数据库系统原理

教程：数据库系统概论（第5版）

结合：CMU 15-445/645 INTRO TO DATABASE SYSTEMS

华中科技大学 计算机学院

左琼

第三章 关系数据库标准语言SQL

Principles of Database Systems

第三章 关系数据库标准语言SQL

3.1 SQL概述

- 三种具有相同表达能力的抽象查询语言：
  - 关系代数　　ISBL
  - 元组关系演算语言　　ALPHA，QUEL
  - 域关系演算语言　　QBE
- SQL(Structured Query Language)则是介于关系代数和关系演算之间的标准查询语言。
  - 由IBM提出，是应用得最广泛的关系数据库标准语言。
  - 与之相比，Ingres的QUEL具有“理论优势”；
  - SQL是一种比关系代数表达式更加自然化的查询需求描述语言。

共同特点：

- 语言具有完备的表达能力；
- 是非过程化的集合操作语言；
- 功能强，能嵌入高级语言中使用。

3.1 概述-SQL发展史

- 1974年，Boyce和Chamberlin提出SEQUEL(STUCTURED ENGLISH QUERY LANGUAGE)；IBM公司对其进行了修改，并用于其SYSTEM R关系数据库系统中。1981年改名SQL；
- 1982年，美国国家标准局ANSI开始制定SQL标准；
- 1986年，ANSI的数据库委员会：SQL语言第一个标准SQL86；
- 1987年，ISO通过了SQL86标准；
- 1989年，ISO对SQL86进行了补充，推出了SQL89标准；
- 1992年，ISO又推出了SQL92标准，也称为SQL2；

DBMS：初级、中级和完全级，后又在初、中级间增加过渡级。

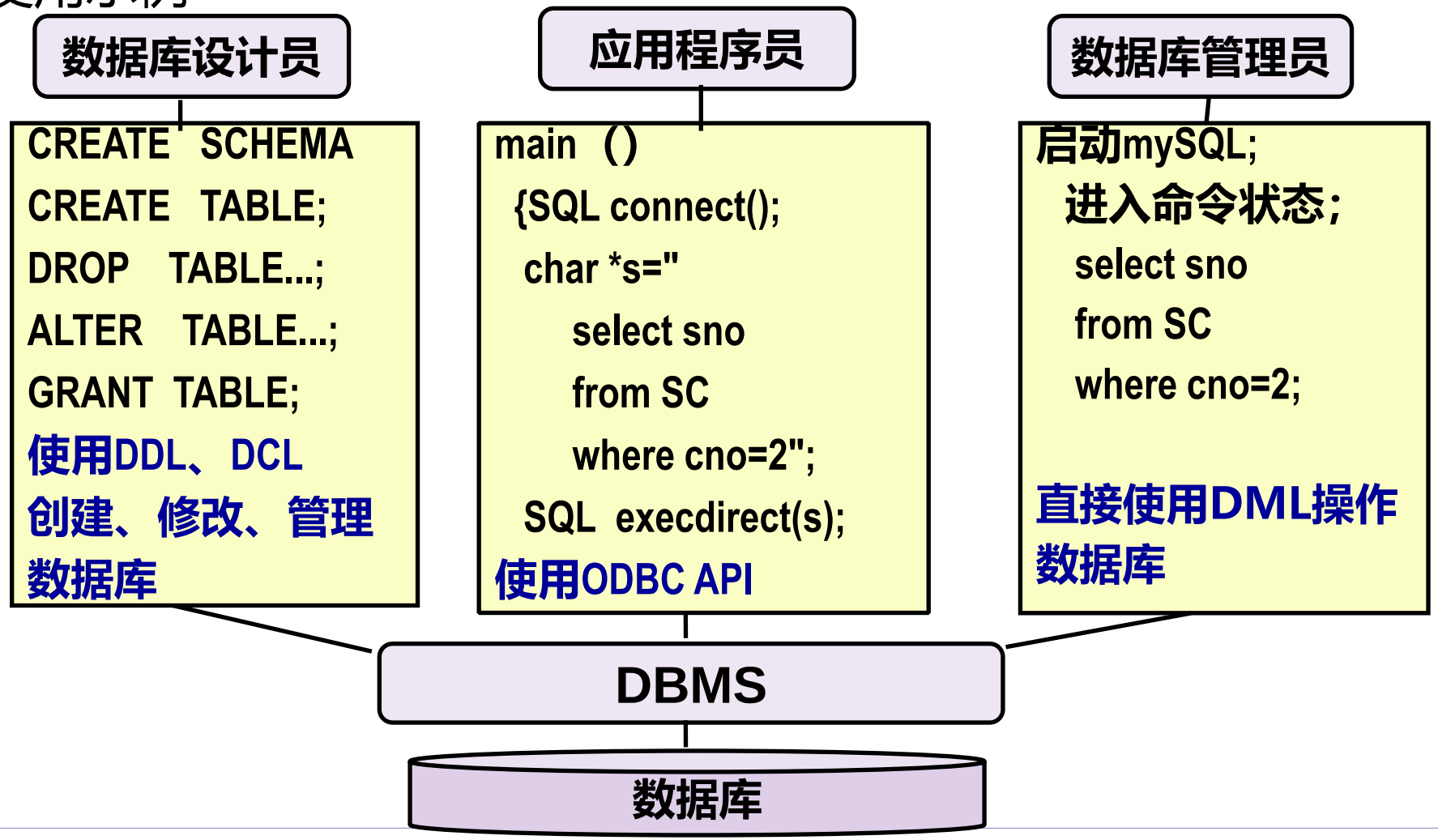
- 1999年，SQL99（即SQL3），分为核心SQL和增强SQL。
- 2003年，SQL2003；SQL2008；2010年，SQL2011。
- SQL标准文本的修改和完善还在继续进行。

其他：

Oracle的PL/SQL语言，SQL Server的T-SQL语言

3.1.2 SQL的特点

- SQL使用示例



3.1.2 SQL的特点

1. 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）功能于一体。
- 可以独立完成数据库生命周期中的全部活动：
  - 定义／修改／删除关系模式，插入数据，建立数据库；
  - 对数据库中的数据进行查询和更新；
  - 数据库重构和维护；
  - 数据库安全性、完整性控制等。
- 用户数据库投入运行后，可根据需要随时逐步修改模式，不影响数据的运行。
- 数据操作符统一（单一的结构——关系，增删改查都只有一种操作符）。

3.1.2 SQL的特点

2. 高度非过程化

- 非关系数据模型的数据操纵语言“面向过程”，必须制定存取路径
- SQL只要提出“What to do”，无须了解存取路径。
- 存取路径的选择以及SQL的操作过程由系统自动完成。

3. 面向集合的操作方式

- 非关系数据模型采用面向记录的操作方式，操作对象是一条记录。
- SQL采用集合（Set-at-a-time）操作方式：
  - 操作对象、查找结果可以是元组的集合
  - 一次插入、删除、更新操作的对象可以是元组的集合

3.1.2 SQL的特点

4.以同一种语法结构提供多种使用方式

- SQL是独立的语言: 能够独立地用于联机交互的使用方式;
- SQL又是嵌入式语言: 能够嵌入到高级语言（例如C，C++，Java）程序中，供程序员设计程序时使用。

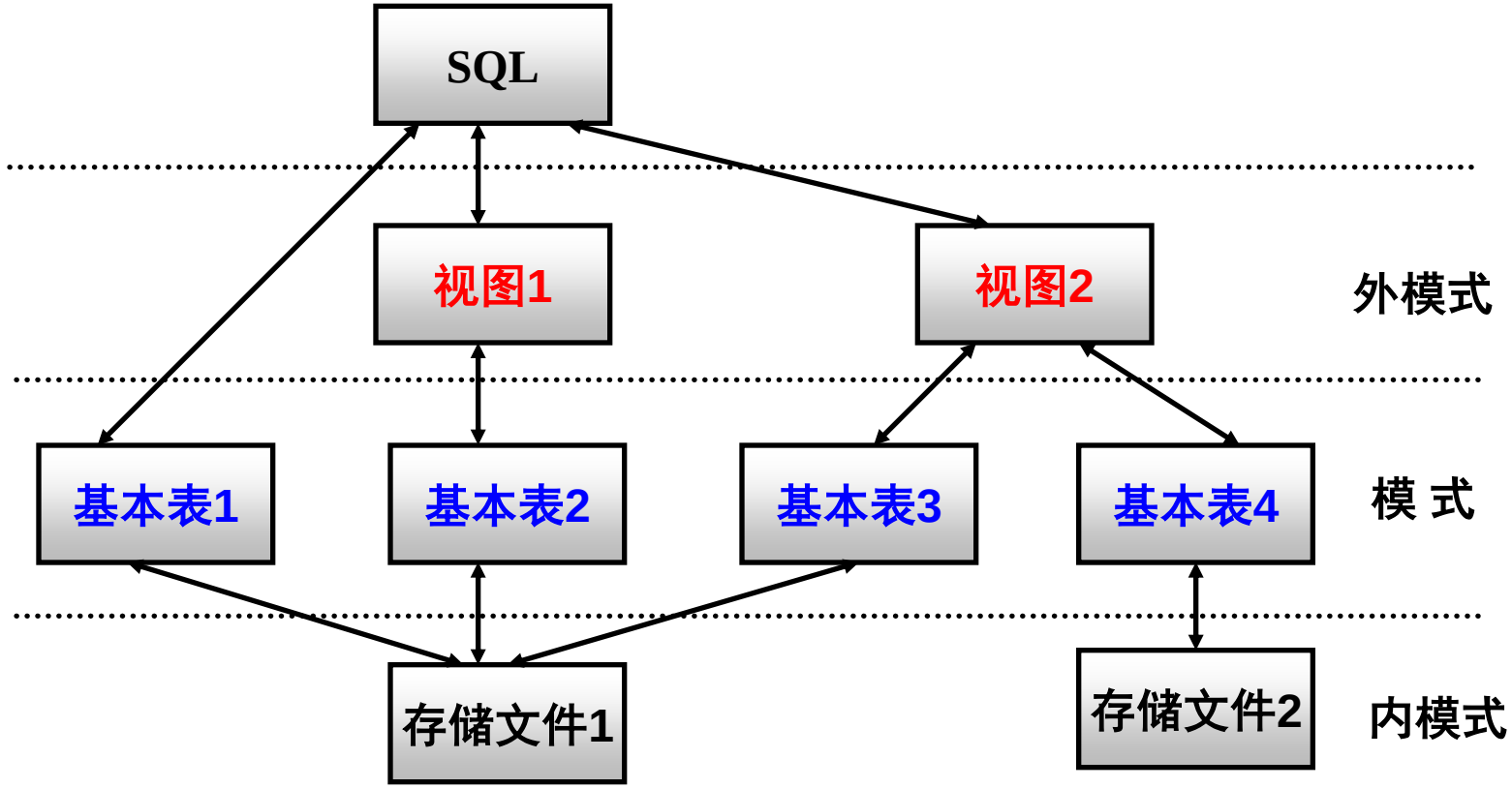
5.语言简洁，易学易用

SQL设计巧妙，核心功能只需9个动词。

- 数据查询（DQL）： select
- 数据定义（DDL）： create, drop, alter
- 数据操纵（DML）： delete, update, insert
- 数据控制（DCL）： grant, revoke

3.1.3 SQL的基本概念

- SQL支持关系数据库三级模式结构：

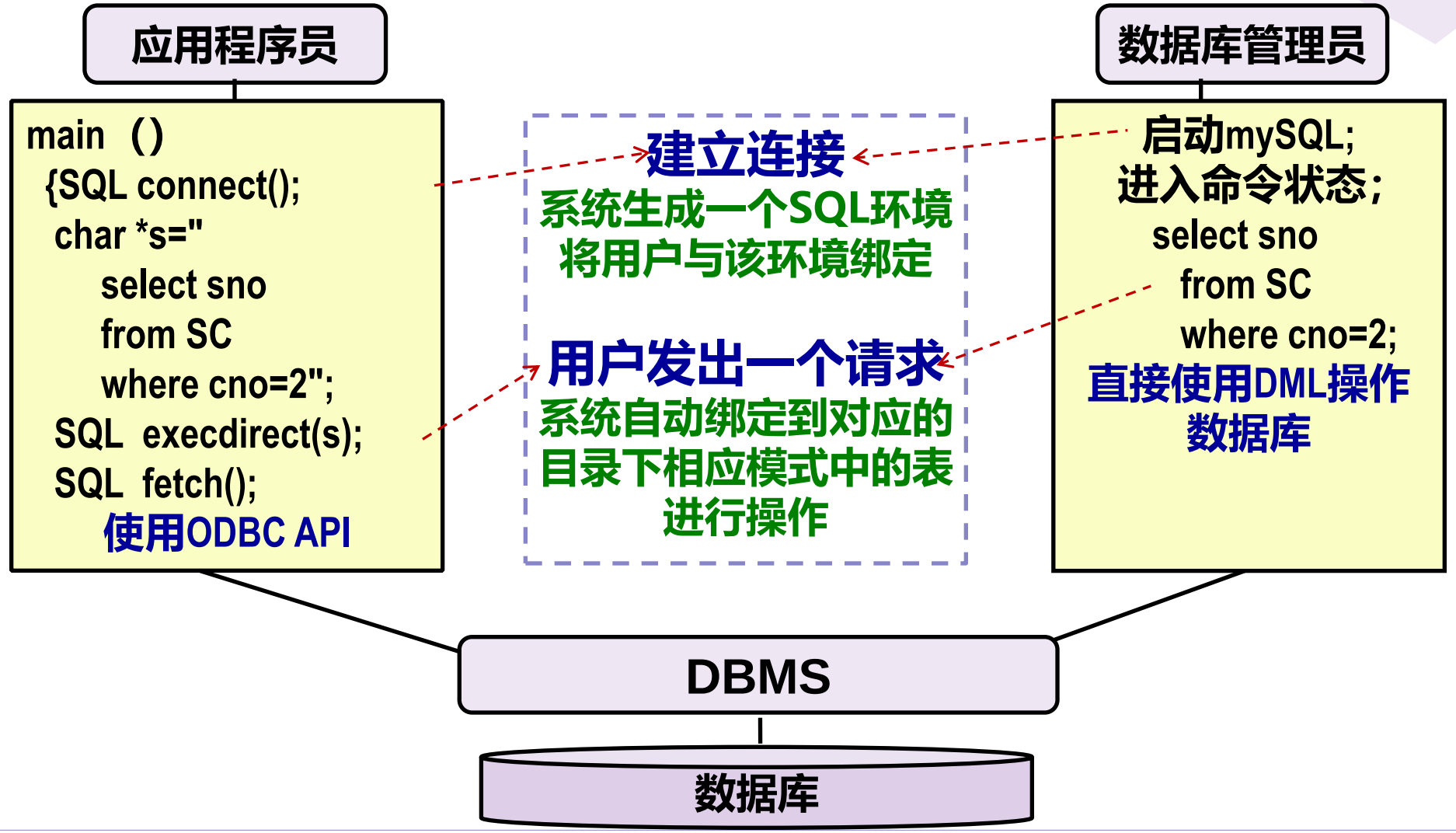


3.1.3 SQL的基本概念

- 基本表（base table）
  - 独立存在的表，SQL中一个关系就对应一个基本表。
  - 一个(或多个)基本表对应一个存储文件；
  - 一个表可以带若干索引，索引也放在存储文件中。
- 存储文件（stored file）
  - 逻辑结构组成了关系数据库的内模式。
  - 物理结构是任意的，对用户透明。
- 视图（view）
  - 从一个或几个基本表导出的表。
  - 数据库中只存放视图的定义而不存放视图对应的数据，视图是一个虚表。
  - 用户可以在视图上再定义视图。

在用户眼中，视图和基本表都是关系，而存储文件对用户是透明的。

SQL使用示例


应用程序员
main （）
{SQL connect();
char *s="
select sno
from SC
where cno=2";
SQL execdirect(s);
SQL fetch();
使用ODBC API
建立连接
系统生成一个SQL环境
将用户与该环境绑定
用户发出一个请求
系统自动绑定到对应的
目录下相应模式中的表
进行操作
数据库管理员
启动mySQL;
进入命令状态；
select sno
from SC
where cno=2;
直接使用DML操作
数据库
DBMS
数据库

SQL使用说明 ·用户/程序如何使用命名空间中的对象?

