本文件中所有 SQL 代碼 和 textbook 中 SQL 重要的東西被抄在了"SQL-quick-pickup"中, 看那個文件即可. "SQL-quick-pickup"中也有解釋, 所以看"SQL-quick-pickup"時不需要看本文件和 textbook. 但若看一些比如 SQL 的歷史甚麼的以及更深入的東西, 還是要看本文件和 textbook, 尤其是 textbook.

SQL Queries

The above figure (only two words SQL Queries): Most databases support some form of SQL. Most data sit in relational databses and I access through SQL.

SQL - History

- SEQUEL: Structured English QUEry Language; part of SYSTEM R, 1973
  Chamberlin Boyce
- SQL: Structured Query Language
- SQL is based on relational tuple calculus and some algebra
- ANSI & ISO standards:
  - SQL/86, SQL/89
  - SQL/92 or SQL2
  - SQL3: 1999
  - Revisions in 2003, 2006, 2008, 2011 ...
- SQL is supported by IBM DB2, ORACLE, SYBASE, SQLServer, MySQL, and many more – sort of ...

Insert, Delete and Update...

```
INSERT INTO UserInterests (Email, Interest, SinceAge)
    VALUES ('user12@gt.edu', 'Reading', 5);
```

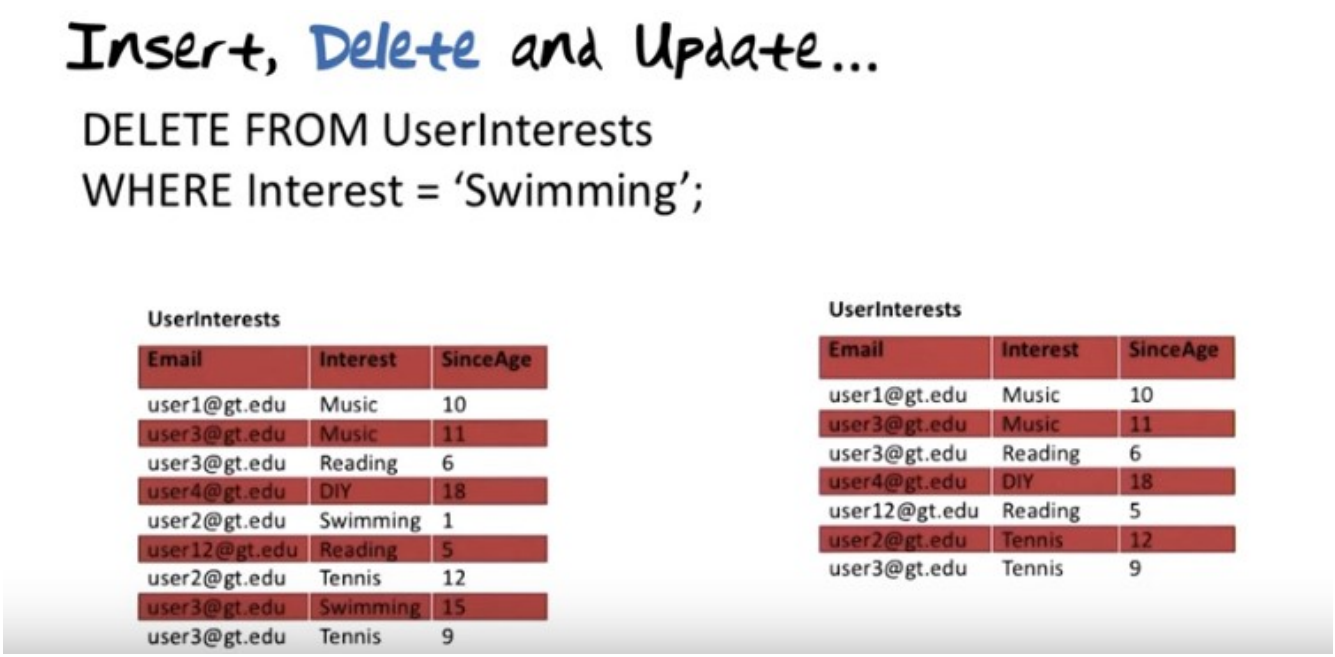
UserInterests

Email	Interest	SinceAge
user1@gt.edu	Music	10
user3@gt.edu	Music	11
user3@gt.edu	Reading	6
user4@gt.edu	DIY	18
user2@gt.edu	Swimming	1
user2@gt.edu	Tennis	12
user3@gt.edu	Swimming	15
user3@gt.edu	Tennis	9

