同济大学

TONGJI UNIVERSITY

数据库应用开发设计报告

课题名称	城市共享单车管理与调度系统
学　　院	电子信息与工程学院
专　　业	数据科学与大数据技术
学生姓名	范潇
学　　号	2254298
日　　期	2024 年 6 月 1 日

目 录

1 需求分析 …… 1
1.1 问题背景 …… 1
1.1.1 共享单车的概念 …… 1
1.1.2 共享单车企业面对的挑战 …… 1
1.2 目标场景 …… 1
1.3 数据需求 …… 2
1.4 功能需求 …… 2
2 概念设计 …… 3
2.1 E-R 图 …… 3
2.2 实体集设计 …… 3
2.3 关系集设计 …… 7
3 逻辑设计 …… 10
3.1 实体集模式 …… 10
3.1.1 模式设计 …… 10
3.1.2 属性类型设计 …… 10
3.2 关系集模式 …… 11
3.2.1 模式设计 …… 11
3.2.2 属性类型设计 …… 12
3.3 约束设计 …… 12
3.4 范式分析 …… 13
3.4.1 第一范式 …… 13
3.4.2 第二范式 …… 15
3.4.3 第三范式 …… 15
3.4.4 BC 范式 …… 15
4 物理设计 …… 16
4.1 索引类型选择 …… 16
4.2 索引设计 …… 16
参考文献 …… 18

1 需求分析

1.1 问题背景

1.1.1 共享单车的概念

共享单车作为共享经济的新形态，是指企业与政府进行合作，在居民区、商业区、地铁站、公交车站、校园等公共场所提供自行车骑行的共享服务。它是借助互联网技术推出的一种分时租赁业务。在整个运营过程中由智能技术提供支持，企业负责整个运营过程中的管理[1]。

1.1.2 共享单车企业面对的挑战

消费者在选择共享单车品牌时，除了定价、押金等经济因素，还会着重考量以下体验因素：

- 单车性能：例如单车重量，可调节性等；
- 维护质量：是否有大量未及时回收的损坏单车；
- 单车分布：单车的时空分布是否符合使用需求的时空分布。

同时，由于共享单车市场的快速发展，交通运输部等 10 部门也对共享单车（互联网租凭自行车）行业发布指导意见，要求“引导有序投放车辆”、“加强互联网租凭自行车标准化建设”、“加强停放管理和监督执法”，“引导用户安全文明用车”等。

为此，想要提高市场竞争力，占据主导地位，共享单车企业首先需要达成以下目标：

- 及时更新迭代投入使用的单车型号；
- 追踪投入使用的单车的情况，如已使用时长；
- 记录并处理用户的反馈，例如单车损坏情况、用车需求；
- 及时回收损坏单车；
- 实时调度闲置单车至需求密集区域；
- 及时对违规停车用户进行处罚。

上述这些目标都可以纳入城市共享单车管理与调度的范畴之中。

1.2 目标场景

本次设计的数据库用于共享单车企业的共享单车管理与调度系统。因此，该数据库中存储的数据均围绕“共享单车”这一主体，并不涉及用户的余额、购买的月卡套餐等信息。

下面列出的是本次设计的数据库的几个可能的使用场景：

- 在共享单车手机应用中，需要使用该数据库中存储的数据来实时显示用户附近的可使用车辆信息；

- 共享单车运维团队定时根据数据库中存储的信息来回收并淘汰使用时间过长或型号过旧的单车；
- 共享单车运维团队定时从数据库中读取近期的用户反馈，并回收涉及的单车；
- 共享单车运维团队根据数据库中存储的行车记录，分析单车使用需求的时空分布，指定调度方案；
- 分析团队利用数据库中存储的行车轨迹数据对用户使用习惯进行分析；
- 分析团队利用聚类等方式对数据库中存储的单车位置数据进行分析；
- 分析团队对数据库中存储的历史数据进行可视化。

1.3 数据需求

针对上述应用场景，本数据库存储和下列对象相关的数据：

- 单车
- 单车轨迹
- 单车回收信息
- 单车调度信息
- 单车投放信息
- 用户
- 用户反馈
- 仓库

1.4 功能需求

根据上述的应用场景，可知该数据库中需要进行的操作具有以下特点：

- 经常需要根据时间属性进行范围选择，并且以附近的时间段为主；
- 需要针对空间数据，即坐标，进行范围查询，即获取一定范围内的数据；
- 更新单车回收、投放、调度信息时，需要同步更新单车坐标，仓库存储情况等数据；
- 特定数据更新频率较高，例如单车的当前坐标、状态信息；
- 有时需要读出大量数据，例如进行数据分析和可视化时；
- 较少进行删除操作。

