维度建模设计 - 确认维度 (8)

- 用户维度
  - 用户维度描述litemall商城的所有用户属性信息。

用户维度表
user_key (PK)
user_id
username
password
gender
birthday
last_login_date
last_login_time
last_login_ip

用户维度表
user_level
nickname
mobile
username
avatar
weixin_openid
session_key
status

用户维度表
update_date
update_time
add_date
add_time
update_date
update_time
deleted

维度建模设计 - 确认维度 (9)

- 地址维度
  - 地址维度描述了每一个收货地址的详细信息。
  - 地址维度是本案例中主要的地理维度，可以按照地理属性进行上卷操作，例如省、市、区等等。

地址维度表
address_key (PK)
address_id
province
city
county
address_detail
area_code

地址维度表
postal_code
is_default
add_date
add_time
update_date
update_time
deleted

维度建模设计 - 确认事实 (1)

- 步骤4：确认事实

事实怎么呈现

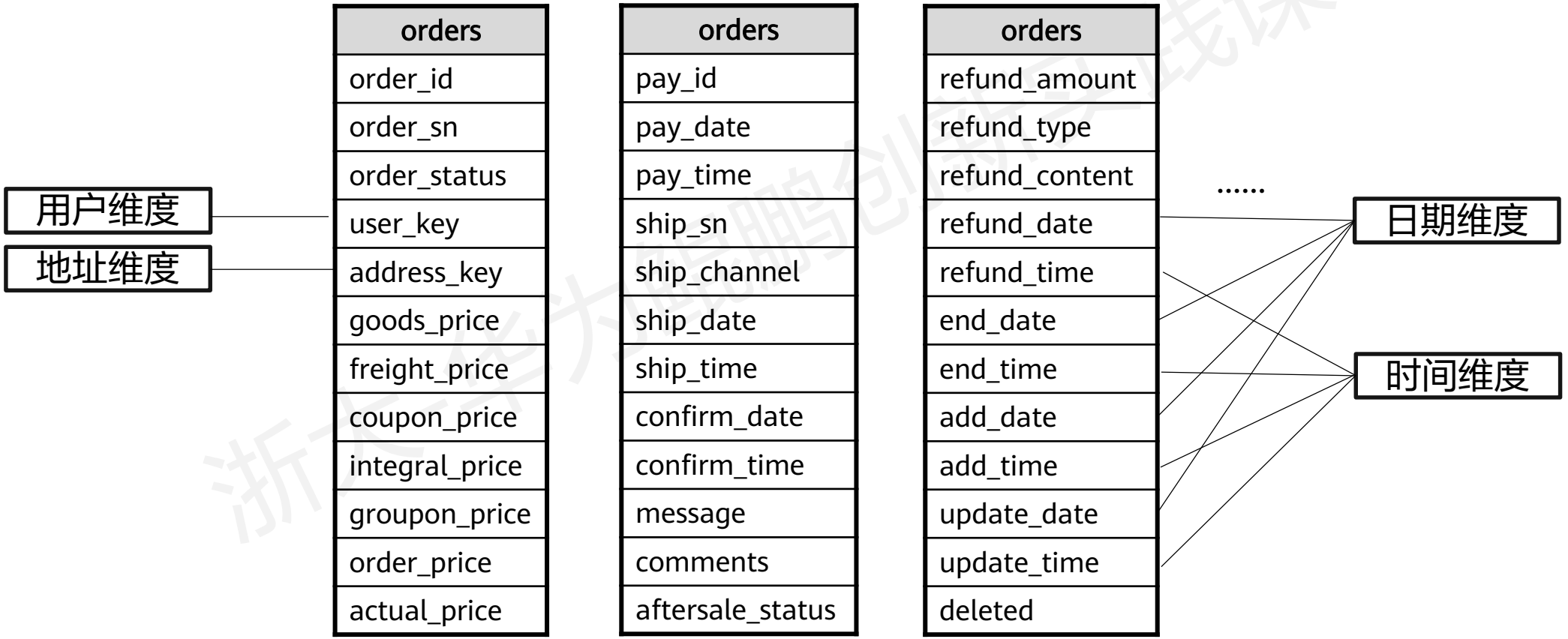
- “事实”这一术语表示某个业务度量，基本上都是以数量值表示。
- 事实表中的每行对应一个度量事件。每行中的数据是一个特定级别的细节数据，称为粒度。因此事实表对应一个物理可观察的事件。
- 维度模型中的事实表存储业务过程事件的度量结果。
- 维度建模的核心原则之一，是同一事实表中的所有度量行必须具有相同的粒度。

