

SQL (Structured Query Language)

1. Relational Algebra

1.0. Overview

1 关系操作：包括五个 Basic Operation

Operations	Notation	原关系	操作(得到新关系)
Projection	π	原关系	<div>1.保留想要的列(竖直过滤)</div> <div>2.去除重复行</div>
Selection	σ	原关系	<div>保留想要的行(水平过滤)</div>
Union	\cup	原关系1+原关系2	<div>1.合并两表所有Tuple</div> <div>2.去除重复项</div>
Intersection	\cap	原关系1+原关系2	<div>1.提取两个表都出现的Tuple</div> <div>2.去除重复项</div>
Set-difference	$-$	原关系1+原关系2	<div>移除关系2所含Tuple(在关系1中)</div>
Cross-product	\times	原关系1+原关系2	<div>Tuple无条件互相组合(笛卡尔积)</div>
Join	\bowtie	原关系1+原关系2	<div>1.Tuple无条件互相组合</div> <div>2.按条件Selection</div>

2 Condition Expressions:

类型	符号
Arithmetic Expressions	$>, <, >=, <=, =, !=$
AND/OR Clauses	$AND \rightarrow \wedge, OR \rightarrow \vee$

1. 示例: $\sigma_{rating \geq 9 \wedge age < 50}(S_2)$

```
SELECT * FROM S2
WHERE rating >= 9 AND age < 50;
```

2. SQL与C++等不同, 等于就是=/不是==

3. 不等于的两种表达: $!=$ 或者 $<>$

1.1. Projection(π) & Selection(σ)

1 Projection(π)

1. 操作: 仅保留Projection List中的列 \rightarrow 去除重复的行

- 特点: Projection is a costly operation, DB won't operate it by default.
- 示例: 注意SQL中可控制投影后去不去出重复项, 但Relational Algebra里默认去除重复项

```
SELECT * FROM Student; -- 选中所有列(左表)
SELECT age FROM Student; -- 选中年龄这一列, 但是SQL操作中默认不去除重复项(中表)
SELECT DISTINCT age FROM Student; -- 选中年龄这一列, 删除重复项(右表)
```

sid	sname	rating	age		age	
28	yuppy	9	35.0	$\pi_{\text{age}}(S)$	35.0	Removed duplicates
31	lubber	8	55.5		55.5	
44	guppy	5	35.0		35.0	
58	rusty	10	35.0		35.0	

age
35.0
55.5

2 Selection(σ)

- 操作: 仅保留Selection条件的行, 并且不会有任何重复行
- 示例: 在SQL中体现在WHERE上

```
SELECT * FROM Student WHERE Student.rating >= 9;
```

sid	sname	rating	age		sid	sname	rating	age
28	yuppy	9	35.0	$\sigma_{\text{rating} \geq 9}(S)$	28	yuppy	9	35.0
31	lubber	8	55.5					
44	guppy	5	35.0					
58	rusty	10	35.0		58	rusty	10	35.0

3 Projection(π) & Selection(σ)示例

```
SELECT sname, rating FROM Student WHERE Student.rating >= 9;
```

sid	sname	rating	age		sname	rating
28	yuppy	9	35.0	$\pi_{\text{sname, rating}}(\sigma_{\text{rating} \geq 9}(S))$	yuppy	9
31	lubber	8	55.5			
44	guppy	5	35.0		rusty	10
58	rusty	10	35.0			

- 一般采用先Selection后Projection的($\pi_A(\sigma_B)$)结构, 这是因为Selection不会改变表格结构
- 如果一定要用先Projection再Selection的($\sigma_A(\pi_B)$)结构, 要确保 $A \subseteq B$

1.2. Union/Set-Difference/Intersection

对应的SQL操作叫做Set Operations

0 条件(Union-Compatible):

1. 两表Column数一致(反而属性名不一定要相同), 相应的列的Data Type相同
2. 该条件适用Union/Set Difference/Intersection三者

1 Union:

1. 操作: 将两表上下拼接在一起, 并在Relational Algebra中默认消除重复项
2. 特性: 对称性(Symmetric)
3. 示例: 在SQL代码中体现在UNION上(但不默认消除重复项)

```
SELECT sid, sname, rating, age FROM S1 -- 此操作去除重复行
UNION
SELECT sid, sname, rating, age FROM S2;
SELECT sid, sname, rating, age FROM S1 -- 此操作保留重复行
UNION ALL
SELECT sid, sname, rating, age FROM S2;
```

sid	sname	rating	age	sid	sname	rating	age
22	dustin	7	45.0	28	yuppy	9	35.0
31	lubber	8	55.5	31	lubber	8	55.5
58	rusty	10	35.0	44	guppy	5	35.0
				58	rusty	10	35.0

