高级数据库系统及其应用

第1部分　数据库系统基础

第2章 关系模型与关系数据库

xshxie@ustc.edu.cn

LOGO

第2章 关系模型与关系数据库

2.1 关系数据模型

2.1.1 关系模型基础

2.1.2 关系模型的约束及其表达

2.1.3 关系数据库

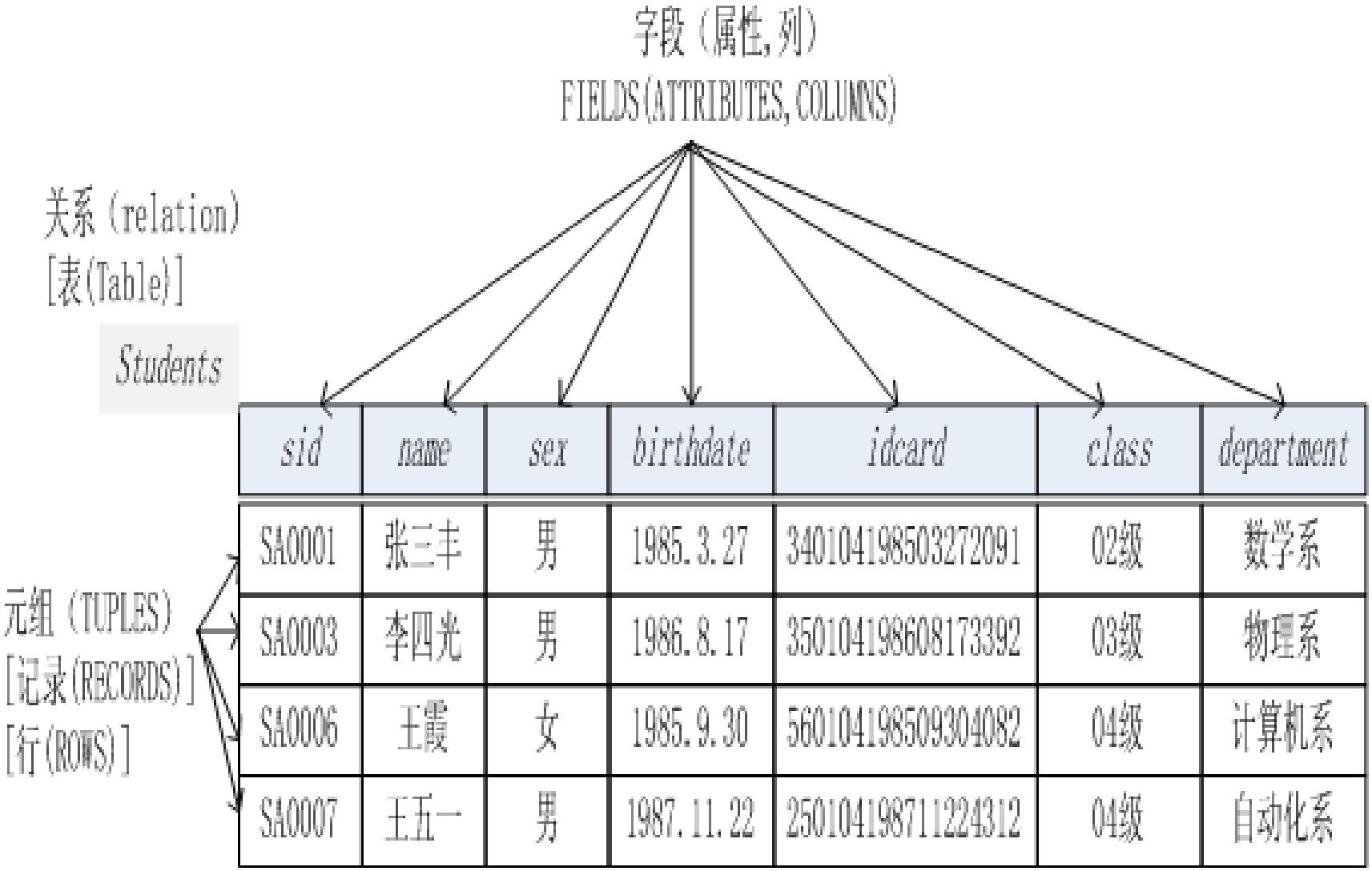
2.1.1 关系数据模型 (1)

关系模型的概念术语集

- 关系(relation)，又称为表 (table)
  - 每个关系好比一张二维的表格
  - 数据库被抽象为一组 “关系表”的集合
- 表列(column)，又称属性(attribute)或字段(field)
- 表行(row)，又称为记录(record)或元组(tuple)
- 关系模式(relation schema)
  - 形式化表示： $R(A_1[:dom(A_1)], \ldots, A_n[:dom(A_n)])$
  - schema定义了关系表的基本结构
- 关系实例或状态(instance or state)
  - 一组关系表数据行的集合，也称记录集或元组集
  - 单个元组的形式化表示： $t=<v_1,v_2,\ldots,v_n>$
  - 关系实例或状态的形式表示：$r(R)=\{t_1, t_2, \ldots, t_n\}$
- 关系数据库模式 (DB Schema) $S=\{R_1,R_2,\ldots,R_n\}$+ICs

Domain指定对列(属性)数据的限制

一个简单关系表的关系模式和关系实例示例(图2.1)



sid	name	sex	birthdate	idcard	class	department
SA0001	张三丰	男	1985.3.27	340104198503272091	02级	数学系
SA0003	李四光	男	1986.8.17	350104198608173392	03级	物理系
SA0006	王霞	女	1985.9.30	560104198509304082	04级	计算机系
SA0007	王五一	男	1987.11.22	250104198711224312	04级	自动化系

