SQL (Structured Query Language)

1. Relational Algebra

1.0. Overview

1️⃣关系操作：包括五个Basic Operation

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

2️⃣Condition Expressions:

类型	符号
Arithmetic Expressions	`>, <, >=, <=, =, !=`
AND/OR Clauses	`AND → ∧, OR → ∨`

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

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

2. 注意事项
   - SQL与C++等不同，等于就是=/不是==
   - 不等于的两种表达：`!=`或者`<>`

1.1. Projection(π) & Selection(σ)

1️⃣Projection(π)

1. 操作：仅保留Projection List中的列→ 去除重复的行
2. 特点：Projection is a costly operation, DB won't operate it by default.
3. 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
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

$\pi_{age}(S)$ →

age
35.0
55.5
35.0
35.0

Removed duplicates →

age
35.0
55.5

2 Selection(σ)

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

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

sid	sname	rating	age
28	yuppy	9	35.0
~~31~~	~~lubber~~	~~8~~	~~55.5~~
~~44~~	~~guppy~~	~~5~~	~~35.0~~
58	rusty	10	35.0

$\sigma_{rating \geq 9}(S)$ →

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

3 Projection(π) & Selection(σ)

1. 注意事项
   - 一般先Selection后Projection的($\pi_A(\sigma_B)$)结构，这是因为Selection不会改变表格结构
   -

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

~~sid~~	sname	rating	~~age~~
28	yuppy	9	35.0
~~31~~	~~lubber~~	~~8~~	~~55.5~~
~~44~~	~~guppy~~	~~5~~	~~35.0~~
58	rusty	10	35.0

$\pi_{sname,\ rating}(\sigma_{rating \geq 9}(S))$ →

sname	rating
yuppy	9
rusty	10

1.

2. 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

0️⃣ 条件(Union-Compatible):

1. 两表Column数一致，相应的列的Data Type相同
   - 两表属性名不一定要相同，SQL默认取上表的名
2. 该条件适用Union/Set Difference/Intersection三者

1️⃣ 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)

```sql
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
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

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

$\xrightarrow[\text{or } S_2 \cup S_1]{S_1 \cup S_2}$

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

2️⃣ 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

```sql
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
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

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

$\xrightarrow{S_1-S_2}$

sid	sname	rating	age
22	dustin	7	45.0

3️⃣ 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$R \cap S \equiv R - (R - S)$
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
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

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

$\xrightarrow[\text{or } S_2 \cap S_1]{S_1 \cap S_2}$

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$(\times)$

1️⃣ 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
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

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

$\xrightarrow{R \times S}$

(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

2️⃣ 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$\rho(\mathrm{C}(1 \to \mathrm{sid1}, 5 \to \mathrm{sid2}), \mathrm{R}\times\mathrm{S})$

(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

$\longrightarrow$

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

   - $1 \to \mathrm{sid1}$的意思是：第一列改成sid1

3️⃣ SQL代码

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

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

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

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

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

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

$\mathrm{R}\times\mathrm{S}$ $\longrightarrow$

(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$(\mathrm{R.sid}<\mathrm{S.sid})$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

