

密级： 保密期限：

北京邮电大学

数据库系统原理实验报告



内含实验：数据查询与修改、数据库访问接口、数据库物理设计、事务与并发控制

组 号： 11

组员姓名： 李志毅、陈硕

组员学号： 2018211582、2018211584

指导老师： 杜军平

学 院： 计算机学院

2021 年 6 月 14 日

目录

| | |
|--------------------------------|----|
| 数据库实例连接及建表数据导入实验 | 3 |
| 一、实验环境 | 3 |
| 二、使用 Navicat 连接 MySQL | 3 |
| 三、数据库建表及数据库导入 | 4 |
| 四、MySQL 导入 SQL Server | 16 |
| 五、问题及解决 | 17 |
| 六、实验总结 | 17 |
| 数据查询与修改实验 | 18 |
| 一、实验目的 | 18 |
| 二、实验环境 | 18 |
| 三、实验内容 | 18 |
| 四、实验步骤 | 18 |
| 五、实验总结 | 37 |
| 数据库访问接口实验 | 38 |
| 一、环境配置 | 38 |
| 二、连接时长的获取和修改 | 38 |
| 2.1 通过 API 函数获取数据库默认连接时间 | 38 |
| 2.2 在默认时长下观察是否会超时 | 39 |
| 2.3 降低默认连接时间，观察是否超时 | 40 |
| 三、查询 | 40 |
| 四、插入 | 41 |
| 五、修改 | 43 |
| 六、删除 | 44 |
| 七、实验总结 | 46 |
| 数据库物理设计实验 | 47 |
| 一、实验目的 | 47 |
| 二、实验环境 | 47 |
| 三、实验内容 | 47 |
| 四、实验步骤 | 48 |
| 五、问题及解决 | 57 |
| 六、实验总结 | 58 |
| 事务与并发控制 | 59 |
| 一、实验目的 | 59 |
| 二、实验环境 | 59 |
| 三、实验步骤 | 59 |
| 四、实验总结 | 93 |

数据库实例连接及建表数据导入实验

一、实验环境

本实验环境为主 MySQL+Navicat，在物理设计实验中使用 SQLServer2012
实验环境配置：

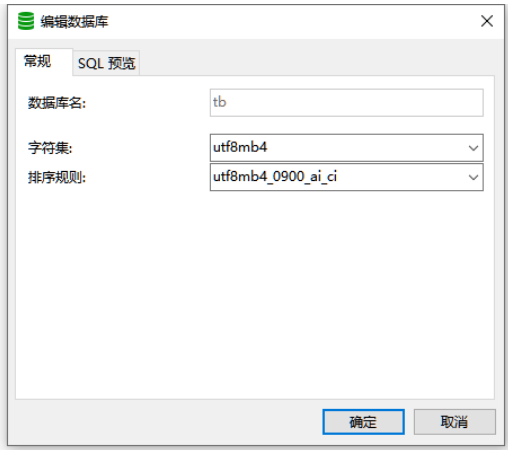
| 设备名称 | 设备型号 |
|------|----------------|
| 数据库 | MySQL 8.2 |
| 数据库 | SQLServer 2012 |

二、使用 Navicat 连接 MySQL

我们使用 Navicat 工具连接本机 MySQL 数据库，连接名为 DB，数据库名为 tb:



连接完成后，创建数据库 tb，以此数据库为实验环境，建立各个表



三、数据库建表及数据库导入

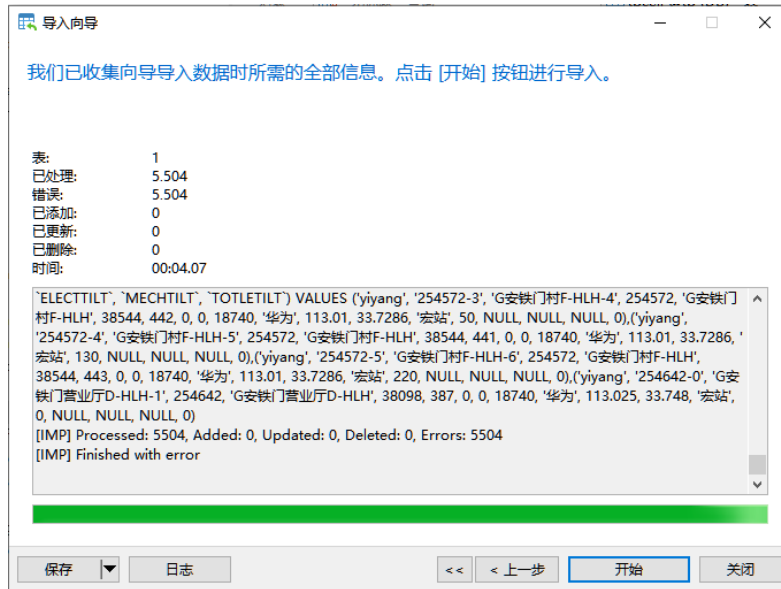
3.1 小区/基站工参表 tbCell

SQL 语句:

```
DROP TABLE IF EXISTS `tbcell`;
CREATE TABLE `tbcell` (
  `CITY` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NULL
  DEFAULT NULL COMMENT '城市/地区名称',
  `SECTOR_ID` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT
  NULL COMMENT '小区 ID',
  `SECTOR_NAME` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT
  NULL COMMENT '小区名称',
  `ENODEBID` int NOT NULL COMMENT '小区基站 ID, 所属基站 eNodeB 的标识',
  `ENODEB_NAME` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT
  NULL COMMENT '基站名称',
  `EARFCN` int UNSIGNED NOT NULL COMMENT '小区配置的频点编号',
  `PCI` int UNSIGNED NULL DEFAULT NULL COMMENT '物理小区标识',
  `PSS` int UNSIGNED NULL DEFAULT NULL COMMENT '主同步信号标识',
  `SSS` int UNSIGNED NULL DEFAULT NULL COMMENT '辅同步信号标识',
  `TAC` int UNSIGNED NULL DEFAULT NULL COMMENT '跟踪区编码',
  `VENDOR` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NULL
  DEFAULT NULL COMMENT '设备厂家',
  `LONGITUDE` decimal(10, 6) UNSIGNED NOT NULL COMMENT '小区所属基站的经度',
  `LATITUDE` decimal(10, 6) UNSIGNED NOT NULL COMMENT '小区所属基站的纬度',
  `STYLE` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NULL
  DEFAULT NULL COMMENT '基站类型',
  `AZIMUTH` int UNSIGNED NOT NULL COMMENT '小区天线方位角',
  `HEIGHT` int UNSIGNED NULL DEFAULT NULL COMMENT '小区天线高度',
  `ELECTTILT` int UNSIGNED NULL DEFAULT NULL COMMENT '小区天线电下倾角',
  `MECHTILT` int NULL DEFAULT NULL COMMENT '小区天线机械下倾角',
  `TOTLETILT` int UNSIGNED NOT NULL COMMENT '总下倾角',
  PRIMARY KEY (`SECTOR_ID`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4_general_ci ROW_FORMAT
= Dynamic;

SET FOREIGN_KEY_CHECKS = 1;
```

再从 tbcell.xlsx 文件中导入数据:



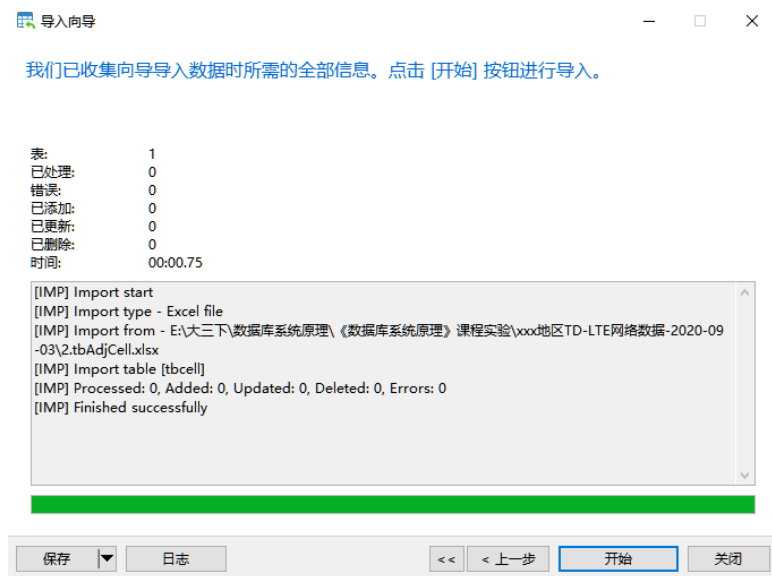
3.2 小区一阶邻区关系表 tbAdjCell

SQL 语句:

```
DROP TABLE IF EXISTS `tbadjcell`;
CREATE TABLE `tbadjcell` (
  `S_SECTOR_ID` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NOT NULL COMMENT '主小区/服务小区 ID',
  `N_SECTOR_ID` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NOT NULL COMMENT '邻小区 ID',
  `S_EARFCN` int UNSIGNED NULL DEFAULT NULL COMMENT '主小区频点',
  `N_EARFCN` int UNSIGNED NULL DEFAULT NULL COMMENT '邻小区频点',
  PRIMARY KEY (`S_SECTOR_ID`, `N_SECTOR_ID`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_general_ci ROW_FORMAT = Dynamic;

SET FOREIGN_KEY_CHECKS = 1;
```

从 tbAdjCell.xlsx 中导入数据:

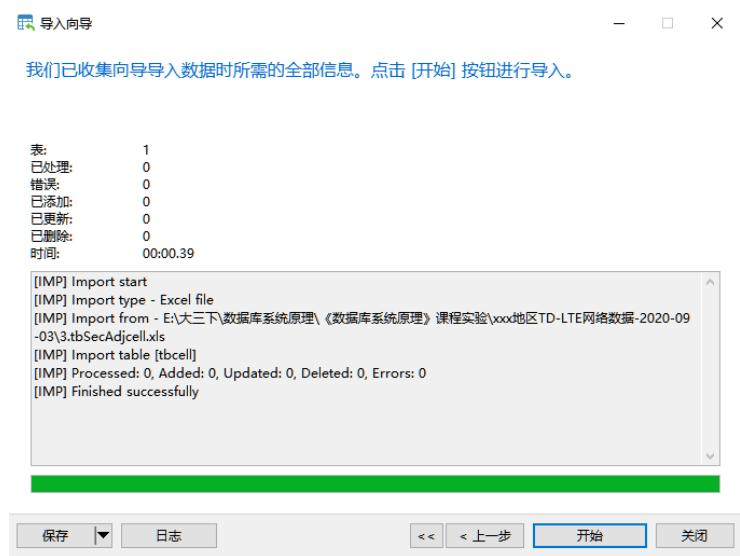


3.3 二阶（同频）邻区关系表 tbSecAdjCell

SQL 语句:

```
DROP TABLE IF EXISTS `tbsecadjcell`;  
CREATE TABLE `tbsecadjcell` (  
  `S_SECTOR_ID` varchar(50) CHARACTER SET utf8mb4 COLLATE  
utf8mb4_general_ci NOT NULL COMMENT '主小区/服务小区 ID',  
  `N_SECTOR_ID` varchar(50) CHARACTER SET utf8mb4 COLLATE  
utf8mb4_general_ci NOT NULL COMMENT '邻小区 ID',  
  PRIMARY KEY (`S_SECTOR_ID`, `N_SECTOR_ID`) USING BTREE  
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE =  
utf8mb4_general_ci ROW_FORMAT = Dynamic;  
  
SET FOREIGN_KEY_CHECKS = 1;
```

从 tbSecAdjCell.xlsx 中导入数据:



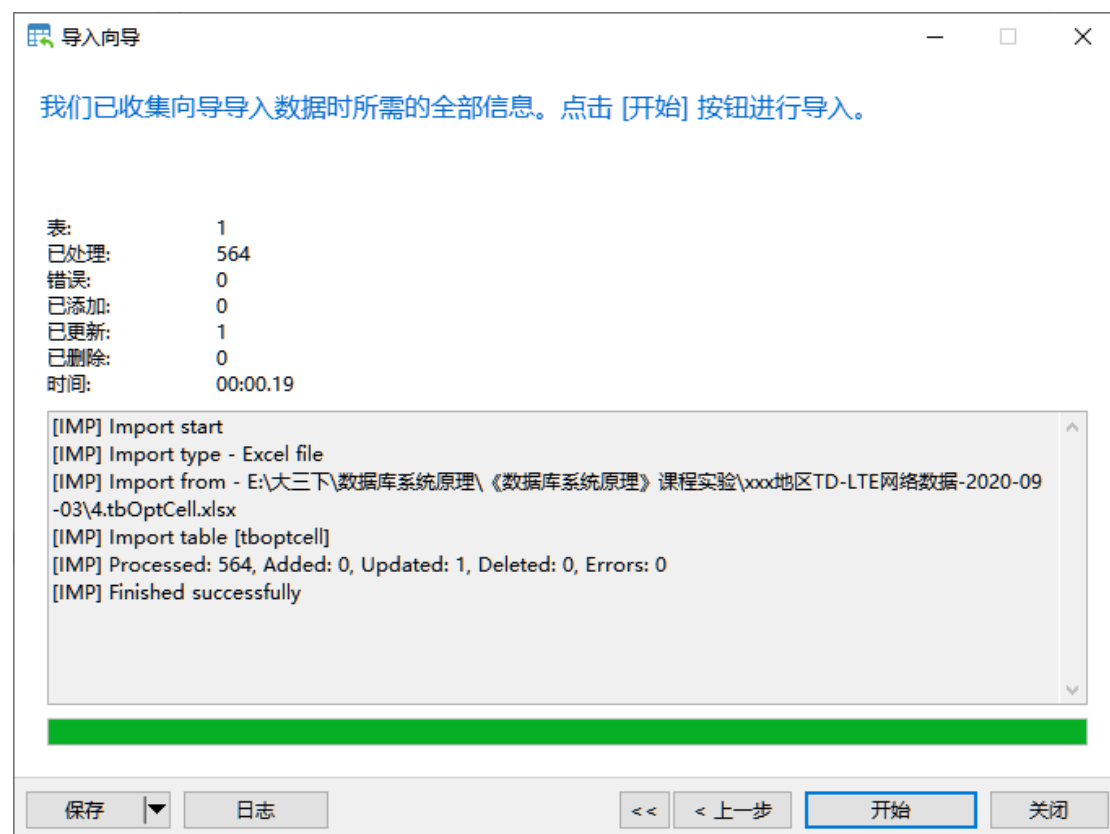
3.4 优化小区/保护带小区表 tbOptCell

SQL 语句:

```
DROP TABLE IF EXISTS `tboptcell`;
CREATE TABLE `tboptcell` (
  `SECTOR_ID` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NOT NULL COMMENT '小区 ID',
  `EARFCN` int UNSIGNED NULL DEFAULT NULL COMMENT '频点编号',
  `CELL_TYPE` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NULL DEFAULT NULL COMMENT '小区类型',
  PRIMARY KEY (`SECTOR_ID`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_general_ci COMMENT = '优化小区/保护带小区表' ROW_FORMAT =
Dynamic;

SET FOREIGN_KEY_CHECKS = 1;
```

从 tbOptCell.xlsx 中导入数据:



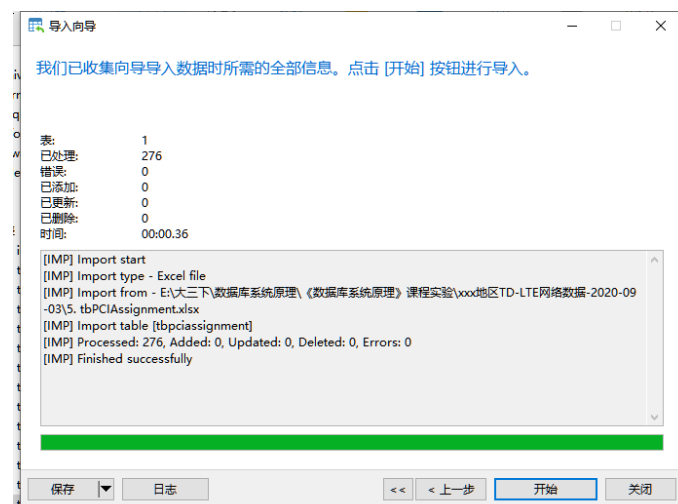
3.5 小区 PCI 优化调整结果表 tbPCIAssignment

SQL语句:

```
DROP TABLE IF EXISTS `tbpciassignment`;
CREATE TABLE `tbpciassignment` (
  `ASSIGN_ID` smallint UNSIGNED NOT NULL COMMENT '分配方案编号',
  `EARFCN` int UNSIGNED NULL DEFAULT NULL COMMENT '小区频点编号',
  `SECTOR_ID` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NOT NULL COMMENT '小区 ID',
  `SECTOR_NAME` varchar(200) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NULL DEFAULT NULL COMMENT '小区名称',
  `ENODEB_ID` int UNSIGNED NULL DEFAULT NULL COMMENT '基站标识',
  `PCI` int UNSIGNED NULL DEFAULT NULL COMMENT '优化调整后的本小区
PCI 值',
  `PSS` int UNSIGNED NULL DEFAULT NULL COMMENT '小区 PSS',
  `SSS` int UNSIGNED NULL DEFAULT NULL COMMENT '小区 SSS',
  `LONGITUDE` float(10, 0) UNSIGNED NULL DEFAULT NULL COMMENT '小区经
度',
  `LATITUDE` float(10, 0) UNSIGNED NULL DEFAULT NULL COMMENT '小区纬
度',
  `STYLE` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NULL DEFAULT NULL COMMENT '基站类型',
  `OPT_DATETIME` datetime NULL DEFAULT NULL COMMENT '优化时间',
  PRIMARY KEY (`ASSIGN_ID`, `SECTOR_ID`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_general_ci COMMENT = '小区 PCI 优化调整结果表' ROW_FORMAT =
Dynamic;

SET FOREIGN_KEY_CHECKS = 1;
```

从 tbPCIAssignment.xlsx 中导入数据:



3.6 路测 ATU 数据表 tbATUData

SQL:语句:

```
DROP TABLE IF EXISTS `tbatudata`;
CREATE TABLE `tbatudata` (
  `seq` bigint UNSIGNED NOT NULL,
  `FileName` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NOT NULL,
  `Time` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NULL DEFAULT NULL,
  `Longitude` float(10, 6) UNSIGNED NULL DEFAULT NULL,
  `Latitude` float(10, 6) UNSIGNED NULL DEFAULT NULL,
  `CellID` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NULL DEFAULT NULL,
  `TAC` int UNSIGNED NULL DEFAULT NULL,
  `EARFCN` int UNSIGNED NULL DEFAULT NULL,
  `PCI` smallint UNSIGNED NULL DEFAULT NULL,
  `RSRP` float NULL DEFAULT NULL,
  `RS_SINR` float NULL DEFAULT NULL,
  `NCell_ID_1` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NULL DEFAULT NULL,
  `NCell_EARFCN_1` int NULL DEFAULT NULL,
  `NCell_PCI_1` smallint NULL DEFAULT NULL,
  `NCell_RSRP_1` float NULL DEFAULT NULL,
  `NCell_ID_2` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NULL DEFAULT NULL,
  `NCell_EARFCN_2` int NULL DEFAULT NULL,
  `NCell_PCI_2` smallint NULL DEFAULT NULL,
  `NCell_RSRP_2` float NULL DEFAULT NULL,
  `NCell_ID_3` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NULL DEFAULT NULL,
  `NCell_EARFCN_3` int NULL DEFAULT NULL,
  `NCell_PCI_3` smallint NULL DEFAULT NULL,
  `NCell_RSRP_3` float NULL DEFAULT NULL,
  `NCell_ID_4` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NULL DEFAULT NULL,
  `NCell_EARFCN_4` int NULL DEFAULT NULL,
  `NCell_PCI_4` smallint UNSIGNED NULL DEFAULT NULL,
  `NCell_RSRP_4` float NULL DEFAULT NULL,
  `NCell_ID_5` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NULL DEFAULT NULL,
  `NCell_EARFCN_5` int NULL DEFAULT NULL,
  `NCell_PCI_5` smallint UNSIGNED NULL DEFAULT NULL,
  `NCell_RSRP_5` float NULL DEFAULT NULL,
```

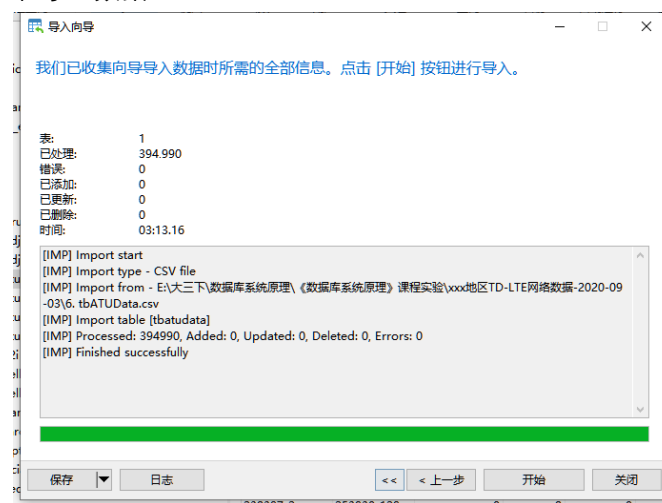
```

`NCell_ID_6` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NULL DEFAULT NULL,
`NCell_EARFCN_6` int NULL DEFAULT NULL,
`NCell_PCI_6` smallint UNSIGNED NULL DEFAULT NULL,
`NCell_RSRP_6` float NULL DEFAULT NULL,
PRIMARY KEY (`seq`, `FileName`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_general_ci COMMENT = '路测 ATU 数据表' ROW_FORMAT = Dynamic;

SET FOREIGN_KEY_CHECKS = 1;

```

从 tbATUData.csv 中导入数据：



3.7 路测 ATU C2I 干扰矩阵表 tbATUC2I

SQL:语句：

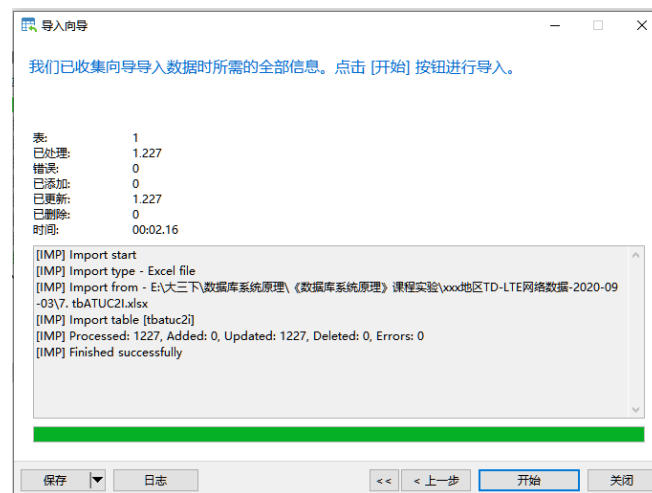
```

DROP TABLE IF EXISTS `tbatuc2i`;
CREATE TABLE `tbatuc2i` (
  `SECTOR_ID` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NOT NULL,
  `NCELL_ID` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NOT NULL,
  `RATIO_ALL` float UNSIGNED NULL DEFAULT NULL,
  `RANK` int UNSIGNED NULL DEFAULT NULL,
  `COSITE` tinyint UNSIGNED NULL DEFAULT NULL,
  PRIMARY KEY (`SECTOR_ID`, `NCELL_ID`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_general_ci COMMENT = '路测 ATU C2I 干扰矩阵表' ROW_FORMAT =
Dynamic;

SET FOREIGN_KEY_CHECKS = 1;

```

从 tbATUC2I.xlsx 中导入数据:

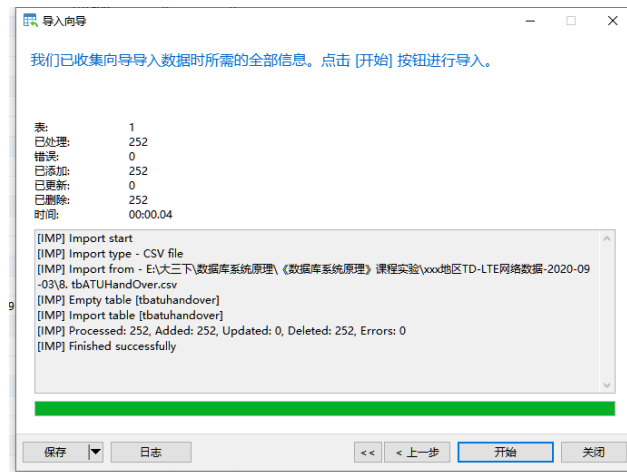


3.8 路测 ATU 切换统计矩阵 tbATUHandOver

SQL:语句:

```
DROP TABLE IF EXISTS `tbatuhandover`;  
CREATE TABLE `tbatuhandover` (  
  `SSECTOR_ID` varchar(50) CHARACTER SET utf8mb4 COLLATE  
utf8mb4_general_ci NULL DEFAULT NULL,  
  `NSECTOR_ID` varchar(50) CHARACTER SET utf8mb4 COLLATE  
utf8mb4_general_ci NULL DEFAULT NULL,  
  `HOATT` int UNSIGNED NULL DEFAULT NULL  
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE =  
utf8mb4_general_ci COMMENT = '路测 ATU 切换统计矩阵' ROW_FORMAT =  
Dynamic;  
  
SET FOREIGN_KEY_CHECKS = 1;
```

从 tbATUHandover.csv 中导入数据:



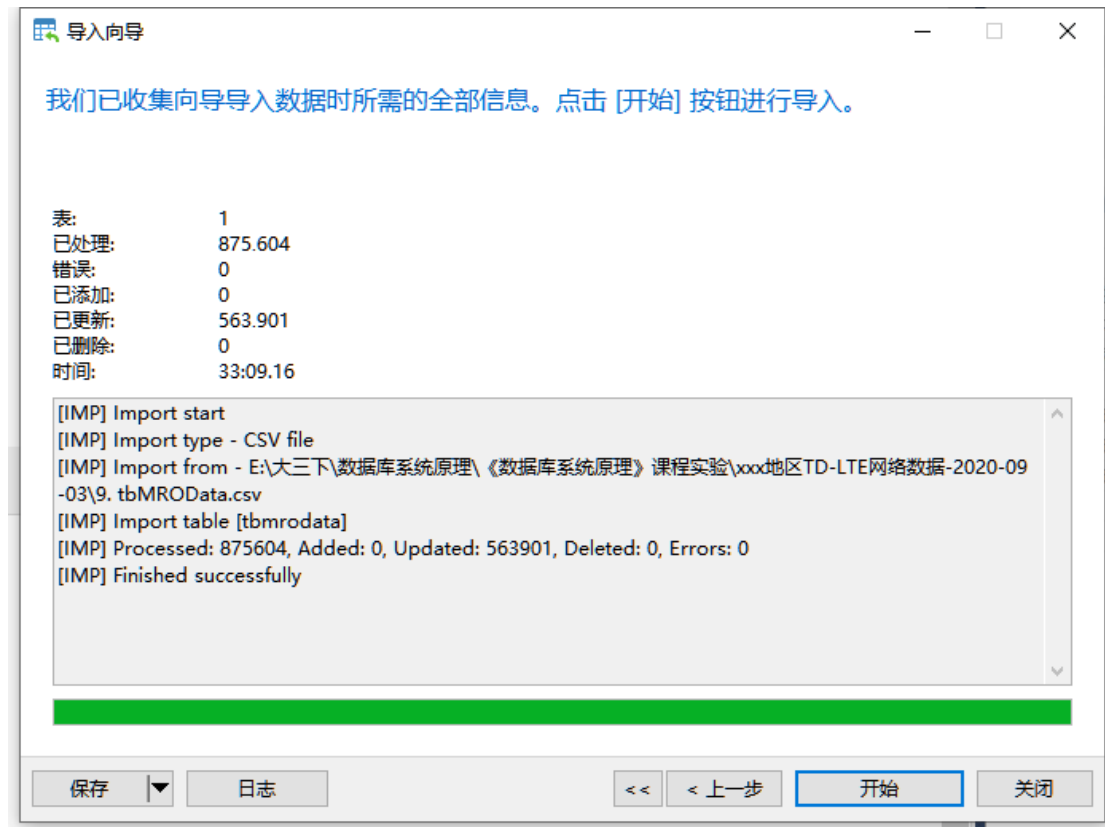
3.9 MRO 测量报告数据表 tbMROData

SQL语句:

```
DROP TABLE IF EXISTS `tbmrodata`;
CREATE TABLE `tbmrodata` (
  `TimeStamp` varchar(30) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NOT NULL COMMENT '测量时间点',
  `ServingSector` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NOT NULL COMMENT '服务小区/主小区 ID',
  `InterferingSector` varchar(50) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NOT NULL COMMENT '干扰小区 ID',
  `LteScRSRP` float UNSIGNED NULL DEFAULT NULL COMMENT '服务小区参考
信号接收功率 RSRP',
  `LteNcRSRP` float UNSIGNED NULL DEFAULT NULL COMMENT '干扰小区参考
信号接收功率 RSRP',
  `LteNcEarfcn` int UNSIGNED NULL DEFAULT NULL COMMENT '干扰小区频点
',
  `LteNcPci` smallint UNSIGNED NULL DEFAULT NULL COMMENT '干扰小区
PCI',
  PRIMARY KEY (`TimeStamp`, `ServingSector`, `InterferingSector`)
USING BTREE,
  INDEX `timeindex` (`TimeStamp`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_general_ci COMMENT = 'MRO 测量报告数据表' ROW_FORMAT =
Dynamic;

SET FOREIGN_KEY_CHECKS = 1;
```

