

# 第3章 关系数据库标准语言SQL

## 3.1 关系DML回顾

### 1、ISBL (Information System Base Language)

- ① 研制：IBM英国研究中心
- ② 特征：纯关系式数据库（每个访问语句均近似于一个关系代数表达式）

### 2、ALPHA

- ① 提出：埃德加.科德 (Edgar Frank Codd) <未实现>
- ② 特征：元组关系演算

### 3、QUEL (Query Language)

- ① 研制：美国加利福尼亚大学
- ② 特征：元组关系演算（参照ALPHA，引进谓词演算到关系时，去掉 $\exists$ 、 $\forall$ ）

#### 4、QBE (Query By Example)

- ① 研制：IBM (1975年提出，1978年IBM370上实现)
- ② 特征：域关系演算语言 (表格界面)

#### 5、SQL (Structured Query Language)

- ① 研制：IBM, ...
- ② 特征：介于关系代数与关系演算之间的语言

# SQL发展里程碑

日期	事件
1970	Codd定义关系数据库模式
1974	IBM开始其System/R计划
1974	首篇文章描述SEQUEL语言 ( <b>S</b> TUCTURED <b>E</b> NGLISH <b>Q</b> UERY <b>L</b> ANGUAGE)
1978	System/R用户试验，引入SEQUEL
1979	Oracle首次引入商用RDBMS
1981	关系技术引入Ingres
1981	IBM公布产品SQL/DS， SEQUEL变为SQL

1982	ANSI成立SQL标准委员会
1986	ANSI批准SQL标准
1986	Sybase引入用于事务处理的RDBMS
1987	ISO批准 SQL标准
1989	首次提供用于OS/2的SQL数据库服务器
1989	ISO对SQL86进行了补充，推出了SQL89标准
1991	公布SQL Access Group规程

1992	Microsoft公布ODBC规程
1992	首次提供用于Netware的SQL数据库服务器
1992	ANSI批准 SQL2（SQL92）标准
1993	首次提供ODBC产品
1999	SQL99（也称为SQL3）增加了抽象数据类型功能。

# SQL的产生

- 1974年，IBM的RayBoyce和DonChamberlin将Codd关系数据库的12条准则的数学定义以简单的关键字语法表现出来，里程碑式地提出了SQL (Structured Query Language)语言。
- SQL的功能包括查询、操纵、定义和控制。

**综合的、通用的**关系数据库语言

**高度非过程化**（只要求用户指出做什么而不需要指出怎么做）。

集成实现了数据库**生命周期中**的全部操作。

提供了与关系数据库交互的方法，可以与标准的编程语言一起工作。

# SQL标准

- SQL语言标准的每一次变更都指导着关系数据库产品的发展方向。然而，直到二十世纪七十年代中期，关系理论才通过SQL在商业数据库Oracle和DB2中使用。
- 1986年，ANSI把SQL作为关系数据库语言的美国标准，同年公布了标准SQL文本。目前SQL标准有3个主要的版本。

# SQL-89

- 基本SQL定义是ANSIX3135-89, “Database Language-SQL with Integrity Enhancement”[ANSI89, 一般叫做SQL-89]。
- SQL-89描述了模式定义、数据操作和事务处理。
- SQL-89和随后的ANSIX3168-1989, “Database Language-Embedded SQL”构成了第一代SQL标准。



# SQL92

- ANSI X3.135-1992 [ANSI92] 描述了一种增强功能的SQL，现在叫做SQL-92标准。
- SQL-92 包括模式操作，动态创建和SQL语句动态执行、网络环境支持等增强特性。

# SQL3

- 在完成SQL-92标准后，ANSI和ISO即开始合作开发SQL3标准。
- SQL3的主要特点在于抽象数据类型的支持，为新一代对象关系数据库提供了标准。

## 3.2 SQL概述

### 1、特点

#### ① 一体化

DDL—— Data Description Language

DML—— Data Manipulate Language

DCL—— Data Control Language

（三种功能可在系统不间断的情况下交替执行；  
风格统一）

#### ② 两种使用方式（交互、嵌入）

SQL Server Enterprise Manager - [控制台根目录\Microsoft SQL Servers\SQL Server 组\PANPENG (Windows NT)]

控制台(C) 窗口(W) 帮助(H)

操作(A) 查看(V) 工具(T)

树

- PANPENG (Windows NT)
  - 数据库
    - hust37041222
    - hustmisrun
    - hustmisrun0412
    - hustmisrun3904
    - hustmisrun\_loc
    - master
    - model
    - msdb
    - Northwind
    - pubs
    - tempdb
    - 表
    - 视图
    - 存储过程
    - 用户
    - 角色
    - 规则
    - 默认值

表 21 个项目

名称	所有者	类型	创建日期
dtproperties	dbo	系统	2005-3-1
student	dbo	用户	2005-3-1
syscolumns	系统	系统	2000-8-6
syscomments	系统	系统	2000-8-6
sysdepends	系统	系统	2000-8-6
sysfilegroups	系统	系统	2000-8-6
sysfiles	系统	系统	2000-8-6
sysfiles1	系统	系统	2000-8-6
sysforeignkeys	系统	系统	2000-8-6
sysfulltextcatalog	系统	系统	2000-8-6
sysfulltextnotify	系统	系统	2000-8-6
sysindexes	系统	系统	2000-8-6
sysindexkeys	系统	系统	2000-8-6
sysmembers	系统	系统	2000-8-6
sysobjects	dbo	系统	2000-8-6
syspermissions	dbo	系统	2000-8-6
sysproperties	dbo	系统	2000-8-6
sysprotects	dbo	系统	2000-8-6

右键菜单:

- 新建表(B)...
- 设计表(S)
- 打开表(O)
  - 返回所有行(A)
  - 返回首行(I)...
  - 查询(Q)
- 全文索引表(F)
- 所有任务(K)
- 复制(C)
- 删除(D)
- 重命名(M)
- 属性(R)
- 帮助(H)


# SQL Server Enterprise Manager - [2:表“student”中的数据，位置

控制台(C) 窗口(W) 帮助(H)


SQL

	name	birthyear
▶	李四	1990
	王五	2000
	张三	1980
*		

## 交互式1

 SQL Server Enterprise Manager - [2:表"student"中的数据，位置是

控制台(C) 窗口(W) 帮助(H)



```
SELECT name, birthyear  
FROM student
```

	name	birthyear
▶	李四	1990
	王五	2000
	张三	1980
✱		

## 交互式2

SQL 查询分析器

文件(F) 编辑(E) 查询(Q) 工具(T) 窗口(W) 帮助(H)

tempdb

查询 — PANPENG.tempdb.PANPENG\Administrator — 无标题1\*

```
select name,birthyear
from student
order by birthyear
```

	name	birthyear
1	张三	1980
2	李四	1990
3	王五	2000

网格 消息

批查询完成。 PANPENG (8.0) PANPENG\Administrator (52) tempdb 0:00:00 3行 行 3, 列 19

Microsoft SQL Server Management Studio

File Edit View Project Debug Tools Window Community Help

New Query

Execute

Object Explorer

Connect

(local) (SQL Server 10.50.1617 - panpl-THINK\panpl)\Databases\pptest\Tables

Name	Schema	Create Date	Policy Health State
System Tables			
Table_person	dbo		
student	dbo		
sc	dbo		

Table\_person

Schema: dbo

Create Date: 2013/4/22 21:23

File Group: PRIMARY

Replicated: False

Properties

Aggregate Status

Connection

Elapsed time 00:00:00.436

Finish time 2017/3/29 1:18:00

Name (local)

Rows returned 1

Start time 2017/3/29 1:18:00

State Open

Connection

Connection (local) (panpl-THINK\panpl)

Connection Details

Connection 00:00:00.436

Connection 2017/3/29 1:18:00

Connection 1

Connection 2017/3/29 1:18:00

Connection Open

Display name (local)

Login name panpl-THINK\panpl

Server name (local)

Server version 10.50.1617

Session Trace

SPID 53

Name

The name of the connection.

Ready





Microsoft SQL Server Management Studio

File Edit View Query Project Debug Tools Window Community Help

New Query

ppptest Execute

Object Explorer

Connect

(local) (SQL Server 10.50.1617 - p...

Databases

System Databases

Database Snapshots

ppptest

Database Diagrams

Tables

System Tables

dbo.sysdiagrams

dbo.Table\_person

dbo.student

dbo.sc

Views

Synonyms

Programmability

Service Broker

Storage

Security

ReportServer

ReportServerTempDB

Security

Server Objects

Replication

Management

SQL Server Agent

SQLQuery3.sql - (L...HINK\panppl (54))\* SQLQuery2.sql - (L...HINK\panppl (55))\* SQLQuery1.sql - (L...-THINK\panppl (53))

```
select student.sno, sname, cno, score
from student, sc
where student.sno=sc.sno
and cno='0123'
```

Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

select student.sno,sname,cno,score from student,sc where student.sno=sc.sno and cno='0123'

Execution Plan Diagram:

- SELECT (Cost: 0 %)
- Nested Loops (Inner Join) (Cost: 0 %)
  - Clustered Index Scan (Cluste... [student].[PK\_student] (Cost: 50 %)
  - Clustered Index Seek (Cluste... [sc].[PK\_sc] (Cost: 50 %)

Query executed successfully. (local) (10.50 RTM) panppl-THINK\panppl (55) ppptest 00:00:00 0 rows

Output

Ready

Ln 4 Col 15

Microsoft SQL Server Management Studio

File Edit View Query Project Debug Tools Window Community Help

New Query | [Icons]

pptest | Execute [Icons]

Object Explorer

- Connect [Icons]
- (local) (SQL Server 10.50.1617 - p...
- Databases
  - System Databases
  - Database Snapshots
  - pptest
    - Database Diagrams
    - Tables
      - System Tables
        - dbo.sysdiagrams
        - dbo.Table\_person
        - dbo.student
        - dbo.sc
      - Views
      - Synonyms
      - Programmability
      - Service Broker
      - Storage
      - Security
      - ReportServer
      - ReportServerTempDB
      - Security
      - Server Objects
      - Replication
      - Management
      - SQL Server Agent

SQLQuery3.sql - (\\...HINK\panppl (54))\* SQLQuery2.sql - (\\...HINK\panppl (55))\* SQLQuery1.sql - (\\...THINK\panppl (53))

```
select sno,sname
from student
where exists (
select *
from sc
where sc.cno='0123'
and sc.sno=student.sno)
```

Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

select sno,sname from student where exists ( select \* from sc where sc.cno='0123' and sc.sno=student...

Execution Plan Diagram:

- SELECT (Cost: 0 %)
- Nested Loops (Inner Join) (Cost: 0 %)
  - Clustered Index Scan (Clustered) [student].[PK\_student] (Cost: 50 %)
  - Clustered Index Seek (Clustered) [sc].[PK\_sc] (Cost: 50 %)

Query executed successfully. (local) (10.50 RTM) panppl-THINK\panppl (54) pptest 00:00:00 0 rows

Output

Ready

嵌入式

Main(){

...

exec sql begin declare section;

char co[10];

int id;

exec sql end declare section;

...

exec sql select company\_name  
from customer  
where id = :id  
into co;

...

}



③统一结构

④高度非过程化

用户无需了解与涉及存取路径

⑤语言简洁（方便易学）

标准动词7个：create, alter, select, delete, update, insert, drop

扩充2个：grant、revoke

集合操作：  
UNION并  
INTERSECT交  
EXCEPT差

SQL功能	操作符
数据查询	SELECT
数据定义	CREATE, ALTER, DROP
数据操纵	INSERT, UPDATE, DELETE
数据控制	GRANT, REVOKE

## 2、功能

① 定义； ② 查询； ③ 更新； ④ 控制（安全、完整性、一致性）

DDL

DML

DCL

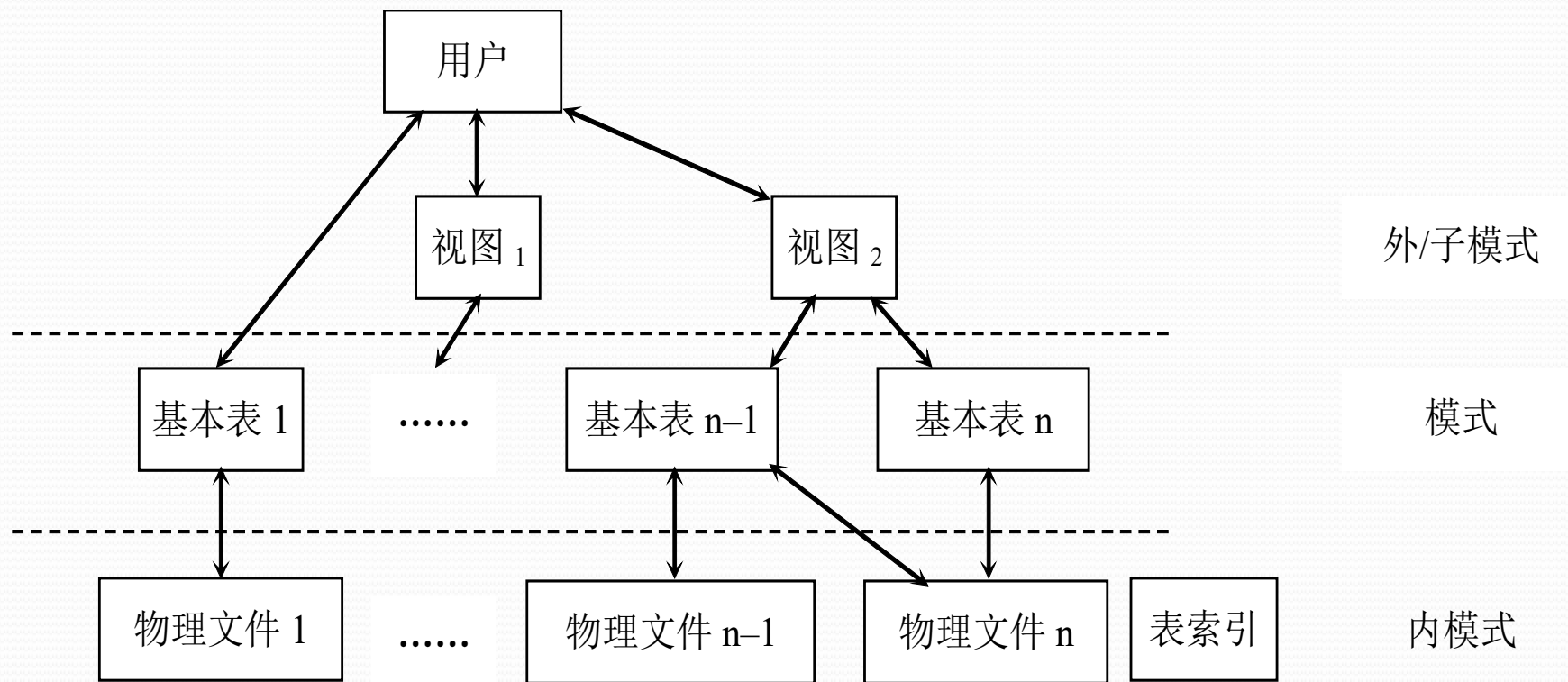
## 基本术语

- **基表Basetable**: 一个关系，是模式的内容，由结构与数据记录两部分组成。
- **视图View**: 从基表中导出的表，在数据库中只有定义，对应数据存放在基表中，视图属于外模式，每次使用时由**DBMS**根据视图的定义进行相应的转换并执行。
- **游标CURSOR**: 在内存中生成的临时表，存放游标包含的**sql**语句的执行结果，便于主语言使用，一次通过**fetch**操作读取一条记录，而非记录的集合。游标的当前值指向临时表中当前正在被处理的元组，相当于一个指示器。