维度建模设计 - 确认事实 (2)

- 针对litemall案例，设计的最后一步是确认应该将哪些事实放到事实表中。事实必须与粒度吻合：订单中某个商品下某种货品的交易信息，包含每个货品的成交金额、成交数量、折扣信息等。
- 用户下单时，一个订单可以包含多个商品，在此我们可以将订单相关的数据组织为订单表和订单商品表两张事实表。另外，我们需要一张库存表来存储商品的实际库存量数据。
- 事实表包含：订单表、订单商品表和库存表。

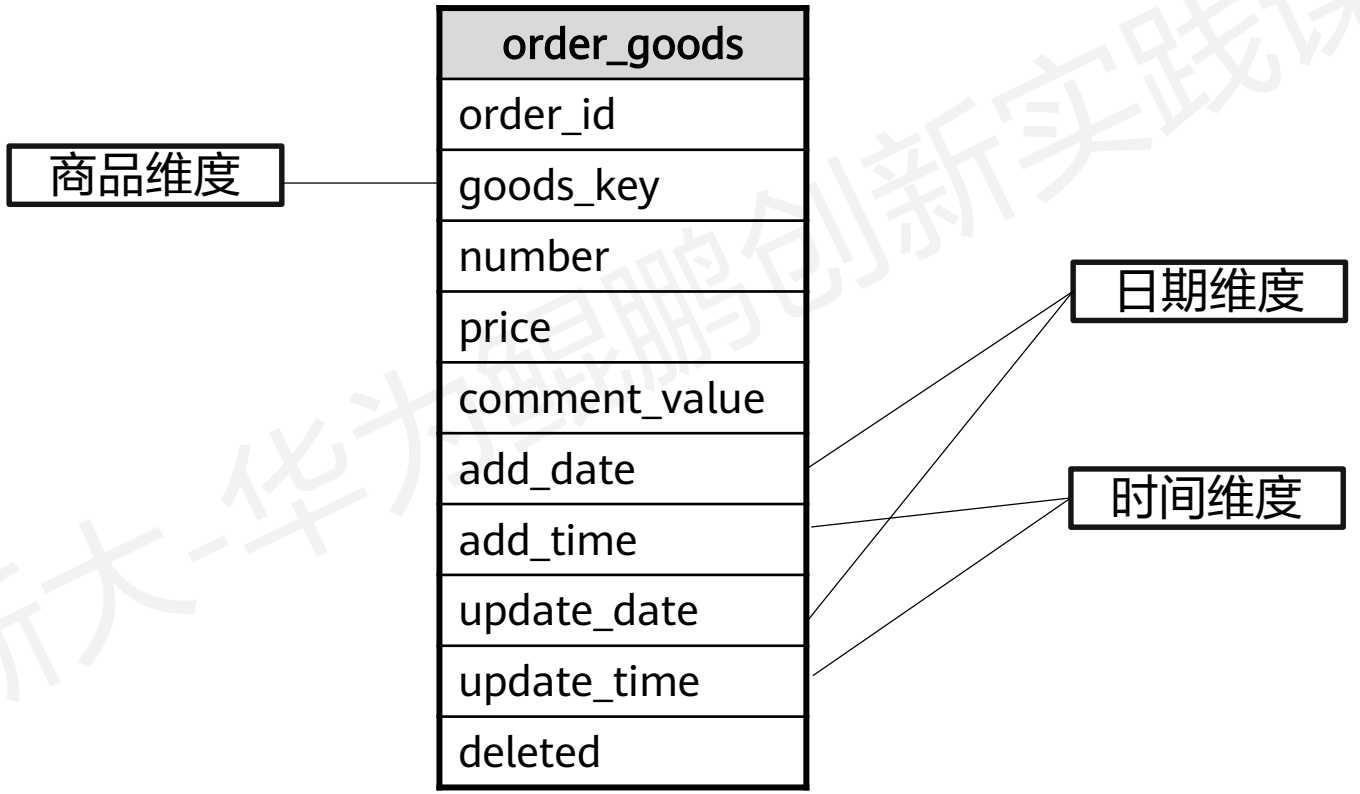
维度建模设计 - 确认事实 (3)