在进行后续的设计时，将针对上述特点对数据库进行优化。

2 概念设计

2.1 E-R 图

根据上述的需求分析，在本次设计的数据库中，包含以下实体集：

- *bike*：拥有属性 (*bike_ID,production_date,coordinate,valid*)；
- *collection*：拥有属性 (*collection_ID,time*)；
- *reallocation*：拥有属性 (*reallocation_ID,start_time,start_coordinate,end_time,end_coordinate*)；
- *release*：拥有属性 (*release_ID,coordinate,time*)；
- *trace*：拥有属性 (*trace_ID,start_time,start_coordinate,end_time,end_coordinate*)；
- *type*：拥有属性 (*type_ID,name,release_date*)；
- *user*：拥有属性 (*user_ID*)；
- *warehouse*：拥有属性 (*warehouse_ID,capacity,load,coordinate*)，

并且包含以下关系集：

- *bike_collection*：将单车回收工单、单车和仓库关联在一起；
- *bike_reallocation*：将单车调度工单和单车关联在一起；
- *bike_release*：将单车投放工单、单车和仓库关联在一起；
- *bike_type*：将单车和单车型号关联在一起；
- *stored_in*：将单车和仓库关联在一起；
- *usage*：将单车、单车轨迹和用户关联在一起；
- *user_feedback*：将单车和用户关联在一起，

图2.1是由以上实体集和关系集组合形成的 E-R 图。

下面将对各个实体集和关系集展开介绍。

2.2 实体集设计

A. *bike*

该实体集的拓展是在现实中公司所拥有的单车。

该实体集有以下属性：

- *bike_ID*：我们认为在同一家共享单车公司中，单车的序列号应当是唯一的，所以将其作为该实体集的主码；
- *production_date*：单车的生产日期；
- *coordinate*：该属性用于追踪单车的当前坐标；
- *valid*：该属性标记单车状态是否合法，当单车被发现遭到人为破坏、故意藏匿等情况，从而导致 GPS 失效、无法进行调度等情况时，将该属性值置为 *False*。

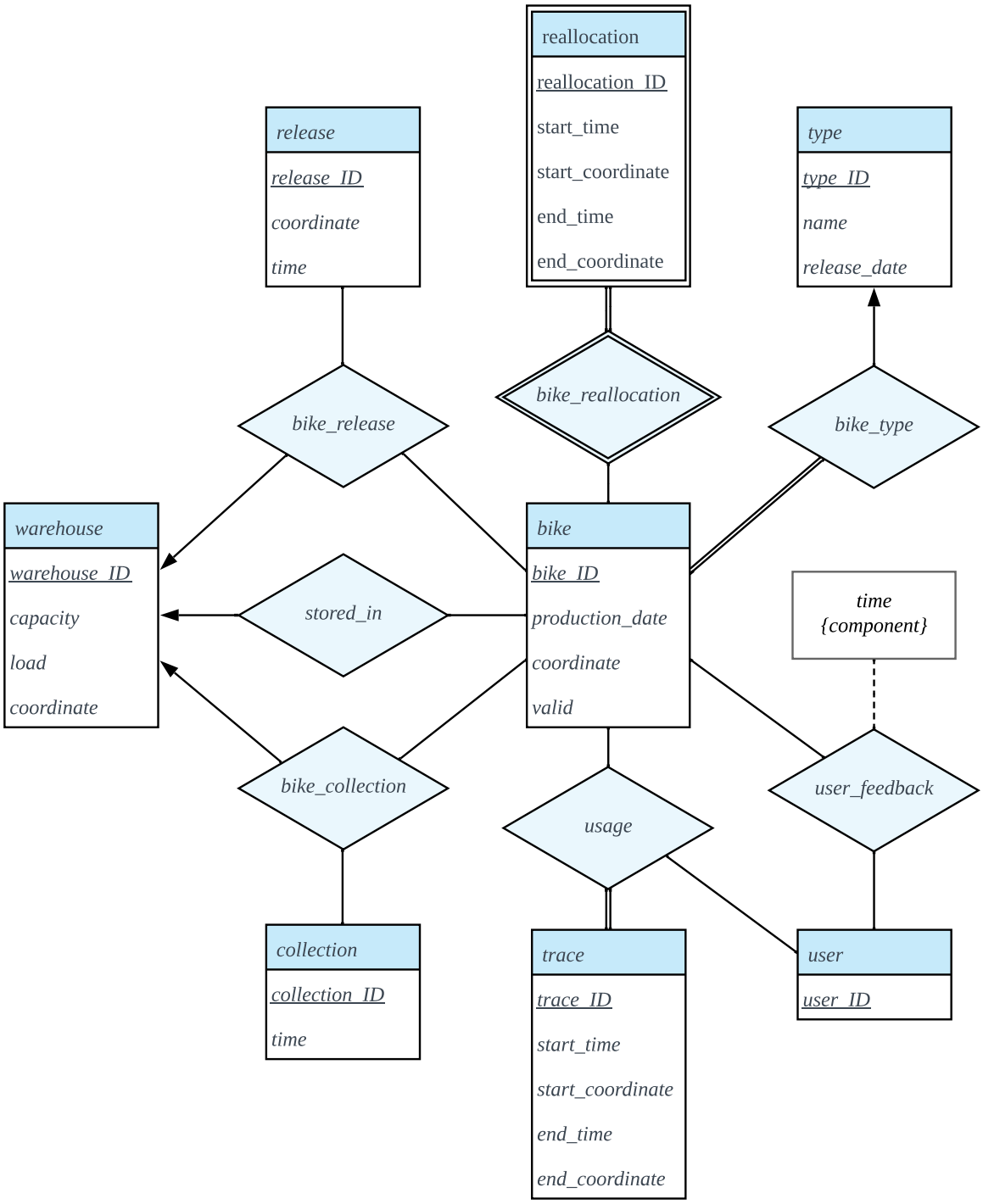
reallocation
reallocation_ID
start_time
start_coordinate
end_time
end_coordinate
release
release_ID
coordinate
time
type
type_ID
name
release_date
bike_reallocation
bike_release
bike_type
warehouse
warehouse_ID
capacity
load
coordinate
stored_in
bike
bike_ID
production_date
coordinate
valid
time
{component}
bike_collection
usage
user_feedback
collection
collection_ID
time
trace
trace_ID
start_time
start_coordinate
end_time
end_coordinate
user
user_ID