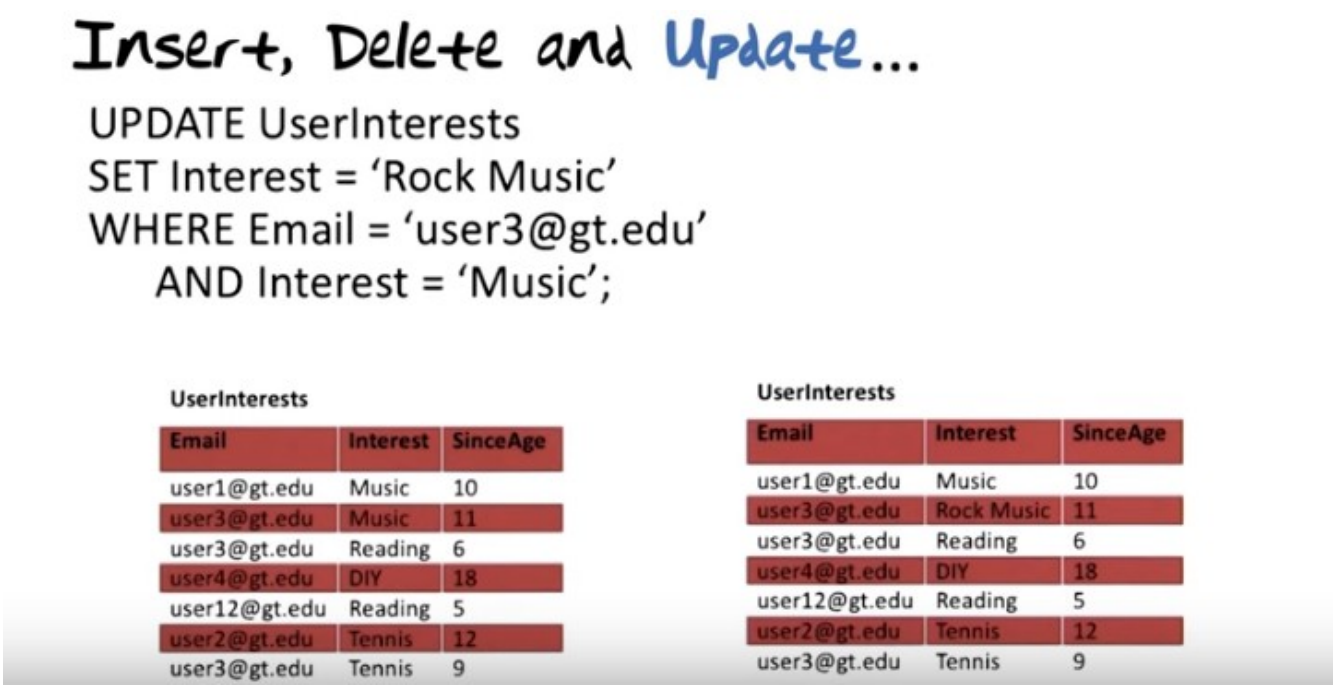
UserInterests

Email	Interest	SinceAge
user1@gt.edu	Music	10
user3@gt.edu	Music	11
user3@gt.edu	Reading	6
user4@gt.edu	DIY	18
user2@gt.edu	Swimming	1
user12@gt.edu	Reading	5
user2@gt.edu	Tennis	12
user3@gt.edu	Swimming	15
user3@gt.edu	Tennis	9

上圖: 一般說來, INSERT 一次只加入一行, 除非是從 database 本身中提取一些東西加入.



上圖: DELETE 一次可以刪除多行.



上圖: UPDATE 一次也可以處理多行.

Knowledge Check

What happens when you issue the following INSERT statement on the RegularUser table?

a) The table becomes inconsistent on BirthYear and Sex

b) The INSERT is rejected due to a primary key violation

c) The table will have six tuples

d) The BirthYear and Sex of RegularUser will be updated to the new values.

```
INSERT INTO RegularUser (Email, BirthYear, Sex)
    VALUES ('user4@gt.edu', 1986, 'F');
```

RegularUser

Email	BirthYear	Sex
user1@gt.edu	1985	M
user2@gt.edu	1969	M
user3@gt.edu	1967	M
user4@gt.edu	1988	M
user6@gt.edu	1988	F

General SQL Query Syntax

```
SELECT column_1, column_2, ..., column_n
FROM table_1, table_2, ..., table_m
WHERE condition;
```

- $column_i$ is the name of a **column**, like BirthYear
- $table_j$ is the name of a **table**, like RegularUser
- **condition** may compare values of columns to constants or to each other, like
  - BirthYear > 1985
  - CurrentCity = HomeTown
- conditions can be combined with **AND, OR, NOT** and **()**

上圖中的 Query 應該是只指 SELECT. In Commercial relational database, we often use the terms column instead of attribute, table instead of relation, and row instead of tuple.

Selection - and the * wildcard

```
SELECT Email, BirthYear, Sex, CurrentCity, HomeTown
FROM RegularUser;
```

Find all RegularUser's

```
SELECT *
FROM RegularUser;
```

RegularUser

Email	Birth Year	Sex	Current City	HomeTown
user1@gt.edu	1985	M	Seattle	Atlanta
user2@gt.edu	1969	M	Austin	Austin
user3@gt.edu	1967	M	San Diego	Portland
user4@gt.edu	1988	M	San Francisco	Atlanta
user6@gt.edu	1988	F	San Diego	Atlanta

RegularUser

Email	Birth Year	Sex	Current City	HomeTown
user1@gt.edu	1985	M	Seattle	Atlanta
user2@gt.edu	1969	M	Austin	Austin
user3@gt.edu	1967	M	San Diego	Portland
user4@gt.edu	1988	M	San Francisco	Atlanta
user6@gt.edu	1988	F	San Diego	Atlanta

上圖: 注意 SQL 語句要以分號結尾. SELECT *那句是跟上面 SELECT Email...等價的.