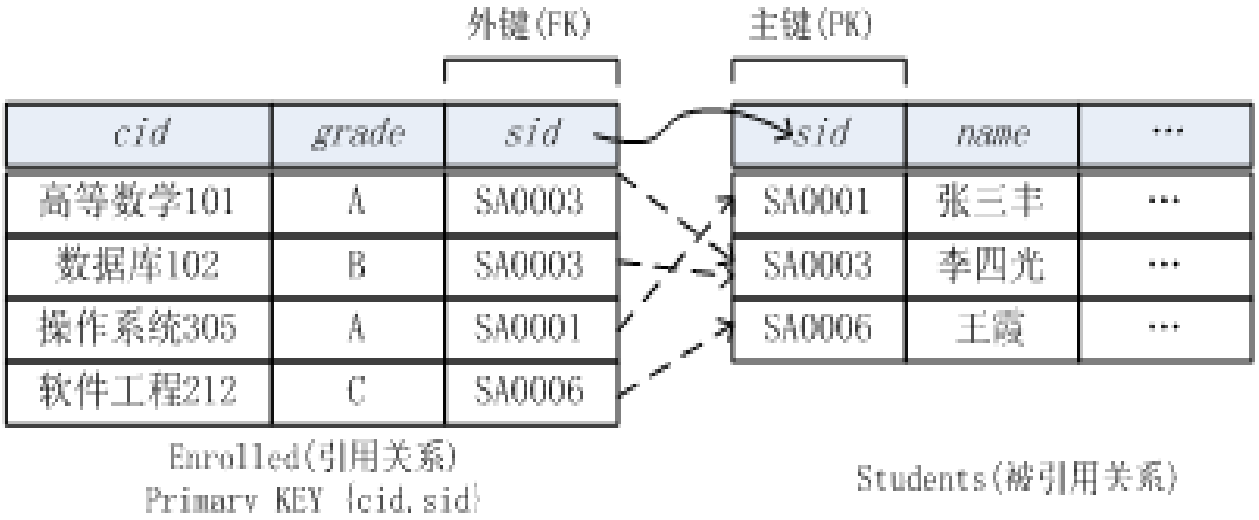
2.1.1 关系数据模型 (2)

- 关系模型的概念术语集
- 关系模型可表达的约束
  - 也称：完整性约束(Integrity Constraints, ICs)
    - DBMS必须能强制实施与DB模式有关的所有约束，以限制允许存储到DB的数据，确保DB中只有满足约束的合法数据；
    - ICs的主要类型：域约束、键约束和引用完整性约束；
  - 键约束
    - 超键(superkey,SK)
      - 能唯一标识关系R中每个元组的一个属性子集
      - 超键中可能会有冗余属性
    - 候选键(Candidate Key)--没有冗余属性的超键
  - 引用完整性约束(referential integrity constraint,RIC)
    - 指用来维护两不同关系元组间一致性的一种约束
    - 当某关系元组引用另一个关系元组时，只能引用已存在的元组
    - 外键(Foreign Key,FK),指定了两个关系R1和R2间的一个RIC

约束指定——应用实例

例 2.4 在 SQL-92 中，引用完整性通过 FOREIGN KEY 子句来指定。大学数据库中的学生选课注册关系 Enrolled 是一个与 Students 存在外键约束的关系，其关系模式定义为：

```
CREATE TABLE Enrolled( sid   CHAR(10),
                       cid   CHAR(20),
                       grade CHAR(10),
                       PRIMARY KEY(sid,cid),
                       FOREIGN KEY (sid) REFERENCE Students
                       ON DELETE CASCADE ON UPDATE NO ACTION);
```



2.2 关系操作与关系查询语言

2.2.1 关系代数

2.2.2 关系演算

关系模型中除了引入描述DB结构和约束的概念外，还引用了一组可操纵DB的操作----通过基于模型的专门语言，来表达模型操作。

◆本节介绍：基于关系模型的两种模型语言--关系代数/关系演算。
◆下节介绍：以这两种形式语言为基础的关系模型标准化语言(SQL)

本章查询表达说明用例模式（“水手值勤服务”）

Sailors(*sid*:integer, *sname*:string, *rating*:integer, *age*:integer);
Boats(*bid*:integer, *bname*:string, *color*:string);
Reserves(*sid*:integer, *bid*:integer, *day*:date);

◆关系查询语言的两种属性引用方法：
(1) 属性名
(2) 属性在关系模式中的位置或顺序号

“水手值勤服务”的一个简单模式实例

sid	sname	rating	age
22	sn22	7	45.0
31	sn31	8	55.5
58	sn58	10	35.0

(a)关系Sailors的实例S1

sid	sname	rating	age
28	sn28	9	35.0
31	sn31	8	55.5
44	sn44	5	35.0
58	sn58	10	35.0

(b)关系Sailors的实例S2

bid	bname	color
101	bn101	blue
102	bn102	red
103	bn103	green
104	bn104	red

(d)关系Boats的实例B1

sid	bid	day
22	101	10/10/98
58	103	11/12/96

(e)关系Reserves的实例R1

sid	sname	rating	age
22	sn22	7	45.0
29	sn29	1	33.0
31	sn31	8	55.5
32	sn32	8	25.5
58	sn58	10	35.0
64	sn64	7	35.0

(c)关系Sailors的实例S3

sid	bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
64	101	9/5/98

(f)关系Reserves的实例R2

2.2.1 关系代数

- 关系代数
  - 由一组操作符构成
    - 操作对象：1或2个关系(实例)
    - 返回结果：也是一个关系(实例) 。
  - 为关系模型操作提供了一个形式化的基础，也是RDBMS查询实现和优化的基础；
  - SQL结合并保留了很多关系代数的基本概念
- 关系代数中--操作符—来源类型

(1) 源自集合论的操作，包括并、交、差和叉积等

- 这些操作的适用性，源于“关系”本质上是元组集合；

(2) 专门为关系查询增加的操作，包括

- 选择、投影、连接，以及聚合等运算符。

（1）并、交、差、叉积运算

sid	*sname*	*rating*	*age*
22	sn22	7	45.0
31	sn31	8	55.5
58	sn58	10	35.0

(a)关系Sailors的实例S1

sid	*sname*	*rating*	*age*
28	sn28	9	35.0
31	sn31	8	55.5
44	sn44	5	35.0
58	sn58	10	35.0

(b)关系Sailors的实例S2

sid	*bid*	*day*
22	101	10/10/98
58	103	11/12/96

关系Reserves的实例R1

S1XR1=?