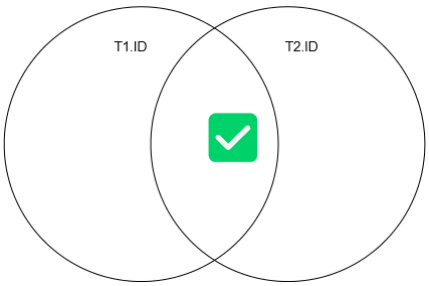
$\longrightarrow$

(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;
```

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



1. 原理/计算步骤
   - 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
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

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

$R \times S \longrightarrow$

(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

   - 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

$R \bowtie S \longrightarrow$

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

2. 注意事项：多个Column相同匹配时，`NATURAL JOIN`要多列相等

bid	sid	sname	rating	age
101	22	dustin	7	45.0
102	31	lubber	8	55.5
102	58	rusty	10	35.0

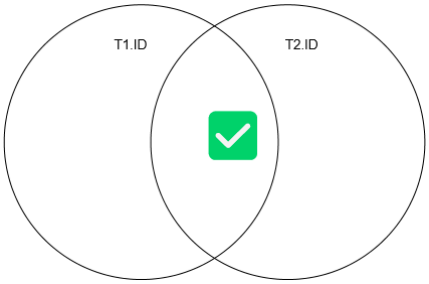
sid	bid	day
22	101	10/10/96
58	103	11/12/96

即为$R \bowtie S \equiv R \bowtie_{R.sid=S.sid \wedge R.bid=S.bid} S$

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

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

3️⃣ 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)



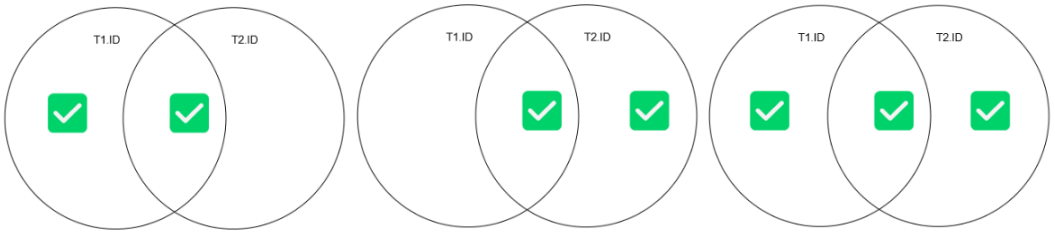

```
SELECT * FROM R INNER JOIN S ON R.sid = S.sid;
```

1.3.3. Outer Join : Include Every Record

1️⃣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

2️⃣示意图：Left/Right/Full



3️⃣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
```

D	属性1	属性2	D	属性3
1	X1	Y1	1	Z1
1	X1	Y1	2	Z2

CID	属性1	属性2	AID	属性3
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 ∪ 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

1️⃣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`	\	\	\	\

2️⃣Inner 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`

2️⃣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

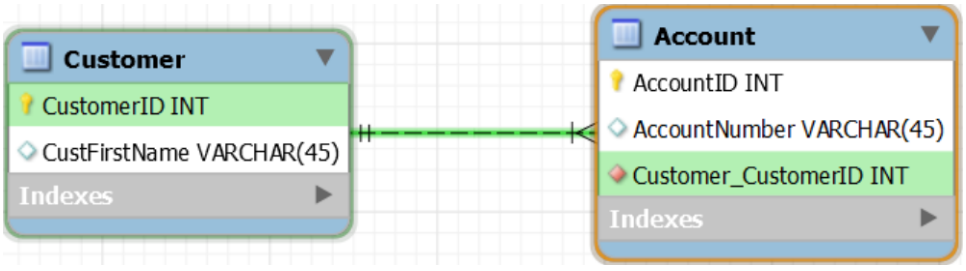
2.1. CRUD Commends

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

1️⃣创建表格操作`CREAT`，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

2️⃣修改表格系列操作

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

```
-- 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
```

3️⃣查看表格操作: `VIEW` 视图，只是一种虚拟表(不存储在内存中)

```sql
-- 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

1️⃣ 创建/删除用户: `CREATE USER/DROP USER`

```sql
-- 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';
```

2️⃣ 分配/撤销用户权限: `GRANT / REVOKE`

```sql
-- 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';
```

3️⃣ 设置密码: `SET PASSWORD`

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

2.1.3. Other: Database Administration

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

2️⃣ 展示表格模式(Scheme): `DESCRIBE <表名>`

```sql
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	

3️⃣ `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)

