

SQL 笔记 (基于 Lab 001-006 综合复习)

Chinook Database Schema Summary (Markdown Table)

以下是 Chinook 数据库的表结构总结，使用 Markdown 表格表示。每个表包括列名、类型、约束和简要描述。

Artist 表

Column	Type	Constraints	Description
ArtistId	INTEGER	PRIMARY KEY	艺术家 ID
Name	NVARCHAR(120)	NULL	艺术家姓名

Album 表

Column	Type	Constraints	Description
AlbumId	INTEGER	PRIMARY KEY	专辑 ID
Title	NVARCHAR(160)	NOT NULL	专辑标题
ArtistId	INTEGER	FOREIGN KEY (Artist)	艺术家 ID

Track 表

Column	Type	Constraints	Description
TrackId	INTEGER	PRIMARY KEY	音轨 ID
Name	NVARCHAR(200)	NOT NULL	音轨名称
AlbumId	INTEGER	FOREIGN KEY (Album)	专辑 ID
MediaTypeId	INTEGER	FOREIGN KEY (MediaType)	媒体类型 ID
GenreId	INTEGER	FOREIGN KEY (Genre)	类型 ID
Composer	NVARCHAR(220)	NULL	作曲家
Milliseconds	INTEGER	NOT NULL	时长 (毫秒)
Bytes	INTEGER	NULL	文件大小
UnitPrice	NUMERIC(10,2)	NOT NULL	单价

Genre 表

Column	Type	Constraints	Description
GenreId	INTEGER	PRIMARY KEY	类型 ID
Name	NVARCHAR(120)	NULL	类型名称

MediaType 表

Column	Type	Constraints	Description
--------	------	-------------	-------------

Column	Type	Constraints	Description
MediaTypeId	INTEGER	PRIMARY KEY	媒体类型 ID
Name	NVARCHAR(120)	NULL	媒体类型名称

Playlist 表

Column	Type	Constraints	Description
PlaylistId	INTEGER	PRIMARY KEY	播放列表 ID
Name	NVARCHAR(120)	NULL	播放列表名称

PlaylistTrack 表

Column	Type	Constraints	Description
PlaylistId	INTEGER	PRIMARY KEY	播放列表 ID
TrackId	INTEGER	PRIMARY KEY, FOREIGN KEY (Track)	音轨 ID

Customer 表

Column	Type	Constraints	Description
CustomerId	INTEGER	PRIMARY KEY	客户 ID
FirstName	NVARCHAR(40)	NOT NULL	名
LastName	NVARCHAR(20)	NOT NULL	姓
Company	NVARCHAR(80)	NULL	公司
Address	NVARCHAR(70)	NULL	地址
City	NVARCHAR(40)	NULL	城市
State	NVARCHAR(40)	NULL	州
Country	NVARCHAR(40)	NULL	国家
PostalCode	NVARCHAR(10)	NULL	邮编
Phone	NVARCHAR(24)	NULL	电话
Fax	NVARCHAR(24)	NULL	传真
Email	NVARCHAR(60)	NOT NULL	邮箱
SupportRepId	INTEGER	FOREIGN KEY (Employee)	支持员工 ID

Employee 表

Column	Type	Constraints	Description
EmployeeId	INTEGER	PRIMARY KEY	员工 ID
LastName	NVARCHAR(20)	NOT NULL	姓
FirstName	NVARCHAR(20)	NOT NULL	名

Column	Type	Constraints	Description
Title	NVARCHAR(30)	NULL	职位
ReportsTo	INTEGER	FOREIGN KEY (Employee)	上级 ID
BirthDate	DATETIME	NULL	生日
HireDate	DATETIME	NULL	入职日期
Address	NVARCHAR(70)	NULL	地址
City	NVARCHAR(40)	NULL	城市
State	NVARCHAR(40)	NULL	州
Country	NVARCHAR(40)	NULL	国家
PostalCode	NVARCHAR(10)	NULL	邮编
Phone	NVARCHAR(24)	NULL	电话
Fax	NVARCHAR(24)	NULL	传真
Email	NVARCHAR(60)	NULL	邮箱

Invoice 表

Column	Type	Constraints	Description
InvoiceId	INTEGER	PRIMARY KEY	发票 ID
CustomerId	INTEGER	FOREIGN KEY (Customer)	客户 ID
InvoiceDate	DATETIME	NOT NULL	发票日期
BillingAddress	NVARCHAR(70)	NULL	账单地址
BillingCity	NVARCHAR(40)	NULL	账单城市
BillingState	NVARCHAR(40)	NULL	账单州
BillingCountry	NVARCHAR(40)	NULL	账单国家
BillingPostalCode	NVARCHAR(10)	NULL	账单邮编
Total	NUMERIC(10,2)	NOT NULL	总金额

