Lecture 3: SQL

Database

Author: Forliage

Email: masterforliage@gmail.com

Date: June 10, 2025

College: 计算机科学与技术学院

Abstract

本讲笔记系统介绍了 SQL 语言的核心内容与常用语句，主要包括以下几个方面：首先，在“数据定义语言”部分，阐述了如何使用 CREATE、ALTER、DROP 等命令定义表结构、域类型及完整性约束，并讲解了索引的创建与管理；其次，“基本结构”章节说明了 SELECT 查询的基本格式，包括 FROM、WHERE、DISTINCT、ORDER BY 等子句的用法，以及 SQL 与关系代数操作的对应关系；随后，讲解了集合操作（UNION、INTERSECT、EXCEPT 及其 ALL 版本）、聚合函数（AVG、SUM、COUNT、MIN、MAX）和 GROUP BY/HAVING 分组统计的实现；然后讨论了 NULL 值的三值逻辑语义及其在算术表达式和比较中的处理规则；接着介绍了嵌套子查询的写法与 EXISTS/IN 等典型用法；在“视图”与“派生关系”部分，说明了 VIEW 定义、WITH 子句本地视图和视图更新的限制；紧接着，覆盖了 INSERT、UPDATE、DELETE 等数据库修改语句；最后，在“连接关系”中详细分类讲解了内连接、外连接及自然连接的定义与语法。通过本讲学习，读者能够全面掌握 SQL 的表结构定义、查询处理、数据聚合与更新操作，为实际数据库开发与查询优化奠定基础。

（该Abstract由ChatGPT-o4-mini-high生成）

Contents

1 数据定义语言

DDL的主要功能包括：

- 为每个关系定义模式
- 定义与每个属性相关的值域
- 定义完整性约束
- 定义每个关系在磁盘上的物理存储结构
- 定义每个关系要维护的索引
- 定义关系的视图

1.1 SQL中的域类型

- char(n):固定长度字符字符串，长度由用户指定
- varchar(n):可变长度字符字符串，最大长度由用户指定n
- int:整数（一个有限的整数子集，依赖于机器）
- smallint:小整数（整数域类型的一个依赖于机器的子集）
- numeric(p,d):定点数，具有用户指定的p位精度，小数点右侧有d位
- real,double precision:浮点数和双精度浮点数，具有机器相关的精度
- float(n):浮点数，用户指定的精度至少为n位
- 所有域类型都允许空值。声明某属性为非空将禁止该属性的空值
- date:日期(4位数字)年份、月份、日期
- Time:一天中的时间，以小时、分钟和秒表示
- timestamp:日期+时间

SQL中有许多函数用于处理各种类型的数据及其类型转换，但各数据库系统中函数的标准化程度不高。

1.2 创建表

```
CREATE TABLE r(A1D1, A2D2,...,AnDn,
                (integrity constraint 1),
                \dots
                (integrity constraint n))
```

r是关系名称；每个Ai是关系r模式中的属性名称；Di是属性Ai域中值的数据类型。

1.3 创建表中的完整性约束

非空

主键$(A_1, A_2, ..., A_n)$

检查(P)，其中P是一个谓词

1.4 删除和修改表

删除表命令会从数据库中删除关于被删除关系的所有信息。

```
DROP TABLE r
```

使用删除命令时请小心。

修改表命令用于向现有关系添加属性：

```
ALTER TABLE r ADD A D;
ALTER TABLE r ADD (A1D1, ..., AnDn);
```

其中A是要添加到关系r的属性名称，D是A的域。

修改表命令也可以用于删除关系的属性：

```
ALTER TABLE r DROP A
```

其中A是关系r中属性的名称。

注意，许多数据库不支持删除属性。

修改表命令也可以用于修改关系的属性。

1.5 创建索引

```
CREATE INDEX <i-name> ON <table-name> (<attribute-list>);

CREATE UNIQUE INDEX <i-name> ON <table-name> (<attribute-list);

DROP INDEX <i-name>;
```

2 基本结构

```
SELECT A1,A2,...,An
FROM r1,r2,...,rm
```

```
3 WHERE P
```

A_i:属性;r_i:关系;P:谓词

等价于：$\Pi_{A_1,A_2,...,A_n}(\sigma_P(r_1 \times r_2 \times ... \times r_m))$

注意：SQL不允许在名称中使用-字符，因此在实际实现中使用branch_name而不是branch-name.

注意：SQL名称不区分大小写，即可以使用大写或小写字母。

SQL允许关系和查询结果中存在重复项。

要强制消除重复项，请在选择后插入关键字distinct.

```
1 SELECT distinct branch_name
2 FROM loan
```

对立关键字all允许重复.

```
1  SELECT all branch_name
2  FROM loan
```

默认情况下，允许重复，即all是默认值。

选择子句中的*表示所有属性

```
1 SELECT * FROM loan
```

然而，选择子句可以包含涉及运算+,-,*和/的算术表达式，以及对常量或元组属性的操作：

```
1 SELECT loan_number , branch_name , amount * 100
2 FROM loan
```

WHERE子句指定结果必须满足的条件。

在WHERE子句中，可以使用逻辑连接词包括AND、OR和NOT来组合比较结果，同时可以使用BETWEEN比较运算符来指定范围。

```
1 SELECT loan_number
2 FROM loan
3 WHERE amount BETWEEN 90000 AND 100000
```

