02 DB 数据库模块函数开发

数据库模块函数开发

项目背景与需求

在本项目中，我们使用 SQLite3 数据库与 C++ 相结合，完成宿舍管理系统的数据库模块。该模块负责处理系统中的数据存储、查询、更新等任务，包括对宿舍信息、用户信息、房间使用情况等数据的管理。本部分报告主要关注于 `Database` 类的实现，该类提供了与数据库的接口，并确保与宿舍管理系统的其他模块之间的数据交互。

项目目标

1. **数据库连接与管理**：确保能够正确连接数据库并执行查询与更新操作。
2. **查询与执行操作**：支持多种形式的 SQL 操作，包括非查询操作（如插入、更新）和查询操作（带参数查询、存在性检查等）。
3. **事务与数据一致性**：确保操作完成后，数据库数据的一致性与完整性。

数据库模块结构与实现

1. `Database` 类设计

`Database` 类是与 SQLite 数据库交互的核心类。它通过 `sqlite3` 库提供的接口，负责建立和关闭数据库连接，以及执行各种数据库操作。该类包含以下重要成员和方法：

- **构造函数**：`explicit Database(const string &dbPath)`
  用于打开数据库连接，如果数据库连接失败则输出错误信息。
- **析构函数**：`~Database()`
  用于关闭数据库连接并释放资源。
- `execute` **方法**：
  执行非查询 SQL 语句（如 `INSERT`、`UPDATE`、`DELETE`）。如果数据库连接失败，方法会返回 `false` 并输出错误信息。
- `query` **方法**：
  执行查询 SQL 语句并打印结果。该方法通过 `sqlite3_step` 执行查询，并通过 `sqlite3_column_text` 获取每一列的数据。
- `queryExists` **方法**：
  用于执行存在性检查的查询，判断某个条件是否符合，返回布尔值结果。

- `queryWithParams` **方法**：
  用于执行带有参数的查询。通过 `sqlite3_bind_text` 绑定查询参数，确保 SQL 注入防范。
- `executeWithParams` **方法**：
  用于执行带参数的非查询操作。与 `queryWithParams` 方法相似，通过绑定参数执行操作，适用于 `INSERT`、`UPDATE` 等语句。
- `updateRoomStatus` **方法**：
  更新宿舍房间的占用状态。通过 SQL 子查询统计 `student_rooms` 表中的学生数量，并更新 `rooms` 表中的 `occupied` 字段。
- `getQueryResult` **方法**：
  获取查询结果中的某一列值，通常用于获取查询的结果。

2. 数据库操作流程

- **打开数据库**：
  在 `Database` 类的构造函数中，通过 `sqlite3_open` 打开 SQLite 数据库。如果连接失败，会输出错误信息。
- **执行 SQL 语句**：
  在执行 SQL 时，使用 `sqlite3_exec` 处理非查询类型的 SQL 语句，如插入、删除、更新等。如果是查询类型的操作（如 `SELECT`），使用 `sqlite3_prepare_v2` 编译 SQL 语句，并通过 `sqlite3_step` 执行。查询过程中，根据返回结果逐行处理数据。
- **参数绑定**：
  对于带有动态参数的 SQL 语句，使用 `sqlite3_bind_text` 将参数绑定到 SQL 语句中，避免 SQL 注入。

3. 数据库查询与更新

- **查询操作**：
  通过 `query` 方法，查询结果会以列为单位进行打印。每次查询返回的数据会显示在控制台，方便管理员查看。
- **带参数查询**：
  使用 `queryWithParams` 方法时，我们将 SQL 查询中的参数进行绑定。这样能确保用户输入的数据安全，并防止恶意 SQL 注入。
- **更新房间状态**：
  通过 `updateRoomStatus` 方法，系统能自动更新每个房间的占用情况。该方法通过查询 `student_rooms` 表中每个房间的学生人数，并更新 `rooms` 表中的对应字段。

4. 错误处理

数据库操作过程中，遇到的错误通过输出错误信息来提示用户。例如，如果 SQL 执行失败，`sqlite3_errmsg` 会返回详细的错误信息，以便开发人员定位问题。

遇到的问题与解决方案

1. 数据库连接失败

在最初的开发阶段，我们遇到数据库无法打开的错误。通过检查路径和权限，发现由于数据库文件路径错误导致无法正确连接数据库。最终，我们改进了数据库路径的管理，确保路径是动态计算的，并通过日志记录了数据库连接的详细错误信息。

2. SQL 查询语法错误

在多次调试过程中，执行的 SQL 查询语句可能存在语法错误，导致查询结果为空或返回错误。解决此问题时，我们采用了逐步检查 SQL 语句的方式，通过输出调试信息来验证每个查询的正确性。

3. 参数绑定的问题

在处理带参数的 SQL 查询时，初期出现了绑定参数失败的情况。通过深入阅读 SQLite3 的文档，我们解决了使用 `sqlite3_bind_text` 绑定字符串类型参数的问题，并确保每个绑定的参数都能正确映射到 SQL 语句中。

收获与反思

1. **SQL 注入防范**：
   在处理数据库时，确保 SQL 注入防范是至关重要的。在项目开发中，我们通过绑定参数（而不是直接拼接字符串）来有效防止 SQL 注入攻击，从而提高了系统的安全性。
2. **调试技巧**：
   在调试过程中，我们学会了如何通过 `sqlite3_errmsg` 获取详细的错误信息，帮助快速定位问题。对于复杂查询，逐步拆解查询并打印结果是高效的调试方式。
3. **数据库优化**：
   在更新房间状态时，使用了 `UPDATE` 和 `SELECT` 语句的结合。这个操作需要对数据库进行一定的优化，例如使用子查询来避免多次查询，确保系统性能。
4. **项目整体设计**：
   通过设计一个数据库模块，我们将与数据库交互的操作进行了封装，降低了其他模块与数据库的耦合度。这个设计使得数据库操作的逻辑更加清晰，易于维护和扩展。

总结

本数据库模块的实现确保了宿舍管理系统能够高效、稳定地与数据库交互。通过对查询、执行等操作的封装，我们提高了代码的可读性和可维护性。在开发过程中，我们不仅掌握了 SQLite 的使用，还学会了如何高效地处理带参数的 SQL 查询和避免 SQL 注入等常见安全问题。