SQL 语法

任何情况下, 一个 table 都不能直接当做内容或判断条件使用, 必须 select
引号只能用单引号, 小数可用 Float 或 Real

Create Table

```
Create Table MyTest (
 a Integer Primary Key,
 b Varchar2(20),
 c Integer Not Null,
 d Integer Not Null,
 e Integer Initially Deferred Deferrable,
 // 可选, 推迟检查, 插入时还没有就要
 f Integer,
 Primary Key (a), // or like this
 Unique (b,c),
 Check (f>10),
 Foreign Key (e) References E(eid)
 // 可加 On Delete Cascade
);
```

Select

```
Select ... // a | A.a | A.a As a1 | * |
           // Count(*) | Distict a
From ... // A | City C | A, B
Where ... // id=1 | x>3
Order By A.a DESC, B.b ASC;
```

Group By

可选 Min Max Sum Count(*) Avg
在 select 里面的, 除了 group by 的那一列, 不能直接使用那一列, 必须使用 min sum 等, 即使能够确保在里面相同也不行, 可以在 group by 那里输入两列 (Group By b1, b2) 解决

```
Select ..., SUM(b2), MAX(b4) From ...
Group By b1
Having SUM(b3)>10; // 可选
// order by 要放在最后
```

Drop

```
Drop Table Users Cascade Constraints;
// 如果没有 Cascade Constraints,
// 如果别人依赖了这个表, 就会报错
Drop View UserInfo; // No Cascade Constraints
Drop Sequence seq1; // No Cascade Constraints
Drop Trigger T1;
// 需要注意大概率因为依赖了 table 就已经被删了,
// 不要再删, 会报错
```

With, Tmp Table

对于正常的 select, with 必须放在外面, 并且就在前面, 只对后面一个有效, 但是可以放在 select 里面的括号里面使用
对于 create view, with必须放在里面 (就是 create view 的后面一行)

```
With Tmp1 As (
Select ...
), // 后面还有 As 就要逗号
Tmp2 As (
Select ...
) // 没有了不要逗号
Select ...

Create Global Temporary Table AAs (
// 后面都一样, 用起来也和正常的一样
```

运算符

Union, Intersect, Minus (Minus 和 Except 一样), 合并时不看属性名, 就按顺序, 合并后的名称以第一个表的为准 (会自动去重, 即使一个表里重复也会被去掉, 未验证, 仍然应该使用select distinct)
Is Null, Is Not Null, And, Or, Not, In (必须 select), **Not In Min Max Sum Count(*) Avg**, 这些即使不是在 group by 里面也能用, 输出一个值, 可以直接和数值比较
等于是单个等号 (=), 不等于用 <>
Null 进行 and or, 在无法确定时结果是 null, 如果可以确定就是那个结果, Where null 视为 false

Exists R : true if R is not an empty relation
Unique R : true if no duplicates in R
attr **> Any R** : some element of R satisfies the condition that attr > that element
这些均未测试, 请使用 Count(*) Min Max

Insert Into

```
Insert Into Pairs (id1, id2) // 调整顺序时需要
Select ... // 接下来是 select 内容
Insert Into Pairs // 严格按照顺序, 不会考虑属性名
Select ...
Insert Into Pairs Values (1,3); // 一次只能一个
```

Create View (可以直接 select)

```
Create View UserInfo As
Select ... // 接下来是 select 内容
```

Alter Table

```
Alter Table photos
Add ( // or Drop Column Email;
      // or Modify Date Integer;
    Foreign Key (aid) References Albums(aid)
);
```

Create Trigger (注意最后有斜杠)

```
Create Trigger Order_Friend_Pairs
 Before Insert On Friends
 // or After, or Update Delete
 For Each Row
  Declare temp Integer; Begin
   If :new.u1_id > :new.u2_id Then
    Temp := :new.u2_id; // 可为 OLD
    :new.u2_id := :new.u1_id;
    :new.u1_id := Temp;
   End If; End;
/
```

Create Sequence (注意最后有斜杠)

```
Create Sequence seq_cid
    Start With 1 Increment By 1
/
Create Trigger cid_tgr
 Before Insert On Cities
  For Each Row
   Begin Select seq_cid.nextval
   Into :new.cid From Dual; End;
/
```

Update

```
UPDATE B
SET b2 = 100, b3 = 200 // 可以只有1个,
                       // 可以用 select 表达式
WHERE b1 = 1; // 没有where就是整个表全部update
```

