记录编号：_____

《嵌入式应用开发》
Harmony 关系型数据库案例实战实验指导手册

版本：V 1.0

目录

（一）实验目的

1. 掌握数据库和表的创建。
2. 掌握数据库表数据的增删改查。
3. 掌握数据库的删除。

（二）实验涉及知识点

1. relationalStore

（三）实验准备

参考开发环境：

- 操作系统：Window 10
- 开发工具：DevEco Studio 3.1.1
- HarmonyOS SDK 版本：API version 9 及以上版本
- 开发语言：ArkTS
- 内存：8G 及以上

（四）实验内容及课时准备

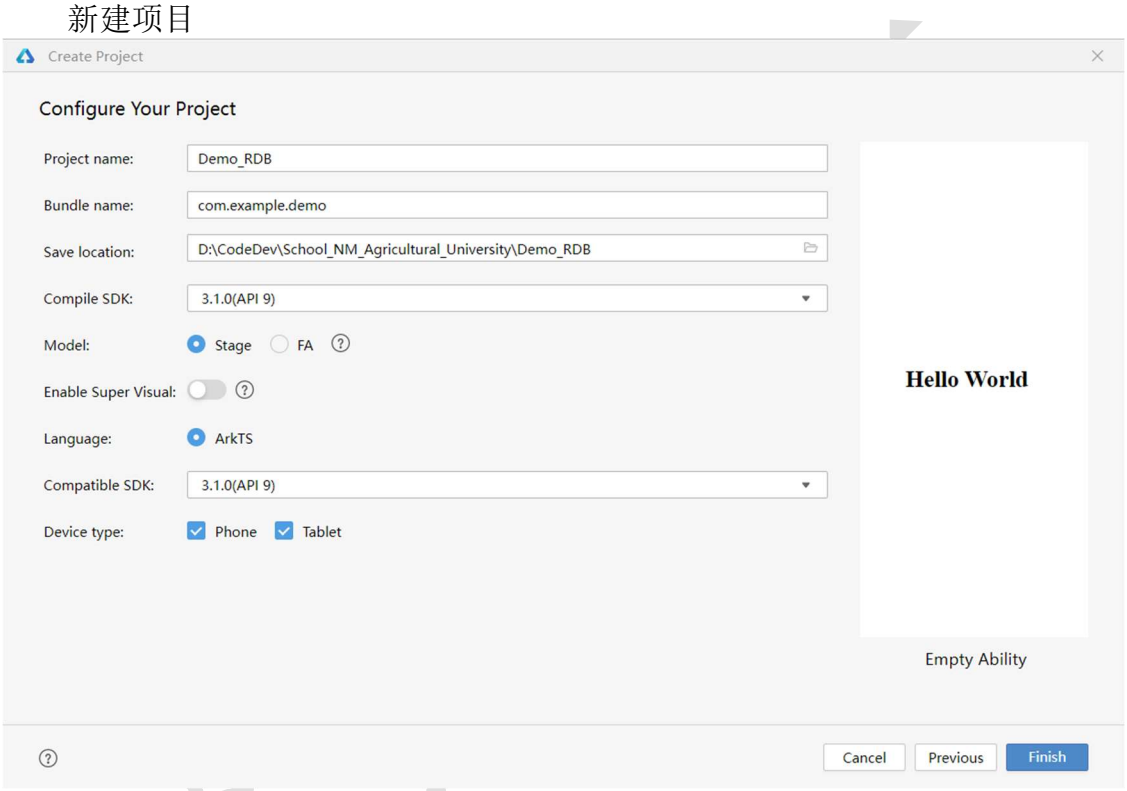
序号	实验内容	实验课时	对应核心知识点	对应实验目标点
1	数据库和表操作	1	1	1、2、3
	总计	1		

