



浙江大学
ZHEJIANG UNIVERSITY

第三章 关系数据库标准语言

陶煜波

计算机科学与技术学院

第一章 地理空间数据库概论回顾

- 关系数据库基本概念

- 概念数据模型 → 逻辑数据模型 → 物理数据模型
- 逻辑数据模型
 - 层次、网状、关系、面向对象、对象关系
- 关系模型
 - 模式、内模式、外模式
 - 映像关系

- 地理空间数据库基本概念

- 矢量数据，栅格数据
- 特点：空间特征，非结构化特征，空间关系特征，时态特征，多尺度特征
- 地理空间数据库 = 关系数据库管理系统 + 空间扩展
 - PostgreSQL + PostGIS (空间数据类型、空间分析、空间索引)

第一章 地理空间数据库概论回顾

- 空间数据管理技术的产生与发展
 - 文件系统
 - 文件与关系数据库混合管理系统
 - 空间数据引擎
 - 对象关系型数据库管理系统
- 空间数据库标准
 - SFA SQL
 - SQL/MM
- 空间计算
 - 空间相关性/关联性 (因果关系→相关关系)
 - 时空查询

第二章 关系模型与关系代数回顾

- 基本概念
 - A **Database Management System (DBMS)** is a piece of software designed to store and manage databases
 - A **data model** is a collection of concepts for describing data
 - A **schema** is a description of a particular collection of data, using **the given data model**
- 什么是逻辑数据模型的数据？
 - 概念数据模型
 - 实体和实体之间的联系
- 如何表达实体之间的联系？
 - 层次模型和网状模型：指针
 - 关系模型：关系/外码

第二章 关系模型与关系代数回顾

- 数据结构

- 关系：关系名、关系模式和关系实例

- 基本关系、查询表、视图表

- 关系模式： $R(A_1, A_2, \dots, A_n)$

- 逻辑结构：二维表

- 属性、域、码、元组、分量

- 数据操作

- 完整性约束

第二章 关系模型与关系代数回顾

- 数据结构
- 数据操作
 - 集合操作 (输入和输出都是集合)
 - 查询
 - 选择、投影、连接、除、并、交、差
 - 数据更新
 - 插入、删除、修改
 - 关系代数
- 完整性约束

第二章 关系模型与关系代数回顾

- 数据结构
- 数据操作
- 完整性约束 (关系的约束条件)
 - 实体完整性
 - 主码(primary key), 唯一, 不能取NULL (unknown or undefined)
 - 参照完整性
 - 外码(foreign key), 取NULL或参照关系中的主码或Unique值
 - 用户定义完整性

sid	Name	GPA
101	Bob	3.2
123	Mary	3.8

Students

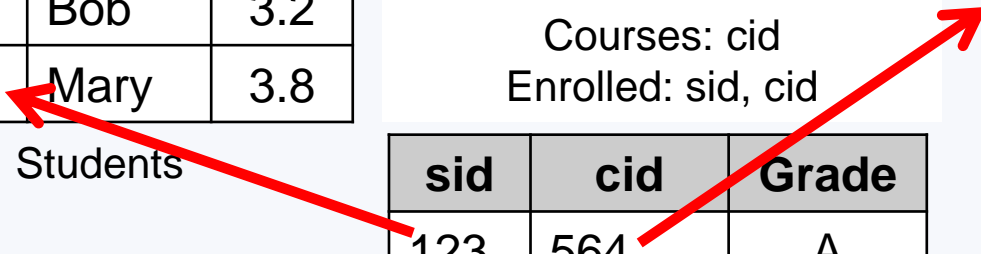
Primary Key:
Students: sid
Courses: cid
Enrolled: sid, cid

sid	cid	Grade
123	564	A

Enrolled

cid	cname	credits
564	564-2	4
308	417	2

Courses



第二章 关系模型与关系代数回顾

- 运算符
 - 集合运算符
 - 并 \cup , 交 \cap , 差 $-$, 笛卡尔积 \times
 - 专门的关系运算符
 - 选择 σ , 投影 π , 连接 \bowtie , 除 \div
 - 算术比较符
 - $> \geq < \leq = \neq$
 - 逻辑运算符
 - $\neg \wedge \vee$
 - 辅助操作
 - 重命名 ρ

第二章 关系模型与关系代数回顾

- 关系代数语义
 - set - 标准关系代数 (RA)
 - 并, 交, 差, 选择, 投影, 连接
 - $\{1, 2, 3\}$
 - multiset, bag - 扩展关系代数 (SQL)
 - 重复消除, 分组与聚集, 排序
 - $\{1, 1, 2, 3\}$
- 规则
 - Every paper will assume set semantics
 - Every implementation will assume bag semantics
- A relation or table is a multiset of tuples having the attributes specified by the schema

第三章 关系数据库标准语言

- 3.1 SQL概述
- 3.2 数据定义
- 3.3 数据更新
- 3.4 数据查询

参考教材：
数据库系统概念 3.1-3.10, 4.1, 4.4

3.1 SQL概述

- SQL (stands for **S**tructured **Q**uery **L**anguage)
 - A standard language for querying and manipulating data
 - A very **high-level** (optimized) programming language
- Dark times 10 years ago
 - Are databases dead?
 - NoSQL = No SQL
- Now, as before: everyone sells SQL
 - Pig, Hive, Impala
 - NoSQL = Not Only SQL
- “合久必分，分久必合”



SQL的产生与发展

- 1974年由IBM公司的Boyce和Chamberlin提出
- 1986年10月美国国家标准局 (American National Standard Institute, 简称ANSI)的数据库委员会X3H2批准了SQL作为关系数据库语言的美国标准。同年公布了SQL标准文本 (简称SQL-86)
- 1987年国际标准化组织 (ISO)也通过了这一标准
- 1989年公布了SQL-89标准
- 1992年公布了SQL-92标准 (又称SQL2)
- 1999年公布了SQL-99标准 (SQL3)
- 当前版本SQL-2016

SQL内容发展

注意：不同数据库系统，如 Oracle, SQL Server, MySQL, PostgreSQL等，支持SQL标准不同，语法也有所不同。掌握原理，具体数据库参考帮助文档

- SQL92 is a basic set
 - Most systems support at least this
- SQL-1999 Introduced “Object-Relational” concepts
 - Not fully supported yet
- Current standard is SQL-2016
 - 2003 was last major update: XML, window functions, sequences, auto-generated IDs
 - 2008 added TRUNCATE, x-query stuff, new triggers (instead of)
 - 2011 added temporal data definition and manipulation
 - 2016 added JSON, polymorphic tables
 - Also not fully supported yet

SQL内容

- Data Definition Language (DDL)
 - Define relational *schemata*
 - Create/alter/delete tables and their attributes
- Data Manipulation Language (DML)
 - Insert/delete/modify tuples in tables
 - Query one or more tables

A relation or table is a **multiset** of tuples having the attributes specified by the schema

A multiset is an unordered list, {1, 1, 2, 3, 2}

A tuple or row or **record** is a single entry in the table having the attributes specified by the schema

Product

PName	Price	Manufacturer
iPhone X	¥ 6349	Apple
Galaxy Note9	¥ 6569	Sumsung
Mate 20 Pro	¥ 5399	Huawei
MIX3	¥ 3299	MI

An attribute or column is a typed data entry present in each tuple in the relation

*Attributes must have an **atomic** type in standard SQL, i.e. not a list, set, etc.*

SQL特点

- 综合统一

- 集数据定义语言**DDL**、数据操纵语言**DML**、数据控制语言**DCL**的功能于一体，语言风格统一，可以独立完成数据库生命周期中的全部活动
- 在关系模型中实体和实体间的联系均用关系表示，数据结构单一性带来了数据操作符的统一，查找、插入、删除、更新等操作都只需一种操作符