1）用户（程序）使用数据库时首先必须连接（建立一个会话）

对每个用户，系统自动产生一个目录；

连接时，系统将该用户与其自动目录对应；

用户连接时需提供用户名和密码认证。

2）对应每个连接，系统自动生成一个SQL环境

一个SQL环境是动态的，只与当前用户/程序相关；

系统根据SQL环境，控制当前用户/程序的行为；

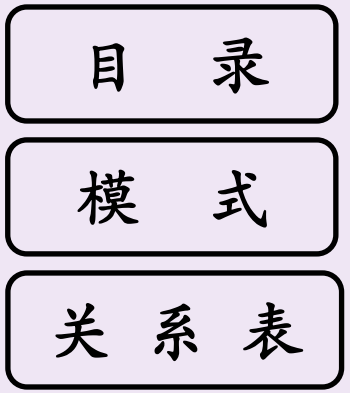
SQL环境中包括当前用户的目录、模式、授权等信息。

3）用户连接后，默认已处于自己的目录，表的定位无需指出目录名和模式名

在一个目录下可以建立多个模式；

在一个模式下可以建立多个关系表。

A schema is a collection of database objects (used by a user).



学生-课程 数据库

- 学生-课程模式 S-T :
  - 学生表： Student(Sno,Sname,Ssex,Sage,Sdept)
  - 课程表： Course(Cno,Cname,Cpno,Ccredit)
  - 学生选课表： SC(Sno,Cno,Grade,Semester, TeachingClass)

学 号 Sno	姓 名 Sname	性 别 Ssex	年 龄 Sage	所在系 Sdept
202315121	李勇	男	20	CS

课程号 Cno	课程名 Cname	先行课 Cpno	学分 Ccredit
1	数据库	5	4

学 号 Sno	课程号 Cno	成绩 Grade	学期 Semester	教学班 TeachingClass
202315121	1	92	202501	CS2301

3.2 数据定义

- 建立数据库时首先要对数据库中数据的结构及特征进行描述，该过程称为数据定义。SQL语言的数据定义功能是通过SQL模式语句来实现的。
- 在SQL语言中，关于数据的描述信息以SQL模式（Schema）的形式组织。每个模式包含0到多个对象，这些对象被称为模式对象，如基表、视图、索引等。
- 用于模式及模式对象的创建、修改和销毁的语句被统称为SQL模式语句。SQL模式语句可分为三类：CREATE语句、DROP语句和ALTER语句。

3.2 数据定义

- SQL的数据定义语言DDL功能: 模式定义、表定义、视图和索引的定义。

表 SQL 的数据定义语句

操作对象	操作方式		
	创 建	删 除	修 改
模式	CREATE SCHEMA	DROP SCHEMA	
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视 图	CREATE VIEW	DROP VIEW	
索 引	CREATE INDEX	DROP INDEX	

DATABASE

PROCEDURE

TRIGGER

FUNCTION

- 层次化数据库对象命名机制:

1个DBMS实例：多个DB;

=> 1个DB中：多个模式;

=> 1个模式下：多个表、视图、索引.....

创建、删除和修改数据库*

1. 创建数据库

一般格式：

```
CREATE DATABASE <数据库名>
 DATAFILE    '<文件路径>'
 SIZE <文件大小>;
```

功能：创建一个新数据库及存储该数据库的文件。

2. 删除数据库

一般格式：

```
DROP DATABASE <数据库名>;
```

功能：删除指定数据库及其包含的所有文件。

3. 修改数据库属性

一般格式：

```
ALTER DATABASE <数据库名>
  ADD DATAFILE    '<文件路径>'
  SIZE <文件大小>;
  |
  MODIFY DATAFILE    '<文件路径>'
  INCREASE <文件增量>;
```

功能：

增大数据库原有文件大小；或在数据库中加入新的数据库文件。

SQL Server创建数据库*

```
CREATE DATABASE database_name
[ ON
    [ < filespec > [ ,...n ] ]
    [ , < filegroup > [ ,...n ] ]
]
 [ LOG ON { < filespec > [ ,...n ] } ]
< filespec > ::=
[ PRIMARY ]
( [ NAME = logical_file_name , ]
   FILENAME = 'os_file_name'
[ , SIZE = size ]
[ , MAXSIZE = { max_size | UNLIMITED } ]
[ , FILEGROWTH = growth_increment ] ) [ ,...n ]
< filegroup > ::=
FILEGROUP filegroup_name < filespec > [ ,...n ]
```

数据库的名称，必须唯一。

指定数据库的数据文件和文件组。其中<filespec>用来定义主文件组的数据文件<filegroup>用来定义用户文件组及其文件。

指定数据库的事务日志文件属性。

指定主文件。一个数据库只能有一个主文件

数据库的逻辑文件名。

数据库的物理文件名及其存储路径。

数据文件的初始大小。

数据文件大小的最大值

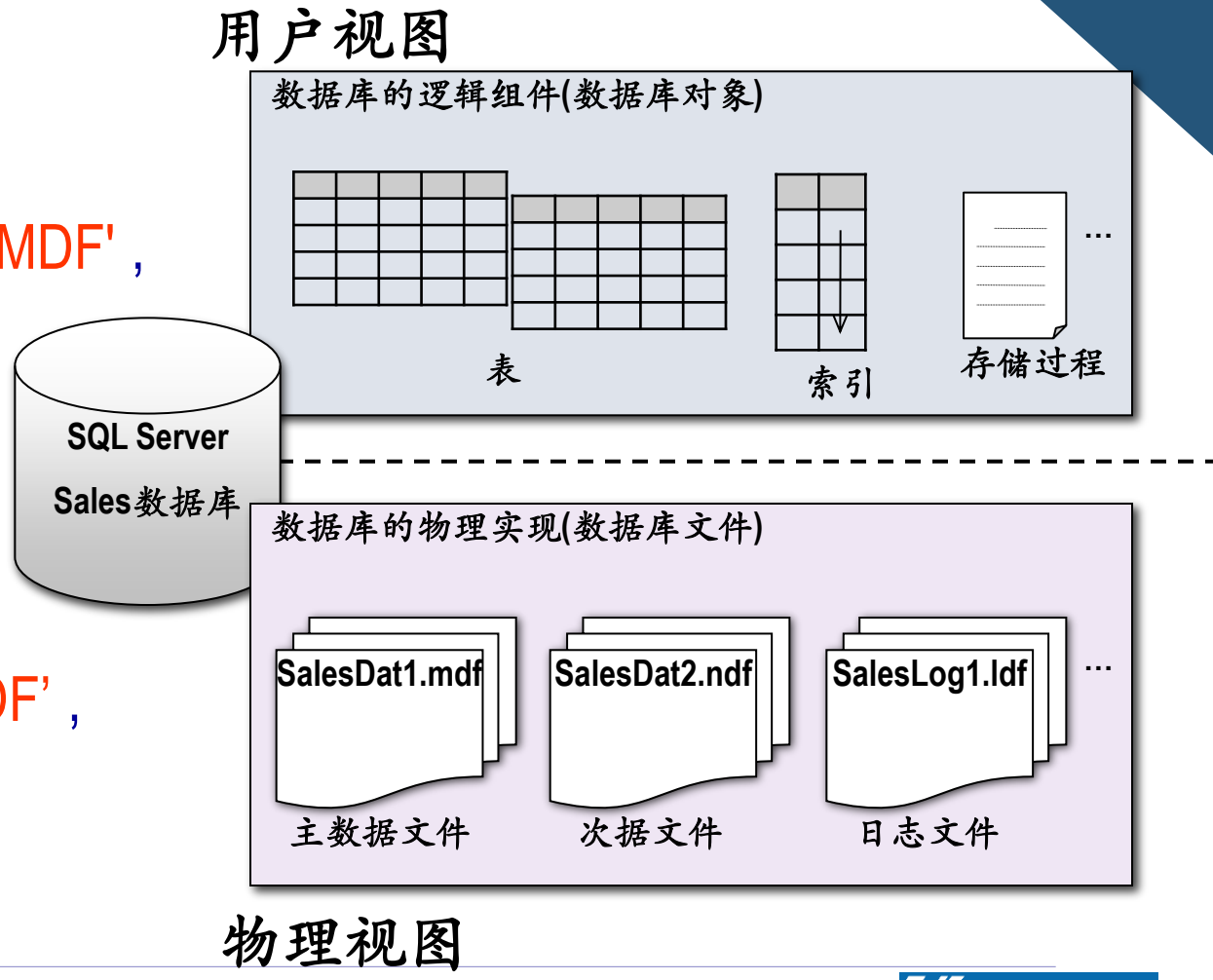
数据文件的增量。0值表示不增长。

指定文件组属性。

SQL Server创建数据库*

例：创建数据库example：

```
CREATE DATABASE [example]  ON
(NAME = 'example_Data',
 FILENAME = 'C:\MSSQL\data\example_Data.MDF' ,
SIZE = 2,
FILEGROWTH = 10%)
LOG ON
(NAME = 'example_Log',
FILENAME ='C:\MSSQL\data\example_Log.LDF' ,
SIZE = 1,
FILEGROWTH = 10%)
```



Mysql创建数据库*

Mysql

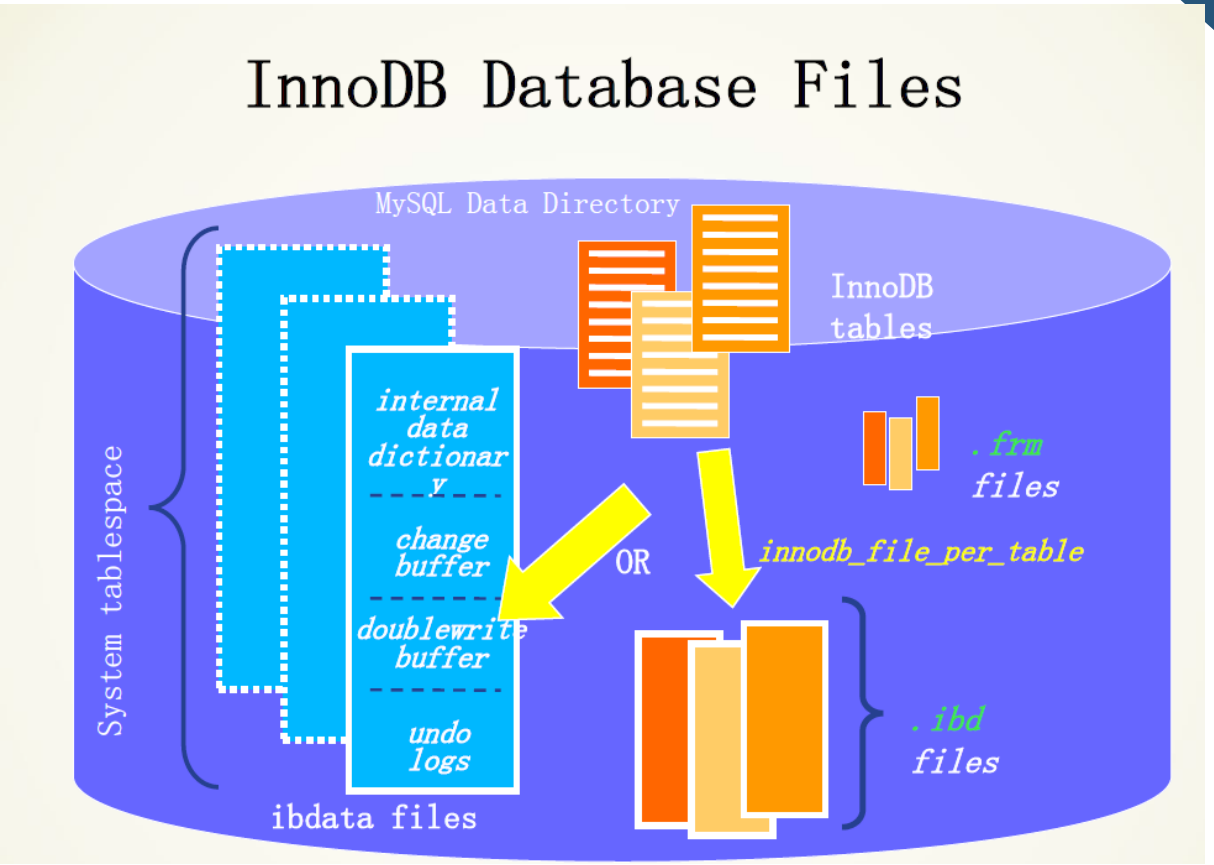
CREATE DATABASE [IF NOT EXISTS] <数据库名>

[[DEFAULT] CHARACTER SET <字符集名>]