Selection – with a WHERE clause

```
SELECT *
FROM RegularUser
WHERE HomeTown = 'Atlanta';
```

Find all RegularUser's with HomeTown Atlanta

RegularUser

Email	Birth Year	Sex	Current City	HomeTown
user1@gt.edu	1985	M	Seattle	Atlanta
user4@gt.edu	1988	M	San Francisco	Atlanta
user6@gt.edu	1988	F	San Diego	Atlanta

RegularUser

Email	Birth Year	Sex	Current City	HomeTown
user1@gt.edu	1985	M	Seattle	Atlanta
user2@gt.edu	1969	M	Austin	Austin
user3@gt.edu	1967	M	San Diego	Portland
user4@gt.edu	1988	M	San Francisco	Atlanta
user6@gt.edu	1988	F	San Diego	Atlanta

Selection – with composite WHERE clause

```
SELECT *
FROM RegularUser
WHERE CurrentCity = HomeTown OR
    HomeTown = 'Atlanta';
```

Find all RegularUser's who have the same CurrentCity and HomeTown or who live in Atlanta

RegularUser

Email	Birth Year	Sex	Current City	HomeTown
user1@gt.edu	1985	M	Seattle	Atlanta
user2@gt.edu	1969	M	Austin	Austin
user3@gt.edu	1967	M	San Diego	Portland
user4@gt.edu	1988	M	San Francisco	Atlanta
user6@gt.edu	1988	F	San Diego	Atlanta

RegularUser

Email	Birth Year	Sex	Current City	HomeTown
user1@gt.edu	1985	M	Seattle	Atlanta
user2@gt.edu	1969	M	Austin	Austin
user4@gt.edu	1988	M	San Francisco	Atlanta
user6@gt.edu	1988	F	San Diego	Atlanta

Projection

Find Email, BirthYear, and Sex for RegularUser's in Atlanta

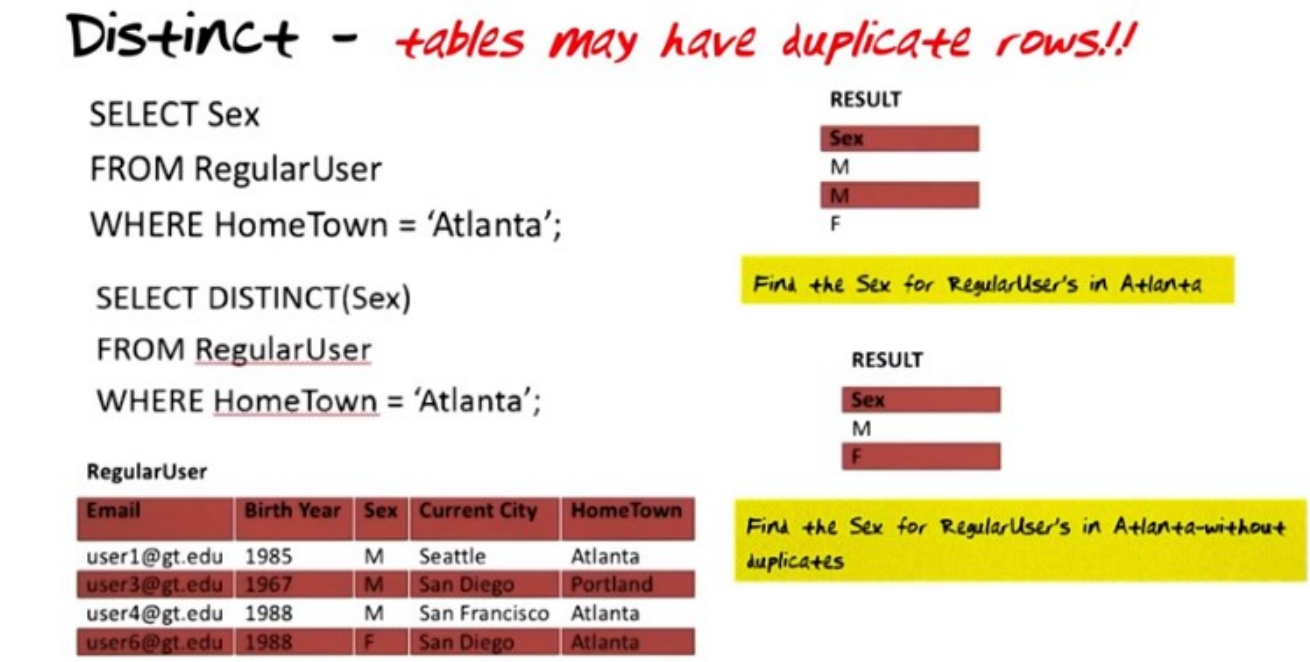
```
SELECT Email, BirthYear, Sex
FROM RegularUser
WHERE HomeTown = 'Atlanta';
```

