Xi'an Jiaotong-Liverpool University
西交利物浦大學

SQL Select

Jianjun Chen (Jianjun.Chen@xjtlu.edu.cn)

SELECT: Overview

- The select statement is designed to allow database clients to look up data from tables.
- Many behaviours of SELECT can be described using arrays and loops in C/Java.
  - Database is simply a more sophisticated program

This lecture

```
SELECT [DISTINCT | ALL]
   column-list FROM table-names
    [WHERE condition]
    [ORDER BY column-list]
    [GROUP BY column-list]
    [HAVING condition]
```

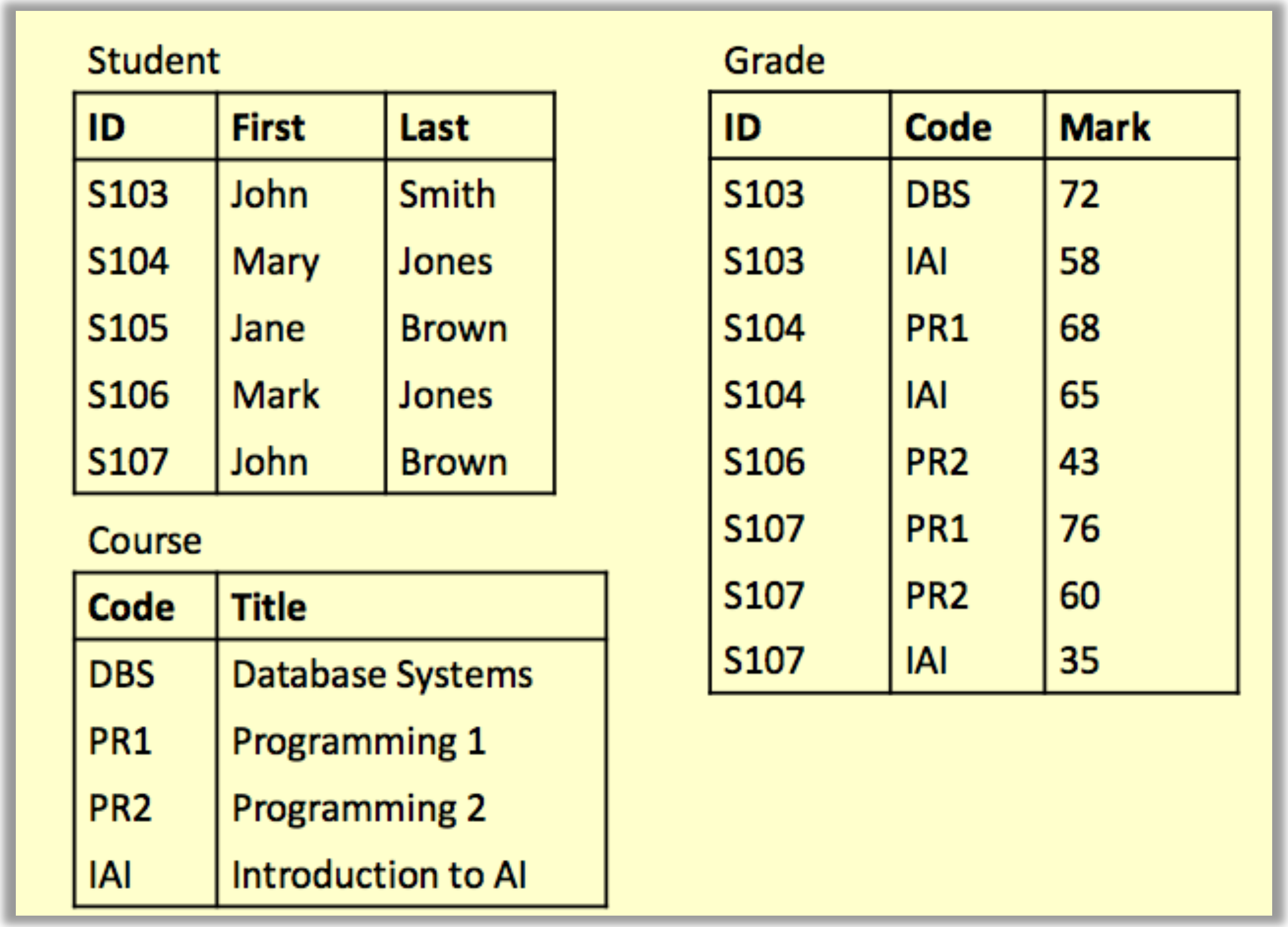
Contents

- Filtering rows and columns.
  - The where clause

- Cartisian product
  - Alias
  - Self-join

- Subqueries
  - Handling set

Examples `SEL-1`



Student

ID	First	Last
S103	John	Smith
S104	Mary	Jones
S105	Jane	Brown
S106	Mark	Jones
S107	John	Brown

Course

Code	Title
DBS	Database Systems
PR1	Programming 1
PR2	Programming 2
IAI	Introduction to AI

Grade

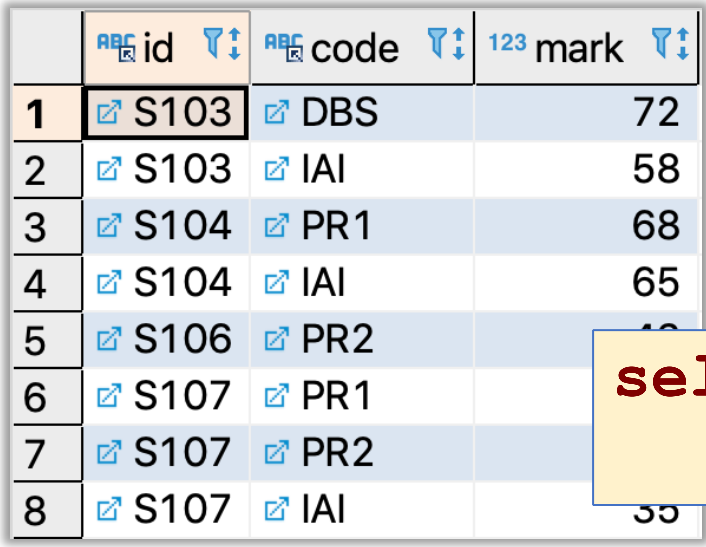
ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

SELECT

- In its most basic form, SELECT is able to fetch the columns and rows of a table.
- To filter columns of a table:

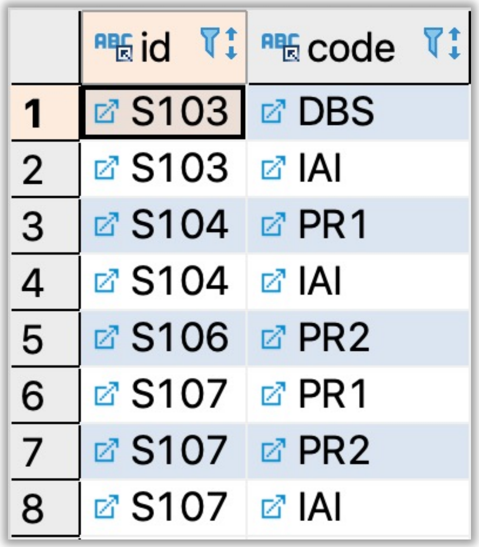
```
SELECT col1[,col2…] FROM table-name;
```

- Example SEL-1:



	id	code	mark
1	S103	DBS	72
2	S103	IAI	58
3	S104	PR1	68
4	S104	IAI	65
5	S106	PR2	[illegible]
6	S107	PR1	[illegible]
7	S107	PR2	[illegible]
8	S107	IAI	35