Sailors.sid	*sname*	*rating*	*age*	*Reservies.sid*	*bid*	*day*
22	sn22	7	45.0	22	101	10/10/98
22	sn22	7	45.0	58	103	11/12/96
31	sn31	8	55.5	22	101	10/10/98
31	sn31	8	55.5	58	103	11/12/96
58	sn58	10	35.0	22	101	10/10/98
58	sn58	10	35.0	58	103	11/12/96

关系代数操作符

(1) U,∩,-,×

- R U S={t| t∈R ∨ t ∈S}
- R ∩ S={t| t∈R ∧ t ∈S}
- R - S={t| t∈R ∧ t ∉ S}; S - R={t| t∈S ∧ t ∉ R}

(2)选择与投影操作

（2）选择操作与投影操作

- 选择操作表示$\sigma_c(R)$，c为条件表达式。
- 投影操作表示 $\pi_L(R)$，L为投影输出的属性子集列表$A_1', \ldots, A_m'$。

sid	*sname*	*rating*	*age*
28	sn28	9	35.0
31	sn31	8	55.5
44	sn44	5	35.0
58	sn58	10	35.0

关系Sailors的实例S2

$\sigma_{rating>8}(S2)$

sid	*sname*	*rating*	*age*
28	sn28	9	35.0
58	sn58	10	35.0

$\pi_{sname,age}(\sigma_{rating>8}(S2))$

sname	*age*
sn28	35.0
sn58	35.0

（3）重命名操作

S1xR1

Sailors .sid	*sname*	*rating*	*age*	*Reservies .sid*	*bid*	*day*
22	sn22	7	45.0	22	101	10/10/98
22	sn22					
31	sn31					
31	sn31					
58	sn58					
58	sn58					

sid1	*sname*	*rating*	*age*	*sid2*	*bid*	*day*
22	sn22	7	45.0	22	101	10/10/98
22	sn22	7	45.0	58	103	11/12/96
31	sn31	8	55.5	22	101	10/10/98
31	sn31	8	55.5	58	103	11/12/96
58	sn58	10	35.0	22	101	10/10/98
58	sn58	10	35.0	58	103	11/12/96

- 关系模式中，也允许用位置或序号来代替属性名
- 举例，试写出表达式 **ρ (R′(1→sid1, 5→sid2), S1×R1)** 的结果模式*R′*。

关系代数操作符

(1) U,∩,-,×

- R U S={t| t∈R ∨ t ∈S}
- R ∩ S={t| t∈R ∧ t ∈S}
- R - S={t| t∈R ∧ t ∉ S}; S - R={t| t∈S ∧ t ∉ R}

(2)选择与投影操作

- 选择操作$\sigma_c(R)$，c为条件表达式。
- 投影操作 $\pi_L(R)$，L投影的属性子集列表$A_1,\ldots,A_m$

(3)重名操作 ρ (R'(A_1→ A'$_1$, A_2→ A'$_2$,….), E)

(4)连接操作⋈$_c$

（4）连接操作（$\bowtie_c$）

- 用来合并两个关系中的信息——将两关系中相关的“元组对”合并成一个元组输出。
- 在概念上，$R \bowtie_c S = \pi_L(\sigma_c(R \times S))$

sid	*sname*	*rating*	*age*
22	sn22	7	45.0
31	sn31	8	55.5
58	sn58	10	35.0

关系Sailors的实例S1

sid	*bid*	*day*
22	101	10/10/98
58	103	11/12/96

关系Reserves的实例R1

S1⋈ R1　自然连接

sid	*sname*	*rating*	*age*	*bid*	*day*
22	sn22	7	45.0	101	10/10/96
58	sn58	10	35.0	103	11/12/96

关系代数操作符

(1) U,∩,-,×

(2)选择与投影操作

- 选择操作$\sigma_c(R)$，c为条件表达式。
- 投影操作 $\pi_L(R)$，L投影的属性子集列表$A_1,\ldots,A_m$

(3)重名操作 ρ (R'(A_1→ A'$_1$, A_2→ A'$_2$,….), E)

(4)连接操作$\bowtie_c$

- R ⋈ S 自然连接
- 概念上，$R\bowtie_c S = \pi_L(\sigma_c(R\times S))$。

（5）消除重复与排序运算

- **消除重复运算的表达式为 δ（R_B），其操作对象可以是包型关系或集合型关系。**
  - 消除重复是一个代价较大的操作，故在很多实际系统中，默认情况下的投影输出都不自动消除重复。
- **排序操作的代数符号表达式为$T_L(R)$。该操作用来将关系R的所有元组，按L所指定方式排序输出。**
  - L是一个属性表达列表，具有形如A_1 [asc|desc], A_2 [asc|desc], … 的形式

（6）分组与聚合操作

- **分组(grouping)与聚合(aggregating)操作的代数表达式为$\gamma_L(R)$，该操作将关系*R*的所有元组按*L*指定方式进行分组。**
  - *L*是一个属性列表,A_i,A_j,A_k,…
    - 首先按A_i分组, A_i分值相同的一个组中,如必要再按A_j值不同进一步分组
- **聚合关系与原关系*R*一般具有不同的模式结构。**
  - 聚合则是将同一分组中的若干元组—整合为1个元组
    - 聚合元组属性:分组属性 + [若干聚合计算属性]
    - 整合计算类: count数, avg, min,max

关系代数操作符

(1) U,∩,-,×

(2)选择$\sigma_c(R)$与投影操作$\pi_L(R)$

(3)重名操作 ρ (R'(A_1→ A'_1, A_2→ A'_2 ,....), E)

(4)连接操作$\bowtie_c$【 R ⋈ S；$R\bowtie_c S = \pi_L(\sigma_c(R \times S))$】

(5)消除重复$\delta(R_B)$与排序$T_L(R)$

- 排序准则，L形如, A_1 [asc|desc], A2 [asc|desc], …

(6)分组、聚合$\gamma_L(R)$

- L 是一个属性列表,A_i, A_j, A_k, …
- 聚合是将同一分组中的若干元组—整合为1个元组
  - 聚合元组模式: 分组属性 + [若干聚合计算属性]

2.2.2 关系演算(relational calculus)