RegularUser

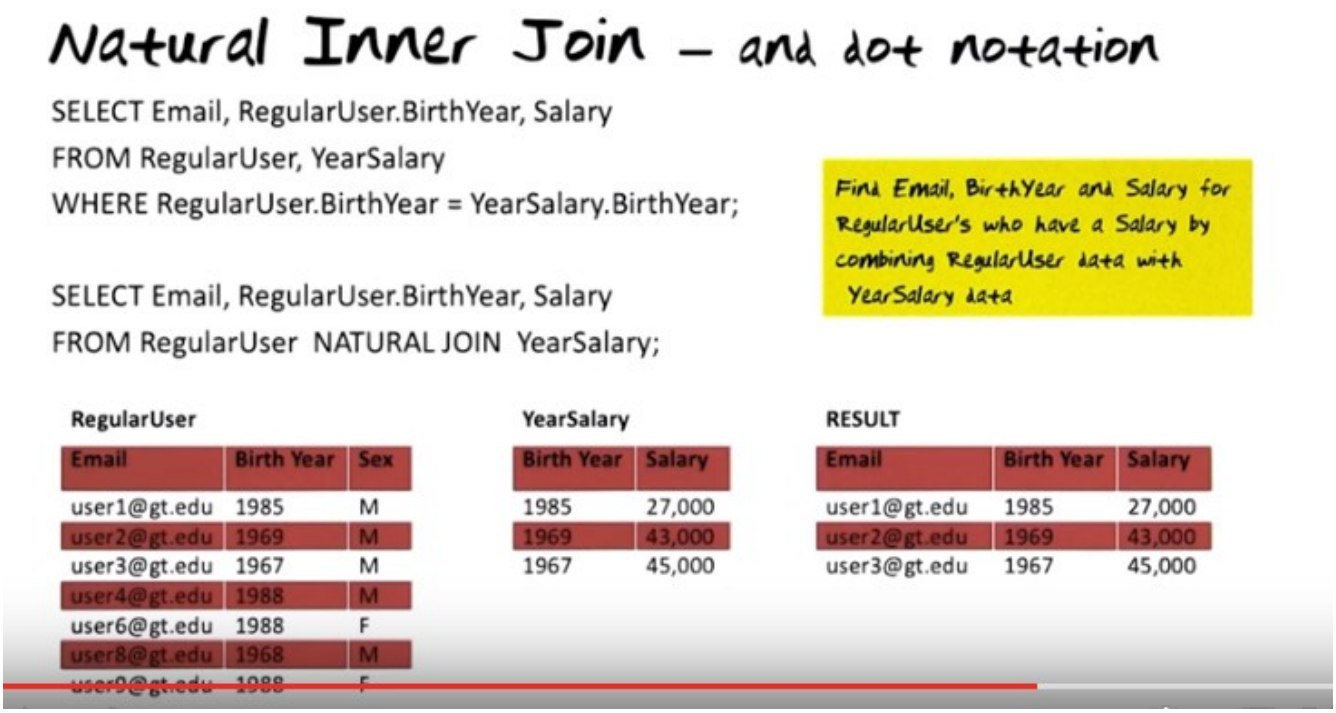
Email	Birth Year	Sex	Current City	HomeTown
user1@gt.edu	1985	M	Seattle	Atlanta
user2@gt.edu	1969	M	Austin	Austin
user3@gt.edu	1967	M	San Diego	Portland
user4@gt.edu	1988	M	San Francisco	Atlanta
user6@gt.edu	1988	F	San Diego	Atlanta

RegularUser

Email	Birth Year	Sex
user1@gt.edu	1985	M
user4@gt.edu	1988	M
user6@gt.edu	1988	F



The above figure: In SQL, tables may have duplicate rows.



The above figure: when no ambiguity is present like with Email and Salary, we don't have to use dot notation, we can but we don't have to. For Birth Year, it could be the birth year form RegularUser table of from the YearSalary table. So we have to tell the database system which table it's from.

We must include Birth Year (ie, in the result table) and we must say which table it is pulled from.

In the relational algebra, we didn't have to specify the join condition when the attribute or column names are the same. In SQL you have to.

有 NATURAL JOIN 的那段跟它上面的 SELECT Email...等價.

How many tuples are in the result of the SQL query?

- a) One
- b) Two
- c) Three
- d) Four

```
SELECT BirthYear
FROM RegularUser
WHERE Sex = 'F';
```

RegularUser

Email	BirthYear	Sex
user1@gt.edu	1985	M
user2@gt.edu	1969	M
user3@gt.edu	1967	F
user4@gt.edu	1988	F
user6@gt.edu	1988	F

Natural Inner Join – aliases

```
SELECT Email , R.BirthYear, Salary
FROM RegularUser AS R, YearSalary AS Y
WHERE R.BirthYear = Y.BirthYear;
```

Same thing, but with aliases

Find Email, BirthYear and Salary for RegularUser's who have a Salary by combining RegularUser data with YearSalary data

- Aliases (tuple variables!) save typing
- Aliases are used to disambiguate table references
- Aliases MUST be used when joining a table to itself

RegularUser

Email	Birth Year	Sex
user1@gt.edu	1985	M
user2@gt.edu	1969	M
user3@gt.edu	1967	M
user4@gt.edu	1988	M
user6@gt.edu	1988	F
user8@gt.edu	1968	M
user9@gt.edu	1988	F

YearSalary

Birth Year	Salary
1985	27,000
1969	43,000
1967	45,000

RESULT

Email	Birth Year	Salary
user1@gt.edu	1985	27,000
user2@gt.edu	1969	43,000
user3@gt.edu	1967	45,000

The above figure: I strongly recommend using aliases when you write queries.

Left Outer Join

Find Email, BirthYear and Salary for RegularUser's who have a Salary by combining RegularUser data with YearSalary data. Return Email and BirthYear even when the RegularUser has no Salary

```
SELECT Email , RegularUser.BirthYear, Salary
FROM RegularUser LEFT OUTER JOIN YearSalary;
```

RegularUser

Email	Birth Year	Sex
user1@gt.edu	1985	M
user2@gt.edu	1969	M
user3@gt.edu	1967	M
user4@gt.edu	1988	M
user6@gt.edu	1988	F
user8@gt.edu	1968	M
user9@gt.edu	1988	F

YearSalary

Birth Year	Salary
1985	27,000
1969	43,000
1967	45,000

RESULT

Email	Birth Year	Salary
user1@gt.edu	1985	27,000
user2@gt.edu	1969	43,000
user3@gt.edu	1967	45,000
user4@gt.edu	1988	NULL
user6@gt.edu	1988	NULL
user8@gt.edu	1968	NULL
user9@gt.edu	1988	NULL

String Matching

Find data about RegularUser's who live in a CurrentCity that starts with „San"

```
SELECT Email, Sex, CurrentCity
FROM RegularUser
WHERE CurrentCity LIKE 'San%';
```