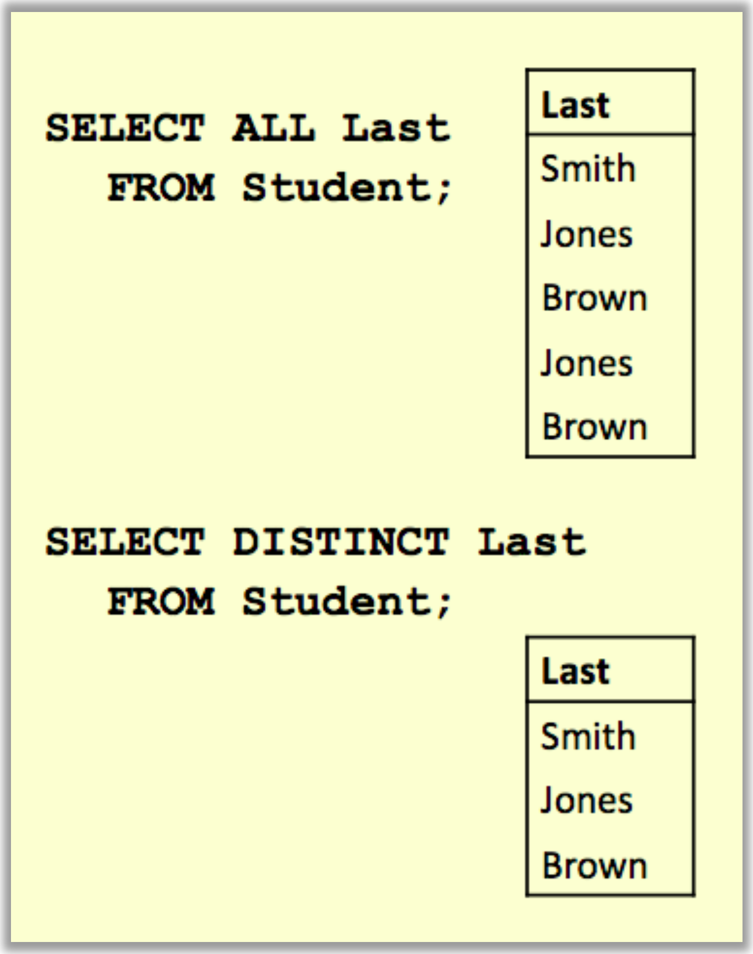
```
select id, code
    from grade;
```



	id	code
1	S103	DBS
2	S103	IAI
3	S104	PR1
4	S104	IAI
5	S106	PR2
6	S107	PR1
7	S107	PR2
8	S107	IAI

DISTINCT and ALL

- By default, select keeps duplicate tuples.
- Using DISTINCT after the SELECT keyword removes duplicates .
- Using ALL retains duplicates
  - ALL is used as a default if neither is supplied
- These will work over multiple columns (How?)
  - See example SEL-1



```
SELECT ALL Last
   FROM Student;
```

Last
Smith
Jones
Brown
Jones
Brown

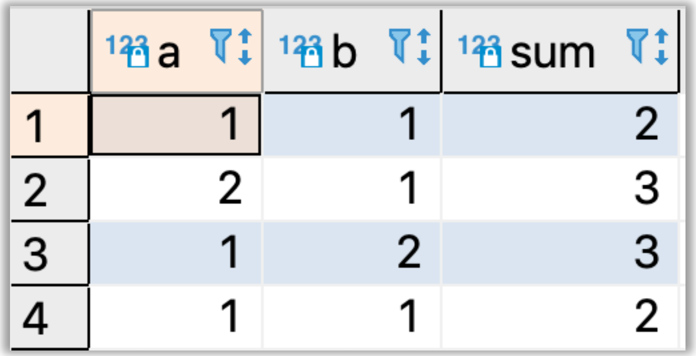
```
SELECT DISTINCT Last
   FROM Student;
```

Last
Smith
Jones
Brown

Expressions in SELECT

- You can put simple expressions in `SELECT` statements.
- The `AS` keyword is explained later. It simply gives a column a new name.
    - Code is in example `SEL-1`

```
select a, b, a+b as sum
    from dup_test;
```



	a	b	sum
1	1	1	2
2	2	1	3
3	1	2	3
4	1	1	2

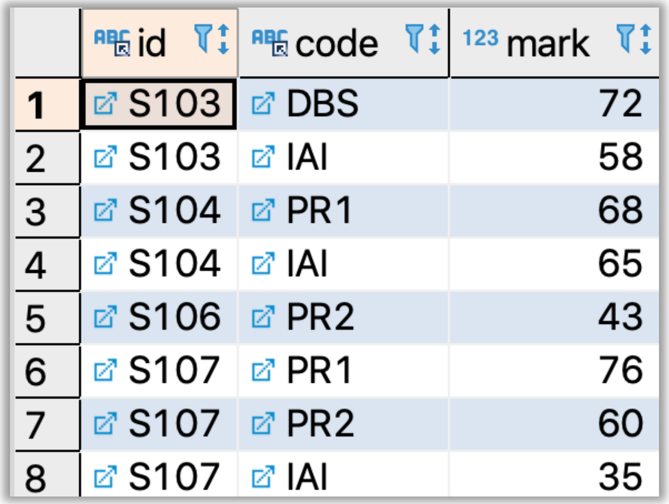
Where

- To filter rows of a table:

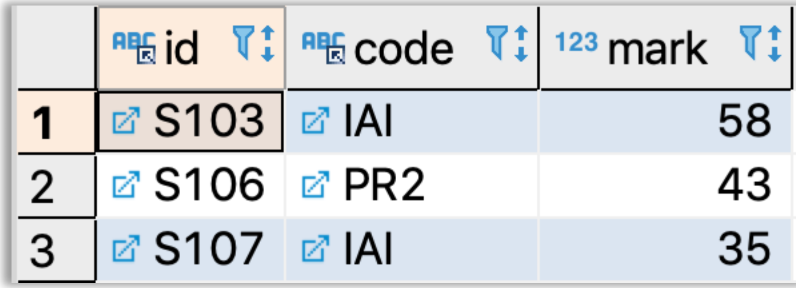
```
SELECT * FROM table-name
    WHERE predicate;
```

  - Asterisk (*) means getting all columns of that table.

- Example `SEL-1`:



	id	code	mark
1	S103	DBS	72
2	S103	IAI	58
3	S104	PR1	68
4	S104	IAI	65
5	S106	PR2	43
6	S107	PR1	76
7	S107	PR2	60
8	S107	IAI	35



	id	code	mark
1	S103	IAI	58
2	S106	PR2	43
3	S107	IAI	35

```
select * from grade
    where mark < 60;
```

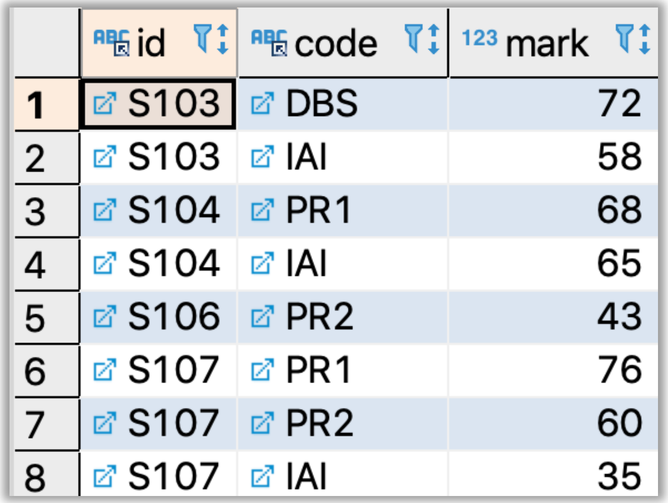
The WHERE Clause

- A WHERE clause restricts rows that are returned
  - It takes the form of a Predicate.
  - **Predicate**: can be understood as an expression that either returns a true or false (for numbers, non-zero or zero).
- Only rows that satisfy the condition will appear in the final result.