- **关系演算为关系DB查询提供了一种高级描述性表示法。**
  - 它是一种形式化语言，其基础是被称为谓词(predicate)演算的数理逻辑分支——一阶谓词逻辑(First Order Logic, FOL)。
  - 在演算表达式中，不需给出如何应获取结果的操作次序指示信息，只需描述了所要求的结果信息(“要什么”)。
- **关系演算表达式描述了一个新的关系，这个新目标关系以变量形式来指定**
  - 变量取值范围: ①是新关系中的元组（元组演算）；②是新关系中的属性域（域演算）。
  - 元组演算，对商业化关系查询语言SQL发展有重要影响;
  - 域演算，则是QBE (Query-By-Example语言)的基础。

元组关系演算(Tuple Relational Calculus,TRC)

- **元组变量**
  - 取值范围限为特定关系元组的变量。
- **TRC查询的基本表达形式**
  - $\{t \mid p(t)\}$，
    - t 代表一个元组变量，而$p(t)$则是t应当满足的逻辑公式。
  - 查询结果是能使逻辑公式$p(t)$为真值的所有元组t集合。
  - 构造TRC查询表达的核心任务是给出逻辑公式$p(t)$, 本质上TRC逻辑公式是FOL公式的一个子集。
- **应用举例**
  - 要检索职级超过7的水手，
    - TRC表达为：$\{t \mid t \in \text{Sailors} \wedge t.rating > 7\}$

TRC查询的语法与语义

- 令*Rel*是关系名； 逻辑操作符*op* ∈{<,>, =,≤,≥,≠}；
- *R*和*S*是元组变量，*a*与*b*分别是*R*与S的一个属性；
- *p*和*q*是一个TRC公式。

- **TRC原子公式，是下面形式之一：**
  - $R \in Rel$ 是最基本原子公式，表达元组变量*R*的取值。
  - *R.a op S.b* 或 *R.a op const* 或 *const op R.a* 是比较型原子公式，表达*R*在其指定属性上的取值限定。
- **任何TRC公式可由如下任一方法递归地构造产生：**
  - 任何一个原子公式。

通常一个公式*p(R)*中会包含一个*R*∈*Rel*条件，以及一些关于*R*的量词限定表达。为简洁起见，

常用∃R∈Rel(p(R)) 替代∃R(R∈Rel∧p(R))

常用∀R∈Rel(p(R)) 替代∀R(R∈Rel ∧ p(R))

TRC查询示例（1）

- **(Q1) 找职级超过7的水手名字和年龄**
  - $\{p \mid \exists s \in Sailors\ (s.rating>7 \wedge p.name=s.sname \wedge p.age=s.age)\}$
  - 本查询展示了*TRC*确定类型的约定用法：
    - *P*被认为是具有*name*和*age*属性的元组变量，因表达中与*P*有关的提及属性只有这两个，且*P*未出现在任何其它关系中。
- **(Q2) 检索每个值勤记录对应的水手名字、船ID号和日期**
  - $\{p \mid \exists R \in Reserves\ \exists S \in Sailors(R.sid=S.sid \wedge p.bid=R.bid \wedge p.day=R.day \wedge p.sname=S.sname)\}$
    - 本查询展示了如何从两关系合并信息、构造新关系（即连接）的表达方法。
    - 对每个Reserves元组，要找Sailors中具有相同*sid*值的元组配对组合，再从组合元组中复制相应字段值来构造结果元组*P*。

TRC查询示例（2）

- **(Q3) 查在103号船上值勤过的水手名**
  - $\{ p \mid \exists S \in Sailors\ \exists R \in Reserves\ (R.sid=S.sid \wedge R.bid=103 \wedge p.sname= S.sname)\}$
  - 关系代数表达：$\pi_{sname}((\sigma_{bid=103}(Reserves)) \bowtie Sailors)$
- **(Q4) 查询曾在一条红船上值勤过的所有水手名字**
  - $\{ P \mid \exists S \in Sailors\ \exists R \in Reserves\ \exists B \in Boats\ (R.sid=S.sid \wedge B.bid=R.bid \wedge B.color=\text{'}red\text{'} \wedge P.sname=S.sname)\}$
  - 关系代数表达：

$\pi_{sname}((\sigma_{color='red'}(Boats)) \bowtie Reserves \bowtie Sailors$

域关系演算(Domain Relational Calculus, DRC)

- **DRC公式可按类似TRC公式的方式进行形式定义。这两类公式定义的主要差别是变量的取值范围。令*X*和*Y*是域变量。**
- **DRC原子公式是下面形式之一：**
  - $<x_1,x_2,\ldots,x_n> \in Rel$ *Rel*是含有*n*个属性的关系名，每个x_i , $1\le i\le n$，或是一个变量，或是一个常数。
  - *X op Y* 或 *X op constant* 或 *constant op X*
- **任何DRC公式可由如下任一方法递归地构造产生。**
  - 任何一个DRC原子公式；
  - ┐*p*(取反)，$p \wedge q$ (与连接构造)，$p \vee q$ (或连接构造)，$p \Rightarrow q$(蕴涵: 若p为真,q必为真)
  - $\exists X(p(X))$，*X*是$<x_1,x_2,\ldots,x_n>$ 。
  - $\forall X(p(X))$，X是$<x_1,x_2,\ldots,x_n>$ 。

DRC查询示例

- **(Q3) 查询被指派到103号船值勤的水手名**
  - $\{ <N> \mid \exists I,T,A(<I,N,T,A> \in Sailors \wedge \exists I_r, B_r, D (<I_r, B_r, D> \in Reserves \wedge I_r = I \wedge B_r=103))\}$
  - 若引入简记法，也可改写为：
  - $\{ <N> \mid \exists I,T,A(<I,N,T,A> \in Sailors \wedge \exists <I_r, B_r, D> \in Reserves(I_r = I \wedge B_r=103))\}$
  - 该式还可用如下更简洁的写法：
    - $\{ <N> \mid \exists I,T,A(<I,N,T,A> \in Sailors \wedge \exists D(<I,103,D> \in Reserves))\}$
- **(Q4) 查询曾在一个红船上值勤过的所有水手名字**
  - $\{ <N> \mid \exists I,T,A(<I,N,T,A> \in Sailors \wedge \exists <I, B_r, D> \in Reserves \wedge \exists < B_r, B_n, 'red' > \in Boats \}$