从 tbMROData.csv 中导入数据:



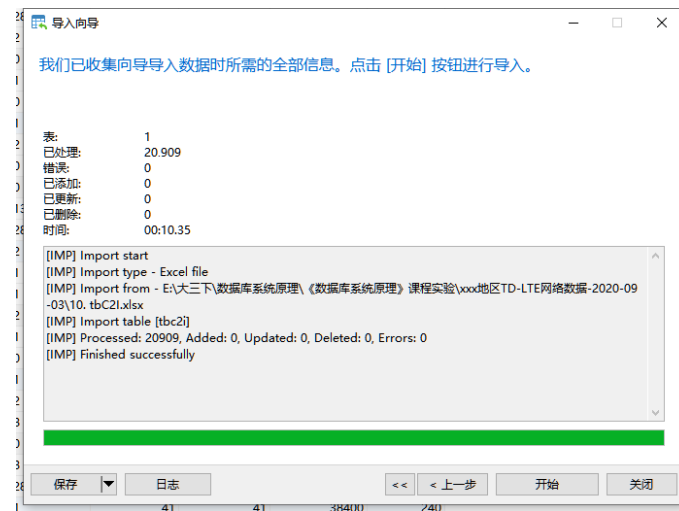
3.10 基于 MR 测量报告的干扰分析表 tbC2I

SQL语句:

```
DROP TABLE IF EXISTS `tbc2i`;
CREATE TABLE `tbc2i` (
  `CITY` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NULL DEFAULT NULL COMMENT '城市名称',
  `CELL` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NOT NULL COMMENT '主小区 ID',
  `NCELL` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NOT NULL COMMENT '邻小区 ID',
  `PrC2I9` float NULL DEFAULT NULL COMMENT '邻小区对主小区的 C2I 干扰
概率',
  `C2I_Mean` float NULL DEFAULT NULL COMMENT 'C2I 干扰的均值',
  `Std` float NULL DEFAULT NULL COMMENT 'C2I 干扰的标准差',
  `SampleCount` float NULL DEFAULT NULL COMMENT '邻区 NCELL 出现次数',
  `WeightedC2I` float NULL DEFAULT NULL COMMENT '加权 C2I 干扰',
  PRIMARY KEY (`CELL`, `NCELL`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_general_ci COMMENT = '基于 MR 测量报告的干扰分析表'
ROW_FORMAT = Dynamic;

SET FOREIGN_KEY_CHECKS = 1;
```

从 tbC2l.csv 中导入数据:

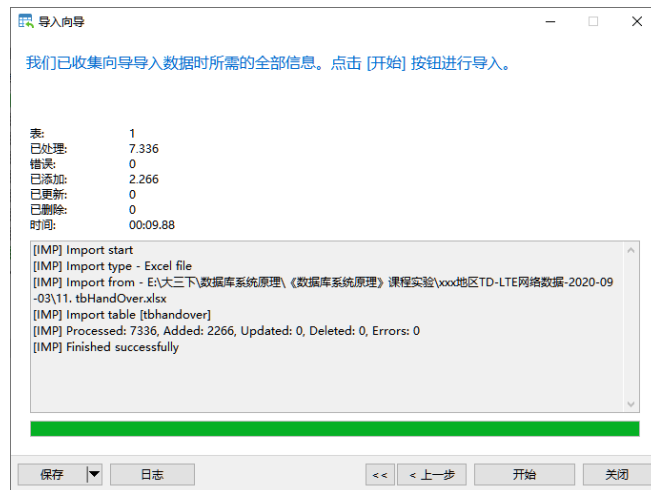


3.11 小区切换统计性能表 tbHandOver

SQL:语句:

```
DROP TABLE IF EXISTS `tbhandover`;  
CREATE TABLE `tbhandover` (  
  `CITY` varchar(255) CHARACTER SET utf8mb4 COLLATE  
utf8mb4_general_ci NULL DEFAULT NULL COMMENT '城市名称',  
  `SCell` varchar(50) CHARACTER SET utf8mb4 COLLATE  
utf8mb4_general_ci NOT NULL COMMENT '切换源小区 ID',  
  `NCell` varchar(50) CHARACTER SET utf8mb4 COLLATE  
utf8mb4_general_ci NOT NULL COMMENT '切换目标小区 ID',  
  `HOATT` int UNSIGNED NULL DEFAULT NULL COMMENT '切换尝试次数',  
  `HOSUCC` int UNSIGNED NULL DEFAULT NULL COMMENT '切换成功次数',  
  `HOSUCCRATE` float UNSIGNED NULL DEFAULT NULL COMMENT '切换成功率',  
  PRIMARY KEY (`SCell`, `NCell`) USING BTREE  
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE =  
utf8mb4_general_ci COMMENT = '小区切换统计性能表' ROW_FORMAT =  
Dynamic;  
  
SET FOREIGN_KEY_CHECKS = 1;
```

导入数据:



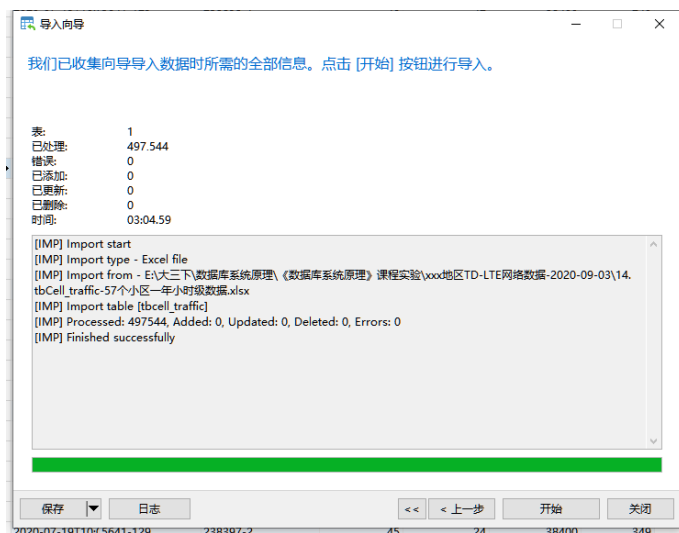
3.12. tbCell_traffic——57 个小区 2019-2020 年一年的小时级话务数据

SQL语句:

```
DROP TABLE IF EXISTS `tbcell_traffic`;
CREATE TABLE `tbcell_traffic` (
  `Date` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NOT NULL COMMENT '统计日期',
  `Hour` smallint UNSIGNED NOT NULL COMMENT '统计时间',
  `Sector_ID` varchar(255) CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci NOT NULL COMMENT '小区标识',
  `Traffic` float UNSIGNED NULL DEFAULT NULL COMMENT '小区话务量',
  PRIMARY KEY (`Date`, `Hour`, `Sector_ID`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_general_ci COMMENT = '57 个小区 2019-2020 年一年的小时级话务
数据' ROW_FORMAT = Dynamic;

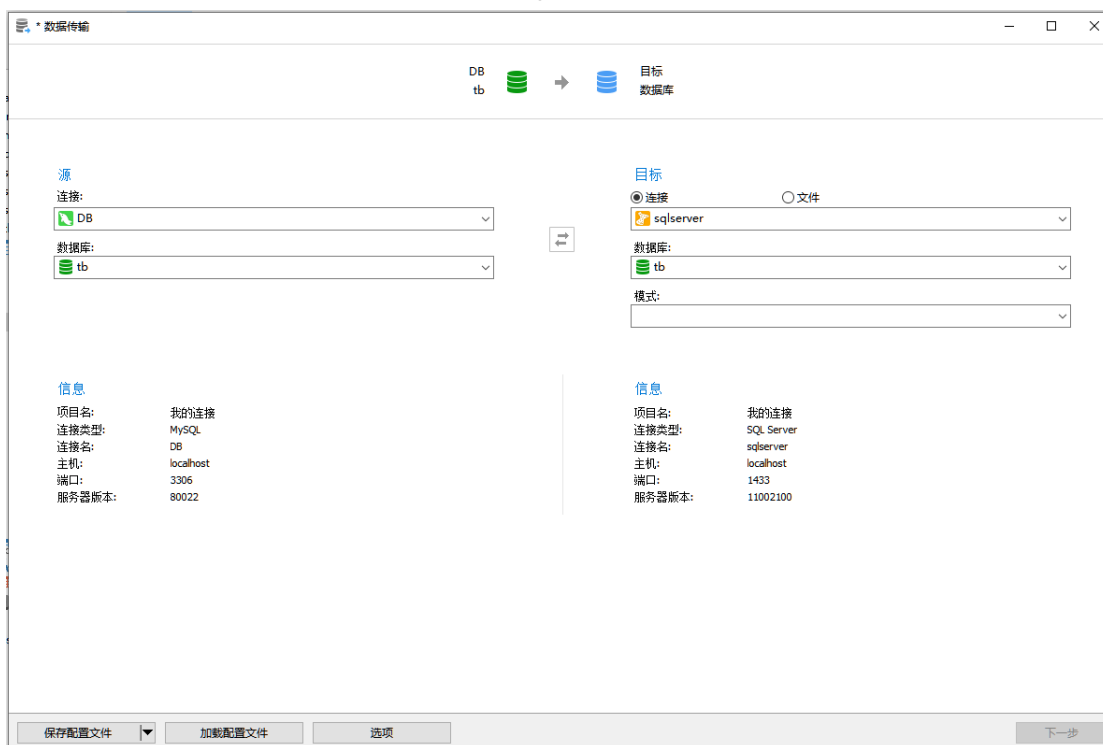
SET FOREIGN_KEY_CHECKS = 1;
```

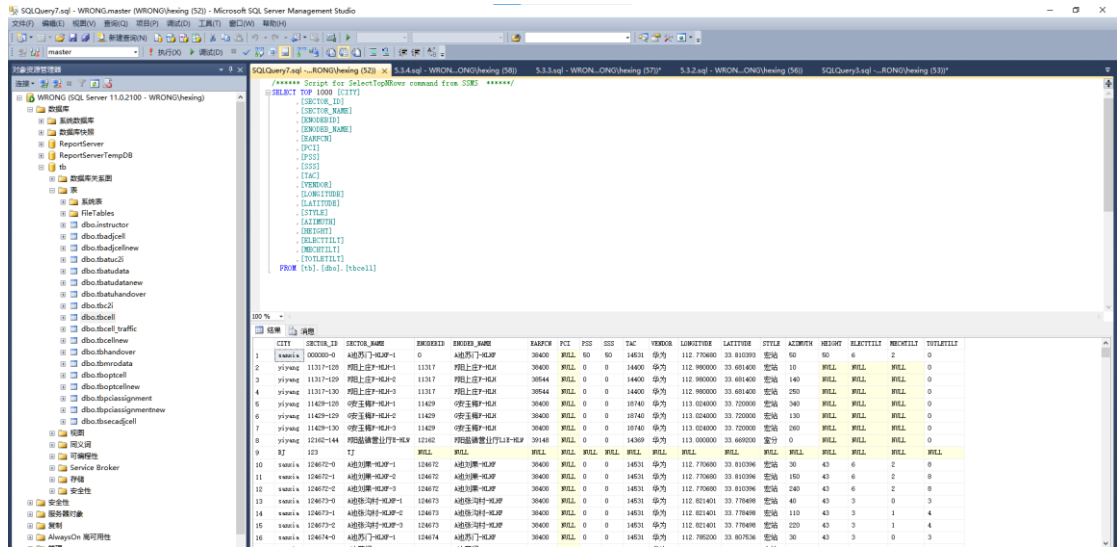
导入数据:



四、MySQL 导入 SQL Server

使用 Navicat 的数据传输功能，可在 SQL Server 上也建立表：





五、问题及解决

1. 在导入表格过程中，第一行就导入失败。反复检查后，发现是在建表过程中，属性值命名与 excel 数据中第一行的属性名不同导致。说明导入表格时，数据库会检查属性名。
2. 在导入表格 9 时，导入错误。发现是在建表中设置了主键，而数据中设置的主键的几个属性并不对全部属性有函数依赖，因此错误。
3. 在首次登录以及登录数据库的过程中出现了用户名，密码等登录问题，熟悉软件后，问题得到解决。

六、实验总结

在实验中，在登陆数据库后，感受了建表和数据导入的过程，以及一定限制条件的设置，使得我们体会到了数据库建表的过程，以及遇到相关问题的解决方法。当下的数据库的功能，界面如此便捷与强大也是超乎我们的想象，期待后续实验的进行。同时也要感谢老师以及助教学长学姐提供的实验指导书，过程十分具体详细，帮助我们避免了众多问题

数据查询与修改实验

一、实验目的

对实验三中建立的 GSM 数据库关系表和视图进行各种类型的查询操作和修改操作，加深对 SQL 语言中 DML 的了解，掌握相关查询语句和数据修改语句的使用方法。

二、实验环境

采用 Navicat+MySQL8.2 数据库管理系统作为实验平台

三、实验内容

- 1. 简单的查询操作，包括单表的查询、选择条件、结果排序等的练习；
- 2. 复杂的查询操作，包括等值连接、自然连接等；
- 3. 统计查询操作，包括带有分组、集函数的查询操作；
- 4. 嵌套查询操作，包括带有 in、exists、not exists、集合操作的嵌套查询；
- 5. 练习对关系表的其他操作如插入、删除、更新；
- 6. 练习视图查询、视图修改等视图操作。

四、实验步骤

5.1 Basic structure of SQL Queries

5.1.1 Query on a A Single Relation---The select, where Clause

根据路测 ATU 数据表，使用 distinct 语句列出服务小区频点为 38400 的所有去重后的服务小区 ID。

【SQL 语句】:

```
SELECT DISTINCT
    ( CellID )
FROM
    `tbatudata`
WHERE
    EARFCN = 38400;
```

【查询结果】:

| CellID |
|------------|
| 253903-0 |
| 253903-2 |
| 259778-2 |
| 259778-0 |
| 253901-2 |
| 253901-1 |
| 253904-2 |
| 253930-130 |
| 253904-1 |
| 259775-1 |
| 253936-2 |
| 253936-1 |
| 253890-2 |
| 253890-1 |
| 253935-0 |
| 253914-1 |
| 253899-0 |
| 253891-2 |
| 253891-1 |
| 253923-1 |
| 253900-0 |

5.1.2 Query on multiple relations——The from Clause

根据路测 ATU C2I 干扰矩阵表和路测 ATU 切换统计矩阵表，查询主小区 ID 为“238397-1”的 小区的同站干扰小区 ID 和切换目标小区 ID。

【SQL 语句】:

```
SELECT
    SECTOR_ID,
    NCELL_ID,
    NSECTOR_ID
FROM
    tbatuc2i,
    tbatuhandover
WHERE
    tbatuc2i.SECTOR_ID=tbatuhandover.SSECTOR_ID and SECTOR_ID='238397-1'
```

【查询结果】:

| SECTOR_ID | NCELL_ID | NSECTOR_ID |
|-----------|-----------|------------|
| 238397-1 | 15578-130 | 253890-1 |
| 238397-1 | 238397-0 | 253890-1 |
| 238397-1 | 238397-2 | 253890-1 |
| 238397-1 | 253904-1 | 253890-1 |
| 238397-1 | 253921-2 | 253890-1 |
| 238397-1 | 253927-2 | 253890-1 |
| 238397-1 | 253931-0 | 253890-1 |
| 238397-1 | 259772-2 | 253890-1 |
| 238397-1 | 15578-130 | 253914-1 |
| 238397-1 | 238397-0 | 253914-1 |
| 238397-1 | 238397-2 | 253914-1 |
| 238397-1 | 253904-1 | 253914-1 |
| 238397-1 | 253921-2 | 253914-1 |
| 238397-1 | 253927-2 | 253914-1 |
| 238397-1 | 253931-0 | 253914-1 |
| 238397-1 | 259772-2 | 253914-1 |
| 238397-1 | 15578-130 | 253931-0 |
| 238397-1 | 238397-0 | 253931-0 |
| 238397-1 | 238397-2 | 253931-0 |
| 238397-1 | 253904-1 | 253931-0 |
| 238397-1 | 253921-2 | 253931-0 |

5.1.3 natural join

使用 nature join 语句重写 1.2 中的查询。

【SQL 语句】:

```
SELECT
    SECTOR_ID,
    NCELL_ID,
    NSECTOR_ID
FROM
```

```
tbatuc2i NATURAL JOIN tbatuhandover
WHERE
    SECTOR_ID='238397-1'
```

【查询结果】:

| SECTOR_ID | NCELL_ID | NSECTOR_ID |
|-----------|-----------|------------|
| 238397-1 | 15578-130 | 253890-1 |
| 238397-1 | 238397-0 | 253890-1 |
| 238397-1 | 238397-2 | 253890-1 |
| 238397-1 | 253904-1 | 253890-1 |
| 238397-1 | 253921-2 | 253890-1 |
| 238397-1 | 253927-2 | 253890-1 |
| 238397-1 | 253931-0 | 253890-1 |
| 238397-1 | 259772-2 | 253890-1 |
| 238397-1 | 15578-130 | 253914-1 |
| 238397-1 | 238397-0 | 253914-1 |
| 238397-1 | 238397-2 | 253914-1 |
| 238397-1 | 253904-1 | 253914-1 |
| 238397-1 | 253921-2 | 253914-1 |
| 238397-1 | 253927-2 | 253914-1 |
| 238397-1 | 253931-0 | 253914-1 |
| 238397-1 | 259772-2 | 253914-1 |
| 238397-1 | 15578-130 | 253931-0 |
| 238397-1 | 238397-0 | 253931-0 |
| 238397-1 | 238397-2 | 253931-0 |
| 238397-1 | 253904-1 | 253931-0 |
| 238397-1 | 253921-2 | 253931-0 |

5.2 Additional Basic Operations

5.2.1 The Rename Operation

根据基于 MR 测量报告的干扰分析表，使用 as 语句查询所有比主小区 ID 为“124673-0”，邻小区 ID 为“259772-0”的小区 C2I 干扰均值高的主小区 ID、邻小区 ID。

【SQL 语句】:

```
SELECT
    a1.SCELL,
    a1.NCELL
FROM
    tbc2i AS a1,
    tbc2i AS a2
WHERE
    a1.C2I_Mean > a2.C2I_Mean
    AND a2.SCELL = '124673-0'
    AND a2.NCELL = '259772-0'
```

【查询结果】:

| SCELL | NCELL |
|----------|------------|
| 124673-0 | 124673-1 |
| 124673-0 | 124673-2 |
| 124673-0 | 124683-2 |
| 124673-0 | 124711-2 |
| 124673-0 | 124813-0 |
| 124673-0 | 15290-129 |
| 124673-0 | 15476-128 |
| 124673-0 | 15513-128 |
| 124673-0 | 253771-0 |
| 124673-0 | 253785-2 |
| 124673-0 | 253787-1 |
| 124673-0 | 253787-2 |
| 124673-0 | 253819-1 |
| 124673-0 | 253821-1 |
| 124673-0 | 253821-2 |
| 124673-0 | 253823-1 |
| 124673-0 | 253901-0 |
| 124673-0 | 253917-2 |
| 124673-0 | 253929-0 |
| 124673-0 | 253929-1 |
| 124673-0 | 253930-128 |

5.2.2 String Operations & 5.2.3 Attribute Specification

根据小区 PCI 优化调整结果表，使用 like 语句查询小区名中包含“B 马”的相关信息。

【SQL 语句】:

```
SELECT
    *
FROM
    tbpciassignment
WHERE
    SECTOR_NAME LIKE "%B 马%"
```

【查询结果】:

| ASSIGN_ID | EARFCN | SECTOR_ID | SECTOR_NAME | ENODEB_ID | PCI | PSS | SSS | LONGITUDE | LATITUDE | STYLE | OPT_DATETIME |
|-----------|--------|------------|---------------|-----------|-----|-----|-----|-----------|----------|-------|---------------------|
| 10 | 38400 | 124818-0 | B马礼品-HLHF-1 | 124818 | 174 | 0 | 58 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 124818-1 | B马礼品-HLHF-2 | 124818 | 176 | 2 | 58 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 124818-2 | B马礼品-HLHF-3 | 124818 | 175 | 1 | 58 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 124893-0 | B马石嘴-HLHF-1 | 124893 | 179 | 2 | 59 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 124893-1 | B马石嘴-HLHF-2 | 124893 | 177 | 0 | 59 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 124893-2 | B马石嘴-HLHF-3 | 124893 | 178 | 1 | 59 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 124898-0 | B马310南园家场-HLH | 124898 | 281 | 2 | 93 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 124898-1 | B马310南园家场-HLH | 124898 | 279 | 0 | 93 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 124923-0 | B马东曹元-HLHF-1 | 124923 | 276 | 0 | 92 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 124923-1 | B马东曹元-HLHF-2 | 124923 | 277 | 1 | 92 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 124924-0 | B马香山南寺-HLHF-1 | 124924 | 199 | 1 | 66 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 124924-1 | B马香山南寺-HLHF-2 | 124924 | 198 | 0 | 66 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 124924-2 | B马香山南寺-HLHF-3 | 124924 | 200 | 2 | 66 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 151113-128 | B马金枫叶小区-HLHF | 151113 | 203 | 2 | 67 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 151113-129 | B马金枫叶小区-HLHF | 151113 | 201 | 0 | 67 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 151113-130 | B马金枫叶小区-HLHF | 151113 | 202 | 1 | 67 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |
| 10 | 38400 | 151114-128 | B马兴隆小区-HLHF-1 | 151114 | 197 | 2 | 65 | 113 | 113 | 34 宏站 | 2020-01-18 11:13:41 |

5.2.4 Ordering the Display of Tuples

根据路测 ATU 切换统计矩阵表，查询各小区的最大切换次数及相应的切换目标小区 ID，并 按降序排列。

【SQL 语句】:

```
SELECT
    SSECTOR_ID,
    NSECTOR_ID,
    HOATT
FROM
    tbatuhandover
GROUP BY
    SSECTOR_ID,
    NSECTOR_ID
HAVING
    HOATT = MAX( HOATT )
ORDER BY
    MAX( HOATT ) DESC
```

【查询结果】:

| SSECTOR_ID | NSECTOR_ID | HOATT |
|------------|------------|-------|
| 253911-1 | 253910-2 | 10 |
| 259772-0 | 253935-2 | 9 |
| 253910-2 | 253911-1 | 8 |
| 253935-2 | 259772-0 | 8 |
| 253894-2 | 253939-0 | 6 |
| 253912-0 | 253935-2 | 6 |
| 253923-2 | 259772-1 | 6 |
| 253934-2 | 253932-1 | 6 |
| 253939-0 | 253939-2 | 6 |
| 253939-2 | 5641-130 | 6 |
| 253890-1 | 253914-1 | 5 |
| 253890-1 | 253935-0 | 5 |
| 253895-1 | 253916-1 | 5 |
| 253899-2 | 259772-1 | 5 |

5.2.5 Where Clause Predicates

根据小区/基站工参表和基于 MR 测量报告的干扰分析表，使用 between 语句查询经度位于 112 到 113 之间、纬度位于 33.7 到 33.9 之间的小区的 C2I 干扰的均值最大的邻小区 ID。

【SQL 语句】:

```
SELECT
    SCELL,
    NCELL,
    MAX( C2I_Mean )
FROM
    tbcell,
    tbc2i
WHERE
    LONGITUDE BETWEEN 112
    AND 113
    AND LATITUDE BETWEEN 33.7
    AND 33.9
    AND tbcell.SECTOR_ID = tbc2i.SCELL
GROUP BY
    SCELL
```

【查询结果】:

| SCELL | NCELL | MAX(C2I_Mean) |
|-----------|-----------|-----------------|
| 15483-128 | 124815-0 | 30.91 |
| 15483-129 | 124678-0 | 15.84 |
| 15483-130 | 124678-2 | 20.36 |
| 15484-128 | 124675-2 | 23.79 |
| 15484-129 | 124684-2 | 22.23 |
| 15484-130 | 124684-2 | 20.53 |
| 15485-128 | 11429-128 | 21.5 |
| 15485-129 | 11429-128 | 17.92 |
| 15485-130 | 11429-128 | 21.07 |
| 15486-128 | 124673-0 | 22.48 |
| 15486-129 | 124673-0 | 23.84 |
| 15486-130 | 11429-128 | 24.63 |
| 15487-128 | 11429-128 | 16.44 |
| 15487-129 | 11429-130 | 19.72 |
| 15487-130 | 11429-128 | 14.52 |
| 15496-128 | 124686-1 | 19 |
| 15496-129 | 124681-0 | 20.14 |
| 15496-130 | 124686-1 | 14.52 |
| 15513-128 | 124673-0 | 12.86 |
| 15513-129 | 124675-0 | 23.15 |
| 15513-130 | 124675-0 | 12.34 |

5.3 Set Operation

5.3.1 根据小区/基站工参表,使用 union 语句中查询所属城市为 yiyang、频点为 38544,

或所属城市为 sanxia、频点为 38400 的小区。

【SQL 语句】:

```
SELECT
    *
FROM
    tbcell
WHERE
    CITY = 'yiyang'
    AND EARFCN = 38544 UNION
SELECT
    *
FROM
    tbcell
WHERE
    CITY = 'sanxia'
    AND EARFCN = 38400
```

【查询结果】:

| CITY | SECTOR_ID | SECTOR_NAME | ENODEBID | ENODEB_NAME | EARFCN | PCI | PSS | SSS | TAC | VENDOR | LONGITUDE | LATITUDE | STYLE | AZIMUTH | HEIGHT | ELECTTILT | MECHTILT |
|--------|-----------|--------------|----------|-------------|--------|-----|-----|-----|-------|--------|------------|-----------|-------|---------|--------|-----------|----------|
| yiyang | 254572-3 | G安铁门村F-HLH-4 | 254572 | G安铁门村F-HLH | 38544 | 442 | 0 | 0 | 18740 | 华为 | 113.010000 | 33.728600 | 宏站 | | 50 | (Null) | (Null) |
| yiyang | 254572-4 | G安铁门村F-HLH-5 | 254572 | G安铁门村F-HLH | 38544 | 441 | 0 | 0 | 18740 | 华为 | 113.010000 | 33.728600 | 宏站 | | 130 | (Null) | (Null) |
| yiyang | 254572-5 | G安铁门村F-HLH-6 | 254572 | G安铁门村F-HLH | 38544 | 443 | 0 | 0 | 18740 | 华为 | 113.010000 | 33.728600 | 宏站 | | 220 | (Null) | (Null) |
| sanxia | 124672-0 | A池刘果-HLHF-1 | 124672 | A池刘果-HLHF | 38400 | 32 | 0 | 0 | 14531 | 华为 | 112.770680 | 33.810396 | 宏站 | | 30 | 43 | 6 |
| sanxia | 124672-1 | A池刘果-HLHF-2 | 124672 | A池刘果-HLHF | 38400 | 30 | 0 | 0 | 14531 | 华为 | 112.770680 | 33.810396 | 宏站 | | 150 | 43 | 6 |
| sanxia | 124672-2 | A池刘果-HLHF-3 | 124672 | A池刘果-HLHF | 38400 | 31 | 0 | 0 | 14531 | 华为 | 112.770680 | 33.810396 | 宏站 | | 240 | 43 | 6 |
| sanxia | 124673-0 | A池张沟村-HLHF-1 | 124673 | A池张沟村-HLHF | 38400 | 200 | 0 | 0 | 14531 | 华为 | 112.821401 | 33.778498 | 宏站 | | 40 | 43 | 3 |
| sanxia | 124673-1 | A池张沟村-HLHF-2 | 124673 | A池张沟村-HLHF | 38400 | 198 | 0 | 0 | 14531 | 华为 | 112.821401 | 33.778498 | 宏站 | | 110 | 43 | 3 |
| sanxia | 124673-2 | A池张沟村-HLHF-3 | 124673 | A池张沟村-HLHF | 38400 | 199 | 0 | 0 | 14531 | 华为 | 112.821401 | 33.778498 | 宏站 | | 220 | 43 | 3 |
| sanxia | 124674-0 | A池苏门-HLHF-1 | 124674 | A池苏门-HLHF | 38400 | 327 | 0 | 0 | 14531 | 华为 | 112.785200 | 33.807536 | 宏站 | | 30 | 43 | 3 |
| sanxia | 124674-1 | A池苏门-HLHF-2 | 124674 | A池苏门-HLHF | 38400 | 329 | 0 | 0 | 14531 | 华为 | 112.785200 | 33.807536 | 宏站 | | 180 | 30 | 6 |
| sanxia | 124674-2 | A池苏门-HLHF-3 | 124674 | A池苏门-HLHF | 38400 | 328 | 0 | 0 | 14531 | 华为 | 112.785200 | 33.807536 | 宏站 | | 290 | 30 | 6 |
| sanxia | 124675-0 | A池南湾-HLHF-1 | 124675 | A池南湾-HLHF | 38400 | 94 | 0 | 0 | 14563 | 华为 | 112.822746 | 33.681016 | 宏站 | | 60 | 43 | 3 |
| sanxia | 124675-1 | A池南湾-HLHF-2 | 124675 | A池南湾-HLHF | 38400 | 93 | 0 | 0 | 14563 | 华为 | 112.822746 | 33.681016 | 宏站 | | 150 | 30 | 6 |
| sanxia | 124675-2 | A池南湾-HLHF-3 | 124675 | A池南湾-HLHF | 38400 | 95 | 0 | 0 | 14563 | 华为 | 112.822746 | 33.681016 | 宏站 | | 310 | 43 | 3 |
| sanxia | 124676-0 | A池峨嵋-HLHF-1 | 124676 | A池峨嵋-HLHF | 38400 | 74 | 0 | 0 | 18833 | 华为 | 112.742883 | 33.694259 | 宏站 | | 0 | 43 | 3 |
| sanxia | 124676-1 | A池峨嵋-HLHF-2 | 124676 | A池峨嵋-HLHF | 38400 | 72 | 0 | 0 | 18833 | 华为 | 112.742883 | 33.694259 | 宏站 | | 120 | 43 | 3 |
| sanxia | 124676-2 | A池峨嵋-HLHF-3 | 124676 | A池峨嵋-HLHF | 38400 | 73 | 0 | 0 | 18833 | 华为 | 112.742883 | 33.694259 | 宏站 | | 240 | 43 | 3 |
| sanxia | 124677-0 | A池高岭-HLHF-1 | 124677 | A池高岭-HLHF | 38400 | 400 | 0 | 0 | 14563 | 华为 | 112.794595 | 33.732245 | 宏站 | | 60 | 30 | 6 |
| sanxia | 124677-1 | A池高岭-HLHF-2 | 124677 | A池高岭-HLHF | 38400 | 401 | 0 | 0 | 14563 | 华为 | 112.794595 | 33.732245 | 宏站 | | 150 | 48 | 6 |