- 高度非过程化

- 非关系数据模型的数据操纵语言是面向过程的语言，在执行一项工作时必须描述“怎么做”
- **SQL**语言是非过程语言，使用它进行数据库操作时，只须提出“做什么”，而无须指明“怎么做”

SQL特点

- 综合统一
- 高度非过程化
- 面向集合的操作方式
 - 非关系数据模型采用的是面向记录的操作方式，操作对象是一条记录
 - **SQL**语言采用集合操作方式，不仅操作对象、查找结果可以是元组的集合，而且一次插入、删除、更新操作的对象也可以是元组的集合

SQL特点

- 综合统一
- 高度非过程化
- 面向集合的操作方式
- 以同一种语法结构提供两种使用方式
 - SQL语言既是自含式语言，又是嵌入式语言
 - 作为自含式语言，它能够独立地用于联机交互的使用方式，用户可以在终端键盘上直接键入SQL命令对数据库进行操作
 - 作为嵌入式语言，SQL语句能够嵌入到高级语言 (例如C/C++，Java，C#，Python，JavaScript)程序中，供程序员设计程序时使用

SQL特点

- 综合统一
- 高度非过程化
- 面向集合的操作方式
- 以同一种语法结构提供两种使用方式
- 语言简捷，易学易用

SQL 功 能	动 词
数 据 定 义	CREATE, DROP, ALTER
数 据 查 询	SELECT
数 据 操 纵	INSERT, UPDATE, DELETE
数 据 控 制	GRANT, REVOKE

第三章 关系数据库标准语言

- 3.1 SQL概述
- 3.2 数据定义
- 3.3 数据更新
- 3.4 数据查询

参考教材：
数据库系统概念 3.1-3.10, 4.1, 4.4

3.2 数据定义

- 关系数据库的基本对象是基本表、视图和索引。因此**SQL**的数据定义功能包括

- 定义基本表
- 定义视图
- 定义索引

注意:

SQL commands are case insensitive
(Same: SELECT, Select, select)

Values are not

(Different: 'iPhone', 'iphone')

Using single quotes for **constants**
(‘abs’ – yes, “abs” – no)

操 作 对 象	操 作 方 式		
	创 建	删 除	修 改
基本表	CREATE TABLE	DROP TABLE	ALTER TABLE
视 图	CREATE VIEW	DROP VIEW	
索 引	CREATE INDEX	DROP INDEX	

定义表

CREATE TABLE <表名>

(<列名> <数据类型>[<列级完整性约束条件>]
[, <列名> <数据类型> [<列级完整性约束条件>]] ...
[, <表级完整性约束条件>]);

- <表名>: 所要定义的基本表的名称
- <列名>: 组成该表的各个属性（列）
- <列级完整性约束条件>: 涉及相应属性列的完整性约束条件
- <表级完整性约束条件>: 涉及一个或多个属性列的完整性约束条件

数据类型

- 定义表的属性时需要指明该属性的域
- 在SQL中域的概念用数据类型来实现
- SQL提供了一些主要的数据类型，在实际使用中要遵照具体的DBMS规定
 - Microsoft Access、MySQL和SQL Server数据类型
 - http://www.w3school.com.cn/sql/sql_datatypes.asp
 - PostgreSQL数据类型
 - <https://www.postgresql.org/docs/current/static/datatype.html>

数据类型

- Atomic types

- Characters

- char(20), varchar(50), text

- Numbers

- int, bigint, smallint, float, float8

- Times

- date, timestamp

- Geometries

- point, line, polygon, box

- Others

- bit, bool, serial, memory

Name	Aliases	Description
bigint	int8	signed eight-byte integer
bigserial	serial8	autoincrementing eight-byte integer
bit [(n)]		fixed-length bit string
bit varying [(n)]	varbit	variable-length bit string
boolean	bool	logical Boolean (true/false)
box		rectangular box on a plane
bytea		binary data ("byte array")
character [(n)]	char [(n)]	fixed-length character string
character varying [(n)]	varchar [(n)]	variable-length character string
cidr		IPv4 or IPv6 network address
circle		circle on a plane
date		calendar date (year, month, day)
double precision	float8	double precision floating-point number (8 bytes)
inet		IPv4 or IPv6 host address
integer	int, int4	signed four-byte integer
interval [fields] [(p)]		time span
json		textual JSON data
jsonb		binary JSON data, decomposed
line		infinite line on a plane
lseg		line segment on a plane
macaddr		MAC (Media Access Control) address
macaddr8		MAC (Media Access Control) address (EUI-64 format)
money		currency amount
numeric [(p, s)]	decimal [(p, s)]	exact numeric of selectable precision
path		geometric path on a plane
pg_lsn		PostgreSQL Log Sequence Number
point		geometric point on a plane
polygon		closed geometric path on a plane
real	float4	single precision floating-point number (4 bytes)
smallint	int2	signed two-byte integer
smallserial	serial2	autoincrementing two-byte integer
serial	serial4	autoincrementing four-byte integer
text		variable-length character string
time [(p)] [without time zone]		time of day (no time zone)
time [(p)] with time zone	timetz	time of day, including time zone
timestamp [(p)] [without time zone]		date and time (no time zone)
timestamp [(p)] with time zone	timestamptz	date and time, including time zone
tsquery		text search query
tsvector		text search document
txid_snapshot		user-level transaction ID snapshot
uuid		universally unique identifier
xml		XML data

PostgreSQL数据类型

思考：为什么每个属性/列只能是原子类型？

完整性约束

- The **schema** of a table is the table name, its attributes, and their types
 - Product(Pname: string, Price: float, Category: string, Manufacturer: string)
- 实体完整性: PRIMARY KEY
 - A **key** is a **minimal subset of attributes** that acts as a unique identifier for tuples in a relation
 - Product(Pname: string, Price: float, Category: string, Manufacturer: string)
 - A key is an implicit constraint on which tuples can be in the relation
 - i.e. If two tuples agree on the values of the key, then they must be the same tuples
 - multiset → set

完整性约束

- 实体完整性: PRIMARY KEY

- Students(sid: string, name: string, gpa: float)

- Q1: Which would you select as a key?

- Q2: Is a key always guaranteed to exist?

- Q3: Can we have more than one key?

- 参照完整性: FOREIGN KEY

- Enrolled(student_id: string, cid: string, grade: string)

- A student must appear in the Student table to enroll in a class

- student_id is a **foreign key** that refers to Students

- Q4: Which would you select as a key?