（五）详细实验过程

1 数据库和表操作实战演练

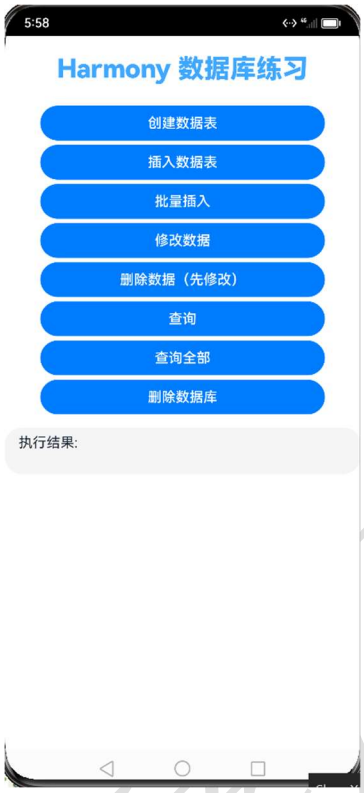
1.1 准备工作

新建项目



1.2 搭建项目页面框架

示意图如下：



UI 布局代码如下

```
build() {
  Scroll() {
    Column({space:5}){
      MyTitle()
      Button("创建数据表").myButton().onClick(()=>{...})
      Button("插入数据表").myButton().onClick(()=>{...})
      Button("批量插入").myButton().onClick(()=>{...})
      Button("修改数据").myButton().onClick(()=>{...})
      Button("删除数据（先修改）").myButton().onClick(()=>{...})
      Button("查询").myButton().onClick(()=>{...})
      Button("查询全部").myButton().onClick(()=>{...})
      Button("删除数据库").myButton().onClick(()=>{...})
      TextArea({ text: "执行结果:\r\n" + this.message }).margin({top:10})
    }.width("100%").alignItems(HorizontalAlign.Center)
  }
}
```

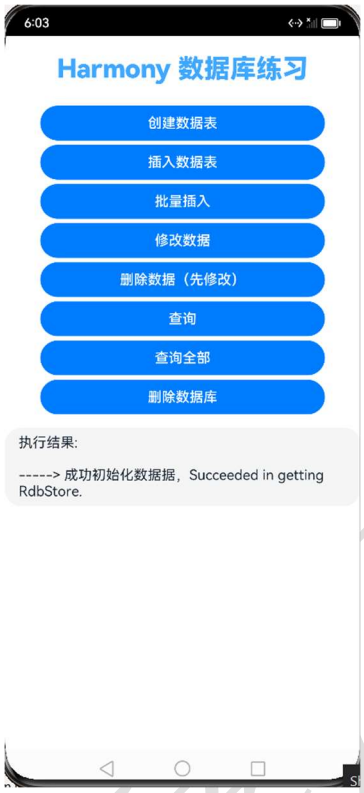
1.3 初始化数据库

如下编写代码进行数据库的初始化，我们在组件生命周期函数

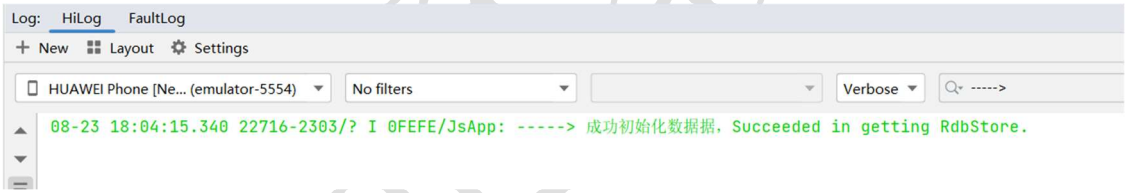
aboutToAppear() 中编写初始化数据库的代码，这样就可以在项目启动的时候自动进行初始化。

```
const STORE_CONFIG = {
  name: 'my.db', // 数据库文件名
  securityLevel: relationalStore.SecurityLevel.S1 // 数据库安全级别
};

//....
@State rdbStore: relationalStore.RdbStore = null
  @State message: string = ""

  aboutToAppear() {
    relationalStore.getRdbStore(getContext(), STORE_CONFIG, (err, store) => {
      if (err) {
        this.message += `${MyTAG} 初始化数据库失败，Failed to get RdbStor
e. Code:${err.code}, message:${err.message}`
        console.error(this.message);
        return;
      }
      this.message += `${MyTAG} 成功初始化数据据，Succeeded in getting Rd
bStore.`
      console.info(this.message);
      this.rdbStore = store
      // 创建数据表
      // 这里执行数据库的增、删、改、查等操作
    })
  }
```