图 2.1 E-R 图

装 订 线

这里的 *production_date,coordinate* 实际上都是复合属性，例如 *coordinate* 由经度和纬度组成，但是在该数据库的实际应用场景中，通常将它们作为整体来使用，所以在概念设计中将它们视为整体。

B. *collection*

该实体集的拓展是在现实中公司的运转过程中所产生单车回收工单。这里针对以下两种工作流程进行设计：

- 系统会向调度员派发单车回收工单，调度员根据该工单将指定的单车回收至指定的仓库中；
- 调度员自行收集单车，将其回收至特定仓库，并形成相应工单。

该实体集有以下属性：

- *collection_ID*：我们认为在同一家共享单车公司中，单车回收工单的序列号应当是唯一的，所以将其作为该实体集的主码；
- *time*：完成该工单时的时间戳；

所收集的单车以及回收至的仓库这两个信息体现在关系集 *bike_collection* 中。

C. *release*

该实体集的拓展是在现实中公司的运转过程中所产生单车投放工单。这里针对以下工作流程进行设计：

- 调度员根据工单将一批指定的单车从仓库中运送至指定坐标。

该实体集有以下属性：

- *release_ID*：我们认为在同一家共享单车公司中，单车投放工单的序列号应当是唯一的，所以将其作为该实体集的主码；
- *coordinate*：投放地点坐标；
- *time*：完成该工单时的时间戳；

所投放的单车以及回收至的仓库这两个信息体现在关系集 *bike_collection* 中。

D. *reallocation*

该实体集的拓展是在现实中所产生的调度行为。

和 *collection,release* 不同，该实体集的数据粒度更细，一个元组不再代表对于一批单车的操作，而是对单个单车的操作。这是因为调度对精细化程度的要求通常更高。我们认为同一单车调度工单可以涉及到多个单车，但是各个单车的最终投放地点和时间都可能不同。因此我将该实体集涉及为弱实体集，它的识别集是 *bike*，识别关系集是 *bike_reallocation*。

该实体集有以下属性：

- *reallocation_ID*：单车调度工单的序列号，作为该弱实体集的识别器；
- *start_time*：该调度操作的开始时间；

- *start_coordinate*：该调度操作的起始坐标；
- *end_time*：该调度操作的结束时间；
- *end_coordinate*：该调度操作的目的地坐标。

E. *trace*

该实体集的拓展是在现实中用户使用单车时所产生的单车轨迹。

该实体集有以下属性：

- *trace_ID*：我们认为在同一家共享单车公司中，单车轨迹的序列号应当是唯一的，所以将其作为该实体集的主码；
- *start_time*：轨迹起始时间；
- *start_coordinate*：轨迹起始坐标；
- *end_time*：轨迹结束时间；
- *end_coordinate*：轨迹结束坐标。

F. *type*

该实体集的拓展是在现实中企业所研发的共享单车型号。

该实体集有以下属性：

- *type_ID*：我们认为在同一家共享单车公司中，单车的型号代码应当是唯一的，所以将其作为该实体集的主码；
- *name*：型号名称。这里我们假设不同型号可能有相同的名称，但是它们的代码是不同的；
- *release_time*：发布日期。