- 基本关系R的任何一个元组在外码上的取值要么是空值，要么是被参照关系S中一个元组的主码值/Unique值

- Referential integrity = Integrity of references = No “dangling pointer”

完整性约束

- 实体完整性: PRIMARY KEY
- 参照完整性: FOREIGN KEY
- 用户定义完整性: NOT NULL
 - NULL = unknown, or undefined
 - 某列不能取空值, 如Students关系中的name属性
- 用户定义完整性: UNIQUE
 - 某列或多个列的组合在关系中唯一
 - UNIQUE(name, age)
- 用户定义完整性: DEFAULT
 - 某列有默认值, 如当前登录用户, 当前时间等
- 用户定义完整性: CHECK
 - Check (age > 0)

注意: 由于性能原因, 实际应用中不会使用太多数据完整性约束

完整性约束

- 用CHECK实现NOT NULL限制
 - CREATE TABLE Student(sID INT, sName TEXT, GPA REAL CHECK(GPA is NOT NULL), sizeHS INT);
 - MySQL: accepts but does no enforce
- 用CHECK实现Keys
 - CREATE TABLE T(A int CHECK(A not in (SELECT A FROM T)));
 - CREATE TABLE T(A int CHECK((SELECT count(distinct A) FROM T) = (SELECT count(*) FROM T)));
 - SQLite, PostgreSQL: several issues
 - MySQL: accepts but does no enforce

完整性约束

- Subqueries in Check Constraints
 - SQLite, PostgreSQL: no subqueries in CHECK constraints
 - MySQL: accepts but does not enforce
 - create table Student(sID int, sName text, GPA real, sizeHS int);
 - create table Apply(sID int, cName text, major text, decision text, check(sID in (select sID from Student)));
 - create table College(cName text, state text, enrollment int, check(enrollment > (select max(sizeHS) from Student)));

注意：不同数据库实现的完整性约束不同

完整性约束

- 域约束
 - SQL语言可以使用CREATE DOMAIN语句定义新的值域
 - 在定义域时声明域的取值范围，如：
 - CREATE DOMAIN GenderDomain CHAR(2)
CHECK (VALUE IN ('男', '女'));
- 域使用举例：

```
CREATE TABLE S
(Sno    char(7) PRIMARY KEY,
Sname  char(8) NOT NULL,
Ssex   GenderDomain,
Sage   int,
Sdept  char(20));
```

完整性约束

- General assertions (断言)
 - SQL standard, but not implemented by any system
 - create assertion Keycheck ((select count(distinct A) from T) = (select count(*) from T));
 - create assertion ReferentialIntegritycheck (not exists (select * from Apply where sID not in (select sID from Student)));
 - create assertion Sizescheck (not exists (select * from College where enrollment <= (select max(sizeHS) from Student)));
 - create assertion AvgAcceptcheck (3.0 < (select avg(GPA) from Student where sID in (select SID from Apply where decision = 'Y')));

完整性约束

- 约束条件 (CONSTRAINT)

Schema and Constraints are how databases understand the semantics (meaning) of data

- NOT NULL 列
 - UNIQUE 列或列集合
 - PRIMARY KEY 列或列集合, 自动获得UNIQUE
 - FOREIGN KEY 列或列集合
- 预防破坏表之间连接的动作, 防止非法数据插入外键列
- CHECK 列或列集合 (attribute, tuple-based)
 - DEFAULT 列

ALTER COLUMN City SET DEFAULT 'SANDNES'

<https://www.postgresql.org/docs/current/static/ddl-constraints.html>

思考: 一张基本表可以建多少个UNIQUE约束, PRIMARY KEY约束?

数据定义举例

- 创建Students关系

```
CREATE TABLE Students (  
    sid    CHAR(10)    PRIMARY KEY,  
    name  VARCHAR(20) NOT NULL,  
    age   INT          CHECK(age > 0));
```

- 创建Enrolled关系

```
CREATE TABLE Enrolled (  
    student_id CHAR(10) REFERENCES Students(sid),  
    cid        CHAR(20),  
    grade      INT,  
    PRIMARY KEY(student_id, cid));
```

注意：属性级完整性
约束与表级完整性约
束的区别

数据定义举例

- 创建Enrolled关系

```
CREATE TABLE Enrolled (  
    student_id CHAR(10),  
    cid        CHAR(20),  
    grade      INT,  
    CONSTRAINT pk_En PRIMARY  
KEY(student_id, cid),  
    CONSTRAINT fk_En FOREIGN KEY  
(student_id) REFERENCES Students(sid));
```

- 多属性外键/外码(foreign key)

```
foreign key (b, c) references other_table (c1, c2)
```

修改和删除表

- 修改表

ALTER TABLE <表名>

[ADD <新列名> <数据类型> [完整性约束]]

[DROP <完整性约束名>]

[MODIFY <列名> <数据类型>];

- 删除表

DROP TABLE <表名>

- 基本表删除，则数据、表上的索引都删除
- 表上的视图往往仍然保留，但无法引用

<https://www.postgresql.org/docs/current/static/ddl-alter.html>

数据定义举例

- 如何增加属性、修改属性类型、删除属性？

ALTER TABLE Students **ADD** Scome DATE;

ALTER TABLE Students **ALTER COLUMN** Scome
type timestamp;

ALTER TABLE Students **DROP** Scome;

- 如何更改列和表的约束条件？

ALTER TABLE Enrolled **ADD CONSTRAINT**
grade_check CHECK(grade >= 0 and grade <= 100);

ALTER TABLE Enrolled **DROP CONSTRAINT**
pk_En;

- 如何删除表？

DROP TABLE Students;

第三章 关系数据库标准语言

- 3.1 SQL概述
- 3.2 数据定义
- 3.3 数据更新
- 3.4 数据查询

参考教材：
数据库系统概念 3.1-3.10, 4.1, 4.4

数据插入

- 语句格式

INSERT

INTO <表名> [(<属性列1>[, <属性列2 >] ...)]

VALUES (<常量1> [, <常量2>] ...)

- 将新元组插入指定表中

Insert into Students Values('200011', '张三', 19);

Insert into Students(sid, age, name)

Values('200012', 20, '李四');

Insert into Students(sid, name)

Values('200013', '王五');

注意：中英文标点符号，
单引号和双引号

数据插入

- 在INTO子句中只指出了表名，没有指出属性名，这表示新元组要在表的所有列上都指定值，列的次序同CREATE TABLE中的次序
- 如果数据违反完整性约束？

完整性约束保证
的数据一致性

```
Insert into Students Values('200011', '刘晓', 19);
```

```
Insert into Students Values('200014', NULL, 19);
```

```
Insert into Students Values('200014', '刘晓', 0);
```

数据修改

- 语句格式

UPDATE <表名>

SET <列名>=<表达式>[, <列名>=<表达式>]...

[WHERE <条件>];

- 修改指定表中满足WHERE子句条件的元组

Update Students Set age = 18 where sid = '200011'

Update Students Set age = 18 where name = '王五'

Update Students Set age = age + 1;

Update Students Set sid = '200012' where sid = '200011';

思考：上面第二句将会修改多少元组/行记录？如何避免这类操作？

数据删除

- 语句格式

DELETE

FROM <表名>

[WHERE <条件>];

注意：Delete from Students删除了Students的所有元组，但保留了表的结构、属性和索引

- 删除指定表中满足WHERE子句条件的元组

- WHERE子句

- 指定要删除的元组
- 缺省表示要修改表中的所有元组

Delete From Students where sid = '200011';

Delete From Students where sid = '200000';

Delete From Students;

参照完整性

- Enrolled的属性sid参照关系Students的属性sid, Enrolled的属性cid参照关系Courses的属性cid
- Insert into Enrolled Values(201, 308, NULL);
- Update Enrolled Set cid = 405;
- Update Students Set sid = 102 where sid = 123;
- Delete Students where sid = 123;

sid	Name	GPA
101	Bob	3.2
123	Mary	3.8

Students

sid	cid	Grade
123	564	A

Enrolled

cid	cname	credits
564	564-2	4
308	417	2

Courses

参照完整性

- 关系R的属性A参照关系S的属性B，可能违反参照完整性的修改：
 - Insert into R
 - Delete from S
 - Update R.A
 - Update S.B
- 修改后的操作 思考：哪类数据库用户决定哪类操作？
 - Insert into R, 属性A不在关系S的属性B中
 - Insert is rejected (foreign keys are constraints)!
 - Delete from S / Update S.B
 - Restrict (default): Disallow the delete
 - Cascade: Remove all of the courses for that student
 - Set NULL: SQL allows a third via NULL

