数据库设计说明书

修订历史记录

变更版本号	日期	变更类型	修改人	摘要
v1.0	2024.10.26	增加内容	杨邑豪	编写说明书 1、2 部分内容
v1.0	2024.10.27	增加内容	杨邑豪	编写说明书 3、4 部分内容
v1.0	2024.10.28	增加内容	杨邑豪	编写说明书 5、6、7、8 部分内容
v1.1	2024.10.29	修改内容	郭剑敏	增加说明书 3.7 存储过程与函数
v1.2	2024.10.29	增加内容	承宇豪	增加说明书 4.3：E-R 图
v1.2	2024.10.29	修改内容	杨邑豪	修改格式及 6 部分
v1.3	2024.10.29	修改内容	杨邑豪	修改 3 部分

目录

数据库设计说明书 1

修订历史记录 1

1 引言 5

1.1 编写目的 5

1.2 范围 5

1.3 参考资料 6

2 数据库环境说明 7

2.1 系统环境 7

2.2 设计工具 7

2.3 编程工具 7

2.4 数据库详细配置 7

3 数据库命名规则 9

3.1 前缀约定 9

3.2 命名风格 9

3.3 表名 9

3.4 字段名 10

3.5 索引命名 10

3.6 视图命名 11

3.7 存储过程和函数命名 11

4 逻辑设计 12

4.1 实体（Entities） 12

4.2 关系（Relationships） 13
4.3 主要信息 E-R 图 14
4.4 注意 14
4.5 简化后的 ERD 描述 14
5 物理设计 16
5.1 数据库表结构 16
5.2.1 表 tbl_users 16
5.2.2 表 tbl_travel_guides 16
5.2.3 表 tbl_destinations 17
5.2.4 表 tbl_routes 17
5.2.5 表 tbl_route_stops 18
6 数据规划 20
6.1 标识与命名约定 20
6.2 内存与外存设计 20
6.2.1 内存安排 20
6.2.2 外存设备及空间组织 20
6.3 访问数据方式 21
6.4 表空间设计 21
6.4.1 表空间设计原则 21
6.4.2 数据文件设计 21
6.5 表、索引分区设计 21
6.5.1 逻辑与物理设计 21

6.5.2 分区原则 22
6.5.3 表与索引的表空间 22
6.6 优化方法 22
6.6.1 时空效率分析 22
6.6.2 折衷方案 22
6.6.3 具体措施 22
7 安全性设计 24
7.1 防止用户直接操作数据库 24
7.2 用户账号加密处理 24
7.3 角色与权限控制 25
7.4 额外安全措施 25
8 数据库的备份策略和方式 27
8.1 备份策略 27
8.1.1 备份频率 27
8.1.2 备份时长与所需时间 27
8.1.3 备份保留周期 27
8.2 备份方式 28
8.2.1 备份类型 28
8.2.2 备份工具 28
8.2.3 注意事项 28

1 引言

1.1 编写目的

随着旅游业的蓬勃发展和移动互联网技术的不断进步，用户对于旅游信息的需求日益个性化和便捷化。为了满足这一市场需求，我们计划开发一款旅游地点推荐APP，旨在为用户提供以下核心功能：