如果有依赖于它的, 会无法删除, 如果使用了on update cascade, 那个值也会更新

Delete

```
DELETE FROM students WHERE age > 18;
```

如果有依赖于它的, 会无法删除, 如果使用了on delete cascade, 那个记录就会删除
On Delete Cascade, On Delete Set Null, On Update Set Default

Selection σ: Selects a subset of rows from relation
Projection π: Deletes unwanted columns from relation } 1 reln
Cross-product ✕: Allows us to combine two relations
Set-difference −: Tuples in reln. 1, but not in reln. 2
Union ∪: Tuples in reln. 1 and in reln. 2 } 2 reln

$\sigma_{condition}(relation)$ $\sigma_{sport='gymnastics' \wedge country='USA'}(Athlete)$
$\pi_{sport,country}(Athlete)$
$\rho(AFTER, BEFORE)$ $\rho(A1, Athlete) \times \rho(A2, Athlete)$
$\rho(C(A1.name \rightarrow name1, A2.name \rightarrow name2),$
$\rho(A1, Athlete) \times \rho(A2, Athlete))$

Conditional join or Θ-join (e.g., $R \bowtie_{\{R.sid<S.sid\}} S$)
$R \bowtie_c S = \sigma_c(R \times S)$, where c is a condition

Equijoin e.g., $R \bowtie_{\{R.sid=S.sid\}} S$
- Join condition consists only of equalities on one or more columns (S.sid column dropped as a duplicate)

Natural Join $R \bowtie S$
- equijoin occurs on all the columns with the same name. Duplicated columns dropped.

Solution 1: $\pi_{sname}((\sigma_{bid=103} Reserves) \bowtie Sailors)$

Solution 2:
$\rho(Temp1, \sigma_{bid=103} Reserves)$
$\rho(Temp2, Temp1 \bowtie Sailors)$
$\pi_{sname}(Temp2)$

Solution 3: $\pi_{sname}(\sigma_{bid=103}(Reserves \bowtie Sailors))$

A/B = {<x> | ∀<y>∈ B, <x, y>∈A}

cid	bname
c1	b1
c1	b2
c1	b3
c1	b4
c2	b1
c2	b2
c3	b2
c4	b2
c4	b4

A

bname
b2

B1

bname
b2
b4

B2

bname
b1
b2
b4

B3

cid
c1
c2
c3
c4

A/B1

cid
c1
c4

A/B2

cid
c1

A/B3

Basics

Physical / Conceptual / External Schema
Conceptual 只能是原始的表格，稍微改一点 (哪怕是去掉一列) 都不行

Logical / Physical independence
Logical: External / Conceptual schema interface
Physical: Conceptual / Physical schema interface

Primary key: 表中唯一的, 用户定义的
Candidate key: 最小的唯一组合, 去掉一个就不唯一, 可作为潜在的 primary key (we can not say anything is definitely a candidate key, 即使是 ID 都不行)
Partial key: weak entity 的确定元素

Schema: Students(sid:string, name:string, login:string, age:integer), 类似于这样的

没有 bool 类型, 只能 1=1

ER Diagram

方框: Entity 菱形: 关系 圆圈: 属性 (Primary key 用下划线)
箭头: 至多一个 粗线: 至少一个
双线: 适用于同种类型两个参与的, 比如 交朋友 或 发消息 或 A代表B (可在线旁边标记对应属性)
Weak Entity (本身不能确定, 依赖另一个对象可确定): 加粗, 对应的方框和菱形都要加粗, key 用虚线下划线
IS A: 三角形, 箭头朝向更加广泛的类, 下面的边接向更具体的
Relationship with relationships (Aggregation): 虚线把一组图表框起来, 再连一根线到菱形, 注意, 这个关系本身也需要一张表

Normalize

1 NF: 只有一个表格, 每条记录都包含所有属性
BCNF (Boyce-Codd Normal Form): for all X → A in F+, ~~then A ⊆ X (trivial FD)~~, or X is a super key (super key 等价于 unique)
3 NF: for all X → A in F+, ~~then A ⊆ X (trivial FD)~~, or X is a super key, or A is part of some (minimal) key
BCNF 一定是 3 NF, 反过来不一定

Functional Dependencies: X → Y, 知道了 X 就知道 Y
F+: Closure of F = Set of all valid FDs, 就是所有的, 自己推出自己、被包含都算 , = Set of all FDs that can be derived from F using Armstrong's axioms

