414-18au

1a

CREATE TABLE Movie(

mid INT PRIMARY KEY,

name VARCHAR(50),

year INT,

revenue FLOAT

);

CREATE TABLE Casts(

pid INT REFERENCE Actor(pid),

mid INT REFERENCE Movie(mid),

role VARCHAR(50)

);

1b

SELECT A.fname, A.lname

FROM Actor A, Movie M, Casts C

WHERE A.pid = C.pid and M.mid = C.mid and M.name = ‘The Third Man’;

1c

SELECT D.fname, D.lname, count(\*) AS CT

FROM Directors D, Movie\_directors S

WHERE D.did and S.did

GROUP BY D.did, D.fname, D.lname

HAVING count(\*) >= 200

ORDER BY CT DESC;

1d

SELECT M.year, count(\*) AS CT

FROM Movie M

WHERE NOT EXISTS (

SELECT \*

FROM Casts C, Actor A

WHERE A.pid = C.pid and C.mid = M.mid and A.gender != ‘F’

)

GROUP BY M.year;

1e

SELECT M1.name , M1.year

FROM Movie M1

WHERE M1.revenue > ALL(SELECT M2.revenue

FROM Movie M2

WHERE M2.year = 2000)

;

2a

SELECT DISTINCT C.role

FROM Casts C, Actor A

WHERE C.pid = A.pid and A.fname = ‘Kevin’ and A.lname = ‘Bacon’;

2b

SELECT A.fname, A.lname

FROM Actor A, Casts C1, Casts C2, Movie M1, Movie M2

WHERE A.pid = C1.pid and M1.mid = C1.mid and M1.name = ‘In the Mood for Love’

and A.pid = C2.pid and M2.mid = C2.mid and M2.name = ‘Chungking Express’;

2c

SELECT M.year count(\*) AS C

FROM Movie M

GROUP BY M.year

ORDER BY C DESC;

2d

SELECT D1.did

FROM Directors D1

WHERE NOT EXISTS (

SELECT \*

FROM Directors D2, Movie\_Directors MD1, Movie\_Directors MD2

WHERE D1.did = MD1.did and MD1.mid = MD2.mid and MD2.did = D2.did and D1.did != D2.did);

1b

**SELECT** P.pid, P.PName, count(\*) AS C

**FROM** Patient P LEFT OUTER JOIN Visit V

**WHERE ON** P.pid = V.pid

**and** V.date <= ‘2021-01-01’

**GROUP BY** P.pid , P.PName;

1c

SELECT DISTINCT P.pid, P.PName, P.Dob

FROM Patient P, Visit V, Doctor D

Visit V2, Doctor D2

WHERE P.pid = V.pid and V.did = D.did

And P.pid = V2.pid and V2.did = D2.did

And D.Speciality = ‘surgeon’ and D2.speciality = ‘ophthalmologist’

GROUP BY P.pid, P.PName, P.Dob;

1d

SELECT D.Speciality, V2.did, V2. DName, D2.Date

FROM Visit V, Doctor D, Visit V2, Doctor D2

WHERE V.did = D.did and D.Speciality = D2.Speciality and V2.did = D2.did

GROUP BY D.Speciality, V2.did, V2. DName, D2.Date

HAVING min(V.date) = V2.date;

WITH Time AS (

SELECT D.Speciality min(V.date) AS md

FROM Visit V, Doctor D

WHERE V.did = D.did

GROUP BY D.Speciality

)

SELECT T.Speciality, D2.did, D2.Name, V2.date

FROM Time T, Visit V2, Doctor D2

WHERE V2.did = D2.did and T.Speciality = D2.Speciality and T.md = V2.date

**SQL**

**SELECT** \* **FROM** xxx;

**SELECT** P.Name, P.ID

**FROM** Payroll AS P

**WHERE** P. Job = ‘TA’;

**SELECT** \*

**FROM** Payroll

**ORDER BY** Name **DESC**;

(**ORDER BY** Name, Job;) 先按Name排，再对同个Name按Job排

**SELECT** Job **FROM** Payroll;

**SELECT DISTINCT** Job **FROM** Payroll;

**CREATE TABLE** Payroll (

UserID INT,

Name TEXT,

Job TEXT,

Salary INT

);

Strings: CHAR(20), VARCHAR(255), TEXT, ...

Numbers: INT, SMALLINT, FLOAT, ...

MONEY, DATETIME...

**INSERT INTO** Payroll **VALUES** (123, ‘Jack’, ‘TA’, 50000);

A **Key** is one or more attributes that **uniquely** identify a row.