- orders表



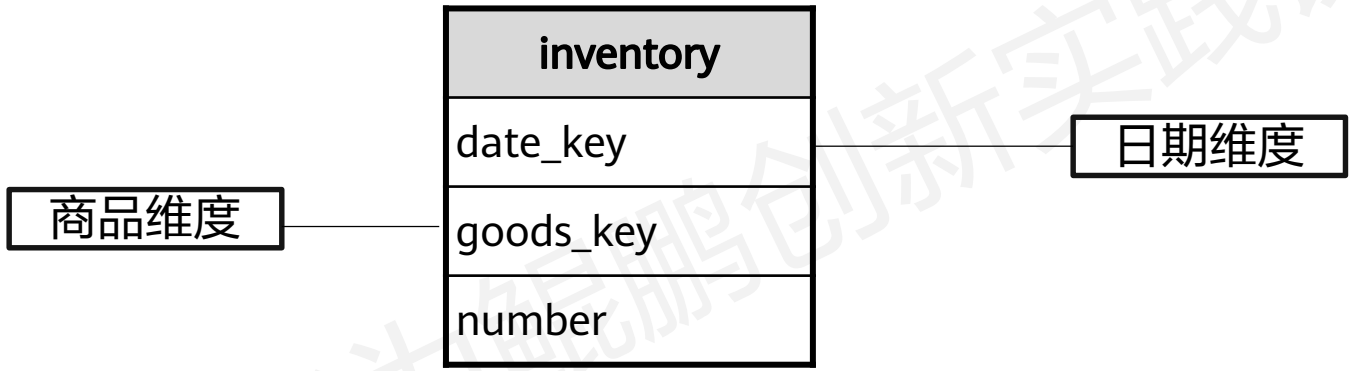
维度建模设计 - 确认事实 (4)

- order_goods表



维度建模设计 - 确认事实 (5)

- inventory表



事实表技术基础

- 三类度量
  - 可加度量：可以按照与事实表关联的任意维度汇总。例如：销量。
  - 半可加度量：可以对某些维度汇总， 但不能对所有维度汇总。 例如：差额、库存等。
  - 不可加度量：不能按照维度进行汇总。例如：比率。
- 一致性事实
  - 如果某些度量出现在不同的事实表中，需要注意，如果需要比较或计算不同事实表中的事实，应保证针对事实的技术定义是相同的。如果不同的事实表定义是一致的，则这些一致性事实应该具有相同的命名。即：针对事实表中的字段，在进行多表联查时，保证同一字段在不同表之间数据类型保持一致，同时尽量保证字段的名称也一致。

维度表技术基础 (1)

- 维度代理键
  - 维度表中会包含一个列，表示唯一主键。该主键不是操作型系统的自然键，这些维度代理键可以是按顺序分配的简单整数，以值1开始。每当需要新键时，键值自动加1。
- 下钻 (drill down)
  - 从汇总数据深入到细节数据进行观察或增加新维。下钻是商业用户分析数据的最基本的方法。下钻在SQL语句中的具体体现是使用GROUP BY表达式。

部门名称	销售总额
Bakery	12,331
Frozen Foods	31,776
.....	
.....	