运行 APP 后，APP 界面如下。



日志控制台输出如下。



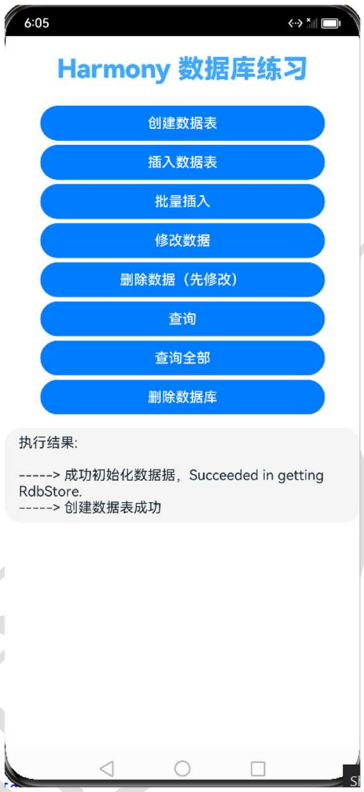
1.4 创建数据表

创建数据表的代码如下，由于 Harmony 的关系型数据库基于 SQLite，我们遵循 SQLite 语法编写 SQL 语句，然后通过 rdbStore 执行该 SQL 语句完成数据表的创建。

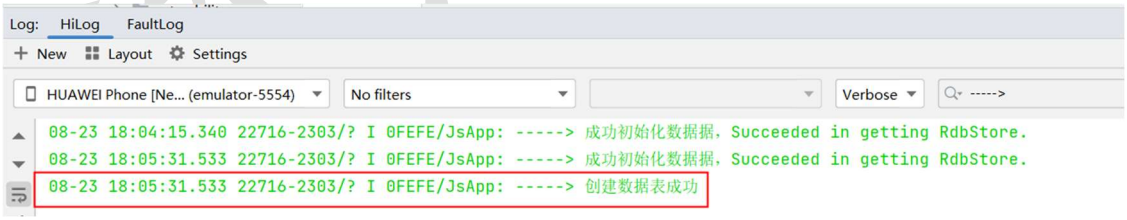
```
const SQL_CREATE_TABLE = 'CREATE TABLE IF NOT EXISTS student (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, age INTEGER, score DOUBLE)';

//...
Button("创建数据表").myButton().onClick(() => {
```

```
            this.rdbStore.executeSql(SQL_CREATE_TABLE)
            this.message += `${MyTAG} 创建数据表成功`
            console.info(this.message);
        })
```

点击“创建数据表”按钮，运行 APP 效果如下。



控制台输出如下。



1.5 查询

（1）条件查询

```
Button("查询").myButton().onClick(() => {
    let predicates = new relationalStore.RdbPredicates('student');
    predicates.equalTo('name', 'Jack');
```

```
        this.rdbStore.query(predicates, ['id', 'name', 'age', 'score'], (err, resultS
et) => {
          if (err) {
            this.message += `${MyTAG}条件查询失败. Code:${err.code}, messa
ge:${err.message}`
            console.error(this.message);
            return;
          }
          this.parseData(resultSet)
        })
      })
```

（2）查询全部

```
    Button("查询全部").myButton().onClick(() => {
      let predicates = new relationalStore.RdbPredicates('student');
      this.rdbStore.query(predicates, ['id', 'name', 'age', 'score'], (err, resultSe
t) => {
        if (err) {
          this.message += `${MyTAG}查询全部失败. Code:${err.code}, messag
e:${err.message}`
          console.error(this.message);
          return;
        }
        this.parseData(resultSet)
      })
    })
```

1.6 插入数据

插入数据表的代码如下。

```
Button("插入数据表").myButton().onClick(() => {
    const valueBucket = {
      'name': 'Jack',
      'age': 18,
```

```
        'score': 80
      };
      this.rdbStore.insert('student', valueBucket, (err, rowId) => {
        if (err) {
          this.message += `${MyTAG} 插入数据失败. Code:${err.code}, messa
${err.message}`
          console.error(this.message)
          return;
        }
        this.message += `${MyTAG} 插入数据成功. rowId:${rowId}`
        console.info(this.message)
      })
    })
```