2.3 SQL语言

2.3.1 用DDL定义数据库

2.3.2 用DML操纵数据库

2.3.3 视图

CRUD操纵（DQL+DML）

SQL 分类

简称	全称	用途
DQL	数据查询语言 SELECT	用于从一张或多张表中获得数据
DML	数据操纵语言 Data Manipulate L	增删改表中的行。 CRUD=DML+DQL
DDL	数据定义语言 Data Defining L	创建、删除、修改 DB 对象
TPL	事务处理语言 Transaction Process L	同时执行多条 SQL 语句的情况
DCL	数据控制语言 Data Control L	获得许可，确定单个或组用户对 DB 对象访问
CCL	指针控制语言 Cursor Control L	用于对一个或多个表—单独行的操作

2.3.1 用DDL定义数据库

- 创建数据库：CREATE DATABASE <db-name>
- 删除数据库：DROP DATABASE <db-name>
- 创建表：CREATE TABLE <table-name>(attrname1 datatype1 contraint1,
  Attrname2 datatype2 contraint2,
  …
)

  CREATE TABLE emp (eid int primary key, ename varchar(10),
  hiredate date, deptno tinyint);
- 删除表：DROP TABLE <table-name>
- 修改表：ALTER TABLE <table-name> add COLUMN <新列表> <新列类型>

……

创建表示例——创建“水手”小模式

```
CREATE TABLE Sailors( sid       INTEGER,
                      Sname     CHAR(10),
                      rating    INTEGER DEFAULT(0),
                      age       REAL,
                      PRIMARY KEY (sid ),          --定义主键约束
                      CHECK (rating >=1 AND rating<=10)   ) ;
CREATE TABLE Boats (bid   INTEGER PRIMARY KEY,
                    bname VARCHAR(20) NOT NULL,
                    color   VARCHAR(10));
CREATE TABLE Reserves (sid INTEGER,
                       bid INTEGER,
                       day date,
                       PRIMARY KEY(sid,bid),
                       CONSTRAINT sidfk FOREIGN KEY (sid) REFERENCE Sailors
                       CONSTRAINT bidfk FOREIGN KEY (bid) REFERENCE Boats );
```

2.3.2 用DML操纵数据库

一、SQL基本查询

（一）SQL查询的基本形式

- SELECT [DISTINCT] *select_list*
- FROM *from-list*
- [WHERE <*condition*>]

（二）SQL基本查询的语义 （概念赋值策略）

- 计算出现在*from-list*中关系表的叉积；
- 删除叉积结果中不满足WHERE中条件的元组；
- 删除未出现在*select-list*中的列；
- 如果指定了DISTINCT，删除重复元组。

(与实际DBMS中的查询赋值策略相比，概念赋值策略通常不考虑效率，更强调概念性和易理解性)

SQL基本查询应用举例（1）

（Q1）查职级超过 7 的所有水手

```
SELECT   DISTINCT   S.sid, S.sname , S.rating, S.age
FROM     Sailors AS S    --可选关键字 AS，可用空格符替代，故可省略。
WHERE    S.rating>7
```

它等效于：

```
SELECT DISTINCT *  FROM  Sailors S  WHERE S.rating>7
```

（Q3）找曾在 103 号船上值勤的水手名字

```
SELECT   SName
FROM     Sailors, Reserves
WHERE    Sailors.sid= Reserves.sid AND bid=103
```

如一个属性列来源存在二义性（如 sid），则须加上所属表前缀以消除歧义。

若引入范围变量，则该查询可改写为：

```
SELECT   S.Name
FROM     Sailors S, Reserves R
WHERE    S.sid=R.sid AND bid=103
```

引入范围变量更有利于简化表达式，同时也有助于消除歧义。

SQL基本查询应用举例（2）

(Q4) 查曾在红船上值勤过的水手名

```
SELECT   S.sname
FROM     Sailors S, Reserves R, Boats B
WHERE    S.sid=R.sid AND R.bid=B.bid AND B.color='red'
```

(Q8) 查曾在红船或绿船上值勤过的水手名

```
SELECT   S.sname
FROM     Sailors S, Reserves R, Boats B
WHERE    S.sid=R.sid AND R.bid=B.bid AND (B.color='red' OR B.color='green')
```

(Q9) 查在红船和绿船上都值勤过的水手名

```
SELECT   S.sname
FROM     Sailors S, Reserves R1, Boats B1, Reserves R2, Boats B2,
WHERE    S.sid=R1.sid AND R1.bid=B1.bid AND
         S.sid=R2.sid AND R2.bid=B2.bid AND
              (B1.color='red' AND B2.color='green')
```

2.3.2 用DML操纵数据库

一、SQL基本查询

二、在SQL命令中使用表达式和字符串

- *select-list*中的每个项，除了可以是相关表的属性名外，还允许是形如 *<expr> AS 输出列名* 这种更一般的表达。
- WHERE语句中的条件项，也允许含一般表达式。

三、集合运算

- SQL提供了并(UNION)、交(INTERSECT)、差(EXCEPT)等三种集合操作，以扩展基本查询。

在SQL中使用表达式和字符串—举例

(Q11) 计算同一天曾在两条不同船上值勤的水手职级，这类水手的职级应自动上调一级。

```
SELECT   S.sname , S.rating+1 AS rating
FROM     Sailors S, Reserves R1, Reserves R2
WHERE    S.sid=R1.sid AND S.sid=R2.sid AND R1.day=R2.day
                      AND R1.bid <> R2.bid
```

SQL-92 中还引入了字串模糊匹配运算符 LIKE。LIKE 可使用类似正则表达式(Regular Expression)的文本模板(pattern)，来匹配搜索关系中的文本类型属性值。在 pattern 构造中，允许使用两个通配符“%”和“_”，前者代表任意个字符，后者代表单个任意字符。

(Q12) 查询名字以 B 开头结尾的、且至少有三字母的水手名

```
SELECT   S.name
FROM     Sailors S
WHERE    S.name LIKE 'B_%B';
```