## 基本术语（续）

- 集函数：**MAX**（列名），**MIN**（列名），**AVG**(列名)，**COUNT**（\*/列名/常数），**SUM**（列名）。
- 子查询：嵌套在查询语句中的查询。
- 子句：一条**sql**语句的某个语法部分，例如**select**子句、**from**子句、**where**子句。
- 应用程序代码中的结果集(**RecordSet**，应用程序的一种缓存机制)：返回给应用程序的查询结果的集合。

### 3、支持三级模式



基本表与物理文件可以一一对应（存储文件、索引文件可多个）



### 3.3 数据库定义与维护

#### 一、功能

- ① 基本表定义与修改； ② 模式定义； ③ 索引的定义与删除；
- ④ 视图的定义与撤消

#### 3.3.1 模式的建立与删除

**CREATE SCHEMA** <模式名> **AUTHORIZATION** <用户名>  
>

例： **CREATE SCHEMA "S-T" AUTHORIZATION WANG;**  
      **CREATE SCHEMA AUTHORIZATION WANG;** 隐含制定模式名为WANG

语法扩充:

**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>]

删除模式

**DROP SCHEMA** <模式名> <CASCADE|RESTRICT>

关于SCHEMA:

教材中的含义、SQL SERVER的含义、ORACLE的含义。

命名空间的作用，和三层模式的区别与联系。

### 3.3.2 表定义、修改与删除

#### 一、定义基本表

##### 1、格式

`create table 表名 (列名1, 类型[NOT NULL] [, 列名2, 类型[NOT NULL]].....) [其它参数]`

2、功能：定义一个基本表：表名、列名、类型、完整性约束  
(key, not NULL ...)

##### 3、说明

- 其它参数与物理存贮相关，由具体DBMS决定
- 一个表一个create table

例子：创建课程种类表（类型编号、类型名称），

```
CREATE TABLE [dbo].[ctype] (  
    [ctпно] [char] (2) NOT NULL PRIMARY KEY,  
    [title] [char] (2) NOT NULL UNIQUE  
)
```



这里主码只含有一个属性，可以直接在表的定义语句中指定该属性为主码。

公共基础课程、  
专业基础课程、  
专业核心课程

例子：创建关系“教室表”，拥有属性：“教室编号、教室名称、教学楼编号、教室容量、教室资源配置、使用对象类型”。

```
CREATE TABLE [dbo].[classroom] (  
    [rmno] [char] (5) NOT NULL PRIMARY KEY,  
    [rmtitle] [varchar] (30) NULL ,  
    [buildno] [char] (4) NOT NULL ,  
    [quantity] [int] NOT NULL ,  
    [resource] [char] (2) NOT NULL ,  
    [usertype] [char] (2) NULL  
)
```



例：定义选课关系SC：

学生编号S#，char 4；课程编号C#，char 4，成绩Grade  
smallint；

主关键字：S#，C#；外键：关系S的主码S#，关系C的主码C#；

约束：0<=成绩<=100 或者为Null

```
CREATE TABLE SC  
  ( S#    CHAR (4) ,  
    C#    CHAR (4) ,  
    GRADE SAMLLINT,  
    PRIMARY KEY (S#, C#),  
    FOREIGN KEY (S#)  
      REFERENCES S(S#),  
    FOREIGN KEY (C#)  
      REFERENCES C(C#),  
    CHECK((GRADE IS NULL) OR  
          GRADE BETWEEN 0 AND 100)
```

)

## 二、表修改

### 1、格式（增加列）

**ALTER Table 表名**

**add 列名 类型**

例：ALTER Table car

add [emission] [float] NOT NULL;

例：ALTER Table student

add 性别 char(2)

### 2、功能：给基本表增加一个属性

### 3、说明：（可增加多个属性）

例: **BEGIN TRANSACTION;**  
**ALTER Table student add**  
    [city] [varchar] (20) NOT NULL ,  
    [province] [varchar] (20) NOT NULL ;  
**GO;**  
**COMMIT;**



## 修改表（删除列）

删除列 “drop column 列名

例如: **BEGIN TRANSACTION;**

```
ALTER TABLE dbo.classroom
```

```
DROP COLUMN isopen, isused;
```

```
GO;
```

```
COMMIT;
```

## 修改表（修改列）

```
ALTER TABLE classroom
```

```
ALTER COLUMN roomtype varchar(30);
```

## 修改表（续）

- 增加完整性约束

```
ALTER TABLE classroom  
    ADD UNIQUE(rmno);
```

- 删除完整性约束

```
ALTER TABLE classroom  
    DROP PK_classroom;
```

一个复杂的例子（涉及到已有的数据）

将教室表中的教室名称列从30位变长字符串改为20位变长字符串。

BEGIN TRANSACTION;——开始事务

CREATE TABLE dbo.Tmp\_classroom——创建临时表

(

[rmno] [char] (5) NOT NULL PRIMARY KEY,

[rmtitle] [varchar] (20) NULL ,

[buildno] [char] (4) NOT NULL ,

[quantity] [int] NOT NULL ,

[resource] [char] (2) NOT NULL ,

[usertype] [char] (2) NULL );

GO;——执行创建语句

```
IF EXISTS(SELECT * FROM dbo.classroom)——如果原表有数据
EXEC('INSERT INTO dbo.Tmp_classroom ——拷贝数据
(rmno,rmtitle,buildno,quantity,resource,usertype)
SELECT
rmno,CONVERT(varchar(20),rmtitle),buildno,quantity,
resource,usertype FROM dbo.js TABLOCKX');
GO;——执行拷贝
DROP TABLE dbo.js;——删除原表
GO;——执行
EXECUTE sp_rename N'dbo.Tmp_classroom', N'classroom',
'OBJECT';——表对象改名，临时表改为正式表
GO;——执行改名
COMMIT;——事务完成
```

### 三、表删除

- 1、格式：Drop table 表名[RESTRICT | CASCADE]
- 2、功能：撤消一个基本表
- 3、说明：撤消结构、元组值、索引、相关视图、释放相关空间

### 3.3.3、索引建立与删除

针对某一个关系的某个或者些属性上的条件查询（选择运算、连接运算），建立辅助数据结构，以加快查询

# 树索引 (Tree Index)

## 数据库管理系统内部的数据结构

- 内部元数据
- 核心数据存储
- 临时数据结构
- 表索引 (table index)

表索引是表属性子集的副本，是对这些属性的组织及排序，目的是为了使用这些属性的子集进行高效的访问（主要针对不是顺序扫描的访问方式）。

## DBMS关于索引的任务：

- 确保表的内容和索引在逻辑上是同步的；
- 执行每个查询时找出最佳的索引；
- 在索引的数量和开销上进行权衡：
  - 存储开销
  - 维护开销

关系数据库的常见索引结构：B+树



## B树概述

B树：一种特定的数据结构，具有平衡的多分树结构。

“B树家族”，可用于泛指一类平衡树的数据结构

→ B-Tree (1971)

→ B+Tree (1973)

→ B\* Tree (1977)

→ B<sup>link</sup>-Tree (1981)

## B+树

B+树是一种自平衡的树型数据结构，它保持数据**排序**，支持在 **$O(\log n)$** 的复杂度内进行搜索、顺序访问、插入和删除。

它是二叉搜索树的一种泛化，一个结点可以有两个以上的子结点。

针对读写大数据块的系统，B树的优化。

## B+树性质

B+树作为一种**M**路搜索树，具有以下属性：

- 完美平衡（即每个叶结点在树中处于相同的深度）；
- 除了根结点，每个结点至少是半满的；  
( $M/2-1 \leq \#keys \leq M-1$ )
- 每个有**k**个键的内部结点有**k+1**个非空子结点。

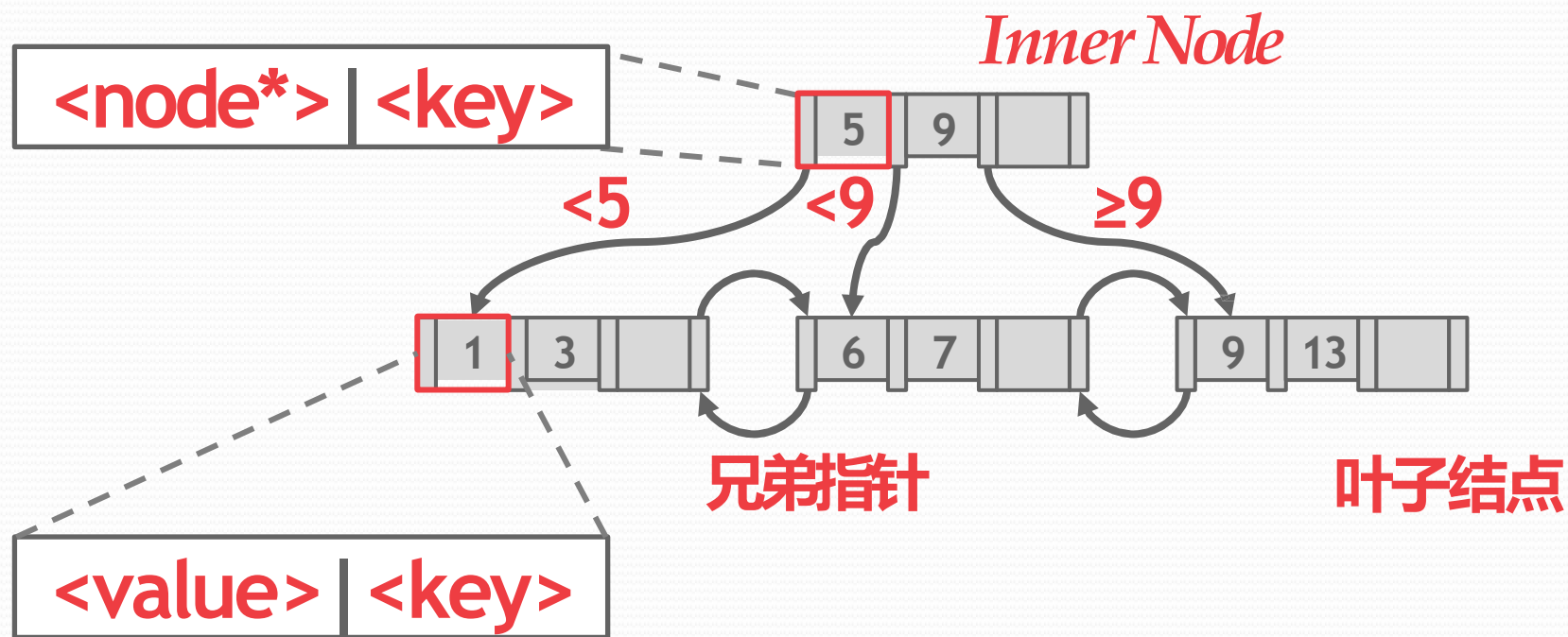
## B+树结点

每个B+树结点由一个“**键-值**”数组组成，其中：

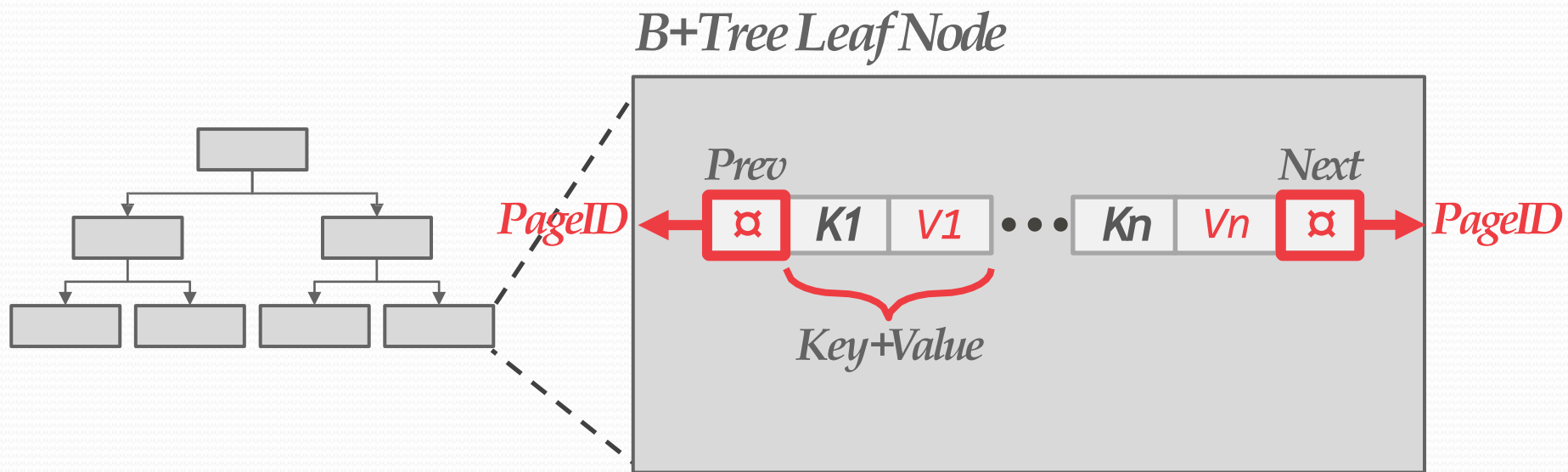
- 键是从索引所基于的属性产生的；
- 值则依据结点是内部结点还是叶子结点而有所不同。