FROM子句列出了查询中涉及的关系，如果在FROM子句中指定了多个关系，则对应于关系代数的笛卡尔积操作。

SQL允许使用as子句重命名关系和属性：old_name as new_name

元组变量通过使用as子句在FROM子句中定义。

SQL包含一个用于字符字符串比较的字符串匹配操作符。模式使用以下两个特殊字符描述：

- %：匹配任何子字符串（类似于文件系统中的*）
- _：匹配任何字符（类似于文件系统中的?）

注意：可以实现模糊匹配（放置WHERE子句，并且必须与LIKE操作一起使用）

```
SELECT customer_name
FROM customer
WHERE customer_name LIKE '%Ze%'
```

我们可以指定desc表示降序，asc表示升序，对于每个属性，升序是默认值。

```
SELECT * FROM customer
ORDER BY customer_city,customer_street desc, customer_name
```

一些关系代数运算符的多重集版本支持重复。

给定多重集关系r_1和r_2:

- $\sigma_\theta(r_1)$:如果在r_1中有c_1个元组t_1的副本，并且t_1满足选择σ_θ，则在$\sigma_\theta(r_1)$中有c_1个$\sigma_\theta(t_1)$的副本
- $\Pi_A(r_1)$:同理
- $r_1 \times r_2$:同理

SQL重复语义:

```
SELECT A1,A2,...,An
FROM r1,r2,...,rm
WHERE P
```

等价于$\Pi_{A_1,A_2,\ldots,A_n}(\sigma_P(r_1 \times r_2 \times \cdots \times r_m))$

3 集合操作

在SQL中，集合运算包括UNION,INTERSECT和EXCEPT，对应于$\cup, \cap, -$

这些操作每个都会自动消除重复项。为了保留重复项，我们可以使用相应的多重集版本，包括UNION ALL,INTERSECT ALL和EXCEPT ALL。

4 聚合函数

- avg(col):平均值
- min(col):最小值
- max(col):最大值
- sum(col):值的总和
- count(col):值的数量

HAVING子句

```
branch(branch-name,branch-city,assets)
account(account-number,branch-name,balance)

SELECT A.branch_name,avg(balance)
FROM account A, branch B
WHERE A.branch_name = B.branch_name and
      branch_city = 'Brooklyn'
GROUP BY A.branch_name
HAVING avg(balance) > 1200
```

SELECT语句的形式：

```
SELECT <[DISTINCT] c1,c2,...>
FROM <r1,...>
[WHERE <condition>]
[GROUP BY <c1,c2,...>[HAVING <cond2>]]
[ORDER BY <c1 [DESC][c2 [DESC|ASC],...]>]
```

SELECT执行顺序：

from→where→group(aggregate)→having→select→distinct→order by

注意：having子句中的谓词在形成组之后应用，而where子句中的谓词在形成组之前应用。

聚合函数不能直接在where子句中使用。

5 空值

其含义是“缺失信息”或“无关信息”,即未知值或值不存在

任何涉及“空值”的算术表达式的结果都是空值。

与空值的任何比较返回“未知”

三值逻辑使用未知真值

如果where子句的谓词评估为未知，则结果被视为假。

P是未知的"当谓词P评估为未知时为真。然而，聚合函数会简单地忽略空值。

所有聚合操作(除了count(*))都忽略聚合属性中具有空值的元组。

6 嵌套子查询

SQL提供了嵌套子查询的机制。

子查询是嵌套在另一个查询中的select_from_where表达式。

子查询的一个常见用途是进行集合成员资格测试和集合比较。

如果参数子查询为空，则exists构造返回值为真。

7 视图

提供了一种机制，以隐藏某些数据，使特定用户无法查看。

```
CREATE VIEW <v_name> AS
    SELECT c1,c2,... From ...
CREATE VIEW <v_name> (c1,c2,...) AS
    SELECT e1,e2,...From\dots
```

好处：安全、易于使用，支持逻辑独立

删除视图：

```
DROP VIEW <V_NAME>
```

8 派生关系

WITH子句允许为查询在本地定义视图，而不是全局定义。

9 数据库修改

```
SELETE FROM <table|view>
[WHERE <condition>]
```

```
INSERT INTO <table|view>[(c1,c2,...)]
VALUES(e1,e2,...)

INSERT INTO <table|view>[(c1,c2,...)]
SELECT e1,e2,\dots
FROM \dots
```

```
UPDATE <table|view>
SET<c1=e1,[c2=e2,\dots]>[WHERE <condition>]
```

建立在单个基本表上的视图，且视图的列对应表的列，称为"行列视图"。

视图是虚表，对其进行的所有操作都将转化为对基表的操作。

在查询操作时，视图与基表没有区别，但对视图的更新操作有严格限制，例如只有行列视图可以更新数据。

大多数SQL实现仅允许对定义在单一关系上且没有聚合的简单视图进行更新。

10 连接关系

连接操作以两个关系为输入，返回另一个关系作为结果。连接条件-定义两个关系中哪些元组匹配，以及连接结果中包含哪些属性。

连接类型-定义在每个关系中哪些元组未与另一个关系中的任何元组匹配(基于连接条件)时的处理方式。连接类型内连接左外连接右外连接全外连接连接条件自然在使用