在执行串比较匹配时，作为串一部分的任何空格都是有意义的。

集合运算 应用举例

(Q8') 查曾在红船或绿船上值勤过的水手名

显然，对前面的Q8、Q9这两查询，利用集合操作符，我们可得到更清晰、更好理解的表达。

```
UNION
SELECT S.sname
FROM   Sailors S, Reserves R, Boats B
WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='green'
```

(Q9') 查同时在红船和绿船上值勤过的水手名

```
SELECT S.sname
FROM   Sailors S, Reserves R, Boats B
WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red'
INTERSECT
SELECT S.sname
FROM   Sailors S, Reserves R, Boats B
WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='green'
```

实用示例---获取登录用户的动态菜单数据

```
select distinct m.menu_id, m.parent_id, m.menu_name, m.path, m.component,
    m.visible, m.status, ifnull(m.perms,'') as perms, m.is_frame, m.is_cache,
    m.menu_type, m.icon, m.order_num, m.create_time
from sys_menu m
left join sys_role_menu  rm  on m.menu_id = rm.menu_id
left join sys_user_role    ur    on rm.role_id = ur.role_id
left join sys_role           ro    on ur.role_id = ro.role_id
left join sys_user           u      on ur.user_id = u.user_id
where u.user_id = #{userId} and m.menu_type in ('M', 'C')
                                and m.status = 0  AND ro.status = 0
order by m.parent_id, m.order_num
```

2.3.2 用DML操纵数据库

一、SQL基本查询

二、在SQL命令中使用表达式和字符串

三、集合运算

四、嵌入查询

- 允许在条件项中引用“基于另一表的即时计算值”，是SQL的一个强有力特性，可有效提高查询表达能力。
- 为获得即时引用值，可通过在条件项中嵌入另一个SQL查询来实现。
- 内嵌的查询称为子查询，含子查询的查询称为嵌入查询。

嵌入查询应用举例

❖ 可用嵌入查询来重新表达Q3、Q4

(Q3) 查曾在 103 号船上执勤过的水手名字

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid IN (SELECT R.sid FROM  Reserves R WHERE R.bid='103')
```

(Q4) 查曾在红色船上执勤过的水手名字

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid IN (SELECT R.sid FROM  Reserves R WHERE R.bid IN
                  (SELECT B.bid  FROM  Boats B WHERE B.color='red'))
```

2.3.2 用DML操纵数据库

一、SQL基本查询

二、在SQL命令中使用表达式和字符串

三、集合运算

四、嵌入查询

五、聚合操作

- 除了简单存取数据，查询也经常需要执行一些汇总(summarization)计算，或分组合计。标准SQL支持以下五种基本聚合操作：
  - COUNT([DISTINCT]A): 计算属性A上(不同)值的个数
  - SUM([DISTINCT]A): 计算属性A上(不同)值的合计值
  - AVG([DISTINCT]A): 计算属性A上(不同)值的平均值
  - MAX(A): 计算属性A上的最大值；
  - MIN(A): 计算属性A上的最小值；
- 除count外，其它四个操作都要求属性A的域为数值型。

五、聚合操作

（一）在查询表达中简单应用聚合函数

(Q15)查职级=2的所有水手平均/最大/最小年龄/总人数

```
SELECT AVG(S.age) AS avg_age,
       MAX(S.age) AS max_age, COUNT(*) AS num
FROM  Sailors S
WHERE S.rating=2
```

(Q16)查比‘职级为2的最大年龄水手’年龄更大的水手

```
SELECT S.sname
FROM   Sailors S
WHERE  S.age > (SELECT MAX(S2.age)
                FROM Sailors S2 WHERE S2.rating=2)
```

（二）分组计算聚合值

五、聚合操作

（一）在查询表达中简单应用聚合函数

- (Q15、Q16)都是针对关系中被选择的所有元组进行聚合计算，这相当于只有一个分组(组中包含了所有被选择元组)。

（二）分组计算聚合值

(Q17)对龄超过 18 岁的所有水手，查有至少包括两人的各职级分组中年龄最小的水手。

```
SELECT      S.rating, MIN(S.age) as min-age
FROM        Sailors S
WHERE       S.age > 18
GROUP BY    S.rating
HAVING      COUNT(*) >1
```

- 分组聚合扩展查询语句表达的赋值策略

2.3.2 用DML操纵数据库

一、SQL基本查询

二、在SQL命令中使用表达式和字符串

三、集合运算

四、嵌入查询

五、聚合操作

六、外连接操作

- 是连接操作的一类重要变体，其结果实例中通常会有大量NULL值，可能产生比条件连接更多的结果元组
- 外连接又分“左外连接”、“右外连接”和“左右外连接”

六、外连接操作(outer joins)

❖ 左外连接(left outer join)

查询示例：

```
SELECT S.sid, R.bid, R.day
FROM  Sailors S
        NATURAL LEFT OUTER JOIN Reserves R
```

- 右表中那些不匹配的元组也会被加入到结果中。

❖ 全外连接(full outer join)