InvoiceLine 表

Column	Type	Constraints	Description
InvoiceLineId	INTEGER	PRIMARY KEY	发票行 ID
InvoiceId	INTEGER	FOREIGN KEY (Invoice)	发票 ID
TrackId	INTEGER	FOREIGN KEY (Track)	音轨 ID
UnitPrice	NUMERIC(10,2)	NOT NULL	单价
Quantity	INTEGER	NOT NULL	数量

1. 基本语法与数据类型

- **数据类型：**
 - INTEGER: 整数 (主键自增用 INTEGER PRIMARY KEY AUTOINCREMENT)
 - TEXT: 字符串 (VARCHAR(n) 在 SQLite 中等同 TEXT)
 - REAL: 浮点数
 - NUMERIC: 精确数值 (如价格)
 - DATETIME: 日期时间 (用 TEXT 存储)
- **注释：** -- 单行注释, /* 多行 */

2. 创建表 (CREATE TABLE)

- 基本语法：

```
CREATE TABLE IF NOT EXISTS TableName (  
    col1 INTEGER PRIMARY KEY AUTOINCREMENT,  
    col2 TEXT NOT NULL,  
    col3 REAL CHECK (col3 >= 0),  
    FOREIGN KEY (col1) REFERENCES OtherTable(col1) ON DELETE CASCADE  
);
```

- 约束：
 - PRIMARY KEY: 唯一标识行
 - FOREIGN KEY: 引用其他表, ON DELETE/UPDATE 行为 (CASCADE, RESTRICT, SET NULL)
 - NOT NULL: 不能为空
 - UNIQUE: 值唯一
 - CHECK: 条件检查 (e.g., CHECK (price > 0))
 - DEFAULT: 默认值

3. 插入数据 (INSERT)

- 语法：

```
INSERT INTO TableName (col1, col2) VALUES (val1, val2);  
INSERT OR IGNORE INTO TableName VALUES (...); -- 忽略冲突  
INSERT OR REPLACE INTO TableName VALUES (...); -- 替换冲突
```

- 批量插入：

```
INSERT INTO TableName (col1, col2) VALUES  
    (val1, val2),  
    (val3, val4);
```

4. 查询 (SELECT)

- 基本语法：

```
SELECT col1 AS "Alias", col2  
FROM TableName  
WHERE condition
```

```
GROUP BY col1
HAVING aggregate_condition
ORDER BY col1 ASC/DESC
LIMIT n OFFSET m;
```

- WHERE 条件:
 - 比较: =, !=, <, >, <=, >=
 - 逻辑: AND, OR, NOT
 - IN: col IN (val1, val2)
 - LIKE: col LIKE 'pattern%' (通配符 % _)
 - IS NULL / IS NOT NULL
- JOIN:
 - INNER JOIN (JOIN): 匹配行
 - LEFT JOIN: 保留左表所有行
 - RIGHT JOIN: SQLite 不支持, 用 LEFT JOIN 反转
 - FULL OUTER JOIN: SQLite 不支持, 用 UNION 模拟
 - 示例: FROM A JOIN B ON A.id = B.id
- 聚合函数:
 - COUNT(*): 行数
 - SUM(col): 求和
 - AVG(col): 平均
 - MIN/MAX(col): 最小/最大
 - ROUND(col, n): 四舍五入到 n 位小数
- 子查询:
 - 在 WHERE: WHERE col IN (SELECT ...)
 - 在 FROM: FROM (SELECT ...) AS sub
 - EXISTS: WHERE EXISTS (SELECT 1 FROM ... WHERE ...)
- 窗口函数 (SQLite 3.25+):
 - ROW_NUMBER() OVER (PARTITION BY col ORDER BY col): 排名
 - RANK(): 并列排名
 - 示例: SELECT *, ROW_NUMBER() OVER (ORDER BY sales DESC) AS rank FROM ...