RESULT

Email	Sex	CurrentCity
user3@gt.edu	M	San Diego
user4@gt.edu	M	San Francisco
user6@gt.edu	F	San Diego

```
SELECT Email, Sex, HomeTown
FROM RegularUser
WHERE HomeTown LIKE 'A_____';
```

RESULT

Email	Sex	HomeTown
user12@gt.edu	F	Austin

RegularUser

Email	Sex	CurrentCity	HomeTown
user1@gt.edu	M	Seattle	Atlanta
user3@gt.edu	M	San Diego	Portland
user4@gt.edu	M	San Francisco	Atlanta
user6@gt.edu	F	San Diego	Atlanta
user9@gt.edu	F	Las Vegas	Atlanta
user12@gt.edu	F	College Park	Austin

% matches any string, including the empty string

_ matches any single character

Sorting – oh no!

Find data about RegularUser's who are Males. Sort the data by ascending CurrentCity

```
SELECT Email, Sex, CurrentCity
FROM RegularUser
WHERE Sex='M'
ORDER BY CurrentCity ASC;
```

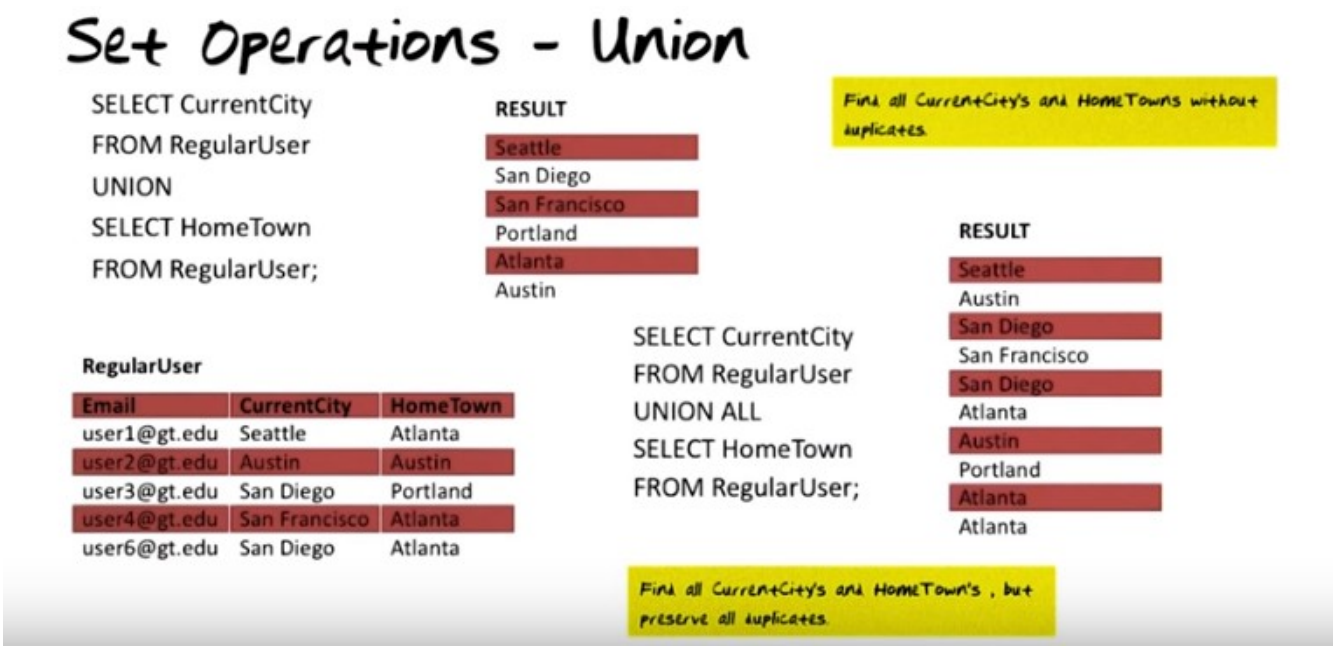
RESULT

Email	Sex	CurrentCity
user2@gt.edu	M	Austin
user3@gt.edu	M	San Diego
user4@gt.edu	M	San Francisco
user1@gt.edu	M	Seattle