节点内的键值数组通常会**按照键的顺序排序**。

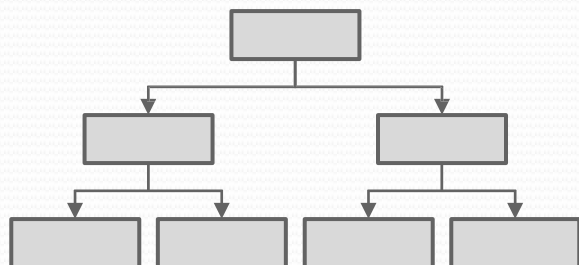
## B+树示例



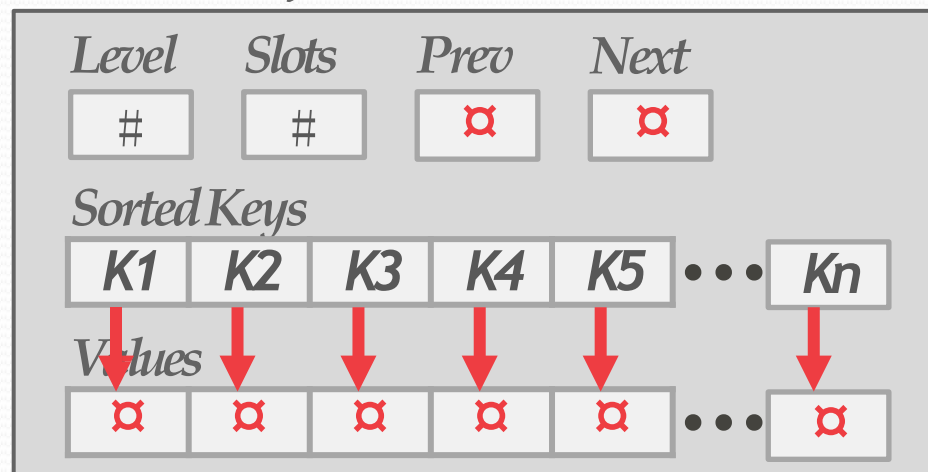
## B+树叶子结点



# B+树叶子结点



## B+Tree Leaf Node



## B+树叶子结点的值

方法一：记录ID

指向索引项所对应元组位置的指针。

方法二：元组数据

元组的实际数据存储在叶子结点中；

二级索引则必须采用将记录ID作为叶节点值的方式。





## B-树 vs. B+树

1972的原始B-树在**所有结点中存储“键+值”**，每个键只在树中出现一次，存储空间更高效。

B+树**只在叶子结点中存储值**，内部结点仅用于引导搜索过程。

## 选择条件 (Selection Conditions)

如果查询提供了B+树搜索关键字的**任何属性**的值，DBMS就可以使用B+树索引。

例：<a, b, c>上的索引，可以支持的查询条件

(a=5 and b=3) ;

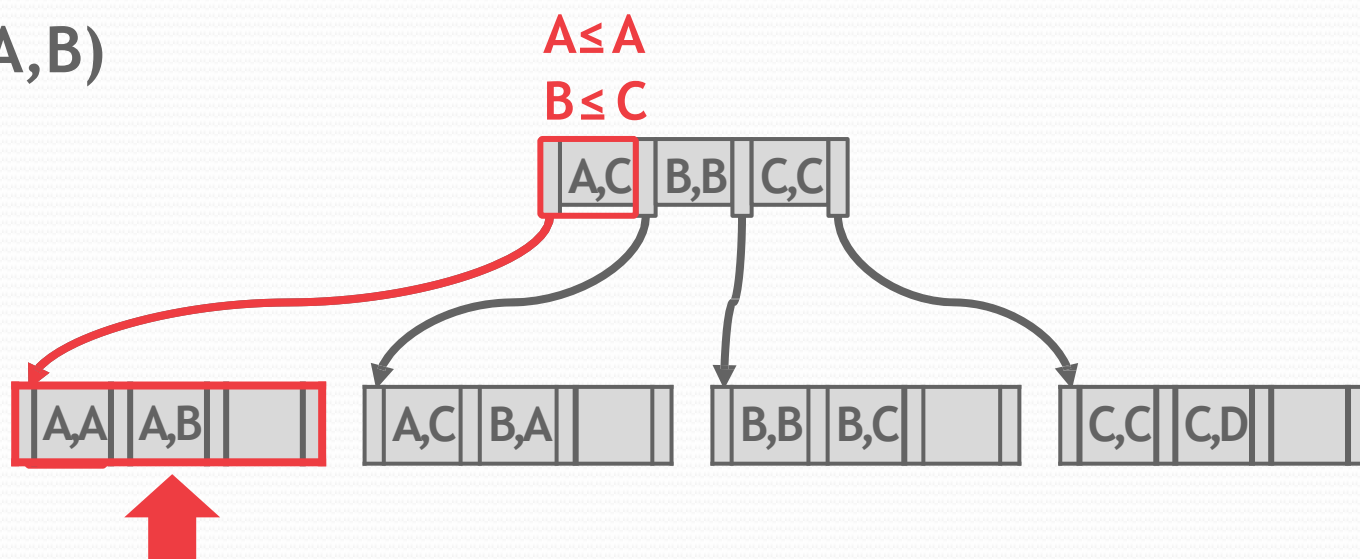
(b=3)

并非所有DBMS都支持如此。

如果采用哈希索引，则需要搜索关键字中的所有属性。

## 选择条件 (Selection Conditions)

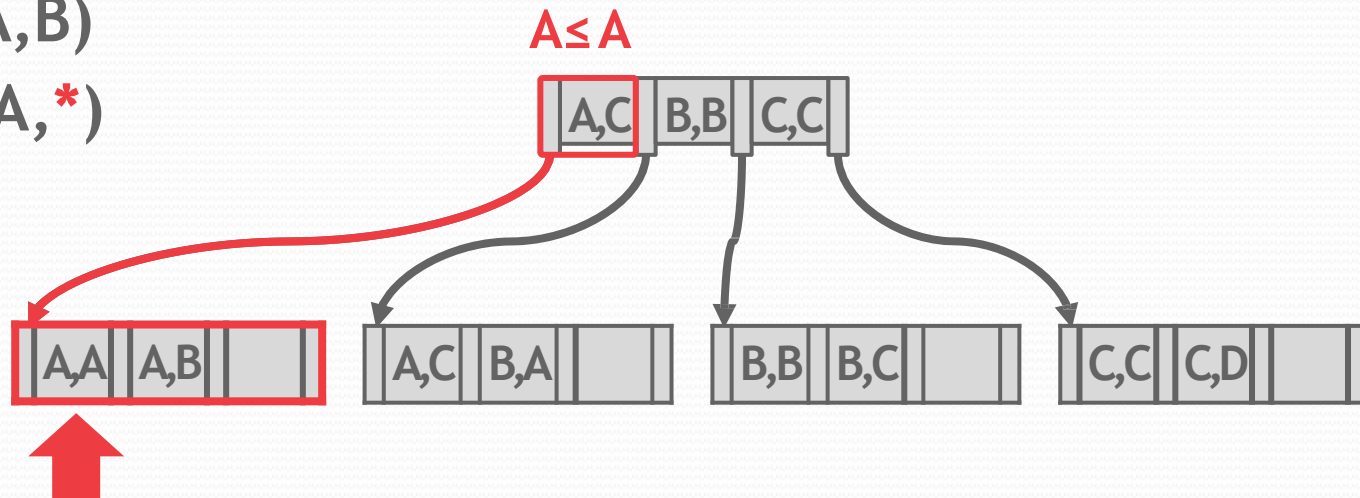
Find Key=(A,B)



## 选择条件 (Selection Conditions)

Find Key=(A,B)

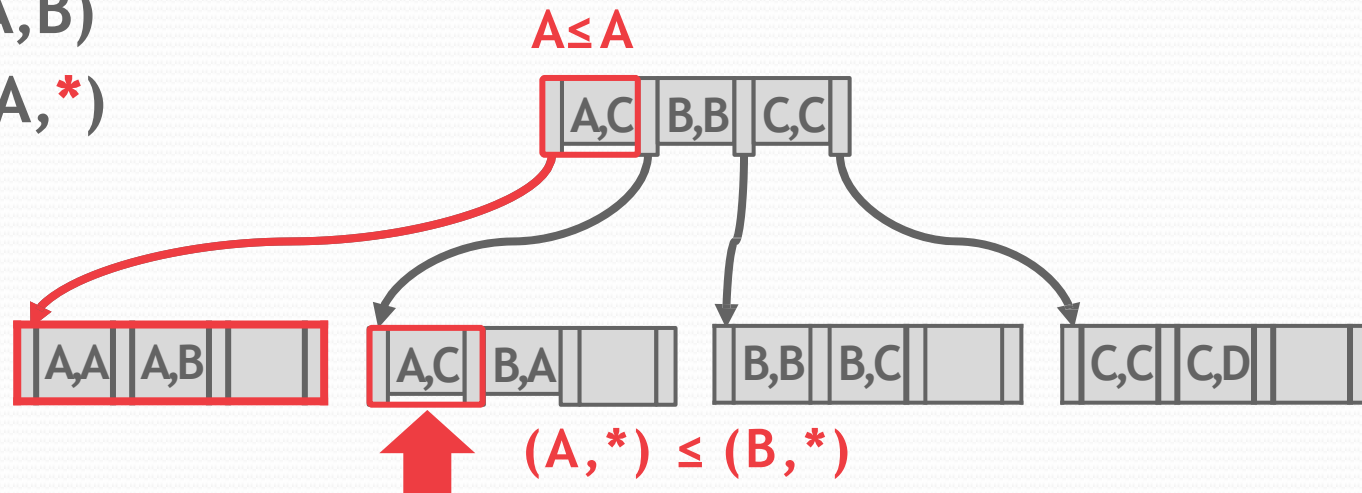
Find Key=(A,\*)



## 选择条件 (Selection Conditions)

Find Key=(A,B)

Find Key=(A,\*)

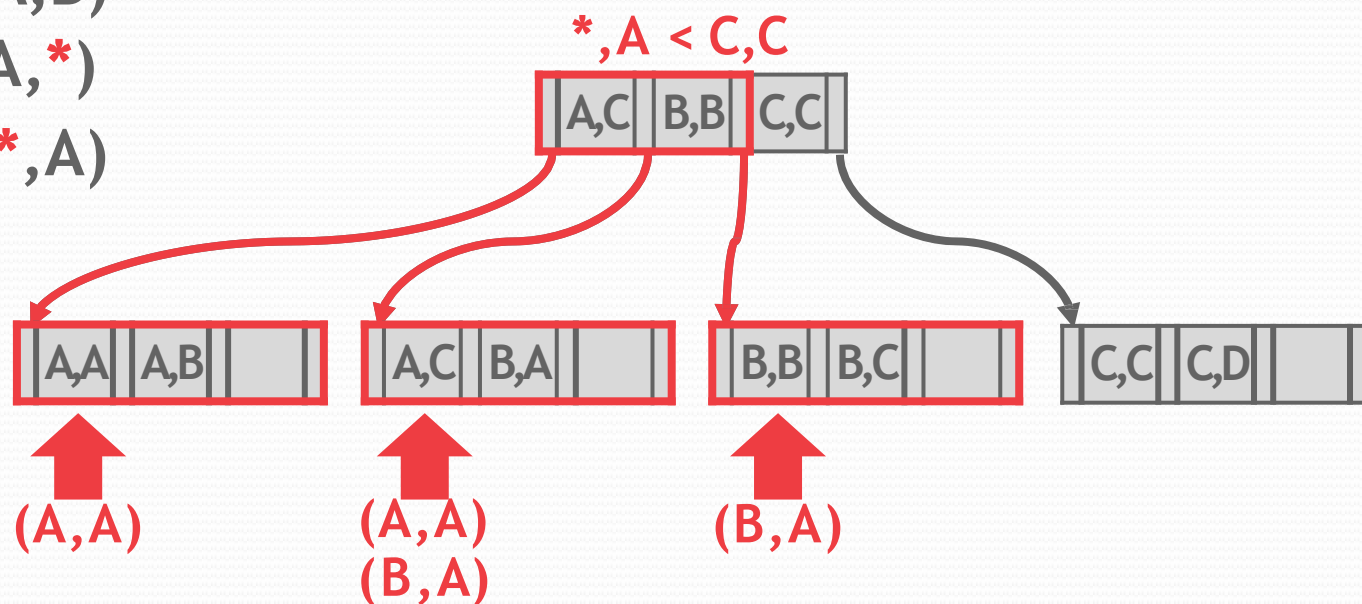


## 选择条件 (Selection Conditions)

Find Key=(A,B)