参照完整性

- 不同数据库系统在外码实现上的差异
 - SQLite: Everything works after setting PRAGMA foreign_keys = ON;
 - MySQL:
 - Requires varchar type for keys
 - Requires foreign key declarations separate from attributes
 - Requires InnoDB storage engine, Otherwise accepts constraints but does not enforce them
 - PostgreSQL: Everything works
 - SQL Server: Set Null does not work, Cascade does not have a loop 思考：当删除已有同学选课的课程，数据库会怎么处理？

```
Create Table Enrolled (  
    student_id char(10) references Students(sid) on delete restrict,  
    cid          char(20) references Courses(cid) on update cascade,  
    .....);
```

参照完整性

- create table T (A int, B int, C int, primary key (A,B), foreign key (B,C) references T(A,B) on delete cascade);
 - insert into T values (1,1,1);
 - insert into T values (2,1,1);
 - insert into T values (3,2,1);
 - insert into T values (4,3,2);
 - insert into T values (5,4,3);
 - insert into T values (6,5,4);
 - insert into T values (7,6,5);
 - insert into T values (8,7,6);
 - delete from T where A=1;
 - select * from T; 结果是什么? [PostgreSQL, MySQL]

第三章 关系数据库标准语言

- 3.1 SQL概述
- 3.2 数据定义
- 3.3 数据更新
- 3.4 数据查询
 - 3.4.1 The basic SELECT statement
 - 3.4.2 Using Table and Attribute Variables
 - 3.4.3 Set Operators in SQL
 - 3.4.4 Subqueries in the WHERE clause
 - 3.4.5 Subqueries in the FROM and SELECT
 - 3.4.6 The Join Operators
 - 3.4.7 Aggregation
 - 3.4.8 NULL values

参考教材：
数据库系统概念 3.1-3.10, 4.1, 4.4

3.4.1 The basic SELECT statement

- 语句格式

SELECT A_1, A_2, \dots, A_n #3: what to return

FROM R_1, R_2, \dots, R_n #1: relations to query

WHERE condition #2: combine, filter relations

condition使用and, or, not组合多个条件,
每个条件可以是 $A >, >=, <, <=, !=, <> B$

- **SELECT * FROM ...** = Select all attributes
- How would you translate this into relational algebra?

$$\pi_{A_1, A_2, \dots, A_n}(\sigma_{\text{condition}}(R_1 \times R_2 \times \dots \times R_n))$$

- The result of a SELECT query is also a relation
 - SQL is a **compositional** language

3.4.1 The basic SELECT statement

- 语句格式

SELECT A_1, A_2, \dots, A_n #3: what to return

FROM R_1, R_2, \dots, R_n #1: relations to query

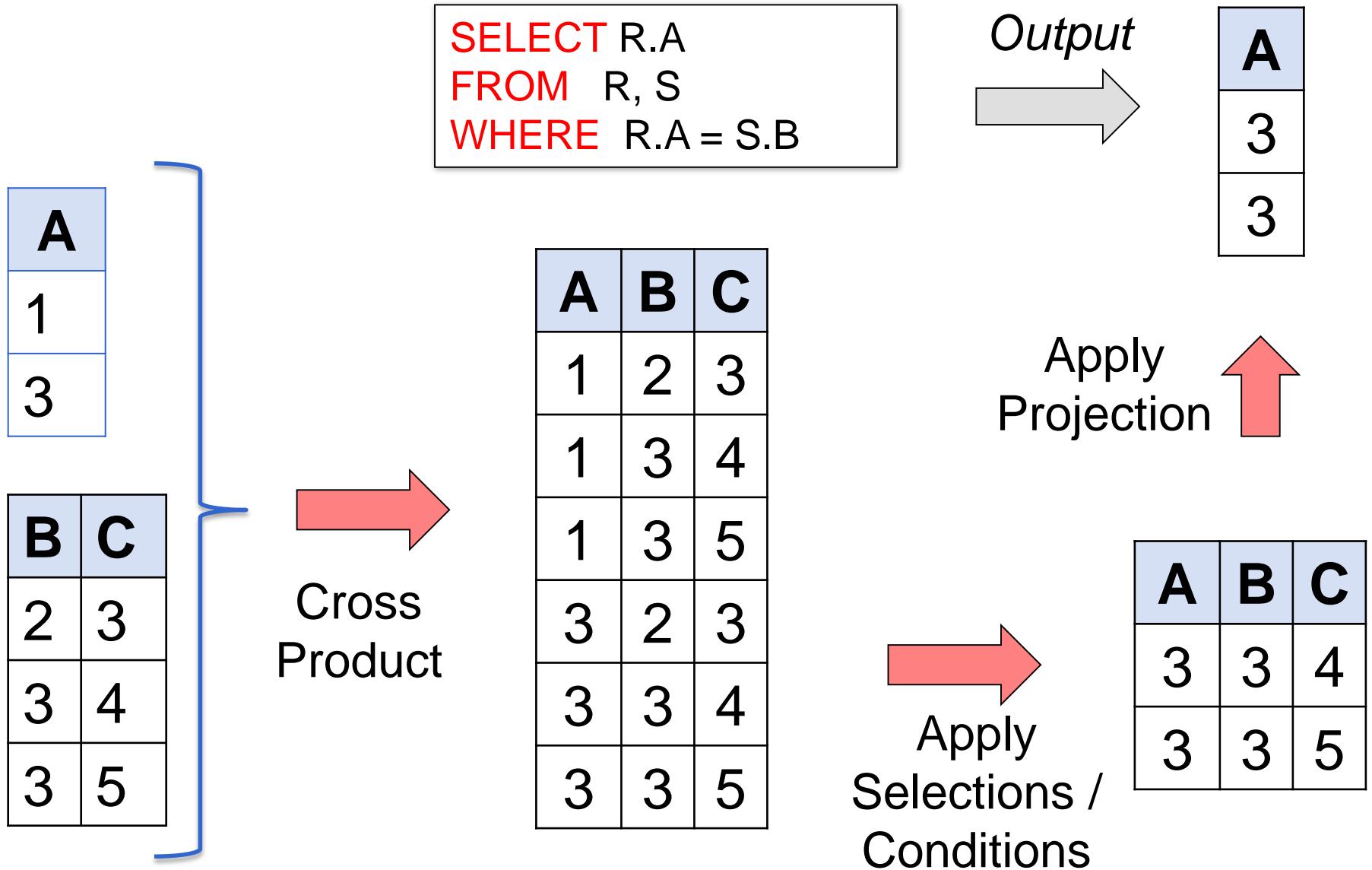
WHERE condition #2: combine, filter relations

- C语言 (顺序很重要, Multiset Union)

```
Answer = {}  
for  $x_1$  in  $R_1$  do  
  for  $x_2$  in  $R_2$  do  
    .....  
    for  $x_n$  in  $R_n$  do  
      if conditions( $x_1, \dots, x_n$ )  
        then Answer = Answer  $\cup$   $\{(x_1.a_1, x_1.a_2, \dots, x_n.a_n)\}$   
return Answer
```

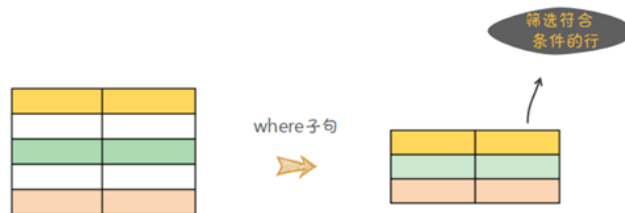
思考：上述只是语义上的转换，帮助理解。DBMS的执行顺序取决哪些因素？

3.4.1 The basic SELECT statement



3.4.1 The basic SELECT statement

- For all SQL examples, we will be using a simple college admissions database
 - College(cName, state, enrollment)
 - Student(sID, sName, GPA, sizeHS)
 - Apply(sID, cName, major, decision)
- Note: The underlined attributes designate a key for that relation - the values for those attributes must be unique across all rows of that table!