$S_1 \cup S_2$
or $S_2 \cup S_1$

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

2 Set-Difference:

1. 操作: 从一个关系中, 删除另一个关系中存在的Tuple
2. 特性: 无对称性(Asymmetric)
3. 示例: SQL中并没有直接的求差集子句, 可以通过子查询变相实现

```
SELECT sid, sname, rating, age
FROM S1
WHERE (sid, sname, rating, age) NOT IN (
    SELECT sid, sname, rating, age
    FROM S2
);
```

sid	sname	rating	age	sid	sname	rating	age
22	dustin	7	45.0	28	yuppy	9	35.0
31	lubber	8	55.5	31	lubber	8	55.5
58	rusty	10	35.0	44	guppy	5	35.0
				58	rusty	10	35.0

$S_1 - S_2$

sid	sname	rating	age
22	dustin	7	45.0

3 Intersection: Compound Operator

1. 操作：找出两个关系中共存的Tuples，本质上是混合运算 $R \cap S = R - (R - S)$
2. 示例：注意操作有对称性；同样也需要用SQL子查询变相实现

```
SELECT sid, sname, rating, age
FROM S1
WHERE (sid, sname, rating, age) IN (
    SELECT sid, sname, rating, age
    FROM S2
);
```

sid	sname	rating	age	sid	sname	rating	age	
22	dustin	7	45.0	28	yuppy	9	35.0	$\xrightarrow{S_1 \cap S_2 \text{ or } S_2 \cap S_1}$
31	lubber	8	55.5	31	lubber	8	55.5	
58	rusty	10	35.0	44	guppy	5	35.0	
				58	rusty	10	35.0	
sid	sname	rating	age					
31	lubber	8	55.5					
58	rusty	10	35.0					

1.3. Cross product & Joins

SQL Inter Table Operations

1.3.1. Cross Product(\times)

1 操作:

1. 用一个表的每行，依次去扫描另一个表的每一行，输出组合
2. 结果会保留所有的列(即使重名了，也不要紧)

sid	sname	rating	age	sid	bid	day	
22	dustin	7	45.0	22	101	10/10/96	$\xrightarrow{R \times S}$
31	lubber	8	55.5	58	103	11/12/96	
58	rusty	10	35.0				
(sid)	sname	rating	age	(sid)	bid	day	
22	dustin	7	45.0	22	101	10/10/96	
22	dustin	7	45.0	58	103	11/12/96	
31	lubber	8	55.5	22	101	10/10/96	
31	lubber	8	55.5	58	103	11/12/96	
58	rusty	10	35.0	22	101	10/10/96	
58	rusty	10	35.0	58	103	11/12/96	

2 重命名 ρ :

1. 合并后的属性名称可能一样，由此需要重命名
2. 比如 $\rho(C(1 \rightarrow \text{sid1}, 5 \rightarrow \text{sid2}), R \times S)$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

sid1	sname	rating	age	sid2	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

3 SQL代码

```
SELECT * FROM R, S; -- Cross product between R, S
```

1.3.2. Inner Join(\bowtie) : Conditional Cross-Product

1 Inner Join(AKA Condition Join): $R \bowtie_{\text{Condition}} S = \sigma_{\text{Condition}}(R \times S)$

1. $R \bowtie_{(S.sid < R.sid)} S$ 中, 先计算 $R \times S$

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0			

$R \times S \rightarrow$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

2. 再筛选满足 $(R.sid < S.sid)$ 的行

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

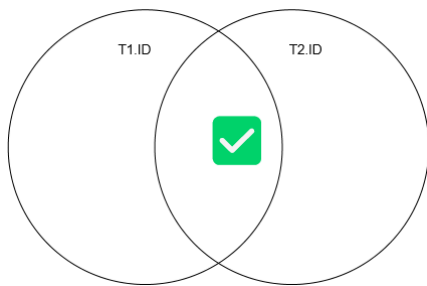
\rightarrow

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

3. SQL: ⚠️⚠️⚠️ 当两表中有Column名撞了, 应该在Column名前加上表名

```
SELECT * FROM R INNER JOIN S ON R.sid < S.sid;
```

2 Natural Join(AKA Join): 一种隐式的Inner Join



1. 先计算 $R \times S$, 找到两组(一组一或多个)名字相同的Column(此处为sid)

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0			

$R \times S$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

2. 筛选两同名列值相同的行, 然后只保留其中一列。即为 $R \bowtie S \equiv R \bowtie_{R.sid=S.sid} S$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