1.导入心仪的旅行攻略链接，一键生成旅行攻略

2.一键规划最优旅行路线

3.显示目的地天气信息

为了支持上述功能，APP 需要存储大量的旅行攻略数据、路线规划数据、用户信息、目的地天气信息等。数据库是高效、安全地存储和管理这些数据的关键：

1.数据查询与检索：用户在使用 APP 时，需要快速查询和检索旅行攻略、路线规划结果、天气信息等。数据库设计将直接影响这些查询操作的效率和准确性。

2.数据一致性与完整性：在多用户并发访问和修改数据的情况下，数据库设计需要确保数据的一致性和完整性，防止数据冲突和丢失。

3.扩展性与灵活性：随着 APP 功能的不断完善和用户数量的增加，数据库设计需要具备良好的扩展性和灵活性，以适应未来可能的业务变化和数据增长。

1.2 范围

基于上述理由，我们编写数据库设计说明书的主要内容包括但不限于以下几个方面：

1.需求分析：详细分析 APP 的功能需求和数据需求，明确数据库需要存储哪些数据以及这些数据之间的关系。

2.概念设计：使用实体-关系图（ER 图）等工具，描述数据库中的实体、属性以及实体之间的关系，形成数据库的概念模型。

3.逻辑设计：将概念模型转换为逻辑模型，定义数据库中的表、字段、数据类型、主键、外键等，以及表的约束条件。

4.物理设计：根据具体的数据库管理系统（如 MySQL、PostgreSQL 等），设计数据库的物理存储结构，包括表的存储引擎、索引策略、分区策略等。

5.数据字典：详细描述数据库中每个表、字段的含义、数据类型、取值范围等，作为数据库开发和维护的参考。

6.安全设计：考虑数据库的安全性，设计用户权限管理、数据加密、数据备份与恢复等策略。

7.性能优化：分析数据库的查询性能，设计索引、缓存、分片等策略，以提高数据库的响应速度和吞吐量。

1.3 参考资料

https://blog.csdn.net/weixin_41039677/article/details/137559715?fromshare=blogdetail&sharetype=blogdetail&sharerId=137559715&sharerefer=PC&sharesource=weixin_74078744&sharefrom=from_link ---数据库说明书设计模板

2 数据库环境说明

2.1 系统环境

- 操作系统：Windows10
- 数据库管理系统：MySQL8.0

2.2 设计工具

- 数据库设计工具：Navicat Premium 16
- ER 图工具：Navicat Premium 16

2.3 编程工具

- 后端开发：django
- 前端开发：Vue.js
- 集成开发环境（IDE）：Pycharm

2.4 数据库详细配置

数据库名称：travel_recommendations

数据表设计：

- 用户表（users）：存储用户信息，如用户 ID、用户名、密码、手机号等。

- 旅行攻略表（travel_plans）：存储旅行攻略信息，如攻略 ID、用户 ID（外键）、攻略标题、攻略内容（可以是链接或文本）、创建时间。
- 旅行路线表（travel_routes）：存储旅行路线信息，如路线 ID、用户 ID（外键）、起点、终点、途经点、路线描述、创建时间等。
- 天气信息表（weather）：存储目的地天气信息，如目的地 ID（可以是与旅行路线或攻略相关联的外键）、日期、天气状况、温度等。
- 景点信息表（attractions）：存储景点信息，如景点 ID、景点名称、地址、开放时间、门票价格等。
- 用户收藏表（favorites）：存储用户收藏的攻略、路线或景点信息，如收藏 ID、用户 ID（外键）、被收藏 ID（可以是攻略、路线或景点的 ID）等。

索引设计：在常用的查询字段上创建索引，以提高查询效率。例如，在用户 ID、攻略 ID、路线 ID 等字段上创建主键索引，以及在用户名、景点名称等字段上创建唯一索引。