Expression	Meaning
Mark < 40	The value of column `mark` is less than 40
First = 'John'	The value of column `First` equals to 'John'
First = Last	`First` equals to `Last`
First IS NULL	`First` column has no value
First != 'John' First <> 'John'	The value of column `First` NOT equals to 'John'
(First = 'John') AND (Last = 'Smith')	`First` equals to 'John' and `Last` equals to 'Smith'
(Mark < 40) OR (Mark > 70)	Mark is lower than 40 or higher than 70

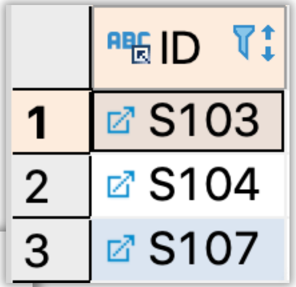
WHERE: Evaluation Process

- The evaluation process of select:
  - Get the table (the `FROM` part)
  - For each tuple, assess the `WHERE` clause.
    - `True` -> accepted
    - `False`-> removed
  - Get columns (the `SELECT` part)



	id	code	mark
1	S103	DBS	72
2	S103	IAI	58
3	S104	PR1	68
4	S104	IAI	65
5	S106	PR2	43
6	S107	PR1	76
7	S107	PR2	60
8	S107	IAI	35

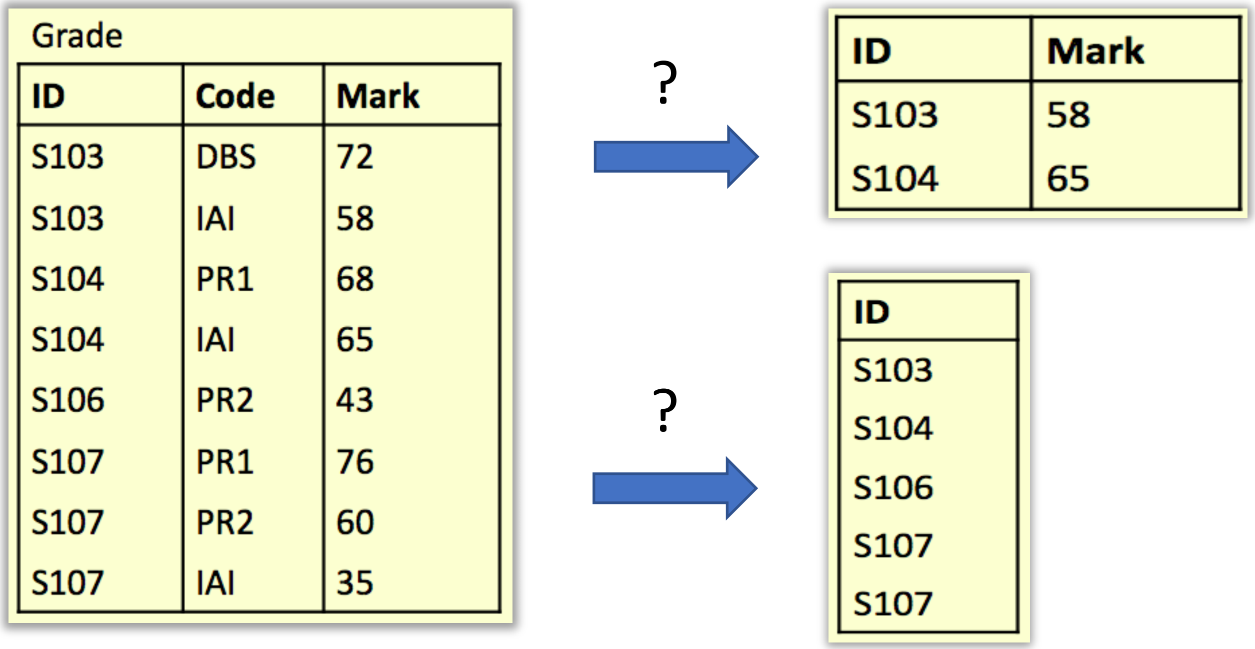
```
SELECT DISTINCT ID
    FROM Grade
    WHERE Mark >= 60;
```



	ID
1	S103
2	S104
3	S107

Question

- Write an SQL query to find a list of the ID numbers and Marks for students who have passed (scored 50% or more) in IAI.
- Write an SQL query to find the combined list of the student IDs for both the IAI and PR2 module.



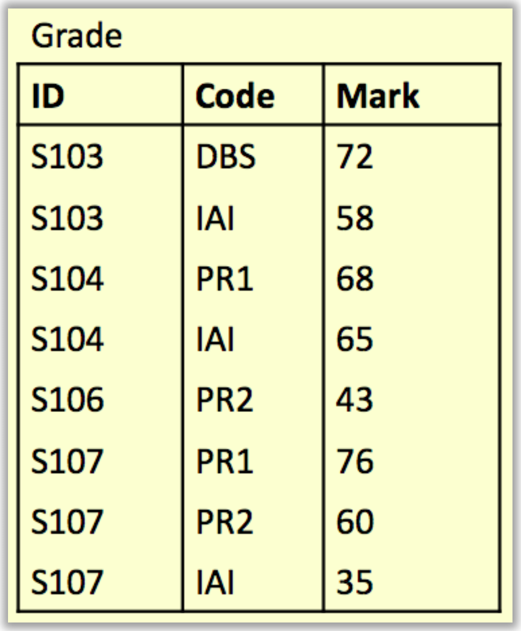
Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

ID	Mark
S103	58
S104	65

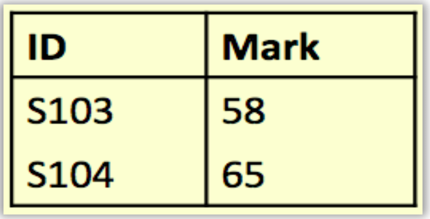
ID
S103
S104
S106
S107
S107

```
SELECT ID, Mark FROM Grade
       WHERE (Code = 'IAI') AND (Mark >= 50);
```

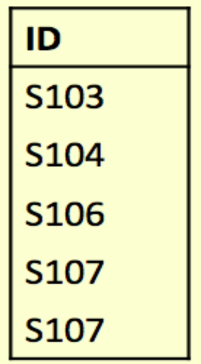


Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35



ID	Mark
S103	58
S104	65



ID
S103
S104
S106
S107
S107

```
SELECT ID FROM Grade
       WHERE (Code = 'IAI' OR Code = 'PR2');
```

Word Search

- Commonly used for searching product catalogues etc.
  - Need to search by keywords
  - Might need to use partial keywords
- For example: given a database of books, searching for “crypt” might return
  - “Cryptonomicon” by Neil Stephenson
  - “Applied Cryptography” by Bruce Schneier
- We can use the `LIKE` keyword to perform string comparisons in queries
  - `Like` is not the same as ‘=’ because it allows wildcard characters
  - It is NOT normally case sensitive

```
SELECT * FROM books
     WHERE bookName LIKE '%crypt%';
```

LIKE

- The ‘%’ character can represent any number of characters, including none
  - The following example will return “Cryptography Engineering” and “Cryptonomicon” but not “Applied Cryptography”

```
bookName LIKE 'crypt%'
```

- The ‘_’ character represents exactly one character
  - The following example will return “Clouds” but not “Cloud” or “cloud computing”

```
bookName LIKE 'cloud_'
```

LIKE

- Sometimes you might need to search for a set of words
  - To find entries with all words you can link conditions with AND

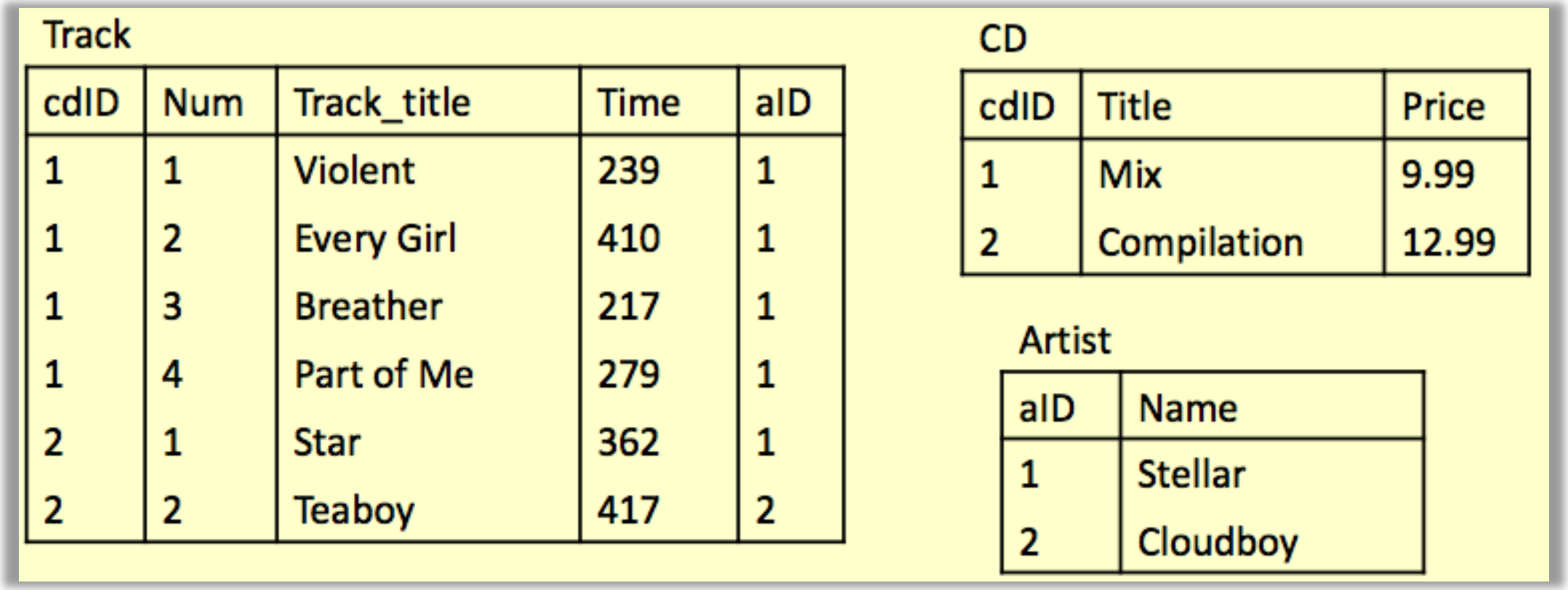
```
SELECT * FROM books WHERE
    bookName LIKE '%crypt%'
    AND bookName LIKE '%cloud%';
```