Find Key=(A,\*)

Find Key=(\*,A)



## B+树-插入操作

- (1) 通过查找确定叶子结点**L**;
- (2) 将排序后的关键字插入结点**L** ;
- (3) 如果空间充足（即结点**L**含有的关键字数目小于阶数），则插入结束；

否则将结点**L**分裂为左右两个叶子结点（**L**和**L<sub>2</sub>**）：

- 重新分配关键字需要使其平衡，因此将结点中间的关键字进位到父结点中；
- 将指向结点**L<sub>2</sub>**的指针插入**L**的父结点中。

## B+树操作的可视化呈现

- <http://cmudb.io/btree>
- 由旧金山大学的助理教授David Galles开发



## B+树-删除操作

- (1) 从根结点出发找到该关键字所在结点L，删除该关键字；
- (2) 如果结点L的关键字数目不少于 $M/2$ ，则删除完成；

如果结点L仅有 $M/2-1$ 个关键字数目，向兄弟结点借一个关键字（兄弟结点指与L有相同父结点的相邻结点）；

如果借关键字失败，则将结点L与其兄弟结点合并，合并时需要删除父结点中的关键字（指向L或其兄弟结点）。

## B+树-重复键 (Duplicate Keys)

在B+树中有两种允许重复键的方法：

### 方法一：附加记录ID作为键的一部分

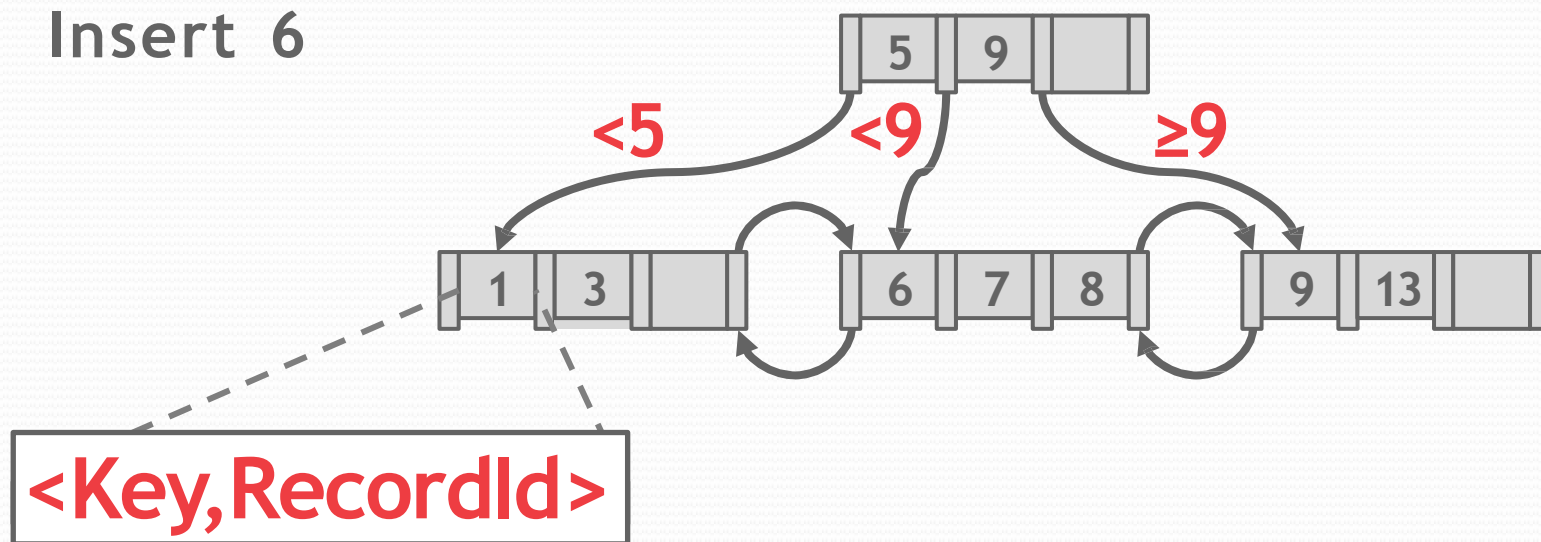
由于每个元组的记录ID是唯一的，因此确保了所有键都是可识别的。

### 方法二：允许叶结点溢出至包含重复键的溢出结点

该方法的维护和修改较为复杂。

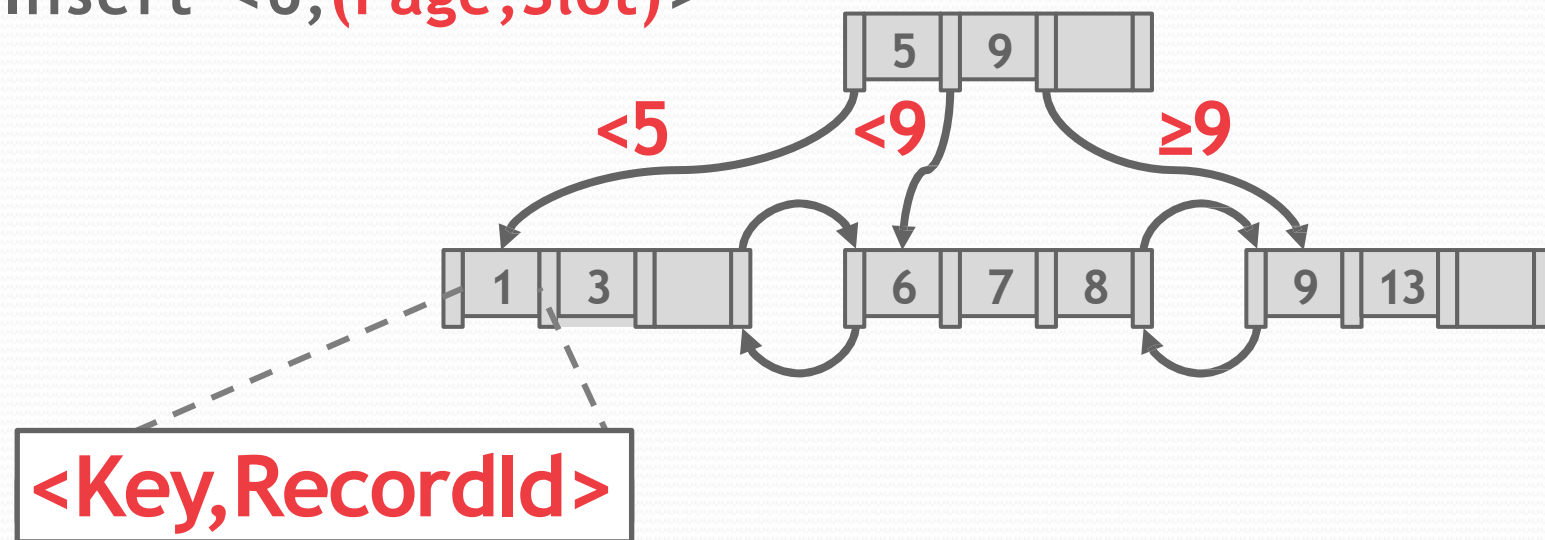
## B+树重复建——附加记录ID

Insert 6



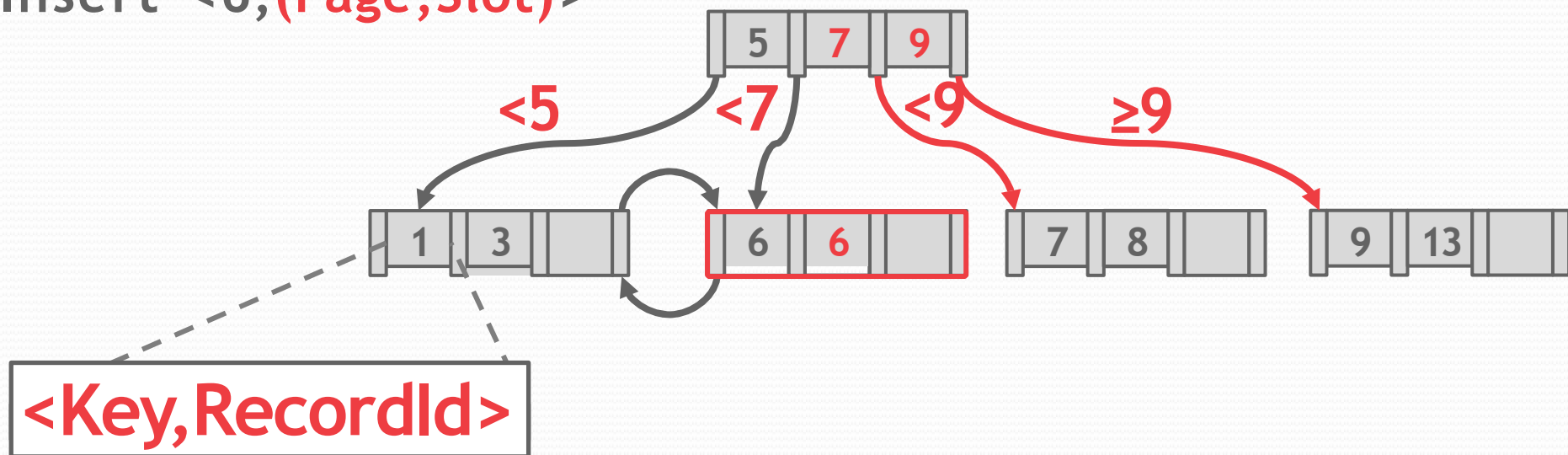
## B+树重复建——附加记录ID

Insert  $\langle 6, (\text{Page}, \text{Slot}) \rangle$



## B+树重复建——附加记录ID

Insert  $\langle 6, (\text{Page}, \text{Slot}) \rangle$

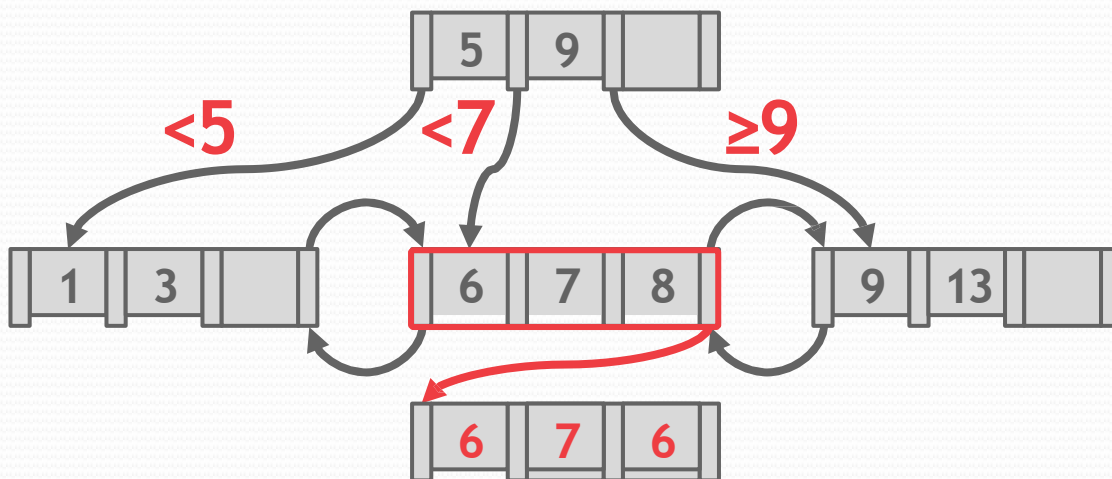


## B+树重复建——叶结点溢出

insert 6

insert 7

insert 6



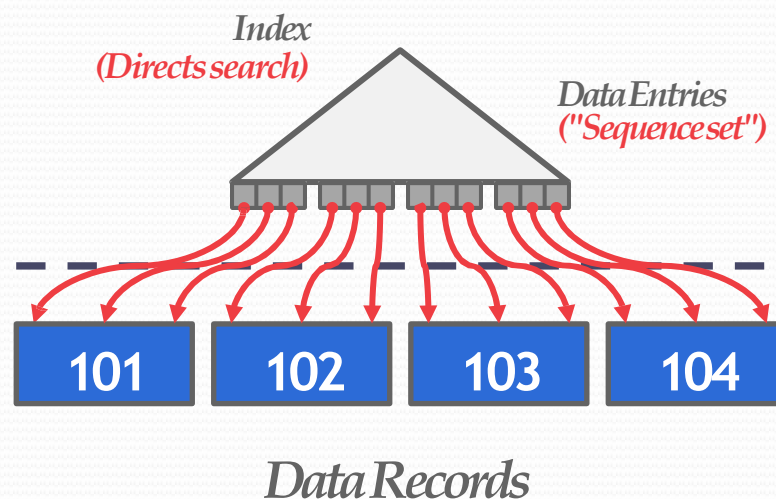
## 聚簇索引 (Clustered Indexes)

- 关系按照主键的排列顺序存储  
可以是基于堆的存储，或者索引组织的存储。
- 一些DBMS使用聚簇索引  
如果一个关系没有主键，DBMS则会自动生成一个隐藏的行ID主键。
- 有些DBMS两种方式都使用。