数据备份与恢复：定期备份数据库数据，以防止数据丢失。同时，制定数据恢复计划，以便在数据丢失或损坏时能够迅速恢复。

安全性配置：使用 HTTPS 协议进行数据传输，确保数据安全。同时，对数据库进行加密处理，以保护用户信息的隐私。

3 数据库命名规则

3.1 前缀约定

1.表名：tbl_

2.视图：view_

3.存储过程：sp_

4.函数：sp_

5.索引：idx_（对于非主键索引）

6.主键索引：无需额外前缀，直接命名为 PK_表名

7.外键：fk_表名_引用表名

8.唯一约束：uq_表名_字段名

9.检查约束：chk_表名_字段名

10.默认值约束：dflt_表名_字段名（如适用）

3.2 命名风格

1.使用小写字母和下划线（snake_case）进行命名。

2.名称应具有描述性，清晰表达其用途。

3.避免使用缩写，除非它们是广泛接受和理解的。

3.3 表名

1.tbl_users：存储用户信息。

2.tbl_destinations：存储旅游目的地信息。

3.tbl_travel_guides：存储旅行攻略信息。

4.tbl_routes：存储旅行路线信息。

5.tbl_weather：存储天气信息。

3.4 字段名

1.user_id：用户 ID（主键）。

2.username：用户名。

3.email：用户邮箱。

4.destination_id：目的地 ID（主键）。

5.destination_name：目的地名称。

6.guide_id：攻略 ID（主键）。

7.guide_url：攻略链接。

8.route_id：路线 ID（主键）。

9.route_name：路线名称。

10.start_point：路线起点。

11.end_point：路线终点。

12.weather_id：天气 ID（主键，可能与目的地 ID 关联）。

13.destination_weather：目的地天气信息。

14.timestamp：记录时间戳。

3.5 索引命名

1.PK_tbl_users：用户表主键索引。

2.idx_tbl_destinations_name：目的地名称的非主键索引。

3.fk_tbl_travel_guides_user：攻略表的外键，引用用户表。

4.uq_tbl_routes_name：路线名称的唯一约束。

3.6 视图命名

1.view_user_travel_guides：显示用户收藏的旅行攻略视图。

2.view_destination_weather：显示目的地当前天气信息的视图。

3.7 存储过程和函数命名

存储过程名称	功能描述	输入参数	输出结果	备注
sp_generate_travel_guide	自动生成旅行攻略	用户 ID, 攻略模板 ID	攻略详细信息	根据用户喜好和历史数据生成个性化攻略
sp_plan_optimal_route	规划最优旅行路线	用户 ID, 起始点, 终点	路线详细信息	考虑交通、天气等因素
sp_update_user_profile	更新用户个人信息	用户 ID, 新的个人信息	操作结果	确保数据一致性和安全
sp_check_inventory	检查目的地资源库存	目的地 ID	库存状态	用于景点门票和周边资源的实获取

4 逻辑设计

以下是一个简化的实体-关系图（ERD）：

4.1 实体（Entities）

1. 用户（Users）
    1. 属性：user_id（主键）、username、phone_number、password（加密存储）、created_at、updated_at
2. 旅行攻略（TravelGuides）
    1. 属性：guide_id（主键）、user_id（外键，引用 Users）、guide_url、guide_name、created_at、updated_at
3. 目的地（Destinations）
    1. 属性：destination_id（主键）、destination_name、description、location、created_at、updated_at、weather（实时获取）
4. 旅行路线（TravelRoutes）
    1. 属性：route_id（主键）、user_id（外键，引用 Users）、route_name、description、created_at、updated_at
    2. 关系：destination_ids（多值属性，存储该路线经过的目的地 ID 列表，或使用关联表）

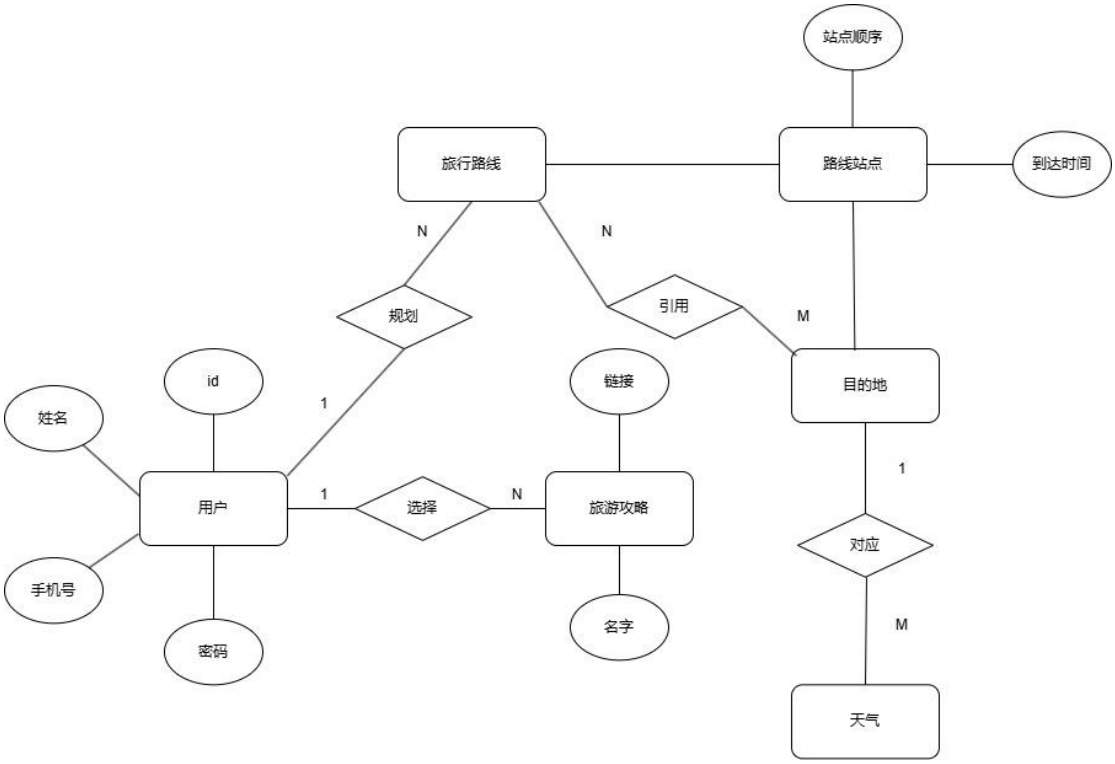
5. 路线站点（RouteStops，可选，如果需要详细站点信息）

    1. 属性：stop_id（主键）、route_id（外键，引用 TravelRoutes）、destination_id（外键，引用 Destinations）、sequence（站点顺序）、arrival_time、departure_time
    2. 若无详细站点信息，直接在 TravelRoutes 中通过 destination_ids 存储路线。