1️⃣SQL的属性名大小/关键词写不敏感(Case Inseneitive)，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');
```

3️⃣SQL的表名区分大小写

3. SQL Core: DML Commands

3.1. 变更表中内容

3.1.1. Insert Data: `Insert Into`

0️⃣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

1️⃣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'),
("Jamnes", "Jones" , 'Company' ),
(""      , "Smythe", 'Company' );
```

2️⃣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' );
```

3️⃣ 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 $\overset{\text{相同}}{\underset{\text{冲突}}{\longleftrightarrow}}$已存在记录的PK

1. `INSERT`：操作费费，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

1️⃣ `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

2️⃣ `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`

1️⃣ 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"
```

2️⃣ 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	如果改行被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

1️⃣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+姓名的组合要DISTINCT,删除重复项
```

2️⃣聚合函数Aggregare 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Uniqe的ID
```

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

- AVG/MAX/MAX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

3️⃣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

4️⃣合并列子句CONCAT：将两个不同的列，合并为一列

```
SELECT CONCAT(FirstName, LastName) AS FullName
FROM Name
```

3.2.2. FROM有关结构

1️⃣最基本功能：选定要操作的表格

```
SELECT * FROM R
```

2️⃣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

1个不确定的字符	0-M个不确定的字符
_	%

2. 匹配子句示例

[illegible]	CustomerName [illegible]	示例
WHERE CustomerName LIKE 'a%'	以a开始的	axxxxx
WHERE CustomerName LIKE '%a'	以a结束的	xxxxxa
WHERE CustomerName LIKE '%a%'	包含a的	xxaxxx
WHERE CustomerName LIKE '_a%'	a在第二位的	xaxxxx
WHERE CustomerName LIKE 'a_%_%'	以a开始，后至少两字符	axx/axxx

Clause	`CustomerName` Maches	示例
`WHERE CustomerName LIKE 'a%o'`	以a开始o结尾	`axxxxo`

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`体质的`WHERE`，只能和`GROUP BY`一起出现

1. 作用：作为聚合函数的补充，用于筛选`GROUP BY`出来的组
2. 对比：

`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

3. 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	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

1️⃣ `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

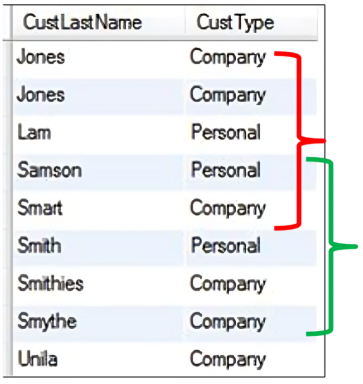
```sql
SELECT Name, Type FROM Customer
ORDER BY Name DESC, Type ASC;
 -- 先按Name降序排序
 -- 再对于Name相同的Tuple，按照Type升序排序
```

2️⃣ `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	CustType
Jones	Company
Jones	Company
Lam	Personal
Samson	Personal
Smart	Company
Smith	Personal
Smithies	Company
Smythe	Company
Unila	Company

```sql
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

0️⃣ 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

1️⃣ 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

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

2️⃣ 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

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

3️⃣ 在`WHERE`子句中

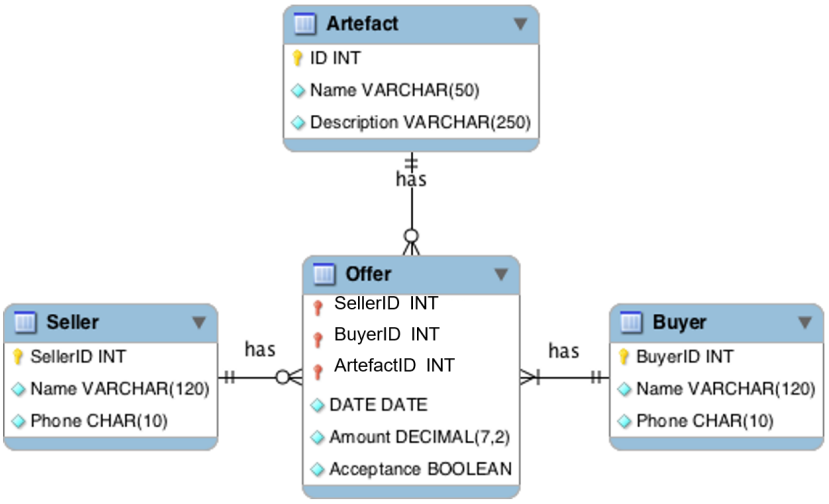
```sql
SELECT name, age
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