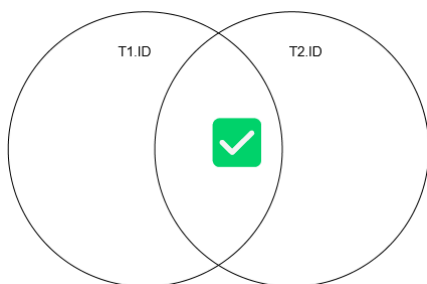
$R \bowtie S$

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

3. SQL: ⚠⚠⚠ 当无Column相同匹配时, `NATURAL JOIN` 语句会被执行为 Cross Product

```
SELECT * FROM R NATURAL JOIN S
```

3 Equi Join: 一种Inner Join的特殊情况, 比如 $R \bowtie_{(S_1.sid=R_2.sid)} S$ (条件是等式)



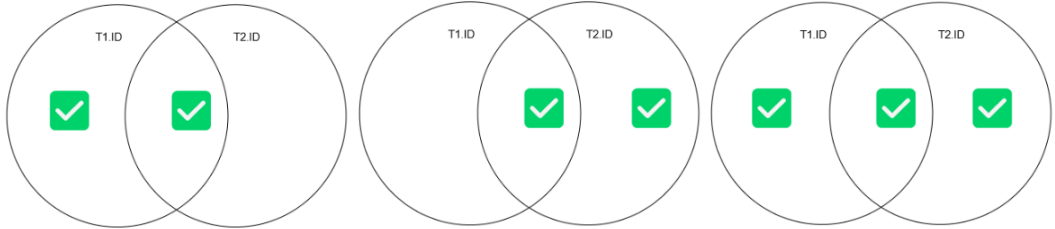
```
SELECT * FROM R INNER JOIN S ON R.sid = S.sid;
```

1.3.3. Outer Join : Include Every Record

1 概述

Outer Join	操作
Left Outer Join	保留左表中所有记录，右表中不匹配的用NULL代替
Right Outer Join	保留右表中所有记录，左表中不匹配的用NULL代替
Full Outer Join	保留左右两表中所有记录，两表中不匹配的用NULL代替

2 示意图: Left/Right/Full



3 示例

客户表	CID	属性1	属性2	账户表	CID	AID	属性3
\	1	X1	Y1	\	1	1	Z1
\	2	X2	Y2	\	1	2	Z2
\	3	X3	Y3	\	2	3	Z3
\	\	\	\	\	4	4	Z4

1. Left Outer Join: `Customer.CustomerID = 3` 时有表无匹配，所以代之以NULL

```
SELECT * FROM Customer LEFT OUTER JOIN Account
ON Customer.CustomerID = Account.CustomerID; -- 左外联，保留左表所有
CustomerID
```

CID	属性1	属性2	AID	属性3
1	X1	Y1	1	Z1
1	X1	Y1	2	Z2
2	X2	Y2	3	Z3
3	X3	Y3	NULL	NULL

2. Right Outer Join: `Account.CustomerID = 4` 时有表无匹配，所以代之以NULL

```
SELECT * FROM Customer RIGHT OUTER JOIN Account
ON Customer.CustomerID = Account.CustomerID; -- 右外联，保留右表所有
CustomerID
```

CID	属性1	属性2	AID	属性3
1	X1	Y1	1	Z1
1	X1	Y1	2	Z2
2	X2	Y2	3	Z3
4	NULL	NULL	4	Z4

3. Full Outer Join: 也可以认为是Right Outer Join结果 \cup Leftt Outer Join结果

```
SELECT * FROM Customer FULL OUTER JOIN Account
ON Customer.CustomerID = Account.CustomerID; -- 全外联，保留两表所有CustomerID
```

CID	属性1	属性2	AID	属性3
1	X1	Y1	1	Z1
1	X1	Y1	2	Z2
2	X2	Y2	3	Z3
3	X3	Y3	NULL	NULL
4	NULL	NULL	4	Z4

1.3.4. 复合Join

1 示例的ABC表格

A	AID	AName	B	BID	AID	BName	C	CID	CType	BName
\	1	AName1	\	101	1	BName1	\	1001	CType1	BName2
\	2	AName2	\	102	1	BName2	\	1002	CType2	BName3
\	\	\	\	201	2	BName3	\	\	\	\

2 Inner Join

```
A INNER JOIN B ON A.AID = B.AID
```

A.AID	A.AName	B.ID	B.AID	B.BName
1	AName1	101	1	BName1
1	AName1	102	1	BName2
2	AName2	201	2	BName3

2 复合Inner Join: 拿上面那个Join的结果再去Join

```
A INNER JOIN B ON A.AID = B.AID INNER JOIN C ON B.BName = C.BName
```


A.AID	A.AName	B.ID	B.AID	B.BName	C.CID	C.CType	C.BName
1	AName1	101	1	BName2	1001	CType1	BName2
2	AName2	201	2	BName3	1002	CType2	BName3

2. SQL Overview

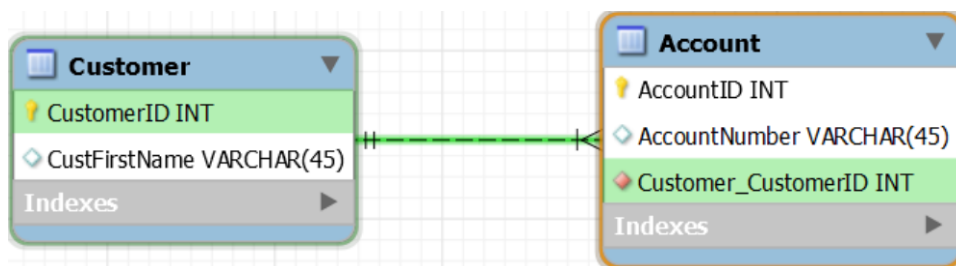
2.1. CRUD Commends

Create/Read/Update/Delete Commands

Command Type	功能	Description
DDL	Definition	定义并建立数据库
DML	Manipulation	维护并使用(查询)数据库, 获取数据的价值与信息
DCL	Control	控制用户对数据的访问权
Other	N/A	管理数据库