4.2 关系（Relationships）

- Users 与 TravelGuides 之间是 1:N 的关系，即一个用户可以创建多个旅行攻略。
- Users 与 TravelRoutes 之间也是 1:N 的关系，即一个用户可以规划多个旅行路线。
- Destinations 可以被多个 TravelRoutes 引用，形成 M:N 的关系，但在这个设计中，我们通过 `destination_ids` 在 TravelRoutes 中直接存储了这种关系（或使用 RouteStops 表）。
- Weather 与 Destinations 之间是 1:M 的关系，但考虑到天气可能随时间变化，我们使用复合主键来存储每个目的地在不同时间点的天气信息。

4.3 主要信息 E-R 图



4.4 注意

- `destination_ids` 字段在 TravelRoutes 表中是一个逗号分隔的字符串（不推荐，因为违反了数据库设计的第一范式），或者后续可能选择使用一个关联表（如 `RouteDestinations`）来存储这种多对多的关系。
- RouteStops 表是可选的，它提供了旅行路线中每个站点的详细信息。如果不需要这种详细信息，可以省略该表，并在 TravelRoutes 中直接通过 `destination_ids` 来规划路线。

4.5 简化后的 ERD 描述

- Users：用户信息

- TravelGuides：旅行攻略，由用户创建，包含攻略链接和名称。
- Destinations：旅游目的地信息。
- TravelRoutes：旅行路线，由用户规划，包含路线名称和描述，以及经过的目的地列表。
- RouteStops：旅行路线中的站点信息，包括到达和离开时间。

5 物理设计

5.1 数据库表结构

库名	表明	功能说明
travel	tbl_users	存储用户信息
travel	tbl_travel_guides	存储旅行攻略信息
travel	tbl_destinations	存储旅游目的地信息
travel	tbl_routes	存储旅行路线信息，包括经过的目的地列表（使用关联表）
travel	tbl_route_stops	存储旅行路线的详细站点信息（可选）
travel	tbl_weather	注意：此表在此设计中未直接使用，因为天气信息通常实时获取，但如需存储历史天气，可添加此表

5.2.1 表 tbl_users

字段名	数据类型	约束/索引	说明
user_id	INT AUTO_INCREMENT	PK_tbl_users	用户 ID（主键）
username	VARCHAR(50)	UNIQUE	用户名
phone_number	VARCHAR(20)	UNIQUE, chk_tbl_users_phone	用户手机号，需唯一，且格式正确
password	VARCHAR(255)		用户密码（加密存储）
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	创建时间
updated_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	更新时间

检查约束：chk_tbl_users_phone 用于确保手机号格式正确（具体实现依赖于数据库系统）。

5.2.2 表 tbl_travel_guides

字段名	数据类型	约束/索引	说明
guide_id	INT AUTO_INCREMENT	PK_tbl_travel_guides	攻略 ID（主键）
user_id	INT	fk_tbl_travel_guides_user	用户 ID（外键）
guide_url	VARCHAR(255)		攻略链接
guide_name	VARCHAR(100)	UNIQUE	攻略名称
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	创建时间
updated_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	更新时间

外键：fk_tbl_travel_guides_user 引用 tbl_users 表的 user_id。

5.2.3 表 tbl_destinations

字段名	数据类型	约束/索引	说明
destination_id	INT AUTO_INCREMENT	PK_tbl_destinations	目的地 ID（主键）
destination_name	VARCHAR(100)	UNIQUE, idx_tbl_destinations_name	目的地名称，需唯一
description	TEXT		目的地描述
location	VARCHAR(255)		目的地位置
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	创建时间
updated_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	更新时间

非主键索引：idx_tbl_destinations_name 用于加速目的地名称的查询。

5.2.4 表 tbl_routes

字段名	数据类型	约束/索引	说明

字段名	数据类型	约束/索引	说明
route_id	INT AUTO_INCREMENT	PK_tbl_routes	路线 ID（主键）
user_id	INT	fk_tbl_routes_user	用户 ID（外键）
route_name	VARCHAR(100)	UNIQUE, idx_tbl_routes_name	路线名称，需唯
description	TEXT		路线描述
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	创建时间
updated_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	更新时间

