CS61A NOTE15 SQL

SQL Basics

SQL is an example of a declarative programming language. Statements do not describe computations directly, but instead describe the desired result of some computation. It is the role of the query interpreter of the database system to plan and perform a computational process to produce such a result.

Creating Tables

You can create SQL tables either from scratch 从头 or from existing tables 现有.

The following statement creates a table by specifying column names and values without referencing another table. Each `SELECT` clause specifies the values for one row, and `UNION` is used to join rows together. The `AS` clauses give a name to each column; it need not be repeated in subsequent rows after the first.

berkeley	stanford	year
30	7	2002
28	16	2003
17	38	2014

纯文本

```
CREATE TABLE big_game AS
  SELECT 30 AS berkeley, 7 AS stanford, 2002 AS year UNION
  SELECT 28,             16,            2003         UNION
  SELECT 17,             38,            2014;
```

SQL operators

Expressions in the `SELECT`, `WHERE`, and `ORDER BY` clauses can contain one or more of the following operators:

- comparison operators: `=`, `>`, `<`, `<=`, `>=`, `<>` or `!=` ("not equal")
- boolean operators: `AND`, `OR`
- arithmetic operators: `+`, `-`, `*`, `/`
- concatenation operator: `||`

```
纯文本
Output the ratio of Berkeley's score to Stanford's score each year:
select berkeley * 1.0 / stanford from big_game;

Output the sum of scores in years where both teams scored over 10 points:
select berkeley + stanford from big_game where berkeley > 10 and stanford >

Output a table with a single column and single row containing the value "he
SELECT "hello" || " " || "world";
```

Selecting From Tables

More commonly, we will create new tables by selecting specific columns that we want from existing tables by using a `SELECT` statement as follows:

```
纯文本
SELECT [columns] FROM [tables] WHERE [condition] ORDER BY [columns] LIMIT [

SELECT name,salary2023-salary2022 FROM salaries ORDER BY salary2023-salary2
```

records

name	division	title	salary	supervisor
Ben Bitdiddle	Computer	Wizard	60000	Oliver Warbucks
Alyssa P Hacker	Computer	Programmer	40000	Ben Bitdiddle
Cy D Fect	Computer	Programmer	35000	Ben Bitdiddle
Lem E Tweakit	Computer	Technician	25000	Ben Bitdiddle
Louis Reasoner	Computer	Programmer Trainee	30000	Alyssa P Hacker
Oliver Warbucks	Administration	Big Wheel	150000	Oliver Warbucks
Eben Scrooge	Accounting	Chief Accountant	75000	Oliver Warbucks
Lana Lambda	Administration	Executive Director	610000	Lana Lambda

纯文本

```
> SELECT "Ben" AS first, "Bitdiddle" AS last ; #select不用写出新表
Ben|Bitdiddle

> SELECT "Ben" AS first, "Bitdiddle" AS last UNION ;   #合并到一张表
> SELECT "Louis", "Reasoner"
Ben|Bitdiddle
Louis|Reasoner
```

纯文本

```
#SELECT specific values from an existing table using a FROM clause

#SELECT ..., ,,, FROM records
> SELECT name, division FROM records ;
Alyssa P Hacker|Computer
...
Robert Cratchet|Accounting

#SELECT * FROM 从表中选择所有列
> SELECT * FROM records;
Alyssa P Hacker|Computer|Programmer|40000|Ben Bitdiddle
...
Robert Cratchet|Accounting|Scrivener|18000|Eben Scrooge

#SELECT, filter out rows using a WHERE clause, sort the resulting rows with

#SELECT [columns] FROM [tables] WHERE [condition] ORDER BY [criteri]
```

```
#升asc 降desc
> SELECT * FROM records WHERE title = "Programmer";
Alyssa P Hacker|Computer|Programmer|40000|Ben Bitdiddle
Cy D Fect|Computer|Programmer|35000|Ben Bitdiddle

> SELECT name, salary FROM records WHERE division = "Accounting" ORDER BY s
Eben Scrooge|75000
Robert Cratchet|18000
```

disc Q3: Oliver Employees

Write a query that outputs the names of employees that Oliver Warbucks directly supervises 监督

```
SELECT name FROM records WHERE supervisor='Oliver Warbucks';#别漏分号
```

discQ4

Write a query that outputs all information about employees that supervise themselves.

```
SELECT * FROM records WHERE name=supervisor; #SELECT *对应全部信息
```

discQ5: Rich Employees

Write a query that outputs the names of all employees with salary greater than 50,000 in alphabetical order.

```
SELECT name FROM records WHERE salary > 50000 ORDER BY name;
```

Q6: Raises

salaries

name	salary2022	salary2023
Ben Bitdiddle	60000	80000
Alyssa P Hacker	40000	80000
Cy D Fect	35000	74000
Lem E Tweakit	25000	28000
Louis Reasoner	30000	30000
Oliver Warbucks	150000	120000
Eben Scrooge	75000	76000
Robert Cratchet	18000	20000
Lana Lambda	610000	610000