G. *user*

该实体集的拓展是在现实中企业所拥有的用户。

该实体集有以下属性：

- *user_ID*：我们认为在同一家共享单车公司中，用户 ID 应当是唯一的，所以将其作为该实体集的主码；

因为本数据库用于共享单车的管理与调度，所以无需将“用户余额”等属性添加入该实体集中。

H. *warehouse*

该实体集的拓展是在现实中企业所拥有的共享单车仓库。

该实体集有以下属性：

- *warehouse_ID*：我们认为在同一家共享单车公司中，仓库 ID 应当是唯一的，所以将其作为该实体集的主码；
- *capacity*：仓库的最大存储容量；

- *load*：仓库当前存储量；
- *coordinate*：仓库坐标。这里我们假设两个仓库可能坐标相同，例如当仓库分为两层，或相邻很近时。

2.3 关系集设计

在本节中，我将图表中的实体集的属性略去。

A. *bike_collection*

如图2.2所示，关系集 *bike_collection* 将实体集 *bike,collection,warehouse* 联系在一起。

该关系集表达的是“单车依据单车收集工单被收集至指定仓库”这一事件中“单车”、“单车收集工单”和“仓库”之间的关系。这里使用了 3 元关系集，而非用若干个 2 元关系集进行代替，是为了使得这三者之间的关系更加明确，并且避免进一步提升 E-R 图的复杂度[2]。

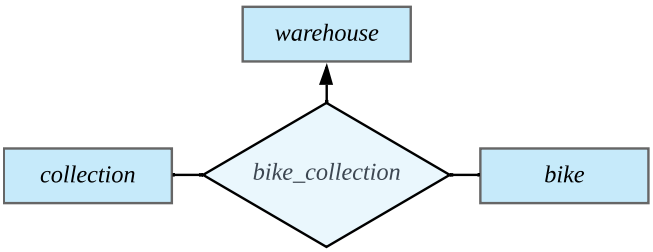

图 2.2 *bike_collection*

图中指向实体集 *warehouse* 的箭头是指给定一个单车和相关联的单车收集工单，最多只有一个仓库与它们相关联。

B. *bike_reallocation*

如图2.3所示，关系集 *bike_reallocation* 将实体集 *bike,reallocation* 联系在一起。

该关系集表达的是“调度单车”这一事件中“单车”和“调度行为”之间的关系。这里的 *reallocation* 是弱实体集，它的存在依附于实体集 *bike*。因此，实体集 *reallocation* 完全参与关系集 *bike_reallocation*。

图 2.3 *bike_reallocation*

C. *bike_release*

如图2.4所示，关系集 *bike_release* 将实体集 *bike,release,warehouse* 联系在一起。

该关系集表达的是“单车依据单车投放工单从指定仓库被投放至指定地点”这一事件中“单车”、“单车投放工单”和“仓库”之间的关系。和 *bike_collection* 一样，这里使用了 3 元关系集。

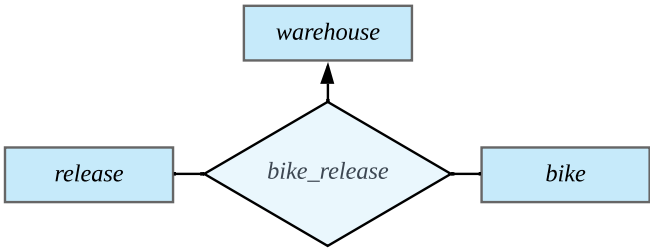

图 2.4 *bike_release*

图中指向实体集 *warehouse* 的箭头是指给定一个单车和相关联的单车投放工单，最多只有一个仓库与它们相关联。由于可能有单车存放在仓库中尚未投入使用，所以实体集 *bike* 只部分参与该关系集。