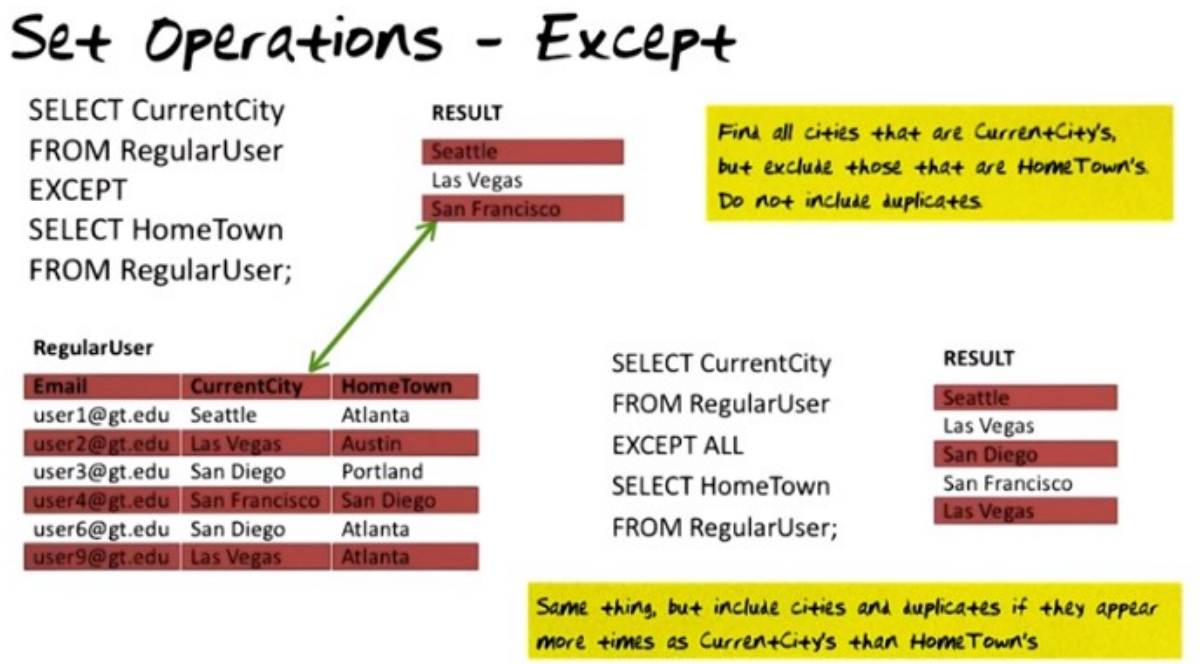
RegularUser

Email	Sex	CurrentCity	HomeTown
user1@gt.edu	M	Seattle	Atlanta
user2@gt.edu	M	Austin	Austin
user3@gt.edu	M	San Diego	Portland
user4@gt.edu	M	San Francisco	Atlanta
user6@gt.edu	F	San Diego	Atlanta

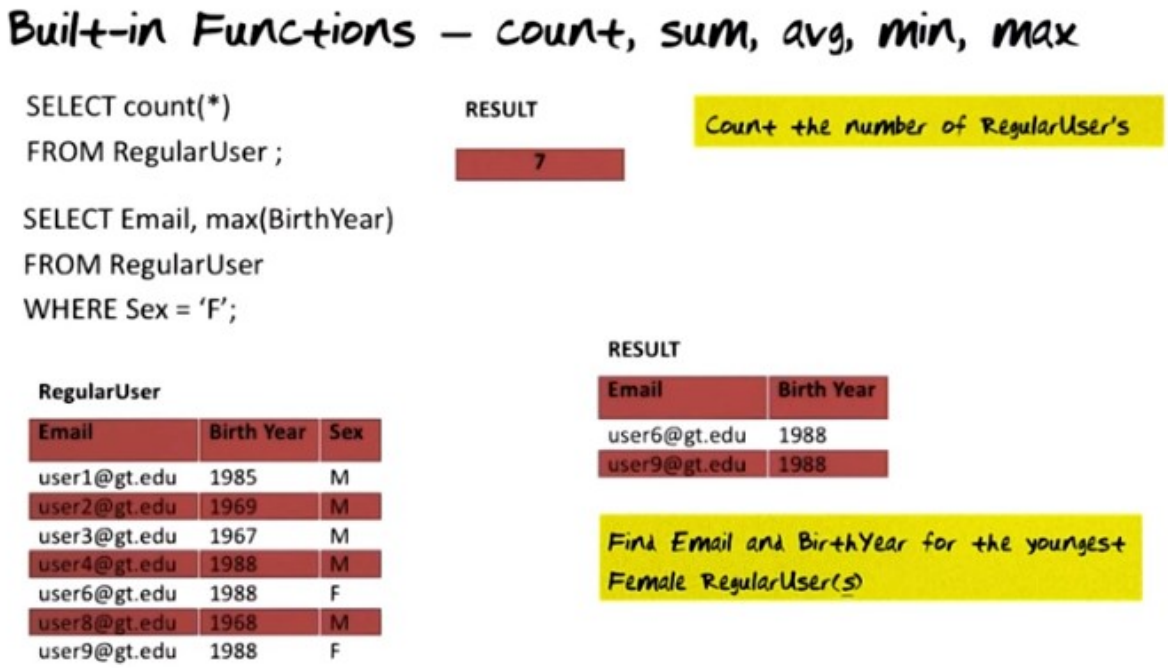
sort on multiple columns ascending (ASC) and/or descending (DESC)



The above figure: INTERSECT ALL 情况時, Austin is counted twice. Why is that? Well, as you see here, Austin is the current city for two people, and Austin is also the hometown for two people.