Write a query that outputs the names of the top 3 employees with the largest salary raises from 2022 to 2023 along with their corresponding salary raises, ordered from largest to smallest raise.(增长,desc)

```
纯文本

SELECT name,salary2023-salary2022 FROM salaries ORDER BY salary2023-salary2
#LIMIT 3的用法—取前三个，ORDER BY（可为运算）ASC/DESC，记得加；
```

Joins 合并

To select data from multiple tables, we can use joins. There are many types of joins, but the only one we'll worry about is the inner join. To perform an inner join on two on more tables, simply list them all out in the `FROM` clause of a `SELECT` statement:

```
SELECT [columns] FROM [table1], [table2], ... WHERE [condition] ORDER BY [c
```

When we join two or more tables, the default output is a cartesian product. For example, if we joined `big_game` with `coaches`, we'd get the following:

合并默认笛卡尔积

berkeley	stanford	year
30	7	2002
28	16	2003
17	38	2014

name	start	end
Jeff Tedford	2002	2012
Sonny Dykes	2013	2016
Justin Wilcox	2017	null

berkeley	stanford	year	name	start	end
30	7	2002	Jeff Tedford	2002	2012
30	7	2002	Sonny Dykes	2013	2016
30	7	2002	Justin Wilcox	2017	null
28	16	2003	Jeff Tedford	2002	2012
28	16	2003	Sonny Dykes	2013	2016
28	16	2003	Justin Wilcox	2017	null
17	38	2014	Jeff Tedford	2002	2012
17	38	2014	Sonny Dykes	2013	2016
17	38	2014	Justin Wilcox	2017	null

纯文本

```
SELECT * FROM big_game, coaches WHERE year >= start AND year <= end;
17|38|2014|Sonny Dykes|2013|2016
28|16|2003|Jeff Tedford|2002|2012
30|7|2002|Jeff Tedford|2002|2012

SELECT name, year FROM big_game, coaches
...>        WHERE berkeley > stanford AND year >= start AND year <= end;
Jeff Tedford|2003
Jeff Tedford|2002
```

In the queries above, none of the column names are ambiguous. For example, it is clear that the `name` column comes from the `coaches` table because there isn't a column in the `big_game` table with that name. However, if a column name exists in more than one of the tables being joined, or if we join a table with itself, we must disambiguate the column names using *aliases* 别名.

For examples, let's find out what the score difference is for each team between a game in `big_game` and any previous games. Since each row in this table represents one game, in order to compare two games we must join `big_game` with itself:有一个自己套用自己的过程

```
纯文本
SELECT b.Berkeley - a.Berkeley, b.Stanford - a.Stanford, a.Year, b.Year FRO
-11|22|2003|2014
-13|21|2002|2014
-2|9|2002|2003
```

SQL Aggregation

Previously, we have been dealing with queries that process one row at a time 一整行查询. When we join, we make pairwise combinations of all of the rows 组合行. When we use `WHERE`, we filter out certain rows based on the condition 选择一些行. Alternatively, applying an aggregate function such as `MAX(column)` combines the values in multiple rows.

By default 默认, we combine the values of the *entire* table. For example, if we wanted to count the number of flights from our `flights` table, we could use:

What if we wanted to group together the values in similar rows and perform the aggregation operations within those groups? We use a `GROUP BY` clause.

Here's another example. For each unique departure, collect all the rows having the same departure airport into a group. Then, select the `price` column and apply the `MIN` aggregation to recover the price of the cheapest departure from that group. The end result is a table of departure airports and the cheapest departing flight.

如果我们想把相似行中的数值分组，并在这些组中进行聚合操作，会怎么样呢？我们使用 GROUP BY 子句。

Just like how we can filter out rows with `WHERE`, we can also filter out groups with `HAVING`. Typically, a `HAVING` clause should use an aggregation function. Suppose we want to see all airports with at least two departures:就像我们可以用 WHERE 过滤掉行一样，我们也可以用 HAVING 过滤掉组。通常，HAVING 子句应该使用一个聚合函数。假设我们想查看所有至少有两个航班起飞的机场：

Note that the `COUNT(*)` aggregate just counts the number of rows in each group. Say we want to count the number of *distinct* airports instead. Then, we could use the following query:注意，COUNT(*)聚合只是计算每组中的行数。假设我们想计算独立机场的数量。那么，我们可以使用下面的查询：

```
纯文本
SELECT COUNT(*) from FLIGHTS;
13

SELECT departure, MIN(price) FROM flights GROUP BY departure;
group by在此列对一些行操作
AUH|932
LAS|50
LAX|89
SEA|32
SFO|40
SLC|42

SELECT departure FROM flights GROUP BY departure HAVING COUNT(*) >= 2;
having类似where但不是列而是行操作了
LAX
SFO
SLC

SELECT COUNT(DISTINCT departure) FROM flights;
计算不同值用count和distinct
6
```

Summary

```
SELECT [columns]
    FROM [tables]
    WHERE [condition]
    GROUP BY [column]
    HAVING [condition]
    ORDER BY [columns] [ASC/DESC]
    LIMIT [limit];
```

Final - Summer 2022 Q8 (c)

```
studios: name, studio
movies: title, budget, boxoffice
```

(c) (6.0 points)

Using the **movies** and **studios** tables from the last 2 problems, write a query that find the top 2 studios with regards to average box office.