外键：fk_tbl_routes_user 引用 tbl_users 表的 user_id。

非主键索引：idx_tbl_routes_name 用于加速路线名称的查询。

注意：此表不包含 `destination_ids` 字段，因为我们将使用关联表来存储路线和目的地的关系。

5.2.5 表 tbl_route_stops

字段名	数据类型	约束/索引	说明
stop_id	INT AUTO_INCREMENT	PK_tbl_route_stops	站点 ID（主键）
route_id	INT	fk_tbl_route_stops_route	路线 ID（外键）
destination_id	INT	fk_tbl_route_stops_destination	目的地 ID（外键）
sequence	INT	UNIQUE(route_id, sequence), idx_tbl_route_stops_sequence	站点顺序，在路线内唯一
arrival_time	TIME		到达时间
departure_time	TIME		出发时间

外键：

- fk_tbl_route_stops_route 引用 tbl_routes 表的 route_id。
- fk_tbl_route_stops_destination 引用 tbl_destinations 表的 destination_id。

  复合唯一约束：(route_id, sequence) 确保同一路线内站点顺序的唯一性。

  非主键索引：idx_tbl_route_stops_sequence 用于加速按站点顺序的查询。

6 数据规划

6.1 标识与命名约定

- 数据库版本使用“主版本号.次版本号.修订号”的格式进行标识，例如“1.0.0”。
- 每次重大更新或功能添加时，主版本号增加；次版本号在添加新功能但不改变现有功能时增加；修订号用于修复错误或进行小调整。

6.2 内存与外存设计

6.2.1 内存安排

- 索引区：为常用查询字段建立 B+树索引，存储在内存中以加快访问速度。
- 缓冲区：设计 LRU（最近最少使用）缓冲区策略，缓存常用数据块，减少磁盘 I/O。

6.2.2 外存设备及空间组织

- 使用 SSD 作为外存设备以提高读写速度。
- 索引区：在 SSD 上建立索引文件，利用磁盘顺序读写优势，优化索引访问。
- 数据块：根据数据访问频率和大小，将数据划分为固定大小的数据块（如 4KB），便于管理和高效读取。

6.3 访问数据方式

- SQL 查询：优先使用参数化查询和预处理语句，避免 SQL 注入并提高查询效率。
- 批量操作：对于大量数据插入、更新或删除，使用批量操作减少事务开销。
- 索引利用：确保查询语句能够充分利用索引，避免全表扫描。

6.4 表空间设计

6.4.1 表空间设计原则

- 基于性能（读写速度、I/O 效率）和业务需求（数据分类、访问频率）设计表空间。

6.4.2 数据文件设计

- 命名规则：基于表空间名称和创建日期命名，例如 user_data_20230101.dbf。
- 大小设计：根据预估数据量、增长速度和存储性能综合考虑，设置合理的初始大小和自动扩展策略。

6.5 表、索引分区设计

6.5.1 逻辑与物理设计

- 逻辑设计：根据业务逻辑和查询需求，将表划分为不同的分区，如按时间（年月）、地域或业务类型。

- 物理设计：将逻辑分区映射到物理分区，确保数据在磁盘上的分布合理，减少碎片和热点。

6.5.2 分区原则

- 提高查询性能：通过分区减少全表扫描的范围。
- 平衡负载：将热点数据分散到不同分区，避免单一分区成为瓶颈。
- 便于管理：按业务模块或数据类型分区，简化数据备份和恢复。

6.5.3 表与索引的表空间

- 根据访问频率和性能需求，将表和索引分配到不同的表空间，以优化 I/O 性能。

6.6 优化方法

6.6.1 时空效率分析

- 识别瓶颈：通过监控和分析 SQL 执行计划，找出性能瓶颈。
- 确定优先级：根据业务影响程度和修复成本，确定优化对象的优先级。

6.6.2 折衷方案

- 在优化对象之间存在对抗时（如提高查询速度与增加存储成本），通过权衡利弊，制定折衷方案。

6.6.3 具体措施

- 优化环境参数：调整数据库的内存分配、并发连接数等参数，以适应业务需求。
- 反规范化：在必要时，对表格进行反规范化以减少查询复杂度，但需注意数据一致性和冗余问题。
- 索引优化：创建合适的索引，避免不必要的索引，保持索引的更新效率。
- 查询优化：重写低效的 SQL 查询，使用更高效的查询算法和技巧。