2.1.1. DDL Commands: Set up DB

1 创建表格操作 `CREATE`, 其实也就是Implementation操作



```

-- 创建Customer表
CREATE TABLE Customer (
  CustomerID INT PRIMARY KEY,
  CustFirstName VARCHAR(45)
);
-- 创建Account表
CREATE TABLE Account (
  -- 若表中现有最后一行ID是1001, 插入的新ID自动变成1002
  AccountID INT AUTO_INCREMENT PRIMARY KEY,
  -- 枚举数据类型, 只允许账户号码为三者之一
  AccountNumber ENUM('Num1', 'Num2', 'Num3');
  Customer_CustomerID INT,
  FOREIGN KEY (Customer_CustomerID) REFERENCES Customer(CustomerID)
);
  
```

1. 关于Key

- 要用 `PRIMARY KEY` 指定哪个变量时主键
- 要用 `FOREIGN KEY REFERENCES` 指定Foreign Key引用的哪个变量, 否则会 **Syntax Error**

2. 补充事项

- `AUTO_INCREMENT`: 用来标记主键字段，插入一个记录后被标记字段自动+1
- `ENUM`: 枚举数据类型类型，只允许属性为预设值

2 修改表格系列操作

操作	关键字
增加表格的Attributes	<code>ALTER TABLE <表名> ADD <属性名> <属性类型></code>
减少表格的Attributes	<code>ALTER TABLE <表名> DROP <属性名></code>
表格重命名	<code>RENAME TABLE <旧表名> TO <新表名></code>
快速清空表格记录(Can't Roll Back)	<code>TRUNCATE TABLE <表名></code>
慢速清空表格记录(Can Roll Back)	<code>DELETE * FROM <表名></code>
彻底删除(杀死)一个表格	<code>DROP * FROM <表名></code>

```
-- 添加属性：客户表中添加CustLastName属性
ALTER TABLE Customer ADD CustLastName VARCHAR(45);
-- 减少属性：账户表中删除AccountNumber属性
ALTER TABLE Account DROP AccountNumber;
-- 重命名：将Customer表重命名为NewCustomers
RENAME TABLE Customer TO NewCustomers
-- 快速清空表记录：清空所有客户的数据，不回退
TRUNCATE TABLE Customer
-- 慢速清空表记录：清空所有客户的数据，可回退
DELETE * FROM Customer
-- 彻底删除表格：把客户表的数据/客户表都根除
DROP TABLE Customer
```

3 查看表格操作: `VIEW` 视图，只是一种虚拟表(不存储在内存中)

```
-- 创建虚拟表格CustomerAccountView视图
CREATE VIEW CustomerAccountView AS
SELECT Customer.CustomerID, Account.AccountNumber
FROM Customer JOIN Account
ON Customer.CustomerID = Account.Customer_CustomerID;

-- 通过视图来查询数据，就像查询普通表格一样，简化了查询
SELECT * FROM CustomerAccountView;
```

2.1.2. DCL Commands: 用户与权限

1 创建/删除用户: `CREATE USER/DROP USER`

```
-- 创建用户, 初始密码为123
CREATE USER 'john'@'localhost' IDENTIFIED BY '123';
-- 删除john用户
DROP USER 'john'@'localhost';
```

2 分配/撤销用户权限: GRANT / REVOKE

```
-- 授予john在mydatabase数据库Table1表的SELECT和INSERT权限
GRANT SELECT, INSERT ON mydatabase.Table1 TO 'john'@'localhost';
-- 撤销john在mydatabase数据库Table1表的INSERT权限
REVOKE INSERT ON mydatabase.Table1 FROM 'john'@'localhost';
```

3 设置密码: SET PASSWORD

```
-- 设置新密码为0123
SET PASSWORD FOR 'john'@'localhost' = PASSWORD('0123');
```

2.1.3. Other: Database Administration

1 意外删除的恢复: BACKUP TABLE/RESTORE TABLE (二者其实并非标准SQL命令)

2 展示表格模式(Scheme): DESCRIBE <表名>

```
CREATE TABLE Customer (
    CustomerID INT AUTO_INCREMENT PRIMARY KEY,
    CustFirstName VARCHAR(45),
    DateOfBirth DATE
);
-- 查看Customer表的结构, 结果如下表
DESCRIBE Customer
```

Field	Type	Null	Key	Default	Extra
CustomerID	INT	NO	PRI	NULL	auto_increment
CustFirstName	VARCHAR(45)	YES		NULL	
DateOfBirth	DATE	YES		NULL	