**CREATE TABLE** Payroll (

UserID INT,

Name TEXT,

Job TEXT,

Salary INT,

**PRIMARY KEY** (UserID)

);

**CREATE TABLE** Payroll (

Name TEXT,

Job TEXT,

Salary INT,

**PRIMARY KEY** (Name, Job)

); Keys of more than one attribute

**CREATE TABLE** Payroll (

UserID INT **PRIMARY KEY**,

Name TEXT,

Job TEXT,

Salary INT

);

**CREATE TABLE** Regist (

UserID INT **REFERENCES** Payroll(UserID),

Car TEXT

);

**CREATE TABLE** Regist (

Name TEXT,

Job TEXT,

Car TEXT,

**FOREIGN KEY** (Name, Job) **REFERENCES** Payroll

); Foreign Keys of more than one attribute

**SELECT** P.Name, R.Car

**FROM** Payroll AS P, Regist AS R

**WHERE** P.UserID = R.UserID

(**and** P. Job = ‘TA’) ;

**=**

**SELECT** P.Name, R.Car

**FROM** Payroll AS P **JOIN** Regist AS R

**ON** P.UserID = R.UserID

(**and/WHERE** P. Job = ‘TA’) ;

INSERT INTO Payroll VALUES(123,'Jack','TA',50000);

INSERT INTO Payroll VALUES(345,'Allison','TA',60000);

INSERT INTO Payroll VALUES(567,'Magda','Prof',90000);

INSERT INTO Payroll VALUES(789,'Dan','Prof',100000);

INSERT INTO Regist VALUES(123,'Charger');

INSERT INTO Regist VALUES(567,'Civic');

INSERT INTO Regist VALUES(567,'Pinto');

**SELECT** P.Name, R.Car

**FROM** Payroll AS P, Regist AS R （找人没有的车, Jack Civic, Jack Pinto...）

(VS **FROM** Regist AS R, Payroll AS P 找车没有的人, Allision Charger, Magda Charger...)

**WHERE** P.UserID != R.UserID;

**Self Joins** a relation occurs twice in the FROM clause

**SELECT** P.Name, R1.Car

**FROM** Payroll AS P, Regist AS R1, Regist AS R2

**WHERE** P.UserID = R1.UserID **AND**

P.UserID = R2.UserID **AND**

R1.Car = ‘Civic’ **AND**

R2.Car = ‘Pinto’;

**SELECT** P.Name, R.Car

**FROM** Payroll AS P **LEFT OUTER JOIN** Regist AS R 加上左边表里没有Join的元素

**ON** P.UserID = R.UserID;

用**WHERE**会先执行WHERE前面的所有，最后执行WHERE

**RIGHT OUTER JOIN**加上右边表里没有Join的元素

**FULL OUTER JOIN**加上左边和右边表里没有Join的元素 很少用

**SELECT** Name, Salary\*1.1 AS NewSalary

**FROM** Payroll;

**isNULL**

false = 0; unknown = 0.5; true = 1

x AND y = min(x,y);

x OR y = max(x,y);

not x = 1-x;

**SELECT** count(DISTINCT Job)

**FROM** Payroll;

**SELECT** avg(P.Salary)

**FROM** Payroll AS P, Register AS R

**WHERE** P.UserID = R.UserID

**AND** R.Car = ‘Pinto’;

**SELECT** count(DISTINCE P.Name)

**FROM** Payroll AS P, Regist AS R

**WHERE** P.UserID = R.UserID

**SELECT** Product, Month, sum(Price\*Quant) as Rev

**FROM** Sales

**GROUP BY Product, Month**;

Rule: every attribute in SELECT must also occur in GROUP BY.