## 聚簇的B+树 (Clustered B+tree)

遍历最左侧的叶子页，然后  
在所有叶子页中检索元组。

该方法方法优于外部排序

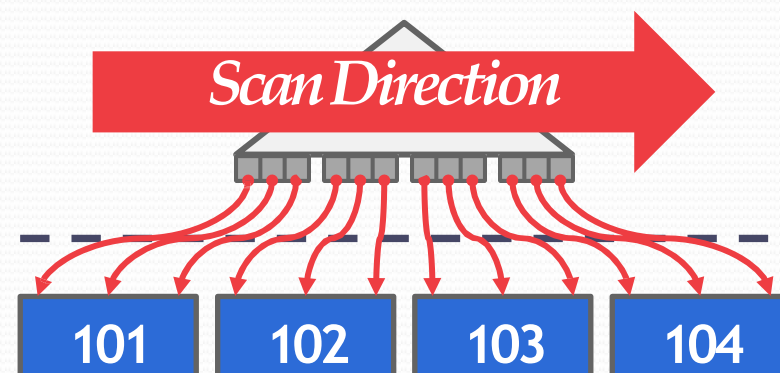




## 堆聚簇 (Heap Clustering)

元组在~~heap~~页面集合中按照聚簇索引指定的顺序排序。

如果使用聚集索引的属性来访问元组，那么DBMS可以直接~~跳转至~~目标页。

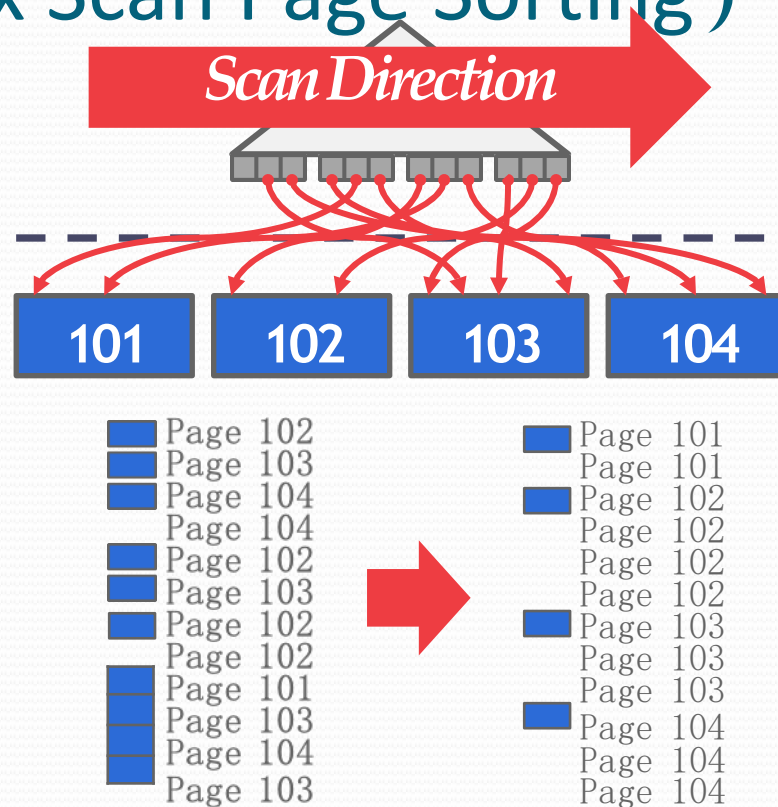


## 索引扫描页面排序 (Index Scan Page Sorting)

在非聚集索引中按照元组出现的顺序检索元组是十分低效的。

### 优化措施:

DBMS可以先找出所需要的所有元组（“ID”），然后根据其页ID对元组进行排序。



## B+树的设计选择

- 结点大小
- 合并阈值
- 变长键
- 结点内搜索

## 结点大小

- 存储设备速度越慢，B+ 树的最佳结点大小就越大。
  - 硬盘 ~1MB
  - 固态硬盘： ~10KB
  - 内存： ~512B
- 最佳大小可能因工作负载而异
  - 叶结点扫描 vs. 根到叶遍历

## 合并阈值

- 某些 DBMS 在结点半满时并不总是合并节点。
- 延迟合并操作可能会减少重组数量。
- 允许下溢的存在然后定期重建整个树，也可能是有益的。

## 变长键

### 方法1：指针

- 将键存储为指向元组属性的指针。

### 方法2：可变长结点

- 索引中每个结点的大小可以改变。
- 需要细致的内存管理。

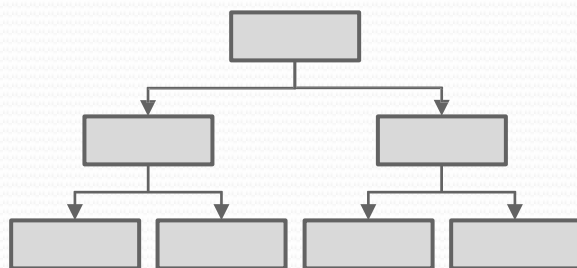
### 方法3：填充（定长）

- 将每个键的大小设置为最大键的大小并填充所有较短的键。

### 方法4：键映射/间接

- 嵌入一个指针数组来映射到节点中的“键+值”列表。

# 键映射/间接



## B+Tree Leaf Node

Level	Slots	Prev	Next
#	#	❌	❌

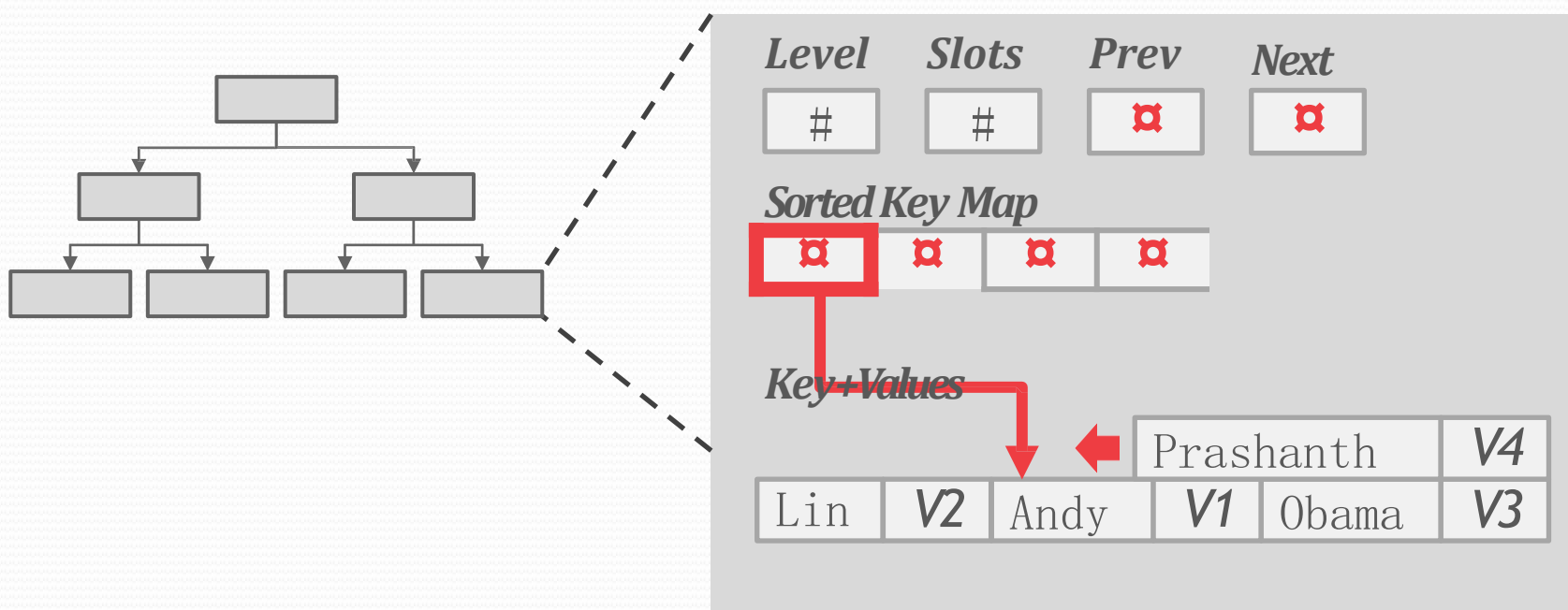
### Sorted Key Map

❌	❌	❌	❌
---	---	---	---

### Key+Values

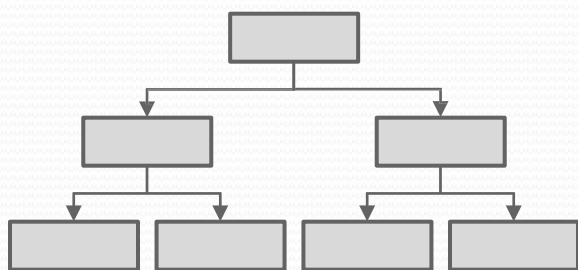
← Prashanth		V4
Lin	V2	Andy
V1	Obama	V3

# 键映射/间接





# 键映射/间接



## B+Tree Leaf Node

Level	Slots	Prev	Next
#	#	❌	❌

### Sorted Key Map

A	L	O	P
---	---	---	---

### Key+Values

Lin	V2	Andy	V1	Obama	V3
				Prashanth	V4

将每个键的前缀放在索引旁边

## 结点内搜索

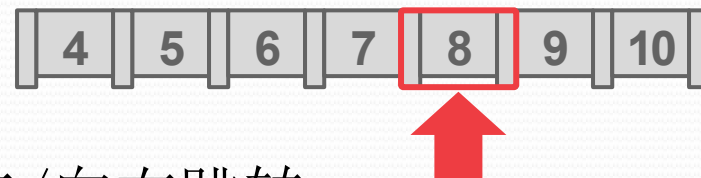
方法1：线性

- 从头到尾扫描节点键。



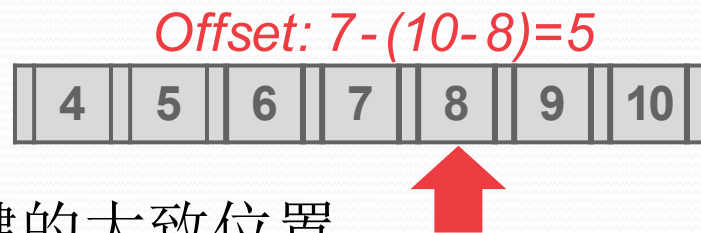
方法2：二分

- 跳转到中间键，根据比较向左/向右跳转。



方法3：插值

- 基于已知键分布，推导所需键的大致位置。



## B+树的优化

- 前缀压缩

同一叶结点中的已排序键可能具有相同的前缀，无需每次都存储整个键，而是提取公共前缀并仅存储每个键的唯一后缀。

- 去重

非唯一索引可能会在结点中存储同一键的多个副本的（相同的）键。叶结点可以只存储键值一次，然后使用该键维护记录 ID 列表。

## B+树的优化（续）

- 后缀截断

内部节点中的键值仅用于“**交通导向**”，不需要存储整个键值。索引存储探测正确**路径所需的最小前缀**。

- 批量插入

为现有表生成新 B+树的最快方法是首先对键进行排序，然后自下而上构建索引。

- 指针混淆

## B+树的优化（续）

- 指针混淆

一般情况下，结点使用页 ID 引用索引中的其他结点，在遍历期间，DBMS 必须从页表中获取内存位置。如果页面固定在缓冲池中，则可以存储原始指针而不是页面ID，从而可以避免从页表中查找地址。

注意：必须跟踪哪些指针被混淆，并在它们指向的页面被解除固定和释放出缓存时将它们切换回页面ID。

## 部分索引(Partial Indexes)

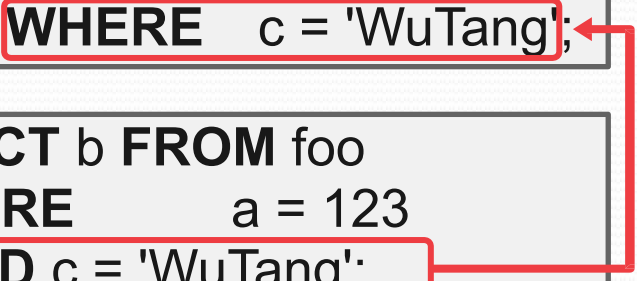
在整个表的子集上创建索引，这可能会减小索引的大小和维护它的开销量。

常见的用例：

按日期范围对索引进行分区。  
每月、每年创建单独的索引。

```
CREATE INDEX idx_foo  
ON foo (a, b)  
WHERE c = 'WuTang';
```

```
SELECT b FROM foo  
WHERE a = 123  
AND c = 'WuTang';
```



## 部分索引失效

```
CREATE INDEX idx_foo  
ON foo (a, b)  
WHERE c = 'WuTang';
```

```
SELECT b FROM foo  
WHERE a = 123
```

## 覆盖索引(Covering Indexes)

如果处理查询所需的所有**字段在索引中都可**用，则 DBMS 不需要检索元组。

这也被称为**仅索引扫描** (index-only scans)，可以减少对 DBMS 缓冲池资源的竞争。

```
CREATE INDEX idx_foo  
ON foo (a, b);
```

```
SELECT b FROM foo  
WHERE a = 123;
```





## 索引包含列(Index Include Columns)

在索引中嵌入  
其他列以支持仅索引（index-only）查询。

这些额外的列仅存储在叶结点中，并不是搜索键的一部分。

```
CREATE INDEX idx_foo  
ON foo (a, b)  
INCLUDE (c);
```

```
SELECT b FROM foo  
WHERE a = 123  
AND c = 'WuTang';
```

不参与路径上层结构

## 函数/表达式索引(Functional/Expression Indexes)

索引不必要以它们在基表中出现的方式存储键。

声明索引时可以使用表达式。

```
SELECT * FROM users
WHERE EXTRACT(dow
FROM login) = 2;
```

```
CREATE INDEX
idx_user_login
ON users (login),
```

```
CREATE INDEX idx_user_login
ON users (EXTRACT(dow FROM login))
```

## 函数/表达式索引(Functional/Expression Indexes) (续)

```
SELECT * FROM users  
WHERE EXTRACT(dow  
FROM login) = 2;
```

```
CREATE INDEX  
idx_user_login  
ON users (login),
```

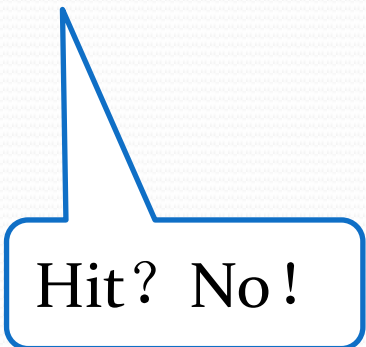
```
CREATE INDEX idx_user_login  
ON users (EXTRACT(dow FROM login))
```

```
CREATE INDEX idx_user_login  
ON foo (login)  
WHERE EXTRACT(dow FROM login) = 2;
```

表达式、  
部分索引

## 观察与思考

- B+树中的内部结点键无法表明索引中是否存在键。每次查找一个key，必须遍历到叶子结点。
- 这意味着，在树中的每一级（至少）有一个缓冲池页“miss”，只是为了找出一个键不存在。



Hit? No!