D. *bike_type*

如图2.5所示，关系集 *bike_type* 将实体集 *bike,type* 联系在一起。

该关系集表达的是“任一单车都具有特定的型号”这一事实中“单车”与“型号”之间的关系。

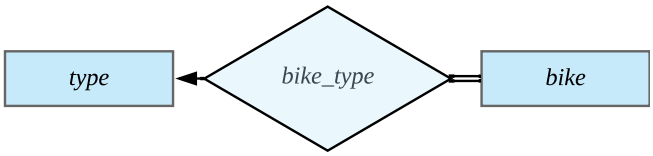

图 2.5 *bike_type*

由于一辆单车有且仅有一个型号，所以图中有指向实体集 *type* 的箭头，并且实体集 *bike* 完全参与该关系集。

E. *stored_in*

如图2.6所示，关系集 *sotred_in* 将实体集 *bike,warehouse* 联系在一起。

该关系集表达的是“单车存储在仓库中”这一状态中，“单车”与“仓库”之间的关系。

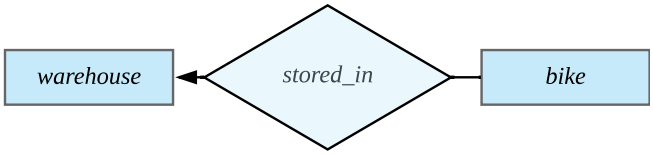

图 2.6 *stored_in*

由于一辆单车至多存放在一个仓库中，所以图中有指向实体集 *warehouse* 的箭头。

F. *usage*

如图2.7所示，关系集 *usage* 将实体集 *bike,user,trace* 联系在一起。

该关系集表达的是“用户使用单车”这一过程中，“单车”、“用户”与“行车轨迹”之间的关系。显然，实体集 *trace* 应完全参与该关系集。

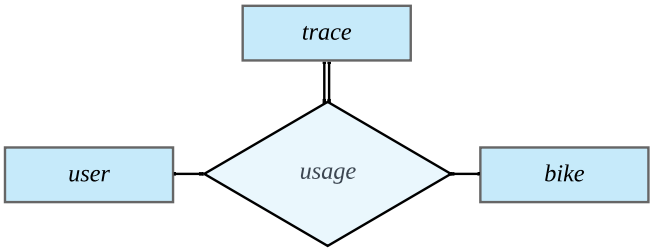

图 2.7 *usage*

G. *user_feedback*

如图2.8所示，关系集 *user_feedback* 将实体集 *bike,user* 联系在一起。

该关系集表达的是“用户反馈单车故障”这一过程中，“单车”与“用户”之间的关系。

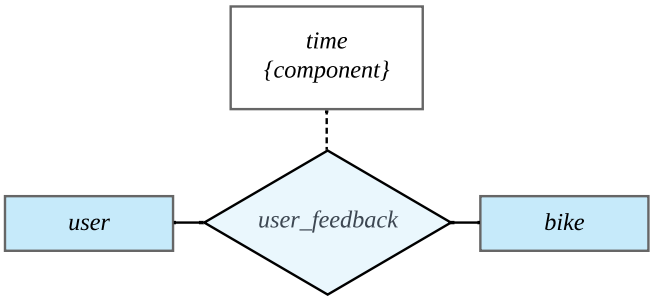

图 2.8 *user_feedback*

用户反馈中的信息十分重要，这里我将反馈的时间 *time* 和反馈的损坏器件 *component* 作为该关系集的描述属性。因为一次反馈可以反馈多处损坏，所以 *component* 为多值属性。

3 逻辑设计

3.1 实体集模式

3.1.1 模式设计

通过转换上述实体集，我们可以得到以下实体集模式：

表 3.1 实体集模式

bike(*bike_ID*,*production_date*,*coordinate*,*valid*)
collection(*collection_ID*,*time*)
reallocation(*reallocation_ID*,*bike_ID*,*start_time*,*start_coordinate*,*end_time*,*end_coordinate*)
release(*release_ID*,*coordinate*,*time*)
trace(*trace_ID*,*start_time*,*start_coordinate*,*end_time*,*end_coordinate*)
type(*type_ID*,*name*,*release_date*)
user(*user_ID*)
warehouse(*warehouse_ID*,*capacity*,*load*,*coordinate*)

在转换过程中，我们认为坐标、日期以及时间都是不可分割的单元。在实际应用场景中，也常常将它们作为整体来进行使用。

对于弱实体集 *reallocation*，它的模式的主码由它自身的识别器 *reallocation_ID* 和识别集 *bike* 的主码 *bike_ID* 组成；对于其余强实体集，它们的模式的主码和原来的主码保持一致。

3.1.2 属性类型设计

图3.1-3.8为各实体集模式中属性的类型。

图 3.1 *bike* 模式属性类型

属性	类型
bike_ID	**char**(20)
production_date	**date**
coordinate	**point**
valid	**int**

图 3.2 *collection* 模式属性类型

属性	类型
collection_ID	**char**(20)
time	**timestamp**

图 3.3 *reallocation* 模式属性类型

属性	类型
reallocation_ID	**char**(20)
bike_ID	**char**(20)
start_time	**timestamp**
start_coordinate	**point**
end_time	**timestamp**
end_coordinate	**point**

图 3.4 *release* 模式属性类型

属性	类型
release_ID	**char**(20)
coordinate	**point**
time	**timestamp**

图 3.5 *trace* 模式属性类型

属性	类型
trace_ID	**char**(20)
start_time	**timestamp**
start_coordinate	**point**
end_time	**timestamp**
end_coordinate	**point**

图 3.6 *type* 模式属性类型

属性	类型
type_ID	**char**(5)
name	**varchar**(50)
release_date	**date**

对于设计过程，有以下几点说明：

图 3.7 *user* 模式属性类型

属性	类型
user_ID	**char**(20)

图 3.8 *warehouse* 模式属性类型

属性	类型
warehouse_ID	**char**(10)
capacity	**int**
load	**int**
coordinate	**point**

- 使用 MySQL 等数据库所支持的 **point** 类型来表示坐标信息；
- 假设 *bike_ID* 等识别码是定长的，所以将它们的类型设为 **char**。由于单车类型和仓库的数量较少，所以 *type_ID,warehouse_ID* 分别的类型分别设为 **char(5)**,**char(10)**。