  - To find entries with any words use OR

```
SELECT * FROM books WHERE
    bookName LIKE '%crypt%'
    OR bookName LIKE '%cloud%';
```

Example

- There are three tables below.
- Write a query to find any track title containing either the string ‘boy’ or ‘girl’.



Track

cdID	Num	Track_title	Time	aID
1	1	Violent	239	1
1	2	Every Girl	410	1
1	3	Breather	217	1
1	4	Part of Me	279	1
2	1	Star	362	1
2	2	Teaboy	417	2

CD

cdID	Title	Price
1	Mix	9.99
2	Compilation	12.99

Artist

aID	Name
1	Stellar
2	Cloudboy

Solution

- Step 1: decide the table you need.

```
SELECT * FROM Track;
```

- Step 2: decide the rows you want.

```
SELECT * FROM Track WHERE
    Track_title LIKE '%boy%'
    OR Track_title LIKE '%girl%';
```

- Step 3: decide the columns you need.

```
SELECT Track_title FROM Track WHERE
    Track_title LIKE '%boy%'
    OR Track_title LIKE '%girl%';
```

Dealing with Date and Time

- The comparison of date and time can be done just like numbers.

```
SELECT * FROM table-name
    WHERE date-of-event < '2012-01-01';
```

- But you can also search for dates like a string:

```
SELECT * FROM table-name
    WHERE date-of-event LIKE '2014-11-%';
```

- Check the example `SEL-1`

Logical Statements in SELECT

- All statements that return Boolean values can also be placed in the `SELECT` section:
  - SELECT postcode LIKE ‘gb%’ FROM places;
  - SELECT id BETWEEN 1 AND 5 FROM staff;
  - …

- What are the results of these? Design some tables and find them out ☺.

More about the WHERE Clause

- In the WHERE expression, you can use any of the functions and operators that MySQL supports.
  - except for aggregate (group) functions.
  - Aggregate functions will be introduced later.

- The full list of supported operations can be found at:
  - https://dev.mysql.com/doc/refman/8.0/en/expressions.html

- Do your own experiments to find out how they works.

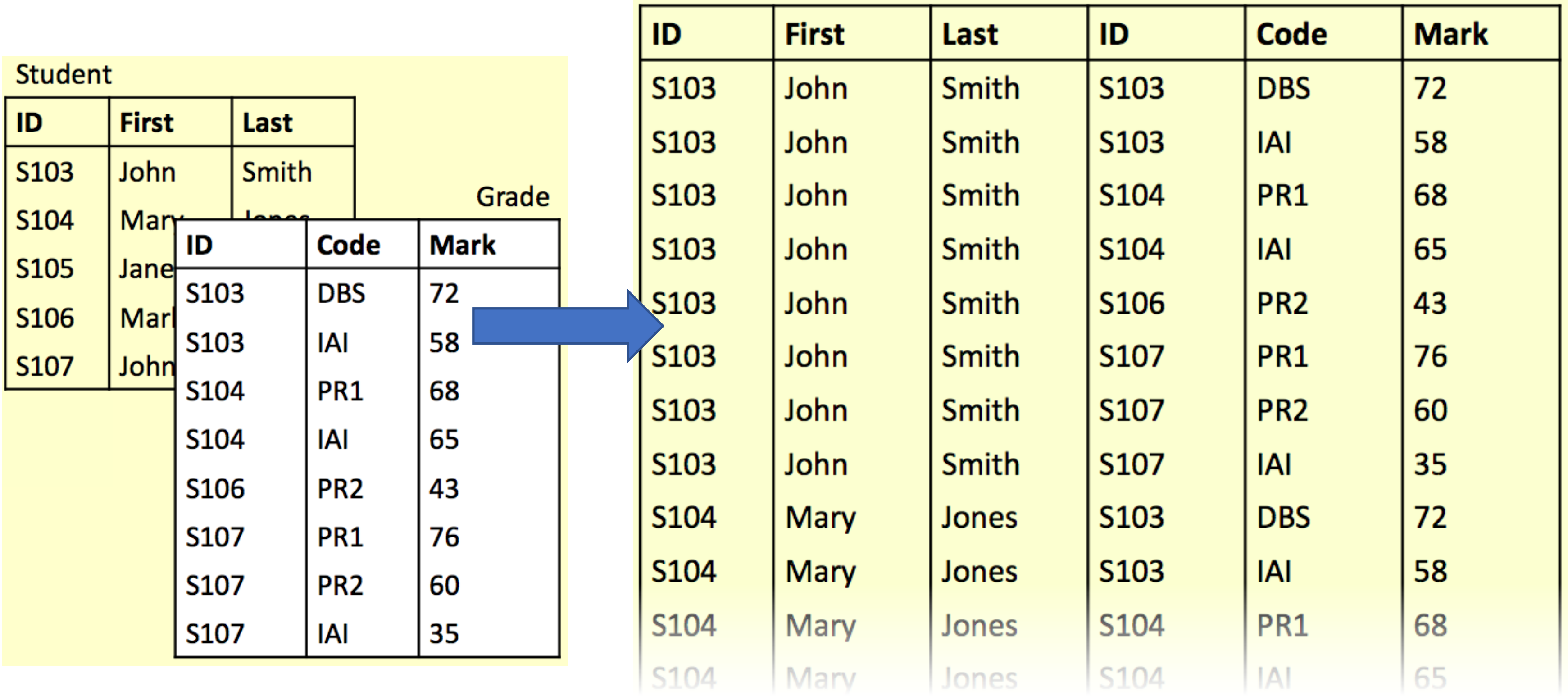
Select and Cartesian Product

Combining multiple tables

SELECT and Cartesian Product

- Cartesian product of two tables can be obtained by using:

```
SELECT * FROM Student, Grade;
```



Student

ID	First	Last
S103	John	Smith
S104	Mary	Jones
S105	Jane	
S106	Mar	
S107	John	

Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

ID	First	Last	ID	Code	Mark
S103	John	Smith	S103	DBS	72
S103	John	Smith	S103	IAI	58
S103	John	Smith	S104	PR1	68
S103	John	Smith	S104	IAI	65
S103	John	Smith	S106	PR2	43
S103	John	Smith	S107	PR1	76
S103	John	Smith	S107	PR2	60
S103	John	Smith	S107	IAI	35
S104	Mary	Jones	S103	DBS	72
S104	Mary	Jones	S103	IAI	58
S104	Mary	Jones	S104	PR1	68
S104	Mary	Jones	S104	IAI	65

SELECT and Cartesian Product

- If the tables have columns with the same name, ambiguity will result.
- This can be resolved by referencing columns with the table name:

```
TableName.ColumnName
```

- For example:

```
SELECT Student.ID FROM Student, Grade
     WHERE Student.ID = Grade.ID;
```