3.4.2 Table and Attribute Variables

SELECT A_1, A_2, \dots, A_n

FROM R_1, R_2, \dots, R_n

WHERE condition

Ambiguity in
Joining Multi-Table

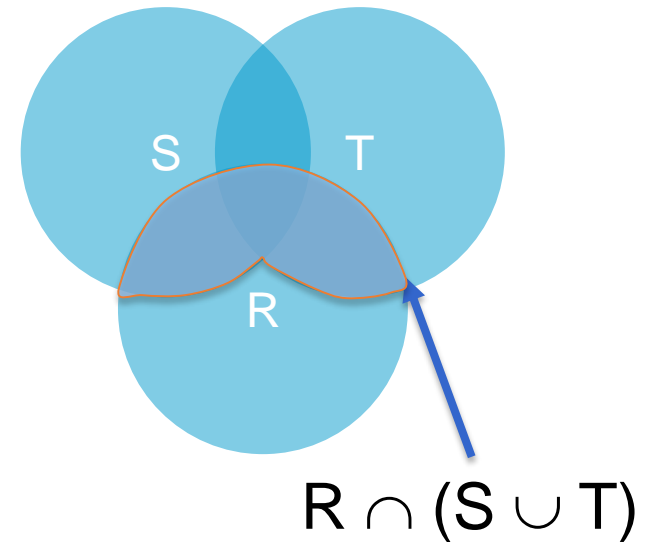
- What if attributes have the same name (e.g. $A_1 == A_2$)?
 - $R_1.A_1, R_2.A_2$
- What if we want to query from the same relation multiple times (e.g. $R_1 == R_2$)?
 - Can rename relations (and attributes!) using the “as” keyword
 - Same as the ρ operator in relational algebra

3.4.2 Table and Attribute Variables

- What does it compute?

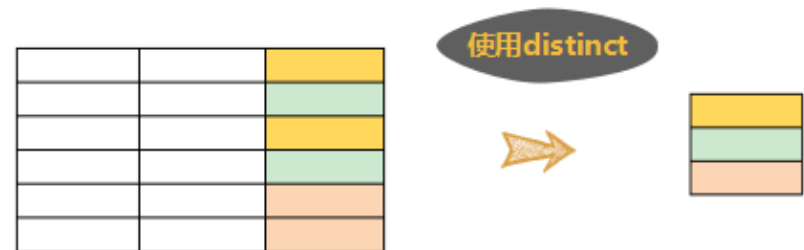
```
SELECT DISTINCT R.A  
FROM R, S, T  
WHERE R.A=S.A OR R.A=T.A
```

- But what if $S = \emptyset$?
- Recall the semantics!
 - Take cross-product
 - Apply selections / conditions
 - Apply projection
- If $S = \{\}$, then the cross product of $R, S, T = \{\}$, and the query result = $\{\}$!



3.4.3 Set Operators in SQL

Relational Algebra	SQL
\cup	union
\cap	intersect
$-$	except



- union eliminates duplicates – to preserve duplicates, use “union all”
- Projection does not eliminates duplicates – to eliminate duplicates, use distinct
- intersect and except are supported in SQLite and PostgreSQL, but not MySQL

3.4.3 Set Operators in SQL

- In-Class Exercise: Write a SQL query that returns the IDs of students who applied to CS but not EE

```
SELECT sID FROM Apply WHERE major = `CS`  
except  
SELECT sID FROM Apply WHERE major = `EE`;
```

- Follow-up question: MySQL doesn't support the except keyword - can this query be rewritten to work in MySQL?

3.4.4 Subqueries in the WHERE clause

SELECT A_1, A_2, \dots, A_n

FROM R_1, R_2, \dots, R_n

WHERE condition

← nested SELECT statements

- 4 operators (can be inverted using not):

- in
- exists
- all
- any

s in R
exists R
s > all R
s < any R

- SQL is compositional (extremely powerful)
 - Everything (inputs / outputs) is represented as multisets—the output of one query can thus be used as the input to another (nesting)!

3.4.4 Subqueries in the WHERE clause

- In-Class Exercise: Write a SQL query that returns the IDs of students who applied to CS but not EE

```
SELECT sID FROM Apply WHERE major = `CS`
```

```
except
```

```
SELECT sID FROM Apply WHERE major = `EE`;
```

- But we couldn't (yet) rewrite this without the except keyword! Now, can we do it?

```
SELECT sID FROM Student
```

```
WHERE sID in (SELECT sID FROM Apply WHERE major = `CS`)
```

```
and sID not in (SELECT sID FROM Apply WHERE major = `EE`);
```

```
SELECT distinct sID FROM Apply A1 WHERE major = 'CS' and
```

```
not exists (SELECT * FROM Apply A2 WHERE A1.sID = A2.sID  
and major = 'EE');
```

3.4.4 Subqueries in the WHERE clause

SELECT A_1, A_2, \dots, A_n

FROM R_1, R_2, \dots, R_n

WHERE condition  nested SELECT statements

- Support for correlated references: You can also refer to relations listed outside of the subquery. For example

SELECT A1

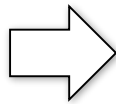
FROM R1

WHERE A1 in (**SELECT** A2 **FROM** R2 **WHERE** R2.A2 = R1.A1);

3.4.4 Subqueries in the WHERE clause

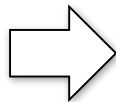
- Nested queries as alternatives to INTERSECT and EXCEPT (MySQL不支持Intersect和Except)

```
(SELECT R.A, R.B  
FROM R)  
INTERSECT  
(SELECT S.A, S.B  
FROM S)
```



```
SELECT R.A, R.B  
FROM R  
WHERE EXISTS (  
    SELECT *  
    FROM S  
    WHERE R.A=S.A AND R.B=S.B)
```

```
(SELECT R.A, R.B  
FROM R)  
EXCEPT  
(SELECT S.A, S.B  
FROM S)
```



```
SELECT R.A, R.B  
FROM R  
WHERE NOT EXISTS (  
    SELECT *  
    FROM S  
    WHERE R.A=S.A AND R.B=S.B)
```

思考：如果R和S没有重复元组，不是子查询，如何实现上述功能？

3.4.5 Subqueries in the FROM and SELECT clauses

- We can also insert subqueries in the SELECT and FROM clauses, too!

SELECT A_1, A_2, \dots, A_n ← nested SELECT statements
FROM R_1, R_2, \dots, R_n ← nested SELECT statements
WHERE condition ← nested SELECT statements

- Be careful! If a subquery is used in the SELECT clause, it must only return a **single** row as its result!
 - Seriously - be careful!! This will throw an error in PostgreSQL and MySQL, but it will "work" in SQLite!!!