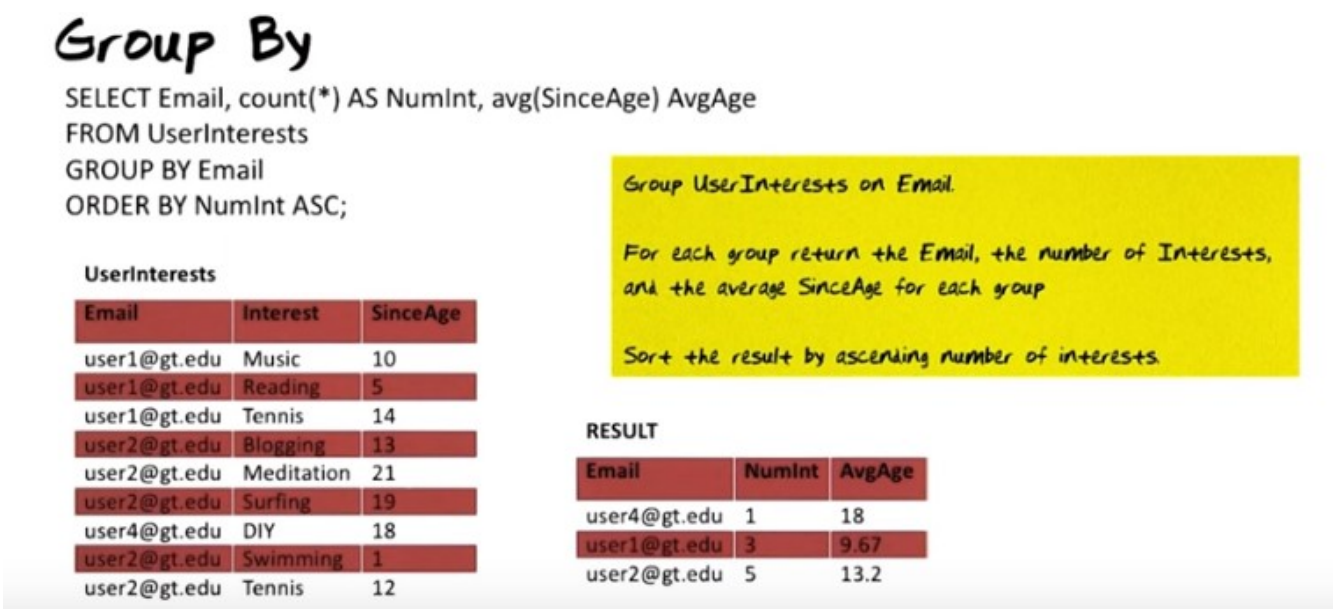


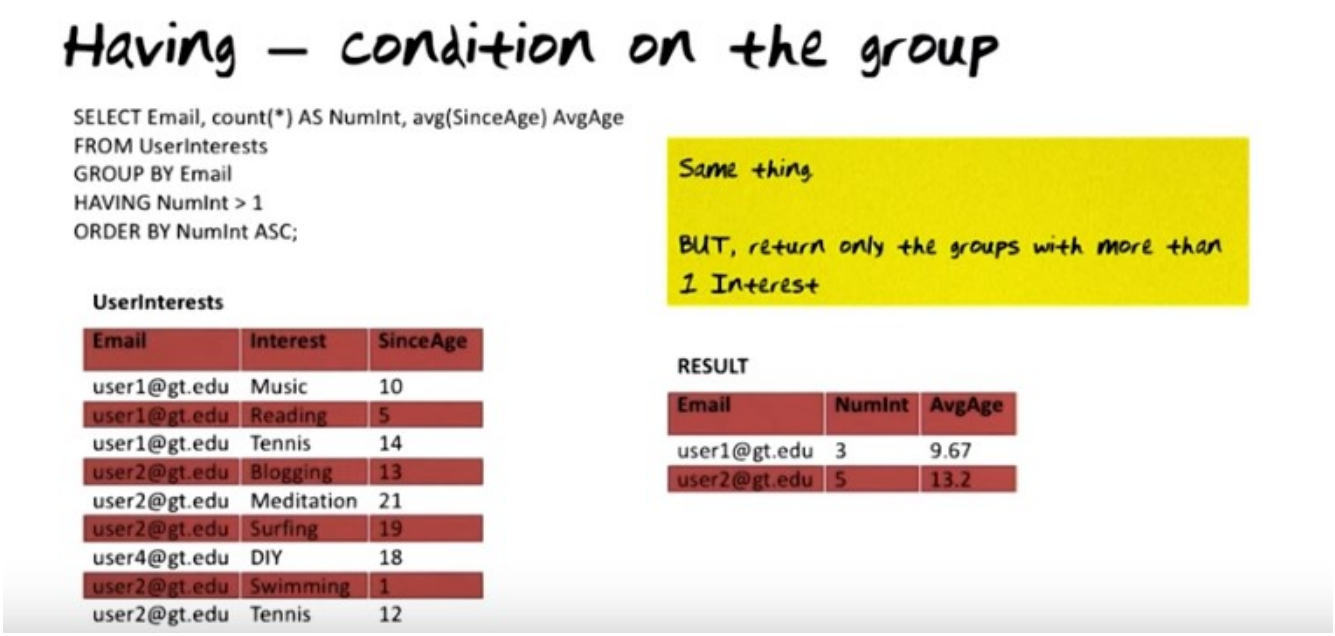
The above figure: EXCEPT ALL 情況時, Las Vegas 在 RESULT 中出現了兩次, 是因為 CurrentCity 中有兩個 Las Vegas, Hometown 中沒有 Las Vegas. San Diego 出現在了 RESULT 中, 是因為 RegularUser 中, CurrentCity 中有兩個 San Diego, Hometown 中有一個 San Diego. Hometown 中的 San Diego 抵消掉了 CurrentCity 中的一個 San Diego, CurrentCity 中還剩下一個 San Diego, 所以出現在了 RESULT 中. 2016.12: 最好的方法就是求 EXCEPT 前, 對 EXCEPT 兩邊的先去重復. 求 EXCEPT ALL 時, 將 EXCEPT ALL 兩邊的 SanDiego 都看成不同的(即兩個不同的城市偶然都叫 SanDiego 而已).



The above figure: count 時, the result here is a table with a single column and a single row and the

number 7 in it.