5. 更新与删除 (UPDATE / DELETE)

- UPDATE:

```
UPDATE TableName SET col1 = val1, col2 = val2 WHERE condition;
```

- DELETE:

```
DELETE FROM TableName WHERE condition;
```

- 事务:

```
BEGIN TRANSACTION;
-- SQL statements
COMMIT;
ROLLBACK; -- 撤销
```

6. SQLite 特定命令

- PRAGMA foreign_keys = ON; -- 启用外键约束
- .commands (在 sqlite3 提示符):
 - .tables: 列出表
 - .schema TableName: 查看表结构
 - .read file.sql: 执行 SQL 文件
 - .dump: 导出数据库为 SQL
 - .exit: 退出
- 类型转换: CAST(col AS INTEGER/TEXT/REAL)

7. 常见查询模式 (基于 Lab 示例)

- 列出所有电影与演员: SELECT m.title, a.name FROM Movie m JOIN Actor a ON m.actId = a.actId;
- 按类型分组统计: SELECT genre, COUNT(*) FROM Movie GROUP BY genre;
- 平均价格: SELECT genre, ROUND(AVG(price), 2) FROM Movie GROUP BY genre;
- 过滤后查询: SELECT * FROM Movie WHERE year > 2000 AND genre IN ('Action', 'Drama');
- 子查询过滤: SELECT * FROM Movie WHERE genre IN (SELECT genre FROM Movie GROUP BY genre HAVING COUNT(*) > 2);
- 总销量: SELECT m.title, SUM(s.sales) FROM Movie m LEFT JOIN Sales s ON m.id = s.id GROUP BY m.id;
- 最受欢迎演员: 用子查询或窗口函数找出每个区域平均销量最高的演员。

8. 考试应对技巧

- **理解问题:** 看清是 INNER/LEFT JOIN, 是否包含 NULL, 排序方向。
- **检查约束:** 外键是否启用, 数据类型匹配。
- **调试:** 用 .schema 检查表, SELECT COUNT(*) 验证行数。

9. Chinook Database SELECT Query Examples

基于提供的 Chinook schema, 以下是两个示例 SELECT 查询, 设计用于覆盖多个考试要点 (e.g., JOIN, GROUP BY, HAVING, ORDER BY, CAST, ROUND, SUM, 子查询等)。每个查询包括简要解释。

示例 1: 每个类型的总销售额 (带过滤)

此查询计算每个类型的总销售额, 但仅包括总销售额超过 100 的类型, 按销售额降序排列。

```
SELECT g.Name AS "Genre",
       ROUND(SUM(il.UnitPrice * il.Quantity), 2) AS "Total_Sales"
FROM Genre g
JOIN Track t ON g.GenreId = t.GenreId
JOIN InvoiceLine il ON t.TrackId = il.TrackId
GROUP BY g.GenreId, g.Name
HAVING SUM(il.UnitPrice * il.Quantity) > 100
ORDER BY Total_Sales DESC;
```

解释:

- **JOIN:** 多个 INNER JOIN 连接 Genre → Track → InvoiceLine。
- **GROUP BY:** 按类型分组以聚合销售额。
- **SUM:** 计算总销售额 (UnitPrice * Quantity)。

- **ROUND**: 将结果四舍五入到 2 位小数。
- **HAVING**: 过滤总销售额 > 100 的组 (不能用 WHERE 处理聚合)。
- **ORDER BY**: 按总销售额降序排序。
- **覆盖**: 聚合、组后过滤、多表连接、数值运算。

示例 2: 每个艺术家的平均音轨长度 (带子查询和 CAST)

此查询查找每个艺术家的平均音轨长度 (分钟), 但仅限于有超过 5 首音轨的艺术家, 按平均长度升序排列。

```
SELECT a.Name AS "Artist",
       ROUND(AVG(CAST(t.Milliseconds AS REAL) / 60000), 2) AS "Avg_Track_Length_Minutes"
FROM Artist a
JOIN Album al ON a.ArtistId = al.ArtistId
JOIN Track t ON al.AlbumId = t.AlbumId
WHERE a.ArtistId IN (
    SELECT al2.ArtistId
    FROM Album al2
    JOIN Track t2 ON al2.AlbumId = t2.AlbumId
    GROUP BY al2.ArtistId
    HAVING COUNT(t2.TrackId) > 5
)
GROUP BY a.ArtistId, a.Name
ORDER BY Avg_Track_Length_Minutes ASC;
```

解释:

- **JOIN**: 连接 Artist → Album → Track。
- **CAST**: 将 Milliseconds 转换为 REAL 以进行除法 (计算分钟)。
- **AVG 和 ROUND**: 计算平均长度并四舍五入到 2 位小数。
- **WHERE 中的子查询**: 使用子查询与 GROUP BY 和 HAVING 过滤有 >5 首音轨的艺术家 (测试子查询和嵌套聚合)。
- **GROUP BY**: 按艺术家分组以计算平均值。
- **ORDER BY**: 按平均长度升序排序。
- **覆盖**: 类型转换、算术运算、子查询、复杂过滤、带条件的聚合。