In the Lecture Series Introduction to Database Systems

SQL

Introduction to Database Systems

Using SQLite for the Lecture

```
.open cs2102.db
```

- Open or create the database

```
.mode column
.header on
```

- Set the column output
- Print the headers of the columns (attribute names) in output

```
PRAGMA foreign_keys = ON;
```

- Enable foreign keys

```
.read small_data.sql
```

- Run the SQL DDL and DML code

Querying One Table

Find all the information about students.

We can use either * or list the columns that we want to retrieve in the SELECT clause.

```
SELECT *
FROM student;
```

```
SELECT name, email, year, faculty, department, graduate
FROM student;
```

name	email	year	faculty	department	graduate
"GOH HUI YING"	gohhuiying1989@gmail.com	2008-01-01	"Faculty of Science"	Biology	
"TAY WEI GUO"	tayweiguo1989@msn.com	2010-08-01	"Faculty of Engineering"	CE	
"PENG JIAYUAN"	pengjiayuan2011@hotmail.com	2008-01-01	"Faculty of Science"	Biology	
"HUANG ZHANPENG"	huangzhanpeng1992@msn.com	2010-08-01	"Faculty of Arts and Social Science"	Geography	
…					

Selecting Columns

Find the names and emails of students.

We can selectively choose the columns we want to retrieve in the SELECT clause.

```
SELECT name, email
FROM student;
```

name	email
"GOH HUI YING"	gohhuiying1989@gmail.com
"TAY WEI GUO"	tayweiguo1989@msn.com
"PENG JIAYUAN"	pengjiayuan2011@hotmail.co
"HUANG ZHANPENG"	huangzhanpeng1992@msn.co
"ZHENG ZHEMIN"	zhengzhemin1991@yahoo.com
"LIU ZHANPENG"	liuzhanpeng2011@msn.com
…	

Selecting Rows

Find the names and emails of computer science students.

The WHERE clause is used for conditions.

```
SELECT name, email
FROM student
WHERE department='CS';
```

name	email
"LIU SHAOJUN"	liushaojun2010@msn.com
"QIN YIYANG"	qinyiyang2010@hotmail.com
"SIOW CAO KHOA"	siowcaokhoa1991@msn.com
"DAVID CHAPMAN"	davidchapman1989@msn.com
…	

Selecting Rows

Find the names and emails of students who joined after '2014-01-01'.

There are several comparison operators for each domain.

```
SELECT name, email
FROM student
WHERE year >= '2010-01-01';
```

name	email
"TAY WEI GUO"	tayweiguo1989@msn.com
"HUANG ZHANPENG"	huangzhanpeng1992@msn.com
"WANG NA"	wangna1990@yahoo.com
"SIOW CAO KHOA"	siowcaokhoa1991@msn.com
"DENNIS BECKHAM"	dennisbeckham1989@msn.com
…	

Selecting Rows

Find the names and emails of students who graduated after '2014-01-01'.

Null values need a special careful treatment.

```
SELECT name, email
FROM student
WHERE graduate >= '2014-01-01';
             No results! (nobody has graduated yet!)
```

Find the names and emails of students who have not graduated before 2014-01-01'.

```
SELECT name, email
FROM student
WHERE graduate >= '2014-01-01' OR graduate iS NULL;
```

The condition must be TRUE (not FALSE, not UNKNOWN)

name	email
"GOH HUI YING"	gohhuiying1989@gmail.com
"TAY WEI GUO"	tayweiguo1989@msn.com
"PENG JIAYUAN"	pengjiayuan2011@hotmail.com
"HUANG ZHANPENG"	huangzhanpeng1992@msn.com
…	

Querying Multiple Tables

Find the names of students and the titles of the books that they own.

We can list all the tables we want to use in the FROM clause.
We are querying the Cartesian product of these tables.

```
SELECT student.name, book.title
FROM student, copy, book
WHERE student.email=copy.owner
AND copy.book=book.ISBN13;
```

This is the Cartesian product:

```
SELECT student.name, book.title
FROM student, copy, book;
```

It is very big! (how big?)

name	title
"DAVID HALL"	"The Digital Photography Book"
"GOH HUI YING"	"Photoshop Elements 9: The Missing Manual"
"HUANG ZHANPENG"	"Where Good Ideas Come From: The Natural History of Innovation"
"JENNY HUNT"	"Photoshop Elements 9: The Missing Manual"
"LIU SHAOJUN"	"The Great Gatsby"
"LIU ZHENCAI"	"The Digital Photography Book"
"NG YONG MING"	"The Digital Photography Book"
"QIN YUWEI"	"Where Good Ideas Come From: The Natural History of Innovation"
"TAY WEI GUO"	"The Great Gatsby"
"TSO HUI LIN"	"Photoshop Elements 9: The Missing Manual"
"WANG NA"	"The Digital Photography Book"
"WANG NA"	"Photoshop Elements 9: The Missing Manual"
"ZHANG HAN"	"Where Good Ideas Come From: The Natural History of Innovation"
"ZHU CHANG"	"Where Good Ideas Come From: The Natural History of Innovation"

Tuple Variables

Find the names of students and the titles of the books they own.