3 USE <db_name>: 挑选进入哪个数据库来操作

2.2. SQL 语法特点(大小写问题)

1 SQL的属性名大小/关键词写不敏感(Case Insensitive), 但一般大写。如下两段查询含义相同

```
SELECT * FROM Furniture WHERE
(Type = 'Chair' AND colour = 'Black') OR (Type = 'Lamp' AND colour =
'Black');
select * from furniture where
(type = 'Chair' and colour = 'Black') or (type = 'Lamp' and colour =
'Black');
```

3. SQL Core: DML Commands

3.1. 变更表中内容

3.1.1. Insert Data: Insert Into

0 示例表格

CustID	FirstName	MiddleName	LastName	BusinessName	CustType
1	Peter	NULL	Smith	NULL	Personal
2	James	NULL	Jones	JJ	Company
3	NULL	NULL	Smythe	NULL	Company

1 显式插入：指名要给哪几个属性插入数据

```
-- 一次性可以插入一条数据
INSERT INTO Customer (FirstName, LastName, CustType) VALUES
("Peter", "Smith", 'Personal');

-- 也可以多条
INSERT INTO Customer (FirstName, LastName, CustType) VALUES
("Peter" , "Smith" , 'Personal'),
("James", "Jones" , 'Company' ),
("", "Smythe", 'Company' );
```



2 隐式插入：省略具体的属性，将提供的值按顺序依次插入属性

```
-- 插入第一条数据，DEFAULT子句会让CustID从0开始自动+1(此处+1后CustID=1)
INSERT INTO Customer VALUES
(DEFAULT, "Peter", "", "Smith", "", 'Personal');

-- 一次性插入多条数据
INSERT INTO Customer VALUES
(DEFAULT, "Peter", "", "Smith", "", 'Personal');
(DEFAULT, "James", NULL, "Jones", "JJ", 'Company' );
(DEFAULT, "", NULL, "Smythe", "", 'Company' );
```

3 从其他表格插入数据

```
-- 将所有Customer中的记录，全部插入NewCustomer中
INSERT INTO NewCustomer SELECT * FROM Customer;
```

👁️ REPLACE 和 INSERT 功能几乎一致，区别仅仅在于，当待插入记录的PK  已存在记录的PK  冲突

1. INSERT：操作失败，插入失败
2. REPLACE：操作合法，并且待插入记录覆盖冲突的已存在记录

3.1.2. Changes Existing Data: UPDATESET 结构

1 UPDATE 示例

```
-- Block1: 所有薪水小于100000的涨薪5%
UPDATE Salaried SET AnnualSalary = AnnualSalary * 1.05
WHERE AnnualSalary <= 100000;
-- Block2: 所有薪水大于100000的涨薪10%
UPDATE Salaried SET AnnualSalary = AnnualSalary * 1.10
WHERE AnnualSalary > 100000;
```

1. 子句执行的顺序会影响结果
2. 没有 WHERE 子句时，更新会应用到表格的每一行

2 CASE Command优化示例

```
UPDATE Salaried SET AnnualSalary =
CASE
    WHEN AnnualSalary <= 100000 THEN AnnualSalary * 1.05
    ELSE AnnualSalary * 1.10
END;
```

3.1.3. Deleting Existing Data: DELETE

1 简答例子

```
-- 删除表中所有记录，危险操作
DELETE FROM Employee
-- 删除表中满足条件的操作
DELETE FROM Employee WHERE Name = "Grace"
```

2 删除的外键约束

约束子句	尝试操作	执行操作
ON DELETE CASCADE	A表尝试删除一行	操作总被允许，引用该行的B表行都删除
ON DELETE RESTRICT	A表尝试删除一行	如果该行被B表引用，则删除操作被禁止

```
CREATE TABLE B (
    Bid INT PRIMARY KEY,
    Bname VARCHAR(50),
    Aid INT,
    FOREIGN KEY (Aid) REFERENCES A(Aid) ON DELETE CASCADE
    FOREIGN KEY (Aid) REFERENCES A(Aid) ON DELETE RESTRICT -- 二选一
);
```

3.2. 查询表中内容: SELECT FROM + XXX

3.2.1. SELECT 有关结构

1 最基本结构:

1. 投影, 原理详见关系代数

```
SELECT * FROM Student; -- 选中所有列, Heap Scan
SELECT age FROM Student; -- 选中年龄这一列, 但是SQL操作中默认不去除重复项
```

2. 关于 DISTINCT, 其后所有属性都要删除重复项

```
SELECT DISTINCT age FROM Student; -- 选中年龄这一列, 删除重复项
SELECT DISTINCT age, name FROM Student; -- 年龄+姓名都要DISTINCT, 删除重复项
```

2 聚合函数Aggregate Function

1. 概述

Function	Description
AVG()	Average Value
MIN()	Minimum Value
MAX()	Maximum Value
COUNT()	Number of Values(行数)
SUM()	Sum of Values