按部门汇总销售额

部门名称	品牌名称	销售总额
Bakery	Baked Well	3,009
Bakery	Fluffy	6,298
Frozen Foods	Coldpack	10,476
Frozen Foods	Icy	6,467
...	...	...

按照品牌下钻

维度表技术基础 (2)

- 退化维度
  - 有时候维度除了主键外没有其他内容。这种退化维度被放入事实表中， 清楚地表明没有关联的维度表。
- 日历日期维度
  - 日历日期维度通常包含许多描述，例如，周数、 月份名称、 财务周期、 国家假日等属性，能够对事实表按照熟悉的日期、 月份、 财务周期和日历上的特殊日期进行导航。
  - SQL暂时不支持计算节假日。

目录

数据库

- 关系型数据库包含一组数据对象，用于存储、管理和访问数据对象，包括表、视图、索引等。
- 创建数据库：CREATE DATABASE

```
CREATE DATABASE database_name;
```

- 修改数据库：ALTER DATABASE

```
ALTER DATABASE database_name RENAME TO new_name;
```

- 删除数据库：

```
DROP DATABASE [ IF EXISTS ] database_name ;
```

模式

- 又叫做 schema。
- 模式提供数据库对象的逻辑分类。
- 在同一个数据库下可以存在多个不同名的模式。
- 一个模式下可以包含函数/存储过程、表、视图、序列和索引等对象。这些对象在不同模式下名字可以相同。
- 操作不同模式下的对象互不干扰。
- 模式方便地将同一数据库下的数据进行分类并统一管理。

模式操作

- CREATE SCHEMA 模式创建

```
CREATE SCHEMA test_sche;
```

- ALTER SCHEMA 模式修改

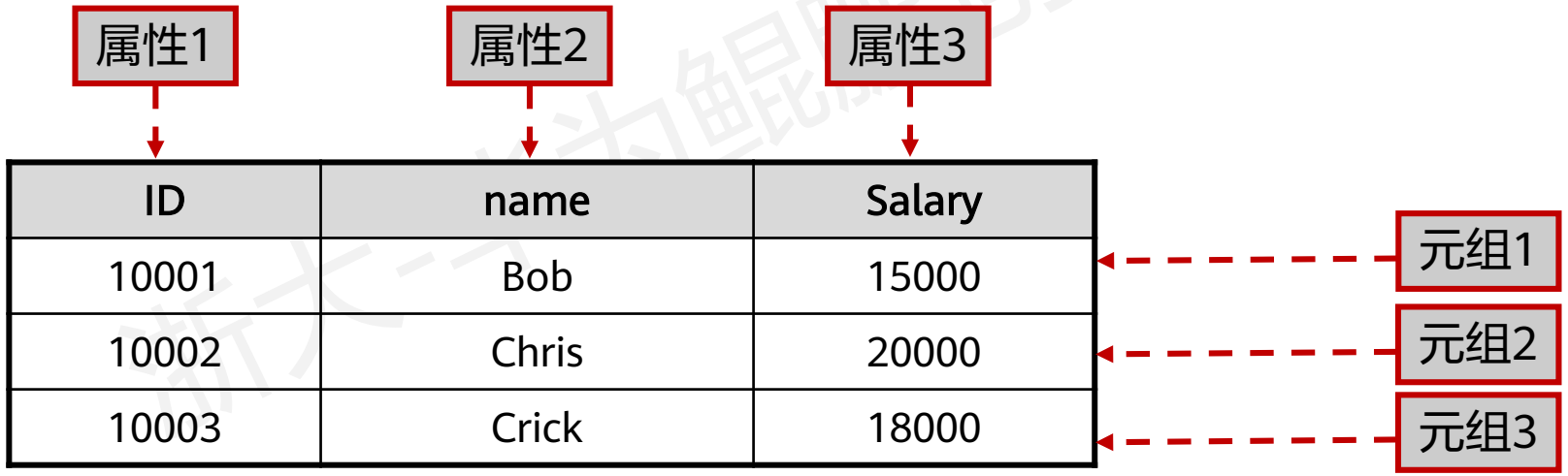
```
ALTER SCHEMA RENAME TO new-name;
```