[[DEFAULT] COLLATE <校对规则名>];

如：

Create database if not exists MyDB;



3.2.1 模式的定义与删除

- Schema（ISO/IEC 9075-1）：a persistent named collection of descriptors（因DBMS而异）
- SQL server

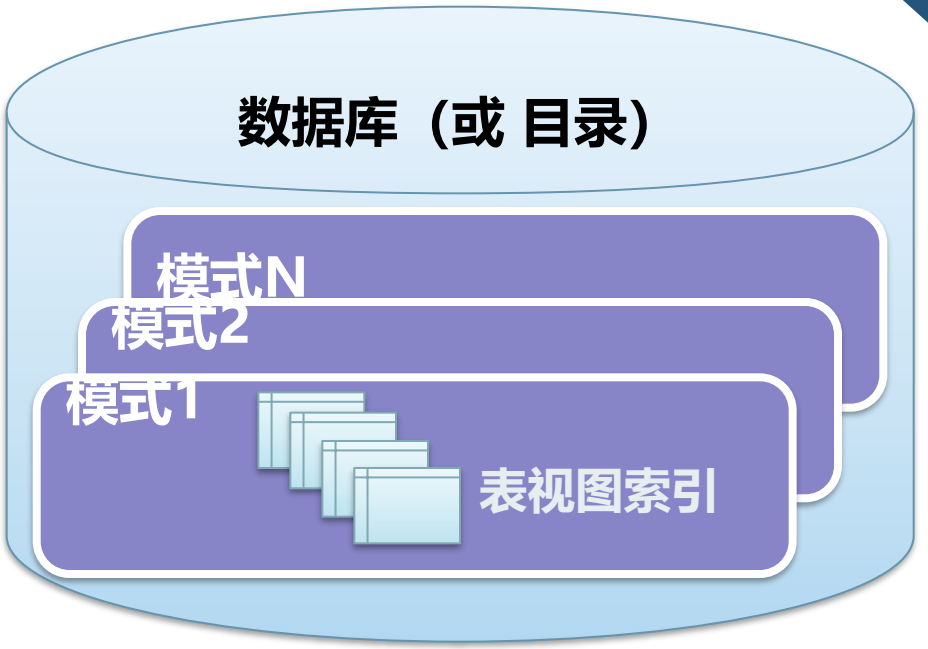
  Database schemas act as namespaces or containers for objects, such as tables, views, procedures, and functions.

- Oracle

  A schema is a collection of logical structures of data. A schema is owned by a database user and has the same name as that user. Each user owns a single schema.

- MySQL

  Physically, a schema is synonymous with a database.



[<数据库名>.][<模式名>.]<表名>

3.2.1 模式的定义与删除

1. 定义模式

- 模式定义语句如下：

```
CREATE SCHEMA <模式名> AUTHORIZATION <用户名>
        [<表定义子句>|<视图定义子句>|<授权定义子句>];
```

- 语义：为某用户创建一个模式。定义模式实际上定义了一个命名空间，其中可以定义该模式包含的数据库对象，例如基本表、视图、索引等。
  - SQL Server中：一个DB下可以有多个架构，不同架构下允许同名数据对象（db.schema.object）。每个DB有一个缺省的schema叫dbo。
  - Oracle：创建user同时创建同名schema；
  - Mysql：一个DB就是一个schema
- 在CREATE SCHEMA中可以接受CREATE TABLE，CREATE VIEW和GRANT子句。

1. 定义模式

创建模式的用户必须拥有DBA权限，或DBA授予的创建模式权限！

[例1] 为用户WANG定义了一个模式S-T

CREATE SCHEMA “S-T” AUTHORIZATION WANG;

[例2]CREATE SCHEMA AUTHORIZATION WANG；

<模式名>隐含为用户名WANG

// 如果没有指定<模式名>，那么<模式名>隐含为<用户名>

[例3] 为用户ZHANG创建一个模式TEST，并在其中定义一个表TAB1 。

```
CREATE SCHEMA TEST AUTHORIZATION ZHANG
    CREATE TABLE TAB1
        (COL1 SMALLINT, COL2 INT, COL3 CHAR(20) );
```

2. 删除模式

DROP SCHEMA <模式名> <CASCADE|RESTRICT>;

- 删除模式时其中已有表如何办?

  在删除语句中提供了CASCADE、RESTRICT选择项，说明如何删除:

  - CASCADE(级联)

    删除模式的同时把该模式中所有的数据库对象全部删除。

  - RESTRICT(限制)

    如果该模式中定义了下属的数据库对象（如表、视图等），则拒绝该删除语句的执行。当该模式中没有任何下属的对象时才能执行。

- [例4] DROP SCHEMA ZHANG CASCADE;

  删除例3定义的模式ZHANG，同时该模式中定义的表TAB1也被删除

3.2.2 基本表的定义、删除与修改

- 定义基本表：

  SQL DDL不仅允许定义一组关系，也要说明每个关系的信息：

  - 每个关系的模式
  - 每个属性的值域
  - 完整性约束
  - 每个关系的安全性和权限
  - 每个关系需要的索引集合
  - 每个关系在磁盘上的物理存储结构

- 备注：

  本章主要阐述如何定义模式、域、基本完整性及索引如何在SQL中定义，其他部分在后面章节

3.2.2 基本表的定义、删除与修改

1. 基本表的定义(创建)

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字
- <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

1. 创建基本表

□ 语法：

```
CREATE TABLE  表名(
    列名  数据类型  [DEFAULT  缺省值]  [NOT NULL]
    [, 列名 数据类型  [DEFAULT 缺省值]  [NOT NULL] … ]
    [, PRIMARY KEY(列名 [, 列名] …)]
    [, FOREIGN KEY (列名 [, 列名] …)
      REFERENCES  表名(列名 [, 列名] …)]
    [,  CHECK (条件表达式)] );
```

注：句法中[ ]表示该成分是可选项。

最常用的完整性约束：

- 主码约束： primary key
- 唯一性约束：unique
- 非空值约束：not null
- 参照完整性约束：foreign key
- 自定义的完整性约束：check(逻辑表达式)

1. 创建基本表

[例] 建立学生表Student，学号是主码，姓名取值唯一。

```
CREATE TABLE Student
 ( Sno   CHAR(9) PRIMARY KEY，  /* 列级完整性约束条件*/
   Sname  CHAR(20) UNIQUE，     /* Sname取唯一值*/
   Ssex    CHAR(2)，
   Sage   SMALLINT，
   Sdept  CHAR(20)
 );
```

Primary key 与 unique 的区别？

前者非空，后者允许一个空

学 号 Sno	姓 名 Sname	性 别 Ssex	年 龄 Sage	所在系 Sdept
200215121	李勇	男	20	CS

1. 创建基本表

[例] 建立一个“课程”表Course

```
CREATE TABLE  Course
( Cno      CHAR(4) PRIMARY KEY,
  Cname  CHAR(40) NOT NULL,
  Cpno     CHAR(4) ,
  Ccredit  SMALLINT,
   FOREIGN KEY (Cpno) REFERENCES Course(Cno)
);
```

课程号 Cno	课程名 Cname	先行课 Cpno	学分 Ccredit
1	数据库	5	4

1. 创建基本表

[例] 建立一个“学生选课”表SC

学 号 Sno	课程号 Cno	成绩 Grade
200215121	1	92

```
CREATE TABLE SC
(Sno CHAR(9),
Cno CHAR(4),
Grade SMALLINT CHECK (grade>=0 and grade <=100),
Semester CHAR(5),
Teachingclass CHAR(8),
PRIMARY KEY (Sno, Cno),   /* 主码由两个属性构成，必须作为表级完整性进行定义*/
FOREIGN KEY (Sno) REFERENCES Student(Sno), /* 表级完整性约束条件，Sno是外码，
  被参照表是Student */
FOREIGN KEY (Cno) REFERENCES Course(Cno) /* 表级完整性约束条件， Cno是外码，被
  参照表是Course*/
);
```

2. 数据类型

- SQL中域的概念用数据类型来实现;
- 定义表的属性时,需要指明其数据类型及长度;
- 选用哪种数据类型:
  - 取值范围
  - 要做哪些运算
- SQL 语言常用数据类型:
  - 精确数值类型;
  - 近似数值类型;
  - 字符串类型;
  - 日期时间类型;
  - 二进制对象BLOB、CLOB等。

数据类型	SQL92数据类型 - 含义
CHAR(n)	长度为n的定长字符串
VARCHAR(n)	最大长度为n的变长字符串
INT	长整数（也可以写作INTEGER）-2^31 ～2^31 -1
SMALLINT	短整数 -2^15 ～2^15 -1
NUMERIC(p,d)	定点数，由p位数字（不包括符号、小数点）组成，小数后面有d位数字
REAL	取决于机器精度的单精度浮点数
Double Precision	取决于机器精度的双精度浮点数
FLOAT(n)	可选精度的浮点数，精度至少为n位数字
DATE	日期，包含年、月、日，格式为YYYY-MM-DD
TIME	时间，包含一日的时、分、秒，格式为HH:MM:SS
INTEVAL	两个date或time类型数据之间的差

3. 模式与表

- 每一个基本表都属于某一个模式
- 一个模式包含多个基本表
- 定义基本表所属模式（常用方法2种）：
  - 方法一：在表名中明显地给出模式名

    Create table “S-T”.Student（.....）；/*模式名为S-T*/

    Create table “S-T”.Cource（.....）；

    Create table “S-T”.SC（.....）；

  - 方法二：在创建模式语句中同时创建表

Mysql

```
Create database SCC;
use SCC;
Create table student(…);
Create table Course(…);
Create table SC(…);
```

4. 修改数据表

一般格式：

```
ALTER TABLE <表名>
  [ ADD <新列名> <数据类型> [ 完整性约束 ] ]
  [ ADD <表级完整性约束> ]
  [ DROP [COLUMN] <列名> [CASCADE | RESTRICT]]
  [ DROP CONSTRAINT<完整性约束名>[CASCADE | RESTRICT] ]
  [ ALTER COLUMN<列名> <数据类型> ]
  [ RENAME COLUMN <列名> TO <新列名>];
```

□ 语义：

对名为 <表名>的表做ADD、DROP或ALTER COLUMN的操作：

- ADD可以增加一个新的列； DROP只能删除表上的完整性约束；
- ALTER只能更改列上的数据类型； RENAME用于修改列名。

4. 修改基本表

- [例] 向Student表增加“S_entrance(入学时间)”列，其数据类型为日期型；删除average列。

```
ALTER TABLE Student ADD S_entrance DATE;
```

说明：不论基本表中原来是否已有数据，新增加的列一律为空值，且除非表为空表或为列指定默认值，不允许新增约束为not null的列。

- [例] 将“入学时间”列的缺省值设为当前日期。

```
ALTER  TABLE   STUDENT  ALTER  S_entrance SET
                                DEFAULT CURRENT_DATE;
```

说明：修改列的定义可能破坏原有数据。

4. 修改基本表

- [例] 将表Course增加约束：先行课cpno必须引用本表的Cno 。

```
ALTER TABLE Course ADD CONSTRAINT fk_cpno FOREIGN KEY(cpno) REFERENCES Course(cno);
```

- [例] 删除列average；

```
ALTER TABLE Student DROP average;
```

- [例] 增加课程名称必须取唯一值的约束条件；删除Course表上的外键约束。

```
ALTER TABLE Student ADD UNIQUE(Cname);
ALTER TABLE Student DROP CONSTRAINT(fk_cpno);
```

5. 删除基本表

DROP TABLE <表名> [RESTRICT|CASCADE];

- RESTRICT：删除表是有限制的。
  - 欲删除的基本表不能被其他表的约束所引用；
  - 如果存在依赖该表的对象，则此表不能被删除。
- CASCADE：删除该表没有限制。
  - 在删除基本表的同时，相关的依赖对象一起删除。

- [例] 删除Student表

  DROP TABLE Student CASCADE ;

  基本表定义被删除，数据被删除；

  表上建立的索引、视图、触发器等一般也将被删除。

5. 删除基本表

- 例：删除前面建的三个表:Student,Course, SC。

```
Drop table SC;
Drop table Student;
Drop table Course;
```

问：三条删除命令可不可以变动次序？比如将第一条挪到后面？

5. 删除基本表

[例] 若表上建有视图，选择RESTRICT时表不能删除。
如果选择CASCADE时可以删除表，视图也自动被删除

```
CREATE VIEW IS_Student
   AS
SELECT Sno，Sname，Sage
   FROM  Student
   WHERE Sdept='IS';
```

```
DROP TABLE Student RESTRICT;
  --ERROR: cannot drop table Student because other objects depend on it.
```

```
DROP TABLE Student CASCADE;
 --NOTICE: drop cascades to view IS_Student
SELECT * FROM IS_Student;
--ERROR: relation " IS_Student " does not exist
```

5. 删除基本表

- DROP TABLE时，SQL99 与 3个RDBMS的处理策略比较。

序号	标准及主流数据库的处理方式 / 依赖基本表的对象	SQL99		Kingbase ES		ORACLE 9i		MS SQL SERVER 2000
		R	C	R	C		C	
1.	索引	无规定		√	√	√	√	√
2.	视图	×	√	×	√	√ 保留	√ 保留	√ 保留
3.	DEFAULT，PRIMARY KEY，CHECK（只含该表的列）NOT NULL 等约束	√	√	√	√	√	√	√
4.	Foreign Key	×	√	×	√	×	√	×
5.	TRIGGER	×	√	×	√	√	√	√
6.	函数或存储过程	×	√	√ 保留	√ 保留	√ 保留	√ 保留	√ 保留