The table should be sorted from highest average to lowest. You should only include studios that have 2 or more movies made. The output table should have 2 columns, one for the studio and one for the average box office.

Your solution should work on any tables with the same columns.

```
SELECT _____________________________________
    FROM movies, studios WHERE _____________________
    GROUP BY _______________ HAVING _____________________
    ORDER BY ____________________ DESC __________;
```

Final - Summer 2022 Q8 (c)

(c) (6.0 points)

Using the **movies** and **studios** tables from the last 2 problems, write a query that find the top 2 studios with regards to average box office.

The table should be sorted from highest average to lowest. You should only include studios that have 2 or more movies made. The output table should have 2 columns, one for the studio and one for the average box office.

Your solution should work on any tables with the same columns.

```
SELECT studio, AVG(boxoffice)
    FROM movies, studios WHERE name = title
    GROUP BY studio HAVING COUNT(*) >= 2
    ORDER BY AVG(boxoffice) DESC LIMIT 2;
```

Final - Spring 2018 Q7 (b)

7. (10 points) Gotta Select 'Em All

For the questions below, assume that the following two SQL statements have been executed. The `pokedex` table describes the names of some Pokémon and their heights in inches. The `evolve` table describes how those Pokémon can evolve into the other Pokémon in the `pokedex`.

```
CREATE TABLE pokedex AS
  SELECT "Eevee" AS name, 12 as height UNION
  SELECT "Jolteon"      , 31           UNION
  SELECT "Leafeon"      , 39           UNION
  SELECT "Bulbasaur"    , 28           UNION
  SELECT "Ivysaur"      , 39           UNION
  SELECT "Venasaur"     , 79           UNION
  SELECT "Charmander"   , 24           UNION
  SELECT "Charmeleon"   , 43           UNION
  SELECT "Charizard"    , 67;
```

```
CREATE TABLE evolve AS
  SELECT "Eevee" AS before, "Jolteon" AS after UNION
  SELECT "Eevee"          , "Leafeon"          UNION
  SELECT "Bulbasaur"      , "Ivysaur"          UNION
  SELECT "Ivysaur"        , "Venasaur"         UNION
  SELECT "Charmander"     , "Charmeleon"       UNION
  SELECT "Charmeleon"     , "Charizard";
```

Final - Spring 2018 Q7 (b)

(b) (6 pt) Write a SELECT statement that results in a table with one row for each Pokémon that can evolve. The table should have two columns: the first contains the name of the Pokémon that can evolve, and the second contains the maximum increase in height that it can attained by evolving. For example, Eevee can grow as much as 27 inches (when evolving to Leafeon), so the result should contain the row `("Eevee", 27)`. Your statement should behave correctly even if the rows in `evolve` and `pokedex` were different. The result should only consider ways of evolving that are described by a single row in the `evolve` table.

```
SELECT before, max(b.height - a.height)


FROM evolve, pokedex as a, pokedex as b


WHERE before=a.name AND after=b.name


GROUP BY before;
```

Final - Summer 2017 Q10 - (a):

10. (18 points) I've brought ice and fire together.

Consider the following schema that represents `users`, `products`, and `sales` in a database management system.

```
create table users(uid, uname, date_created);
create table products(pid, pname, description, rating, price);
create table sales(time, pid, uid);
```

- `uid` (user ID), `pid` (product ID), `rating`, `price` are numbers while all other columns are strings.
- The `uid` uniquely identifies one user because there may be users with the same `uname` and `date_created`.
- The `pid` uniquely identifies one product because there may be products with the same column values.
- The `uid` and `pid` in each row of `sales` references a `uid` in `users` and a `pid` in `products`.

Express the following queries in SQL using only features we've covered in this course.

Recall: Rows can be ordered in either `asc` ending (increasing) or `desc` ending (decreasing) order.

(a) (2 pt) Select the `uname` and product `rating` of any one user who purchased a highest-rated product (a product such that there is no other product rated higher). If there is more than one such product, return any one product.

```
select u.uname, p.rating

    from users as u, products as p, sales as s

    where u.uid = s.uid and s.pid = p.pid

    order by p.rating desc limit 1;
```

SQL!!!!

table name:
movies

2014 Movies

movie budget bo sf

2014 Movies	Budget (millions)	Box Office (millions)	SciFi? (IMDb)
Kingsman	81	406	False
Unfriended	1	54	False
Interstellar	165	672	True

Question 1

Write a query that outputs the budget that appear more than once in the table of movies.

Question 2

Write a query that will output a list of sci-fi movies ordered by the return of investment ratio [(box office - budget)/budget] from greatest to least.

Solution:

```
select budget from movies group by budget having count(*) >1;
```

Solution:

```
select movie from movies where sf = "true" order by -(bo-budget)/budget;
```