2. 示例

CustomerID	AccountID	Balance
1	1	200
1	2	100
2	1	400
3	1	300

- COUNT 示例

```
SELECT COUNT(*) FROM Customer; -- 表中有多少Tuples
SELECT COUNT(CustomerID) FROM Customer; -- 多少个CustomerID
SELECT COUNT(DISTINCT CustomerID) FROM Customer; -- 多少个Unique的ID
```

COUNT(CustomerID)	\	COUNT(DISTINCT CustomerID)
4	\	3

- AVG/MAX/MIN 示例

```
SELECT AVG(Balance) FROM Account; -- 所有账户余额的平均
SELECT MAX(Balance) FROM Account WHERE CustomerID=1; -- 用户1最高账户余额
SELECT SUM(Balance) FROM Account GROUP BY CustomerID; -- 各用户账户总余额
```

AVG(Balance)	\	MAX(Balance)	\	SUM(Balance)
250	\	200	\	300
\	\	\	\	400
\	\	\	\	300

3. SUM() 补充:

- 当用于数值类型属性时，会遍历每行求出并返回总和
- 当用于布尔类型属性时，会遍历每行(布尔真=1/布尔假=0)，返回总和

3 重命名子句 AS Clause

```
SELECT CustType, COUNT(CustomerID) FROM Customer -- 不重命名
SELECT CustType, COUNT(CustomerID) AS Count FROM Customer -- 重命名为Count
```

不重命名	CustType	Count(CustomerID)	重命名	CustType	Count
\	Type1	3	\	Type1	3
\	Type2	6	\	Type2	6

3.2.2. FROM 有关结构

1 最基本功能: 选定要操作的表格

```
SELECT * FROM R
```

2 跨表格操作: Cross-Product/Join, 原理详见关系代数部分

1. Cross-Product

```
SELECT * FROM R, S; -- Cross product between R, S
```

2. Join

```
SELECT * FROM R NATURAL JOIN S -- Nature Join

SELECT * FROM R INNER JOIN S ON R.sid < S.sid; -- Inner Join
SELECT * FROM R INNER JOIN S ON R.sid = S.sid; -- Equi Join

SELECT * FROM R LEFT OUTER JOIN S ON R.sid = S.sid; -- Left Outer
Join
SELECT * FROM R RIGHT OUTER JOIN S ON R.sid = S.sid; -- Right Outer
Join
SELECT * FROM R FULL OUTER JOIN S ON R.sid = S.sid; -- Full Outer
Join
```

3.2.3. +XXX有关结构

3.2.3.1. WHERE 有关结构

1 WHERE Clause: 本是上是一种Selection操作，过滤满足条件

```
SELECT * FROM Student WHERE Student.rating >= 9;
```

2 LIKE Clause: 与WHERE 配合使用，实现字符串的匹配(约等于)

1. 匹配的关键字

一个不确定的字符	多个不确定的字符
<code>_</code>	<code>%</code>

2. 匹配子句示例

Clause	CustomerName Matches	示例
<code>WHERE CustomerName LIKE 'a%'</code>	以a开始的	<code>axxxxx</code>
<code>WHERE CustomerName LIKE '%a'</code>	以a结束的	<code>xxxxxa</code>
<code>WHERE CustomerName LIKE '%a%'</code>	包含a的	<code>xxaxxx</code>
<code>WHERE CustomerName LIKE '_a%'</code>	a在第二位的	<code>xaxxxx</code>
<code>WHERE CustomerName LIKE 'a_ _%'</code>	以a开始，后至少两字符	<code>axx/axxx</code>
<code>WHERE CustomerName LIKE 'a%o'</code>	以a开始o结尾	<code>axxxxo</code>

3. SQL示例


```
SELECT CustLastName FROM Customer WHERE CustLastName LIKE "Sm%"
-- 匹配的会有Smith/Smyth/Smize.....
```

3.2.3.2. GROUP BY 有关结构

1 GROUP BY Clause:

1. 讲记录根据一个/多个属性，分为若干小组
2. 通常和聚合函数结合使用，为每个小组计算出独立结果
3. 示例

```
SELECT CustID, AVG(Balance) AS AveBalance
FROM Account
GROUP BY CustID;
```

CustID	Account	Balance	结果 → 查询	CustID	AveBalance
1	101	500	\	1	325
1	102	150	\	2	250
2	103	200	\	3	450
2	104	300	\	\	\
3	105	450	\	\	\

4. 复杂一些的例子: GROUP BY <复合属性>

```
SELECT Subject, Semester, COUNT(*) FROM Example_Table
GROUP BY Subject, Semester; -- Subject, Semester都相同才能分为一类
```