**SELECT** P.Name, count(\*)

**FROM** Payroll P, Regist R

**WHERE** P.UserID = R.UserID

**GROUP BY P.Name,** P.UserID; P.Name must occur in GROUP BY

**SELECT** P.Name, count(R.Car)

**FROM** Payroll P **LEFT OUTER JOIN** Regist R **ON** P.UserID = R.UserID

**GROUP BY** P.Name, P.UserID;

**SELECT** P1.Job, count(DISTINCT P2.UserID)

**FROM** Payroll P1 **LEFT OUTER JOIN** Payroll P2 **ON** P1.Job = P2.Job

**and** P2.Salary > 75000

**GROUP BY** P1.Job;

**WITH** Cardrivers AS

(**SELECT** DISTINCT P.UserID, P.Salary

**FROM** Payroll P, Regist R

**WHERE** P.UserId = R.UserID)

**SELECT** avg(Salary)

**FROM** Cardrivers;

**CREATE VIEW** CarDrivers AS

**SELECT** DISTINCT P.\*

**FROM** Payroll P, Regist R

**WHERE** P.UserId=R.UserID;

**Virtual View**: means computed at query time.

找到每个工作工资最高的人（可能不止一个）

1. 找到那个最高的工资
2. 输出工资等于最高工资的人

**WITH** JobSal AS

(**SELECT** Job, max(Salary) AS M

**FROM** Payroll

**GROUP BY** Job)

**SELECT** J.Job, P.Name, P.Salary

**FROM** JobSal J, Payroll P

**WHERE** J.Job = P.Job

**and** J.M = P.Salary;

**=**

**SELECT** P1.Job, P2.Name, P2.Salary

**FROM** Payroll AS P1, Payroll AS P2

**WHERE** P1.Job = P2.Job

**GROUP BY** P1.Job, P2.Name, P2.Salary

**HAVING** MAX(P1.Salary) = P2.Salary;

****from>where>group（含聚合）>having>order>select****

**WITH** Cardrivers AS

(**SELECT** DISTINCT P.\*

**FROM** Payroll P, Regist R

**WHERE** P.UserId=R.UserID)

**SELECT** avg(Salary)

**FROM** Cardrivers;

**=**

**SELECT** avg(C.Salary)

**FROM** ( **SELECT** DISTINCT P.\*

**FROM** Payroll P, Regist R

**WHERE** P.UserId=R.UserID) AS C;

**SELECT** P.Name, (**SELECT** AVG(P1.Salary)

**FROM** Payroll AS P1

**WHERE** P.Job = P1.Job)

**FROM** Payroll AS P;

**=**

**SELECT** P1.Name, AVG(P2.Salary)

**FROM** Payroll AS P1, Payroll AS P2

**WHERE** P1.Job = P2.Job

**GROUP BY** P1.UserID, P1.Name;

平均数有零的情况，所以就直接记这种比较好

**SELECT** P.Name, (**SELECT** COUNT(R.Car)

**FROM** Regist AS R

**WHERE** P.UserID = R.UserID)

**FROM** Payroll AS P;

**=**

**SELECT** P.Name, COUNT(R.Car)

**FROM** Payroll AS P **LEFT OUTER JOIN** Regist AS R **ON** P.UserID = R.UserID

**GROUP BY** P.UserID, P.Name;

**SELECT** P.Name, P.salary

**FROM** Payroll P

**WHERE** P.Salary < (**SELECT** avg(P1.salary)

**FROM** Payroll P1

**WHERE** P.Job = P1.Job);

**=**

**SELECT** P.Name, P.salary

**FROM** Payroll P, Payroll P1

**WHERE** P.Job = P1.Job

**GROUP BY** P.Name, P.salary

**HAVING** P.Salary < avg(P1.salary);

**EXISTS (SELECT ….)** get all which exists in

**NOT EXISTS (SELECT …)** get all which does not exist in

**X in (SELECT Y FROM …)** checks output has X

**X not in (SELECT Y …)** checks if it doesn’t have X

**X > ALL(SELECT …)**

**X > ANY(SELECT …)** checks if X is > than one or all values in output

**SELECT** P.UserID, P.Name

**FROM** Payroll P

**WHERE (NOT) EXISTS** ( **SELECT** \*

**FROM** Regist R

**WHERE** P.UserID = R.UserID);

**=**

**SELECT** P.UserID, P.Name

**FROM** Payroll P

**WHERE** P.UserID (not) in (**SELECT** R.UserID

**FROM** Regist R);

**Monotone** 单调——增加input不会减少output

**Monotone 的subquery才能unnest**

SELECT DISTINCT P.\*

FROM Payroll P, Register R

WHERE P.UserID = R.UserID and R.Year > 2017

**=**

**SELECT** P.\*

**FROM** Payroll P

**WHERE** 2017 <

**ANY**( **SELECT** R.Year

**ALL FROM** Regist R

**WHERE** P.UserID = R.UserID);

**1.Selection σcondition (S)**

**SELECT** \*

**FROM** T

**WHERE** condition;



**2.Projection Πattrs (S)**

**SELECT** attrs

**FROM** T;

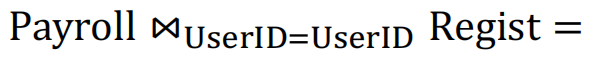


**3.Join R ⋈θ S = σθ (R×S)**

**SELECT** \*

**FROM** S,T

**WHERE** θ;



Eq-join: Payroll **⋈**UserID = UserID Regist

Theta-join: Payroll **⋈**UserID < UserID Regist

Cartesian product: Payroll×Regist 笛卡尔积

Natural Join: Payroll ⋈Regist

在公共属性上Join两个表，没有公共属性就是笛卡尔积，全是公共属性就留重复的

**4.Union ∪**

S **UNION** T; 表头相同，合并两个表

**5. Set difference −**

S **EXCEPT** T; 表头相同，从S里减去ST都有的部分

**6.Rename** *ρ*

*ρattrs’* (T)

**SELECT** a1 as a1’,

a2 as a2’,

...

**FROM** T;

表头本来是UserID和Car，*ρ*UserID,Model(Regist)以后表头变成UserID和Model

**Duplicate elimination *δ*(T)**

**SELECT** DISTINCT \*

**FROM** T;

**Group-by aggregate γattr1,attr2,...,agg1,...**

**SELECT** attr1,...,agg1,...

**FROM** T

**GROUP BY** attr1,...;