It is best to use t-uple variables (see t-uple calculus).

```
SELECT s.name, b.title
FROM student s, copy c, book b
WHERE s.email=c.owner
AND c.book=b.ISBN13;
```

s, c and b are t-uple variables.

Renaming

Find the names of students who lent a book returned after 2010-03-04 to anniechapman1991@yahoo.com.

Renaming of the columns in the result is done with ``AS''.

```
SELECT s.name AS owner
FROM loan l, student s
WHERE  s.email=l.owner
  AND l.returned  > '2010-03-04'
  AND l.borrower =
  'anniechapman1991@yahoo.com';
```

owner
"JENNY HUNT"
"JENNY HUNT"
"JENNY HUNT"
"JENNY HUNT"
"QIN YUWEI"
"QIN YUWEI"

Duplicates

Find the different names of students who lent a book returned after 2010-03-04 to anniechapman1991@yahoo.com.

One can eliminate duplicates rows in the result table using ``DISTINCT''.

```
SELECT DISTINCT s.name AS owner
FROM loan l, student s
WHERE  s.email=l.owner
   AND l.returned  > '2010-03-04'
   AND l.borrower = 'anniechapman1991@yahoo.com';
```

owner
"JENNY HUNT"
"QIN YUWEI"

Ordering Rows

Find the names of students who lent a book returned after 2010-03-04 to anniechapman1991@yahoo.com in descending alpha-numerical order.

The result can be ordered using the ORDER BY clause. (default is ASC)

```
SELECT s.name AS owner
FROM loan l, student s
WHERE s.email=l.owner
  AND l.returned  > '2010-03-04'
  AND l.borrower =
  'anniechapman1991@yahoo.com'
  ORDER BY name DESC;
```

owner
"QIN YUWEI"
"QIN YUWEI"
"JENNY HUNT"
"JENNY HUNT"
"JENNY HUNT"
"JENNY HUNT"

Ordering Rows

Find the ISBN14 of the books that have been borrowed by anniechapman1991@yahoo.com , their borrowing and return dates in ascending order of the borrowing and return dates.

We can order according to several columns.

We can order according to columns not in the result.

```
SELECT l.book, l.returned
FROM loan l
WHERE l.borrower='anniechapman1991@yahoo.com'
ORDER BY l.borrowed, l.returned;
```

book	returned
978-1449389673	2010-03-19
978-1594487712	2010-03-18
978-1449389673	2010-06-13
978-1594487712	2010-06-13
978-1449389673	2010-06-22
978-1449389673	2010-08-13

Arithmetic and Renaming

Find the price of books after tax (18%).

Arithmetic and other operations are available for most domains.

```
CREATE TABLE catalog (
book CHAR(14) PRIMARY KEY,
price number);

SELECT * FROM catalog;
```

```
SELECT book, price * 1.18 AS priceGST
  FROM catalog;
```

book	price
978-0321474049	24
978-0684801520	35
978-1449389673	12
978-1594487712	55

book	priceGST
978-0321474049	28.32
978-0684801520	41.3
978-1449389673	14.16
978-1594487712	64.9

Summary

Syntax	Semantics
1. SELECT	1. FROM
2. FROM	2. WHERE
3. WHERE	3. ORDER BY
4. ORDER BY	4. SELECT

Aggregates

Aggregate Queries: Counting Rows

Find the total number of (different) books.

Aggregate queries use aggregate functions like ``COUNT()'' to combine results over entire tables or columns.

```
SELECT COUNT(*) FROM book b;
```

`COUNT()`, `MAX()`, `MIN()`, `AVG()`, `STD()`, `SUM()` etc.

COUNT(*)
4

Aggregate Queries: Counting Rows

Find the total number of titles.

```
SELECT COUNT(l.book)
FROM loan l;
```

```
SELECT COUNT(ALL l.book)
FROM loan l;
```

COUNT(l.book)
55

Aggregate Queries: Counting Rows

Find the total number of different titles.

```
SELECT COUNT(DISTINCT l.book)
FROM loan l;
```

"COUNT(DISTINCT l.book)"
4

Aggregate Queries: Average, Minimum, etc.

Find the average price of books in the catalog.

```
SELECT AVG(c.price)
FROM catalog c;
```

book	price
978-0321474049	24
978-0684801520	35
978-1449389673	12
978-1594487712	55

AVG(c.price)
31.5

Aggregate Queries: Grouping

Find, for each day, the number of books borrowed by anniechapman1991@yahoo.com.

```
SELECT COUNT(l.book)
FROM loan l
WHERE l.borrower='anniechapman1991@yahoo.com'
GROUP BY l.borrowed;
```

COUNT(l.book)
1
1
2
1
1

Aggregate Queries: Grouping

Let us try and understand what is happening.

We order by instead of grouping.

```
SELECT *
FROM loan l
WHERE l.borrower='anniechapman1991@yahoo.com'
ORDER BY l.borrowed;
```

We see some groups appearing.