3.2 关系集模式

3.2.1 模式设计

通过转换上述关系集，我们可以得到以下关系集模式：

表 3.2 关系集模式

bike_collection(*bike_ID*,*collection_ID*)
collection_warehouse(*collection_ID*,*warehouse_ID*)
bike_release(*bike_ID*,*release_ID*)
release_warehouse(*release_ID*,*warehouse_ID*)
bike_type(*bike_ID*,*type_ID*)
stored_in(*bike_ID*,*warehouse_ID*)
usage(*trace_ID*,*user_ID*,*bike_ID*)
user_feedback(*user_ID*,*bike_ID*,*time*)
feedback_component(*user_ID*,*bike_ID*,*time*,*component*)

在转换的过程中，联系弱实体集 *reallocation* 和其识别集 *bike* 的关系集 *bike_reallocation* 被略去，因为这是冗余的[3]。同时，我将三元关系集 *bike_collection,bike_release* 分别拆分为了两个模式，这是为了保证模式满足第二范式，否则，属性 *warehouse* 将会部分依赖于主码 *collection_ID,bike_ID* 中的 *collection_ID* 和主码 *release_ID,bike_ID* 中的 *release_ID*。

对于“多对一”和“一对多”的二元关系集，它的模式的主码为“多”那一侧的实体集模式的主码；对于“多对多”的二元关系集，它的主码是它关联起来的实体集模式的主码与它的描述属性的并集；对于“一对一”的二元关系集，它的主码可以是它关联起来的任一实体集的主码[4]。表3.2中模式的主码便是按照上述标准进行选取的。

对于三元关系集 *usage*，我则是按照函数依赖来选择它们的模式的主码。

对于关系集 *user_feedback*，它的描述属性之一 *component* 是多值属性。为此，转换为模式时，额外创建一个模式 *feedback_component*，其属性由 *user_feedback* 的主码以及属性 *component* 组成。

3.2.2 属性类型设计

关系集模式中属性的类型与实体集中的相应属性类型保持一致。图3.9-3.17为各实体集模式中属性的类型。

图 3.9 *bike_collection* 模式属性类型

属性	类型
collection_ID	**char**(20)
bike_ID	**char**(20)

图 3.10 *collection_warehouse* 模式属性类型

属性	类型
collection_ID	**char**(20)
warehouse_ID	**char**(10)

图 3.11 *bike_release* 模式属性类型

属性	类型
bike_ID	**char**(20)
release_ID	**char**(20)

图 3.12 *release_warehouse* 模式属性类型

属性	类型
release_ID	**char**(20)
warehouse_ID	**char**(10)

图 3.13 *bike_type* 模式属性类型

属性	类型
type_ID	**char**(5)
bike_ID	**char**(20)

图 3.14 *stored_in* 模式属性类型

属性	类型
warehouse_ID	**char**(10)
bike_ID	**char**(20)

图 3.15 *usage* 模式属性类型

属性	类型
trace_ID	**char**(20)
user_ID	**char**(20)
bike_ID	**char**(20)

图 3.16 *user_feedback* 模式属性类型

属性	类型
user_ID	**char**(20)
bike_ID	**char**(20)
time	**timestamp**

图 3.17 *feedback_component* 模式属性类型

属性	类型
user_ID	**char**(20)
bike_ID	**char**(20)
time	**timestamp**
component	**varchar**(50)

3.3 约束设计

约束设计主要分为两部分：外码约束和其他约束。

A. 外码约束

外码约束主要有三个来源：

1. 关系集转化为关系集模式时所产生的；
2. 转化复杂类型时所产生的；
3. 转化弱实体集时所产生的。

在本数据库中，关系集 *user_feedback* 的多值描述属性 *component* 要求我们在将该关系集转化为模式时，需额外生成模式 *feedback_component*，同时，*feedback_component* 中的属性 *user_ID,bike_ID,time* 遵循外码约束，引用 *user_feedback* 的主码。

在本数据库中，弱实体集 *reallocation* 转换为模式 *reallocation*，该模式中的属性 *bike_ID* 需遵循外码约束，引用模式 *bike* 的主码。

其余外码约束均来自于由关系集转化而来的关系集模式。这些模式需要引用它对应的关系集所联系起来的实体集的主码。

B. **其他约束**

本数据库中的其他约束包括主码约束（非空且唯一）、非空约束、唯一约束，check 约束。

由于本数据库中除了用户反馈外的其他所有数据理论上都可以由相应的设备自动生成，如行车轨迹，又或者是必不可少的一部分，例如仓库的坐标，所以所有属性都遵循非空约束。

对于坐标、时间戳等数据，我们不作唯一约束要求。

在本数据库中，check 约束主要用于以下两处：

1. 因为对于 *bike* 中的 *valid* 属性，我们用 **int** 类型来代替布尔类型，所以需要用 check 约束来确保该属性的取值为 0 或 1；
2. 理论上需要用 check 约束来确保用户反馈中填写的 *component* 值是合法的，即属于给定的集合之中。但是这也可以在应用层面实现，例如只允许用户通过勾选的方式来进行反馈。