R表示RESTRICT , C表示CASCADE

'×'表示不能删除基本表，'√'表示能删除基本表，‘保留’表示删除基本表后，还保留依赖对象

3.2.3 索引的建立与删除

- 建立索引的目的：加快查询速度
- 谁 建立索引：
  - DBA 或 表的属主（即建立表的人）
  - DBMS一般会自动建立以下列上的索引：

    PRIMARY KEY

    UNIQUE
- 谁 维护索引：

  DBMS自动完成
- 谁 使用索引：

  SQL用户并不直接使用索引。DBMS自动选择是否使用索引以及使用哪些索引

3.2.3 索引的建立与删除

- 常用的索引技术

  B+树索引　索引属性值组成B+树，具有动态平衡的优点

  HASH索引　索引属性值分桶，具有查找速度快的特点

  顺序索引　索引属性值排序，可二分查找

  位图索引　索引属性值用位向量描述

- 采用什么索引，由具体的RDBMS来决定
- 索引是关系数据库的内部实现技术，属于内模式的范畴
- CREATE INDEX语句定义索引时，可以定义索引是唯一索引、非唯一索引或聚簇索引

1. 建立索引

- 语句格式：

```
CREATE [UNIQUE] [CLUSTER] INDEX <索引名>
        ON <表名>(<列名>[<次序>][,<列名>[<次序>] ]...);
```

其中，

<次序>—指ASC(升序)/DESC(降序)，缺省为升序。

- 功能：在<表名>的表上，对其中的指定列，建立一个名为<索引名>的索引文件。
- 作用：提高查询速度。如从O(n)到O(log2n)。
- DBA建立、系统自动实现，与编程无关。
- 好处：提高速度
- 坏处：过多或不当的索引会耗费空间，且降低插入、删除、更新的效率。

3.2.3 索引的建立与删除

说明：

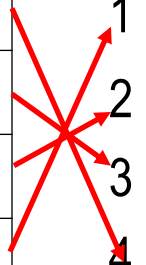
- UNIQUE(单一索引)：
  - 唯一索引，表示索引项值对应元组唯一，不允许存在索引值相同的两行
- CLUSTER(聚集索引)：索引项的顺序与表中记录的物理顺序一致。表中如果有多个记录在索引字段上相同，这些记录构成一簇，只有一个索引值。
- 在最经常查询的列上建立聚簇索引以提高查询效率；
- 经常更新的列不宜建立聚簇索引。
  - 优点：查询速度快。
  - 缺点：维护成本高，且一个表只能建一个聚簇索引。

3.2.3 索引的定义

□ 普通索引 vs 聚簇索引

按学号索引

sno	idx
17121	4
17122	3
17126	2
17128	1

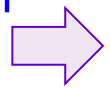


	sno	sname	ssex	sage	sdept
1	17128	李三姐	女	18	IS
2	17126	李一鸣	男	17	IS
3	17122	刘晨	女	19	CS
4	17121	李勇	男	20	CS

按姓名索引

sname	idx
李三姐	1
李一鸣	2
李勇	4
刘晨	3

按学号聚簇索引



sno	sname	ssex	sage	sdept
17121	李勇	男	20	CS
17122	刘晨	女	19	CS
17126	李一鸣	男	17	IS
17128	李三姐	女	18	IS

1. 建立索引

- [例] 为学生-课程数据库中的Student，Course，SC三个表建立索引。

```
CREATE UNIQUE INDEX  Stusno ON Student(Sno);
CREATE UNIQUE INDEX  Coucno ON Course(Cno);
CREATE UNIQUE INDEX  SCno ON SC(Sno ASC，Cno DESC);
    // Student表按学号升序建唯一索引
    // Course表按课程号升序建唯一索引
    // SC表按学号升序和课程号降序建唯一索引
```

- [例] 对Student表按姓名same的字典序升序建立一个简单索引。

```
CREATE  INDEX Stuname ON Student(Sname);
    //没有修饰词cluster,unique，列名sname后省略了asc
```

2. 修改索引

- 一般格式：

```
ALTER  INDEX <旧索引名> RENAME TO <新索引名>;
```

- [例] 将SC表索引Scno改名为SCSno。

```
ALTER  INDEX Scno RENAME TO SCSno;
```

3. 删除索引

- 一般格式：

```
DROP INDEX <索引名>;
或
DROP INDEX <索引名> ON <表名>;
```

删除索引时，系统会从数据字典中删去有关该索引的描述。

- [例] 删除Student表的Stusname索引

```
DROP INDEX Stusname;
DROP INDEX Stusname ON Student;
```

3.2.4 数据字典

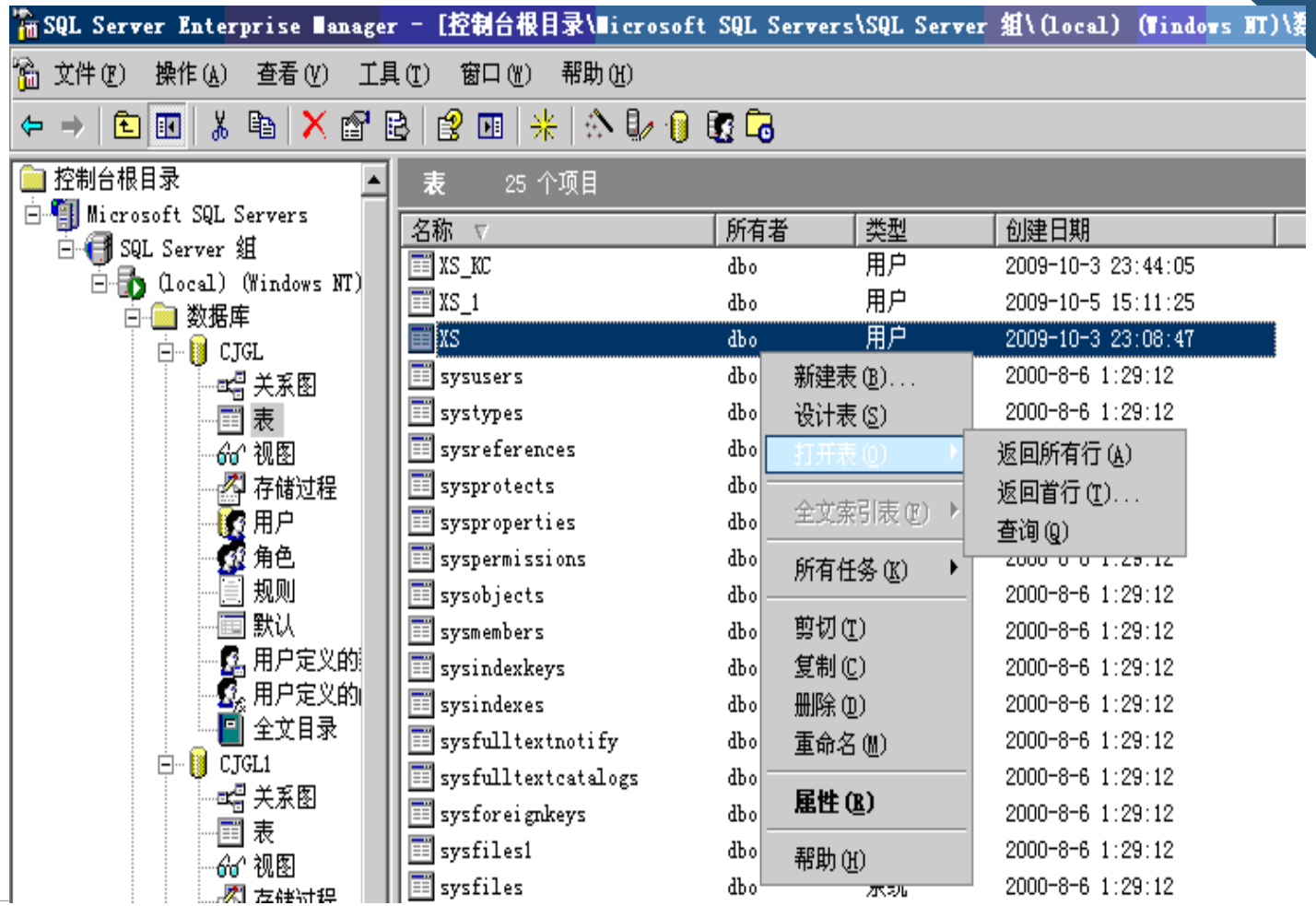
- 数据字典： 关系DBMS内部的一组系统表，记录数据库中所有的定义信息。
- 包括：
  - 关系模式定义；
  - 视图定义；
  - 索引定义；
  - 完整性定义；
  - 操作权限；
  - 统计信息等。

Mysql

information_schema.tables

information_schema.columns

information_schema.statistics



数据库系统原理

教程：数据库系统概论（第5版）

结合：CMU 15-445/645 INTRO TO DATABASE SYSTEMS

华中科技大学 计算机学院

左琼

第三章 关系数据库标准语言SQL

3.3 数据查询

- DML语言

  DBMS给上层提供的DML语言主要包括表的增、删、查，

  其中查询语句是核心DML语句。

[例] 查询选修2号课程且成绩在90分以上的所有学生。

关系代数：

$$\pi_{student.sno,\ sname}\ (\sigma_{cno='2' \wedge grade>90}(Student \bowtie SC)))$$

等价的SQL语句：

```
SELECT    Student.Sno, Sname
FROM    Student, SC
WHERE  Student.Sno = SC.Sno AND
          Cno= '2'  AND  Grade > 90;
```

3.3 数据查询

1. SELECT 语句的基本句法：

在关系代数中最常用的式子是下列表达式：

$$\pi_{A1,\ldots,An}(\sigma_F(R1\times\ldots\times Rm))$$

这里 R1,…,Rm为关系，F是公式，A1,…,An为属性。

针对上述表达式，SQL语言使用下列句型来表示：

```
SELECT  A1,…,An
FROM  R1,…,Rm
WHERE   F
```

此句型是从关系代数表达式演变过来的，但WHERE子句中的条件表达式 F 要比关系代数中公式更灵活。

3.3 数据查询

2. SELECT 语句的完整句法：

```
SELECT [ ALL|DISTINCT ] <目标列表达式> [，<目标列表达式>] …
 FROM <表名或视图名> [，  <表名或视图名> ] …
 [ WHERE <条件表达式> ]
 [ GROUP BY <列名1> [ HAVING <条件表达式> ] ]
 [ ORDER BY <列名2> [ ASC|DESC ]
 [ LIMIT <行数1> [ OFFSET <行数2> ] ];
```

注：

- WHERE子句称为行条件子句；
- GROUP子句称为分组子句；
- HAVING子句称为组条件子句；
- ORDER子句称为排序子句。

3.3 数据查询

3. SELECT 语句的一般执行过程

- 读取FROM子句中基本表及视图的数据，并执行笛卡尔积操作；
- 选取其中满足WHERE子句中条件表达式的元组；
- 按GROUP BY子句中指定列的值分组，同时提取满足HAVING子句中组条件表达式的那些组；
- 按ORDER BY子句对输出的目标表进行排序，按附加说明ASC升序排列，或按DESC降序排列。

查询语句的语法虽然看上去简单易用，实际上却是SQL语言中最难掌握的语句。主要难点在于查询条件的正确表达。

3.3 数据查询

3.3.1 单表查询

3.3.2 连接查询

3.3.3 嵌套查询

3.3.4 集合查询

3.3.5 基于派生表的查询

3.3.1 单表查询

单表查询 —— 只涉及到一张表的查询。

- SELECT子句的<目标列表达式>可以为：
  - 列名[,列名]…；
  - *
  - 算术表达式；
  - 字符串常量；
  - 函数；
  - 列别名

1. 选择表中的若干列

[例] 查询全体学生的学号与姓名（指定列）。

```
SELECT Sno，Sname FROM Student；
```

[例] 查询全部学生的所有信息。

```
SELECT * FROM STUDENT;
```

1. 选择表中的若干列

- 问题：若希望的查询结果表中属性无法直接用SELECT得出，但可以通过运算得出，如何处理这种派生属性?

[例] 列分别为算术表达式、字符串常量、函数的示例，并使用列别名改变查询结果的列标题:

```
SELECT Sname NAME，'Year of Birth: ' BIRTH，
       EXTRACT (year from CURDATE())-Sage  BIRTHDAY，
       LOWER(Sdept)  DEPARTMENT
FROM Student；
```

NAME	BIRTH	BIRTHDAY	DEPARTMENT
李勇	Year of Birth	2000	cs
刘晨	Year of Birth	2001	cs

//Mysql: CURDATE() 返回当前的日期

3.3.1 单表查询

2. 选择表中的若干元组

- 问题：对一个关系SELECT后，结果关系中可能出现重复元组，实现中，为提高效率，SELECT并不消除重复元组。

(1) 消除取值重复的行

如果没有指定DISTINCT关键词，则缺省为ALL。

[例] 查询选修了课程的学生学号。

```
SELECT Sno FROM SC;
```

等价于：

```
SELECT ALL Sno FROM SC;   //结果中可以有多个相同元组
```

[例] 查询全部被选课程的课程号（去除重复值）。

```
SELECT DISTINCT Cno FROM SC;
```

2. 选择表中的若干元组

（2）查询满足条件的元组

- 如何从表中选择指定元组?

  对应于关系代数运算σ_P，SQL提供where子句解决表元组的选择。

- 格式：WHERE <条件表达式>

<条件表达式>是包含属性名的逻辑表达式P，通过P对元组进行筛选。

查 询 条 件	谓　词
比　较	=，>，<，>=，<=，!=，<>，!>，!<； NOT+上述比较运算符
确定范围	BETWEEN AND，NOT BETWEEN AND
确定集合	IN，NOT IN
字符匹配	LIKE，NOT LIKE
空　值	IS NULL，IS NOT NULL
多重条件(逻辑运算)	AND，OR，NOT

2. 选择表中的若干元组

（2）查询满足条件的元组

[例] 查询考试成绩有不及格的学生的学号。

```
SELECT DISTINCT Sno
  FROM  SC
  WHERE Grade<60;   //比较大小
```

[例] 查询年龄不在20~23岁之间的学生姓名、系别和年龄。

```
SELECT Sname，Sdept，Sage
   FROM    Student
   WHERE Sage NOT BETWEEN 20 AND 23;
```

//确定范围：闭区间

确定集合

- SQL中提供了元素与集合之间的比较运算符:
- 谓词： x IN <值表>，x NOT IN <值表>

其中 <值表> 是一个集合，从关系代数的角度看， 它是一个代数式，从SQL角度看，它是一个SELECT语句。

[例] 查询信息系（IS）、数学系（MA）和计算机系（CS）学生的姓名和性别。

```
SELECT Sname，Ssex
FROM  Student
WHERE Sdept IN ( 'IS'，'MA'，'CS' );
```

[例] 查询既不是信息系、数学系，也不是计算机科学系的学生的姓名和性别。

```
SELECT Sname，Ssex
FROM Student
WHERE Sdept NOT IN ( 'IS'，'MA'，'CS' );
```

字符匹配

- 为什么提供字符匹配运算符?