- DROP SCHEMA 模式删除
  - 必要时添加 CASCADE 进行级联删除。
- 访问模式下对象时可以用模式名作为前缀。

```
SELECT * FROM test_sche.test_tbl;
```

表

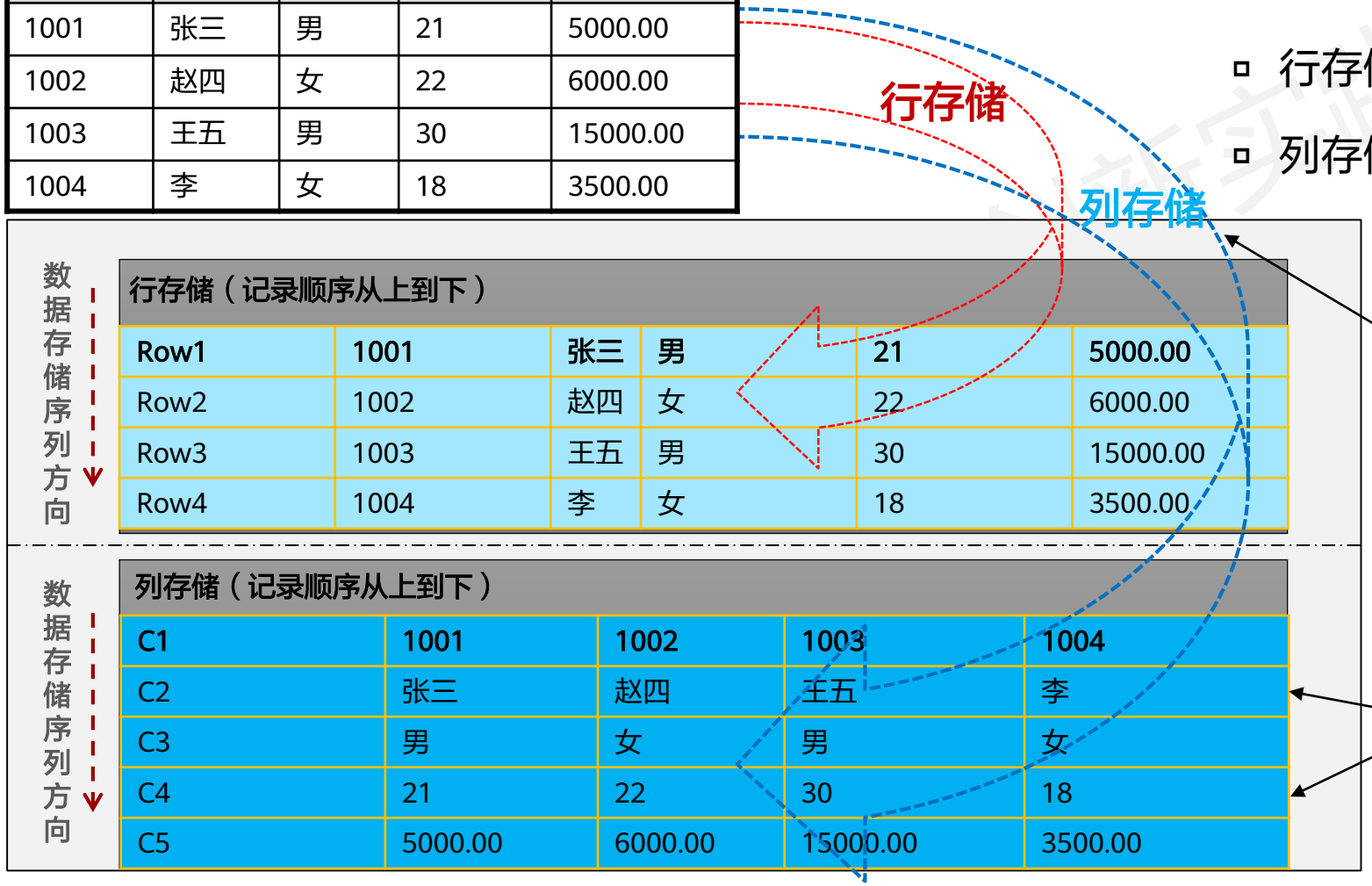
- 在关系数据库中，表用来表示数据和数据之间的联系，关系(relation)表示表。
- 一行的所有列组成一条记录，也叫元组(tuple)。关系是元组的集合。
- 每个表可以有多个列，也称为属性(attribute)。每个列都有唯一的名字，每个列都有特定的类型。