图3.18-3.34为各模式中的属性所需遵循的约束。图3.34中的 *valid_component* 为一个常量，含义是合法的单车部件。

图 3.18 *bike* 模式属性约束

属性	约束
bike_ID	**primary key**
production_date	**not null**
coordinate	**not null**
valid	**check**(valid **in** (0,1))

图 3.19 *collection* 模式属性约束

属性	约束
collection_ID	**primary key**
time	**not null**

图 3.20 *reallocation* 模式属性约束

属性	约束
reallocation_ID	**primary key**
bike_ID	**primary key** **references** *bike*
start_time	**not null**
start_coordinate	**not null**
end_time	**not null**
end_coordinate	**not null**

图 3.21 *release* 模式属性约束

属性	约束
release_ID	**primary key**
coordinate	**not null**
time	**not null**

图 3.22 *trace* 模式属性约束

属性	约束
trace_ID	**primary key**
start_time	**not null**
start_coordinate	**not null**
end_time	**not null**
end_coordinate	**not null**

图 3.23 *type* 模式属性约束

属性	约束
type_ID	**primary key**
name	**not null**
release_date	**not null**

3.4 范式分析

3.4.1 第一范式

在从 E-R 图转换至模式的过程中，我已经将复杂属性拆分开来，所以所有模式满足第一范式。

图 3.24 *user* 模式属性约束

属性	约束
user_ID	**primary key**

图 3.25 *warehouse* 模式属性约束

属性	约束
warehouse_ID	**primary key**
capacity	**not null**
load	**not null**
coordinate	**not null**

图 3.26 *bike_collection* 模式属性约束

属性	约束
collection_ID	**primary key** **references** *collection*
bike_ID	**primary key** **references** *bike*

图 3.27 *collection_warehouse* 模式属性约束

属性	约束
collection_ID	**primary key** **references** *collection*
warehouse_ID	**not null** **references** *warehouse*

图 3.28 *bike_release* 模式属性约束

属性	约束
release_ID	**primary key** **references** *release*
bike_ID	**primary key** **references** *bike*

图 3.29 *release_warehouse* 模式属性约束

属性	约束
release_ID	**primary key** **references** *release*
warehouse_ID	**not null** **references** *warehouse*

图 3.30 *bike_type* 模式属性约束

属性	约束
bike_ID	**primary key** **references** *bike*
type_ID	**not null** **references** *type*

图 3.31 *stored_in* 模式属性约束

属性	约束
bike_ID	**primary key** **references** *bike*
warehouse_ID	**not null** **references** *warehouse*

图 3.32 *usage* 模式属性约束

属性	约束
trace_ID	**primary key** **references** *trace*
user_ID	**not null** **references** *user*
bike_ID	**not null** **references** *bike*

图 3.33 *user_feedback* 模式属性约束

属性	约束
user_ID	**primary key** **references** *user*
bike_ID	**primary key** **references** *bike*
time	**primary key**

图 3.34 *feedback_component* 模式属性约束

属性	约束
user_ID	**primary key** **references** *user*
bike_ID	**primary key** **references** *bike*
time	**primary key**
component	**primary key** (**check** *component* **in** *valid_component*)
(*user_ID*,*bike_ID*,*time*)	**references** *user_feedback*

装 订 线

3.4.2 第二范式

各个模式的函数依赖集的正则覆盖如下：

表 3.3 各模式中的函数依赖的正则覆盖

$FC_{bike} = \{bike\_ID \rightarrow production\_date, coordinate, valid\}$
$FC_{collection} = \{collection\_ID \rightarrow time\}$
$FC_{reallocation} = \{reallocation\_ID, bike\_ID \rightarrow start\_time, start\_coordinate, end\_time, end\_coordinate\}$
$FC_{release} = \{release\_ID \rightarrow coordinate, time\}$
$FC_{trace} = \{trace\_ID \rightarrow start\_time, start\_coordinate, end\_time, end\_coordinate\}$
$FC_{type} = \{type\_ID \rightarrow name, release\_date\}$
$FC_{user} = \varnothing$
$FC_{warehouse} = \{warehouse\_ID \rightarrow capacity, load, coordinate\}$
$FC_{bike\_collection} = \varnothing$
$FC_{collection\_warehouse} = \{collection\_ID \rightarrow warehouse\_ID\}$
$FC_{bike\_release} = \varnothing$
$FC_{release\_warehouse} = \{release\_ID \rightarrow warehouse\_ID\}$
$FC_{bike\_type} = \{bike\_ID \rightarrow type\_ID\}$
$FC_{stored\_in} = \{bike\_ID \rightarrow warehouse\_ID\}$
$FC_{usage} = \{trace\_ID \rightarrow user\_ID, bike\_ID\}$
$FC_{user\_feedback} = \varnothing$
$FC_{feedback\_component} = \varnothing$