SELECT A,
(**SELECT** C **FROM** S **WHERE** B = A)
From R;

A	B	C	A	C
3	3	4	3	4
	3	5		5

3.4.5 Subqueries in the FROM and SELECT clauses

- Max/Min value problem
 - Write a SQL query that returns the IDs of students who have the maximum GPA
 - Student(sID, sName, GPA, sizeHS)
 - Solution 0

```
SELECT sID FROM Student ORDER BY GPA DESC LIMIT 1;
```

- Solution 1 (all/any?)

```
SELECT sID FROM Student  
WHERE GPA >= all (SELECT GPA FROM Student);
```

- Solution 2

```
SELECT sID FROM Student  
WHERE GPA = (SELECT max(GPA) FROM Student);
```

3.4.5 Subqueries in the FROM and SELECT clauses


- Max/Min value problem
 - Write a SQL query that returns the IDs of students who have the maximum GPA
 - Student(sID, sName, GPA, sizeHS)
 - Solution 2

```
SELECT sID FROM Student
WHERE GPA = (SELECT max(GPA) FROM Student);
```

- Solution 3

```
SELECT sID FROM Student,
  (SELECT max(GPA) as maxGPA FROM Student) as T
WHERE GPA = maxGPA;
```




3.4.6 The Join Operators

SELECT A_1, A_2, \dots, A_n
FROM R_1, R_2, \dots, R_n  explicitly Join tables
WHERE condition

- Instead of taking the cross product, you can also find the natural join or theta-join of two relations, just like in relational algebra

思考：通过判断属性是否相同等，关联不同关系中的记录，扩展到空间关联？

3.4.6 The Join Operators

- Inner Join On condition
 - Same as  条件
- Natural Join
 - Same as 
- Inner Join Using (attributes)
 - Same as  , but common attributes to join on must be explicitly specified
- (Left | Right | Full) Outer Join
 - If join condition don't match for certain tuples, include those tuples in the result, but pad with NULL values
- These operators don't add any additional expressive power to SQL

3.4.6 The Join Operators

- By default, Joins in SQL are “Inner Joins”
select R.A, S.B from R, S where R.A = S.A
select R.A, S.B from R **join** S **on** R.A = S.A
- Outer Join
 - If there is an entry in R with A=3, but none in S with A=3
 - A LEFT OUTER JOIN will return a tuple (a, NULL)!select R.A, S.B from R **left outer join** S **on** R.A = S.A

R.A	R.B
1	Cat
2	Dog
3	Dog

S.A	S.B
1	Apple
2	Bana
2	Pear

R.A	S.B
1	Apple
2	Bana
2	Pear

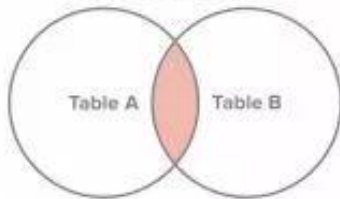
R.A	S.B
1	Apple
2	Bana
2	Pear
3	NULL

数据库常用None表示NULL

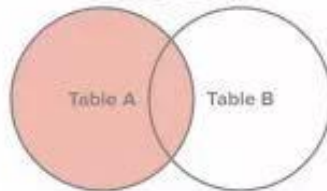
3.4.6 The Join Operators

连接查询示意图

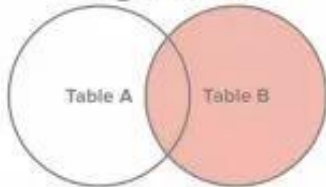
Inner Join



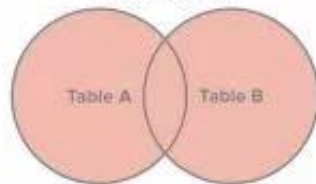
Left Join



Right Join



Full Join



交叉连接: cross join (不常用)

1
2
3

cross
join

A
B
C

=

1	A
1	B
1	C
2	A
2	B
2	C
3	A
3	B
3	C

左表和右表
两两连接

3.4.6 The Join Operators

交叉连接: cross join (不常用)

1	A
2	B
3	C

cross
join

=

1	A
1	B
1	C
2	A
2	B
2	C
3	A
3	B
3	C

左表和右表
两两连接

全外连接: full join

1	A
2	B
3	C

full
join

=

1	null
2	A
3	B
null	C

返回两张表匹配的
记录,以及左右两表
多余的记录

内连接: inner join

1	A
2	B
3	C

inner
join

=

2	A
3	B

只返回两张表
匹配的记录

相同颜色表示相同条件

左外连接: left join

1	A
2	B
3	C

left
join

=

1	null
2	A
3	B

返回两张表匹配的
记录,以及
左表多余的记录

右外连接: right join

1	A
2	B
3	C

right
join

=

2	A
3	B
null	C

返回两张表匹配的
记录,以及右表
多余的记录

3.4.6 The Join Operators

- In-Class Exercise: Is the Full Outer Join operator associative? Specifically is
`SELECT *`
`FROM (T1 natural full outer join T2) natural full outer join T3;`
equivalent to
`SELECT *`
`FROM T1 natural full outer join (T2 natural full outer join T3);`
- No
 - Try testing this out with three sample tables of your own
 - you should be able to see that they won't necessarily produce the same result!

3.4.7 Aggregation

SELECT A_1, A_2, \dots, A_n  “Aggregation” functions
FROM R_1, R_2, \dots, R_n
WHERE condition

- Aggregation functions compute values over multiple rows of the result, such as
 - min, max, sum, avg, and count
 - NULL is special
- Except **count**, all aggregations apply to a single attribute

SELECT A, max(B)
as C **From** R;

A	B
3	5
4	6

A	C
3	6
4	6

A	C
3	6
4	

语句报错，A有2行，而max(B)将整列(一个组)聚集为1行

3.4.7 Aggregation

SELECT A_1, A_2, \dots, A_n

FROM R_1, R_2, \dots, R_n

WHERE condition

GROUP BY A_i, A_j, \dots, A_k ← Partition rows into “groups”

- The aggregation functions are computed over each “group” independently
 - For example, average GPA for each college, number of applications per student, the maximum enrollment for each particular state
- **Warning:** every column in the SELECT clause must either be
 - Also present in the GROUP BY clause AND/OR
 - Used in an aggregation function

3.4.7 Aggregation

- Semantics
 - 1. Compute the **FROM** and **WHERE** clauses
 - 2. Group by the attributes in the **GROUP BY**
 - 3. Compute the **SELECT** clause: grouped attributes and aggregates
 - Example
 - Find total sales after 10/1/2005 per product
- ```
SELECT product, sum(price * quantity) as totalsales
FROM Purchase
WHERE date > '10/1/2005'
GROUP BY product
```

| Product | Date  | Price | Quantity |
|---------|-------|-------|----------|
| apple   | 10/21 | 1     | 20       |
| banana  | 10/3  | 0.5   | 10       |
| banana  | 10/10 | 1     | 10       |
| apple   | 10/25 | 1.5   | 20       |

## 3.4.7 Aggregation

- 1. Compute the **FROM** and **WHERE** clauses

```
SELECT product, sum(price * quantity) as totalsales
```

```
FROM Purchase
```

```
WHERE date > '10/1/2005'
```

```
GROUP BY product
```

| Product | Date  | Price | Quantity |
|---------|-------|-------|----------|
| apple   | 10/21 | 1     | 20       |
| apple   | 10/25 | 1.5   | 20       |
| banana  | 10/3  | 0.5   | 10       |
| banana  | 10/10 | 1     | 10       |

- 2. Group by the attributes in the **GROUP BY**

```
SELECT product, sum(price * quantity) as totalsales
```

```
FROM Purchase
```

```
WHERE date > '10/1/2005'
```

```
GROUP BY product
```

| Product | Date  | Price | Quantity |
|---------|-------|-------|----------|
| apple   | 10/21 | 1     | 20       |
|         | 10/25 | 1.5   | 20       |
| banana  | 10/3  | 0.5   | 10       |
|         | 10/10 | 1     | 10       |

## 3.4.7 Aggregation

- 3. Compute the **SELECT** clause: grouped attributes and aggregates

**SELECT** product, sum(price \* quantity) as totalsales

FROM Purchase

WHERE date > '10/1/2005'

GROUP BY product

| Product | Date  | Price | Quantity |
|---------|-------|-------|----------|
| apple   | 10/21 | 1     | 20       |
|         | 10/25 | 1.5   | 20       |
| banana  | 10/3  | 0.5   | 10       |
|         | 10/10 | 1     | 10       |

| Product | TotalSales |
|---------|------------|
| apple   | 50         |
| banana  | 15         |

**SELECT** product, date, price, quantity

**FROM** Purchase

**WHERE** date > '10/1/2005'

**GROUP BY** product



## 3.4.7 Aggregation

---

**SELECT**  $A_1, A_2, \dots, A_n$

**FROM**  $R_1, R_2, \dots, R_n$

**WHERE** condition

**GROUP BY**  $A_i, A_j, \dots, A_k$

**HAVING** condition  Filter aggregate results

- The **WHERE** conditions apply to **single rows** at a time
- The **HAVING** conditions apply to **the groups** generated by the GROUP BY clause
- **Warning**: Don't use HAVING without GROUP BY!