  关系数据库支持对集合的运算，实际应用中，往往需要从集合中找出类似于某个条件的元组，即模糊查询。SQL的字符匹配运算符为解决这类问题而提出。

- 谓词:

  [NOT] LIKE ‘<匹配串>’ [ESCAPE ‘<换码字符>’]

- 通配符:

  SQL规定符号百分号%及下划线_具有其他含义

  百分号% 代表任意长度的字符串

  下划线_ 代表任意一个字符

  <匹配串> 为可以含有通配符的字符串

  ESCAPE 是将百分号% 或下划线_转回其本意

字符匹配示例

[例] 查询所有姓刘学生的姓名、学号和性别。

```
SELECT Sname，Sno，Ssex FROM Student
        WHERE Sname LIKE '刘%' ;
```

[例] 查询姓“欧阳”且全名为三个汉字的学生的姓名。

```
SELECT Sname FROM Student
        WHERE Sname LIKE '欧阳_ _';
```

[例] 查询DB_Design课程的课程号和学分。

```
SELECT Cno，Ccredit FROM Course
    WHERE Cname LIKE 'DB\_Design' ESCAPE '\';
```

[例] 查询以“DB_”开头，且倒数第3个字符为 i的课程的详细情况。

```
SELECT * FROM Course
     WHERE Cname LIKE 'DB\_%i_ _' ESCAPE '\';
    // ESCAPE ' \ ' 表示 “ \ ” 为换码字符
```

2. 选择表中的若干元组

（2）查询满足条件的元组

[例] 某些学生选修课程后没有参加考试，所以有选课记录，但没有考试成绩。查询缺少成绩的学生的学号和相应的课程号。

```
SELECT Sno，Cno FROM SC
WHERE Grade IS NULL;
```

Notice: “IS” 不能用 “=” 代替。

- 多重条件查询：

  逻辑运算符：AND和 OR来联结多个查询条件

  - AND的优先级高于OR
  - 可以用括号改变优先级
  - 可用来实现多种其他谓词：IN，BETWEEN…AND

多重条件查询

□ 逻辑运算符：AND和OR可以用来将多个简单查询条件复合成更加复杂的条件

[例] 查询信息系、数学系和计算机系20岁以下的学生的姓名和性别。

```
SELECT Sname，Ssex
 FROM Student
 WHERE Sdept IN (‘IS’，‘MA’，‘CS’) AND Sage<20;
```

可改为:

```
SELECT Sname，Ssex
 FROM Student
 WHERE （Sdept=‘IS’ OR Sdept=‘MA’ OR
        Sdept= ‘CS’） AND Sage<20;
```

优先级：NOT > AND > OR

3. ORDER BY子句

ORDER BY 子句：

- 可以按一个或多个属性列排序。如：按(系别，年龄)排序；
- 升序：ASC；降序：DESC；缺省值为升序。

[例] 列出选修了课程号为 ‘C6’ 的所有学生的学号和成绩，并按分数的降序排列。

```
SELECT  sno, grade
  FROM    SC
  WHERE  cno='C6'
  ORDER  BY  grade  DESC;
```

注：当排序列含空值null时，SQLServer、Mysql认为空值最小，ASC：null排最前，DESC：反之。而Oracle认为空值最大，ASC：null排最后，DESC：反之。

4. 聚集函数

- SQL聚集函数：
  - 计数

    COUNT（*）：统计元组个数

    COUNT（[DISTINCT|ALL] <列名>）：统计一列中值的个数
  - 计算总和

    SUM（[DISTINCT|ALL] <列名>）
  - 计算平均值

    AVG（[DISTINCT|ALL] <列名>）
  - 最大最小值

    MAX（[DISTINCT|ALL] <列名>）

    MIN（[DISTINCT|ALL] <列名>）

集函数只能用于Select子句和Having子句中。

注：

-DISTINCT: 统计时去掉指定列中的重复值

-对于NULL：除了count(*),其他函数都跳过空值

4. 聚集函数

[例] 查询学生总人数。

```
SELECT COUNT(*) FROM  Student;
```

[例] 查询选修了课程的学生人数。

```
SELECT COUNT(DISTINCT Sno) FROM SC;
```

[例] 查询选修3号课程学生的最高分、最低分和平均成绩。

```
SELECT  MAX(grade), MIN(grade), AVG(grade)
FROM  SC
WHERE  cno= '3' ;
```

// 当集函数遇到NULL时，除COUNT(*),都跳过空值而只处理非空值。

5. GROUP BY子句

- 分组是按某（些）列的值对查询的结果进行分类。
- 格式： GROUP BY A1，A2，...,An

  其中：Ai为属性名

  按GROUP子句中指定的列的值进行分组，值相等的为一组。
- HAVING子句可以对分组后的结果作进一步的筛选。
- 当查询语句中有GROUP子句时，集函数作用的对象是一个分组的结果，而不是整个查询的结果。

[例] 求各个课程号及相应的选课人数。

```
SELECT Cno，COUNT(Sno) FROM SC
    GROUP BY Cno;
```

Cno	COUNT（Sno）
1	22
2	34
3	44
4	33
5	48

5. GROUP BY子句

- 注：分组后，一些详细信息可能损失，不能出现在SELECT结果中。
- 例如，下面的查询

```
SELECT Sno, Grade, COUNT(*)
    FROM SC
    WHERE Cno='2'
    GROUP BY Grade;
```

Sno	Cno	Grade
1	1	95
2	1	96
3	1	NULL
4	1	NULL

Sno	Grade	Count(*)
1	95	1
2	96	1
???	NULL	2

将出错，想一想，为什么？

- 一般来说，分组查询的SELECT目标列中只允许出现聚集函数和GROUP BY子句中出现过的列。

5. GROUP BY子句

□ 按分组计算结果排序

[例] 列出每门课程号及相应的选修人数，按选修人数的多少，从高到低排序。

```
SELECT   Cno, Count(Sno)
 FROM     SC
 GROUP BY   Cno
 ORDER BY   Count(Sno)   DESC;
```

Cno	Count(Sno)
3	95
1	48
2	33

单表查询

- 特殊语法* 求Top n

[例] 求1号课程的最高成绩,及取得最好成绩的学生学号。

Mysql

```
SELECT  Sno, Grade  FROM  SC
        WHERE   Cno='1'
        ORDER BY  Grade  DESC
        LIMIT 1;
```

SQL Server

```
SELECT TOP 1  WITH TIES sno, grade
FROM  SC
WHERE    cno='1'
ORDER    BY    grade  DESC;
```

等价于：

嵌套查询

```
SELECT
Sno, Grade  FROM  SC
WHERE  Grade
            = (SELECT MAX(Grade) FROM SC WHERE Cno = '1')
          AND  Cno = '1' ;
```

5. GROUP BY子句

- 问题： 有时，对于一个分组以后的结果集合希望使用限定条件选择部分分组，则可以使用Having 子句。
- 格式： Having P; P是谓词
- 注意： 由于HAVING 子句中的谓词P是在分组以后起作用的，因此P中可以使用聚集函数。

[例] 查询选修了三门以上课程的学生学号。

```
SELECT Sno FROM SC
    GROUP BY Sno
    HAVING COUNT(*)>3;
```

注意：统计结果和组别属性不一定非要出现在查询列表中，但非统计函数及非组别属性，一定不允许出现在查询列表中

5. GROUP BY子句

[例] 查询至少有3门课程成绩在90分以上的学生的学号及（90分以上的）课程数。

```
SELECT   Sno, COUNT(*)
 FROM   SC
 WHERE  Grade > =90
 GROUP BY  Sno
 HAVING   COUNT(*) > 3
```

- 注：WHERE子句与HAVING子句的区别: 作用对象不同。
  - WHERE子句作用于基本表或视图，从中选择满足条件的元组；
  - HAVING短语作用于组，从中选择满足条件的组。

5. GROUP BY子句

- 带CASE子句的HAVING*

[例] 列出至少有三门课程成绩90分以上的学生学号及其平均成绩。

```
SELECT  Sno, AVG(grade) as Avg_grade
   FROM  SC
   GROUP  BY  Sno
   HAVING  SUM ( CASE WHEN Grade >= 90 THEN  1 ELSE  0 END ) >= 3
```

Mysql

```
HAVING SUM ( IF ( Grade >= 90, 1,0 ) ) >= 3
```

3.3 数据查询

回顾

SELECT 语句的完整句法：

SELECT [ALL|DISTINCT] <目标列表达式>

[，<目标列表达式>] …

FROM <表名或视图名>[，<表名或视图名>] …

[WHERE <条件表达式>]

[GROUP BY <列名1> [HAVING <条件表达式>]]

[ORDER BY <列名2> [ASC|DESC]];

连接（一般格式）：

- [<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>
- [<表名1>.]<列名1> BETWEEN [<表名2>.]<列名2> AND [<表名2>.]<列名3>

等值连接 / 自然连接；自连接、嵌套连接

3.3.2 连接查询

- 连接查询：同时涉及多个表的查询
- 连接条件或连接谓词：用来连接两个表的条件

- 一般格式：
  - [<表名1>.]<列名1> <比较运算符> [<表名2>.]<列名2>
  - [<表名1>.]<列名1> BETWEEN [<表名2>.]<列名2> AND [<表名2>.]<列名3>

- 连接字段：连接谓词中的列名称
  - 连接条件中的各连接字段类型必须是可比的，但名字不必是相同的

3.3.2 连接查询

1. 等值连接与非等值连接查询

等值连接：连接运算符为=

[例] 查询每个学生及其选修课程的情况。

```
SELECT Student.*, SC.* FROM Student, SC
    WHERE Student.Sno = SC.Sno;
```

查询结果：

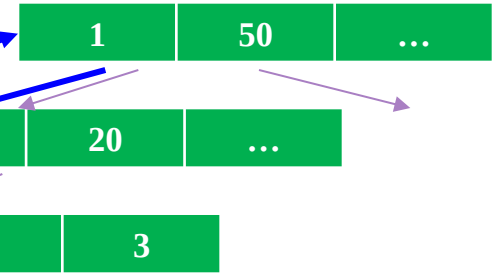
Student.Sno	Sname	Ssex	Sage	Sdept	SC.Sno	Cno	Grade
200215121	李勇	男	20	CS	200215121	1	92
200215121	李勇	男	20	C			
200215121	李勇	男	20	C			
200215122	刘晨	女	19	C			
200215122	刘晨	女	19	CS	200215122	3	80

改为自然连接：

```
SELECT Student.*, Cno, Grade
    FROM Student, SC
    WHERE Student.Sno = SC.Sno;
```

连接操作的执行方法

- 嵌套循环法(NESTED-LOOP)
- 排序合并法(SORT-MERGE)
- 索引连接法(INDEX-JOIN)
- 哈希法(HASH-JOIN)



Student

Sno	Sname	Ssex	Sage	Sdept
1	李勇	男	19	CS
2	刘晨	女	18	CS
3	王敏	女	18	MA

SC

Sno	Cno	Grade
1	1	92
1	2	85
2	1	88
2	2	90
3	1	80

```
SELECT Sname, Cno, Grade
    FROM  Student, SC
    WHERE  Student.Sno = SC.Sno;
```

2. 连接查询

[例] 求“数据结构”课程成绩大于85分的学生姓名和成绩，结果按成绩降序排列。

```
SELECT   STUDENT.sname，grade
FROM     STUDENT，SC，COURSE
WHERE    STUDENT.sno = SC.sno  AND
         SC.cno = COURSE.cno   AND
         COURSE.cname = '数据结构'  AND
         SC.grade > 85
ORDER BY  grade DESC;
```

连接条件

筛选条件

谁先做？

注：当不同的表中有同名属性时，属性名前要用表名限定。

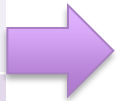
2.自身连接

一个表与其自己进行连接，称为表的自身连接。

[例] 求同时选修了1号课程和2号课程的学生学号。

```
SELECT  a.sno
FROM  CS a, CS b
WHERE  a.sno = b.sno  AND
       a.cno =  '1'  AND b.cno =  '2' ;
```

Sno	Cno	Grade
1	1	92
1	2	85
2	1	88
2	2	90
3	1	80



a.Sno	a.Cno	a.Grade	b.Sno	b.Cno	b.Grade
1	1	92	1	1	92
1	1	92	1	2	85
1	2	85	1	1	92
1	2	85	1	2	85
……					

自身连接 （续）

[例] 查询每一门课的间接先修课（即先修课的先修课）

```
SELECT FIRST.Cno, SECOND.Cpno
FROM Course FIRST, Course SECOND
WHERE FIRST.Cpno = SECOND.Cno;
```

Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL语言	6	4

Cno	Cname	Cpno	Ccredit
1	数据库	5	4
2	数学		2
3	信息系统	1	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理		2
7	PASCAL语言	6	4

查询结果：

Cno	Cpno
1	7
3	5
4	NULL
5	6
7	NULL

连接

```
select student.sno,sname,cname,grade
from sc
join student on student.sno = sc.sno
join course on  course.cno = sc.cno;
```

- SQL中表的连接有两种表示方法：
- 方法1：在FROM子句中指明进行连接的表名，在WHERE子句中指明连接的列名及其连接条件，如前面的例子。
- 方法2：利用关键字JOIN进行连接。具体分为以下几种：
  - INNER JOIN ： 返回符合连接条件的记录；
  - LEFT [OUTER] JOIN：返回符合连接条件的数据行以及左边表中不符合条件的数据行，此时右边数据行以NULL来显示，称为左连接；
  - RIGHT [OUTER] JOIN：返回符合连接条件的数据行以及右边表中不符合条件的数据行，此时左边数据行以NULL来显示，称为右连接；
  - FULL [OUTER] JOIN：返回符合连接条件的数据行以及左边表和右边表中不符合条件的数据行，此时缺乏数据的数据行会以NULL来显示；
  - CROSS JOIN：返回笛卡儿积。
  - 通过关键词ON与JOIN相对应，指明连接条件。

3. 外连接

- LEFT JOIN、RIGHT JOIN与 FULL JOIN统称为外连接。可用来显示不满足连接条件的元组。
- 某些查询要求只能用外连接来表达。

Sno	Sname
1	李勇
2	刘晨
3	王敏

Sno	Cno	Grade
1	1	92
1	2	85
2	1	88

［例］查询所有学生的学号、姓名、所选课程的课程号以及这门课的成绩（没有选修任何课程的同学信息也要求被查询出来，相应的选课信息显示为空）。

```
SELECT STUDENT.sno, sname, cno, grade
FROM   STUDENT  LEFT OUTER JOIN  SC
ON STUDENT.SNO=SC.SNO;
```

Student.Sno	Sname	Cno	Grade
1	李勇	1	92
1	李勇	2	85
2	刘晨	1	88
3	王敏	NULL	NULL

3.3.3 嵌套查询

什么是嵌套查询?