经过验证，各个模式中不存在部分函数依赖，即所有模式满足第二范式。

3.4.3 第三范式

经过验证，各个模式中不存在传递依赖的关系，即所有模式满足第三范式。

3.4.4 BC 范式

经过验证，所有模式满足 BC 范式。

装 订 线

4 物理设计

4.1 索引类型选择

对于本次设计的数据库而言，绝大部分数据具有时序性，并且是按照时间顺序进行插入的。同时，在实际应用中很少对这些具有时序性的数据进行更新和删除的操作。因此，即使存储的数据量较大，当使用顺序存储时，溢出块的数量能够维持在可控范围内。并且在夜间时，该数据库的输入频率将会处于低位，数据库可以利用这一时间对数据进行重新组织。并且，对于具有时序性的数据，在实际应用中常常使用范围查找。综上，对于具有时序性的数据，本数据库中主要对其建立顺序索引。

另一方面，对于 *bike* 等模式，在实际应用中常常需要以坐标为条件进行查询，因此对于这些模式，需要使用 k-d 树或 k-d-B 树等专门用于处理位置信息的索引类型。对于 *bike_ID* 等识别码，在实际应用中常常以等值查询为主，它们的索引以哈希索引为主。

4.2 索引设计

在创建索引时，主要遵循以下几点：

1. 对主码建立索引，以提高检查主码约束的效率；
2. 对被引用的属性建立索引，以提高检查外码约束的效率；
3. 对实际应用中查询频率高或响应时间要求高的属性建立索引。

本数据库中各个模式中的索引信息如图4.1-4.17所示。

图 4.1 *bike* 模式索引信息

属性	索引信息
bike_ID	hash index
production_date	primary index
coordinate	k-d tree
valid	——

图 4.2 *collection* 模式索引信息

属性	索引信息
collection_ID	hash index
time	primary index

图 4.3 *reallocation* 模式索引信息

属性	索引信息
reallocation_ID	hash index
bike_ID	hash index
start_time	primary index
start_coordinate	k-d tree
end_time	secondary index
end_coordinate	k-d tree

图 4.4 *release* 模式索引信息

属性	索引信息
release_ID	hash index
coordinate	k-d tree
time	primary index

图 4.5 *trace* 模式索引信息

属性	索引信息
trace_ID	hash index
start_time	primary index
start_coordinate	k-d tree
end_time	secondary index
end_coordinate	k-d tree

图 4.6 *type* 模式索引信息

属性	索引信息
type_ID	hash index
name	——
release_date	primary index

图 4.7 *user* 模式索引信息

属性	索引信息
user_ID	hash index

图 4.8 *warehouse* 模式索引信息

属性	索引信息
warehouse_ID	hash index
capacity	secondary key
load	primary key
coordinate	k-d tree

图 4.9 *bike_collection* 模式索引信息

属性	索引信息
collection_ID	hash index
bike_ID	hash index

图 4.10 *collection_warehouse* 模式索引信息

属性	索引信息
collection_ID	hash index
warehouse_ID	hash index

图 4.11 *bike_release* 模式索引信息

属性	索引信息
release_ID	hash index
bike_ID	hash index

图 4.12 *release_warehouse* 模式索引信息

属性	索引信息
release_ID	hash index
warehouse_ID	hash index

图 4.13 *bike_type* 模式索引信息

属性	索引信息
bike_ID	hash index
type_ID	hash index

图 4.14 *stored_in* 模式索引信息

属性	索引信息
bike_ID	hash index
warehouse_ID	hash index

图 4.15 *usage* 模式索引信息

属性	索引信息
trace_ID	hash index
user_ID	hash index
bike_ID	hash index

图 4.16 *user_feedback* 模式索引信息

属性	索引信息
user_ID	hash index
bike_ID	hash index
time	primary index

图 4.17 *feedback_component* 模式索引信息

属性	索引信息
user_ID	hash index
bike_ID	hash index
time	primary index
component	hash index

装 订 线

参考文献

[1] 闫佳宜. 共享单车运营规制研究[J]. 合作经济与科技, 2024(07): 179-181.

[2] Database System Concepts[M]. 7th. McGraw-Hill, Inc. New York, NY, USA, 2019: 283-285.

[3] Database System Concepts[M]. 7th. McGraw-Hill, Inc. New York, NY, USA, 2019: 269-270.

[4] Database System Concepts[M]. 7th. McGraw-Hill, Inc. New York, NY, USA, 2019: 257-259.