Armstrong's Axioms:
Reflexivity: If Y⊆X, then X → Y (trivial dependency)
Augmentation: If X → Y, then XZ →YZ for any Z
Transitivity: If X → Y and Y → Z, then X → Z
Union: If X → Y and X → Z, then X → YZ
Decomposition: If X → YZ, then X → Y and X → Z

Lossless Join / Dependency Preservation
X 和 Y 可以 Lossless Join 当且仅当 X ∩ Y → X, or X ∩ Y → Y
R has a dependency-preserving decomposition to X, Y if F+ = (Fx ∪ Fy)+
Lossless-join, dependency-preserving decomposition of a relationship R into a collection of 3NF relations is always possible

Algorithm for BCNF (relation R, FDs F)
done = false;
result = {R};
compute F+;
while (not done) do
  if ∃ Ri ∈ result and Ri is not in BCNF
  let α→β be a nontrivial FD that holds in Ri such that (α→Ri) ∉ F+ and α ∩ β=∅ ;
  result=(result−Ri) ∪ (Ri−β) ∪ (α,β) ;
  else done=true ;

Others (不太重要的)

列名: sql 对列名极度不敏感, ~~基本~~都是直接按照顺序匹配 (即使交换了顺序属性名相同), 甚至一个表中两列的列名可以相同 (select , 有重复), 列名~~凡~~乎只决定显示效果, 内部运算不看
select 没用 as, 会直接保留原来的名称, 如果相同就会有两个相同的列名
Union Intersect 等, 列名取决于第一个的列名
所有的列数量和数据类型必须匹配

Unique 遇到 Null 的情况
UNIQUE attributes can be NULL, unless NOT NULL is specified
一条记录只要有 Null, 就不受任何重复限制, 因为 Null 是未定义的, 可能是任何值, 就无法判断相等

Join
SELECT S.sname
FROM Sailors S JOIN Reserves R ON S.sid = R.sid
WHERE R.bid = 103;
Join Inner Join 和直接逗号没有区别, 但是 Join 必须要有 On, 哪怕就 On 1=1 也行

1. **INNER JOIN (内连接)**: 只返回两个表中匹配的行。如果表 A 中的行与表 B 中的某行按照连接条件相匹配，那么这些行的组合就会出现在查询结果中。
2. **LEFT JOIN (左连接) 或 LEFT OUTER JOIN**: 返回左表（LEFT JOIN 关键字左边的表）的所有行，以及右表中匹配的行。如果左表的行在右表中没有匹配的行，则结果中右表的部分将包含 NULL。
3. **RIGHT JOIN (右连接) 或 RIGHT OUTER JOIN**: 与 LEFT JOIN 相反，它返回右表的所有行以及左表中匹配的行。如果右表的行在左表中没有匹配的行，则结果中左表的部分将包含 NULL。
4. **FULL JOIN (全连接) 或 FULL OUTER JOIN**: 返回左表和右表中所有的行。当某行在另一表中没有匹配时，会在另一表的列位置显示为 NULL。这 essentially combines the results of both LEFT JOIN and RIGHT JOIN.

关于 outer, inner 的相对值, 所以 left right full 一定都是 outer, 没有其它情况

Sailors

sid	sname	rating	age
22	dustin	7	45.0
58	rusty	10	35.0

Reserves

sid	bid	day
22	101	10/10/99

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

Result

sid	bid
22	101
58	null

Similarly:
- Right Outer Join
- Full Outer Join

Java

```
Connection connection = null; // add try here
connection = DriverManager.getConnection("jdbc:sqlite:sample.db");
Statement statement = connection.createStatement();
statement.setQueryTimeout(30); // set timeout to 30 sec.
statement.executeUpdate("drop table if exists person");
statement.executeUpdate("create table person (id integer, name string, age double)");
statement.executeUpdate("insert into person values(1, 'leo', 22.3)");
statement.executeUpdate("insert into person values(10, 'jtc', 25.5)");
ResultSet rs = statement.executeQuery("select * from person");
while (rs.next()) {
  // 如果数据类型不匹配, 能转换就转换, 不能转换会返回 0
  System.out.print("name = " + rs.getString("name") + ", ");
  System.out.print("id = " + rs.getInt("id") + ", ");
  System.out.print("id = " + rs.getInt(1) + ", "); // 使用列数, 从1开始
  System.out.println("age = " + rs.getDouble("age"));
} // catch, finally
connection.close(); // 也要 try
```