## 倒排索引

倒排索引：存储单词到目标属性中包含这些单词的记录的映射

▶ 有时称为全文搜索索引。

主流的DBMS包含这些机制，还有一些专用的DBMS。



## 倒排索引支持的查询类型 (B+树无法支持)

- 短语搜索
  - 查找按给定顺序包含单词列表的记录
- 邻近搜索
  - 查找两个单词出现在  $n$  个单词距离之间的记录
- 通配符搜索
  - 查找包含匹配某些模式（例如，正则表达式）的单词的记录。

索引项属性

## 设计决策

### 决策1：存储什么

- 索引至少需要存储每条记录中包含的单词（以标点符号分隔）
- 还可以存储频率、位置和其他元数据

### 决策2：何时更新

- 维护辅助数据结构以“暂存”更新，然后批量更新索引。

针对某一个关系的某个或者些属性上的条件查询（选择运算、连接运算），建立辅助数据结构，以加快查询

### 3.3.3 索引建立与删除

#### 1、建立

1) 格式: **create [unique][cluster] INDEX 索引名**  
**ON 基本表名 (列名[次序] [, 列名[次序]].....) [其它参数]**

2) 功能: 在一个基本表上建立一个索引

#### 3) 说明

- **unique** 表示索引项值对应元组唯一, **cluster**表示聚簇
- 次序指升/降序, 缺省为升序
- **DBA**建立、系统自动实现, 与应用编程无关
- 好处: 提高速度
- 坏处: 占内存、插入删除的移动 (插入删除少)



例：针对学生关系STU的学号属性SNO建立索引

```
CREATE INDEX IND_STU ON STU(SNO);
```

例：针对学生关系STU的学号属性SNO建立唯一性索引

```
CREATE UNIQUE INDEX IND_STU_UNI  
ON STU(SNO);
```

例：针对学生选课关系SC的学号+课号属性组合(SNO,CNO)建立索引

```
CREATE INDEX IND_SC ON SC(SNO,CNO);
```

例：针对学生选课关系SC的学号属性SNO建立聚簇索引

```
CREATE CLUSTER INDEX IND_SC_CLU  
ON SC(SNO);
```

## 2、删除

DROP INDEX 索引名

删去一个索引。

## 3.4 数据查询

### 一般格式

SELECT [ALL|DISTINCT]            目标属性名或表达式列表  
FROM 关系名或视图名列表  
[WHERE        条件表达式]  
[GROUP BY        分组属性名表达式 [HAVING 条件 ] ]  
[ORDER BY 排序属性名表 {    ASC    }  
                                      {    DESC    } ]



## 1、说明

- 1) 表达式：算术表达式、函数表达式、常量
  - 2) 目标属性名表：要查找的属性值的属性名集合（投影）
  - 3) **FROM**：句子中涉及属性名的所在关系或视图名  
(**WHERE**, **GROUP**, **ORDER**)
  - 4) **WHERE**：查找元组应满足的条件。
  - 5) **GROUP**：输出结果按其指定属性分组，每组输出一行。  
如**GROUP BY SNO**
  - 6) **HAVING**：一个分组元组应满足的条件：如**HAVING**  
**COUNT(\*)>80**
  - 7) **ORDER BY**：输出结果按其规定的属性值的升或降序排序。
- ## 2、功能：根据**WHERE**条件表达式，从表中找出满足条件元组的指定属性值，并按指定（若有**GROUP**）的属性分组统计，最后排序（若有**ORDER BY**）输出。

### 3.4.1 单表查询

student (sno, sname, ssex, birthyear, sdept)

course (cno, cname, cpno, ccredit, cdept)

sc(sno, cno, grade)

1、简单查询:

例① 查指定列: `select sno, sname  
from student;`

例② 全部列: `select *  
from student;`

例③ 去掉重复元祖: `select distinct sno  
from sc;`

例④ 计算结果: `select sname, 2010-birthyear  
from student ;`

使用别名: 2010-birthyear `as sage`

`select 2010-birthyear as sage from student;`

注：别名的使用

“属性名 **as** 别名”，例如：sname as xingming

“表达式 **as** 别名”，例如 2010-birthyear as sage

“表名 别名”，例如 dep depstu

元组变量

Select t1.sno as a, t1.sname, t2.sno as b

From student t1, sc t2

Where a=b

## 2、条件查询 (where)

1) 比较大小: =, >, <, >=, <=, != (<>), 前面加not

例①查计算机系学生信息:

```
select *  
from student  
where sdept = "计算机"
```

例② 年龄小于20岁的学生姓名和年龄:

```
select sname, sage  
from student  
where sage < 20; (或where not sage >= 20);
```

## 2) 范围查询

between ... and ...

not between ... and...

闭区间

例① 查询20~25岁间(包括20和25岁)的学生姓名, 年龄

```
select sname, sage  
from student  
where sage between 20 and 25;
```



例② 查年龄不在20~25间的学生姓名，年龄

```
select sname, sage
```

```
from student
```

```
where sage not between 20 and 25;
```

3) 集合查询（IN查询）（查某属性值属于指定集合的元组）

例：查计算机、管理、控制系的学生姓名与单位

```
select sname, sdept
```

```
from student
```

```
where sdept in ("计算机", "管理", "控制");
```



#### 4) 匹配查询（采用like来进行匹配查询）

例①查询学号为“96001”的学生的情况

```
select *  
from student  
where sno like '96001';
```

此处没有使用通配符，  
相当于“等于”

例② 查所有姓刘的学生的情况

```
select *  
from student  
where sname like '刘%';
```

（使用了通配符，代表任意长度字符串匹配）

例③ 查姓刘且全名包含3个汉字的学生情况

```
select *
```

```
from student
```

```
where sname like '刘_ _ _';
```

（一个下划线可和任何半个字符匹配，一个汉字占2个字符）

例④ 查名称中第二个字为“若”的学生情况

```
select *
```

```
from student
```

```
where sname like '_ _ 若%';
```

例⑤ 查所有不姓刘的学生情况

```
select *
```

```
from student
```

```
where sname not like '刘%'
```

转义字符,使通配符“%、\_”不再表示通配含义,而是表示普通的字符,例如: escape '\', escape '/’。。。。

例⑥ select cno, ccredit

```
from course
```

```
where cname like 'DB\_Design' escape '\'
```

例⑦

select cno, ccredit

from course

where cname like 'DB\\_Design\\_ ' escape '\'

转? 还是不转?

## 5) 空值查询 (is null, is not null)

例① 查无成绩的学生的学号及课程号 (选课未考试)

```
select sno, cno
```

```
from sc
```

```
where grade is null;
```

附注：空值——null，在数据库中是“未知类型变量”



任何对null的逻辑比较运算，数据库不做处理。

跳过该记录

例② 查全部有成绩的学生学号及课程号

```
select sno, cno
```

```
from sc
```

```
where grade is not null;
```

6) 复合（条件）查询（AND, OR, <, =, >, =, >=, <=, ≠）

```
select sno, sname
```

```
from student
```

```
where ssex = '女' and sage >25;
```

例① 查计算机系年龄小于20岁的学生姓名及年龄

```
select sname, sage
```

```
from student
```

```
where sdept = '计算机' and sage < 20;
```

例② 查计算机或（和）管理系的学生姓名及年龄

```
select sname, sage
```

```
from student
```

```
where sdept = '计算机' or sdept = '管理';
```

### 3、排序查询（order by ...asc（升序），...desc（降序））

例① 查选修3号课学生的学号与成绩，结果按**grade**降序排列

```
select sno, grade  
from sc  
where cno =3  
order by grade desc;
```

例② 查全体学生基本情况，结果按所在系升序排列，**同一系学生按年龄降序排列**

```
select *  
from student  
order by sdept, sage desc;
```



## 4、集函数查询

集函数: count ([distinct | all] \*) 统计元组数

count ([distinct | all] <列名> ) 统计一列中值的个数

sum ([distinct | all] <列名>) 数值字段求和

avg ([distinct | all] <列名>) 列平均值

max ([distinct | all] <列名>) 列取最大值

min ([distinct | all] <列名>) 列取最小值

例① 查学生总人数

```
select count (*)
```

```
from student;
```

例② 查选修了课程的学生总人数

```
select count (distinct sno)
```

```
from sc; (一学生选多门课, 只统计一个)
```

注: 集函数的参数为列名的, 遇到取空值的元组? 跳过该元组



## 5、分组查询（查询结果按一或多列值分组）

例1 查各个课程号与相应选课人数

```
select cno, count (sno)  //count(1)
from sc
group by cno;
```

例2 查每个人的成绩总和

```
select sno, sum (grade)
from sc
group by sno;
```

例3 查询学号以“96”开头的每个学生的平均成绩，并将结果按照平均成绩的降序排列。

```
select sno, avg(grade)
from sc
where sno like '96%'
group by sno
order by avg(grade) desc;
```

例4 查询学号以“96”开头的每个学生的选课门数，并将结果按照平均成绩的降序排列。

```
select sno, count(cno)
from sc
where sno like '96%'
group by sno
order by avg(grade) desc;
```

例5 查询每个学生的及格课程的平均成绩。

```
select sno, avg(grade)
from sc
where grade >= 60
group by sno;
```

例6 查询学号以'96'开头且至少选修了5门课的每个学生的平均成绩。

```
select sno, avg(grade)
from sc
where sno like '96%'
group by sno
having count(cno) >= 5;
```

【问题】查询96级的至少5门课及格的每个学生的平均成绩?

查询96级的至少5门课及格的每个学生的平均成绩

```
select sno, avg(grade)
from sc
where sno like '96%'
and sno in (
select sno from sc
  Where grade >= 60
 group by sno
 having count(cno) >= 5 )
group by sno;
```

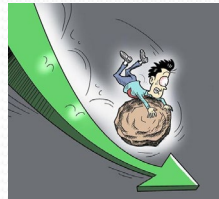
sno	avg(grade)
001	70
002	50

例7 查询没有不及格课程的每个学生的平均成绩

```
1)select sno, avg(grade)
   from sc
  group by sno
 having min(grade)>=60;
```

```
2)select sno, avg(grade)
   from sc
  where sno not in (
      select sno from sc where grade<60)
 group by sno;
```

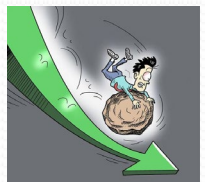
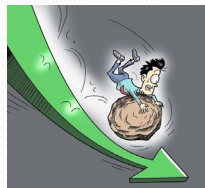
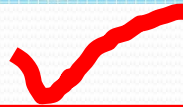
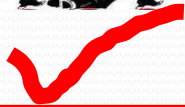
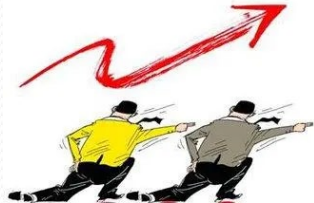
# Where、having两种方法的区别？





# having

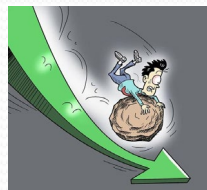
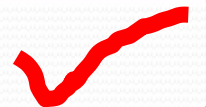
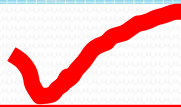
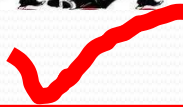
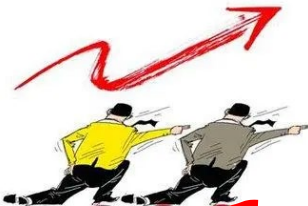
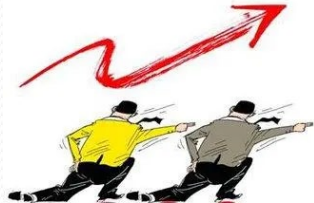
1)





# Where

2)



例8 查学号以' 96'开头、选修了2号课程且选修了5门以上课程的学生学号

```
select sno  
from sc  
where sno in (select sno from sc where cno=2)  
and sno like '96%'  
group by sno  
having count (*)>5;
```

例9 查询1号院系的每个学生的平均成绩

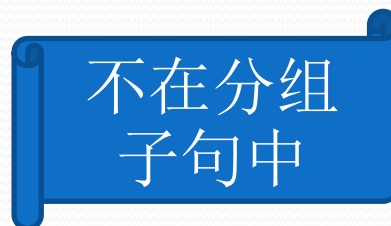
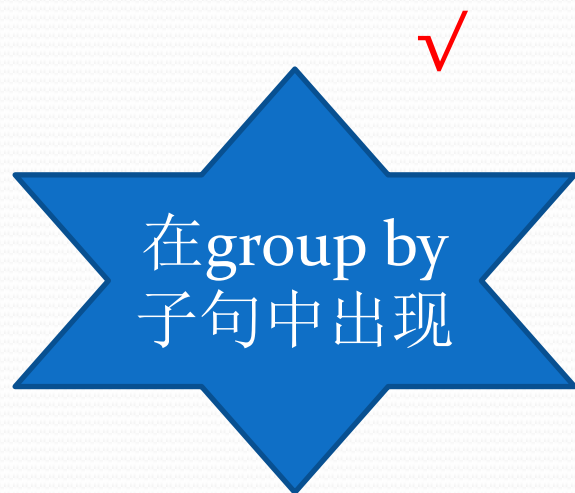
```
select sc.sno, avg(grade)
from sc, student
where sc.sno=student.sno
and student.sdept=1
group by sc.sno;
```

【注】学习、掌握分组查询的灵活、严谨应用。

属性空值的情况?