Subject	Semester	Attendee	结果 → 查询	Subject	Semester	COUNT(*)
ITB001	1	John	\	ITB001	1	3
ITB001	1	Bob	\	ITB001	2	2
ITB001	1	Mickey	\	IMK114	1	2
ITB001	2	Jenny	\	\	\	\
ITB001	2	James	\	\	\	\
IMK114	1	John	\	\	\	\
IMK114	1	Erica	\	\	\	\

2 HAVING Clause: 适用于 GROUP BY 体制的 WHERE

1. 作用:
 - 作为聚合函数的补充

HAVING Clause	WHERE Clause
后接关系Attribute有关条件	后接聚合函数(数结)果有关条件

◦ 用于对 GROUP BY 筛选出来的组

2. 示例：注意区分 WHERE 和 HAVING

CustID	Account	Balance
1	101	500
1	102	150
2	103	200
2	104	300
3	105	450

◦ WHERE <条件表达式>：直接过滤原始的数据(过滤数据)

```
SELECT CustID, Balance FROM Customer WHERE Balance > 400;
```

◦ HAVING <条件表达式>：对分组的结果进行过滤(过滤组)

查询结果	CustID	AveBalance	HAVING 后	CustID	AveBalance
\	1	325	\	1	325
\	2	250	\	3	450
\	3	450	\	\	\

```
SELECT CustID, AVG(Balance) AS AveBalance FROM Account  
GROUP BY CustID HAVING AveBalance > 300;
```

3.2.3.3. ORDER BY 有关结构

1 ORDER BY Clause:

1. 将查询得到的结果按照某一属性排序

```
SELECT Name, Type FROM Customer ORDER BY Name; -- 按名字(字典序)默认升序  
SELECT Name, Type FROM Customer ORDER BY Name ASC; -- 升序  
SELECT Name, Type FROM Customer ORDER BY Name DESC; -- 降序
```

2. 将查询得到的结果按照符合属性排序

```
SELECT Name, Type FROM Customer
ORDER BY Name DESC, Type ASC;
-- 先按Name降序排序
-- 再对于Name相同的Tuple, 按照Type升序排序
```

2 LIMIT/OFFSET Clause:

1. 含义

Clause	功能
LIMIT N	选取排序结果的前N个
OFFSET M	跳过M个排序结果
LIMIT N OFFSET M	跳过排序结果的前M个, 依次选取后面的N个

2. 示例:

CustLastName	Cust Type
Jones	Company
Jones	Company
Lam	Personal
Samson	Personal
Smart	Company
Smith	Personal
Smithies	Company
Smythe	Company
Unila	Company

```
SELECT Name, Type FROM Customer ORDER BY Name LIMIT 5; -- 红色
SELECT Name, Type FROM Customer ORDER BY Name LIMIT 5 OFFSET 3;
-- 绿色
```

3.3. 查询表中内容: Sub/Nesting Query

3.3.1. Subquery结构

0 概述

1. SELECT 结构相当于一个查询, 在一个 SELECT 中插入另一个 SELECT, 则后者就是前者子查询
2. 执行顺序: 先执行子查询 → 将子查询结果传给主查询 → 执行主查询
3. 相关子查询: 子查询可以直接使用主(外层)查询的列/值

1 在 SELECT 字句中插入子查询, 一般使用 AS 重命名子查询返回的列

```
SELECT
    name,
    (SELECT COUNT(*) FROM orders) AS order_count
FROM customers;
```

2 在 FROM 字句中插入子查询, 子查询此时相当于一个临时表, 所以必须使用 AS 重命名

```
SELECT tmp.average_sales
FROM (SELECT AVG(amount) AS average_sales FROM sales) AS tmp;
```

3 在 WHERE 子句中

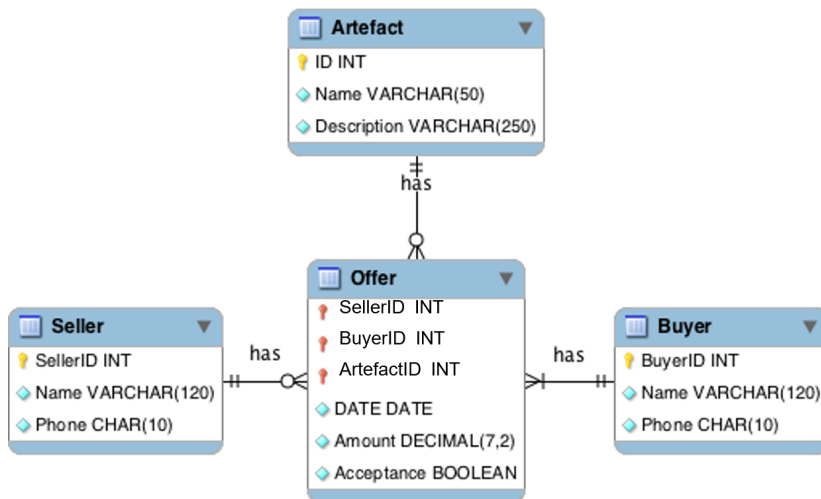
```
SELECT name, age
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

4 在 EXISTS 中，返回布尔值的特殊类子查询