5.3.2 根据小区一阶邻区关系表和二阶（同频）邻区关系表，使用 intersect 语句查询一阶邻区和二阶邻区相同的小区。

由于 MySQL 不支持 intersect 语句，因此使用别的方法实现：

【SQL 语句】:

```
SELECT
    tbadjcell.S_SECTOR_ID,
    tbadjcell.N_SECTOR_ID
FROM
    tbadjcell
    INNER JOIN tbsecadjcell USING ( S_SECTOR_ID, N_SECTOR_ID )
```

【查询结果】:

| 信息 | 结果 1 | 剖析 | 状态 |
|----|-------------|-------------|----|
| | S_SECTOR_ID | N_SECTOR_ID | |
| ▶ | 124673-0 | 124673-1 | |
| | 124673-0 | 124673-2 | |
| | 124673-0 | 124683-1 | |
| | 124673-0 | 124711-0 | |
| | 124673-0 | 124711-1 | |
| | 124673-0 | 124711-2 | |
| | 124673-0 | 124812-0 | |
| | 124673-0 | 124812-2 | |
| | 124673-0 | 15290-128 | |
| | 124673-0 | 15290-129 | |
| | 124673-0 | 253901-0 | |
| | 124673-0 | 253930-130 | |
| | 124673-0 | 259595-0 | |
| | 124673-0 | 259595-1 | |
| | 124673-0 | 259654-1 | |
| | 124673-0 | 259655-1 | |
| | 124673-0 | 47443-0 | |
| | 124673-0 | 47444-1 | |
| | 124673-0 | 47588-16 | |
| | 124673-0 | 5691-128 | |
| | 124673-0 | 7201-128 | |

使用 SQL Server 实现:

【SQL 语句】:

```
SELECT S_SECTOR_ID,N_SECTOR_ID
FROM [tb].[dbo].[tbadjcell]
INTERSECT
SELECT S_SECTOR_ID,N_SECTOR_ID
FROM [tb].[dbo].[tbsecadjcell]
```

【查询结果】:

| | | |
|-------|-------------|-------------|
| 146 % | 结果 | 消息 |
| | S_SECTOR_ID | N_SECTOR_ID |
| 1 | 124673-0 | 124673-1 |
| 2 | 124673-0 | 124673-2 |
| 3 | 124673-0 | 124683-1 |
| 4 | 124673-0 | 124711-0 |
| 5 | 124673-0 | 124711-1 |
| 6 | 124673-0 | 124711-2 |
| 7 | 124673-0 | 124812-0 |
| 8 | 124673-0 | 124812-2 |
| 9 | 124673-0 | 15290-128 |
| 10 | 124673-0 | 15290-129 |
| 11 | 124673-0 | 253901-0 |
| 12 | 124673-0 | 253930-130 |
| 13 | 124673-0 | 259595-0 |
| 14 | 124673-0 | 259595-1 |
| 15 | 124673-0 | 259654-1 |
| 16 | 124673-0 | 259655-1 |

5.3.3 根据一阶邻区关系表和二阶（同频）邻区关系表，使用 except 语句查询二阶邻区不是一阶邻区的小区。

由于 MySQL 不支持 except 语句，因此使用别的方法实现：

【SQL 语句】：

```
SELECT
    S_SECTOR_ID,
    N_SECTOR_ID
FROM
    tbsecadjcell
WHERE
    ( S_SECTOR_ID, N_SECTOR_ID ) NOT IN (
        SELECT
            S_SECTOR_ID,
            N_SECTOR_ID
        FROM
            tbadjcell)
```

【查询结果】：

| S_SECTOR_ID | N_SECTOR_ID |
|-------------|-------------|
| 124673-0 | 124683-0 |
| 124673-0 | 124683-2 |
| 124673-0 | 124685-0 |
| 124673-0 | 124685-1 |
| 124673-0 | 124685-2 |
| 124673-0 | 124687-0 |
| 124673-0 | 124687-1 |
| 124673-0 | 124687-2 |
| 124673-0 | 124812-1 |
| 124673-0 | 15113-128 |
| 124673-0 | 15113-130 |
| 124673-0 | 15578-128 |
| 124673-0 | 15578-129 |
| 124673-0 | 238397-0 |
| 124673-0 | 238397-1 |
| 124673-0 | 238397-2 |
| 124673-0 | 238455-16 |
| 124673-0 | 253771-0 |
| 124673-0 | 253786-0 |
| 124673-0 | 253893-0 |
| 124673-0 | 253893-1 |

使用 SQL Server 实现

【SQL 语句】：

```
SELECT S_SECTOR_ID,N_SECTOR_ID
FROM [tb].[dbo].[tbsecadjcell]
EXCEPT
SELECT S_SECTOR_ID,N_SECTOR_ID
FROM [tb].[dbo].[tbadjcell]
```

【查询结果】：

161 %

结果 消息

| | S_SECTOR_ID | N_SECTOR_ID |
|----|-------------|-------------|
| 1 | 124673-0 | 124683-0 |
| 2 | 124673-0 | 124683-2 |
| 3 | 124673-0 | 124685-0 |
| 4 | 124673-0 | 124685-1 |
| 5 | 124673-0 | 124685-2 |
| 6 | 124673-0 | 124687-0 |
| 7 | 124673-0 | 124687-1 |
| 8 | 124673-0 | 124687-2 |
| 9 | 124673-0 | 124812-1 |
| 10 | 124673-0 | 15113-128 |
| 11 | 124673-0 | 15113-130 |
| 12 | 124673-0 | 15578-128 |
| 13 | 124673-0 | 15578-129 |
| 14 | 124673-0 | 238397-0 |
| 15 | 124673-0 | 238397-1 |
| 16 | 124673-0 | 238397-2 |

5.3.4 根据路测 ATU C2I 干扰矩阵表，使用 except 语句查询主小区和邻小区间干扰强度最大的小区。

使用 SQL Server 实现：

【SQL 语句】：

```
select SECTOR_ID,RATIO_ALL from [tb].[dbo].[tbatuc2i]
except
select T.SECTOR_ID,T.RATIO_ALL
from [tb].[dbo].[tbatuc2i] as T,[tb].[dbo].[tbatuc2i] as S
where T.RATIO_ALL < S.RATIO_ALL
```

【查询结果】：

161 %

结果 消息

| | SECTOR_ID | RATIO_ALL |
|---|-----------|-----------|
| 1 | 7400-130 | 300 |

5.4 Null Values

根据路测 ATU 数据表，查询第 1 邻小区/干扰小区物理小区标识不为空的服务小区 ID、服务小区 PCI。

【SQL 语句】:

```
SELECT
    CellID,
    PCI
FROM
    tbatudata
WHERE
    ISNULL( NCell_ID_1 )= 0
    AND LENGTH(
        trim( NCell_ID_1 ))> 0;
```

【查询结果】:

| CellID | PCI |
|----------|-----|
| 253903-0 | 166 |
| 253903-0 | 166 |
| 253903-0 | 166 |
| 253903-0 | 166 |
| 253903-0 | 166 |
| 253903-0 | 166 |
| 253903-0 | 166 |
| 253903-0 | 166 |
| 253903-0 | 166 |
| 253903-0 | 166 |
| 253903-0 | 166 |
| 253903-0 | 166 |
| 253903-2 | 167 |
| 253903-2 | 167 |
| 253903-2 | 167 |
| 253903-2 | 167 |
| 253903-2 | 167 |
| 253903-2 | 167 |
| 253903-2 | 167 |
| 253903-2 | 167 |
| 253903-2 | 167 |
| 253903-2 | 167 |
| 253903-2 | 167 |
| 253903-2 | 167 |

5.5 Aggregate Functions

5.5.1 根据优化小区/保护带小区表和小区一阶邻区关系表，查询一阶邻区数大于 10 的优化小区，并将查询结果降序排列。

【SQL 语句】:

```
SELECT
    tboptcell.SECTOR_ID,
    COUNT( tbadjcell.N_SECTOR_ID ) AS count_n
FROM
    tboptcell,
    tbadjcell
WHERE
    tboptcell.SECTOR_ID = tbadjcell.S_SECTOR_ID
    AND tboptcell.CELL_TYPE = '优化区'
GROUP BY
    tboptcell.SECTOR_ID
HAVING
    COUNT( tbadjcell.N_SECTOR_ID )>= 10
ORDER BY
    COUNT( tbadjcell.N_SECTOR_ID ) DESC
```

【查询结果】:

| SECTOR_ID | count_n |
|-----------|---------|
| 238397-1 | 57 |
| 253935-0 | 54 |
| 253936-2 | 54 |
| 238397-0 | 53 |
| 253892-2 | 53 |
| 253929-0 | 53 |
| 253929-2 | 53 |
| 253935-2 | 53 |
| 253892-0 | 52 |
| 253920-0 | 52 |
| 253923-1 | 52 |
| 253931-0 | 52 |
| 253940-0 | 52 |
| 238397-2 | 51 |
| 253918-0 | 51 |

5.5.2 根据小区/基站工参表和路测 ATU 数据表, 查询所属基站为“253903”的小区的最大信噪比 SINR, 最小信噪比 SINR, 平均信噪比 SINR。

【SQL 语句】:

```
SELECT
    MAX( RS_SINR ),
    MIN( RS_SINR ),
    AVG( RS_SINR )
FROM
    tbccll,
    tbatudata
WHERE
    tbccll.ENODEBID = '253903'
    AND tbccll.SECTOR_ID = tbatudata.CellID
```

【查询结果】:

| MAX(RS_SINR) | MIN(RS_SINR) | AVG(RS_SINR) |
|----------------|----------------|-------------------|
| 30 | -17.3 | 4.341939504076183 |

5.6 Nested (嵌套) Subqueries

5.6.1 Set Membership

根据优化小区/保护带小区表和小区 PCI 优化调整结果表, 查询小区类型为“优化区”的小区经调整后的 PCI。

【SQL 语句】:

```
SELECT
    SECTOR_ID,
    SECTOR_NAME,
```

```

        PCI
FROM
        tbpciassignment
WHERE
        tbpciassignment.SECTOR_ID IN (
        SELECT
                SECTOR_ID
        FROM
                tboptcell
        WHERE
                CELL_TYPE = '优化区')

```

【查询结果】:

| SECTOR_ID | SECTOR_NAME | PCI |
|-----------|-------------|-----|
| 124711-0 | 三峽? | 185 |
| 124711-1 | 三峽? | 183 |
| 124711-2 | 三峽? | 184 |
| 124712-0 | 三峽? | 188 |
| 124712-1 | 三峽? | 186 |
| 124712-2 | 三峽? | 187 |
| 124713-0 | 三峽? | 181 |
| 124713-1 | 三峽? | 182 |
| 124713-2 | 三峽? | 180 |
| 124818-0 | B马礼殿-HLHF-1 | 174 |
| 124818-1 | B马礼殿-HLHF-2 | 176 |
| 124818-2 | B马礼殿-HLHF-3 | 175 |
| 124893-0 | B马石佛-HLHF-1 | 179 |
| 124893-1 | B马石佛-HLHF-2 | 177 |
| 124893-2 | B马石佛-HLHF-3 | 178 |

5.6.2 Set Comparison – “some” Clause

1. 根据路测 ATU 数据表和小区/基站工参表, 使用 some 语句查询“服务小区参考信号接收功率 RSRP”大于部分 (至少一个) 所属基站 ID 为 5660 的小区的“服务小区参考信号接收功率 RSRP”的服务小区。

【SQL 语句】:

```

SELECT
        CellID
FROM
        tbatudata
WHERE
        RSRP > SOME ( SELECT RSRP FROM tbatudata, tbccll
                WHERE tbatudata.PCI = tbccll.PCI AND ENODEBID = 5660 )

```

【查询结果】:

| CellID |
|--------|
| (Null) |

结果为空

2. 根据路测 ATU 切换统计矩阵表和 MRO 测量报告数据表, 使用 all 语句查询切换次数最多的小区的干扰小区 ID, 干扰小区 PCI。

【SQL 语句】:

```

SELECT
        InterferingSector,

```

```

        LteNcPci
FROM
        tbmrodata mro,
        tbatuhandover atu
WHERE
        mro.ServingSector = atu.SSECTOR_ID
        AND atu.HOATT >= ALL ( SELECT HOATT
                                FROM tbmrodata mro, tbatuhandover atu
                                WHERE mro.ServingSector = atu.SSECTOR_ID )

```

【查询结果】:

| InterferingSector | LteNcPci |
|-------------------|----------|
| 253899-0 | 444 |
| 253899-1 | 445 |
| 253899-2 | 446 |
| 253907-0 | 471 |
| 253909-2 | 87 |
| 253935-1 | 247 |
| 253935-2 | 248 |
| 253899-0 | 444 |
| 253899-1 | 445 |
| 253899-2 | 446 |
| 253907-0 | 471 |
| 253909-2 | 87 |
| 253923-0 | 63 |
| 253899-0 | 444 |
| 253899-1 | 445 |
| 253899-2 | 446 |
| 253907-0 | 471 |

5.6.3 Test for Empty Relations---Use of “not exists” Clause

根据路测 ATU 数据表和优化小区表，使用 not exists 语句查询小区类型不为保护带小区的第 1 邻小区/干扰小区的标识、第 1 邻小区/干扰小区频点、第 1 邻小区/干扰小区物理小区标识、第 1 邻小区/干扰小区参考信号接收强度。

【SQL 语句】:

```

SELECT
        NCell_ID_1,
        NCell_PCI_1,
        NCell_RSRP_1,
        NCell_EARFCN_1
FROM
        tbatudata
WHERE
        NOT EXISTS ( SELECT * FROM tboptcell WHERE tbatudata.CellID =
                    tboptcell.SECTOR_ID AND CELL_TYPE = '保护带' )

```

【查询结果】:

| NCell_ID_1 | NCell_PCI_1 | NCell_RSRP_1 | NCell_EARFCN_1 |
|------------|-------------|--------------|----------------|
| (Null) | (Null) | (Null) | (Null) |
| 253894-0 | 168 | -73.25 | 38400 |
| 253894-0 | 168 | -71.25 | 38400 |
| 253894-0 | 168 | -71.25 | 38400 |
| 253894-0 | 168 | -65.69 | 38400 |
| 253894-0 | 168 | -68.81 | 38400 |
| 253894-0 | 168 | -68.81 | 38400 |
| 253894-0 | 168 | -70.94 | 38400 |
| 253894-0 | 168 | -69.75 | 38400 |
| 253894-0 | 168 | -69.75 | 38400 |
| (Null) | (Null) | (Null) | (Null) |
| 253894-0 | 168 | -68.5 | 38400 |
| (Null) | (Null) | (Null) | (Null) |
| (Null) | (Null) | (Null) | (Null) |
| 253894-0 | 168 | -68.75 | 38400 |
| 253894-0 | 168 | -68.13 | 38400 |

5.6.4 Test for Absence of Duplicate Tuples

根据基于 MR 测量报告的干扰分析表和路测 ATU 切换统计矩阵表，查询主小区 ID 在路测 ATU 切换统计矩阵表中只出现过一次的加权 C2I 干扰。

【SQL 语句】：

```
SELECT
    WeightedC2I
FROM
    tbc2i
WHERE
    SCELL IN ( SELECT SSECTOR_ID FROM tbatuhandover GROUP BY
SSECTOR_ID HAVING COUNT( * ) = 1 )
```

【查询结果】：

| WeightedC2I |
|-------------|
| 105148 |
| 608344 |
| 1138400 |
| 75240 |
| 3078 |
| 492568 |
| 1580350 |
| 32032 |
| 5390 |
| 16826 |
| 0 |
| 15288 |
| 51832 |
| 3440 |
| 1674570 |

5.6.5 Subqueries in the From Clause

根据路测 ATU 数据表，查询服务小区参考信号接收功率 RSRP 的均值大于 -70 的小区。

【SQL 语句】：

```
SELECT DISTINCT
    CellID
FROM
    tbatudata
WHERE
```

```
CellID IN ( SELECT CellID FROM tbatudata GROUP BY CellID
HAVING AVG( RSRP ) > - 70 )
```

【查询结果】:

| CellID |
|----------|
| 259775-1 |
| 253890-2 |
| 253891-2 |
| 253941-0 |
| 253894-0 |
| 253931-2 |
| 253890-0 |
| 259775-2 |
| 253914-2 |

5.6.6 With Clause

根据路测 ATU 切换统计矩阵表和 MRO 测量报告数据表，使用 with 语句找出所有具有最低切换次数的小区 MRO 测量信息。

【SQL 语句】:

```
WITH Min AS
( SELECT SSECTOR_ID FROM tbatuhandover
WHERE HOATT <= ANY ( SELECT HOATT FROM tbhandover ) )
SELECT DISTINCT
*
FROM
    tbmrodata,
    Min
WHERE
    ServingSector = SSECTOR_ID
```

【查询结果】:

| TimeStamp | ServingSector | InterferingSector | LteScRSRP | LteNcRSRP | LteNcEarfcn | LteNcPci | SSECTOR_ID |
|-------------------------|---------------|-------------------|-----------|-----------|-------------|----------|------------|
| 2020-07-19T10:00:03.840 | 5641-129 | 15513-128 | 26 | 20 | 38400 | 499 | 5641-129 |
| 2020-07-19T10:00:03.840 | 5641-129 | 238397-2 | 33 | 23 | 38400 | 349 | 5641-129 |
| 2020-07-19T10:00:03.840 | 5641-129 | 253893-0 | 31 | 30 | 38400 | 241 | 5641-129 |
| 2020-07-19T10:00:03.840 | 5641-129 | 253893-1 | 46 | 42 | 38400 | 240 | 5641-129 |
| 2020-07-19T10:00:03.840 | 5641-129 | 253894-0 | 26 | 22 | 38400 | 168 | 5641-129 |
| 2020-07-19T10:00:03.840 | 5641-129 | 253905-1 | 46 | 46 | 38400 | 325 | 5641-129 |
| 2020-07-19T10:00:03.840 | 5641-129 | 253931-2 | 33 | 22 | 38400 | 160 | 5641-129 |
| 2020-07-19T10:00:03.840 | 5641-129 | 253939-0 | 46 | 47 | 38400 | 144 | 5641-129 |
| 2020-07-19T10:00:03.840 | 5641-129 | 259658-0 | 33 | 13 | 38400 | 233 | 5641-129 |
| 2020-07-19T10:00:03.840 | 5641-129 | 274161-130 | 26 | 16 | 38400 | 38 | 5641-129 |
| 2020-07-19T10:00:03.840 | 5641-130 | 15513-128 | 44 | 42 | 38400 | 499 | 5641-130 |
| 2020-07-19T10:00:03.840 | 5641-130 | 238397-2 | 43 | 39 | 38400 | 349 | 5641-130 |
| 2020-07-19T10:00:03.840 | 5641-130 | 253893-1 | 43 | 41 | 38400 | 240 | 5641-130 |
| 2020-07-19T10:00:03.840 | 5641-130 | 253903-1 | 58 | 40 | 38400 | 165 | 5641-130 |
| 2020-07-19T10:00:03.840 | 5641-130 | 253903-2 | 57 | 52 | 38400 | 167 | 5641-130 |
| 2020-07-19T10:00:03.840 | 5641-130 | 253905-1 | 43 | 42 | 38400 | 325 | 5641-130 |

5.6.7 Scalar Subquery——Subqueries in the Select Clause

根据小区/基站工参表和一阶邻区关系表，列出频点为 38400 的所有小区的一阶邻区数目。

【SQL 语句】:

```
SELECT COUNT(N_SECTOR_ID)
FROM tbadjcell
WHERE S_EARFCN = 38400
GROUP BY S_SECTOR_ID
```

【查询结果】:

| COUNT(N_SECTOR_ID) | |
|--------------------|----|
| ▶ | 24 |
| | 27 |
| | 30 |
| | 23 |
| | 26 |
| | 26 |
| | 14 |
| | 18 |
| | 28 |
| | 18 |
| | 23 |
| | 21 |
| | 27 |
| | 30 |
| | 19 |

5.7 综合---查询语句

根据小区/基站工参表和小区切换统计性能表，查询具有最多二阶邻区数的小区的最大切换成功次数、相应的切换目标小区 ID、尝试切换次数。

【SQL 语句】:

```
WITH Num ( id, num ) AS
( SELECT S_SECTOR_ID, COUNT( N_SECTOR_ID ) FROM tbsecadjcell GROUP BY
S_SECTOR_ID ORDER BY COUNT( N_SECTOR_ID ) DESC ),
MaxNum ( id, maxNum ) AS
( SELECT id, MAX( num ) FROM Num )
SELECT
SCELL,
NCELL,
MAX( HOSUCC ),
HOATT
FROM
        tbhandover,
        MaxNum
WHERE
        SCELL = id
GROUP BY
        SCELL
```

【查询结果】:

| SCELL | NCELL | MAX(HOSUCC) | HOATT |
|----------|----------|---------------|-------|
| 124712-0 | 124712-1 | 3282 | 593 |

5.8 Modification of the Database

5.8.1 Deletion

根据路测 ATU 切换统计矩阵表和小区切换统计性能表，删除切换次数均值小于 3 的小区切换性能统计数据。

【SQL 语句】:

```
DELETE
FROM
    tbhandover
WHERE
    tbhandover.SCELL IN (
        SELECT
            SSECTOR_ID
        FROM
            tbatuhandover
        GROUP BY
            SSECTOR_ID
        HAVING
            AVG( HOATT ) < 3
    )
```

【查询结果】:

```
DELETE
FROM
    tbhandover
WHERE
    tbhandover.SCELL IN (
        SELECT
            SSECTOR_ID
        FROM
            tbatuhandover
        GROUP BY
            SSECTOR_ID
        HAVING
            AVG( HOATT ) < 3
    )
> Affected rows: 0
> 时间: 0.004s
|
```

5.8.2 Insertion

向小区/基站工参表中插入一条新信息

【SQL 语句】:

```
INSERT INTO tbcell
(CITY, SECTOR_ID, SECTOR_NAME, ENODEBID, ENODEB_NAME, EARFCN, PCI, PSS, SSS, T
AC, VENDOR, LONGITUDE, LATITUDE, STYLE, AZIMUTH, HEIGHT, ELECTTILT, MECHTILT,
TOTLETILT ) VALUES (
    'sanxia',
    '000000-0',
    'A 池苏门-HLHF-1',
```

```
'000000',
'A 池苏门-HLHF',
'38400',
'50',
'50',
'50',
'14531',
'华为',
'112.77068',
'33.810393',
'宏站',
'50',
'50',
'6',
'2',
'0')
```

【查询结果】:

```
INSERT INTO tbcell (CITY,SECTOR_ID,SECTOR_NAME,ENODEBID,ENODEB_NAME,EARFCN,PCI,PSS,SSS,TAC,VENDOR,LONGITUDE,LATITUDE,STYLE,AZIMUTH,HEIGHT,ELECTTILT,MECHTILT,TOTLETLT) VALUES (
'sanxia',
'000000-0',
'A池苏门-HLHF-1',
'000000',
'A池苏门-HLHF',
'38400',
'50',
'50',
'50',
'14531',
'华为',
'112.77068',
'33.810393',
'宏站',
'50',
'50',
'6',
'2',
'0')
> Affected rows: 1
> 时间: 0.008s
```

| CITY | SECTOR_ID | SECTOR_NAME | ENODEBID | ENODEB_NAME | EARFCN | PCI | PSS | SSS | TAC | VENDOR | LONGITUDE | LATITUDE | STYLE | AZIMUTH | HEIGHT | ELECTTI |
|---------|-----------|-------------|----------|-------------|--------|-----|-----|-----|-------|--------|------------|-----------|-------|---------|--------|---------|
| sanxia | 000000-0 | A池苏门-HLHF-1 | | 0 A池苏门-HLHF | 38400 | 50 | 50 | 50 | 14531 | 华为 | 112.770680 | 33.810393 | 宏站 | | 50 | 50 |
| yiayang | 11317-128 | F阳上庄F-HLH-1 | 11317 | F阳上庄F-HLH | 38400 | 373 | 0 | 0 | 14400 | 华为 | 112.980000 | 33.681400 | 宏站 | | 10 | (Null) |
| yiayang | 11317-129 | F阳上庄F-HLH-2 | 11317 | F阳上庄F-HLH | 38544 | 372 | 0 | 0 | 14400 | 华为 | 112.980000 | 33.681400 | 宏站 | | 140 | (Null) |
| yiayang | 11317-130 | F阳上庄F-HLH-3 | 11317 | F阳上庄F-HLH | 38544 | 374 | 0 | 0 | 14400 | 华为 | 112.980000 | 33.681400 | 宏站 | | 250 | (Null) |
| yiayang | 11429-128 | G安玉梅F-HLH-1 | 11429 | G安玉梅F-HLH | 38400 | 468 | 0 | 0 | 18740 | 华为 | 113.024000 | 33.720000 | 宏站 | | 340 | (Null) |
| yiayang | 11429-129 | G安玉梅F-HLH-2 | 11429 | G安玉梅F-HLH | 38400 | 470 | 0 | 0 | 18740 | 华为 | 113.024000 | 33.720000 | 宏站 | | 130 | (Null) |
| yiayang | 11429-130 | G安玉梅F-HLH-3 | 11429 | G安玉梅F-HLH | 38400 | 469 | 0 | 0 | 18740 | 华为 | 113.024000 | 33.720000 | 宏站 | | 260 | (Null) |

5.8.3 Insertion

1.将优化小区/保护带小区表中，小区 ID 为“246506-3”的小区的小区类型改为“优化区”。

【SQL 语句】:

```
UPDATE tboptcell
SET CELL_TYPE = '优化区'
WHERE
    SECTOR_ID = '246506-3'
```

【查询结果】:

| | | |
|----------|-------|-----|
| 246341-3 | 38544 | 保护带 |
| 246506-0 | 38098 | 保护带 |
| 246506-1 | 38098 | 保护带 |
| 246506-2 | 38098 | 保护带 |
| 246506-3 | 37900 | 优化区 |
| 246506-4 | 37900 | 保护带 |
| 246506-5 | 37900 | 保护带 |
| 246702-0 | 38544 | 保护带 |
| 246702-1 | 38544 | 保护带 |
| 246702-2 | 38544 | 保护带 |
| 246703-0 | 38400 | 保护带 |
| 246703-1 | 38400 | 保护带 |

2.用小区 PCI 优化调整结果表中“优化调整后的本小区 PCI 值”，替换小区/基站工参表中小区的“物理小区标识”。

【SQL 语句】:

```
UPDATE tbcell
SET PCI =( SELECT PCI
FROM tbpciassignment AS tb1
WHERE tbcell.SECTOR_ID = tb1.SECTOR_ID );
```

【查询结果】:

```
UPDATE tbcell
SET PCI =( SELECT PCI
FROM tbpciassignment AS tb1
WHERE tbcell.SECTOR_ID = tb1.SECTOR_ID )
> Affected rows: 5505
> 时间: 1.12s
```

3.针对路测 ATU C2I 干扰矩阵表表，使用 case 语句作出如下修改：如果主小区与干扰小区为同站 小区且干扰强度排序不小于 1，则干扰强度排序减 1；如果主小区与干扰小区不为同站，干扰 强度排序加 1。