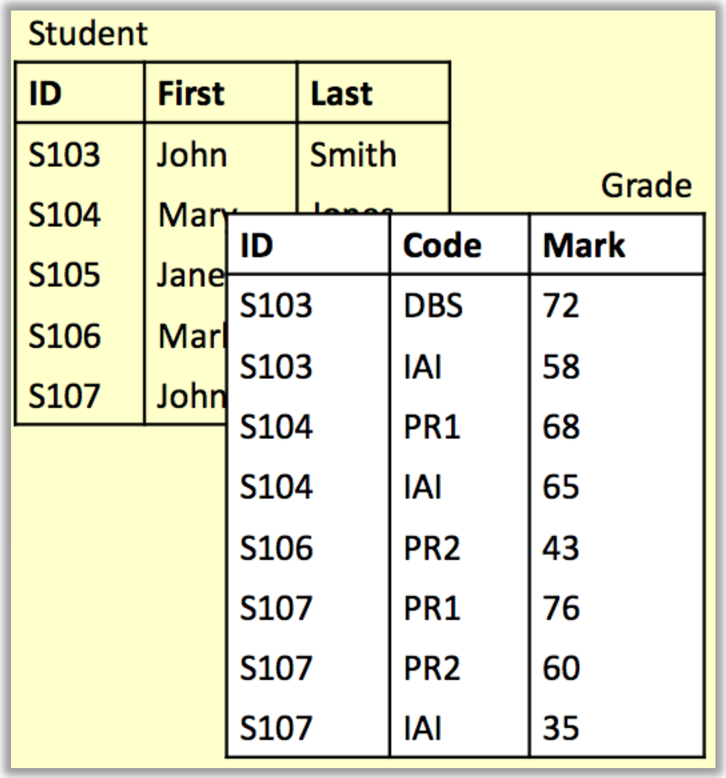
- The statement below is wrong (ambiguous):

```
SELECT ID FROM Student, Grade
     WHERE Student.ID = Grade.ID;
```

Cartesian Product: Example

- In this example, we want to find the names and marks of students.

```
SELECT
   First, Last, Mark
   FROM Student,
Grade
   WHERE
(Student.ID =
Grade.ID)
AND (Mark >= 40);
```



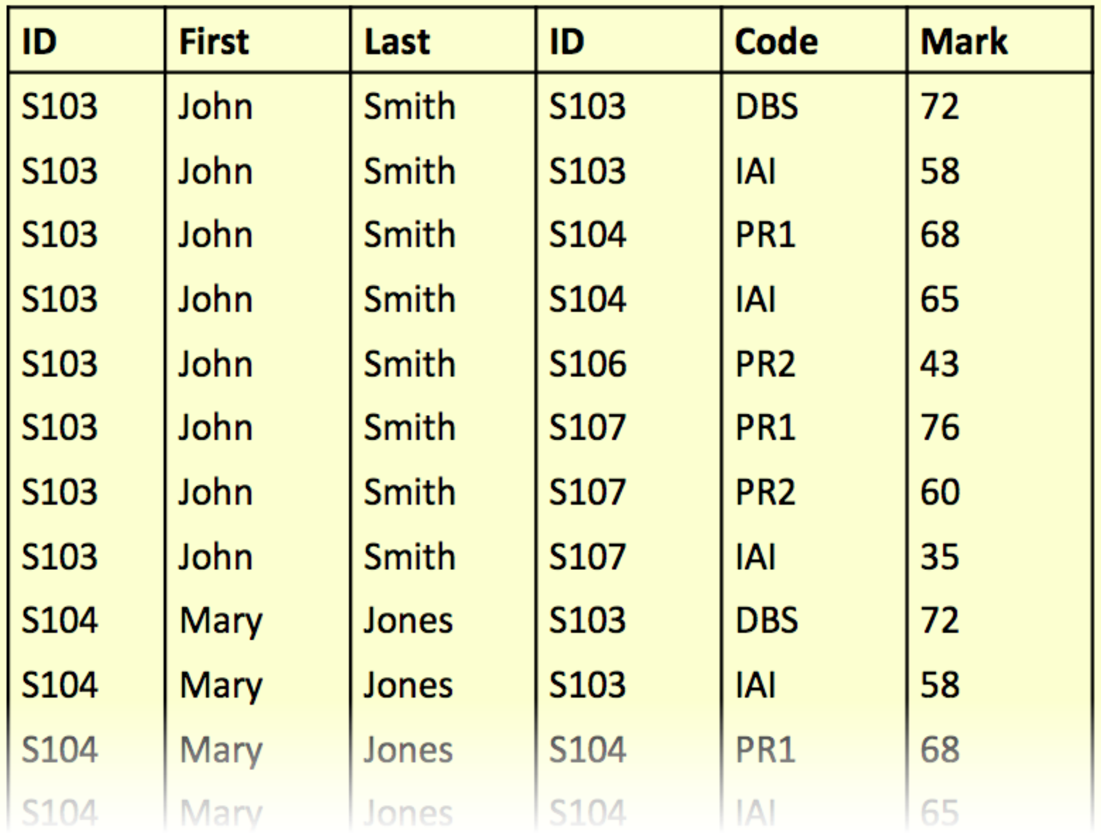
Student

ID	First	Last
S103	John	Smith
S104	Mary	Jones
S105	Jane	
S106	Mark	
S107	John	

Grade

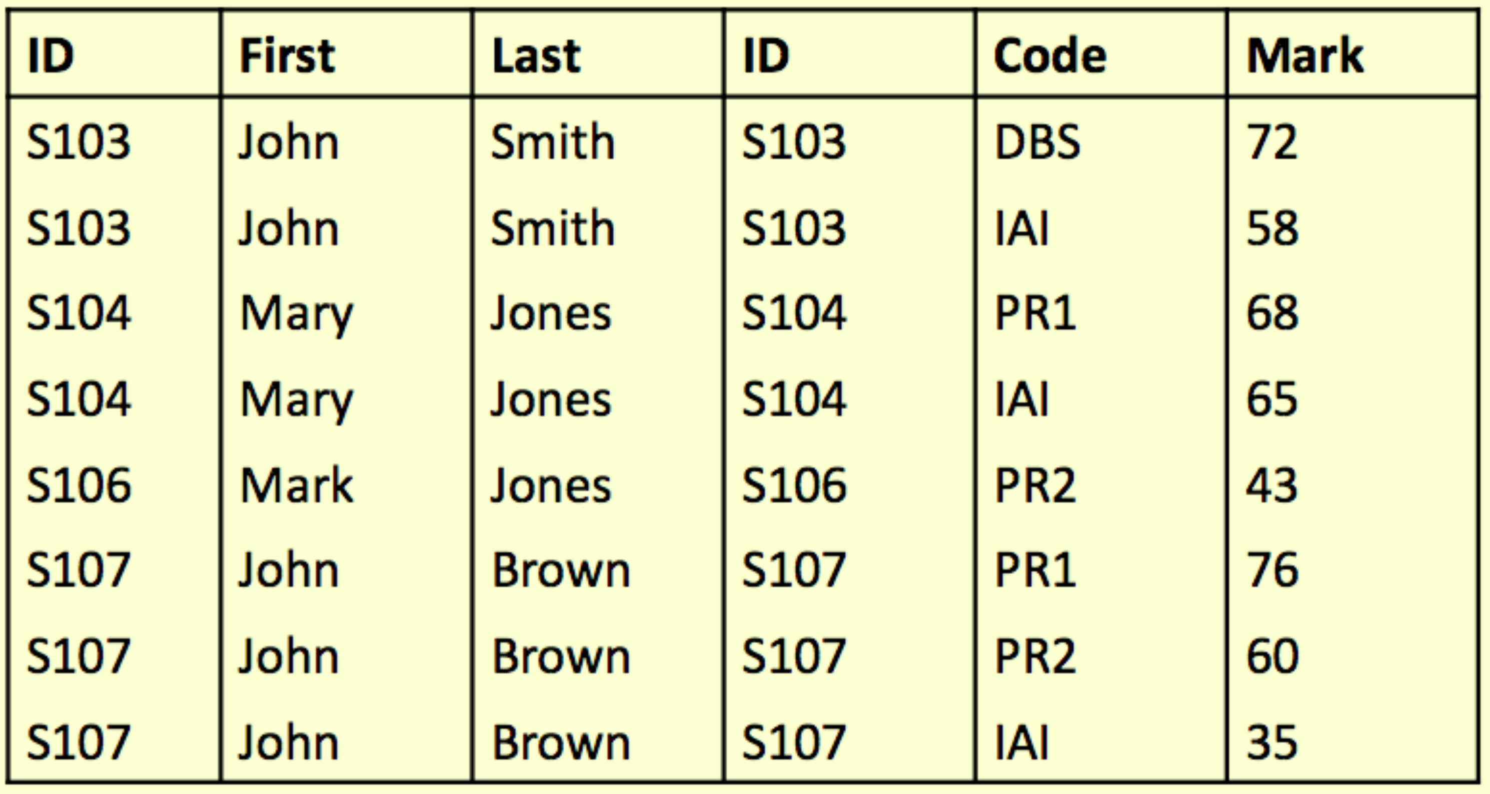
ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

SELECT ... **FROM** Student, Grade **WHERE** ...