- 一个SELECT-FROM-WHERE语句称为一个查询块；将一个查询块嵌套在另一个查询块的 WHERE子句或HAVING短语的条件中的查询称为嵌套查询，相当于在SELECT中调用另一段SELECT。

【例】查询选修了2号课程的学生姓名。

```
SELECT Sname          /*外层查询/父查询*/
 FROM Student
 WHERE Sno IN
        (SELECT Sno   /*内层查询/子查询*/
         FROM SC
         WHERE Cno= '2') ;
```

- 子查询不能使用ORDER BY子句；
- 层层嵌套方式反映了 SQL语言的结构化；
- 有些嵌套查询可以用连接运算替代。

嵌套查询的应用场景

- 一个查询块是对关系集合进行搜索找出满足资格的元组。对目标关系中成员t的判断可能会出现情况：

(1) 属于运算：该成员t是否属于某个select结果集合R； IN谓词

$t \in R$ 是否成立?

(2) 比较运算：该成员是否比某个select集中所有成员或至少一个成员大或小；

t > R中所有或某个成员是否成立?

(3) 逻辑运算：该成员是否能使得集合逻辑式成立；

t是否能使得逻辑命题L成立？ 其中：L是一个通过t求出的集合逻辑式。

L（R(t)）

3.3.3 嵌套查询

1. 当子查询的返回值只有一个时，可以使用比较运算符（=, >, <, >=, <=, !=）将父查询和子查询连接起来。

[例] 查询与学号为95003的学生同系的学生学号和姓名。

```
SELECT sno, sname   //父查询
FROM STUDENT
WHERE sdept = (SELECT sdept     //子查询
FROM STUDENT
WHERE sno= '95003' );
```

本例改为：

```
SELECT sno, sname
   FROM STUDENT
     WHERE
       ( SELECT sdept
           FROM STUDENT
        WHERE sno= '95003' );
```

对吗？

3.3.3 嵌套查询

当子查询的返回结果不止一个，而是一个
这时可以使用IN、ANY、ALL或EXISTS谓词

本例用自身连接：

```
SELECT a.Sno, a.Sname, a.Sdept
FROM Student a, Student b
WHERE a.Sdept = b.Sdept AND b.Sname = '李冬';
```

2. 带有IN谓词的子查询

[例] 查询与“李冬”同系的学生学号和姓名（可能有同名）。

```
SELECT Sno, Sname
FROM STUDENT
WHERE sdept IN
(SELECT Sdept
 FROM STUDENT
WHERE sname= ‘李冬’ );
```

子查询的查询条件不依赖于父查询——不相关子查询

3.3.3 嵌套查询

[例] 找出每个学生超过他选修课程平均成绩的课程号。

```
SELECT Sno, Cno
FROM SC x
WHERE Grade >=(SELECT AVG(Grade)
               FROM SC y
               WHERE y.Sno=x.Sno);
```

子查询的查询条件与父查询相关——相关子查询

如何实现?

Sno	Cno	Grade
1	1	92
1	2	85
2	1	88
2	2	90
3	1	80

带有比较运算符的子查询问题分析

- 相关嵌套查询效率低，如何提高？ 将相关查询改为无关查询。

[例] 对于上例，先从选课表中找出每个学生选课平均成绩，
再从选课表中找出所选课程成绩大于平均成绩的学生。

```
SELECT Sno, Cno
FROM SC, (SELECT Sno, Avg(Grade) FROM SC
          Group by Sno)
          AS Avg_sc(avg_sno, avg_grade)
WHERE SC.sno = avg_sno
      AND grade > avg_grade;
```

必须为派生关系指定别名

子查询执行方式

子查询分为非相关子查询和相关子查询。二者执行方式不同：

- 非相关子查询的执行顺序是：
  - 首先执行子查询；
  - 父查询涉及的所有元组都与子查询的查询结果进行比较，以确定查询结果集合。
- 相关子查询的执行顺序是：
  - 首先选取父查询表中的一个元组，内部的子查询利用此元组中相关的属性值进行查询；
  - 然后父查询根据子查询返回的结果判断此行是否满足查询条件。如果满足条件，则把该行放入父查询的查询结果集合中。
  - 重复执行上述过程，直到处理完父查询表中的所有元组。

由此可以看出，非相关子查询只执行一次；而相关子查询的执行次数是由父查询表的行数决定的。

3.3.3 嵌套查询

- 课堂练习：查询平均成绩高于“王军”同学平均成绩的学生姓名和学号；

```
select sno,sname
from student s join sc on sc.sno=s.sno
group by s.sno,sname
having avg(grade)>(select avg(grade)
                   from sc,student
                   where sc.sno=student.sno
                     and sname='王军');
```

```
select sno,sname from student
where sno in (select sno from sc
      group by sno having avg(grade)>(
         select avg(grade) from sc
         where sno=(select sno from student
                    where sname='王军'))
     )
```

```
select sno,sname from student
where (select avg(grade) from sc as
sc1 where sc1.sno=student.sno)
   >
  (select avg(grade) from sc as
sc2,student s2 where
sc2.sno=s2.sno and sname='王军')
```

3.3.3 嵌套查询

- 问题：在嵌套查询情形二中，若需判断关系中元组t与一个集合的“任意一个”或“所有”是否满足某个关系式，该如何解决?

3. 带有ANY（SOME）或ALL谓词的子查询

- 谓词语法：

  ANY(R)： R的任意一个值

  ALL(R)： R所有值

- 谓词语义：与关系运算符配合使用

  > ANY(R)：至少比R...的某一个大

  > ALL(R)：比R...所有大

> ANY	大于子查询结果中的某个值
> ALL	大于子查询结果中的所有值
< ANY	小于子查询结果中的某个值
< ALL	小于子查询结果中的所有值
>= ANY	大于等于子查询结果中的某个值
>= ALL	大于等于子查询结果中的所有值
<= ANY	小于等于子查询结果中的某个值
<= ALL	小于等于子查询结果中的所有值
= ANY	等于子查询结果中的某个值
=ALL	等于子查询结果中的所有值（通常没有实际意义）
!= （或<>） ANY	不等于子查询结果中的某个值
!= （或<>） ALL	不等于子查询结果中的任何一个值

3.3.3 嵌套查询

[例]：查询其他系中比信息系某一学生的年龄小的学生姓名及年龄。

```
SELECT sname, sage
 FROM STUDENT
  WHERE  sage < ANY
          (SELECT sage
            FROM STUDENT
            WHERE sdept =  'IS' )
   AND sdept <>  'IS' ;
```

ANY或ALL谓词可用集函数或IN谓词等价表示：

```
SELECT Sname，Sage
   FROM  Student
   WHERE Sage <
         (SELECT MAX(Sage)
               FROM Student
               WHERE Sdept= 'IS ')
     AND Sdept <> ' IS ';
```

执行过程：

1.RDBMS执行此查询时，首先处理子查询，找出IS系中所有学生的年龄，构成一个集合;

2. 处理父查询，找所有不是IS系且年龄小于上述集合中某个值的学生。

带有ANY或ALL谓词的子查询

表3.5 ANY、ALL谓词与聚集函数、IN谓词的等价转换关系

	=	<> 或 !=	<	<=	>	>=
ANY	IN	--	<MAX	<=MAX	>MIN	>=MIN
ALL	--	NOT IN	<MIN	<=MIN	>MAX	>=MAX

3.3.3 嵌套查询

4.带有EXISTS谓词的子查询

- EXISTS表示存在量词。本质上是一个返回值为“真”/“假”的集函数，用于判断一个集合是否为空。 EXISTS（R），R为非空则返回真。NOT EXISTS与此相反。
- 由EXISTS引出的子查询，其目标列表达式通常都用*，因为带EXISTS的子查询只返回真值或假值，给出列名无实际意义。
- 带EXISTS的子查询的用法：
  - 不同形式的查询间的替换: 带IN谓词、比较运算符、ANY和ALL谓词的子查询 与 带EXISTS谓词的子查询 等价替换
  - 用EXISTS/NOT EXISTS实现全称量词(难点)
  - 用EXISTS/NOT EXISTS实现逻辑蕴函(难点)

3.3.3 嵌套查询

- 不同形式的查询间的替换示例

[例] 查询选修了1号课程的学生姓名。

```
SELECT sname
   FROM STUDENT
   WHERE EXISTS
     ( SELECT *  FROM  SC
         WHERE SC.Sno=Student.Sno
           AND Cno=  '1'  );
```

item in (select attribute from T)

等价于：

```
exists (select *  from  T
        where  attribute = item  )
```

等价于：

方法1：

```
SELECT Sname
   FROM Student, SC
   WHERE Student.Sno=SC.Sno
         AND SC.Cno= '1';
```

方法2：

```
SELECT  Sname
 FROM   Student
 WHERE  Sno IN
      （SELECT Sno
        FROM SC
        WHERE Cno='1');
```

3.3.3 嵌套查询

```
SELECT Sname，Sage
FROM  Student
WHERE Sage < ANY (SELECT  Sage
                  FROM   Student
                  WHERE Sdept=  'CS' )
                  AND Sdept <>  ‘CS’ ;
```

□ 不同形式的查询间的替换示例

[例2] 查询其他系中比计算机科学系某一学生年龄

【分析】

1）用EXISTS函数判断任一个学生t，其年龄比计算机系任一学生年龄小

2）从其他系学生表中，搜索让上述逻辑式为真的元组

```
SELECT Sname，Sage
FROM  Student S1
WHERE EXISTS (SELECT * FROM Student S2
                    WHERE Sdept = 'CS' AND S1.Sage < S2.Sage)
              AND Sdept <> 'CS';
```

注：S1是其他系学生元组变量，S2是计算机系元组变量。

3.3.3 嵌套查询

[实例] 查询2015-3-20,2015-5-1分别

- 2015-3-20 9:00pm – 次日 2:00am
  中国地质大学(28991,7202)
- 2015-5-1 9:00pm – 次日 2:00am:
  湖北大学(28961,10522)

主叫 Oid	被叫 Tid	开始时间 start	时长 period	区域 lac	基站 cel

```
select oid,tid,start,period,lac,cell
from  calling  A
where lac = 28991 and
        cell = 7202 and
        start  between '2015-3-20 21:00' and
                       '2015-3-21 2:00' and
        exists  (
                select * from calling B
                where A.oid = B.oid  and
                        lac =  28961 and
                        cell = 10522 and
                        start between '2015-5-1 21:00'
                                and   '2015-5-2 2:00'
```

3.3.3 嵌套查询

把带有全称量词的谓词转换为等价的带有存在量词的谓词：

$(\forall x)P \equiv \neg (\exists x(\neg P))$

- 用EXISTS/NOT EXISTS实现全称量词示例

[例] 查询选修了全部课程的学生姓名。

【分析】“选修了全部课程” ⇔ “没有一门课他没有选”

```
SELECT sname  FROM STUDENT
WHERE  NOT EXISTS          //不存在学生t没选任一门课c
      (SELECT *
       FROM  COURSE
       WHERE  NOT EXISTS          //学生t 选修了一门课c，为
                                  “假”，即：t没选课程c
        (SELECT *
           FROM   SC
           WHERE  sno=STUDENT.sno AND          //学生t 选修
                  cno=COURSE.cno))             了一门课c
```

3.3.3 嵌套查询

用EXISTS/NOT EXISTS实现逻辑蕴含示例

[例]查询至少选修了学生200215122选修的全部课程的学生号码。

解题思路：

- 用逻辑蕴函表达：查询学号为x的学生，对所有的课程y，只要200215122学生选修了课程y，则x也选修了y。
- 形式化表示：

  用p表示谓词 “学生200215122选修了课程y”

  用q表示谓词 “学生x选修了课程y”

  则上述查询为: $(\forall y)\, p \rightarrow q$
- 等价变换：

  $(\forall y)p \rightarrow q \equiv \neg (\exists y (\neg(p \rightarrow q)))$

  $\equiv \neg (\exists y (\neg(\neg p \vee q))) \equiv \neg \exists y(p \wedge \neg q)$

变换后语义：不存在这样的课程y，学生200215122选修了y，而学生x没有选。

3.3.3 嵌套查询

- [续上例]查询至少选修了学生200215122选修的全部课程的学生号码。

不存在这样的课程y，学生200215122选修了y，而学生x没有选。

```
SELECT DISTINCT Sno
  FROM SC SCX
  WHERE NOT EXISTS
        (SELECT *
         FROM SC SCY
         WHERE SCY.Sno = ' 200215122 '  AND
               NOT EXISTS
               (SELECT *
                FROM SC SCZ
                WHERE SCZ.Sno=SCX.Sno AND
                      SCZ.Cno=SCY.Cno));
```

学生200215122选修了y

学生x选修了课程y

数据库系统原理

教程：数据库系统概论（第5版）

结合：CMU 15-445/645 INTRO TO DATABASE SYSTEMS

华中科技大学 计算机学院

左琼

第三章 关系数据库标准语言SQL

Principles of Database Systems

第三章 课堂练习

1. 关于SQL语言，下列说法正确的是（）。

A. 数据控制功能不是SQL语言的功能之一。

B. SQL采用的是面向记录的操作方式，以记录为单位进行操作。

C. √ SQL是非过程化语言，用户无须指定存储路径。

D. SQL 作为嵌入式语言，语法与独立式的语言有较大的差异。

2. 设关系R（ABC）包含的数据如右图所示：

```
SELECT * FROM R
    WHERE C IN   (SELECT B FROM R);
```