【SQL 语句】:

```
UPDATE tbatuc2i AS T
SET T.RANK =CASE
WHEN T.COSITE = 1 AND T.RANK >= 1 THEN
T.RANK - 1
ELSE T.RANK + 1
END;
```

【查询结果】:

```

UPDATE tbatuc2i AS T
SET T.RANK =CASE
                WHEN T.COSITE = 1 AND T.RANK >= 1 THEN
                    T.RANK - 1
                ELSE T.RANK + 1
            END
> Affected rows: 1227
> 时间: 0.02s

```

| SECTOR_ID | NCELL_ID | RATIO_ALL | RANK | COSITE |
|-----------|------------|-----------|------|--------|
| 15113-129 | 253890-1 | 51.4442 | 2 | 0 |
| 15113-129 | 253899-0 | 7.02191 | 4 | 0 |
| 15113-129 | 253904-1 | 0 | 7 | 0 |
| 15113-129 | 253914-1 | 26.1454 | 3 | 0 |
| 15113-129 | 253935-1 | 5.07968 | 5 | 0 |
| 15113-129 | 253936-1 | 0 | 8 | 0 |
| 15113-129 | 259775-1 | 0.398406 | 6 | 0 |
| 238397-1 | 15578-130 | 0 | 7 | 0 |
| 238397-1 | 238397-0 | 2.73061 | 1 | 1 |
| 238397-1 | 238397-2 | 0.137678 | 1 | 1 |
| 238397-1 | 253904-1 | 0 | 6 | 0 |
| 238397-1 | 253921-2 | 3.57962 | 5 | 0 |
| 238397-1 | 253927-2 | 20.8811 | 4 | 0 |
| 238397-1 | 253931-0 | 30.771 | 2 | 0 |
| 238397-1 | 259772-2 | 21.2483 | 3 | 0 |
| 238397-2 | 238397-1 | 5.52359 | 1 | 1 |
| 238397-2 | 253806-1 | 0 | 8 | 0 |
| 238397-2 | 253893-0 | 12.6007 | 2 | 0 |
| 238397-2 | 253905-1 | 0 | 7 | 0 |
| 238397-2 | 253930-129 | 0 | 9 | 0 |
| 238397-2 | 259772-2 | 0.057537 | 4 | 0 |
| 238397-2 | 259778-1 | 0.057537 | 5 | 0 |
| 238397-2 | 5641-129 | 0 | 6 | 0 |
| 238397-2 | 7400-130 | 11.0472 | 3 | 0 |
| 253890-0 | 15113-128 | 0.442478 | 4 | 0 |
| 253890-0 | 15113-129 | 10.8723 | 3 | 0 |
| 253890-0 | 15113-130 | 0.094817 | 5 | 0 |
| 253890-0 | 253890-1 | 1.80152 | 1 | 1 |
| 253890-0 | 253890-2 | 2.93932 | 1 | 1 |
| 253890-0 | 253914-2 | 0.031606 | 6 | 0 |
| 253890-0 | 253934-1 | 30.7206 | 2 | 0 |
| 253890-1 | 15113-129 | 14.5469 | 3 | 0 |
| 253890-1 | 253890-0 | 2.06041 | 1 | 1 |
| 253890-1 | 253890-2 | 2.29234 | 1 | 1 |
| 253890-1 | 253891-0 | 0 | 16 | 0 |
| 253890-1 | 253891-2 | 1.51025 | 10 | 0 |

五、实验总结

- 1.From 子句中，嵌套使用时，表必须要使用别名。
- 2.重新导入表之前，需要先清空表，不然会重复。
- 3.order by 语句必须在 group up 之后使用。
- 4.except 可以在 sqlserver 和 oracle 中使用 但是在 mysql 中没有 except/minus; 解决办法：考虑采用 not in 方法作为替代

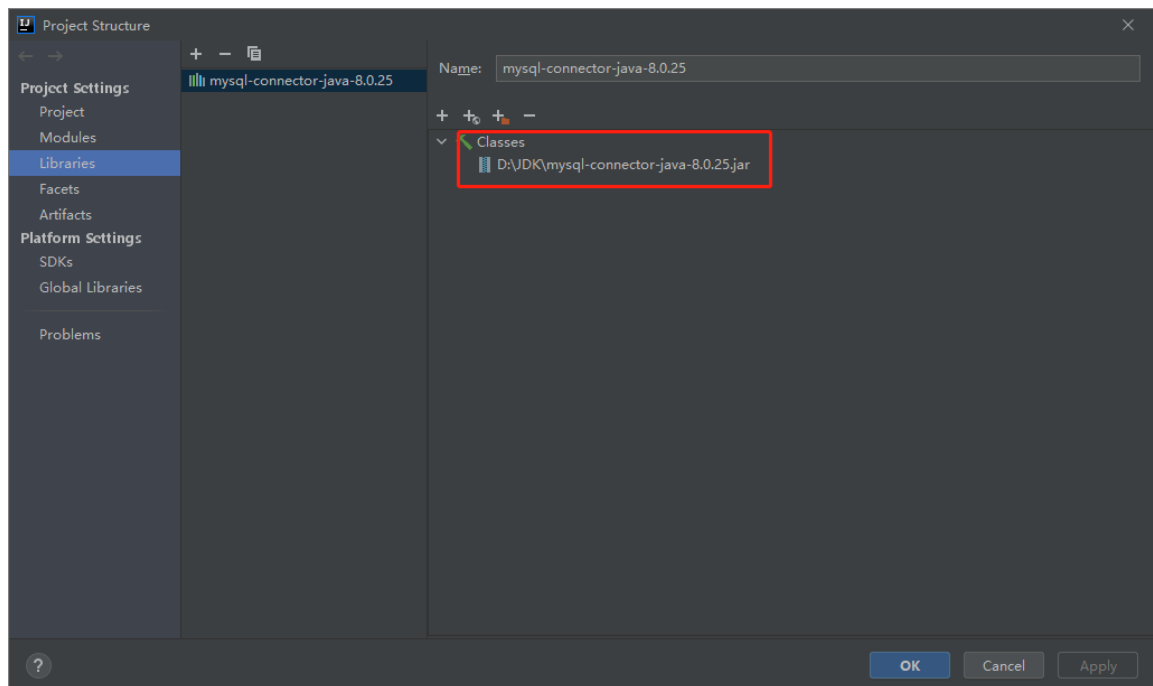
数据库访问接口实验

一、环境配置

本次实验中提供了 ODBC、JDBC、python 三种用于数据库接口连接的方式, 鉴于 ODBC 与 JDBC 已经在课本中进行过详尽的学习, 在这里我们进一步学习并采用 Java 来实现本次实验。

本次实验开始前的环境为 JDK14、Windows10、IntelliJ IDEA 2020.2.3 x64。

首先我们下载 mysql 的 jdbc 驱动, 在新建的 Java 项目中加入该驱动。



二、连接时长的获取和修改

在 JDBC 中 Statement 类可以通过 `getQueryTimeout()` 获取驱动程序等待 Statement 对象执行的秒数。`setQueryTimeout()` 来限制 statement 的执行时长。即 `java.sql.Statement.setQueryTimeout(int timeout)`, 在创建 statement 实例后直接输入限制的时长即可。

可用相应的函数获取和修改连接时长（连接数据库成功后的语句执行时间）

2.1 通过 API 函数获取数据库默认连接时间

部分代码如下：

```
Class.forName(driverName);
Connection conn=DriverManager.getConnection(dbURL,userid,passwd);
Statement stmt=conn.createStatement();
Integer a = stmt.getQueryTimeout();
System.out.println(a);
stmt.close();
conn.close();
```

得到结果：

```
D:\JDK\bin\java.exe "-javaagent:D:\IntelliJ IDEA 2020.2.3\
0
```

默认连接时长为 0 表示无限制

2.2 在默认时长下观察是否会超时

完成查询 1：（对 tbAdjCell 的连接访问）根据 tbAdjCell 表给出的小区 1 阶邻区关系，计算小区间的二阶邻区表 tbSecAdjCell。

原理：如果 $\langle a,b \rangle \in \text{tbAdjCell}$, $\langle a,c \rangle \in \text{tbAdjCell}$ ，即 b 为 a 的 1 阶邻区，c 为 a 的 1 阶邻区，则 b 与 c 间存在 2 阶邻区关系。

方法：

```
CREATE TABLE tbNewSecAdjCell
Select T1.N_Sector_ID, T2.N_Sector_ID as S_Sector_ID
From tbAdjCell as T1, tbAdjCell as T2
Where T1.S_Sector_ID = T2.S_Sector_ID
```

部分代码如下：

```
Class.forName(driverName);
Connection conn=DriverManager.getConnection(dbURL,userid,passwd);
Statement stmt=conn.createStatement();
System.out.println(new Date());
boolean rset=stmt.execute( sql: "create table tbNewSecAdjCell Select
System.out.println(new Date());
stmt.close();
conn.close();
```

运行结果：

```
Wed Jun 09 00:14:38 CST 2021
Wed Jun 09 00:16:15 CST 2021
```

发现语句运行了 97 秒。

新建表的部分内容：

| N_Sector_ID | S_Sector_ID |
|-------------|-------------|
| 124673-1 | 124673-1 |
| 124673-1 | 124673-2 |
| 124673-1 | 124683-1 |
| 124673-1 | 124711-0 |
| 124673-1 | 124711-1 |
| 124673-1 | 124711-2 |
| 124673-1 | 124812-0 |
| 124673-1 | 124812-2 |
| 124673-1 | 15290-128 |
| 124673-1 | 15290-129 |
| 124673-1 | 253901-0 |
| 124673-1 | 253930-130 |

2.3 降低默认连接时间，观察是否超时

将连接时间设置为 10:

```
Class.forName(driverName);
Connection conn=DriverManager.getConnection(dbURL,userid,passwd);
Statement stmt=conn.createStatement();
stmt.setQueryTimeout(10);
int a = stmt.getQueryTimeout();
System.out.println("Timeout: " + Integer.toString(a));
System.out.println(new Date());
boolean rset=stmt.execute( sql: "create table tbNewSecAdjCell Select
System.out.println(new Date());
stmt.close();
conn.close();
```

运行结果:

```
Timeout: 10
Wed Jun 09 00:22:54 CST 2021
Exception:com.mysql.cj.jdbc.exceptions.MySQLTimeoutException: Statement cancelled due to timeout or client request
```

程序抛出超时异常，查询失败。

三、查询

查询如下语句并打印结果:

```
SELECT * FROM tbcell WHERE SECTOR_ID LIKE '124681-%';
```

全部代码如下:

```
import java.sql.*;
public class API {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String driverName = "com.mysql.cj.jdbc.Driver";
```



```

        String dbURL = "jdbc:mysql://localhost:3306/tb?useUnicode=true&
characterEncoding=UTF-8&serverTimezone=UTC";
        String userid = "root"; //默认用户名
        String passwd = "123456"; //密码
        try
        {
            Class.forName(driverName);
            Connection conn=DriverManager.getConnection(dbURL,userid,passwd);
            Statement stmt=conn.createStatement();
            ResultSet rset=stmt.executeQuery(
"SELECT * FROM tbcell WHERE SECTOR_ID LIKE '124681-%'");
            while(rset.next())
            {
                System.out.println(rset.getString("SECTOR_ID") + " " +
rset.getString("SECTOR_NAME"));
            }
            stmt.close();
            conn.close();
        }
        catch(Exception sqle)
        {
            System.out.println("Exception:"+sqle);
        }
    }
}

```

执行结果如下：

```

124681-0 A池杨大池东洼-HLHF-1
124681-1 A池杨大池东洼-HLHF-2
124681-2 A池杨大池东洼-HLHF-3

Process finished with exit code 0

```

执行成功

四、插入

向数据库的 tbcell 表中插入一行数据，若插入失败则会抛出异常。

插入语句如下：

```

INSERT INTO tbcell(CITY, SECTOR_ID, SECTOR_NAME, ENODEBID,
ENODEB_NAME) values('beijing','0000001','海淀-HLHF-1',000001,'海淀-
HLHF');

```

若插入成功，则通过 SELECT 查询该数据。

```

import java.sql.*;

```

```

public class API {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String driverName = "com.mysql.cj.jdbc.Driver";
        String dbURL = "jdbc:mysql://localhost:3306/tb?useUnicode=true&
characterEncoding=UTF-8&serverTimezone=UTC";
        String userid = "root"; //默认用户名
        String passwd = "123456"; //密码
        try
        {
            Class.forName(driverName);
            Connection conn=DriverManager.getConnection(dbURL,userid,passwd);
            Statement stmt=conn.createStatement();
            try
            {
                stmt.executeUpdate(
"INSERT INTO tbcell(CITY, SECTOR_ID, SECTOR_NAME, ENODEBID, ENODEB_NAME)
values('beijing','0000001','海淀-HLHF-1',000001,'海淀-HLHF');");
            }catch(SQLException sqle)
            {
                System.out.println("Could not insert tuple."+sqle);
            }
            ResultSet rset=stmt.executeQuery(
"SELECT * FROM tbcell WHERE CITY LIKE 'beijing';");
            while(rset.next())
            {
                System.out.println(rset.getString("SECTOR_ID") + " " +
rset.getString("SECTOR_NAME"));
            }
            stmt.close();
            conn.close();
        }
        catch(Exception sqle)
        {
            System.out.println("Exception:"+sqle);
        }
    }
}

```

结果如下：

```
0000001 海淀-HLHF-1

Process finished with exit code 0
```

| CITY | SECTOR_ID | SECTOR_NAME | ENODEBID | ENODEB_NAME | E |
|---------|-----------|-------------|----------|-------------|---|
| beijing | 0000001 | 海淀-HLHF-1 | 1 | 海淀-HLHF | |

按照图中执行结果可以看出，向数据库中插入一行 CITY = Beijing 的数据后，再次 select 得到这一刚刚插入的数据。执行成功。

五、修改

仿照先前的操作对刚刚向数据库中插入的元组进行修改。注意到刚刚插入的数据中对于 PCI 的值为空，现在我们对这一信息进行更新。若更新失败则会抛出异常。

修改语句如下：

```
UPDATE tbcell SET EARFCN=12345, PCI=32, PSS=1, SSS=10, TAC=10000 WHERE CITY LIKE 'beijing';
```

若更新成功，则通过 SELECT 查询该数据。

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

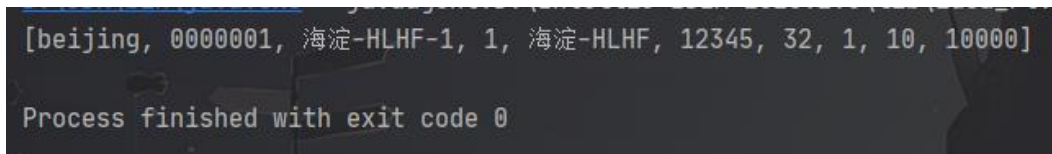
public class API {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String driverName = "com.mysql.cj.jdbc.Driver";
        String dbURL = "jdbc:mysql://localhost:3306/tb?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC";
        String userid = "root"; //默认用户名
        String passwd = "123456"; //密码
        try
        {
            Class.forName(driverName);
            Connection conn=DriverManager.getConnection(dbURL, userid, passwd);
            Statement stmt=conn.createStatement();
            try
            {
                stmt.executeUpdate(
"UPDATE tbcell SET EARFCN=12345, PCI=32, PSS=1, SSS=10, TAC=10000 WHERE CITY LIKE 'beijing';");
            }catch(SQLException sqle)
            {
                System.out.println("Could not insert tuple."+sqle);
            }
        }
    }
}
```

```

        ResultSet rset=stmt.executeQuery(
"SELECT * FROM tbcell WHERE CITY LIKE 'beijing'");
        while(rset.next())
        {
            List<String> res = new ArrayList<String>();
            for (int i = 1; i <= 10; ++i){
                res.add(rset.getString(i));
            }
            System.out.println(res);
        }
        stmt.close();
        conn.close();
    }
    catch(Exception sqle)
    {
        System.out.println("Exception:"+sqle);
    }
}

```

运行结果



```

[beijing, 0000001, 海淀-HLHF-1, 1, 海淀-HLHF, 12345, 32, 1, 10, 10000]
Process finished with exit code 0

```

| CITY | SECTOR_ID | SECTOR_NAME | ENODEBID | ENODEB_NAME | EARFCN | PCI | PSS | SSS | TAC |
|---------|-----------|-------------|----------|-------------|--------|-----|-----|-----|-------|
| beijing | 0000001 | 海淀-HLHF-1 | 1 | 海淀-HLHF | 12345 | 32 | 1 | 10 | 10000 |

修改成功。

六、删除

向数据库的 tbcell 表中删除一行数据，若删除失败则会抛出异常。

删除语句如下：

```
DELETE FROM `tbcell` WHERE CITY='beijing' ;
```

若删除成功，则通过 SELECT 查询该数据。并判断返回结果是否为空

```

ResultSet rset=stmt.executeQuery( sql: "SELECT * FROM tbcell WHERE CITY LIKE 'beijing'");
if(rset.getRow() == 0){
    System.out.println("查询结果为空!");
}

```

代码如下：

```

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

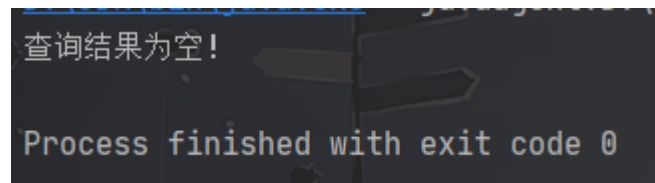
```

```

public class API {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String driverName = "com.mysql.cj.jdbc.Driver";
        String dbURL =
"jdbc:mysql://localhost:3306/tb?useUnicode=true&characterEncoding=UTF-
8&serverTimezone=UTC";
        String userid = "root"; //默认用户名
        String passwd = "123456"; //密码
        try
        {
            Class.forName(driverName);
            Connection conn=DriverManager.getConnection(dbURL,userid,passwd);
            Statement stmt=conn.createStatement();
            try
            {
                stmt.executeUpdate("DELETE FROM `tbcell` WHERE
CITY='beijing'");
            }catch(SQLException sqle)
            {
                System.out.println("Could not insert tuple."+sqle);
            }
            ResultSet rset=stmt.executeQuery(
"SELECT * FROM tbcell WHERE CITY LIKE 'beijing'");
            if(rset.getRow() == 0){
                System.out.println("查询结果为空!");
            }
            stmt.close();
            conn.close();
        }
        catch(Exception sqle)
        {
            System.out.println("Exception:"+sqle);
        }
    }
}

```

结果如下：



```

查询结果为空!
Process finished with exit code 0

```

七、实验总结

在本次实验中通过 Java 所提供的 jdbc 接口实现对数据库的访问。首先下载 mysql 的驱动文件并将其导入到项目中。而后通过 vscode IDE 编写相关 python 脚本程序，通过 python3 指令执行验证。

实验过程中进行了对于连接时长的设置，增删改查操作的实现。

在选用的 Java 方法中，使我更进一步掌握到了数据库的连接函数方法、用于数据库一般语句执行的 excute 函数和 executeQuery 函数等、用于获取查询结果的 ResultSet 类。

实验过程中使我们掌握了如何通过外部程序连接到数据库并对数据库进行相关操作的方法，了解数据库应用程序设计采用的 ODBC、JDBC 等多种接口，掌握接口及相关软件配置能力。编写数据库应用程序，通过数据库访问接口访问数据库，培养数据库应用程序开发能力。我将这一过程总结为两部分重要内容，一方面是基础的对于数据库直接操作的 sql 语句的编写，另一方面是如何通过配置并编写相关的程序语句来执行和运用这些语句。后者是本次实验的关键内容。

数据库物理设计实验

一、实验目的

- 1.理解 SQL SERVER 数据库的物理设计的几个主要部分。
- 2.掌握 SQL SERVER 数据库的文件（包括主文件、辅助文件和日志文件）和文件组（主文件组、其他文件组）的设计。
- 3.掌握 SQL SERVER 数据库索引的设计，包括聚集索引、非聚集索引和复合索引等，分析索引对 select/update/insert/delete 的影响。
- 4.掌握利用 SQL Server Management Studio 分析观察 SQL 语句的查询执行计划，对比查询需求相同、实现方式不一样的不同 SQL 语句在查询结果、查询执行计划、执行时间上的区别。
- 5.了解掌握定长、变长属性对关系表存储空间和访问速度的影响。

二、实验环境

采用 SQL Server 数据库管理系统作为实验平台。SQL Server 2012

三、实验内容

了解数据库文件及文件组设计，理解不同类型文件和文件组的区别和工作机制，掌握查看数据库文件存储格式和创建文件及文件组的方法。

索引是物理设计的重要内容，有助于提高与索引相关部分查询的执行效率，但会增加空间开销和数据增删改的成本，而对不相关查询的效率则无影响。所以在实际数据库物理设计中需要综合平衡考虑后决定要建立什么样的索引。

设计 2 条 SQL 语句，这 2 条语句查询需求相同，但 1 条使用索引，另 1 条不使用索引。将这 2 条 SQL 语句同时提交给 DBMS，比较它们在执行效果、执行速度、查询执行计划方面的区别。

设计 2 条 SQL 语句，这 2 条语句 insert/update/delete 需求相同，但 1 条使用索引，另 1 条不使用索引。将这 2 条 SQL 语句同时提交给 DBMS，比较它们在执行效果、执行速度、查询执行计划方面的区别。

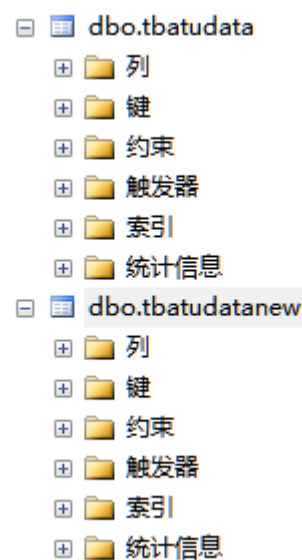
设计 2 条 SQL 语句，这 2 条语句查询需求相同，但 1 条使用聚集索引，另 1 条使用非聚集索引。将这 2 条 SQL 语句同时提交给 DBMS，比较它们在执行效果、执行速度、查询执行计划方面的区别。

四、实验步骤

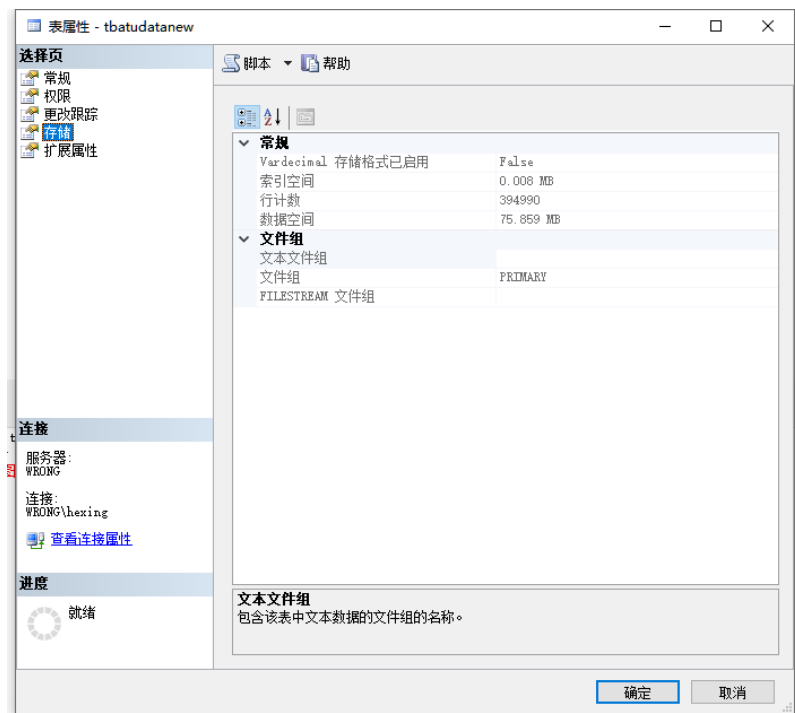
4.1 数据库索引的设计及对 select 操作的影响分析

4.1.1 创建 tbATUData 的备份表 tbATUDataNew

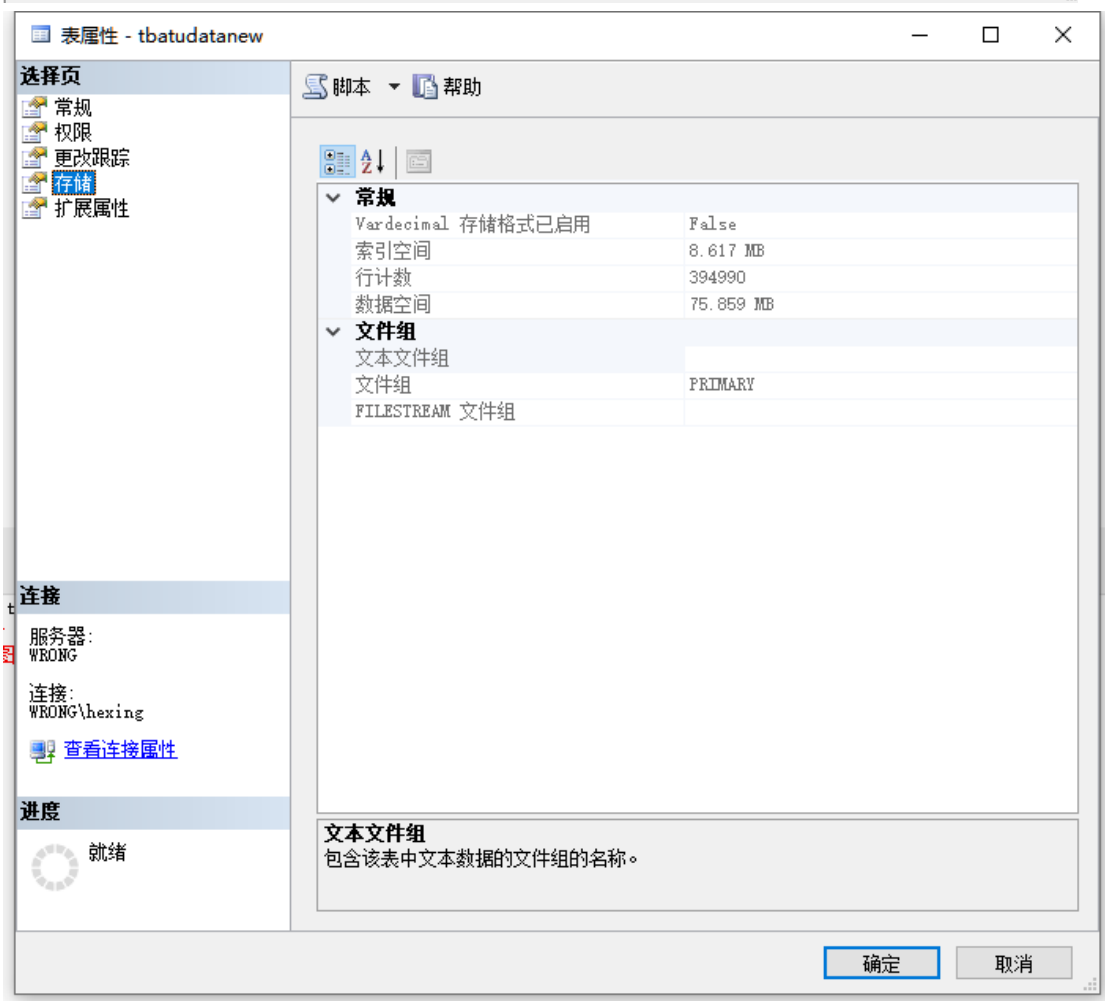
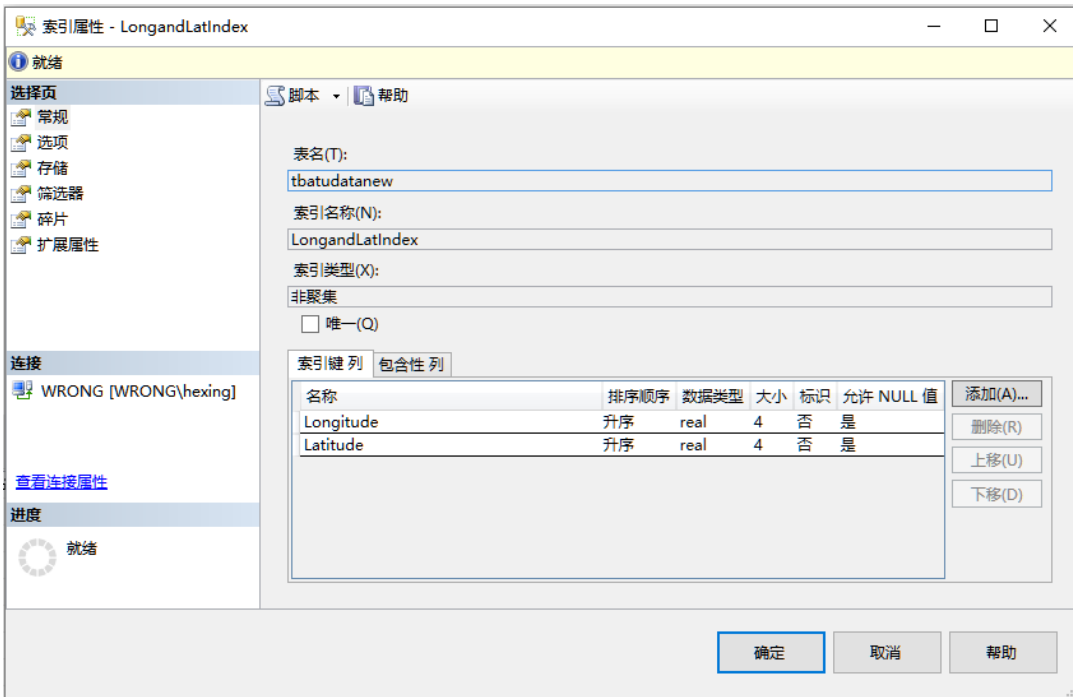
```
CREATE TABLE tbatudatanew SELECT
*
FROM
    tbatudata;
```



4.1.2 查看表属性里面的索引和数据存储空间



4.1.3 在 tbATUDatanew 创建索引



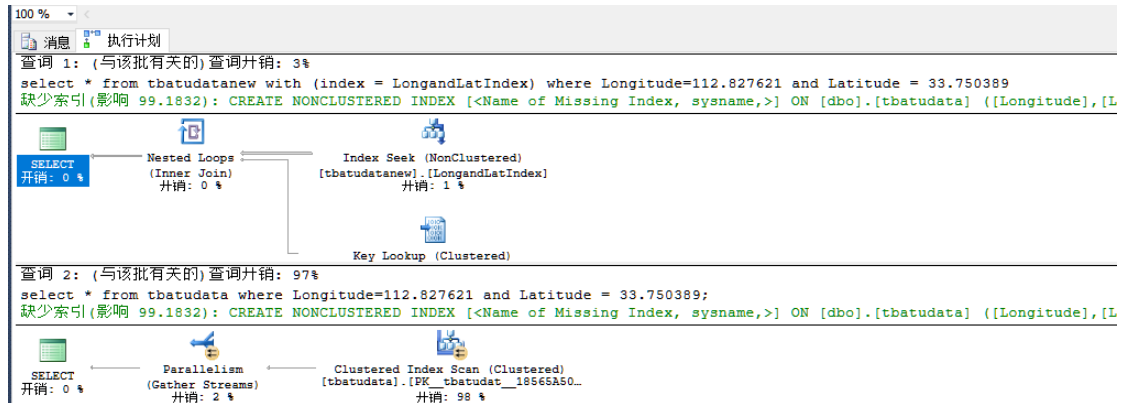
4.1.4 编写 select 语句，使用索引访问 tbATUDatanew；编写 select 语句，不使用索引方式，访问 tbATUData，将 2 条 select 语句，同时提交给 DBMS 执行后，从 2 个执行结果

窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

【SQL 语句】:

```
select * from tbatudatanew with (index = LongandLatIndex) where Longitude=112.827621
and Latitude = 33.750389
select * from tbatudata where Longitude=112.827621 and Latitude = 33.750389;
```

【执行计划】:



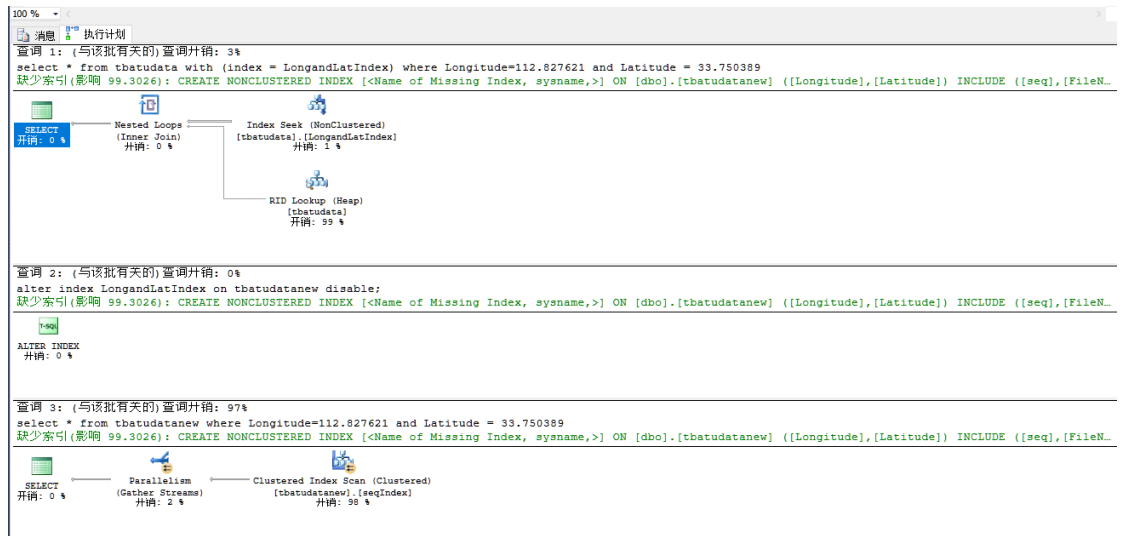
【分析】: 从执行计划可以看出，使用索引的查询开销占 3%，不使用索引的查询开销 97%，可以看到使用索引可以明显降低查询开销

4.1.5 编写 select 语句，强制使用索引访问 tbATUDatanew; 编写 select 语句，强制不使用索引方式，访问 tbATUDData，将 2 条 select 语句，同时提交给 DBMS 执行后，从 2 个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

【SQL 语句】:

```
select * from tbatudata with (index = LongandLatIndex) where Longitude=112.827621 and
Latitude = 33.750389
alter index LongandLatIndex on tbatudatanew disable;
select * from tbatudatanew where Longitude=112.827621 and Latitude = 33.750389
```

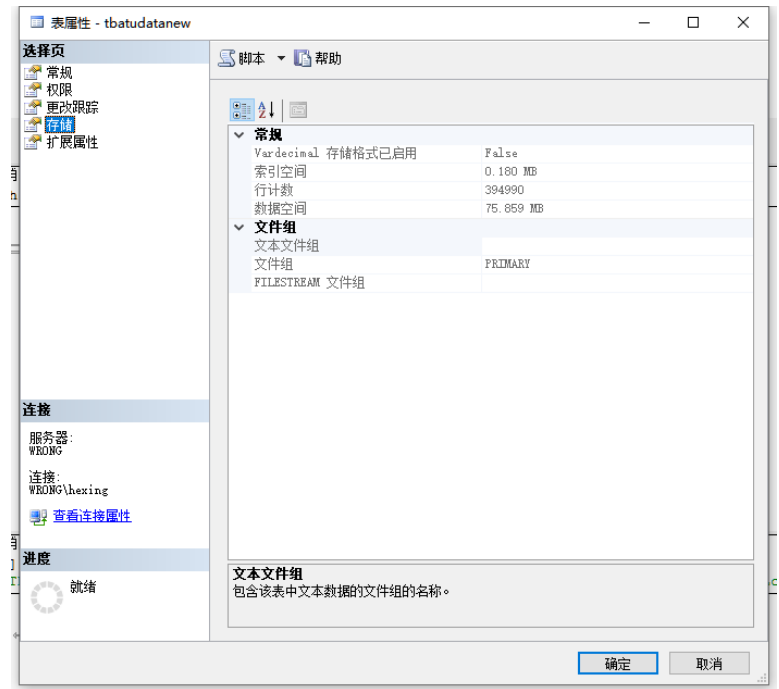
【执行计划】:



【分析】：强制使用索引的查询开销占 3%，强制不使用索引的查询开销占 97%，执行结果一致，强制使用索引开销占比少，使用索引查询速度更快，cost 更小

4.1.6 查看索引空间和数据空间

这里的索引空间大小是，做过 4.1.5 之后，禁用了索引 LongandLatIndex 后的结果，若重新生成该索引，则索引空间占比应该为 8M 多



4.2 数据库索引的设计及对 insert/update/delete 操作的影响分析

4.2.1 创建 tbAdjCell 的备份表 tbAdjCellNew

系统表

FileTables

dbo.instructor

dbo.tbadjcell

dbo.tbadjcellnew

列

键

约束

触发器

索引

统计信息

dbo.tbatauc2i

dbo.tbatudata

dbo.tbatudatanew

dbo.tbatauhandover

dbo.tbtc2i

dbo.tbcell

dbo.tbcell_traffic

dbo.tbhandover

dbo.tbmrodata

dbo.tboptcell

dbo.tbpcassignment

dbo.tbsecadjcell

视图

同义词

可编程性

Service Broker

存储

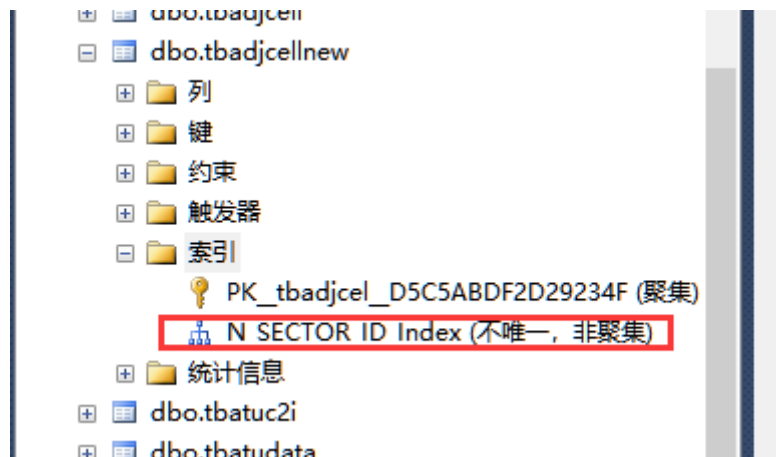
安全性

100 %

结果 消息

| | S_SECTOR_ID | N_SECTOR_ID | S_EARFCN | N_EARFCN |
|----|-------------|-------------|----------|----------|
| 1 | 123449 | 123449 | 38400 | 38400 |
| 2 | 124673-0 | 124673-1 | 38400 | 38400 |
| 3 | 124673-0 | 124673-2 | 38400 | 38400 |
| 4 | 124673-0 | 124683-1 | 38400 | 38400 |
| 5 | 124673-0 | 124711-0 | 38400 | 38400 |
| 6 | 124673-0 | 124711-1 | 38400 | 38400 |
| 7 | 124673-0 | 124711-2 | 38400 | 38400 |
| 8 | 124673-0 | 124812-0 | 38400 | 38400 |
| 9 | 124673-0 | 124812-2 | 38400 | 38400 |
| 10 | 124673-0 | 15290-128 | 38400 | 38400 |
| 11 | 124673-0 | 15290-129 | 38400 | 38400 |
| 12 | 124673-0 | 253901-0 | 38400 | 38400 |
| 13 | 124673-0 | 253930-130 | 38400 | 38400 |
| 14 | 124673-0 | 259595-0 | 38400 | 38400 |
| 15 | 124673-0 | 259595-1 | 38400 | 38400 |
| 16 | 124673-0 | 259654-1 | 38400 | 38400 |

4.2.2 在 tbAdjCellNew 创建索引

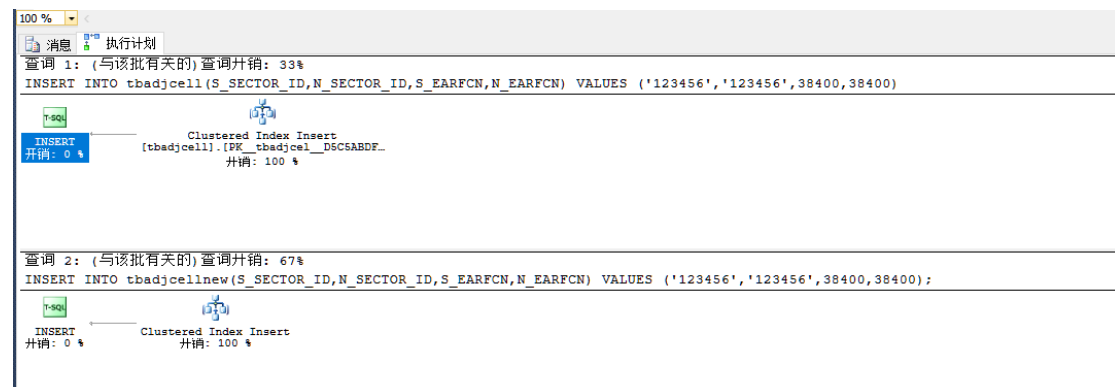


4.2.3 编写 insert 语句，添加一条元组数据（包含索引）；在 tbATUData 编写同样的一条 insert 语句，将 2 条语句同时提交给 DBMS 执行后，从 2 个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

【SQL 语句】：

```
INSERT INTO tbAdjCell(S_SECTOR_ID,N_SECTOR_ID,S_EARFCN,N_EARFCN)
VALUES ('123456','123456',38400,38400)
INSERT INTO tbAdjCellNew(S_SECTOR_ID,N_SECTOR_ID,S_EARFCN,N_EARFCN)
VALUES ('123456','123456',38400,38400);
```

【执行计划】：



【分析】：

由于索引的存在，使用 insert 语句在关系表中插入新元组后，DBMS 将根据新插入的元组在索引属性上的值，对表中的索引进行调整重组，必然引起额外系统开销，降低 insert 的执行速度。如图可以看出，不使用索引的查询开销占比 33%，使用索引的查询开销占比为 67%，不使用索引的关系表执行插入语句会引起额外的系统开销，速度也会变慢。

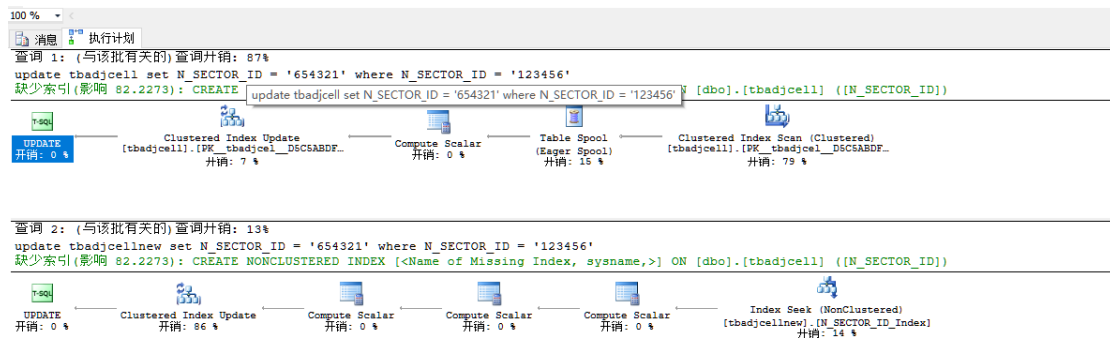
4.2.4 编写 update 语句，修改一条元组数据（包含索引）；在 tbATUData 编写同样的一条 update 语句，将 2 条语句同时提交给 DBMS 执行后，从 2 个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

4.2.4.1 修改索引的 update 操作

【SQL 语句】:

```
update tbadjcell set N_SECTOR_ID = '654321' where N_SECTOR_ID = '123456'  
update tbadjcellnew set N_SECTOR_ID = '654321' where N_SECTOR_ID = '123456'
```

【执行计划】:



【分析】:

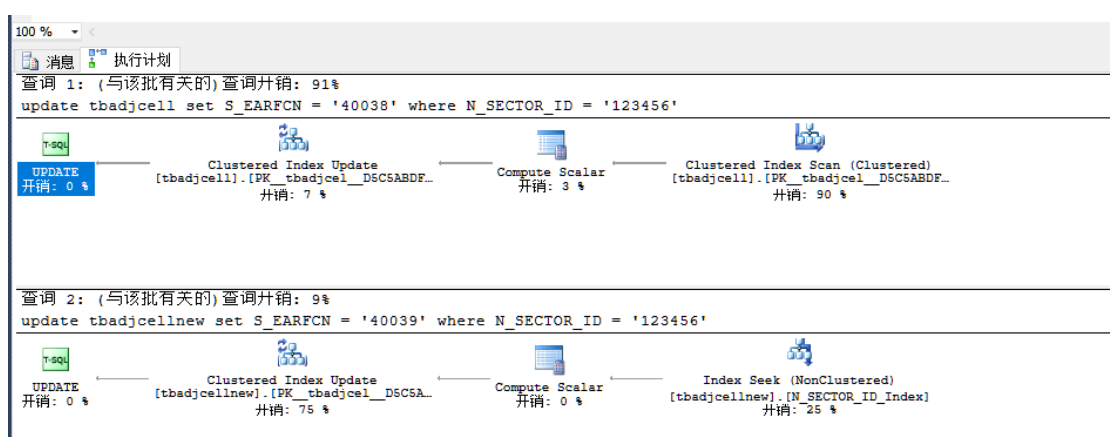
对 update-set-where 操作, 如果在 where 查询条件属性上建立了索引, 有利于在数据库文件中快速定位 update 需要访问的元组。但如果 set 操作修改了索引涉及到的属性的值, 则会引发 DBMS 对索引的重组。索引重组带来额外的系统开销, 降低了 update 的执行速度。这里可以看出, SQL 语句中 where 查询条件属性上建立了索引, 本应该会极大的加快查询速度, 但由于 set 语句又对索引涉及的属性 N_SECTOR_ID 进行了修改, 所以查询开销还是又相对于增大了很多。

4.2.4.2 不修改索引的 update 操作

【SQL 语句】:

```
update tbadjcell set S_EARFCN = '40038' where N_SECTOR_ID = '123456'  
update tbadjcellnew set S_EARFCN = '40039' where N_SECTOR_ID = '123456'
```

【执行计划】:



【分析】:

这里的 set 语句并没有对索引涉及的属性值进行修改, 因此由于索引的存在, update 操作能够快速定位需要访问的元组, update 执行速度明显变快, 查询开销仅占 9%, 而不使用索引的 update 操作上查询开销占比高达 91%。

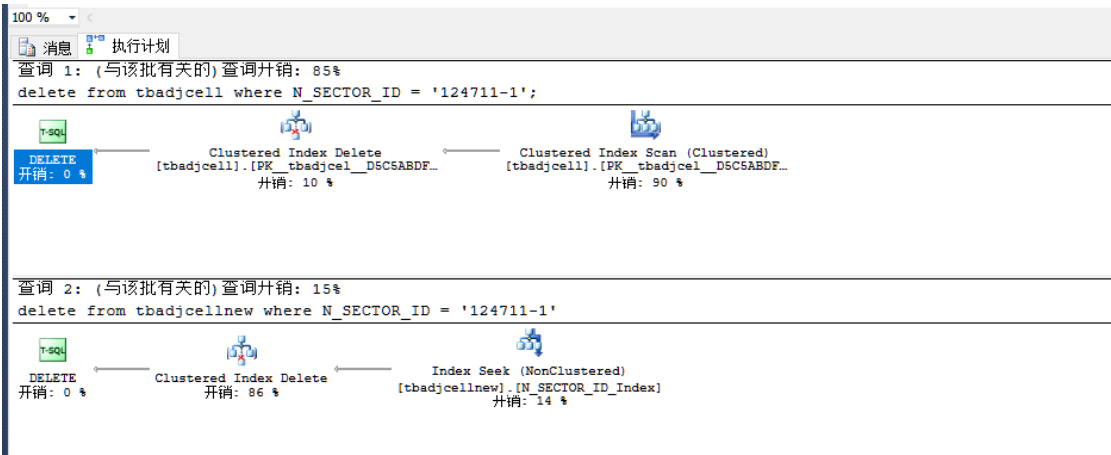
4.2.5 编写 delete 语句, 删除第三步添加的数据; 在 tbATUDData 编写同样的一条 delete 语句, 将 2 条语句同时提交给 DBMS 执行后, 从 2 个执行结果窗口中, 判断执行的结果

是否一致，并观察各自执行效果、查询执行计划、时间对比。

【SQL 语句】:

```
delete from tbadjcell where N_SECTOR_ID = '124711-1';
delete from tbadjcellnew where N_SECTOR_ID = '124711-1';
```

【执行计划】:



【分析】:

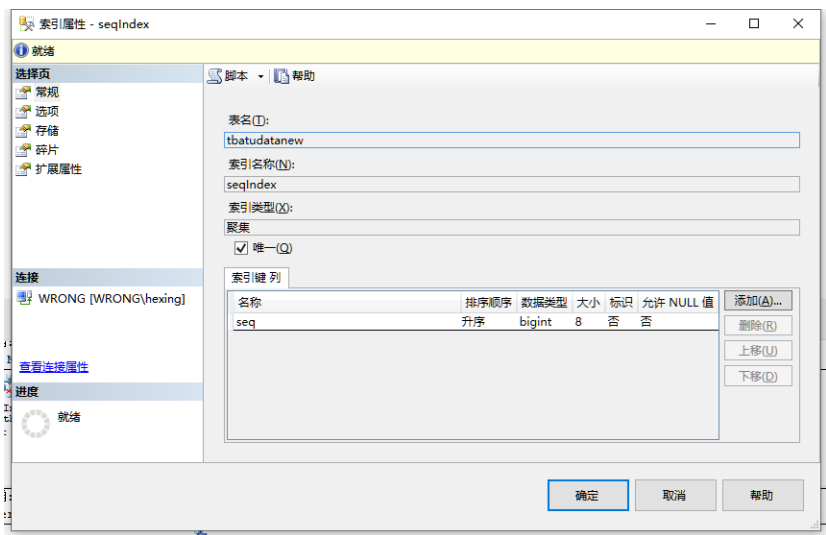
对 delete-from-where 操作，如果在 where 查询条件属性上建立了索引，有利于在数据库文件中快速定位由 where 条件定义的需要删除的元组。但删除这些元组后，将引起 DBMS 将对索引进行调整重组，引起额外的系统开销，可能会降低 delete 的执行速度。因此可以从图中看出，在建立了索引的新表 tbadjcellnew 上执行 delete 语句，查询开销仅占 15%，在其单独的过程中，Index Seek 操作仅开销占比 14%，而在没有索引的表上执行 delete 操作，Index Scan 操作就占比达到了 90%，但由于删除索引元素 N_SECTOR_ID 的元组后，引起额外的系统开销，delete 的执行速度下降。

4.3 聚集索引和非聚集索引的比较

4.3.1 创建 tbATUDData 的备份表 tbATUDDataNew

4.3.2 在 tbATUDatanew 创建聚集索引

创建以 seq 为索引键的聚集索引

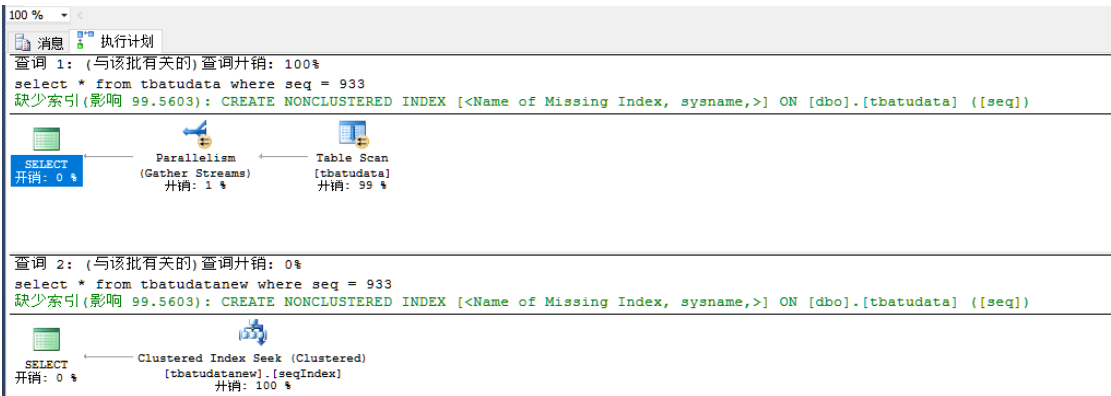


4.3.2.1 编写 select 语句，使用聚集索引访问 tbATUDatanew；编写 select 语句，不使用聚集索引方式，访问 tbATUDData，将 2 条 select 语句，同时提交给 DBMS 执行后，从 2 个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比。

【SQL 语句】:

```
select * from tbatudata where seq = 933
select * from tbatudatanew with(index = seqIndex) where seq = 933
```

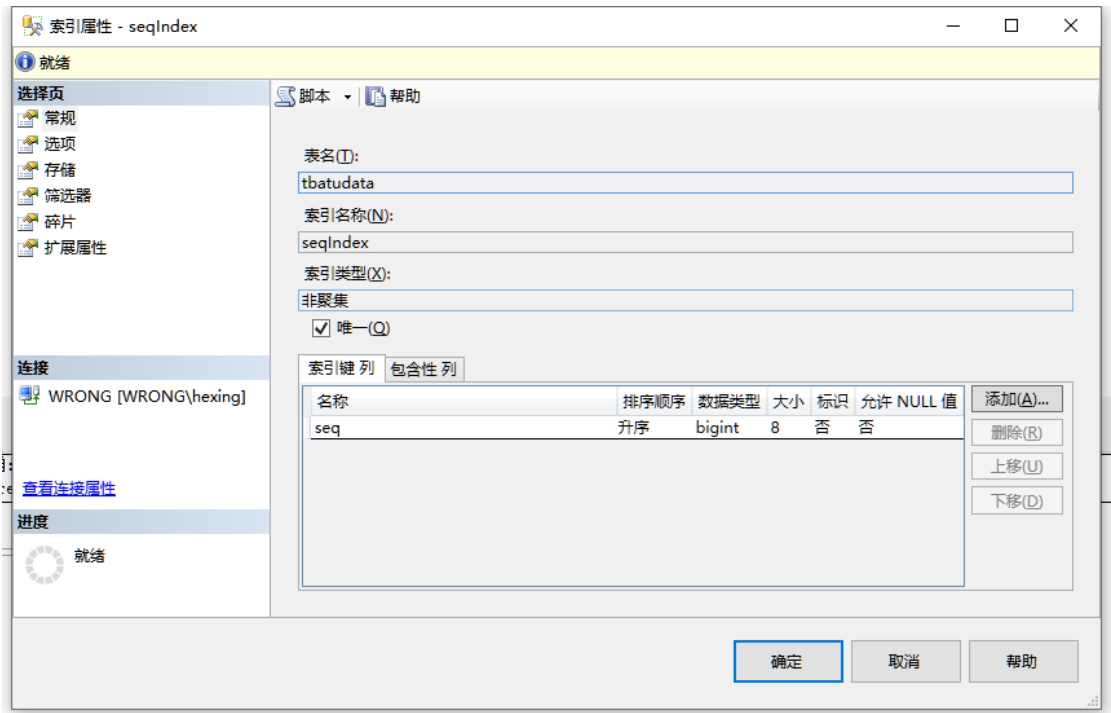
【执行计划】:

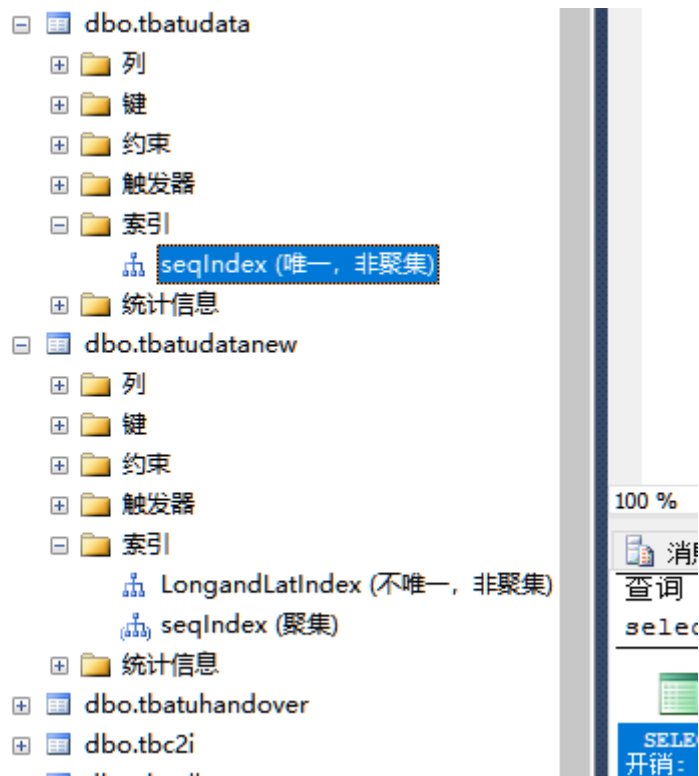


【分析】:

使用聚集索引访问的查询开销占比为 0%，不使用的为 100%，说明使用聚集索引后，由于索引提前建立完成，查询开销大幅减小。

4.3.3 在 tbATUDData 创建非聚集索引。



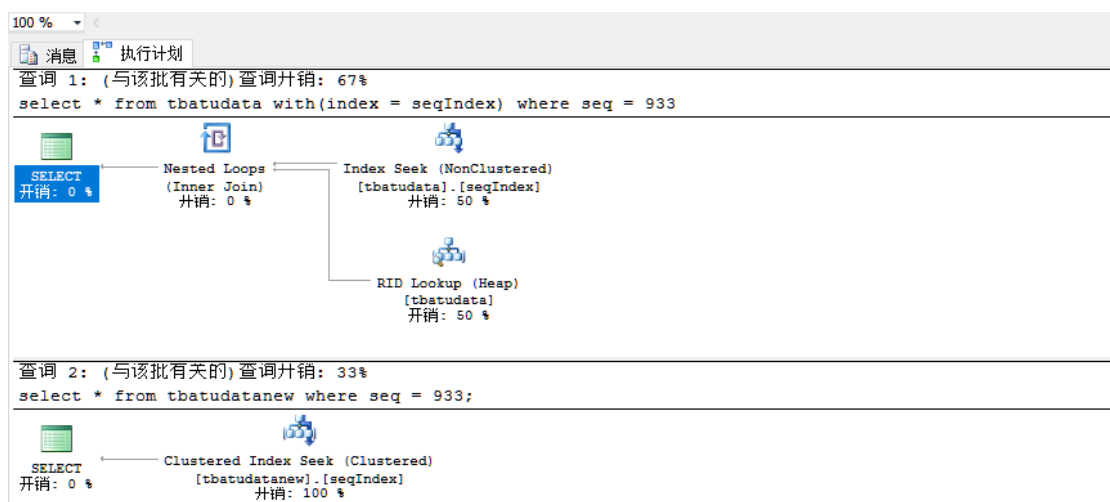


4.3.3.1 编写 select 语句，使用聚集索引访问 tbATUDatanew；编写 select 语句，使用非聚集索引方式，访问 tbATUData，将 2 条 select 语句，同时提交给 DBMS 执行后，从 2 个执行结果窗口中，判断执行的结果是否一致，并观察各自执行效果、查询执行计划、时间对比

【SQL】:

```
select * from tbatudata with(index = seqIndex) where seq = 933
select * from tbatudatanew where seq = 933;
```

【执行计划】:



【分析】:

不使用聚集索引的查询开销占比为 67%，使用聚集索引的查询开销占比为 33%，使用聚集索引的表上对于聚集索引键的查询速度更快，查询开销占比更小。非聚集索引叶节点仍然是索引节点，只是有一个指针指向对应的数据块，因此如果使用非聚集索引查询，而查询列

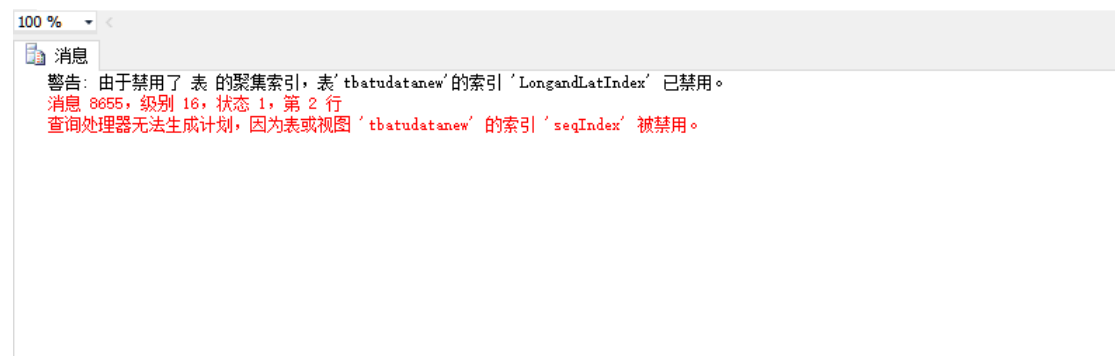
中包含了其他该索引没有覆盖的列，那么他还要进行第二次的查询，查询节点上对应的数据行的数据，这里的 Index Seek 就是索引占比时间，为 50%，RID Lookup 为二次查询时间，占比 50%，可以看的出二次查询所花费的查询开销占比很大。

4.3.4 强制不使用索引

【SQL 语句】:

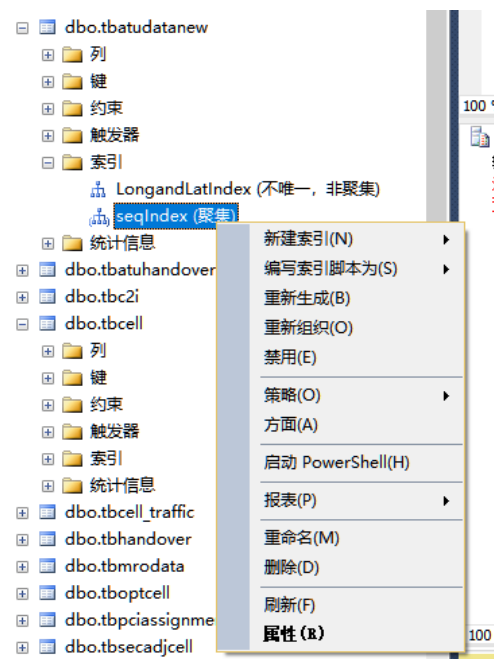
```
alter index seqIndex on tbatudatanew disable  
select * from tbatudatanew where seq = 933;
```

【执行计划】:



显示索引已被禁用，此时重新生成索引即可

```
alter index seqIndex on tbatudatanew rebuild
```



五、问题及解决

1.一开始使用 MySQL 以及 Navicat 做本次实验，做到中途发现，工具中无法显示执行计划和具体的两个 SQL 语句各自的查询开销占比，因此决定放弃 MySQL，采用 SQL Server 继续实验。

2. MySQL 迁移到 SQLServer 2012 耗费了一定的时间
3. 最开始没有将两个查询语句放在一起比较，得不到对比的开销占比图，后来仔细阅读指导书后得知需要两个一起提交运行
4. 实验指导书个别勘误，不知道该在哪个表上做实验

六、实验总结

在实验中，体会了索引对于数据库物理设计的重要性，体会了聚集索引和非聚集索引查询过程和开销，通过直观的开销占比，使得我对于数据库物理设计中索引设计的重要性有了更加深刻的认知，同时对于 update/delete/insert 操作和索引带来的影响有了更深的认识。

事务与并发控制

一、实验目的

通过单事务、串行事务、并发事务实验，了解 SQL SERVER 数据库系统中 1) 事务组成方式和执行模式；2) 对单事务和串行事务的原子性保障机制；3) 基于锁和隔离级别的事务并发控制和对并发事务的一致性、独立性保障机制。

具体目标为：

1. 通过 update 操作对现有关系表进行修改，观察违反 check 约束的 update 执行结果，理解 SQL SERVER 对单个事务提供的原子性保障机制；
2. 在 SQL SERVER 三种事务执行模式下，完成串行事务提交与回滚实验，对数据库表进行查询 (select)、更新 (update)、插入(insert)、删除(delete)，了解 SQL SERVER 事务组成方式和 commit/rollback 对查询结果的影响，了解数据库内容/状态发生变化时 SQL SERVER 提供的事务原子性的保障机制；
3. 在显式执行模式下，通过串行执行多个事务，对现有数据库表增加、删除属性列，观察 commit/rollback 对执行结果的影响，了解数据库模式结构发生变化时 SQL SERVER 提供的事务原子性保障机制；
4. 观察单事务或串行事务中保存点 (Save Point) 回滚 rollback 对数据库访问结果的影响，了解 SQL SERVER 保存点机制；
5. 通过 set XACT_ABORT ON/OFF 语句，控制多条 SQL 语句组成的单一事务的整体回滚、局部回滚，理解实际数据库系统中原子性保障机制与教科书中事务原子性概念的差异。
6. 观察 SQL SERVER 的加锁机制，包括对数据库、关系表、数据行等不同粒度的数据对象所施加的各种不同类型的锁；
7. 了解 SQL SERVER 提供的各种事务隔离级别。观察分析在“读提交”隔离级别下，并发事务执行导致的数据不一致性，如丢失修改（写-写错误）、读脏数据（写-读错误）、不可重复读（读-写错误）、幻象；
8. 分析对比多种隔离级别下，如“读提交”与“读未提交”、“可串行化”，并发事务执行导致的数据不一致性，了解 SQL SERVER 提供的事务一致性和独立性保障机制。
9. 观察单事务、串行事务、并发事务执行 update/insert/delete 等操作时，SQL Server 记录的日志类型和日志内容，了解 SQL Server 日志机制；观察分析当事务以 commit 结束、rollback 结束，和系统发生 crash 时，DBMS 采取的故障恢复动作日志内容。

二、实验环境

采用 SQL SERVER 2012 数据库管理系统作为实验平台。

三、实验步骤

11.1 单事务/串行事务提交与回滚

11.1.1 违反 check 约束的 update 操作

【实验要求】:

根据现网实际情况, 小区/基站工参表 tbCell 中的小区天线高度不能小于 0。在 tbCell 的备份表 tbCellnew 上, 用 Alter table add check 添加约束, 并在该备份表上完成以下实验内容:

Step1. 查询小区/基站工参表的小区天线高度 (HEIGHT) 小于 20 的 SECTOR_ID、SECTOR_NAME 和 HEIGHT;

Step2. 更新小区/基站工参表将 step1 中的 HEIGHT 设置为当前值减去 15 (注意此时有可能违反 check 约束)

Step3. 查询小区/基站工参表的小区天线高度 (HEIGHT) 小于 20 的 SECTOR_ID、SECTOR_NAME 和 HEIGHT;

针对以上操作分别进行如下的操作:

(1) 将以上操作组织成普通的 SQL 语句, 顺序执行。124

(2) 将以上操作组织成事务执行 (以 begin tran 开始, 以 commit tran 结束)。

查看数据库, 观察两次的执行结果有何异同。

【添加 check 约束】:

```
alter table tbcellnew add CHECK (HEIGHT >=0);
```

100 % <



命令已成功完成。

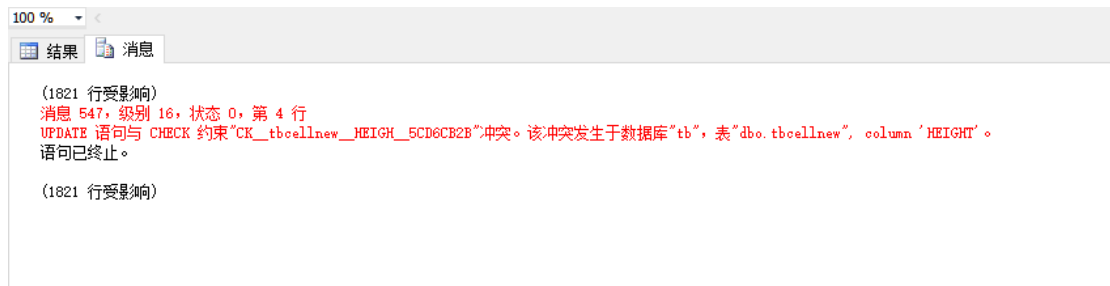
11.1.1.1 顺序执行

【SQL 语句】:

```
select SECTOR_ID, SECTOR_NAME, HEIGHT from tbcellnew
where HEIGHT < 20

update tbcellnew set HEIGHT = HEIGHT - 15
where HEIGHT < 20

select SECTOR_ID, SECTOR_NAME, HEIGHT from tbcellnew
where HEIGHT < 20
```



【说明】：update 操作由于违反 HEIGHT 不能小于 0 的 check 约束，更新失败

11.1.1.2 组织成事务执行

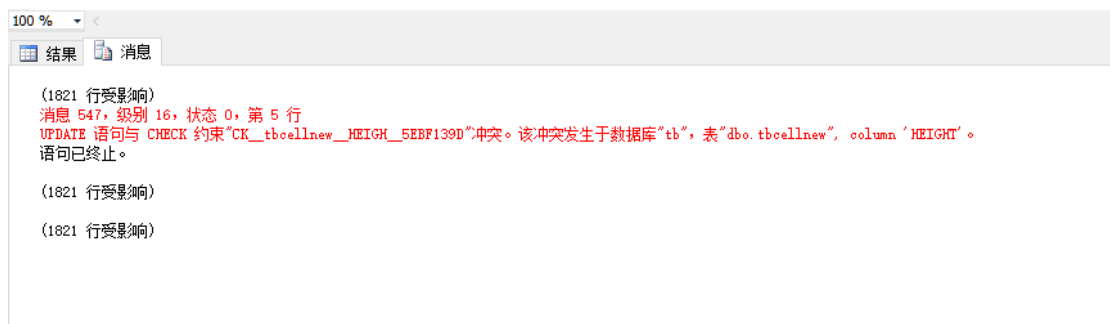
【SQL 语句】：

```
begin tran
    select SECTOR_ID, SECTOR_NAME, HEIGHT from tbcellnew
    where HEIGHT < 20

    update tbcellnew set HEIGHT = HEIGHT - 15
    where HEIGHT < 20

    select SECTOR_ID, SECTOR_NAME, HEIGHT from tbcellnew
    where HEIGHT < 20
commit tran

select SECTOR_ID, SECTOR_NAME, HEIGHT from tbcellnew
where HEIGHT < 20
```



【说明】：update 操作由于违反 HEIGHT 不能小于 0 的 check 约束，更新失败

11.1.2 数据表更新

11.1.2.1 显示执行模式

【实验要求】：

分别以三种事务执行方式，在 tbCell 的备份表 tbCellnew 上执行以下操作，并观察、分

析、解释执行结果

Step1. 查看小区 ID 在'122880-0'和'122882-2'之间的小区配置的频点编号;

Step2. 将小区 ID 在'122880-0'和'122882-2'之间的小区配置的频点编号更新为 37900;

Step3. 再次查看小区 ID 在'122880-0'和'122882-2'之间的小区配置的频点编号。

将 step1、step2 和 step3 的数据库访问组织成 1 个单一事务 T1，再将 step3 作为 1 个独立事务，提交 DBMS，串行执行这 2 个事务，观察 T1 中的 rollback、commit 对事务执行结果的影响。

由 step1、step2 和 step3 组成的事务 T1 采用以下 2 种结束方式：

(1) 以 commit 结束；

(2) 以 rollback 结束。

(1) 以 commit 结束

【SQL 语句】：

由于 tbccl 表中并没有数据的 SECTOR_ID 位于'122880-0'和'122882-2'之间，我们改为'15310-0'到'15380-0'之间

```
begin tran
begin try
    select SECTOR_ID, EARFCN from tbcclnew
    where SECTOR_ID between '15310-0' and '15380-0'
    update tbcclnew set EARFCN=37900
    where SECTOR_ID between '15310-0' and '15380-0'
    select SECTOR_ID, EARFCN from tbcclnew
    where SECTOR_ID between '15310-0' and '15380-0'
end try
begin catch
    select  Error_number() as ErrorNumber, --错误代码
           Error_severity() as ErrorSeverity, --错误严重级别，级别小于try catch 捕获
           Error_state() as ErrorState , --错误状态码
           Error_Procedure() as ErrorProcedure , --出现错误的存储过程或触发器的名称。
           Error_line() as ErrorLine, --发生错误的行号
           Error_message() as ErrorMessage --错误的具体信息
    if @@trancount>0) --全局变量@@trancount，事务开启此值+1，他用来判断是有开启事务
        rollback tran --由于出错，这里回滚到开始，第一条语句也没有插入成功。
end catch
if @@trancount>0)
    commit tran
select SECTOR_ID, EARFCN from tbcclnew
where SECTOR_ID between '15310-0' and '15380-0'
```

【结果】

100 %

结果 消息

| | SECTOR_ID | EARFCN |
|---|-----------|--------|
| 1 | 15310-128 | 38400 |
| 2 | 15310-129 | 38400 |
| 3 | 15310-130 | 38400 |
| 4 | 15311-128 | 38400 |
| 5 | 15311-129 | 38400 |
| 6 | 15311-130 | 38400 |
| 7 | 15312-128 | 38400 |

| | SECTOR_ID | EARFCN |
|---|-----------|--------|
| 1 | 15310-128 | 37900 |
| 2 | 15310-129 | 37900 |
| 3 | 15310-130 | 37900 |
| 4 | 15311-128 | 37900 |
| 5 | 15311-129 | 37900 |
| 6 | 15311-130 | 37900 |

| | SECTOR_ID | EARFCN |
|---|-----------|--------|
| 1 | 15310-128 | 37900 |
| 2 | 15310-129 | 37900 |
| 3 | 15310-130 | 37900 |
| 4 | 15311-128 | 37900 |
| 5 | 15311-129 | 37900 |

(2) 以 rollback 结束

【SQL 语句】

```

begin tran
    select SECTOR_ID,EARFCN from tbcclnew
    where SECTOR_ID between '15310-0' and '15380-0'
    update tbcclnew set EARFCN=37900
    where SECTOR_ID between '15310-0' and '15380-0'
    select SECTOR_ID,EARFCN from tbcclnew
    where SECTOR_ID between '15310-0' and '15380-0'

rollback tran
select SECTOR_ID,EARFCN from tbcclnew
where SECTOR_ID between '15310-0' and '15380-0'

```

【结果】

100 % <

结果 消息

| | SECTOR_ID | EARFCN |
|---|-----------|--------|
| 1 | 15310-128 | 38400 |
| 2 | 15310-129 | 38400 |
| 3 | 15310-130 | 38400 |
| 4 | 15311-128 | 38400 |
| 5 | 15311-129 | 38400 |
| 6 | 15311-130 | 38400 |

| | SECTOR_ID | EARFCN |
|---|-----------|--------|
| 1 | 15310-128 | 37900 |
| 2 | 15310-129 | 37900 |
| 3 | 15310-130 | 37900 |
| 4 | 15311-128 | 37900 |
| 5 | 15311-129 | 37900 |
| 6 | 15311-130 | 37900 |

| | SECTOR_ID | EARFCN |
|---|-----------|--------|
| 1 | 15310-128 | 38400 |
| 2 | 15310-129 | 38400 |
| 3 | 15310-130 | 38400 |
| 4 | 15311-128 | 38400 |

【分析】

只用 rollback 结束事务，在事务内部结果改变，但是事务结束后结果没有改变。
而执行 commit 后结果发生改变并提交至数据库，不能再进行回滚。

11.1.2.2 隐式事务

(1) 以 commit 结束

【SQL 语句】:

由于 tbccll 表中并没有数据的 SECTOR_ID 位于'122880-0'和'1228822-2'之间，我们改为'15310-0'到'15380-0'之间

```

Set IMPLICIT_TRANSACTIONS ON
select SECTOR_ID,EARFCN from tbccllnew
where SECTOR_ID between '15310-0' and '15380-0'
update tbccllnew set EARFCN=37900
where SECTOR_ID between '15310-0' and '15380-0'
select SECTOR_ID,EARFCN from tbccllnew
where SECTOR_ID between '15310-0' and '15380-0'
COMMIT TRANSACTION
Set IMPLICIT_TRANSACTIONS OFF
select SECTOR_ID,EARFCN from tbccllnew
where SECTOR_ID between '15310-0' and '15380-0'

```

【结果】

100 %

结果 消息

| | SECTOR_ID | EARFCN |
|----|-----------|--------|
| 6 | 15311-130 | 38400 |
| 7 | 15312-128 | 38400 |
| 8 | 15312-129 | 38400 |
| 9 | 15312-130 | 38400 |
| 10 | 15313-128 | 38400 |
| 11 | 15313-129 | 38400 |
| 12 | 15313-130 | 38400 |
| 13 | 15314-128 | 38400 |

| | SECTOR_ID | EARFCN |
|---|-----------|--------|
| 6 | 15311-130 | 37900 |
| 7 | 15312-128 | 37900 |
| 8 | 15312-129 | 37900 |
| 9 | 15312-130 | 37900 |

| | SECTOR_ID | EARFCN |
|---|-----------|--------|
| 1 | 15310-128 | 37900 |
| 2 | 15310-129 | 37900 |
| 3 | 15310-130 | 37900 |
| 4 | 15311-128 | 37900 |

(2) 以 rollback 结束

【SQL 语句】

```

Set IMPLICIT_TRANSACTIONS ON
select SECTOR_ID,EARFCN from tbcellnew
where SECTOR_ID between '15310-0' and '15380-0'
update tbCellnew set EARFCN=37900
where SECTOR_ID between '15310-0' and '15380-0'
select SECTOR_ID,EARFCN from tbcellnew
where SECTOR_ID between '15310-0' and '15380-0'
ROLLBACK TRANSACTION
Set IMPLICIT_TRANSACTIONS OFF
select SECTOR_ID,EARFCN from tbcellnew
where SECTOR_ID between '15310-0' and '15380-0'

```

【结果】

100 %

<

结果

消息

| | SECTOR_ID | EARFCN |
|---|-----------|--------|
| 1 | 15310-128 | 38400 |
| 2 | 15310-129 | 38400 |
| 3 | 15310-130 | 38400 |

| | SECTOR_ID | EARFCN |
|---|-----------|--------|
| 1 | 15310-128 | 37900 |
| 2 | 15310-129 | 37900 |
| 3 | 15310-130 | 37900 |
| 4 | 15311-128 | 37900 |
| 5 | 15311-129 | 37900 |

| | SECTOR_ID | EARFCN |
|---|-----------|--------|
| 1 | 15310-128 | 38400 |
| 2 | 15310-129 | 38400 |
| 3 | 15310-130 | 38400 |
| 4 | 15311-128 | 38400 |
| 5 | 15311-129 | 38400 |
| 6 | 15311-130 | 38400 |

【说明】

结果与显式提交事务一致。

11.1.2.3 自动提交模式

【SQL 语句】:

```
select SECTOR_ID,EARFCN from tbcclnew
where SECTOR_ID between '15310-0' and '15380-0'
update tbcclnew set EARFCN=37900
where SECTOR_ID between '15310-0' and '15380-0'
select SECTOR_ID,EARFCN from tbcclnew
where SECTOR_ID between '15310-0' and '15380-0'
```

【结果】

100 %

结果

消息

| | SECTOR_ID | EARFCN |
|---|-----------|--------|
| 1 | 15310-128 | 38400 |
| 2 | 15310-129 | 38400 |
| 3 | 15310-130 | 38400 |
| 4 | 15311-128 | 38400 |
| 5 | 15311-129 | 38400 |
| 6 | 15311-130 | 38400 |
| 7 | 15312-128 | 38400 |

| | SECTOR_ID | EARFCN |
|---|-----------|--------|
| 1 | 15310-128 | 37900 |
| 2 | 15310-129 | 37900 |
| 3 | 15310-130 | 37900 |
| 4 | 15311-128 | 37900 |
| 5 | 15311-129 | 37900 |
| 6 | 15311-130 | 37900 |
| 7 | 15312-128 | 37900 |
| 8 | 15312-129 | 37900 |

【分析】

自动提交模式下，每个 SQL 语句相当于 1 个事务，多条顺序提交的 SQL 语句对应于串行执行多个事务，未出错不会发生回滚。

11.1.3 数据库模式修改

【实验要求】

Step1. 修改 TD-LTE 数据库中的 tbOptCell 表的备份表 tbOptCellnew 的模式，删除列 CELL_TYPE（使用 alter table drop）。

Step2. 修改 tbOptCell 表的备份表 tbOptCellnew 的模式，增加列 CELL_TYPE（使用 alter table add）。

将 step1、step2 的数据库访问组织成 1 个单一事务（以显式事务的方式），采用以下 2 种结束方式：

- (1) 以 commit 结束；
- (2) 以 rollback 结束。

查看数据库（用 select 语句查看被删除/增加的列），观察数据库模式修改语句（alter table），是否会受到 rollback，commit 语句的影响。

也可以自行创建表、删除表，重复以上两步，查看数据库，观察数据库模式定义语句（create table）模式修改语句（drop table）是否会受到 rollback，commit 语句的影响。

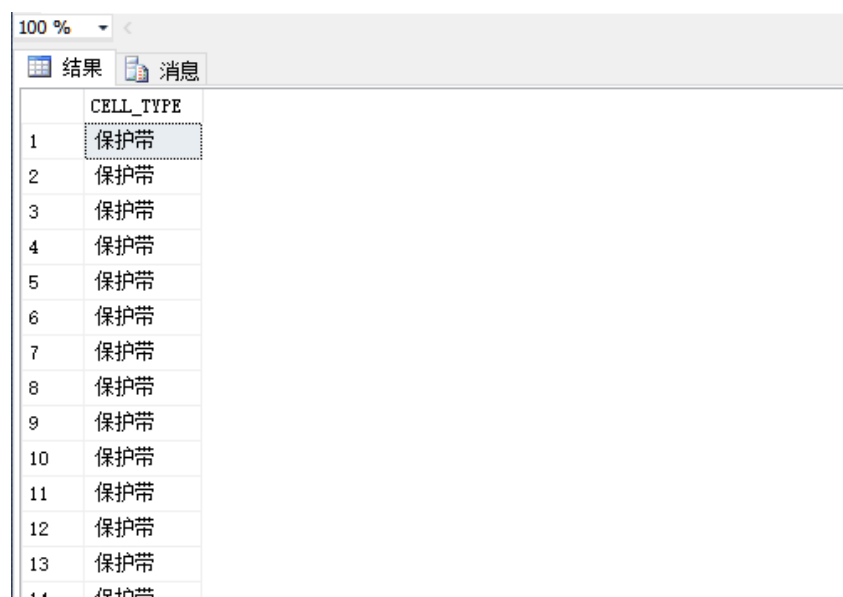
11.1.3.1 删除列 CELL_TYPE

11.1.3.1.1 只有 rollback 的过程：

【SQL 语句】

```
begin tran
    alter table tbOptCellnew
        drop column CELL_TYPE
rollback tran
select CELL_TYPE from tbOptCellnew
```

【结果】



| | CELL_TYPE |
|----|-----------|
| 1 | 保护带 |
| 2 | 保护带 |
| 3 | 保护带 |
| 4 | 保护带 |
| 5 | 保护带 |
| 6 | 保护带 |
| 7 | 保护带 |
| 8 | 保护带 |
| 9 | 保护带 |
| 10 | 保护带 |
| 11 | 保护带 |
| 12 | 保护带 |
| 13 | 保护带 |
| 14 | 保护带 |

【说明】

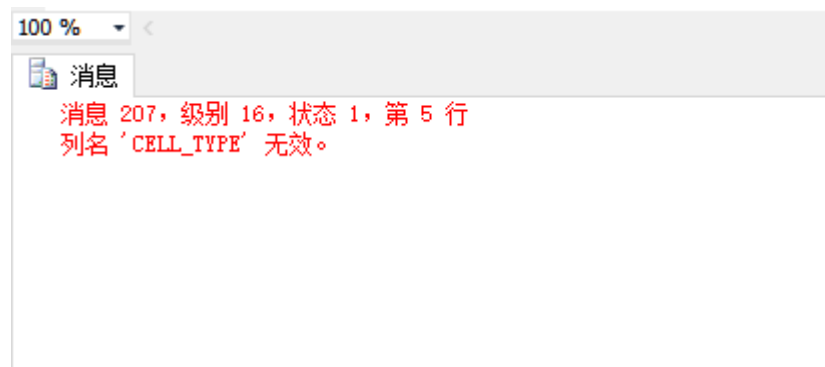
只执行 rollback 语句时，事务结束后没有真正提交结果，没有真正删除 CELL_TYPE。

11.3.1.1.2 执行 commit

【SQL 语句】

```
begin tran
    alter table tboptcellnew
    drop column CELL_TYPE
commit tran
select CELL_TYPE from tboptcellnew
```

【结果】



【说明】

执行 commit 语句后，成功删除 CELL_TYPE，事务结束后结果已提交，已不能再进行回滚。

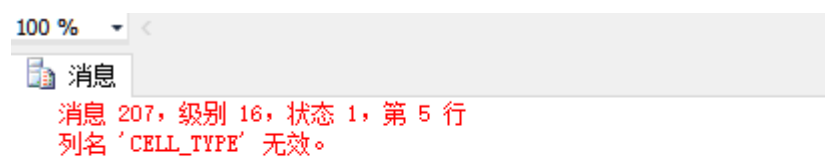
11.1.3.2 增加列 CELL_TYPE

11.1.3.2.1 只有 rollback 的过程：

【SQL 语句】

```
begin tran
    alter table tboptcellnew
    add CELL_TYPE varchar NULL
rollback tran
select CELL_TYPE from tboptcellnew
```

【结果】



【说明】

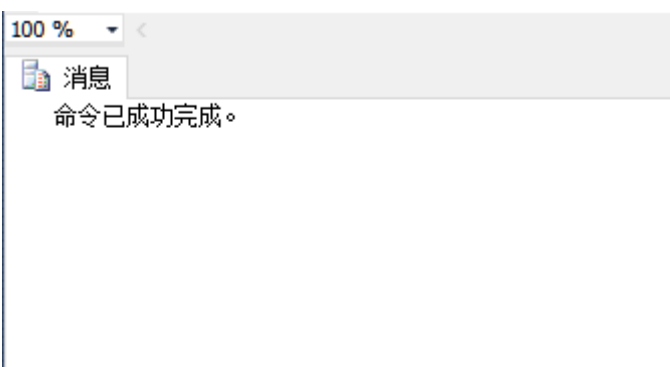
执行 rollback 语句后，事务结束后会回滚，查找 CELL_TYPE 显示无效，没有真正添加进去。

11.3.1.1.2 执行 commit

【SQL 语句】

```
begin tran
    alter table tboptcellnew
    add CELL_TYPE varchar NULL
commit tran
```

【结果】



【SQL 语句】