## 3.4.7 Aggregation

---

**SELECT**       $A_1, A_2, \dots, A_n$   
**FROM**         $R_1, R_2, \dots, R_n$   
**WHERE**         $C_1$   
**GROUP BY**    $A_i, A_j, \dots, A_k$   
**HAVING**        $C_2$

- $A_1, A_2, \dots, A_n$  = Can **ONLY** contain attributes  $A_i, \dots, A_k$  and/or aggregates over other attributes
- $C_1$  = is any condition on the attributes in  $R_1, \dots, R_n$
- $C_2$  = is any condition on the aggregate expressions

## 3.4.7 Aggregation

---

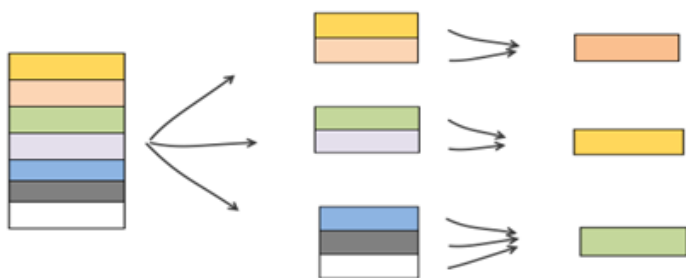
**SELECT**       $A_1, A_2, \dots, A_n$   
**FROM**         $R_1, R_2, \dots, R_n$   
**WHERE**         $C_1$   
**GROUP BY**    $A_i, A_j, \dots, A_k$   
**HAVING**        $C_2$

- Evaluation steps:
  - Evaluate **FROM-WHERE**: apply condition  $C_1$  on the attributes in  $R_1, \dots, R_n$
  - **GROUP BY** the attributes  $A_i, \dots, A_k$
  - Apply condition  $C_2$  to each group (may have aggregates)
  - Compute aggregates in  $A_1, A_2, \dots, A_n$  and return the result

# 3.4.7 Aggregation

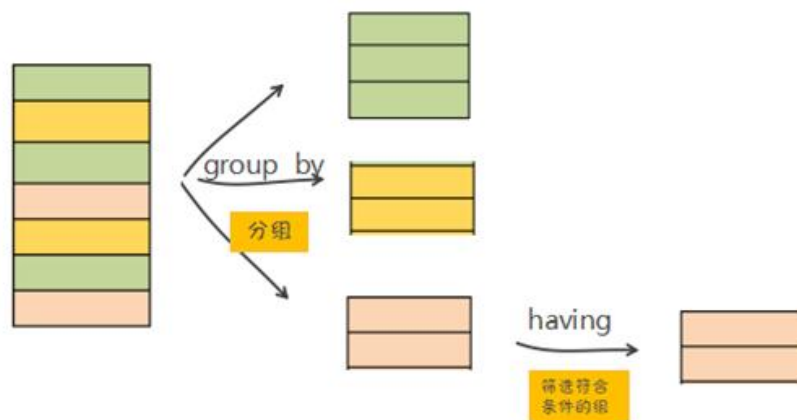
先分组

后聚合



分组聚合实例

(group by + having) 常常一起使用



初始表

分组结果

聚合结果

| 学号 | 成绩  |
|----|-----|
| 1  | 90  |
| 1  | 100 |
| 1  | 89  |
| 2  | 56  |
| 2  | 79  |
| 3  | 89  |
| 3  | 56  |
| 3  | 78  |
| 4  | 49  |
| 4  | 99  |

group by



分组

|   |     |
|---|-----|
| 1 | 90  |
| 1 | 100 |
| 1 | 89  |
| 2 | 56  |
| 2 | 79  |
| 3 | 89  |
| 3 | 56  |
| 3 | 78  |
| 4 | 49  |
| 4 | 99  |

max(成绩)



聚合

| 学号 | 成绩  |
|----|-----|
| 1  | 100 |
| 2  | 79  |
| 3  | 89  |
| 4  | 99  |

## 3.4.7 Aggregation

---

- Group by vs. Nested Query
  - Author(login, name), Wrote(login, url)
  - Question: Find authors who  $\geq 10$  documents
  - Solution 1: with nested queries

```
SELECT DISTINCT Author.name
```

```
FROM Author
```

```
WHERE count(
```

```
 SELECT Wrote.url
```

```
 FROM Wrote
```

```
 WHERE Author.login = Wrote.login) >= 10
```

## 3.4.7 Aggregation

---

- Group by vs. Nested Query
  - Author(login, name), Wrote(login, url)
  - Question: Find authors who  $\geq 10$  documents
  - Solution 1: with nested queries
  - Solution 2: SQL style (with GROUP BY)

```
SELECT Author.name
FROM Author, Wrote
WHERE Author.login = Wrote.login
GROUP BY Author.name
HAVING count(Wrote.url) >= 10
```

注意：可能存在多种SQL查询方法，选择查询效率最高，使用Group By效率更高

## 3.4.7 Aggregation

---

- Group by vs. Nested Query
  - Author(login, name), Wrote(login, url)
  - Question 2: Find authors and their login number
  - Solution 1: with group by

**SELECT** Author.name, count(\*)

**FROM** Author, Wrote

**WHERE** Author.login = Wrote.login

**GROUP BY** Author.name

## 3.4.7 Aggregation

---

- Group by vs. Nested Query
  - Author(login, name), Wrote(login, url)
  - Question 2: Find authors and their login number
  - Solution 1: with group by
  - Solution 2: with nested queries

```
SELECT name, (select count(*) from Wrote where
Wrote.login = Author.login)
FROM Author
```

注意：可能存在多种SQL查询方法，选择查询效率最高，使用Group By效率更高

## 3.4.7 Aggregation

---

- In-Class Exercise: Write a SQL query that returns the number of colleges applied to by each student, including 0 for those who applied nowhere

```
SELECT Student.sID, count(distinct cName)
FROM Student, Apply
WHERE Student.sID = Apply.sID
GROUP BY Student.sID
union
SELECT sID, 0
FROM Student
WHERE sID NOT IN (SELECT sID FROM Apply);
```

思考：如何通过outer join 简化SQL语句？

## 3.4.7 Aggregation

---

- Max/Min value problem in aggregation
  - Write a SQL query that returns the name of colleges who have the maximum number of applications

- Apply(sID, cName, major, decision)

- Solution

```
SELECT CName
```

```
FROM Apply
```

```
GROUP BY CName
```

```
HAVING count(*) >= ALL
```

```
(SELECT count(*) FROM Apply GROUP BY CName);
```

- Follow-up question: Write a SQL query that returns the name of colleges who have the maximum number of applicants

思考：最大/小值问题有4种方法，分组最大/小值问题还有其他查询方法？

## 3.4.8 NULL values

---

- We use NULL to represent "unknown" or "undefined" values in our database - the semantic meaning of NULL can vary from situation to situation
- For example, you could use NULL to represent:
  - A student with no middle name
  - A credit card that doesn't expire
  - A car that hasn't been given a license plate yet
  - Or whatever you want it to be!