- 两个表中不匹配元组都会分别与另一表的空元组组合，加入到连接结果集中。

2.3.2 用DML操纵数据库

一、SQL基本查询

二、在SQL命令中使用表达式和字符串

三、集合运算

四、嵌入查询

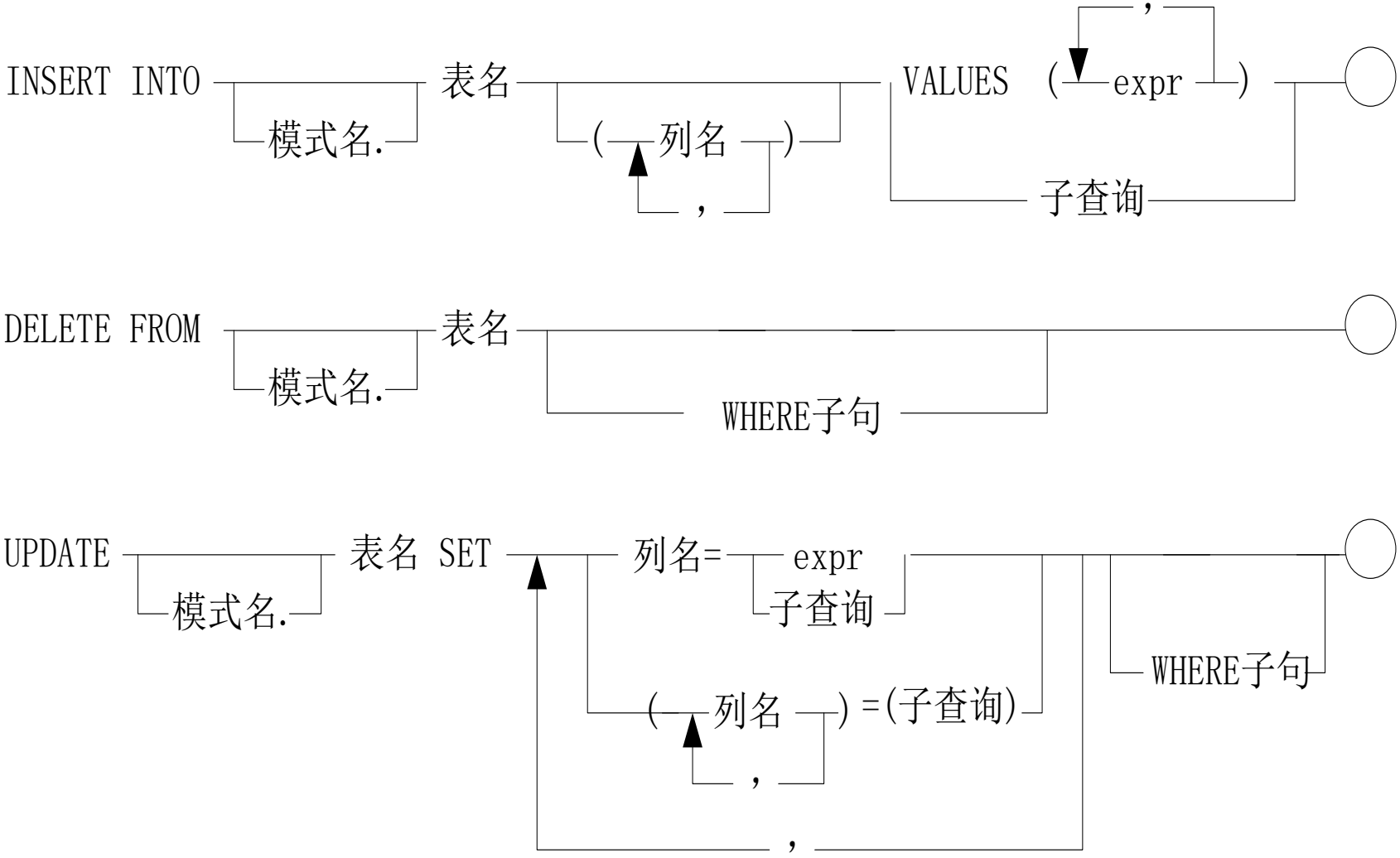
五、聚合操作

六、外连接操作

七、修改数据库

- 更新或修改DB主要包括三种基本操作：插入（insert）、删除(delete)和修改（update）。

修改数据库操作的基本语法说明



修改数据库操作——应用示例

```
INSERT INTO Sailors VALUES (88,'sn88',8, 50);

INSERT INTO Sailors(sid,name) VALUES (88,'sn88');

INSERT INTO Sailors SELECT * FROM Sailors2;

DELETE FROM <表名> [WHERE <元组选择条件>];

UPDATE Sailors SET sname='sn8888',age=38 WHERE sid=88;

UPDATE Sailors SET (sname,age)=
       (SELECT sname,age FROM Sailors WHERE sid=22)
```

2.4 应用关系数据库

2.4.1 SQL-DB视图、索引和存储过程

2.4.2 通过DB客户端工具访问数据库

2.4.3 基于API接口访问数据库:ODBC与JDBC

作为一种非过程化的、声明性语言，SQL为用户使用、操纵RDBMS带来了极大的方便。允许用户基于关系模型的概念层次来表达操作请求。用户不必关心数据在DB中的物理存储细节，以及具体查询实现过程。

本节将集中讨论：用户或应用程序如何访问数据库。

2.4.1.1 视图（∈DB对象）

- 让用户看都DB中所有数据是不合适的。
  - 首先，从安全的角度考虑，我们可能更希望各用户只看到与其有关的哪些数据。
  - 其次，不同用户，可能需要以特有的、个性化视角来观察所感兴趣的数据子集。
- 在DB中，通过引入视图(view)这一概念来解决以上问题。原则上，基于一组给定的实关系，我们能创建一组任意大小视图集合。
- 视图定义的基本形式为：
  - CREATE VIEW <*视图名*> AS <查询表达>
- 视图的物化----数据仓库(Dataware,DW)

2.4.1.2 索引（∈DB对象）

- 索引
  - 是一种能帮助我们有效找出满足指定条件数据的辅助数据结构，是一种特殊类型的表结构文件。
  - 创建索引是应用系统及DB优化的重要主题内容。
- 索引记录（元组）
  - 也称，索引项(index entry)，简记为k*
  - 索引项组织，除了可按索引键值顺序组织，还有按树结构(如B+树)、散列结构等进行组织。
- 创建索引语句
  - CREATE INDEX rtidx ON reserves (sid,bid); 或
  - ALTER TABLE reserves
    ADD INDEX rd_idx USING BTREE (sid, bid);