```
SELECT name
FROM products
WHERE EXISTS (SELECT 1 FROM inventory WHERE inventory.quantity > 0);
```

3.3.2. Subquery 有关子句

0 示例表格



1. Artefact/Seller/Buyer表

AID	Name	Descript	/	SID	Name	Phone	/	BID	Name	Phone
1	Vase	Old Vase	/	1	Abby	0232	/	1	Magg	0333
2	Knife	Old Knife	/	2	Ben	0311	/	2	Nicole	0444
3	Pot	Old Pot	/	3	Carl	0333	/	3	Oleg	0555

2. Offer表

AID	SID	BID	Date	Amount	Acceptance
1	1	1	2012-06-20	81223.23	N
1	1	2	2012-06-20	82223.23	N
1	2	1	2012-06-20	19.95	N
2	2	2	2012-06-20	23.00	N

1 IN/NOT IN Clause: 记录是否在子查询的结果中

```
SELECT * FROM Buyer
WHERE BID IN (SELECT BID FROM Offer WHERE AID = 1)
```

1. 子查询结果: BuyerID = 1/2, 主查询结果

BID	Name	Phone
1	Magg	0333
2	Nicole	0444

2. 基于Join的优化: 执行的效率会更高

```
ELECT Buyer.*
FROM Buyer JOIN Offer ON Buyer.BID = Offer.BID
WHERE Offer.AID = 1;
```

2 ANY/ALL/EXISTS

1. ANY: 满足至少一个内部条件, 实际上可以把所有的 IN 改为 =ANY (至少一个等于)

```
SELECT empno, sal FROM emp
WHERE sal > ANY (SELECT...返回:Tuple1, Tuple2, Tuple3);

-- equals to
SELECT empno, sal FROM emp
WHERE sal > Tuple1 OR sal > Tuple2 OR sal > Tuple3;
```

2. ALL: 满足所有内部条件

```
SELECT empno, sal FROM emp
WHERE sal > ALL (SELECT...返回:Tuple1, Tuple2, Tuple3);

-- equals to
SELECT empno, sal FROM emp
WHERE sal > Tuple1 AND sal > Tuple2 AND sal > Tuple3;
```

3 EXISTS: 内部查询会返回True/False, 从而决定外查询执行/不执行

```
SELECT * FROM Buyer WHERE EXISTS(
  SELECT * FROM Offer -- 内查询中, SELECT什么不重要, 因为最后只会输出
  True/False
  WHERE Buyer.BuyerID = Offer.BuyerID AND ArtefactID = 1
)
```

1. 先遍历 Buyer (外表), 对其每一行向内 Offer (内表)表查询是否有行满足条件

```
Buyer.BuyerID = Offer.BuyerID AND ArtefactID = 1
```

2. 如果至少有一行条件满足, 则执行下列外查询, 把满足条件的行打印出来

```
SELECT * FROM Buyer
```

3. 一行行遍历+打印后, 结果为:

BID	Name	Phone
1	Magg	0333
2	Nicole	0444

3.4. 其他SQL子句

1 COALESCE(): 返回列表中第一个非NULL值

```
-- 遍历每一行, 如果A2非空则返回A2, 如果A2空则返回''(首个非空值)
SELECT A1, COALESCE(A2, '') AS A2New, A3 FROM users;
```

初始 表格	A1	A2	A3	查询 结果	A1	A2New	A3
\	John	NULL	Doe	\	John		Doe
\	Jane	A.	Smith	\	Jane	A.	Smith
\	Emily	NULL	Davis	\	Emily		Davis

2 LENGTH(): 求出一段字符串的长度

```
ORDER BY LENGTH(Steing)      -- String长度从小到大排列
ORDER BY LENGTH(Steing) ASC  -- String长度从小到大排列
ORDER BY LENGTH(Steing) DEC  -- String长度从大到小排列
```

3 CASE WHEN 语法

1. 语法

```
CASE WHEN <条件表达式> THEN <条件满足时返回这个值> <条件不满足时返回这个值>
END
```

2. 不重命名示例

```
SELECT
  id,
  amount,
  CASE WHEN amount > 100 THEN 'High' ELSE 'Low' END
  CASE WHEN amount > 100 THEN 'High' ELSE 'Low' END AS class -- 重命名
FROM orders;
```

不重命名	id	CASE	amount	重命名	id	CASE	class
\	1	High	150	\	1	High	150
\	2	Low	80	\	2	Low	80

不重命名	id	CASE	amount	重命名	id	CASE	class
\	3	High	120	\	3	High	120

4 Set Operation: 目的是合并两个查询结果，原理详见关系代数

```
SELECT sid, sname, rating, age FROM S1 -- 此操作去除重复行
UNION
SELECT sid, sname, rating, age FROM S2;
SELECT sid, sname, rating, age FROM S1 -- 此操作保留重复行
UNION ALL
SELECT sid, sname, rating, age FROM S2;
```