跳过

select子句里的属性?



✓  
作为集函  
数参数

✗  
未作为集  
函数参数

例10 查询1号院系的每个学生的学号、姓名和平均成绩

```
select sc.sno, student.sname, avg(grade)
from sc, student
where sc.sno=student.sno
and student.sdept=1
group by sc.sno, student.sname;
```

### 3.4.2 连接查询

- 什么是连接查询：一个查询涉及两个以上表
- 为什么？一个表中信息不足
- 条件：连接相关联的属性

#### 1、等值连接

例① 查成绩不及格的学生单位及姓名

```
select sdept, sname  
from student, sc
```

```
where student.sno = sc.sno and grade < 60;
```

问题：若要查询成绩不及格的学生学号、姓名及单位呢？

```
select student.sno, sname, sdept  
from student, sc
```

```
where student.sno = sc.sno and grade < 60;
```

#### 2、自然连接（相同属性，明确写出select项的来源）

### 3、自身连接（自己与自己连接，看成两个表）

例：查每一门的间接先修课（即先修课的先修课）

课程表：course (cno, cname, cpno, ccredit, cdept)

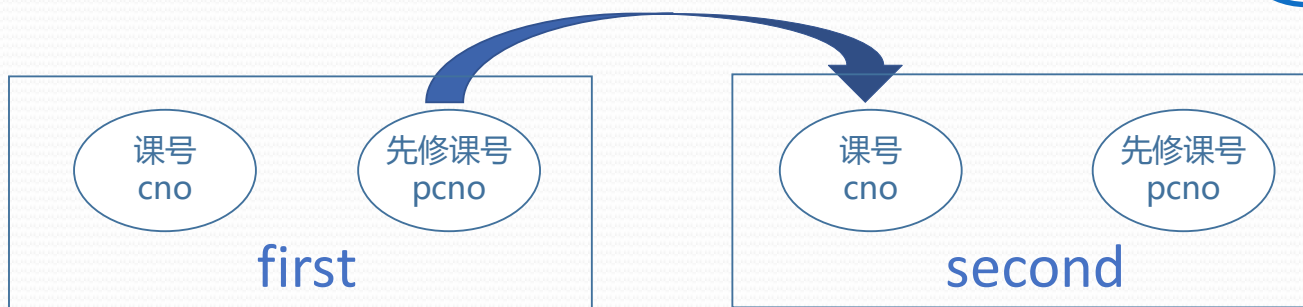
select first.cno, second.pcno

from course first, course second

where first.pcno = second.cno

将course分别看成first和second两个表

=



在DBMS的查询处理过程中，取了两次别名的表对应两份不同的中间数据，占据不同的缓存资源

#### 4、多表连接

查成绩不及格的学生号、姓名及课程名

```
select sc.sno, sname, cname
```

```
from student, course, sc
```

```
where student. sno = sc. sno.
```

```
and course. cno = sc. cno
```

```
and grade <60;
```

注：别名的使用

“表名 别名”，例如dep depstu

查询选修计算机系开设的1号课程的学生们的学号、姓名、学生所在院系名称。

```
select  stu.sno as xuehao,  
        sname as xingming,  
        deps.depname as xuesheng_yuanxi  
from    stu, course, dep deps, dep depc, sc  
where   stu.sno=sc.sno  
and     sc.cno=course.cno  
and     course.depno=depc.depno  
and     stu.sdeptno=deps.depno  
and     depc.depname= “计算机系”  
And     sc.cno=1
```





## 5、外连接

课本第102页例3.53，查询所有学生的基本信息以及选课信息，如果没有选课，则只列出基本信息。

Sql server写法：

```
select student.sno,sname,ssex,sage,sdept,cno,grade  
from student left outer join sc  
on student.sno = sc.sno
```

某些参考书上的写法：

```
select student.sno,sname,ssex,sage,sdept,cno,grade  
from student, sc  
where student.sno = sc.sno(*)
```

实例：查询某种商品的顾客反馈意见【顾客意见表（顾客、商品号、对该种商品的意见）、商品表（商品号、商品名称）】

```
select 商品表.商品号,商品名称,顾客,对该种商品的意见
from 商品表 left outer join 顾客意见表
on 商品表.商品号 = 顾客意见表.商品号
```

商品号	商品名称
1	妈咪宝贝
2	好奇
3	帮宝适

商品号	顾客	意见
1	甲	较厚
2	甲	贵
2	乙	较贵

商品号	商品名称	顾客	意见
1	妈咪宝贝	甲	较厚
2	好奇	甲	贵
2	好奇	乙	较贵
3	帮宝适		

No news is  
good news!

### 3.4.3 嵌套查询

什么是嵌套？一个select嵌套在另一个select条件中查询。

```
select sname  
from student  
where sno in  
    ( select sno  
      from sc  
      where cno = '02'  
    )
```

为什么嵌套？一般是一个关系中找不到条件对应的属性。

属性：公共属性。

执行：先里后外（不相关嵌套查询）

特征：逐步分解，层次分明，易理解，易书写，结构化。

### 1、简单嵌套查询。

例：成绩不及格的学生姓名

```
select sno, sname
```

```
from student
```

```
where sno in ( select sno
```

```
from sc where grade <60);
```

## 2、多层嵌套查询

例：查选修“DBS”课程学生的学号及姓名

```
select sno, sname from student where sno in  
(select sno  
  from sc  
  where cno in  
    (select cno  
     from course  
     where cname='DBS')  
);
```

### 3、同表嵌查询

例：查与“马超”同在一个系的学生的单位及学号，姓名，

```
select sdept, sno, sname
```

```
from student
```

```
where sdept in
```

```
(select sdept
```

```
from student
```

```
where sname=' 马超')
```

```
and sname!='马超'
```

#### 4、相关嵌套查询

例：查选修了 ‘01’号课程的学生姓名。

```
select sname  
from student where '01' in (  
select cno  
from sc  
where sno=student.sno);
```

工作过程；

- (1) 外层student 找到第一个元组，取出sno；
- (2) 在里层SC中找SNO选修课的集合（多门课程，01，02.....）；
- (3) ‘01’在该集合中，则取出sname；
- (4) 再取外层第二个元组。

。

。

。

注：与之相对应的——不相关嵌套查询



## 5、比较嵌套查询

（当查询结果列值唯一时，可用比较，而不用in）

```
select sno, sname, sdept  
from student  
where sdept =  
    (select sdept  
     from student  
     where sname = '马超' )  
and sname <> '马超' ;
```





>any, 大于子查询结果中的某个值

<any, 小于子查询结果中的某个值。

>=any, 大于等于子查询结果中的某个值。。。。。

例：查询其它系比计算机系某一学生年龄小的学生名单及其对应年龄

```
select sname, sage
```

```
from student
```

```
where sage <any(
```

```
    select sage
```

```
    from student
```

```
    where sdept = '计算机' )
```

```
and sdept < > '计算机' ;
```

## 7、all 嵌套查询

>all大于子查询结果中的**所有**值

<all小于子查询结果中的所有值。

。 。 。

例：查询其它系中比**计算机系所有学生**年龄都小的学生名单及其对应年龄，并且将结果按照年龄的降序排列。

```
select sname, sage
from student
where sage<all (
    select sage
    from student
    where sdept='计算机' )
and sdept< >'计算机'
order by sage desc ;
```

## 8、存在量词查询 (exists) (一种特殊的相关查询)

例1：查选修‘01’课程的学生姓名

```
select sname
from student
where exists
( select *
  from sc
  where sno=student.sno
    and cno='01');
```

特征：

- 1) “\*” 是因为带**Exists** 的子查询不返回实际数据，只返回真假值，给出列名无实际意义。
- 2) 子查询条件依赖于外层父查询的某个列值。  
(本例即**student** 中的**sno**值)
- 3) 子查询为真，(则内层子查询非空)，取该元组放入结果

例2: 查询未选修 ‘01’课程的学生姓名。

```
select sname  
from student  
where not exists  
( select *  
  from sc  
  where sno=student.sno and cno='01');
```

Having?

等价表示:

```
select sname  
from student  
where sno <> all (  
  select sno  
  from sc  
  where cno='01');
```

sql语言中没有全称量词，但用存在量词exist等价表示全称量词。

教材第110页例3.62：查询选修了全部课程的学生姓名  
——找出这样的学生“不存在一门课，该学生没有选”

解法一：不存在加上not in

```
select sname from student A
where not exists
(select * from course where cno
not in
(select cno from sc B
where A.sno = B.sno
)
)
```

关系代数中的  
除运算

没有一门课  
不被X的像集  
包含

Y属性

X属性

sql语言中没有全称量词，但用存在量词exist等价表示全称量词。——找出这样的学生“不存在一门课，该学生没有选”

例3.62：查询选修了全部课程的学生姓名（两次not exists）

```
select sname from student
where not exists
(select * from course
where not exists
(select * from sc
where sc.cno = course.cno
and sc.sno = student.sno
)
)
```

例3.62解法三：计数加上外连接

```
select distinct sno from sc A
where
( select count(*) from course
  left outer join sc B on course.cno = B.cno
  and A.sno = B.sno
  where B.sno is null) = 0 ;
```

例3.62解法四：两次计数

```
select distinct sno from student
where (
  select count(*) from course
  where (select count(*) from sc where sc.cno =
course.cno and sc.sno = student.sno
)=0
)=0
```

其他思路？  
数总数？

教材第111页例3.63：查询至少选修了学生200215122选修的全部课程的学生号码

```
select distinct sno
from sc SCX
where not exists
( select * from sc SCY
  where sno='200215122'
  and not exists
    ( select * from sc SCZ
      where SCZ.cno = SCY.cno
      and SCZ.sno = SCX.sno))
```



## 例3.63另解

```
select distinct sno from sc A
where not exists
( select * from sc B
  where B.sno = '200215122'
    and cno not in
      ( select cno from sc C
        where C.sno = A.sno
      )
    )
)
```

之前例8： 查学号以' 96'开头、选修了2号课程且选修了5门以上课程的学生学号

```
select sno  
from sc  
where sno in (select sno from sc where cno=2)  
and sno like '96%'  
group by sno  
having count (*)>5;
```

例8 查学号以'96'开头、选修了2号课程且选修了5门以上课程的学生学号

```
select sno,avg(score)
from sc
```

```
group by sno having 2 in (cno) //语法错误！！
```



---

```
select sno,avg(score)
```

```
from sc sc1
```

```
group by sno
```

```
having 2 in (
```

```
    select cno
```

```
    from sc sc2
```

```
    where sc2.sno=sc1.sno)
```



---

```
select sno,avg(grade) from sc sc1
```

```
group by sno
```

```
having sno in (
```

```
select sno from sc sc2 where sc2.cno=2)
```



例：设有关系

**OSCAR(YEAR, DNAME, FNAME)**记录历史上各个年份（**YEAR**）获得了奥斯卡奖的电影名称(**FNAME**)及其导演姓名(**DNAME**),

另有关系

**FILM(DNAME, FNAME, INCOME)**记录每一部票房收入过千万的电影的导演姓名(**DNAME**)、电影名称(**FNAME**)及其票房收入金额(**INCOME**)。

请分别用一条**SQL**语句实现下列两小题的查询需求。

- 1) 查询每一位获得了奥斯卡奖的导演及其获奖影片数量，并且结果按照获奖数量的降序排列。
- 2) 查询**FILM**表中得过奥斯卡奖的导演拍摄的但未获奖的电影的名称、导演及其票房收入。

这些电影可能别的导演拍摄并获奖了

1) 查询每一位获得了奥斯卡奖的导演及其获奖影片数量，并且结果按照获奖数量的降序排列。

```
SELECT DNAME, COUNT(FNAME)
FROM OSCAR
GROUP BY DNAME
ORDER BY COUNT(FNAME) DESC;
```

2) 查询FILM表中得过奥斯卡奖的导演拍摄的但未获奖的电影的名称、导演及其票房收入。

SELECT \* FROM FILM

WHERE DNAME IN (

SELECT DNAME FROM OSCAR)

AND NOT EXIST (

SELECT \* FROM OSCAR

WHERE OSCAR.DNAME=FILM.DNAME

AND OSCAR.FNAME=FILM.FNAME)

NOT IN  
行不行?