A	B	C
30	20	NULL
10	NULL	30

该SQL语句的查询结果为：

A. √ NULL

B. (30,20,NULL)

3.3.4 集合查询

- 集合操作的种类
  - 并操作UNION
  - 交操作INTERSECT
  - 差操作EXCEPT
- 参加集合操作的各查询结果的列数必须相同；对应项的数据类型也必须相同

3.3.4 集合查询

[例] 查询计算机科学系的学生及年龄不大于19岁的学生。

方法一：

```
SELECT *
FROM Student
WHERE Sdept= 'CS'
UNION
SELECT *
FROM Student
WHERE Sage<=19;
```

方法二：

```
SELECT  DISTINCT  *
 FROM Student
 WHERE Sdept= 'CS'
        OR  Sage<=19;
```

- UNION：将多个查询结果合并起来时，系统自动去掉重复元组。
- UNION ALL：将多个查询结果合并起来时，保留重复元组

3.3.4 集合查询

[例] 查询选修课程1的学生集合与选修课程2的学生集合的交集

方法一：

```
SELECT Sno
  FROM SC
  WHERE Cno=' 1 '
  INTERSECT
  SELECT Sno
  FROM SC
  WHERE Cno='2 '
```

方法二：

```
SELECT Sno FROM SC
    WHERE Cno=' 1 ' AND Sno IN
            (SELECT Sno
              FROM SC
              WHERE Cno=' 2 ');
```

3.3.4 集合查询

[例] 查询计算机科学系的学生与年龄不大于19岁的学生的差集。

方法一：

```
SELECT *
  FROM Student
  WHERE Sdept='CS'
  EXCEPT
  SELECT  *
  FROM Student
  WHERE Sage <=19;
```

方法二：

```
SELECT *
  FROM Student
  WHERE Sdept= 'CS' AND  Sage>19;
```

3.3.5 基于派生表的查询

- 当子查询出现在FROM子句中，这时子查询生成临时派生表（derived table）成为主查询的查询对象。

[例] 找出每个学生超过他选修课程平均成绩的课程号。

```
SELECT Sno, Cno
FROM SC, (SELECT Sno, Avg(Grade) FROM SC
                Group by Sno)
              AS Avg_sc(avg_sno, avg_grade)
Where SC.sno = avg_sno
      AND grade > avg_grade;
```

必须为派生关系指定别名

总结：SELECT语句的分解

- 单表查询：

```
SELECT ……
FROM  t1
WHERE ……
……
```

- 多表连接查询：

```
SELECT ……
FROM  t1，t2, …
WHERE ……
 t1.col1 谓词 t2.col2
```

- 集合查询：

```
SELECT 查询块
集合操作
SELECT 查询块
```

- 派生表查询：

```
SELECT …
FROM (SELECT 子
查询) ……
```

- 嵌套查询：

```
SELECT ……
FROM ……
WHERE
   [ col1 谓词 (SELECT查询块) ]
   [ | EXISTS ( SELECT查询块) ]
   ……
```

```
SELECT ……
FROM ……
WHERE …
GROUP BY …
HAVING
 [ col1 谓词 (SELECT查询块) ]
 [ | EXISTS (SELECT查询块) ] ……
```

SELECT综合实例

假设银行数据库关系模式为：

银行： Branch=(Bname, city, assets)

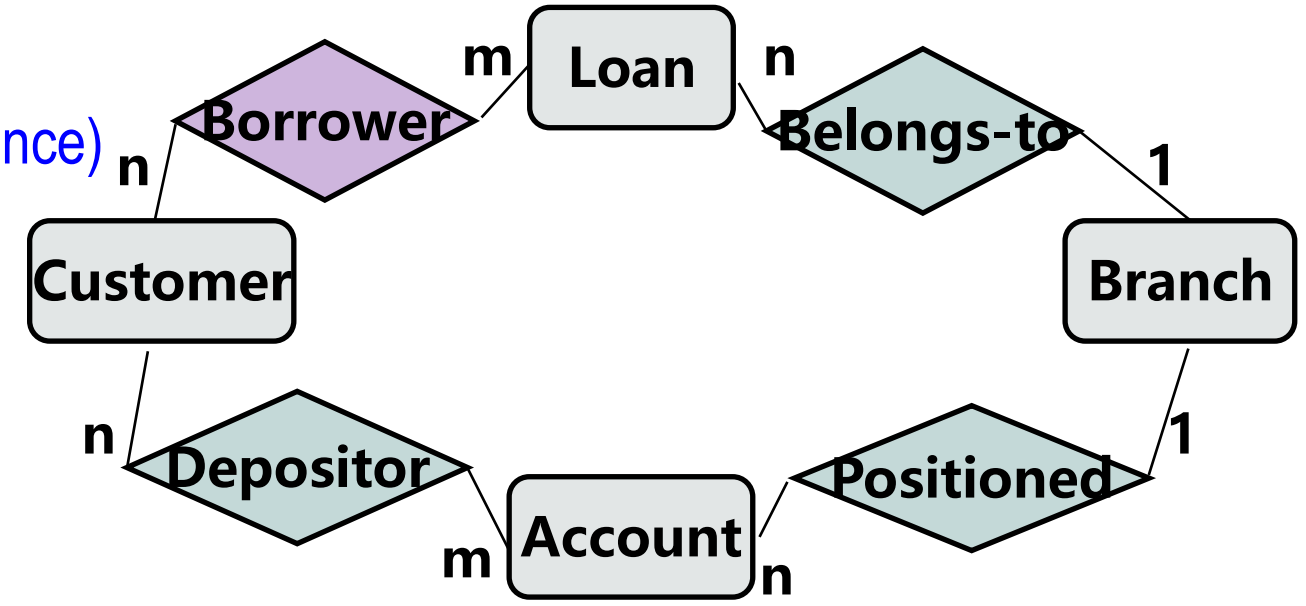
顾客： Customer=(Cno, Cname, street, city)

贷款： Loan=(Lno, Bname, amount)

借贷： Borrower=(Cno, Lno)

账户： Account=(Ano, Bname, balance)

存款： Depositor=(Cno, Ano)



SELECT综合实例

银行数据库关系模式为：

银行 Branch=(Bname, Bcity, Bassets)
顾客 Customer=(Cno, Cname, Cstreet, Ccity)
贷款 Loan=(Lno, Bname, amount)
借贷 Borrower=(Cno, Lno)
账户 Account=(Ano, Bname, balance)
存款 Depositor=(Cno, Ano)

- 思考题：

1） 找出在“Perry”银行有贷款的客户姓名及贷款数；
2） 找出资产至少比位于Brooklyn的某一家支行高的支行名；
3）找出银行中在Perry银行既有贷款又有账户的客户姓名；
4）找出平均余额最高的支行；
5）找出住在Harrison且在银行中至少有三个账户的客户的平均余额；
6）找出在Brooklyn的所有支行都有账户的客户；

SELECT综合实例

银行数据库关系模式为：

Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

□ 思考题解答：

1、找出在“Perry”银行有贷款的客户姓名及贷款数；

解法一：整体法 关键词（客户、有贷款、贷款）

1）找from　　customer ⋈ borrower ⋈ loan

2）用where过滤　Bname="perry"

3）用select投影　Cname,amount

```
SELECT  Cname,amount
    FROM   customer C, borrower B, loan L
    WHERE  C.Cno=B.Cno and B.Lno = L.Lno and Bname='perry';
```

SELECT综合实例

银行数据库关系模式为：
Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

1、找出在“Perry”银行有贷款的客户姓名及贷款数；

□ 解法二：分步法

1）找出在perry有贷款的所有Cno集合

用一个select块从borrower ⋈ loan 中查询；

2）从customer中选择其Cno IN 1）结果集的元组

```
select  Cname,amount
    from   customer C
    where  C.Cno  IN   (select  Cno
                        from    borrower B, loan L
                        where  B.Lno = L.Lno
                           and Bname='perry');
```

SELECT综合实例

银行数据库关系模式为：
Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

- 解法三：相关法

1）用exists函数构造一个能够判断任意一个顾客元组t，t.Cno是否在perry银行有贷款集合中的逻辑函数；

2）对customer中每个元组，调用exists判断

```
select  Cname,amount
      from   customer C
      where  EXISTS  (select  *
                      from   borrower B, loan L
                      where  C.Cno = B.Cno
                             and B.Lno = L.Lno
                             and Bname='perry');
```

SELECT综合实例

银行数据库关系模式为：
Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

- 思考题解答：

2、找出资产至少比位于Brooklyn的某一家支行多的支行名；

解法一：整体法 关键词（一家支行、位于Brooklyn的支行）

1）找from　　branch(角色T) × branch(角色S)

2）用where过滤　T.assets>S.assets and S.city="Brooklyn"

3）用select投影　T.Bname

```
select  T.Bname
   from   branch T, branch S
   where  T.assets > S.assets
              and S.city = 'Brooklyn';
```

SELECT综合实例

银行数据库关系模式为：

Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

2、找出资产至少比位于Brooklyn的某一家支行多的支行名；

解法二：分步法

1）找出位于Brooklyn的支行的总资产集合，用一个select块从branch中查询；

2）对上述结果用集合函数any()求出集合的任意一个

3）从branch中选择其总资产大于any()函数返回值的元组

```
select  Bname
from   branch
where  assets > any(select  assets
                    from   branch
                    where  city='Brooklyn');
```

SELECT综合实例

银行数据库关系模式为：
Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

- 思考题解答

3、找出银行中在Perry银行既有贷款又有账户的客户姓名；

解法一：使用交运算

1）找出在Perry银行有贷款的客户

2）找出在Perry银行有账户的客户

3）用intersect求交集

```
(select  distinct Cno
 from   borrower B, loan L
 where  B.Lno=L.Lno and Bname='Perry')
  intersect
(select  distinct Cno
 from   depositor D, account A
 where  D.Ano=A.Ano and Bname='Perry');
```

SELECT综合实例

银行数据库关系模式为：
Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

3、找出银行中在Perry银行既有贷款又有账户的客户姓名；

□ 解法二： 1）找出在Perry银行有贷款的客户（集合S1）

2）找出在Perry银行有账户的客户（集合S2）

3）对于集合S1中每个元组，判断是否属于S2

```
select  distinct Cno
from    borrower, loan
where  B.Lno=L.Lno and Bname='Perry'
          and Cno IN (select  distinct Cno
                              from   depositor, account
                              where  D.Ano=A.Ano and Bname='Perry');
或   (Bname,Cno) IN (select  distinct Bname,Cno
                              from   depositor, account
                              where  D.Ano=A.Ano );
```

SELECT综合实例

3、找出银行中在Perry银行既有贷款又有账户的客户姓名；

- 解法三：相关法

1）用exists函数构造一个能够判断任意一个顾客元组t，t.Cno是否在perry银行有账户的逻辑函数；

2）对borrower,loan中每个在perry银行有贷款元组，调用exists判断该元组是否有账户

```
select  Cname
    from   borrower B, loan L
    where  L.Bname='Perry' and B.Lno=L.Lno and
           EXISTS  (select  *
                    from   depositor D, account A
                    where  B.Cno = D.Cno
                           and D.Ano = A.Ano
                           and A.Bname='perry');
```

SELECT综合实例

银行数据库关系模式为：
Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

- 思考题解答

4、找出平均余额最高的支行；

分步法：

1）找出每个银行的平均余额集合，用一个select块和集函数avg()从account中查询；

2）对上述结果用集合函数all()求出平均余额的所有

3）从account中选择平均余额大于all()函数返回值的元组

```
select  Bname
from   account
group by  Bname
having (avg(balance) >=all(select  avg(balance)
                           from   account
                           group by  Bname));
```

Mysql

```
select  Bname
        from   account
        group by  Bname
order by avg(balance) desc
limit  1;
```

SELECT综合实例

- 思考题解答

5、找出住在Harrison且在银行中至少有三个账户的客户的平均余额；

解法：整体法 关键词（客户、有账户、账户余额）

1）找from　　customer ⋈ depositor ⋈ account

2）用where过滤　Ccity=“Harrison”　，得出住在Harrison且在银行中有账户的客户

3）用group分组 求出每个Harrison客户的账户数

4）用having 对每个分组过滤

```
select  C.Cno, avg(balance)
from   customer C, depositor D, account A
where  C.Cno=D.Cno and D.Ano = A.Ano  and C.city="Harrison"
group by D.Cno
having  count(D.Ano>=3)
```

银行数据库关系模式为：

Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

SELECT综合实例

银行数据库关系模式为：

Branch=(Bname, city, assets)
Customer=(Cno, Cname, street, city)
Loan=(Lno, Bname, amount)
Borrower=(Cno, Lno)
Account=(Ano, Bname, balance)
Depositor=(Cno, Ano)

6、找出在Brooklyn的所有支行都有账户的客户;

方法一：双重否定。

```
select  distinct S.Cno
from   Depositer S
where   not exists ( select  *
                from   Branch B
                where   Bcity=  'Brooklyn'
               and not exists
                    ( select  *
                      from   depositor T, account R
                      where   T.Ano=R.Ano
                              and  S.Cno=T.Cno
                              and B.Bname= R.Bname ));
```

不存在Brooklyn的某家支行，该客户在该支行没有账户

SELECT综合实例

6、找出在Brooklyn的所有支行都有账户的客户;

方法二：集合A包含集合B 等价于 not exists (B - A);

```
select  distinct  Cno
   from   depositor S
   where   not exists (( select  Bname
                         from   branch
                         where   Bcity='Brooklyn')
                         except
                        ( select  Bname
                        from   depositor T, account R
                        where   T.Ano=R.Ano
                                and  S.Cno=T.Cno));
```

不存在Brooklyn的某家支行，该客户在该支行没有账户

第三章 关系数据库标准语言SQL

3.4 数据更新

3.4.1 插入数据

插入数据是把新的记录插入到一个存在的表中。

1. 元组值的插入

语法：

```
INSERT INTO 基本表[(列名[, 列名]...)] VALUES(元组值);
```

作用：将一条元组值插入到表中。

注意：

- 列名的排列顺序不一定和表定义时的顺序一致。
- INTO子句中指定列名时，元组值的字段排列顺序必须和列名的排列顺序一致、个数相等、数据类型匹配。
- INTO子句中未指定列名时，元组值必须在每个属性列上均有值，且值的排列顺序与表中属性列的排列顺序一致。