表的存储方式 (1)

工号	姓名	性别	年龄	薪水
1001	张三	男	21	5000.00
1002	赵四	女	22	6000.00
1003	王五	男	30	15000.00
1004	李	女	18	3500.00

- 按照数据的存储方式，表分为两种
  - 行存储表
  - 列存储表



写入过程要把记录拆开后不同的列数据分别写入不同的存储区域，多次写入过程会导致IO次数增加，效率相对较慢。

select name,age from employee;
对于列存储表的查询，只要扫描少量所需要列对应的存储即可，IO开销比较小。

表的存储方式 (2)

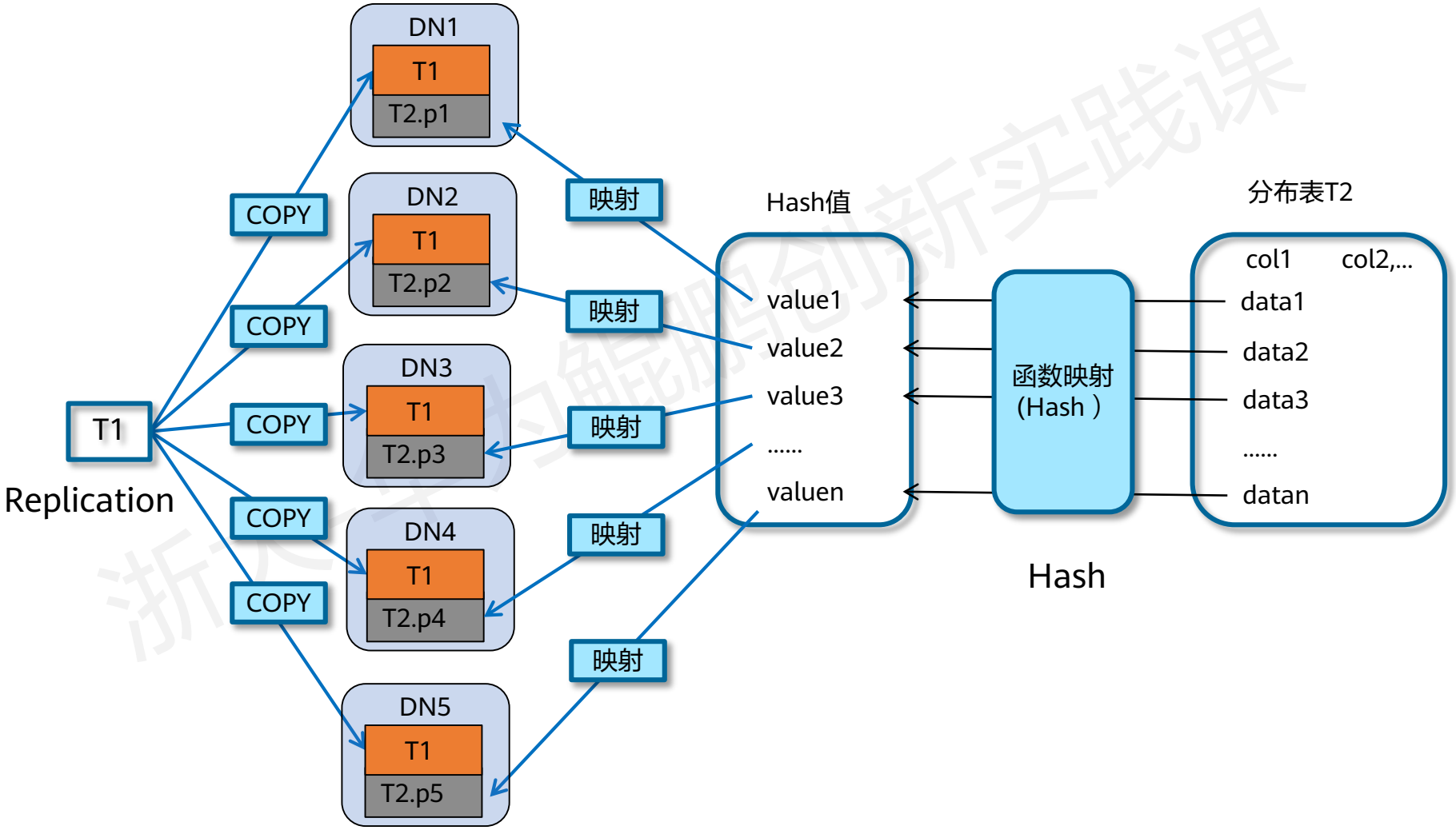
- 行存表
  - 表的数据将以行式存储，每行的所有属性存储到一起。
  - 适合查询一行的所有属性。
  - INSERT、UPDATE效率高。
  - 默认方式。
- 列存表
  - 表的数据将以列式存储，每列的多个纪录存储到一起。
  - 适用于海量数据查询，减少磁盘访问数据量。投影高效。
  - INSERT、UPDATE较为麻烦。