7 安全性设计

7.1 防止用户直接操作数据库

目的：防止未经授权的用户或恶意行为者直接访问或修改数据库。

措施：

- 应用层访问：所有数据库操作应通过应用服务器进行，不允许用户直接连接到数据库服务器。应用服务器将作为中间层，处理所有来自用户的请求，并对数据库执行相应的操作。
- 防火墙规则：配置数据库服务器的防火墙，仅允许应用服务器和其他必要的内部服务访问数据库端口。
- 数据库访问控制列表（ACLs）：设置数据库访问控制列表，限制只有特定的 IP 地址或子网才能访问数据库。

7.2 用户账号加密处理

目的：保护用户账号信息，防止泄露或被恶意利用。

措施：

- 密码存储：用户密码应使用强加密算法（如 bcrypt、Argon2 等）进行哈希处理，并存储哈希值而非明文密码。

- 密码传输：在客户端和服务器之间传输密码时，应使用 HTTPS 或其他安全协议进行加密，防止密码在传输过程中被截获。
- 密码策略：实施强密码策略，要求用户创建包含大小写字母、数字和特殊字符的复杂密码，并定期更换密码。

7.3 角色与权限控制

目的：确保只有授权用户能够执行特定的数据库操作。

措施：

- 角色定义：在数据库中定义不同的角色，如管理员、普通用户等，并为每个角色分配特定的权限。
- 权限分配：根据业务需求，为不同的角色分配不同的数据库操作权限。例如，管理员可能具有创建、修改和删除数据库对象的权限，而普通用户可能只有读取数据的权限。
- 最小权限原则：为每个用户分配完成其任务所需的最小权限，以减少潜在的安全风险。
- 审计和监控：实施数据库审计和监控机制，记录用户对数据库的访问和操作行为，以便在发生安全事件时进行追踪和调查。

7.4 额外安全措施

- 多因素认证：为增加安全性，考虑实施多因素认证（MFA），如结合密码和短信验证码、指纹识别等。

- 定期备份：定期备份数据库，以防数据丢失或损坏。备份数据应存储在安全的位置，并定期进行恢复测试。
- 安全更新和补丁：及时关注数据库系统的安全更新和补丁，确保系统免受已知漏洞的攻击。

8 数据库的备份策略和方式

8.1 备份策略

8.1.1 备份频率

- 全量备份:建议每天进行一次全量备份,以确保数据的完整性和可恢复性。备份时间可以安排在凌晨的低峰时段，如 2:00-4:00，以减少对业务的影响。
- 增量/差异备份：在全量备份的基础上，每天进行多次增量或差异备份，以捕捉全量备份后的数据变化。增量备份记录自上次备份以来发生变化的数据块，而差异备份则记录自上次全量备份以来发生变化的数据。这些备份可以每 4 小时或每 6 小时进行一次,具体取决于数据变化的频率和重要性。

8.1.2 备份时长与所需时间

- 全量备份：根据数据库的大小和硬件性能，全量备份可能需要数分钟到数小时不等。对于大型数据库，可能需要使用压缩和并行处理技术来缩短备份时间。
- 增量/差异备份：由于只备份变化的数据，增量/差异备份通常比全量备份更快，所需时间也较短。

8.1.3 备份保留周期

- 备份数据应至少保留一个月，对于重要数据或法规要求，可能需要保留更长的时间。过期的备份数据应及时删除或归档，以节省存储空间。

8.2 备份方式

8.2.1 备份类型

- 逻辑备份：逻辑备份是通过导出数据库的结构和数据来实现的，通常使用 SQL 脚本进行。逻辑备份易于迁移和恢复，但可能受到数据库大小和复杂性的限制。
- 物理备份：物理备份是直接复制数据库的存储文件（如数据文件、日志文件等），通常使用数据库自带的备份工具（如 MySQL 的 mysqlbackup、PostgreSQL 的 pg_basebackup 等）或第三方工具进行。物理备份速度更快，恢复时间更短，但可能需要额外的存储空间来存储备份文件。

8.2.2 备份工具

MySQL：可以使用 mysqldump 进行逻辑备份，或使用 mysqlbackup 进行物理备份。

8.2.3 注意事项

- 备份操作应自动化进行，以减少人为错误和遗漏。
- 备份数据应存储在安全的位置，如加密的存储设备或远程备份服务器。
- 定期对备份数据进行验证和恢复测试，以确保备份数据的可用性和准确性。
- 在进行重大数据库操作（如升级、迁移等）之前，应提前进行全量备份。