3.4.1 插入数据

【例1】往基本表SC中插入一个元组值

```
INSERT INTO SC VALUES('S004','C011',90);
```

【例2】往基本表SC中插入部分元组值

```
INSERT INTO SC(sno, cno) VALUES('S004','C025');
```

注：对于INTO子句中没有出现的列，则新插入的记录在这些列上将取空值或缺省值，如上例的grade字段即赋空值。但有NOT NULL约束的属性列不能取空值。

Mysql

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES
(value1, value2, value3, ...),
(value1, value2, value3, ...),
...; //插入多行
```

3.4.1 插入数据

2. 查询结果的插入

将一个表中的数据抽取若干行插入另一表中，可以通过子查询来实现。

语法：

```
INSERT INTO <基本表名> [(<列名表)] 子查询;
```

作用：将子查询返回的结果数据集插入到表中。其功能是批量插入。

要求：查询语句的目标列必须与INTO子句匹配。

3.4.1 插入数据

【例3】将信息系（IS）平均成绩大于80分的学生学号和平均成绩存入另一个基本表S_GRADE(Sno, Avg_grade)。

```
INSERT INTO  S_GRADE (Sno, Avg_grade)
SELECT Sno, AVG(grade)
        FROM  SC
        WHERE Sno IN
              ( SELECT Sno
                FROM STUDENT
                WHERE sdept = 'IS')
        GROUP BY sno
        HAVING AVG(grade)>80;
```

3.4.1 插入数据

RDBMS在执行插入/更新语句时会检查所插元组是否破坏表上已定义的完整性规则：

- 实体完整性
- 参照完整性
- 用户定义的完整性
  - NOT NULL约束
  - UNIQUE约束
  - 值域约束

3.4.2 删除数据

语法：DELETE FROM <表名> [WHERE 条件表达式]

作用：从表中删除符合WHERE子句中删除条件的元组；若WHERE子句缺省，则表示要删除表中的所有元组。

【例4】删除学号为 ‘S001’的学生信息。

```
DELETE FROM STUDENT WHERE sno='S001';
```

【例5】删除所有学生信息。

```
DELETE FROM STUDENT;
```

不带条件的DELETE语句与DROP TABLE语句的区别：

- DROP TABLE语句删除表和表中的所有数据；
- DELETE语句只删除表中的数据，表仍然保留。

3.4.2 删除数据

□ 带子查询的删除语句

子查询同样也可以嵌套在DELETE语句中，用以构造执行删除操作的条件。

【例】 删除计算机科学系所有学生的选课记录。

```
DELETE  FROM SC
WHERE  Sno  IN  ( SELETE Sno
                  FROM Student
                  WHERE Sdept='CS');
```

3.4.3 更新数据

- 语法：

```
UPDATE  <基本表名>
   SET <列名> = 值表达式 [, <列名> = 值表达式…]
   [WHERE  条件表达式]
```

- 作用：对表中满足WHERE条件的元组，按SET子句修改相应列的值。若WHERE子句缺省，则表示对所有元组进行修改。

【例5】把所有学生的年龄加1。

```
UPDATE  STUDENT SET Sage = Sage+1;
```

【例6】把课程号为 ‘C5’的课程名改为“电子商务”。

```
UPDATE  COURSE  SET Cname='电子商务'  WHERE Cno = 'C5';
```

3.4.3 更新数据

□ 带子查询的修改语句

子查询也可以嵌套在UPDATE语句中，用以构造执行修改操作的条件。

例 将计算机科学系全体学生的成绩加10分。

```
UPDATE SC
SET grade=grade+10
WHERE  Sno  IN  ( SELETE Sno
                  FROM Student
                  WHERE Sdept='CS');
```

修改操作与数据库的一致性

UPDATE语句一次只能操作一个表。这会带来一些问题。

例如，学号为95007的学生因病休学一年，复学后需要将其学号改为96089，由于Student表和SC表都有关于95007的信息，因此两个表都需要修改，这种修改只能通过两条UPDATE语句进行。

- 第一条UPDATE语句修改Student表：

```
UPDATE Student
    SET Sno='96089'
    WHERE Sno='95007';
```

- 第二条UPDATE语句修改SC表：

```
UPDATE SC
    SET Sno='96089'
    WHERE Sno='95007';
```

必须保证这两条UPDATE语句要么都做，要么都不做。

事务

3.5 空值的处理

- 空值：不知道、不存在、或无意义的值。
- 空值的产生：
  - 插入 Insert into <表名> [(列名)] values (…, NULL, …);
  - 更新 Update <表名> SET <列名> = NULL;
- 空值的判断： IS NULL 、 IS NOT NULL
- 空值约束： NOT NULL，(Primary Key != Unique)
- 空值的算术运算、比较运算、逻辑运算
  - 空值与另一个值（包括空值）的算术运算结果为NULL；
  - 空值与另一个值（包括空值）的比较运算结果为UNKNOWN；
  - 带UNKNOWN的逻辑运算（3值逻辑）：
    - 任何值 AND UNKNOWN 为 UNKNOWN；
    - UNKNOWN OR UNKNOWN / FALSE 为 UNKNOWN；
    - NOT UNKNOWN 为 UNKNOWN

3.6 视图

视图是从一个或几个基本表（或视图）导出的一个虚表。

- 数据库中只存放视图的定义而不存放视图的数据，这些数据仍放在原来的基表中。当基表中的数据发生变化时从视图中查出的数据也随之改变了。
- 视图一经定义就可以对其进行查询，但对视图的更新操作有一定的限制。

3.7.1 视图的定义

1. 建立视图

语法：

```
CREATE  VIEW 视图名 [(列名[,列名]…)]
AS  子查询
[WITH  CHECK  OPTION]
```

3.6.1 视图的定义

说明：

视图的列名为可选项，缺省时，其列名即为子查询的查询目标列。但是在以下情况下，视图列名不可省略：

- 视图由多个表连接得到，在不同的表中存在同名列，则需指定列名；
- 当视图的列名为表达式或集函数的计算结果时，而不是单纯的属性名时，则需指明列名。
- 在子查询中不许使用ORDER BY子句，如果需要排序，则可在视图定义后，对视图查询时再进行排序。
- 如果指明了WITH CHECK OPTION选项，则在对视图进行更新时，所影响的元组必须满足视图定义中的WHERE条件。

3.6.1 视图的定义

- 行列子集视图：从一个基本表中导出，只是去掉了某些行或列(保留原表的主码)，这样的视图称为行列子集视图。
- 多表视图：从多个基本表或视图中导出。
- 带表达式的视图，即带虚拟列的视图。
- 分组视图，子查询带集函数和GROUP BY分组的视图。

3.6.1 视图的定义

【例1】建立计算机学院（CS）学生视图，并要求进行修改和插入操作时仍需保证该视图只有CS系的学生（单表视图）。

```
CREATE VIEW CS_STUDENT AS
        SELECT sno, sname
        FROM  STUDENT
        WHERE sdept = 'CS'
        WITH CHECK OPTION;
```

【例2】建立计算机学院选修5号课程的学生视图（多表视图）。

```
CREATE VIEW CS_S1(sno,sname,grade) AS
        SELECT STUDENT.sno,sname,grade
        FROM  STUDENT, SC
        WHERE sdept='CS'  AND
        STUDENT.sno=SC.sno  AND  cno='5';
```

3.6.1 视图的定义

【例3】建立计算机学院选修了5号课程且成绩在90分以上的学生视图（视图之上建视图）。

```
CREATE VIEW CS_S2 AS
        SELECT sno, sname, grade
        FROM  CS_S1
        WHERE grade >= 90;
```

【例4】建立学生出生年份的视图（表达式视图）。

```
CREATE VIEW student-year (sno, sname, sbirth)
AS
Select sno，  sname，  1999-sage
From student；
```

3.6.1 视图的定义

【例5】求学生平均成绩视图（分组视图）。

```
CREATE VIEW Sc-AVG(SNO,FAVG)
AS
Select sno，  AVG(grade)
from SC
Group By Sno;
```

【例6】将Student表中所有女生记录定义为一个视图（不指定属性列）

```
CREATE VIEW F_Stu(F_Sno,name,sex,age,dept)
 AS
 SELECT *  FROM  Student
 WHERE Ssex='女';
```

缺点：修改基表Student的结构后，Student表与F_Stu视图的映象关系被破坏，导致该视图不能正确工作。

3.6.1 视图的定义

2. 删除视图

□语法：

```
DROP  VIEW <视图名>  [CASCADE] ;
```

□作用：从数据库中删除一个视图的定义信息。

CASCADE表示把该视图和它导出的视图一起删除。

【例】撤消视图CS_S2。

```
DROP VIEW CS_S2;
```

注：视图删除后，只会删除该视图在数据字典中的定义，而与该视图有关的基本表中的数据不会受任何影响。

3.6.2 查询视图

- 用户角度：查询视图与查询基本表相同
- DBMS执行对视图的查询时，
  - 首先进行有效性检查，检查查询涉及的表、视图等是否在数据库中存在；
  - 如果存在，则从数据字典中取出查询涉及的视图的定义；
  - 把定义中的子查询和用户对视图的查询结合起来，转换成等价的基本表的查询；
  - 然后再执行这个经过修正的查询。

将对视图的查询转换为对基本表的查询的过程称为视图的消解（View Resolution）。

3.6.2 查询视图

例：查询计算机学院学习了5号课程且成绩为95分的学生学号和姓名。

方案1：从基本表中查询。

```
SELECT STUDENT.sno,sname
FROM  STUDENT, SC
WHERE STUDENT.sno=SC.sno AND sdept='CS'
AND cno='5' AND grade=95 ;
```

```
CREATE VIEW CS_S1(sno,sname,grade) AS
    SELECT STUDENT.sno,sname,grade
    FROM  STUDENT, SC
    WHERE sdept='CS'  AND
    STUDENT.sno=SC.sno  AND  cno='5';
```

方案2：利用视图CS_S1查询。

```
SELECT  sno, sname
FROM CS_S1
WHERE  grade=95;
```

由此可见，利用视图可以简化复杂的查询语句。

3.6.2 查询视图

- 视图消解法的局限
  - 有些情况下，视图消解法不能生成正确查询，需要特殊判断处理。

[例]在S_G视图中查询平均成绩在90分以上的学生学号和平均成绩

```
SELECT *
FROM   S_G
WHERE  Gavg>=90;
```

S_G视图的子查询定义：

```
CREATE VIEW S_G (Sno，Gavg)
  AS
   SELECT  Sno，AVG(Grade)
   FROM  SC
   GROUP BY Sno;
```

查询执行转换：

错误：

```
SELECT Sno，AVG(Grade)
FROM    SC
WHERE  AVG(Grade)>=90
GROUP BY Sno;
```

正确：

```
SELECT  Sno，AVG(Grade)
FROM  SC
GROUP BY Sno
HAVING AVG(Grade)>=90;
```

3.6.3 更新视图

- 视图的更新操作包括插入、修改和删除数据，其语法格式如同对基本表的更新操作一样。
- 由于视图是一张虚表，所以对视图的更新，最终实际上是转换成对基本表的更新。

```
CREATE VIEW CS_STUDENT AS
    SELECT sno, sname
    FROM  STUDENT
    WHERE sdept = 'CS'
    WITH CHECK OPTION;
```

[例] 对视图CS_STUDENT的更新：

```
INSTER  INTO CS_STUDENT VALUES ('04008' , '张立'  );
```

在执行时将转换为对表STUDENT的更新：

```
INSERT INTO STUDENT
            VALUES ('04008', '张立'  , NULL, NULL, 'CS');
```

3.6.3 更新视图

- 更新视图的限制：一些视图是不可更新的，因为对这些视图的更新不能唯一地有意义地转换成对相应基本表的更新。

```
CREATE VIEW S_G (Sno，Gavg)
  AS
    SELECT Sno，AVG(Grade)
    FROM SC
    GROUP BY Sno；
```

例：视图S_G为不可更新视图。

```
UPDATE S_G
SET   Gavg=90
WHERE Sno= '200215121';
```

这个对视图的更新无法转换成对基本表SC的更新。

3.6.3 更新视图

视图更新的约束：

根据视图的定义可将视图分为可更新视图和不允许更新的视图。如下列情况的视图为不允许更新的视图：

- 字段由表达式或常数组成，不允许执行INSERT和该字段上的UPDATE，但可以执行DELETE；
- 字段由集函数组成，不允许更新；
- 视图定义含有GROUP BY或DISTINCT，不允许更新；
- 有嵌套查询，且内外查询使用相同表时，不允许更新；
- 定义中有多表联接时，不允许更新；
- 由不允许更新的视图导出的视图，不允许更新。

3.6.3 更新视图

视图的作用：

- 简化用户的操作
- 使用户能以多种角度看待同一数据
- 对重构数据库提供一定程度的逻辑独立性
- 是数据共享与保密的一种安全机制
- 适当的利用视图可以更清晰的表达查询

课堂练习

1.（多选题）下列说法不正确的是（ ）

A. 基本表和视图一样，都是关系

B. 可使用SQL对视图进行操作，视图都可以进行插入

C. √ 可以从多个基本表或视图上定义视图

D. √ 基本表和视图中都存储数据

小结

- SQL语言概述（SQL与关系代数、关系演算是等价的；SQL的5大特点）。
- SQL语言具有数据定义、数据查询、数据更新、数据控制4大功能。数据定义语句（请注意与关系数据库的理论定义（数据结构、完整性约束）相对照。数据查询功能最为丰富和复杂。
- 进一步介绍了索引和视图的概念及其作用。
- SQL学习，应反复上机加强练习。
  - MySQL 8.0

- 本章作业：P123 4，9，随堂作业改为附加题

（第3，4，5章作业一起发布）

SQL随堂练习

考虑下面关系数据库：　公司 com (cno, cname， city)

员工 emp (eno, ename, street, city, mno)

;mno 是外键，引用emp(eno)，表示该员工的上级经理

工作 works (eno, cno, salary)

请用关系代数表达式表示下面查询?

1）找出与其经理居住在同一个城市的所有员工姓名和他的经理姓名。

2）找出“DM”公司中不在其分公司（编号为1001）工作的所有员工编号和姓名。

3）找出比公司（编号为1002）的所有员工收入都高的公司名和员工姓名。

4）假设公司可以在几个城市部署分部(cname相同，但cno不同)。找出在“DM”公司所在的各个城市都有分部的公司。