表的存储方式 (3)

- 使用ORIENTATION 指定存储方式
  - ROW
  - COLUMN
- 使用COMPRESSION 指定压缩级别
  - 压缩级别越高，压缩比越大。

```
CREATE TABLE test_tpcds(a INT, b VARCHAR(10))WITH (
    ORIENTATION = COLUMN,
    COMPRESSION = MIDDLE
);
```

表数据的分布方式 (1)



表的分布方式 (2)

- 复制表
  - 每个数据节点都有完整的表数据。
- 哈希表
  - 对表中指定的列进行哈希，根据哈希值映射到指定的数据节点。
  - 所有数据节点合起来才是完整的表数据。
  - 默认创建方式。
- DISTRIBUTE BY 指定分布方式

```
CREATE TABLE test_tpcds(a INT, b VARCHAR(10)) DISTRIBUTE BY HASH(a);
```

```
CREATE TABLE test_tpcds(a INT, b VARCHAR(10)) DISTRIBUTE BY REPLICATION;
```

表类型介绍 - 分区表 (1)

- 逻辑上的一张表根据某种方案分成几张物理块进行存储，这张逻辑上的表称之为分区表，物理块称之为分区。
- 分区表优势：
  - 改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索效率。
  - 增强可用性：如果分区表的某个分区出现故障，表在其他分区的数据仍然可用。
  - 方便维护：如果分区表的某个分区出现故障，需要修复数据，只修复该分区即可。
  - 均衡I/O：可以把不同的分区映射到不同的磁盘以平衡I/O，改善整个系统性能。
- GaussDB(DWS)支持范围分区。
  - 范围分区：根据表的指定几列（分区键），将插入数据分成若干范围。
  - 每个范围都是一个分区，分区没有重叠。

表类型介绍 - 分区表 (2)

- PARTITION BY RANGE(partition_key)
  - partition_key 分区键属性名。
- 从句VALUES LESS THAN 格式指定多个分区。
  - 每个分区都需要指定上边界。
  - 每个分区的上边界需要递增排列。
  - 分区键最多支持4列。

```
CREATE TABLE part_tbl1 (a int, b int)
PARTITION BY RANGE(a)
(
    PARTITION part1 VALUES LESS THAN (10),
    PARTITION part2 VALUES LESS THAN (100),
    PARTITION part3 VALUES LESS THAN (MAXVALUE)
);
```

表类型介绍 - 分区表 (3)