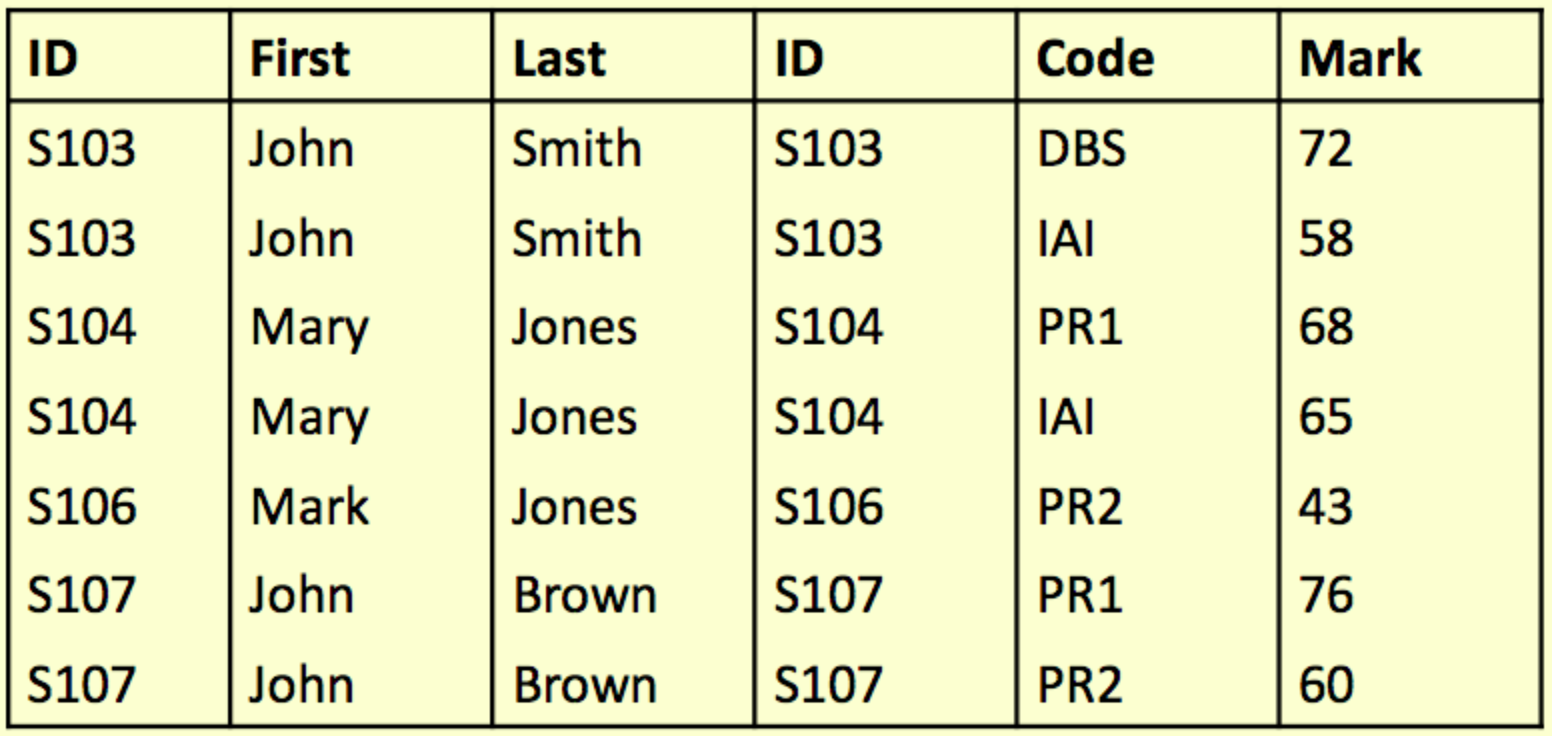
ID	First	Last	ID	Code	Mark
S103	John	Smith	S103	DBS	72
S103	John	Smith	S103	IAI	58
S103	John	Smith	S104	PR1	68
S103	John	Smith	S104	IAI	65
S103	John	Smith	S106	PR2	43
S103	John	Smith	S107	PR1	76
S103	John	Smith	S107	PR2	60
S103	John	Smith	S107	IAI	35
S104	Mary	Jones	S103	DBS	72
S104	Mary	Jones	S103	IAI	58
S104	Mary	Jones	S104	PR1	68
S104	Mary	Jones	S104	IAI	65

```
SELECT ... FROM Student, Grade WHERE
    (Student.ID = Grade.ID) AND ...
```



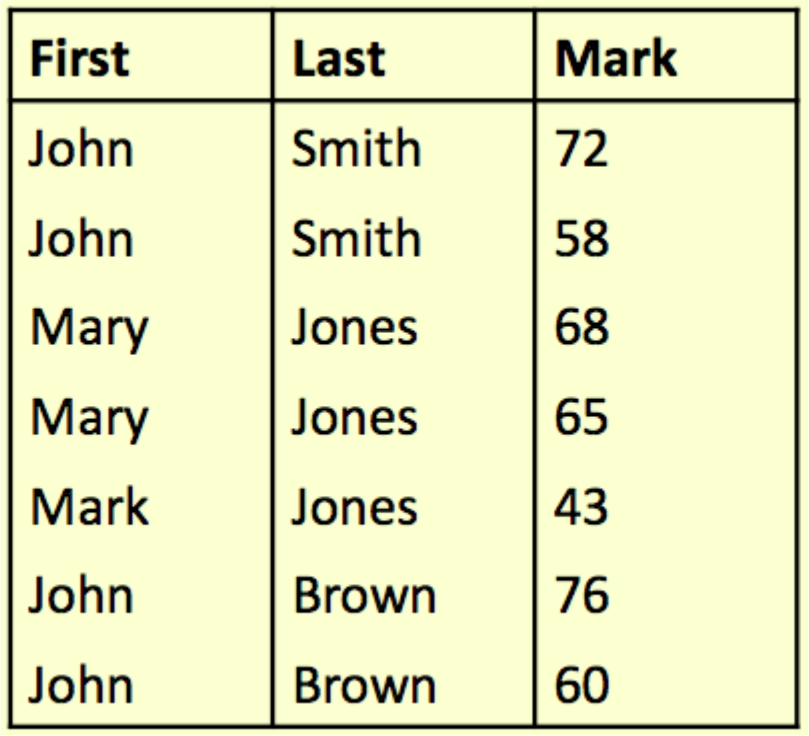
ID	First	Last	ID	Code	Mark
S103	John	Smith	S103	DBS	72
S103	John	Smith	S103	IAI	58
S104	Mary	Jones	S104	PR1	68
S104	Mary	Jones	S104	IAI	65
S106	Mark	Jones	S106	PR2	43
S107	John	Brown	S107	PR1	76
S107	John	Brown	S107	PR2	60
S107	John	Brown	S107	IAI	35

```
SELECT ... FROM Student,Grade
WHERE (Student.ID = Grade.ID)
  AND (Mark >= 40)
```