## 3.4.8 NULL values

---

- *For numerical operations, NULL -> NULL:*
  - If  $x = \text{NULL}$  then  $4 \cdot (3 - x) / 7$  is still NULL
- *For boolean operations, in SQL there are three values:*

**FALSE**            = 0

**UNKNOWN**      = 0.5

**TRUE**            = 1

- If  $x = \text{NULL}$  then  $x = \text{"Joe"}$  is UNKNOWN
- $C1 \text{ AND } C2 = \min(C1, C2)$
- $C1 \text{ OR } C2 = \max(C1, C2)$
- $\text{NOT } C1 = 1 - C1$

## 3.4.8 NULL values

---

- Will return (1, NULL) in R(A, B)?  
`select * from R where B > 1`
- Rule in Selection SQL
  - Include only tuples that yield TRUE / 1.0
- Rule in Insert SQL
  - Exclude only tuples that yield FALSE / 0.0
- Can test for NULL explicitly:
  - `x IS NULL`
  - `x IS NOT NULL`

## 3.4.8 NULL values

---

- NULL values won't necessarily be captured by the appropriate conditions in the WHERE clause. For example, suppose we have the following query:

```
SELECT * FROM Student WHERE GPA >= 3.5 or GPA < 3.5;
```

Will this return every student?

- No! There may be student who have NULL as their GPA! Instead, the query should be

```
SELECT * FROM Student WHERE GPA >= 3.5 or GPA < 3.5
or GPA is NULL;
```

## 3.4.8 NULL values

---

- NULL参与的数值或布尔运算，结果都是NULL
- WHERE子句只有条件为true才保留这个记录
- HAVING子句只有条件为true才保留这个GROUP
- JOIN NULL != NULL
- GROUP BY NULL算一个GROUP
- NULL在ORDER BY时默认排序最前面，有语法可以改变顺序
- 对于AGGREGATE函数
  - 如果输入空集，COUNT返回0，其他任何函数返回NULL
  - 如果COUNT(\*), NULL的记录参与计算，COUNT属性，NULL的记录忽略
  - 其他AGGREGATE函数，忽略NULL

# Data modification

---

- Inserting new data

- **INSERT INTO** Table  
**VALUES** ( $A_1, A_2, \dots, A_n$ )
- **INSERT INTO** Table  
**SELECT** statement

- Deleting data

- **DELETE FROM** Table  
**WHERE** condition

- Updating existing data

- **UPDATE** Table  
**SET**  $A = \text{Expr}_1, \dots, A_n = \text{Expr}_n$   
**WHERE** condition

Expr can a **SELECT** statement  
That return a single value

# Other Keywords

- Distinct

- Eliminates duplicates

- Order by  $A_1, A_2, \dots, A_n$

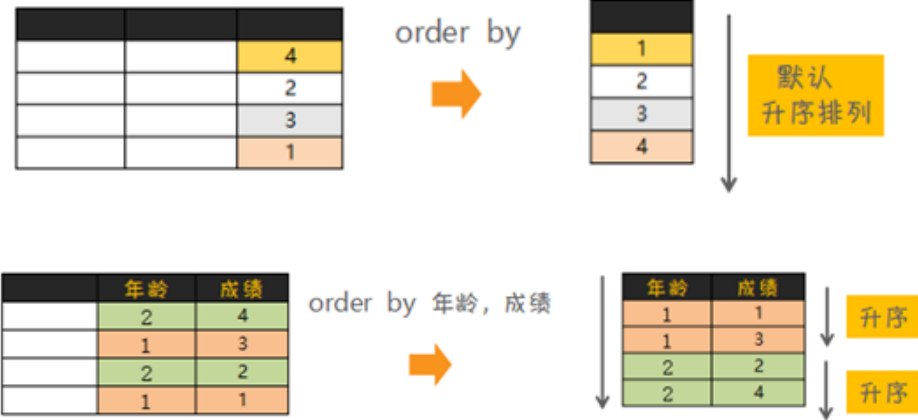
- asc/desc
- Default is asc

- Text comparison

- s **LIKE** p: pattern matching on strings
- % = any sequence of characters
- \_ = any single character

- Between... and ...

- With子句, 仅在当前事务中能使用的查询表



```
SELECT *
FROM World
WHERE "Someone"
LIKE "%You%"
```

# Multiset

- $\lambda(X)$  = “Count of tuple in  $X$ ”
  - Items not listed have implicit count 0

**Multiset X**

| Tuple  |
|--------|
| (1, a) |
| (1, a) |
| (1, b) |
| (2, c) |
| (2, c) |
| (2, c) |
| (1, d) |
| (1, d) |



Equivalent  
Representations  
of a **Multiset**

**Multiset X**

| Tuple  | $\lambda(X)$ |
|--------|--------------|
| (1, a) | 2            |
| (1, b) | 1            |
| (2, c) | 3            |
| (1, d) | 2            |

*Note: In a set all counts are  $\{0, 1\}$ .*

# Multiset

**Multiset X**

| Tuple  | $\lambda(X)$ |
|--------|--------------|
| (1, a) | 2            |
| (1, b) | 0            |
| (2, c) | 3            |
| (1, d) | 0            |

$\cap$

**Multiset Y**

| Tuple  | $\lambda(Y)$ |
|--------|--------------|
| (1, a) | 5            |
| (1, b) | 1            |
| (2, c) | 2            |
| (1, d) | 2            |

$=$

**Multiset Z**

| Tuple  | $\lambda(Z)$ |
|--------|--------------|
| (1, a) | 2            |
| (1, b) | 0            |
| (2, c) | 2            |
| (1, d) | 0            |

$$\lambda(Z) = \min(\lambda(X), \lambda(Y))$$

For sets, this is  
**intersection**

# Multiset

**Multiset X**

| Tuple  | $\lambda(X)$ |
|--------|--------------|
| (1, a) | 2            |
| (1, b) | 0            |
| (2, c) | 3            |
| (1, d) | 0            |

**U**

**Multiset Y**

| Tuple  | $\lambda(Y)$ |
|--------|--------------|
| (1, a) | 5            |
| (1, b) | 1            |
| (2, c) | 2            |
| (1, d) | 2            |

**=**

**Multiset Z**

| Tuple  | $\lambda(Z)$ |
|--------|--------------|
| (1, a) | 7            |
| (1, b) | 1            |
| (2, c) | 5            |
| (1, d) | 2            |

$$\lambda(Z) = \lambda(X) + \lambda(Y)$$

For sets, this is  
**union**

# Multiset

---

- All RA operations need to be defined carefully on bags
  - $\sigma_C(R)$ : preserve the number of occurrences
  - $\Pi_A(R)$ : no duplicate elimination
  - Cross-product, join: no duplicate elimination
- This is important - relational engines work on multisets, not sets!

# 第三章 关系数据库标准语言

- 3.1 SQL概述
- 3.2 数据定义
- 3.3 数据更新
- 3.4 数据查询



1. SQL是一个描述型语言，有很多细节(包括语法和DBMS具体实现)，需要理解其中的细微差别
2. 具体问题进行逻辑等价转换，获得容易使用SQL解决的描述 (逻辑思维很重要)  
Find all companies with products all having price < 100  
Find all companies that make only products with price < 100
3. 遇到复杂问题，可以先查询部分结果，再通过嵌套查询、集合操作等进行组合
4. 每个问题可能存在多种SQL解决方案，选择查询效率较高的解决方案 (nested query vs. group by)
5. 多练习，熟能生巧