```
select CELL_TYPE from tboptcellnew
```

【结果】

| 结果 | |
|----|-----------|
| | CELL_TYPE |
| 1 | NULL |
| 2 | NULL |
| 3 | NULL |
| 4 | NULL |
| 5 | NULL |
| 6 | NULL |
| 7 | NULL |
| 8 | NULL |
| 9 | NULL |
| 10 | NULL |
| 11 | NULL |
| 12 | NULL |
| 13 | NULL |
| 14 | NULL |

【说明】

执行 commit 语句后，成功添加 CELL_TYPE，事务结束后结果已提交，已不能再进行回滚。因此能够查询到。

11.1.4 多条 insert/delete 操作执行比较

【实验要求】

- 以备份表 tbCellnew 为访问对象，
- Step1. 查询小区/基站工参表的小区天线高度（HEIGHT）小于 7 的 SECTOR_ID、

SECTOR_NAME 和 HEIGHT;

Step2. 在小区/基站工参表中, 添加一条 SECTOR_ID 为“211100-2”、HEIGHT 为 6 的信息;

Step3. 删除 step2 所添加的信息;

Step4. 查询小区/基站工参表的小区天线高度 (HEIGHT) 小于 7 的 SECTOR_ID、SECTOR_NAME 和 HEIGHT;

针对以上操作分别进行如下的操作:

- (1) 将以上操作组织成普通的 SQL 语句, 顺序执行。
- (2) 将以上操作组织成事务执行 (以 begin tran 开始, 以 commit tran 结束)。

查看数据库, 观察两次的执行结果有何异同。

11.1.4.1 顺序执行

【SQL 语句】

```
select SECTOR_ID,SECTOR_NAME,HEIGHT from tbcellnew
where HEIGHT < 7

insert into tbcellnew (SECTOR_ID,HEIGHT) values ('211100-2',6)
delete from tbcellnew where SECTOR_ID = '211100-2'

select SECTOR_ID,SECTOR_NAME,HEIGHT from tbcellnew
where HEIGHT < 7
```

【结果】

100 % <

结果 消息

| | | | |
|-----|----------|-----------------|---|
| 524 | 7389-144 | 德F苑小区三期一-HLWE-1 | 0 |
| 525 | 7390-144 | 德F苑小区三期二-HLWE-1 | 0 |
| 526 | 7391-144 | 海洋新天域-HLWE-1 | 0 |
| 527 | 7392-144 | D县交警队办公楼-HLWE-1 | 0 |
| 528 | 7393-144 | D县天鹅湾一-HLWE-1 | 0 |
| 529 | 7395-144 | 砥柱大厦-HLWE-1 | 0 |
| 530 | 7396-144 | 迎宾花园一期二-HLWE-1 | 0 |
| 531 | 7397-144 | 迎宾花园一期三-HLWE-1 | 0 |

| | SECTOR_ID | SECTOR_NAME | HEIGHT |
|-----|-----------|-----------------|--------|
| 524 | 7389-144 | 德F苑小区三期一-HLWE-1 | 0 |
| 525 | 7390-144 | 德F苑小区三期二-HLWE-1 | 0 |
| 526 | 7391-144 | 海洋新天域-HLWE-1 | 0 |
| 527 | 7392-144 | D县交警队办公楼-HLWE-1 | 0 |
| 528 | 7393-144 | D县天鹅湾一-HLWE-1 | 0 |
| 529 | 7395-144 | 砥柱大厦-HLWE-1 | 0 |
| 530 | 7396-144 | 迎宾花园一期二-HLWE-1 | 0 |
| 531 | 7397-144 | 迎宾花园一期三-HLWE-1 | 0 |

【说明】

先增加一行再删除同一行最后的结果不变, 两次查询的数目相同, 只是结果排列的顺序

可能不同。

11.1.4.2 组织成事务执行

【SQL 语句】

```
begin tran
    select SECTOR_ID, SECTOR_NAME, HEIGHT from tbcellnew
    where HEIGHT < 7

    insert into tbcellnew (SECTOR_ID, HEIGHT) values ('211100-2', 6)
    delete from tbcellnew where SECTOR_ID = '211100-2'

    select SECTOR_ID, SECTOR_NAME, HEIGHT from tbcellnew
    where HEIGHT < 7
commit tran
select SECTOR_ID, SECTOR_NAME, HEIGHT from tbcellnew
where HEIGHT < 7
```

【结果】

100 % <

| | | | |
|-------|----------|----------------|---|
| 结果 消息 | | | |
| 528 | 7393-144 | D县天鹅湾一-HLWE-1 | 0 |
| 529 | 7395-144 | 砥柱大厦-HLWE-1 | 0 |
| 530 | 7396-144 | 迎宾花园一期二-HLWE-1 | 0 |
| 531 | 7397-144 | 迎宾花园一期三-HLWE-1 | 0 |

| | | | |
|-----|-----------|-----------------|--------|
| | SECTOR_ID | SECTOR_NAME | HEIGHT |
| 524 | 7389-144 | 德P苑小区三期一-HLWE-1 | 0 |
| 525 | 7390-144 | 德P苑小区三期二-HLWE-1 | 0 |
| 526 | 7391-144 | 海洋新天域-HLWE-1 | 0 |
| 527 | 7392-144 | D县交警队办公楼-HLWE-1 | 0 |
| 528 | 7393-144 | D县天鹅湾一-HLWE-1 | 0 |

| | | | |
|-----|-----------|-----------------|--------|
| | SECTOR_ID | SECTOR_NAME | HEIGHT |
| 524 | 7389-144 | 德P苑小区三期一-HLWE-1 | 0 |
| 525 | 7390-144 | 德P苑小区三期二-HLWE-1 | 0 |
| 526 | 7391-144 | 海洋新天域-HLWE-1 | 0 |
| 527 | 7392-144 | D县交警队办公楼-HLWE-1 | 0 |
| 528 | 7393-144 | D县天鹅湾一-HLWE-1 | 0 |
| 529 | 7395-144 | 砥柱大厦-HLWE-1 | 0 |
| 530 | 7396-144 | 迎宾花园一期二-HLWE-1 | 0 |
| 531 | 7397-144 | 迎宾花园一期三-HLWE-1 | 0 |

【说明】

在事务提交后再添加一条 select 语句查看，先增加一行再删除同一行，表中的数据没有发生改变。

11.1.5 保存点 Savepoint 设置与回滚实验

本实验要求在事务内部不同执行位置设置，例如添加之后、添加之前、删除之后等，使用 SAVE TRANSACTION savepoint_name 语句创建保存点,使用 ROLLBACK TRANSATCTION

savepoint_name 语句将事务回滚，观察每次操作的结果。保存点提供了回滚部分事务的机制，而不是回滚到事务的开始。

【实验要求】

以小区 PCI 优化调整结果表 tbPCIAssignment 为访问对象，在创建的事务中 insert 插入语句后设置保存点，然后删除添加的信息，并回滚至保存点并提交事务；事务完成后再查询相应的行，观察执行结果是否插入成功，具体如下：

Step1. 查询小区 PCI 优化调整结果表的小区 PCI 为 106 的小区的 SECTOR_ID；

Step2. 在小区 PCI 优化调整结果表中，添加一条 SECTOR_ID 为“220102-5”、PCI 为 106 的信息；

Step3. 设置保存点；

Step4. 删除 step2 所添加的信息；

Step5. 回滚至保存点；

Step6. 事务提交结束；

Step7. 查询小区 PCI 优化调整结果表的小区 PCI 为 106 的小区的 SECTOR_ID；

事务结构如下：

begin tran

select 语句（检查表中原始数据）

insert 语句（向表中添加一行新的数据）

save transaction ppp（设置保存点）

delete 语句（删除 insert 语句添加的行）

rollback transaction ppp（回滚至保存点）

commit tran（提交事务）

select 语句（检查插入数据是否成功）

【SQL 语句】

```
begin tran
    select SECTOR_ID, SECTOR_NAME, PCI from tbpciassignment
    where PCI = 106

    insert into tbpciassignment (SECTOR_ID, PCI) values ('220102-5', 106)
save tran ppp
    delete from tbpciassignment where SECTOR_ID = '220102-5'
rollback tran ppp
commit tran
select SECTOR_ID, SECTOR_NAME, PCI from tbpciassignment
where PCI = 106
```

【结果】

100 % <

结果 消息

| | SECTOR_ID | SECTOR_NAME | PCI |
|---|-----------|-------------|-----|
| 1 | 47470-0 | B马南河-HLHF-1 | 106 |
| 2 | 220102-5 | NULL | 106 |

| | SECTOR_ID | SECTOR_NAME | PCI |
|---|-----------|-------------|-----|
| 1 | 47470-0 | B马南河-HLHF-1 | 106 |
| 2 | 220102-5 | NULL | 106 |
| 3 | 220102-5 | NULL | 106 |

【说明】

提交前 delete 语句被回滚，因此最后的结果中没有执行 delete 语句，PCI 为 106 的元组被插入数据库。

11.1.6 事务局部回滚/整体回滚

【实验要求】

根据现网实际情况，小区 PCI 优化调整结果表 tbPCIAssignment 中的小区 PCI 在 0 到 503 之间。在 tbPCIAssignment 的备份表 tbPCIAssignmentnew 上，用 Alter table add check 添加约束，并在该备份表上完成以下实验内容：

Step1. 在 tbPCIAssignmentnew 表上添加约束：加入约束 check(PCI between 0 and 503)。

Step2. 以备份表 tbPCIAssignmentnew 表为访问对象，依次添加 PCI 为‘300’和‘600’的两条数据，将 2 条对备份表 tbPCIAssignmentnew 表进行顺序访问的 insert 语句组织成 1 个显示执行模式下的事务；

Step3. 利用 SET XACT_ABORT ON/OFF 语句，控制事务执行过程中的进行整体回滚、局部回滚。观察并对比当事务执行违反约束时，事务结束后 tbPCIAssignmentnew 的内容，分析在整体回滚、局部回滚两种情况下，SQL Server 提供的事务原子性保障机制。

添加约束：

```
alter table tbpciassignmentnew add CHECK (PCI between 0 and 503);
```

【结果】

| | |
|----------|---|
| 100 % | < |
| 消息 | |
| 命令已成功完成。 | |

11.1.6.1 局部回滚

【SQL 语句】

```
begin tran
    insert into tbpciassignmentnew (PCI) values (300)
    insert into tbpciassignmentnew (PCI) values (600)
commit tran
```

【结果】

100 % <

消息

(1 行受影响)
 消息 547，级别 16，状态 0，第 3 行
 INSERT 语句与 CHECK 约束“CK_tbpciassign_PCI_6477ECF3”冲突。该冲突发生于数据库“tb”，表“dbo.tbpciassignmentnew”，column ‘PCI’。
 语句已终止。

检查是否插入：

```
select * from tbpciassignmentnew where PCI = 300;
```

【结果】

100 % <

结果 消息

| | ASSIGN_ID | EARFCN | SECTOR_ID | SECTOR_NAME | ENODEB_ID | PCI | FSS | SSS | LONGITUDE | LATITUDE | STYLE | OPT_DATETIME |
|---|-----------|--------|-----------|-------------|-----------|-----|------|------|-----------|----------|-------|--------------|
| 1 | NULL | NULL | NULL | NULL | NULL | 300 | NULL | NULL | NULL | NULL | NULL | NULL |

【说明】

不设置开关，执行两个插入，第一个插入遵守约束，插入成功，第二个违反约束插入失败。

11.1.6.2 整体回滚

【SQL 语句】

```
SET XACT_ABORT ON
begin tran
    insert into tbpciassignmentnew (PCI) values (300)
    insert into tbpciassignmentnew (PCI) values (600)
commit tran
```

【结果】

100 % <

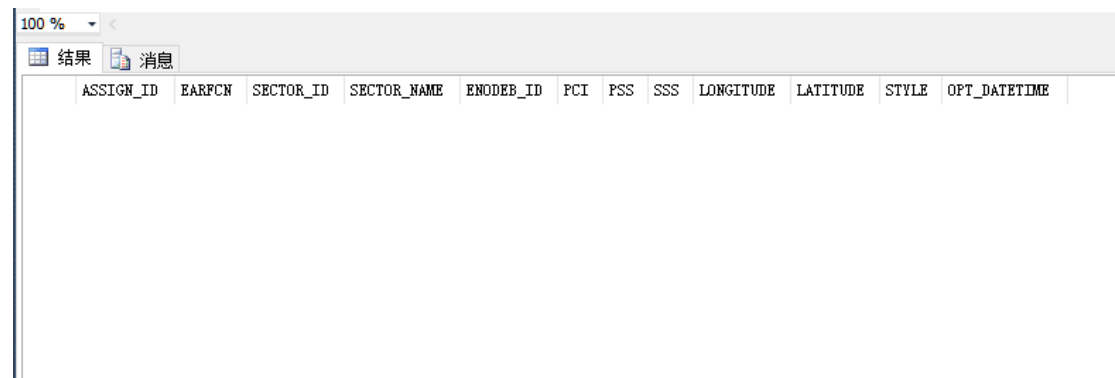
消息

(1 行受影响)
 消息 547，级别 16，状态 0，第 4 行
 INSERT 语句与 CHECK 约束“CK_tbpciassign_PCI_6477ECF3”冲突。该冲突发生于数据库“tb”，表“dbo.tbpciassignmentnew”，column ‘PCI’。

检查是否插入：

```
select * from tbpciassignmentnew where PCI = 300
```

【结果】



| ASSIGN_ID | EARFCN | SECTOR_ID | SECTOR_NAME | ENODEB_ID | PCI | PSS | SSS | LONGITUDE | LATITUDE | STYLE | OPT_DATETIME |
|-----------|--------|-----------|-------------|-----------|-----|-----|-----|-----------|----------|-------|--------------|
|-----------|--------|-----------|-------------|-----------|-----|-----|-----|-----------|----------|-------|--------------|

【说明】

本来表中没有 PCI=300 的这一行的, 不设置 XACT_ABORT, 事务出错时只会回滚更新, 所以插入会成功。设置 XACT_ABORT 为 ON, 事务出错会全部回滚, 所以插入也不成功。

11.2 事务并发控制机制

11.2.1 查看锁信息

【实验要求】：

在 SQL Server 的默认隔离级别 read committed 下, 从元数据表 sys.dm_tran_locks 中, 查看由 select/update/insert/delete 等 SQL 访问语句组成的事务的锁信息。

可自行选择 TD-LTE 数据库中的一张表 (tbCell/tbCellnew 除外) 作为事务的访问对象, 要求事务中至少包含两种不同类型 SQL 访问语句。

可用前面实验中编写的事务语句。并根据相应的参数作简要分析。

【SQL 语句】：

```
begin tran
    insert into tbpciassignmentnew (PCI) values (500)
    select PCI from tbpciassignmentnew where PCI = 500
    delete from tbpciassignmentnew where PCI = 500
    update tbpciassignmentnew set PCI = PCI -5
    where PCI = 180
    select
resource_type, resource_description, request_mode, request_status, request_type, request_lif
etime
        from sys.dm_tran_locks
commit tran
```

【结果】：

| | | | |
|-------|-----|-------|--|
| 100 % | | 结果 消息 | |
| | PCI | | |
| 1 | 500 | | |
| 2 | 500 | | |

| | resource_type | resource_description | request_mode | request_status | request_type | request_lifetime |
|----|---------------|----------------------|--------------|----------------|--------------|------------------|
| 1 | DATABASE | | S | GRANT | LOCK | 0 |
| 2 | DATABASE | | S | GRANT | LOCK | 0 |
| 3 | DATABASE | | S | GRANT | LOCK | 0 |
| 4 | DATABASE | | S | GRANT | LOCK | 0 |
| 5 | DATABASE | | S | GRANT | LOCK | 0 |
| 6 | DATABASE | | S | GRANT | LOCK | 0 |
| 7 | DATABASE | | S | GRANT | LOCK | 0 |
| 8 | PAGE | 1:1215 | IX | GRANT | LOCK | 33554432 |
| 9 | PAGE | 1:1211 | IX | GRANT | LOCK | 33554432 |
| 10 | RID | 1:1211:8 | X | GRANT | LOCK | 33554432 |
| 11 | OBJECT | | IX | GRANT | LOCK | 33554432 |
| 12 | RID | 1:1215:40 | X | GRANT | LOCK | 33554432 |
| 13 | RID | 1:1215:41 | X | GRANT | LOCK | 33554432 |

【分析】:

可以看到 DATABASE 施加了 S 共享锁, 页 PAGE 和表 OBJECT 施加了意向互斥锁, 表中的行 RID 施加了互斥锁 X。在 **resource_description** 部分, 不是所有的锁粒度都支持, 其中 PAGE 的取值表示: <file_id>:<page_in_file>, RID 的取值表示: <file_id>:<page_in_file>:<row_on_page>, 对应与此例中 9 行的锁是加在 file_id 为 1 的 page_id 为 1211 的意向互斥锁。

11.2.2 单事务隔离级别及加锁信息和执行结果观察

【实验要求】:

比较不同隔离级别下, 事务加锁的粒度、锁类型。

Step1. 以 LTE 数据库中某个关系表为对象, 设计 1 个由多条 SQL 访问语句(增、删、改、查)组成的显示执行事务;

Step2. 使用 DBCC USEROPTIONS 查看当前的隔离级别和锁信息; 然后设置不同的隔离级别, 查看对比在不同的隔离级别的并发副作用;

Step3. 分析对比不同隔离级别下, 事务的执行结果差异。

11.2.2.1 查看系统默认的当前隔离级别

【SQL】:

```
DBCC USEROPTIONS
```

【结果】:

| | Set Option | Value |
|----|-------------------------|----------------|
| 1 | textsize | 2147483647 |
| 2 | language | 简体中文 |
| 3 | dateformat | ymd |
| 4 | datefirst | 7 |
| 5 | lock_timeout | -1 |
| 6 | quoted_identifier | SET |
| 7 | arithabort | SET |
| 8 | ansi_null_dflt_on | SET |
| 9 | ansi_warnings | SET |
| 10 | ansi_padding | SET |
| 11 | ansi_nulls | SET |
| 12 | concat_null_yields_null | SET |
| 13 | isolation level | read committed |

【分析】:

最后一行表示隔离级别，为 read committed 即“读提交”

11.2.2.2 设置不同的隔离级别并查看并发副作用

11.2.2.2.1 未提交的相关性（脏读）

【实验内容】:

对 2 个并发事务，当第 2 个事务选择访问第 1 个事务正在更新的行时，会发生未提交的相关性问题。第 2 个事务正在读取的数据还没有确认提交，并且之后可能由正在更新此行的第 1 个事务所更改。

脏读：读取其他事务未提交的数据。

【实验要求】:

- (1) 开启两个连接，模拟两个并行的事务。简称会话一和会话二；
- (2) 在会话一中将隔离级别设置为未提交读 read uncommitted，以 tbCellnew 为访问对象，查询 SECTOR_ID 为‘124686-2’的小区的 SECTOR_NAME 和 PCI；
- (3) 在会话二中开始事务，更新 SECTOR_ID 为‘124686-2’的小区的 PCI 为当前 PCI 值加 20，保持未提交状态执行语句；
- (4) 在会话一中将隔离级别设置为未提交读，以 tbCellnew 为访问对象，查询 SECTOR_ID 为‘124686-2’的小区的 SECTOR_NAME 和 PCI；
- (5) 消除并发副作用。

【实验过程】:

Step1: 先使用下面语句将数据库系统隔离级别修改为 read-uncommitted

```
1 SET global transaction isolation level read uncommitted;
```

Step2. 开启事务一：执行查询 sector_id=999 的数据行，没有数据返回

```
1. start transaction;
2. select SECTOR_ID, HEIGHT from tbcell where SECTOR_ID='999';
```

```
1 start transaction;
2 • select SECTOR_ID, HEIGHT from tbcell where SECTOR_ID='999';
```

Result Grid

| SECTOR_ID | HEIGHT |
|-----------|--------|
|-----------|--------|

tbcell 16 x

Output

Action Output

| # | Time | Action | Message |
|-----|----------|---|---|
| ✓ 1 | 20:00:27 | SET global transaction isolation level read uncommitted | 0 row(s) affected |
| ✓ 2 | 20:05:40 | start transaction | 0 row(s) affected |
| ✗ 3 | 20:05:40 | select SECTOR_ID, HEIGHT from tbcell where SEC... | Error Code: 1054. Unknown column 'SECTOR_ID, .. |
| ✓ 4 | 20:06:11 | commit | 0 row(s) affected |
| ✓ 5 | 20:06:16 | start transaction | 0 row(s) affected |
| ✓ 6 | 20:06:16 | select SECTOR_ID, HEIGHT from tbcell where SEC... | 0 row(s) returned |

Step3. 开启事务二，执行插入 sector_id=999 的操作，此时并未提交事务。

1. start **transaction**;
2. insert **into** tbcell (SECTOR_ID, HEIGHT) value ('999', 999);

Limit to 50000 rows

```

1 start transaction;
2 insert into tbccl (SECTOR_ID, HEIGHT) value ('999', 999);

```

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|-----|----------|--|---|-----------------------|
| ✓ 2 | 20:05:40 | start transaction | 0 row(s) affected | 0.016 sec |
| ✗ 3 | 20:05:40 | select SECTOR_ID, HEIGHT from tbccl where S... | Error Code: 1054. Unknown column 'SECTOR_ID... | 0.063 sec |
| ✓ 4 | 20:06:11 | commit | 0 row(s) affected | 0.063 sec |
| ✓ 5 | 20:06:16 | start transaction | 0 row(s) affected | 0.016 sec |
| ✓ 6 | 20:06:16 | select SECTOR_ID, HEIGHT from tbccl where SE... | 0 row(s) returned | 0.015 sec / 0.000 sec |
| ✓ 7 | 20:09:08 | start transaction | 0 row(s) affected | 0.016 sec |
| ⚠ 8 | 20:09:08 | insert into tbccl (SECTOR_ID, HEIGHT) value (99... | 1 row(s) affected, 7 warning(s): 1364 Field 'SECTO... | 0.015 sec |

Step4. 在事务一中再次执行查询，发现有了数据，返回查看正是事务二中插入的数据。

```

1. select SECTOR_ID, HEIGHT from tbccl where SECTOR_ID='999';

```

1 • `select SECTOR_ID, HEIGHT from tbcell where SECTOR_ID='999';`

The screenshot shows a database management tool interface. At the top, a query is entered: `select SECTOR_ID, HEIGHT from tbcell where SECTOR_ID='999';`. Below the query, the 'Result Grid' displays two columns: 'SECTOR_ID' and 'HEIGHT'. The first row contains the values '999' and '999'. The second row contains 'NULL' and 'NULL'. Below the result grid, the 'Output' pane shows a list of actions performed. The actions include a commit, a start transaction, a select query (which returned 0 rows), another start transaction, and an insert statement (which affected 1 row with 7 warnings). The final action is a select query (which returned 1 row).

| # | Time | Action | Message | Duration / Fetch |
|---|----------|--|---|-----------------------|
| 3 | 20:05:40 | select SECTOR_ID, HEIGHT from tbcell where S... | Error Code: 1054. Unknown column 'SECTOR_ID... | 0.063 sec |
| 4 | 20:06:11 | commit | 0 row(s) affected | 0.063 sec |
| 5 | 20:06:16 | start transaction | 0 row(s) affected | 0.016 sec |
| 6 | 20:06:16 | select SECTOR_ID, HEIGHT from tbcell where SE... | 0 row(s) returned | 0.015 sec / 0.000 sec |
| 7 | 20:09:08 | start transaction | 0 row(s) affected | 0.016 sec |
| 8 | 20:09:08 | insert into tbcell (SECTOR_ID, HEIGHT) value ('99... | 1 row(s) affected, 7 warning(s): 1364 Field 'SECTO... | 0.015 sec |
| 9 | 20:10:24 | select SECTOR_ID, HEIGHT from tbcell where SE... | 1 row(s) returned | 0.015 sec / 0.000 sec |

Step5. 返回事务二窗口，将事务回滚。

1 `rollback;`

Step6.在事务一中再次执行查询，发现没有数据返回，此前查询到的数据就是脏数据。

1. `select SECTOR_ID, HEIGHT from tbcell where SECTOR_ID='999';`

The screenshot shows the same database management tool interface. The query entered is: `select SECTOR_ID, HEIGHT from tbcell where SECTOR_ID='999';`. The 'Result Grid' displays two columns: 'SECTOR_ID' and 'HEIGHT'. The first row contains the values 'NULL' and 'NULL'. The second row contains 'NULL' and 'NULL'.

| SECTOR_ID | HEIGHT |
|-----------|--------|
| NULL | NULL |
| NULL | NULL |

11.2.2.2.2 不一致的分析（不可重复读）

【实验内容】：

当第 2 个事务多次访问表中同一行数据而且每次读取不同内容时，会发生不一致的分析问题。不一致的分析与未提交的相关性类似，因为其它事务也是正在更改第二个事务正在读取的数据。然而，在不一致的分析中，第二个事务读取的数据是由已进行了更改的事务提交的。而且，不一致的分析涉及多次（两次或更多）读取同一行，而且每次信息都由其它事务更改；因而该行被非重复读取。

【实验要求】：

- (1) 开启两个连接，模拟两个并行的事务。简称会话一和会话二；
- (2) 在会话一中将隔离级别设置为提交读或者未提交读，以 tbCellnew 为访问对象，查询 SECTOR_ID 为'124694-0'的小区的 SECTOR_NAME 和 PCI；设置推迟等待 5 秒语句，然后再写一条查询语句，同样是查询 SECTOR_ID 为'124694-0'的小区的 SECTOR_NAME 和 PCI，执行语句后的五秒内开始第三步；
- (3) 第二步语句执行的五秒内在会话二中执行语句，开始事务，更新 SECTOR_ID 为'124694-0'的小区的 PCI 为 400，并提交事务；148
- (4) 在会话一中查看执行结果；
- (5) 消除并发副作用。

【实验过程】：

Step1. 先使用下面语句将数据库系统隔离级别修改为 read-committed:

```
SET global transaction isolation level read committed;
```

```
SET global transaction isolation level read committed;
```

Step2. 先显式开启一个事务，并执行查询，此时为原始数据：

```
1. start transaction;  
2. select PCI from tbcell where SECTOR_ID='124672-1'
```

The screenshot shows a database client interface. At the top, there are two SQL statements entered in a text area:

```
1 start transaction;  
2 select PCI from tbcell where SECTOR_ID='124672-1'
```

Below the text area, there is a 'Result Grid' tab. The 'Result Grid' is active, showing a single row with the column 'PCI' and the value '30'. The row is highlighted with a red border.

At the bottom, there is an 'Output' tab. The 'Action Output' is selected, showing a log of actions:

| # | Time | Action | Message | Duration / Fetch |
|---|----------|--|-------------------|-----------------------|
| 1 | 20:20:07 | start transaction | 0 row(s) affected | 0.016 sec |
| 2 | 20:20:07 | select PCI from tbcell where SECTOR_ID='124672-1'... | 1 row(s) returned | 0.016 sec / 0.000 sec |

Step3. 在另一个窗口使用隐式事务（每条语句都看作是自动提交的事务），执行更新操作：

```
1      update tbcell set PCI=9999 where SECTOR_ID='124672-1'
```

```
update tbcell set PCI=9999 where SECTOR_ID='124672-1'
```

Step4. 返回显式事务中再次执行查询：

```
1. select PCI from tbcell where SECTOR_ID='124672-1'
```

查询到了刚更新的数据

```
1 • select PCI from tbcell where SECTOR_ID='124672-1'
```



The screenshot shows a database interface with a 'Result Grid' tab. Below the tab, there is a table with two columns: 'PCI' and '9999'. The 'PCI' column is highlighted with a red box.

| PCI |
|------|
| 9999 |

Step5 在隐式事务下再次执行更新操作，将 PCI 更新为 9998：

```
1      update tbcell set PCI=9998 where SECTOR_ID='124672-1'
```

Step6. 返回事务中查询，返回的是 9998，在这次事务中，执行了三次相同的查询，但是出现了三个不同的结果， 出现不可重复读。