ID	First	Last	ID	Code	Mark
S103	John	Smith	S103	DBS	72
S103	John	Smith	S103	IAI	58
S104	Mary	Jones	S104	PR1	68
S104	Mary	Jones	S104	IAI	65
S106	Mark	Jones	S106	PR2	43
S107	John	Brown	S107	PR1	76
S107	John	Brown	S107	PR2	60

```
SELECT First, Last, Mark
   FROM Student, Grade
   WHERE (Student.ID = Grade.ID)
     AND (Mark >= 40)
```

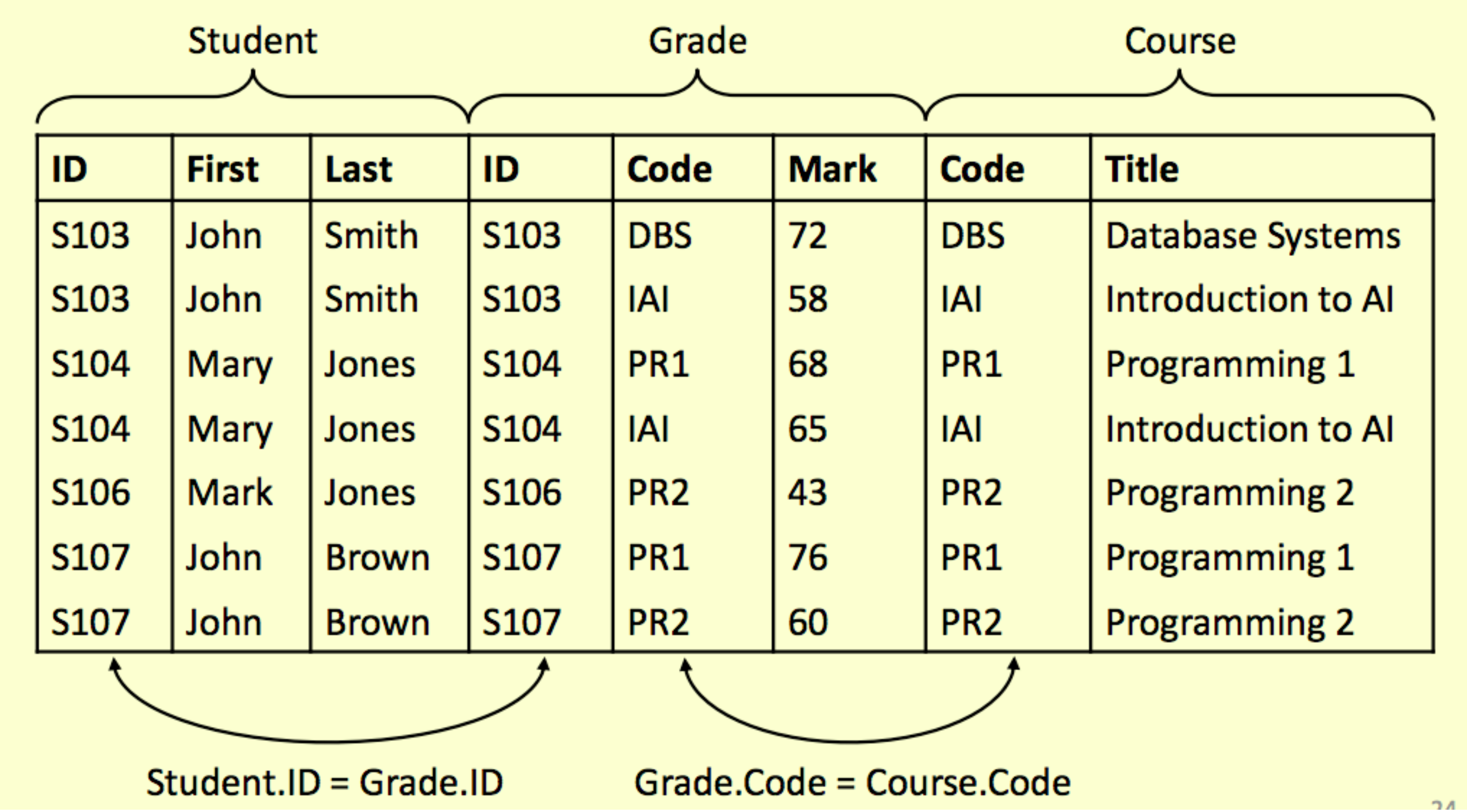


First	Last	Mark
John	Smith	72
John	Smith	58
Mary	Jones	68
Mary	Jones	65
Mark	Jones	43
John	Brown	76
John	Brown	60

SELECT from Multiple Tables

- WHERE clause is a key feature when selecting from multiple tables.
- Unrelated combinations can be filtered out.
- Another query example with 3 tables:

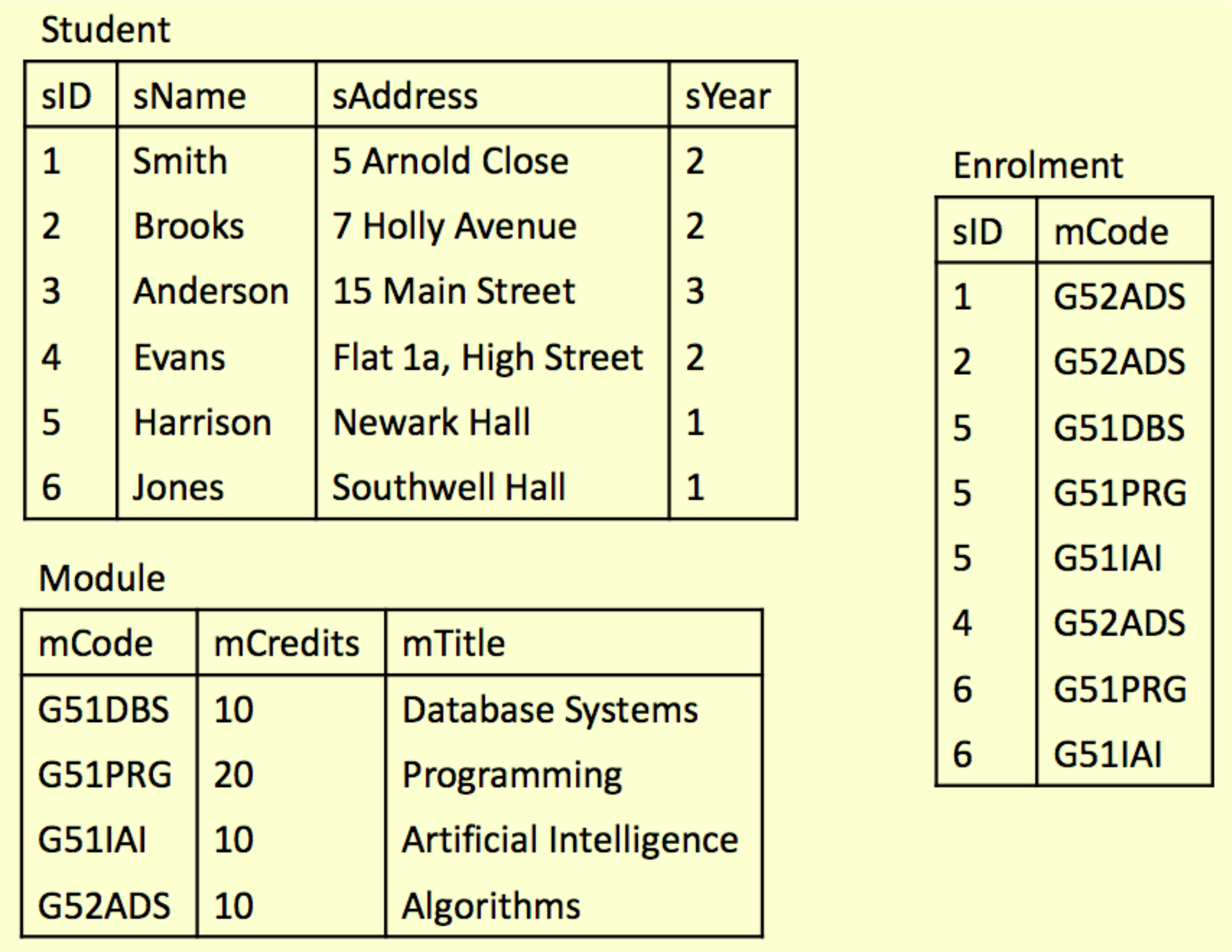
```
SELECT * FROM
   Student, Grade, Course
WHERE
   Student.ID = Grade.ID AND
   Course.Code = Grade.Code
```