重新运行 APP，依次执行：

- 常见数据表
- 插入数据表
- 查询全部

运行结果如下。

1.7 批量插入

批量插入数据的代码如下，我们通过 for 循环模拟一些数据进行插入。

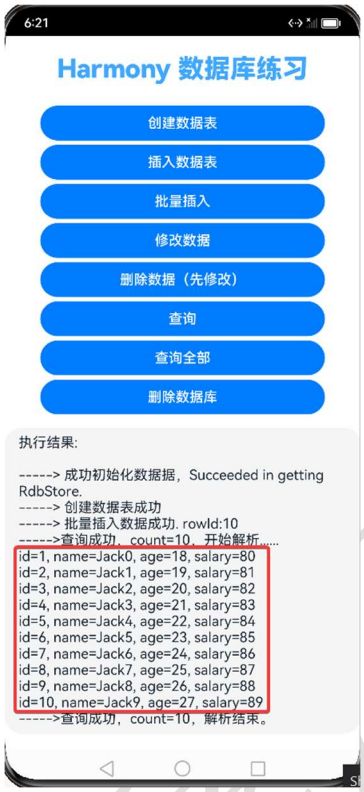
```
Button("批量插入").myButton().onClick(() => {
        let valueBucketList = []
        for (let i = 0;i < 10; i++) {
          let valueBucket = {
            'name': 'Jack' + i,
            'age': 18 + i,
            'score': 80 + i
          };
          valueBucketList[i] = valueBucket
        }
```

```
        this.rdbStore.batchInsert("student", valueBucketList, (err, rowId) => {
          if (err) {
            this.message += `${MyTAG} 批量插入数据失败. Code:${err.code}, message:${err.message}`
            console.error(this.message)
            return;
          }
          this.message += `${MyTAG} 批量插入数据成功. rowId:${rowId}`
          console.info(this.message)
        })
      })
```

重新运行 APP，依次执行：

- 创建数据表
- 批量插入
- 查询全部

运行效果如下图所示。



1.8 修改数据

修改数据的代码如下。

```
Button("修改数据").myButton().onClick(() => {
        const newValueBucket = {
          'name': 'Rose',
          'age': 20,
          'score': 93
        };
        //构建查询条件
        let predicates = new relationalStore.RdbPredicates('student'); // 创建表'student'的predicates
        predicates.equalTo('name', 'Jack'); // 匹配表'student'中'name'为'Jack'的字段
        this.rdbStore.update(newValueBucket, predicates, (err, rows) => {
          if (err) {
            this.message += `${MyTAG} 更新数据失败. Code:${err.code}, message:${err.message}`
```

```
            console.error(this.message)
            return;
          }
          this.message += `${MyTAG} 更新数据成功. row count: ${rows}`
          console.info(this.message)
        })
      })
```

重新运行 APP，依次执行：

- 创建数据表
- 插入数据表
- 查询全部
- 修改数据
- 查询全部

运行效果如下图所示。

1.9 删除数据

删除数据的代码如下。

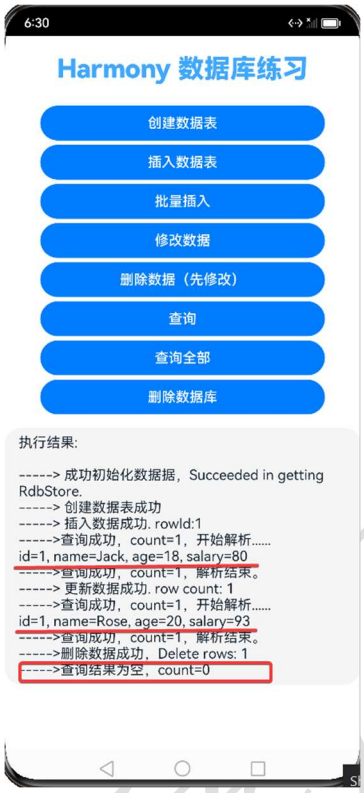
```
  Button("删除数据（先修改）").myButton().onClick(() => {
      //构建删除条件
      let predicates = new relationalStore.RdbPredicates('student');
      predicates.equalTo('name', 'Rose');

      this.rdbStore.delete(predicates, (err, rows) => {
        if (err) {
          this.message += `${MyTAG}删除数据失败. Code:${err.code}, message:
${err.message}`
          console.error(this.message);
          return;
        }
```

```
        this.message += `${MyTAG}删除数据成功，Delete rows: ${rows}`
        console.info(this.message)
      })
    })
```