4️⃣ 在`EXISTS`中，返回布尔值的特殊类子查询

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

3.3.2. Subquery有关子句

0️⃣ 示例表格



1. Artefact/Seller/Buyer表

AID	Name	描述	\	SID	Name	Phone	\	BID	Name	电话
1	Vase	Old	\	1	Abby	0232	\	1	Magg	0333
2	Knife	Old	\	2	Ben	0311	\	2	Nicole	0444
3	Pot	Old	\	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

1️⃣ `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;
```

2️⃣ `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;
```

3️⃣ `EXISIS`：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

1️⃣ `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

2️⃣ `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
```

3️⃣ `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

4️⃣ 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;
```

4. 补充Data Type

4.1. 字符类型

1️⃣ 概述

类型	存储	备注
CHAR(M)	定长字符串	$M\in(0,255)$表示字符长度
VARCHAR(M)	变长字符串	$M\in(1,65535)$
BIT/BOOL/CHAR	同CHAR(1)	N/A
BLOB	二进制文件(音视频)	存储最多65535字节的0/1序列
TEXT	大段文字	最多65535个字符
ENUM(a,b,c...)	列表(多选一)	列表中最多65535个成员，索引从左到右/从0开始
SET(a,b,c...)	列表(多选多)	列表中最多64个成员

2️⃣ 补充

1. CHAR(M)中对于定长字符，如果长度不够则会用空格填充

```
Name CHAR(5) -- 如果赋值Name='Kan'则实际存储Name='Kan<空格><空格>'
```

2. TEXT中说的是最多65535字符，不是字节；一个字符可占1-4字节
3. SET(a,b,c...)示例

```
SET(a1,a2,a3) -- 可存储<空>/a1/a1a2/a1a2a3
```

4.2. 整数类型

1️⃣ 概述

Type	Signed Range	Unsigned Range
`TINYINT[(M)]`	−128 → 127	0 → 255
`SMALLINT[(M)]`	−32768 → 32767	0 → 65535
`INT[(M)]/INTEGER[(M)]`	.....	.....
`BIGINT[(M)]`	.....	.....

2️⃣ 示例：`TINYINT(4)`

1. 数字显示宽度为4
2. 如果存储的数字为12，则会显示为0012

4.3. 实数类型

1️⃣ 概述

Type	Precision	存储类型
`FLOAT[(M,D)]`	单精度	Binary
`DOUBLE[(M,D)]`	双精度(范围更大)	Binary
`REAL[(M,D)]`	双精度(范围更大)	Binary
`DECIMAL[(M[,D])]`	定点类型	字符串

2️⃣ 参数含义

1. `M`: 小数点左右加起来，一共有多少位
2. `D`: 小数位数

4.4. Data & Time Type

Type	Format	Range
`DATA`	YYYY-MM-DD	1000-01-01 → 9999-12-31
`TIME`	HH:MM:SS	-838:59:59 → 838:59:59
`DATETIME`	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00 → 9999-12-31 23:59:59
`TIMESTAMP`	YYYY-MM-DD HH:MM:SS	1970-01-01 00:00:00 UTC → 2037
`YEAR[4]`	YYYY	1901 → 2155

1️⃣ `TIMESTAMP`是DB记录的时间点，会将时间转换为当地时间

2️⃣ `NOW()`函数会以YYYY-MM-DD HH:MM:SS格式返回当前时间