语法可以,  
但两次NOT IN合起来的意思可能不同。  
获奖导演拍的、且从未有导演因此获奖的。

### 3.4.4 集合查询:union、intersect、except

例：查询计算机系的学生以及年龄不大于19岁的学生

```
SELECT *  
FROM Student  
WHERE Sdept='CS'  
UNION      //INTERSECT, EXCEPT  
SELECT *  
FROM Student  
WHERE Sage<=19
```

注：若使用**UNION ALL**则会保留重复元组

### 3.4.5 基于派生表的查询

派生表：子查询生成的临时表，可以出现在主查询的FROM子句中，作为主查询的查询对象。

```
SELECT col1,col2,tempcol4
FROM TABLE1,
    ( SELECT col3,avg(col4) FROM TABLE2
      GROUP BY col3 ) AS
    TABLE2_TMP(tempcol3,tempcol4)
WHERE col1=TABLE2_TMP.tempcol3
```



例：查询成绩表中每个学生选的所有课程中自己学的比较好（大于自己的平时成绩）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_sc.avg_sno AND
sc.grade>=avg_sc.avg_grade
```

相关嵌套？  
物化、临时表

派生表的属性列：子查询中**没有聚集函数**的情况下可以不显示命名。

```
SELECT sname
```

```
FROM student, (SELECT sno FROM sc  
WHERE cno='1') AS sc1
```

```
WHERE student.sno=sc1.sno
```

分组查询再举例: `select cno, max(grade)`  
`from sc`

`group by cno;` 最高分学生的学号? ? ?

`group by`的局限性: `select`子句中不允许出现分组、集函数之外的查询列。

ORACLE例子:

`select substr(maxsno,1,3) as grade,`  
`substr(maxsno,4,5) as sno, cno`

`from`

`(select max( to_string(grade,'000')|sno) as`  
`maxsno, cno`

`from sc`

`group by cno);` 前两名? ? ?

ORACLE例子:

```
select *  
from  
(select sno, cno, grade, rank () over(partition cno  
    order by grade desc ) as rk  
from sc  
)  
where rk = 1 or rk = 2
```

Sql server例子（针对某一门课程）：

```
select top 3 cno,sno,grade from sc  
where cno='001'  
order by grade desc;
```

## 3.5 数据更新

### 3.5.1 插入数据

1、插入单个元组。

(1) 格式

`insert into <表名>[ (<属性名1>, ..... ) ]`

`values (<常量1>[, <常量2>].....)`

例：插入一个新生到student 中去

`insert into student`

`values('961', '杨柳', '男', '25', '计算机')`

(2) 使用说明

- 属性与值在类型、个数上须一致；
- 完全一致可省属性名；
- 值缺失则送null，但属性指定了缺省值的例外。

## 2、插入子查询结果（可多个元组）

格式：insert into <表名>[（< 属性名1>[， < 属性名2>.....）] 子查询；

例：insert into student\_female

select \*

from student

where ssex = ‘女’；

迪丽热巴	。 。 。	女
古力娜扎	。 。 。	女
冯莫提	。 。 。	女

姓名	学号	性别
夏秋冬	。 。 。	女

### 3.5.2 修改数据

#### 1、一般格式

update <表名>

set <属性名>=<exp>[, <属性名>=<exp>].....

[where <条件exp>]

#### 2、修改一个元组值

例: update student

set sname =‘吴向上’ , note=‘new name’

where sname=‘吴天天’ ;

### 3、修改多个元组值

update sc

set grade=grade-10; 所有成绩减10分

### 4、带子查询修改

例：将“计算机”系全体学生 ‘01’号课程的成绩取平方根再乘以10。

update **sc**

set grade=sqrt(grade) \*10

where ‘计算机’ = (select sdept

from student

where student. sno=**sc.sno**)

and cno='01';



## 5、一致性修改

因为修改一次只对一个表，这就带来问题，如某一个人学号变了，必须同时修改student和sc中的sno，否则就不一致。

必须通过两个update实现（要么都不执行，要么都执行）

```
update student
```

```
update sc
```

```
Set sno='9610'
```

```
Set sno='9610'
```

```
where sno='9680';
```

```
where sno='9680'
```

附注：通过begin transaction 和commit实现一致性。

### 3.5.3 删除数据

1、一般格式;

```
delete from <表名>  
[where <条件exp>];
```

2、删去一个元组;

删去 ‘9610’选修 ‘01’课程元组;

```
delete  
from sc  
where sno='9610' and cno='01';
```

3、删去多个元组

例①

```
delete  
from sc  
where sno = '9620'; （一学生选多门课）
```

例②

delete

from sc;    删去所有选课元组



#### 4、带子查询删除

例：删去计算机系所有学生选课记录

delete

from **sc**

where ‘计算机’ =

(select sdept

from student

where student.sno=**sc**. sno)

### 3.6 空值的处理

- 空值（NULL）表示不知道、不存在、没有意义。
- 空值含有不确定性（甚至有时被认为是没有数据类型的值）。
- 空值的算数、比较、逻辑运算
  - 空值与另一个非空值的算数运算结果为空值；
  - 空值的比较运算的结果为UNKNOWN，这将传统的二值逻辑（TRUE, FALSE）扩展为三值逻辑（T, F, U）；
  - 空值的逻辑运算？

## 包含空值的逻辑运算符真值表

X	Y	X AND Y	X OR Y	NOT X
T	T	T	T	F
T	U	U	T	F
T	F	F	T	F
U	T	U	T	U
U	U	U	U	U
U	F	F	U	U
F	T	F	T	T
F	U	F	U	T
F	F	F	F	T

## 3.7 视图定义与操作

### 3.7.1 建立视图

1、一般格式: `create view <视图名>[ (<属性名1>, [, <属性名2>) .....]`

`as <子查询>`

`[with check option ];`

若选用, 则此后对视图进行`update`、`insert` 操作时必须满足子查询中`where`子句给出的条件 (某些系统自动加上相关条件)。

说明: 列名在下列情况下**不可省**:

- ①列非单纯的属性名, 而是集函数或表达式;
- ②同名属性名作为视图中的属性 (多表中定义视图);
- ③视图用新的列名。

## 2、单表视图

例：① 建立计算机系学生的视图

```
create view computer-student  
as  
select sno, sname, ssex, sage  
from student  
where sdept='计算机' ;
```

例② create view computer-student

as

select sno , sname, ssex, sage

from student

where sdept='计算机'

with check option

以后对computer-student进行插入、修改和删除操作时，  
(DBMS自动加上“sdept = ‘计算机’ ”的条件)。



### 3、多表视图

例：create view computer-sc1 (**sno**, sname, grade)  
as  
select student.sno, sname, grade  
from student, sc  
where sdept = '计算机' and **student. sno = sc. sno**  
and sc. cno='01';

建立一个计算机系选修01号课程的学生视图；

由于sno为student及sc中同名列，故视图名后必须明确说明列名。

## 4、视图上建立视图

上一个视图中成绩**90**分及以上的记录。

例：create view computer-sc2

as

select sno ,sname, grade

from computer-sc1

where grade >=90;

这个视图建立在上一个视图之上。

## 5、表达式视图

例: create view student-year (sno, sname, **sage**)

as

select sno, sname, **year(sysdate)-sbirth**

From student;

建立一个虚拟列

## 6、集函数视图

create view sc-avg(sno, **gavg**)

as

select sno, **avg(grade)**

from sc

group by sno;

## 7、视图删除

**drop view** 视图名

删去该视图（从数据字典中）

删去该视图的视图（可选择）

基本表删去了，则（可选择）相关视图也删去。

### 3.7.2 视图查询

一般对基表去查询操作均可作用于视图

DBMS处理步骤如下：

- 1、有效性检查；（表、视图存在否）
  - 2、取出视图定义（从数据字典DD中，若存在）
  - 3、转换（对基本表的查询）  
（对视图的查询与定义视图的子查询结合起来）
  - 4、执行转换后的查询。
- 视图消解（View Resolution）

### 3.7.3 视图更新

1、操作分类: insert, delete, update

2、约束

- ① sql通常只允许对单个表导出的视图可更新;
- ② 若视图属性由表达式或常数组成, 则不可对之 insert、update, 但可 delete;

- ③ 若视图属性为**集函数**，则不可更新；
- ④ 若视图中有**group by** 则不可更新；
- ⑤ 定义中若使用**distinct**，则不可更新；
- ⑥ **嵌套子查询**定义视图时，若**内外层表相同**，则不可更新；
- ⑦ **不允许更新的视图**导出的视图不可更新。

存在  
不确定性

什么样的视图理论上可更新是一个待研究课题

不可更新：理论上证明不可更新；

不允许更新：本身可能是可更新的，但实际系统不支持。

### 3、insert

insert

into computer-student

values('9690', '程由之' , 20)

DBMS将其转换为对其本表的更新

insert

into student (sno ,sname , sage ,sdept)

Values ('9690', '程由之' ,20, 'computer')



#### 4、update更新

update computer-student

set sname='牛一飞'

where sno = '9670'

检查后转换对基本表更新

update student

set sname='牛一飞'

where sno='9670' and sdept='计算机'

## 5、delete删除

delete

from computer-student

where sno = '9609'

转换: delete from student

where sno = '9609' and sdept = '计算机'

### 3.7.4 视图的作用

外模式

1. 简化用户的操作
2. 使用户能以多种角度看待同一数据
3. 对重构数据库提供了一定程度的逻辑独立性
4. 对数据提供安全保护
5. 更清晰地表达查询（提供逻辑数据集）。

**[例1]** 查询赊销收入凭证。(用友财务)

gl\_accvouch:

会计期间 ipperiod	凭证类型 csign	凭证号 ino_id	行号 inid	科目 ccode	摘要 cdigest	借方 md	贷方 mc
3	记	8	1	113		23400	
3	记	8	2	501			20000
3	记	8	3	221			3400

赊销收入凭证特征:

借方: 应收账款 (113)

贷方: 销售收入 (501)

应缴税金 (221)

售给天津A公司软件100套, 每套不含税价为200元, 货款未收, 适用税率: 17%.

摘要: 售A软件, 款未收。

借: 应收账款 23400

贷: 主营业务收入 20000

应交税金/应交增值税/销项税额 3400

```
select iperiod,csign,ino_id,inid,ccode, cdigest,md,mc
from gl_accvouch A
where exists (
    select * from gl_accvouch B
    where A.iperiod = B.iperiod and
        A.csign = B.csign and
        A.ino_id = B.ino_id and
        B.ccode like '113%' and
        B.md > 0 )
and exists (
    select * from gl_accvouch C
    where A.iperiod = C.iperiod and
        A.csign = C.csign and
        A.ino_id = C.ino_id and
        C.ccode like '501%' and
        C.mc > 0)
//and A.ccode like '221%'
order by iperiod,csign,ino_id,inid
```

会计期间 iperiod	凭证类型 csign	凭证号 ino_id	行号 inid	科目 ccode	摘要 cdigest	借方 md	贷方 mc
3	记	8	1	113	售A软件, 款未收	23400	
3	记	8	2	501			20000
<del>3</del>	<del>记</del>	<del>8</del>	<del>3</del>	<del>221</del>			<del>3400</del>

售给天津A公司软件100套, 每套不含税价为200元, 货款未收, 适用税率: 17%.  
摘要: 售A软件, 款未收。

借: 应收账款                23400

贷: 主营业务收入        20000

    应交税金/应交增值税/销项税额    3400

按照会计制度的相关规定，不论是否发放工资，每个月都要计提工资。

(1)计提工资时：

借：管理费用--工资

贷：应付职工薪酬--工资

(2)发放工资时：

借：应付职工薪酬--工资

贷：应交税费--个人所得税

贷：其他应付款--社保(个人)

贷：其他应付款--公积金(个人)

贷：库存现金（或银行存款）

**[例2]** 查询2015-3-20至2015-5-1分别在两个特定区域有通信记录的手机号码。

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

**2015-3-20 9:00pm – 次日 2:00am:**

**中国地质大学区域、基站(28991,7202)**

**2015-5-1 9:00pm – 次日 2:00am:**

**湖北大学区域、基站(28961,10522)**



```
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'
)
```

主叫号码是  
同一个人

**[例3]** 某男子20年前涉嫌犯罪潜逃，彼时，其一双儿女尚在蹒跚学步。其子女现已长大成家，一个在深圳工作，一位在西安工作。其家人声称从未与该嫌犯有任何联系，但刑侦人员坚信嫌疑人与其子女有电话联系。

在“清网行动”中，成功通过一条SQL语句查清嫌犯使用的手机号和使用地。侦察人员前往该地排查时，恰逢该号码联系顺丰快递网点，侦察人员在快递网点守候，抓获嫌犯。

设子女手机号分别为1和2，深圳和西安区号分别为0755和029，设嫌犯原籍区号为0715。请写出从通话记录R中查询刑侦人员需要重点特别排查的手机号的SQL语句。

主叫号码 Oid	被叫号码 Tid	...	通话地区号 area	...	

提示：与子女都有通话记录，且通话地不在二位工作地，也不在原籍的号码将大大缩小侦察范围。

```
SELECT R1.oid, R1.area  
FROM R R1, R R2  
WHERE R1.oid=R2.oid  
AND R1.Tid=1  
AND R2.Tid=2  
AND R1.area NOT IN (0715, 0755, 029)
```

```
SELECT R1.oid, R1.area  
FROM R R1  
WHERE R1.Tid=1  
AND R1.area NOT IN (0715, 0755, 029)  
AND EXISTS (  
  SELECT * FROM R R2  
  WHERE R1.oid = R2.oid  
  AND R2.Tid=2)
```

## 3.8 数据库控制

### 3.8.1 安全控制

1、功能：控制用户对数据的存取权力

2、方法：

① DBA实施权利授予说明

② DBMS保存（DD）、检查、作出决定与处理。

3、授权语句

**grant** 权力[, 权力]...[on 对象类型 对象名] **to** 用户[, 用户]...

[with grant option];

4、权力撤消

**revoke** 权力1[, 权力2]...[on 对象类型 对象名] **from** 用户1[, 用户2]...

基本表:	Select	Insert	update	Delete	CREATE INDEX	ALTER
视图:	Select	Insert	update	Delete		ALTER
属性:	Select	Insert	update	Delete		ALTER
DB:	Create Table					
All Privileges						
.						
.						
. 不同系统不一样						

### 3.8.2完整性控制

1. unique
2. Not null
3. 参照完整性
4. 相容性（范围）
5. 函数依赖

# 慕课讨论题

- NoSQL的时代背景下是否要学好SQL?

SQL语言，以其非过程化的特点而简单易用，以其自然化描述能力强而广泛用于关系型数据库中。当前NOSQL型数据库在大数据应用中广泛采用，有人称，“没有必要再学SQL语言”。你如何认为呢？

- 连接查询好还是嵌套查询好？

对于涉及多表的复杂查询语句，你认为是采用多表连接的方式来实现好呢？还是采用嵌套方式来实现更好？请给出理由。