Nested Queries – IN/NOT IN

```
SELECT Email, Interest
FROM UserInterests
WHERE Email IN
      (SELECT Email
      FROM  RegularUser
      WHERE HomeTown = 'Atlanta';
```

OR

```
SELECT U.Email, Interest
FROM UserInterests I, RegularUser U
WHERE I.Email = U.Email AND
HomeTown='Atlanta';
```

Find Email and Interests for RegularUser's in Atlanta

RegularUser

Email	Birth Year	Sex	HomeTown
user1@gt.edu	1985	M	Atlanta
user2@gt.edu	1969	M	Austin
user3@gt.edu	1967	M	Portland
user4@gt.edu	1988	M	Atlanta

RESULT

Email	Interest
user1@gt.edu	Music
user1@gt.edu	Reading
user1@gt.edu	Tennis
user4@gt.edu	DIY

UserInterests

Email	Interest	SinceAge
user1@gt.edu	Music	10
user1@gt.edu	Reading	5
user1@gt.edu	Tennis	14
user2@gt.edu	Blogging	13
user3@gt.edu	Music	11
user4@gt.edu	DIY	18

上圖先看 inner query, 再看 outer query

Nested Queries: =, ≠, ≤, ≥, <, > SOME/ALL

```
SELECT CurrentCity
FROM RegularUser R, YearSalary Y
WHERE R.BirthYear = Y.BirthYear AND  Salary > ALL
(SELECT Salary
FROM RegularUser R, YearSalary Y
WHERE R.BirthYear = Y.BirthYear AND HomeTown = 'Austin');
```

OR

```
SELECT CurrentCity
FROM RegularUser R, YearSalary Y
WHERE R.BirthYear = Y.BirthYear AND  Salary >
(SELECT max(Salary)
FROM RegularUser R, YearSalary Y
WHERE R.BirthYear = Y.BirthYear AND HomeTown = 'Austin');
```

Find CurrentCitys with at least one RegularUser with a Salary that's higher than all Salaries of RegularUsers with HomeTown Austin

YearSalary

Birth Year	Salary
1985	27,000
1969	43,000
1967	45,000
1968	44,000
1988	24,000
1986	26,000
1974	38,000

RESULT

Current City
San Diego
College Park

RegularUser

Email	Birth Year	Sex	Current City	HomeTown
user1@gt.edu	1985	M	Seattle	Atlanta
user2@gt.edu	1969	M	Austin	Austin
user3@gt.edu	1967	M	San Diego	Portland
user6@gt.edu	1988	F	San Diego	Atlanta
user8@gt.edu	1968	M	College Park	Atlanta
user12@gt.edu	1974	F	College Park	Austin

上圖中 Hometown 是 Austin 的人(user2 和 user12)的收入是 43,000 和 38,000. 要找的是收入比這兩個都高的. 題目中的 Find CurrentCitys with at least one RegularUser...的意思是, 例如 San Diego 有兩個人(user3 和 user6), 他們的收入分別為 45,000 和 24,000, 有一個人的數入(45,000)比 43,000 和 38,000 都大, 所以 San Diego 要在 RESULT 中. College Park 同理也在 RESULT 中. 代碼中的 ALL 應該是和它下面的{}連在一起的, 即一個 ALL{...}語句.

Nested Queries – correlated

Find Email and BirthYear of RegularUsers who have no Interests

```
SELECT R.Email, BirthYear
FROM RegularUser R
WHERE NOT EXIST
    (SELECT *
    FROM  UserInterests  U
    WHERE U.Email = R.Email);
```

think of it as a sub-query evaluated once for each row of the outer query

RegularUser

Email	Birth Year	Sex
user1@gt.edu	1985	M
user2@gt.edu	1969	M
user3@gt.edu	1967	M
user4@gt.edu	1988	M
user6@gt.edu	1988	F
user8@gt.edu	1968	M
user9@gt.edu	1988	F

UserInterests

Email	Interest	SinceAge
user1@gt.edu	Music	10
user2@gt.edu	Blogging	13
user2@gt.edu	Meditation	21
user3@gt.edu	Music	11
user3@gt.edu	Reading	6
user4@gt.edu	DIY	18

RESULT

Email	Birth Year
user6@gt.edu	1988
user8@gt.edu	1968
user9@gt.edu	1988

(2016.12 表示從內往外看也沒問題)
上圖代碼中 inner query 用到 R, 但 R 是在 outer query 中定義的, 所以的 R 是一個 reference out of scope, 這是 correlated 的. 對於這種代碼, 不能從裡往外看, 而是要 think of it as a sub-query evaluated once for each row of the outer query. 即在表中一行一行地取, 代入代碼中由外向裡看. 比如 RegularUser 第一行 user1, 代入代碼外層中, R 就是 user1, 然後看內層, UserInterest 中能夠找出 user1@gt.edu, 所以 WHERE NOT EXIST{...}是 false 的, 所以 user1 沒在 RESULT 中. 同理, user6 在 RESULT 中.

Knowledge Check

How many rows are in the result of the SQL query?

- a) One
- b) Two
- c) Three
- d) Four

```
SELECT U.Email, SinceAge
FROM UserInterests  U
WHERE EXIST
    (SELECT *
    FROM  RegularUser R
    WHERE U.Email = R.Email);
```

UserInterests

Email	Interest	SinceAge
user2@gt.edu	Meditation	21
user3@gt.edu	Music	11
user3@gt.edu	Reading	6
user4@gt.edu	DIY	18

RegularUser

Email	Birth Year	Sex
user1@gt.edu	1985	M
user2@gt.edu	1969	M
user3@gt.edu	1967	M
user4@gt.edu	1988	M
user6@gt.edu	1988	F

上題中的四個即 user2, user3, user3, user4, 注意可以有 duplicate.