```
1. select PCI from tbcell where SECTOR_ID='124672-1'
```

```
1 • select PCI from tbcell where SECTOR_ID='124672-1'
```



The screenshot shows a database interface with a 'Result Grid' tab. Below the tab, there is a table with two columns: 'PCI' and '9998'. The 'PCI' column is highlighted with a red box.

| PCI |
|------|
| 9998 |

| | | | | |
|---|---|----------|---|--|
| ✓ | 1 | 20:20:07 | start transaction | 0 row(s) affected |
| ✓ | 2 | 20:20:07 | select PCI from tbccl where SECTOR_ID='124672-1'... | 1 row(s) returned |
| ✓ | 3 | 20:30:02 | update tbccl set PCI=9999 where SECTOR_ID='124... | 1 row(s) affected Rows matched: 1 Changed: 1 War.. |
| ✓ | 4 | 20:31:39 | select PCI from tbccl where SECTOR_ID='124672-1'... | 1 row(s) returned |
| ✓ | 5 | 20:33:12 | update tbccl set PCI=9998 where SECTOR_ID='124... | 1 row(s) affected Rows matched: 1 Changed: 1 War.. |
| ✓ | 6 | 20:34:42 | select PCI from tbccl where SECTOR_ID='124672-1'... | 1 row(s) returned |
| ✓ | 7 | 20:35:52 | commit | 0 row(s) affected |

11.2.2.2.3 幻读

【实验内容】:

当事务 1 对表中某行执行插入或删除操作时，如果该行属于某个事务 2 正在读取的行的范围内，会发生幻读问题。事务 2 第一次读的行范围显示出其中一行已不复存 在于第二次读或后续读中，因为该行已被其它事务 1 删除。同样，由于其它事务 1 的插入操作，事务 2 的第二次或后续读显示有一行已不存在于原始读中。

【实验要求】:

- (1) 开启两个连接，模拟两个并行的事务。简称会话一和会话二；
- (2) 在会话一中将隔离级别设置为 READ UNCOMMITTED、READ COMMITTED 或 REPEATABLE READ，以 tbCellnew 为访问对象，查询 LONGITUDE 小于 110.37644 的小区的 LONGITUDE 并去重；设置推迟等待 5 秒语句，然后再写一条查询语句，同样是查询 LONGITUDE 小于 110.37644 的小区的 LONGITUDE 并去重；
- (3) 第二步语句执行的五秒内在会话二中执行语句，开始事务，添加 LONGITUDE 为 110.366666 的信息，并提交事务；150
- (4) 查看会话一执行结果；
- (5) 消除并发副作用。

【实验过程】:

Step1. 先使用下面语句将数据库系统隔离级别修改为 Repeatable-read:

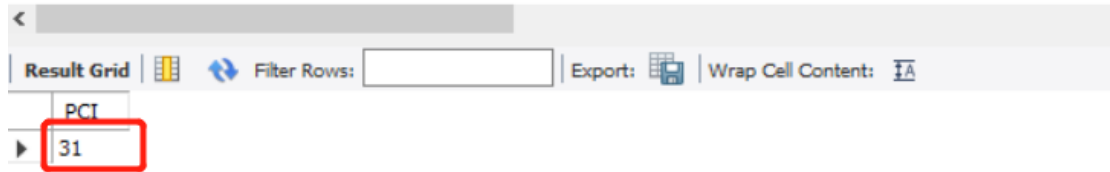
```
1. SET global transaction isolation level repeatable read;
```

```
SET global transaction isolation level repeatable read;
```

Step2. 开启一个事务，并执行查询语句：

```
1. start transaction;
2. select PCI from tbccl where SECTOR_ID between '124672-2' and '124672-3'
```

```
1 start transaction;
2 • select PCI from tbccl where SECTOR_ID between '124672-2' and '124672-3'
```



The screenshot shows a database interface with a 'Result Grid' tab. The grid has two columns: 'PCI' and an unnamed column. The first row contains the value '31' in the unnamed column. A red rectangle highlights the '31' value.

| PCI | |
|-----|----|
| | 31 |

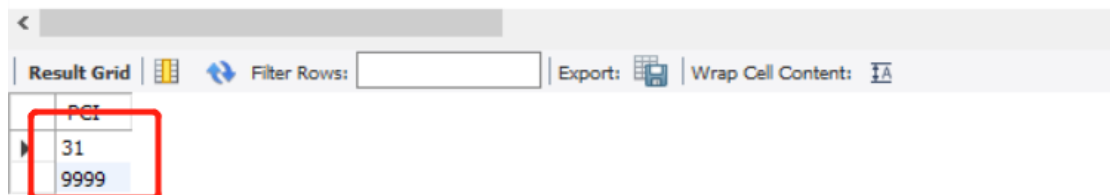
Step3. 在另一个窗口执行隐式事务，执行下面这条插入语句：

```
1. insert into tbccl(SECTOR_ID, PCI) values ('124672-3', 9999)
```

```
1 insert into tbccl(SECTOR_ID, PCI) values ('124672-3', 9999)
```

Step4. 返回事务窗口中再次执行查询，发现返回集中多了一条数据，出现幻读：

```
1 start transaction;
2 • select PCI from tbccl where SECTOR_ID between '124672-2' and '124672-3'
```



The screenshot shows the same database interface as before, but now the result grid has two rows. The first row has '31' in the unnamed column, and the second row has '9999' in the unnamed column. A red rectangle highlights both rows.

| PCI | |
|-----|------|
| | 31 |
| | 9999 |

小结：

幻读和不可重复读表现形式上看似都是多次查询但是结果不同，但两者本质上是不同的。不可重复读关注点在 update 操作上，而幻读关注点则是 insert 和 delete 操作。简单来说，幻读前后数据量不同，不可重复读是同一行数据前后读取到的内容不同。

如果使用锁机制实现这两种隔离级别，在可重复读中，该 SQL 语句第一次读取到数据后，就将这些数据加锁，其它事务无法修改这些数据，从而实现可重复读。但这种方法却无法锁住 insert 的数据，所以当事务 A 先前读取了数据，或者修改了全部数据，事务 B 还是可以 insert 数据提交，这时事务 A 就会发现莫名其妙多了一条之前没有的数据，这就是幻读，无法通过行锁来避免。

为避免幻读，需要采用 Serializable 隔离级别，读用读锁，写用写锁，读锁和写锁互斥，可以有效避免幻读、不可重复读、脏读等问题，但会极大的降低数据库的并发能力。

Mysql 的最高隔离级别是 serializable 此时事务对某个表操作时会把表加上表锁，其他事务要想也操作这个表那就只能等带锁的事务完成推出后才能执行，此时失去了并发能力。

【消除副作用】:

设置隔离级别为 Serializable，检测是否出现脏读，不可重复读，幻读等数据不一致。

实验过程:

1. `SET global transaction isolation level serializable;`

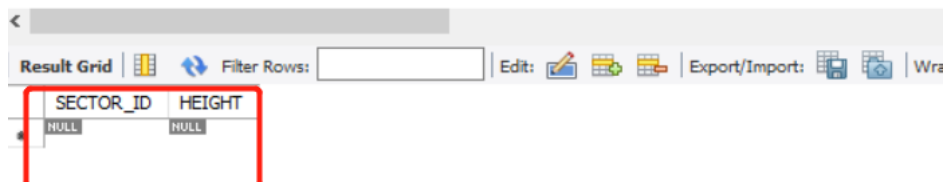
```
1 • SET global transaction isolation level serializable;
```

脏读:

Step1. 开启事务一：执行查询 sector_id=999 的数据行，没有数据返回

1. `start transaction;`
2. `select SECTOR_ID, HEIGHT from tbcell where SECTOR_ID='999';`

```
1 start transaction;
2 • select SECTOR_ID, HEIGHT from tbcell where SECTOR_ID='999';
3
```



| SECTOR_ID | HEIGHT |
|-----------|--------|
| NULL | NULL |

Step2. 开启事务二，执行插入 sector_id=999 的操作，此时并未提交事务。

1. `start transaction;`
2. `select into tbcell (SECTOR_ID, HEIGHT) value ('999', 999);`

```

1 start transaction;
2 • insert into tbcell (SECTOR_ID, HEIGHT) value ('999', 999);
3

```

| Output | | | | |
|---------------|----------|--|--|------------------|
| Action Output | | | | |
| # | Time | Action | Message | Duration / Fetch |
| 1 | 21:05:39 | start transaction | 0 row(s) affected | 0.016 sec |
| 2 | 21:05:39 | insert into tbcell (SECTOR_ID, HEIGHT) value ('999', ... | Error Code: 1062. Duplicate entry "for key 'UQ_SECT... | 0.015 sec |

可以看到，当隔离级别为 Serializable，当有一个事务查询后，若有其他事务通过插入的方式改变查询结果是不可能的，事务插入会被阻断。可见在该隔离级别下，通过限制可能造成脏读的插入，来保证不会发生脏读。

不可重复读：

Step1. 先显式开启一个事务，并执行查询，此时为原始数据：

```

1. start transaction;
2. select PCI from tbcell where SECTOR_ID='124672-1';

```

```

1 • start transaction;
2 • select PCI from tbcell where SECTOR_ID='124672-1';
3

```

| Result Grid | |
|-------------|------|
| PCI | 9998 |

Step2. 在另一个窗口使用隐式事务（每条语句都看作是自动提交的事务），执行更新操作：

```

1 • update tbcell set PCI=30 where SECTOR_ID='124672-1'

```

发现不能更新。

由于在上一部查询中，已经对于该行加锁，更新操作只能等待。这样不会造成在一个事务内的多次查询造成查询结果不一致的错误，也就是不可重复读错误。

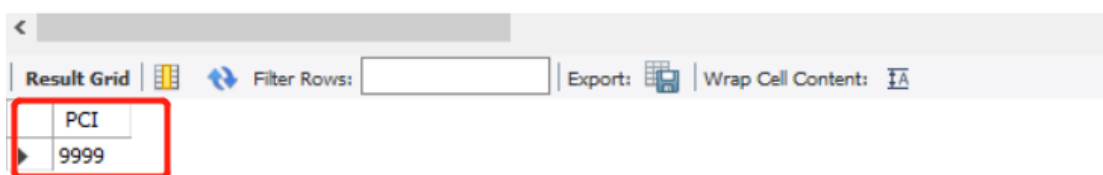
幻读：

Step1. 开启一个事务，并执行查询语句：

```
1. start transaction;
2. select PCI from tbccl where SECTOR_ID between '124672-3' and '124672-4'
```



```
1 • start transaction;
2 • select PCI from tbccl where SECTOR_ID between '124672-3' and '124672-4'
3
```

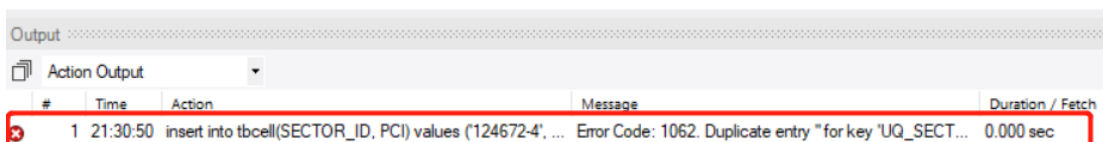


Step2. 在另一个窗口执行隐式事务，执行下面这条插入语句：

```
1. insert into tbccl(SECTOR_ID, PCI) values ('124672-4', 8888)
```



```
1 • insert into tbccl(SECTOR_ID, PCI) values ('124672-4', 8888)
```



可以看到，插入失败。

插入失败，也就保证了事务的两次查询之间不会造成查询结果数量的差异，保证了不会出现幻读。

总结：

| 隔离级别 | 可能出现的不一致现象 |
|------------------|-------------|
| Read-uncommitted | 脏读，不可重复读，幻读 |
| Read-committed | 不可重复读，幻读 |
| Repeatable-read | 幻读 |
| Serializable | - |

11.3 事务日志观察

11.3.1 单条 SQL 语句(select/update/insert/delete)在自动提交模式下的日志观察

察

【实验要求】：

以备份表 tbCellnew 为访问对象，

Step1. 查询小区/基站工参表的 SECTOR_ID 为'124674-1'的小区基站类型；

Step2. 修改小区/基站工参表的 SECTOR_ID 为'124674-1'的小区基站类型为'室分'；

Step3. 在小区/基站工参表中，添加一条 SECTOR_ID 为"211100-2"、HEIGHT 为 6 的信息；

Step4. 删除 step3 所添加的信息；

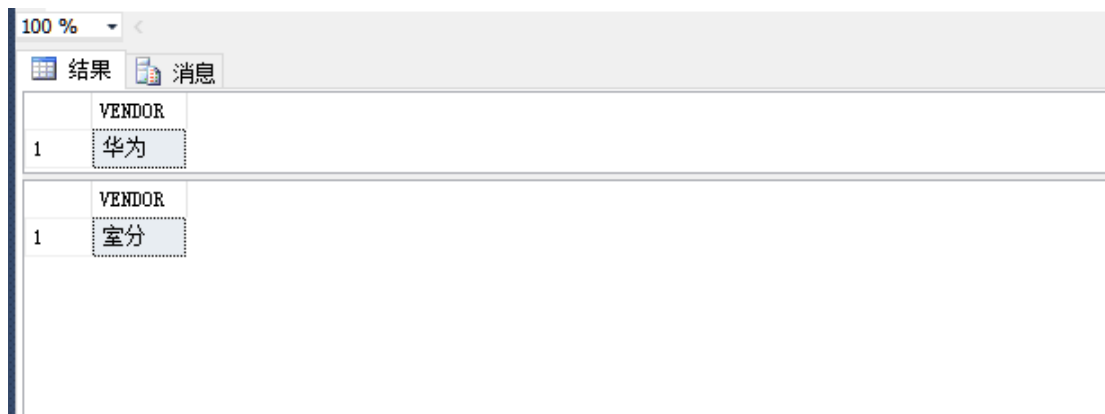
Step5. 查看事务日志，观察 operation 中具体的操作内容、事务日志记录中 update 语句修改前和修改后的数据值，并找出一对 checkpoint begin 和 checkpoint end 的值；观察各个属性，理解每个属性的含义；

【SQL 语句】：

```
select VENDOR from tbcellnew where SECTOR_ID = '124674-1'
update tbcellnew set VENDOR = '室分' where SECTOR_ID = '124674-1'
select VENDOR from tbcellnew where SECTOR_ID = '124674-1'
insert into tbcellnew (SECTOR_ID,HEIGHT) values ('211100-2',6)
delete from tbcellnew where SECTOR_ID = '211100-2'

SELECT * FROM [sys].[fn_dblog](NULL,NULL)
```

【结果】：



| | VENDOR |
|---|--------|
| 1 | 华为 |

| | VENDOR |
|---|--------|
| 1 | 室分 |

【分析】：

(1) LOP_BEGIN_XACT 表示一个事务的开始，对应于,LOP_COMMIT_XACT 表示事务的提交，对应于；以 update 为例，从第 2198 行开始，第 2200 行结束

| | | | | | | |
|------|------------------------|-----------------|----------|---------------|---|--------|
| 2198 | 000000d3:0000016e:0001 | LOP_BEGIN_XACT | LCX_NULL | 0000:00002b71 | 0 | 0x0000 |
| 2199 | 000000d3:0000016e:0002 | LOP_MODIFY_ROW | LCX_HEAP | 0000:00002b71 | 0 | 0x0000 |
| 2200 | 000000d3:0000016e:0003 | LOP_COMMIT_XACT | LCX_NULL | 0000:00002b71 | 0 | 0x0000 |

(2) 与 Operaiton 对应的 Transaction Name，比如 LOP_MODIFY_ROW 对应于 UPDATE；

| | | | | | | | |
|------|---|------|------------|------|-------------------------|--------|---|
| 2196 | | NULL | NULL | NULL | NULL | NULL | NULL |
| 2197 | | NULL | NULL | NULL | NULL | NULL | NULL |
| 2198 | 0 | 60 | 0x01000000 | 1 | 2021/06/12 18:10:17:963 | UPDATE | 0x010500000000000005150000000153F8A7F8AAD6A0E1A4D1... |
| 2199 | | NULL | NULL | NULL | NULL | NULL | NULL |
| 2200 | | NULL | NULL | NULL | NULL | NULL | NULL |

(3) 修改前数据是[RowLog Contents 0]字段内容，如果是 UPDATE 操作，修改后数据存放在 [RowLog Contents 1]字段内单条语句下，如下图。UPDATE 操作的事务日志是一种 类型的。

| int | RowLog Contents 0 | RowLog Contents 1 | RowLc |
|-----|---|--|-------|
| | 0x10001B0078A0987D030000000100000001... | 0x | 0x010 |
| | 0x1678A0987D020000000100000003000000... | 0x | 0x010 |
| | 0x20 | 0x64 | 0x020 |
| | 0x000000000000 | 0xAC0000000100 | 0x |
| | NULL | NULL | NULL |
| | NULL | NULL | NULL |
| | 0x08000E0000002348000000000300 | 0x | 0x010 |
| | 0x | 0x07000000A9752B0145AD0000821500000... | 0x |
| | 0x20 | 0x60 | 0x020 |
| | 0x000000000000 | 0xAD0000000100 | 0x |
| | NULL | NULL | NULL |
| | NULL | NULL | NULL |
| | 0x08000E0000002348000000000300 | 0x | 0x010 |
| | NULL | NULL | NULL |
| | 0x | 0x390035002D0032003700300033002D003... | 0x |
| | NULL | NULL | NULL |
| | 0x20010022007F010000000000 | 0x000200240048807F01821500000000000... | 0x160 |
| | NULL | NULL | NULL |
| | NULL | NULL | NULL |
| | 0x4E533A4E8F5BD97A00000000000000059 | 0xA45B06528F5BD97A800100000100000071 | 0x |
| | NULL | NULL | NULL |

读取字段数据：

```
select cast(0x4E533A4E8F5BD97A00000000000000059 as  
nvarchar(255)), cast(0xA45B06528F5BD97A800100000100000071 as nvarchar(255))
```

100 %

结果 消息

| | (无列名) | (无列名) |
|---|-------|-------|
| 1 | 华为宏站 | 室分宏站 |

11.3.2 显示执行模式下，多条语句组成的单个事务的日志的观察

11.3.2.1 以 commit 结束的事务日志实验

【实验要求】:

以备份表 tbCellnew 为访问对象,

Step1. 查询小区/基站工参表的小区天线高度 (HEIGHT) 小于 7 的 SECTOR_ID;

Step2. 修改小区/基站工参表的 SECTOR_ID 为'124678-2'的小区的高度为'10';

Step3. 在小区/基站工参表中, 添加一条 SECTOR_ID 为"211100-2"、HEIGHT 为 6 的信息;

Step4. 删除 step3 所添加的信息;

Step5. 将以上操作组织成事务执行 (以 begin tran 开始, 以 commit tran 结束)

Step6. 查看事务日志, 观察 operation 中具体的操作内容, 并观察各个属性, 理解每个属性的含义;

【SQL 语句】:

```
begin tran
    select SECTOR_ID from tbcellnew where HEIGHT < 7
    update tbcellnew set HEIGHT = 10 where SECTOR_ID = '1246778-2'
    insert into tbcellnew (SECTOR_ID, SECTOR_NAME, HEIGHT) values ('211100-2', '123', 6)
    delete from tbcellnew where SECTOR_ID = '211100-2'
commit tran
SELECT * FROM [sys].[fn_dblog] (NULL, NULL)
```

【结果】:

| | Current LSN | Operation | Context | Transaction ID | LogI |
|------|------------------------|--------------------|----------|----------------|------|
| 2198 | 000000d3:0000016c:0001 | LOP_BEGIN_XACT | LCX_NULL | 0000:00002b71 | 0 |
| 2199 | 000000d3:0000016c:0002 | LOP_MODIFY_ROW | LCX_HEAP | 0000:00002b71 | 0 |
| 2200 | 000000d3:0000016c:0003 | LOP_COMMIT_XACT | LCX_NULL | 0000:00002b71 | 0 |
| 2201 | 000000d3:0000016d:0001 | LOP_BEGIN_XACT | LCX_NULL | 0000:00002b72 | 0 |
| 2202 | 000000d3:0000016d:0002 | LOP_LOCK_XACT | LCX_NULL | 0000:00002b72 | 0 |
| 2203 | 000000d3:0000016d:0003 | LOP_MODIFY_COLUMNS | LCX_C... | 0000:00002b72 | 0 |
| 2204 | 000000d3:0000016d:0004 | LOP_COMMIT_XACT | LCX_NULL | 0000:00002b72 | 0 |

【分析】:

此事务从第 2198 行开始, 2200 行结束。

11.3.2.2 以 rollback 结束的事务日志实验

【实验要求】:

以备份表 tbCellnew 为访问对象,

Step1. 查询小区/基站工参表的小区天线高度 (HEIGHT) 小于 7 的 SECTOR_ID;

Step2. 修改小区/基站工参表的 SECTOR_ID 为'124678-2'的小区的高度为'10';

Step3. 在小区/基站工参表中, 添加一条 SECTOR_ID 为"211100-2"、HEIGHT 为 6 的信息;

Step4. 删除 step3 所添加的信息;

Step5. 将以上操作组织成事务执行 (以 begin tran 开始, 以 rollback tran 结束)

Step6. 查看事务日志, 观察 operation 中具体的操作内容, 并观察各个属性, 理解每个属性的含义;

【SQL 语句】:

```

begin tran
    select SECTOR_ID from tbcellnew where HEIGHT < 7
    update tbcellnew set HEIGHT = 10 where SECTOR_ID = '1246778-2'
    insert into tbcellnew (SECTOR_ID, SECTOR_NAME, HEIGHT) values ('211100-2', '123', 6)
    delete from tbcellnew where SECTOR_ID = '211100-2'
rollback tran
SELECT * FROM [sys].[fn_dblog] (NULL, NULL)

```

【结果】:

| | | | | | | |
|------|------------------------|-----------------|----------|---------------|---|--------|
| 2224 | 000000d3:00000179:0009 | LOP_BEGIN_XACT | LCX_NULL | 0000:00002b76 | 0 | 0x0000 |
| 2225 | 000000d3:00000179:000a | LOP_INSERT_ROWS | LCX_HEAP | 0000:00002b76 | 0 | 0x0000 |
| 2226 | 000000d3:00000179:000b | LOP_DELETE_ROWS | LCX_HEAP | 0000:00002b76 | 0 | 0x0000 |
| 2227 | 000000d3:00000179:000c | LOP_ABORT_XACT | LCX_NULL | 0000:00002b76 | 0 | 0x0000 |

【分析】:

因为以 rollback 结束事务，此事务以 LOP_ABORT_XACT 结束，对应于<T,abort>，这里的第 2227 行。

11.3.2.3 事务发生中断的事务日志实验

【实验要求】:

以备份表 tbCellnew 为访问对象，

Step1. 查询小区/基站工参表的小区天线高度 (HEIGHT) 小于 7 的 SECTOR_ID;

Step2. 修改小区/基站工参表的 SECTOR_ID 为'124678-2'的小区的高度为'10';

Step3. 添加一个延时 2~3 分钟;

Step4. 在小区/基站工参表中，添加一条 SECTOR_ID 为“211100-2”、HEIGHT 为 6 的信息删除 step3 所添加的信息

Step5. 将以上操作组织成事务执行 (以 begin tran 开始，以 commit tran 结束)，要求在延时的时间范围内发生一个外部冲突 (实验时可启动任务管理器结束 SQLServer 进程);

Step6. 重新连接数据库，并查看事务日志。

【SQL 语句】:

```

begin tran
    select SECTOR_ID from tbcellnew where HEIGHT < 7
    update tbcellnew set HEIGHT = 10 where SECTOR_ID = '1246778-2'
    WAITFOR DELAY '00:03:00'
    insert into tbcellnew (SECTOR_ID, SECTOR_NAME, HEIGHT) values ('211100-2', '123', 6)
    delete from tbcellnew where SECTOR_ID = '211100-2'
commit tran
SELECT * FROM [sys].[fn_dblog] (NULL, NULL)

```

【结果】:

| | | | | | |
|------|------------------------|--------------------|----------|---------------|---|
| 2235 | 000000d3:00000179:0014 | LOP_INSERT_ROWS | LCX_HEAP | 0000:00002b77 | 0 |
| 2236 | 000000d3:00000179:0015 | LOP_SET_FREE_SPACE | LCX_PFS | 0000:00000000 | 0 |
| 2237 | 000000d3:00000179:0016 | LOP_COMMIT_XACT | LCX_NULL | 0000:00002b77 | 0 |
| 2238 | 000000d3:0000017f:0001 | LOP_BEGIN_XACT | LCX_NULL | 0000:00002b79 | 0 |
| 2239 | 000000d3:0000017f:0002 | LOP_MODIFY_COLUMNS | LCX_HEAP | 0000:00002b79 | 0 |
| 2240 | 000000d3:0000017f:0003 | LOP_MODIFY_COLUMNS | LCX_HEAP | 0000:00002b79 | 0 |
| 2241 | 000000d3:0000017f:0004 | LOP_ABORT_XACT | LCX_NULL | 0000:00002b79 | 0 |

100 % ▾

结果

消息

| | CITY | SECTOR_ID | SECTOR_NAME | ENODEBID | ENODEB_NAME | EARFCN | PCI | PSS | SSS | TAC | VENDOR | LONGITUDE | LATITUDE | STYLE | AZIMUTH | HEIGHT | ELECTTILT | MECHTILT | TOTILETILT |
|---|------|-----------|-------------|----------|-------------|--------|------|------|------|------|--------|-----------|----------|-------|---------|--------|-----------|----------|------------|
| 1 | NULL | 1246778-2 | 123 | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | 6 | NULL | NULL | NULL |

【分析】:

在事务的执行过程中，如果发生外部中断，会产生型的事务日志，事务会被回滚，事务内的各项更新不成功，这里的 HEIGHT 仍然为 6。

11.3.3 显示执行模式下，多条或单条语句组成的多个并发事务的日志观察

【实验要求】:

以备份表 tbCellnew 为访问对象，

step1. 开启两个连接，模拟两个并行的事务。简称会话一和会话二；

step2. 在会话一中将隔离级别设置为提交读或者未提交读，以 tbCellnew 为访问对象，查询 SECTOR_ID 为'124694-0'的小区的 SECTOR_NAME 和 PCI；设置推迟等待 5 秒语句，然后再写一条查询语句，同样是查询 SECTOR_ID 为'124694-0'的小区的 SECTOR_NAME 和 PCI，执行语句后的五秒内开始第三步；

step3. 第二步语句执行的五秒内在会话二中执行语句，开始事务，更新 SECTOR_ID 为'124694-0'的小区的 PCI 为 400，并提交事务；

step4. 在会话一中查看执行结果；

step5. 查看事务日志，观察 operation 中具体的操作内容，观察并观察各个属性，理解每个属性的含义；

【会话一 SQL】:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
begin tran
select PCI,SECTOR_NAME from tbcellnew where SECTOR_ID = '124694-0'
WAITFOR DELAY '00:00:05'
select PCI from tbcellnew where SECTOR_ID = '124694-0'
commit tran
select * from [sys].[fn_dblog](NULL,NULL)
```

【会话二 SQL】:

```
update tbcellnew set PCI = 400 where SECTOR_ID = '124694-0'
```

【结果】:

| | | | | | | | |
|------|------------------------|--------------------|----------|---------------|---|--------|----|
| 2238 | 000000d3:0000017f:0001 | LOP_BEGIN_XACT | LCX_NULL | 0000:00002b79 | 0 | 0x0000 | 76 |
| 2239 | 000000d3:0000017f:0002 | LOP_MODIFY_COLUMNS | LCX_HEAP | 0000:00002b79 | 0 | 0x0000 | 62 |
| 2240 | 000000d3:0000017f:0003 | LOP_MODIFY_COLUMNS | LCX_HEAP | 0000:00002b79 | 0 | 0x0000 | 62 |
| 2241 | 000000d3:0000017f:0004 | LOP_ABORT_XACT | LCX_NULL | 0000:00002b79 | 0 | 0x0000 | 80 |
| 2242 | 000000d3:0000017f:0005 | LOP_BEGIN_XACT | LCX_NULL | 0000:00002b7a | 0 | 0x0000 | 76 |
| 2243 | 000000d3:0000017f:0006 | LOP_MODIFY_COLUMNS | LCX_HEAP | 0000:00002b7a | 0 | 0x0000 | 62 |
| 2244 | 000000d3:0000017f:0007 | LOP_COMMIT_XACT | LCX_NULL | 0000:00002b7a | 0 | 0x0000 | 80 |
| 2245 | 000000d3:00000181:0001 | LOP_BEGIN_XACT | LCX_NULL | 0000:00002b7b | 0 | 0x0000 | 76 |

【分析】:

在并发事务里，第 2238 行到 2241 行是会话一的事务，第 2242 行到 2245 行是会话二的事务。

四、实验总结

在本次实验中通过实际的事务操作，让我更加透彻地理解了事务相关的概念，事务应该具有 4 个属性：原子性、一致性、隔离性、持久性。这四个属性通常称为 ACID 特性。实验过程中掌握了保存点的保存与回滚相关，事务执行的提交与回滚，了解到了回滚对于数据库操作本身的要求。

在多事务并发相关部分的操作过程中，通过实际的操作对四种隔离级别以及潜在的并发访问问题进行了深入理解。事务隔离级别定义了当一个事务与其它事务同时并发访问相同资源或数据时，为避免多个事务并发访问带来的副作用，必须与由其它事务间保持的资源或数据更改相隔离的程度。本次实验中涉及到的隔离级别包括，read uncommitted、read committed、repeatable read、serializable；而相关存在的并发访问问题包括，并发事务执行导致的数据不一致性，如丢失修改（写-写错误）、读脏数据（写-读错误）、不可重复读（读-写错误）、幻象。SQL Server 为上述内容提供了详尽的事物保障机制。

同时，在多事务并发过程中还涉及到相关的各类锁问题，我们通过简单的示例对锁的工作流程以及锁表的内容进行了简要的分析与锁相关的总结。

理解了实际数据库系统中原子性保障机制与教科书中事务原子性概念的差异，了解了 SQL SERVER 提供的各种事务隔离级别。观察分析在“读提交”隔离级别下，并发事务执行导致的数据不一致性，如丢失修改（写-写错误）、读脏数据（写-读错误）、不可重复读（读-写错误）、幻象；了解了 SQL SERVER 提供的事务一致性和独立性保障机制，了解了 SQL Server 日志机制；观察分析当事务以 commit 结束、rollback 结束，和系统发生 crash 时，DBMS 采取的故障恢复动作日志内容。