The GROUP BY clause creates groups before the aggregation but after the WHERE clause.

We count for each group. Can we get 0?

borrower	owner	book	copy	borrowed	returned	
anniechapman1991@yahoo.com	jennyhunt1991@gmail.com	978-1449389673	1	2010-02-17	2010-03-19	1
anniechapman1991@yahoo.com	qinyuwei2011@hotmail.com	978-1594487712	1	2010-03-17	2010-03-18	1
anniechapman1991@yahoo.com	jennyhunt1991@gmail.com	978-1449389673	1	2010-05-17	2010-06-13	2
anniechapman1991@yahoo.com	qinyuwei2011@hotmail.com	978-1594487712	1	2010-05-17	2010-06-13	
anniechapman1991@yahoo.com	jennyhunt1991@gmail.com	978-1449389673	1	2010-06-17	2010-06-22	1
anniechapman1991@yahoo.com	jennyhunt1991@gmail.com	978-1449389673	1	2010-07-17	2010-08-13	1

Aggregate Queries: Grouping

Which one eliminates duplicates?

```
SELECT DISTINCT l.book
FROM loan l;

SELECT l.book
FROM loan l
GROUP BY l.book;
```

Interestingly, the GROUP BY clause can be used to eliminate duplicates. Sometimes it is the only way to do so.

For readability of the queries, unless impossible, prefer ``DISTINCT''.

book
978-1449389673
978-1594487712
978-0321474049
978-0684801520

book
978-0321474049
978-0684801520
978-1449389673
978-1594487712

Aggregate Queries: Grouping

Find, for each day, the number of books borrowed by anniechapman1991@yahoo.com, print the day and the quantity.

```
SELECT l.borrowed, COUNT(l.book)
FROM loan l
WHERE l.borrower='anniechapman1991@yahoo.com'
GROUP BY l.borrowed;
```

Only attributes in the GROUP BY clause and aggregate functions can be used in the SELECT (and HAVING) clauses.

borrowed	COUNT(l.book)
2010-02-17	1
2010-03-17	1
2010-05-17	2
2010-06-17	1
2010-07-17	1

Aggregate Queries: Grouping

Find, for each day, the number of books borrowed by anniechapman1991@yahoo.com print the borrower, the day and the quantity.

```
SELECT l.borrower, l.borrowed, COUNT(l.book)
FROM loan l
WHERE l.borrower='anniechapman1991@yahoo.com'
GROUP BY l.borrowed, l.borrower;
```

It is required to have l.borrower in the GROUP BY clause (even though some system like SQLite accept it, they may give random answers)

Aggregate Queries: You Try!

```
SELECT l.owner, l.borrowed, COUNT(l.book)
FROM loan l
WHERE l.borrower='anniechapman1991@yahoo.com'
GROUP BY l.borrowed;
```

```
SELECT l.owner, l.borrowed
FROM loan l
WHERE l.borrower='anniechapman1991@yahoo.com';
```

borrower	borrowed	COUNT(l.book)
anniechapman1991@ yahoo.com	2010-02-17	1
anniechapman1991@ yahoo.com	2010-03-17	1
anniechapman1991@ yahoo.com	2010-05-17	2
anniechapman1991@ yahoo.com	2010-06-17	1
anniechapman1991@ yahoo.com	2010-07-17	1

Aggregate Queries: Grouping

What does this query find?

```
SELECT l.borrower, l.borrowed, COUNT(l.book)
FROM loan l
GROUP BY l.borrowed, l.borrower, l.book;
```

What does this query find?

```
SELECT l.borrower, l.borrowed, COUNT(l.book)
FROM loan l;
```

What does this query find?

```
SELECT COUNT(l.book)
FROM loan l;
```

What does this query find?

```
SELECT COUNT(DISTINCT l.book)
FROM loan l;
```

Aggregate Queries: Condition

Find the students who borrowed more that one book on any given day.

```
SELECT l.borrower
FROM loan l
GROUP BY l.borrowed, l.borrower
WHERE COUNT(l.book) >1;
```

“Incorrect syntax near the keyword 'WHERE'.”

Aggregate functions cannot be used in the WHERE clause.

Aggregate Queries: Condition

Find the students who borrowed more that one book on any given day.

We need the HAVING clause.

```
SELECT l.borrower
FROM loan l
GROUP BY l.borrowed, l.borrower
HAVING COUNT(l.book) >1;
```

borrower
anniechapman1991@yahoo.com

Summary

Syntax	Semantics
1. SELECT	1. FROM
2. FROM	2. WHERE
3. WHERE	3. GROUP BY
4. GROUP BY	4. HAVING
5. HAVING	5. ORDER BY
6. ORDER BY	6. SELECT

Credits

The content of this lecture is based on chapter 5 of the book "Introduction to database Systems"
By
S. Bressan and B. Catania,
McGraw Hill publisher

Clipart and media are licensed from Microsoft Office Online Clipart and Media