- 从句是START END格式。
  - 相邻两个分区，第一个END值必须等于第二个分区的START值。
  - START END和LESS THAN不可混用。
  - 分区键只支持一列。
  - 每个分区包含起始值，不包含终点值。起始值是MINVALUE时不包含。

```
CREATE TABLE part_tbl2 (a int, b int)
PARTITION BY RANGE(a)
(
    partition part1 START(1) END(100) EVERY(50),
    partition part2 END(200),
    partition part3 START(200) END(300),
    partition part4 start(300)
);
```

表 - CREATE TABLE AS

- 创建新表，表的字段与AS 后面SELECT查询输出字段保持一致。
- SELECT查询源表，将查询结果写入新表。
- 分区表不能采用此方式创建。
- 指定WITH NO DATA 只创建表，不插入数据。默认插入。

```
CREATE TABLE test_tpcds2 AS SELECT * FROM test_tpcds WITH NO DATA;
```

表设计原则 (1)

- 字段设计
  - 选择数据类型时，在满足业务精度条件下，优先级从高到低依次为整数、浮点数、NUMERIC。
  - 多个表间存在逻辑关系时，表示同一含义字段使用相同类型。
  - 字符串字段尽量使用变长数据类型。不建议使用定长数据类型。
- 约束设计
  - 对约束命名时，区分不同类型约束名字。
  - 字段明确不存在NULL值时添加NOT NULL约束，数据库内部在特定场景会进行优化。
  - 约束尽量显式命名。

表设计原则 (2)

- 存储方式
  - 行存：增删改操作较多的场景。查询返回记录少，返回全部属性。
  - 列存：查询复杂，涉及多表关联，分组等操作。
- 分布方式
  - 哈希表：表数据量较大；事实表
  - 复制表：表数据量较小；维度表

表设计原则 (3)

- 减少查询扫描数据量。通过分区表可以将扫描限制在某些分区，减少数据量。
  - 将具体明显区间性的字段进行分区，如日期。
  - 分区上边界值定义为MAXVALUE，避免数据溢出。
- 数据在各个节点尽量均匀分布，避免数据倾斜。分布列选择原则：
  - 作为分布列的字段取值尽量离散。
  - 选择查询中关联条件作为分布列。

Sequence（1）

- 序列，一个存放等差数列的特殊表，通常用于生成唯一的标识符。
- 创建序列

```
CREATE SEQUENCE serial START 100 INCREMENT 3 MAXVALUE 110;
```

  - START 指定序列起始值
  - INCREMENT 序列变化的步长，可以为负。
  - MAXVALUE 指定序列最大值。
  - CYCLE 在序列达到最大值后继续循环使用。定义 CYCLE 后不能保证序列唯一性。
  - 使用 nextval 函数获取下一个序列值。

Sequence (2)

- 从序列中选出下一个数字：

```
SELECT NEXTVAL('serial');
 nextval
---------
     100
```

- 创建与表关联的序列：

```
CREATE TABLE customer_address
(
    ca_address_sk             integer               not null,
    ca_address_id             char(16)               not null,
    ca_street_number          char(10)                      ,
    ca_location_type          char(20)
) ;
```

```
CREATE SEQUENCE serial1
 START 101
 MAXVALUE 1000
OWNED BY customer_address.ca_address_sk;
```

Sequence (3)

- 序列整型：serial（四字节序列整型）
- serial类型不是真正意义上的类型，只是为在表中设置唯一标识做的概念上的便利。
- 可以创建一个整数字段，并且把它的缺省数值安排为从一个序列发生器读取。

```
CREATE TABLE serial_type_tab(a SERIAL);
INSERT INTO serial_type_tab VALUES(default);
INSERT INTO serial_type_tab VALUES(default);

SELECT * FROM serial_type_tab;
a
---
1
2
```

本章总结

- 本章主要讲述了：
  - 数据仓库、维度建模基本概念；
  - 主流的数据仓库架构；
  - 维度建模流程；
  - 维度建模案例分析；
  - 数据库对象的管理。


谢 谢
www.huawei.com