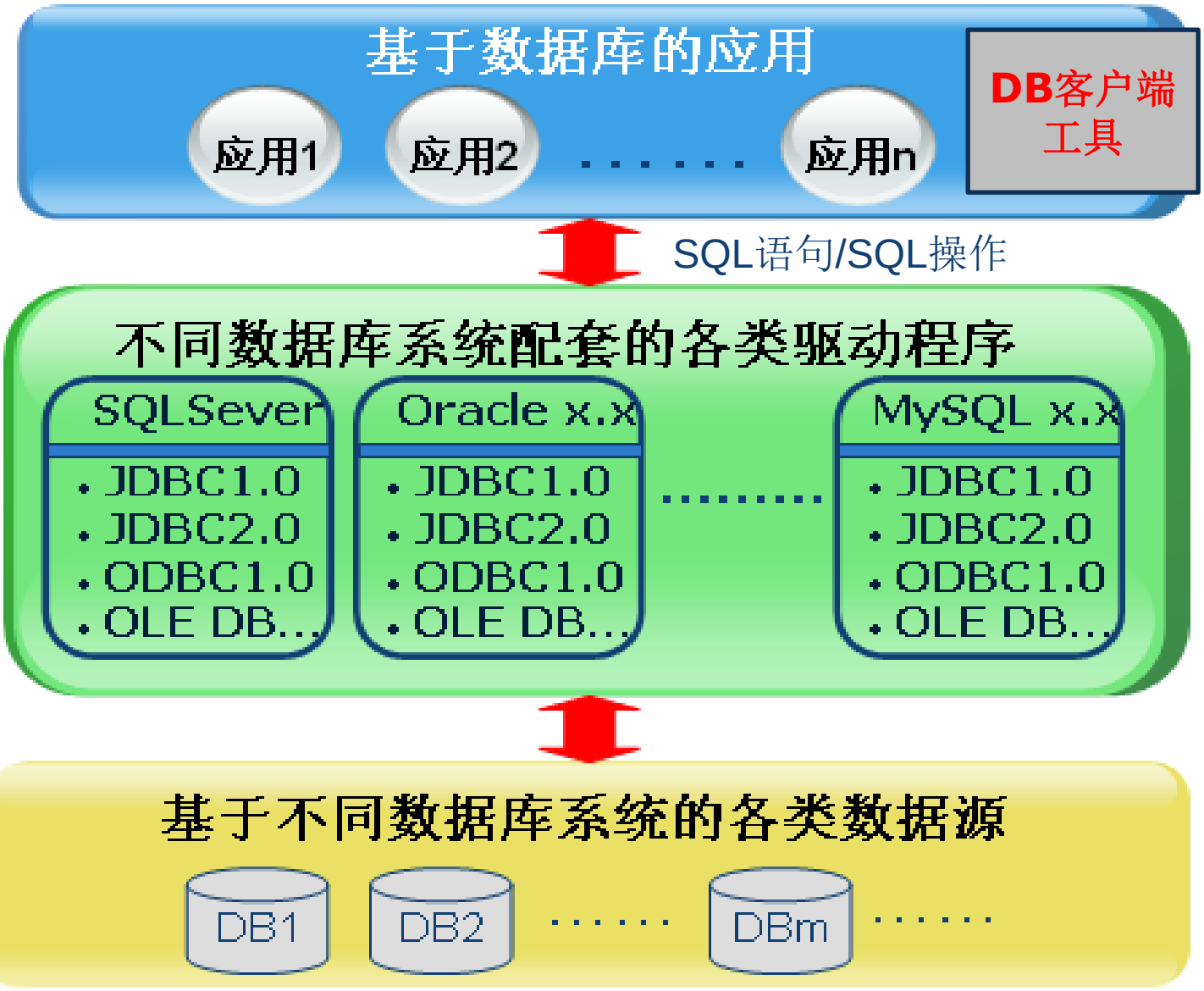
2.4.1.3 SQL-DB存储过程

- **DB存储过程是SQL的过程化扩展。**
- **存储过程的主要作用，在以下情况下很有用：**
  - 对一些同时为多个应用所需，且没有交互工作界面的DB处理过程，如果存储在DB服务器上供相关应用共享调用，将有助于减少重复工作，并增强软件的模块化特性。
  - 存储过程直接在服务端执行，有助于减少客户机与服务器之间的数据传输和通讯代价。
  - DB触发器也需要借助PSM过程实现。
- **示例 ---**
  - CREATE PROCEDURE `my_proc`()
    BEGIN
    insert into reserves values (4, 103, '2024-08-20','John');
    select * from reserves;
    END
  - call my_proc ;

2.4.2 基于API接口访问数据库:ODBC与JDBC

- **ODBC/JDBC是应用程序访问DB的标准接口(API)。**
  - 以可动态安装的驱动程序形式提供。
  - 程序中通过调用相关API函数访问DB
  - ODBC—Open DataBase Connection
  - JDBC—Java DataBase Connection
  - **参考《 java 访问DB编码模式》**

基于ODBC/JDBC方法的应用体系结构



Go语言操作DB

❖ 导入包与连接数据库

```
import (
        “database/sql”        //访问DB标准接口包
        _ “github.com/go-sql-driver/mysql     //匿名导入驱动
)
```

- 在导入一个驱动后，该驱动会自行初始化并注册到Golang的database/sql上下文中.
- sql.DB包中连接数据库函数的Open()函数:

```
db, err := sql.Open(“mysql”,
    “用户名:密码@tcp(IP：端口号)/db-name?   charset=utf8”)
```

Go语言操作DB

CRUD

- 直接调用Exec()方法来执行CRUD.

```
func (db *DB) Exec( sql_query string , args …inerface{})
                                        (Result, error)
result, err := db.Exec("INSERT INTO userinfo(uname, dept,
                    created) VALUES(?, ?, ?) " ,
                    "Steven", "销售部", "2025-01-01")
count, err := result.RowsAffected() //获取数据集的行数
```

- 通过预编译 Prepare + Exec()来执行CRUD

```
 func (db *DB) Prepare( sql_query string) (*Stmt, error)
stmt, err := db.Prepare("INSERT INTO userinfo(uname,
                        dept, created) VALUES(?, ?, ?) " )
result, err := stmt.Exec("Steven", "销售部", "2025-01-01")
```

Go语言操作DB

❖ 查询数据

```
stmt, err := db.Prepare("SELECT * FROM user_info WHERE uid< ? ")
rows, err := stmt.Query(10)
defer rows.close()
for rows.Next() {
    err := rows.Scan(&user.Uid, &user.Username, &user.Department, &user.Created)
    if err != nil {
        panic(err)
        continue
    }
    fmt.Println(*user)
}
```