重新运行 APP，依次执行：

- 创建数据表
- 插入数据，name=jack
- 修改数据，修改后，name=Rose
- 查询全部
- 删除数据，删除 name=Rose 的记录
- 查询全部

运行效果如下图所示。



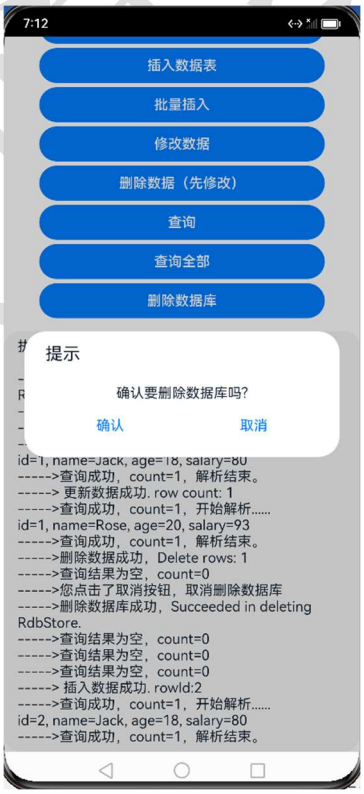
1.10 删除数据库

由于删除数据库是一个很严重的操作，我们给用户一个对话框，如果用户确认删除才真正进行删库操作。删除数据库的代码如下。

```
Button("删除数据库").myButton().onClick(() => {
          AlertDialog.show({
            title: "提示",
            message: "确认要删除数据库吗?",
            primaryButton: {
              value: "确认",
              action: () => {
                relationalStore.deleteRdbStore(getContext(), 'my.db', (err) => {
                  if (err) {
                    this.message += `${MyTAG}删除数据库失败. Code:${err.code}, me
ssage:${err.message}`
                    console.error(this.message);
                    return;
```

```
                }
                this.message += `${MyTAG}删除数据库成功，Succeeded in deletin
g RdbStore.`
                console.info(this.message);
              });
            }
          },
          secondaryButton: {
            value: "取消",
            action: () => {
              this.message += `${MyTAG}您点击了取消按钮，取消删除数据库`
              console.info(this.message)
            }
          }
        })
      })
```

需要注意的是，删除数据库存在缓存，不能立即见效，删除示意图如下。



补充：

parseData 函数定义：

```
    //输出查询结果集
parseData(resultSet: relationalStore.ResultSet) {
  console.info(`${this.MyTAG}ResultSet column names:
${resultSet.columnNames}`);
  console.info(`${this.MyTAG}ResultSet column count:
${resultSet.columnCount}`);

  let count = resultSet.rowCount
  if (count == 0) {
    this.message += `$this.MyTAG}查询结果为空，count=${count}`
    console.info(this.message)
    return
  }
  this.message += `${this.MyTAG}查询成功，count=${count}，开始解
析.....`
  console.info(this.message)

  resultSet.goToFirstRow() //开始第一条
  for (let i = 0; i < count; i++) {
    let id = resultSet.getDouble(resultSet.getColumnIndex("id"))
    let name = resultSet.getString(resultSet.getColumnIndex("name"))
    let age = resultSet.getDouble(resultSet.getColumnIndex("age"))
    let score =
resultSet.getDouble(resultSet.getColumnIndex("score"))

    //输出结果看一下
    let msg = `id=${id}, name=${name}, age=${age}, salary=${score}`
    this.message += "\r\n" + msg
    console.info(this.message)
    //继续下一条
    resultSet.goToNextRow()
  }

  this.message += `${this.MyTAG}查询成功，count=${count}，解析结束。`
  console.info(this.message)
}
```

myButton 函数定义

```
    @Extend(Button) function myButton()
{
  .backgroundColor(0x0f0f0f)
  .width('100%')
  .height('10%')
  .borderRadius(5)
  .type(ButtonType.Normal)
}
```