Student			Grade			Course	
ID	**First**	**Last**	**ID**	**Code**	**Mark**	**Code**	**Title**
S103	John	Smith	S103	DBS	72	DBS	Database Systems
S103	John	Smith	S103	IAI	58	IAI	Introduction to AI
S104	Mary	Jones	S104	PR1	68	PR1	Programming 1
S104	Mary	Jones	S104	IAI	65	IAI	Introduction to AI
S106	Mark	Jones	S106	PR2	43	PR2	Programming 2
S107	John	Brown	S107	PR1	76	PR1	Programming 1
S107	John	Brown	S107	PR2	60	PR2	Programming 2

Student.ID = Grade.ID

Grade.Code = Course.Code



Student

sID	sName	sAddress	sYear
1	Smith	5 Arnold Close	2
2	Brooks	7 Holly Avenue	2
3	Anderson	15 Main Street	3
4	Evans	Flat 1a, High Street	2
5	Harrison	Newark Hall	1
6	Jones	Southwell Hall	1

Enrolment

sID	mCode
1	G52ADS
2	G52ADS
5	G51DBS
5	G51PRG
5	G51IAI
4	G52ADS
6	G51PRG
6	G51IAI

Module

mCode	mCredits	mTitle
G51DBS	10	Database Systems
G51PRG	20	Programming
G51IAI	10	Artificial Intelligence
G52ADS	10	Algorithms

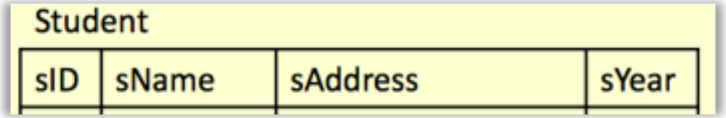
Try It Yourself

Write SQL statements to do the following:

- Produce a list of all student names and all their enrolments (module codes)

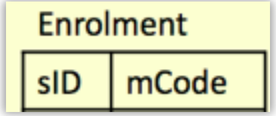
- Find a list of module titles being taken by the student named “Harrison”

- Find a list of module codes and titles for all modules currently being taken by first year students



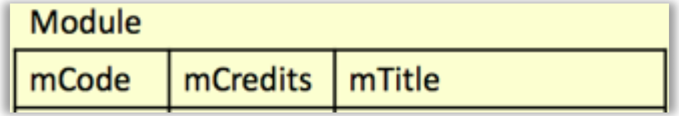
Student

sID	sName	sAddress	sYear



Enrolment

sID	mCode



Module

mCode	mCredits	mTitle

```
SELECT sName, mCode
  FROM Student, Enrolment
  WHERE Student.sID = Enrolment.sID;

SELECT mTitle
  FROM Module, Student, Enrolment
WHERE(Module.mCode = Enrolment.mCode)
   AND(Student.sID = Enrolment.sID)
   AND Student.sName = ‘Harrison’;

SELECT Module.mCode, mTitle
  FROM Enrolment, Module, Student
WHERE (Module.mCode = Enrolment.mCode)
   AND (Student.sID = Enrolment.sID)
   AND sYear = 1;
```

Aliases

- Aliases rename columns or tables
  - Can make names more meaningful
  - Can shorten names, making them easier to use
  - Can resolve ambiguous names

- Column alias

```
SELECT column [AS] new-col-name
```

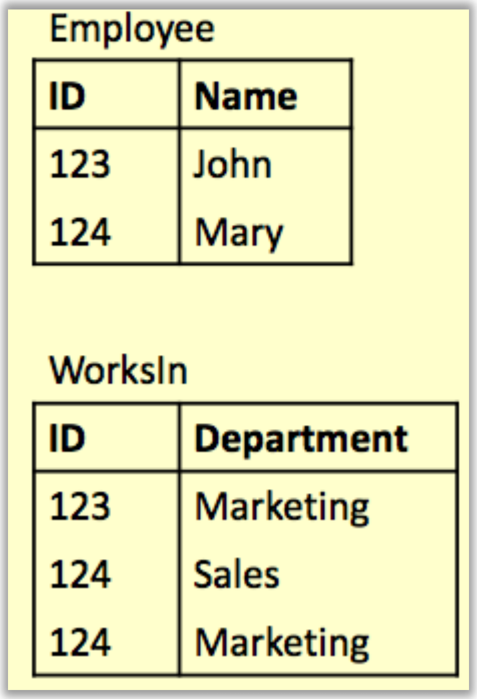
- Table alias

The AS keyword is optional

```
SELECT * FROM table [AS] new-table-name
```

Alias Example

```
SELECT
    E.ID AS empID,
    E.Name, W.Department
FROM
    Employee E,
    WorksIn W
WHERE
    E.ID = W.ID;
```

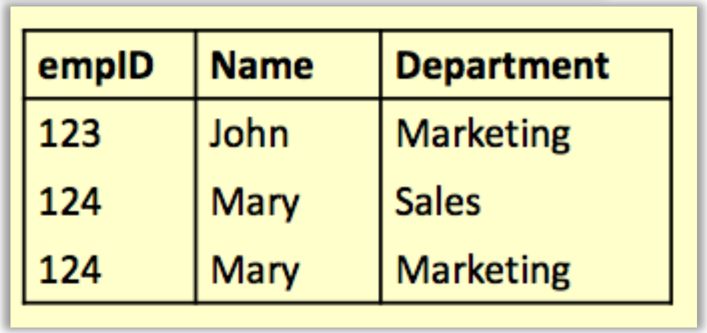


Employee

ID	Name
123	John
124	Mary

WorksIn

ID	Department
123	Marketing
124	Sales
124	Marketing



empID	Name	Department
123	John	Marketing
124	Mary	Sales
124	Mary	Marketing

Note: You cannot use a column alias in a WHERE clause:

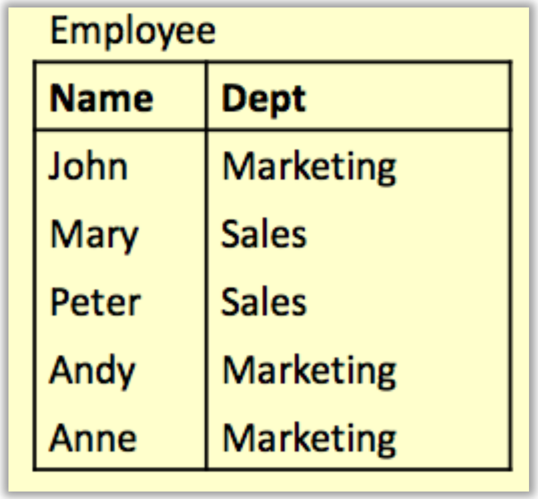
```
...WHERE E.ID AS empID = W.ID;
```

Wrong!

Aliases and ‘Self-Joins’

- Aliases can be used to copy a table, so that it can be combined with itself.
- The example below finds the names of all employees who work in the same department as Andy.

```
SELECT A.Name FROM
    Employee A,
    Employee B
WHERE
    A.Dept = B.Dept
AND
    B.Name = 'Andy';
```

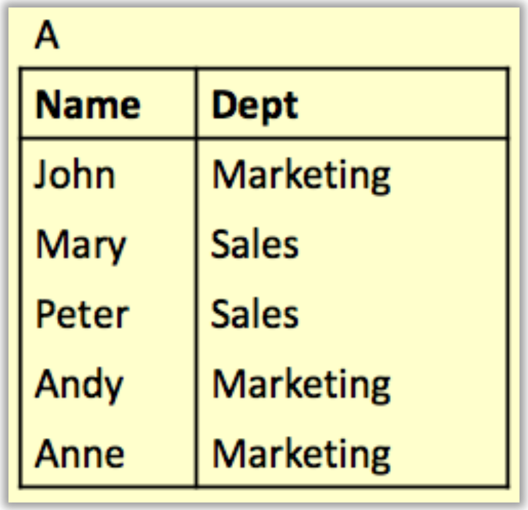


Employee

Name	Dept
John	Marketing
Mary	Sales
Peter	Sales
Andy	Marketing
Anne	Marketing

Aliases and 'Self-Joins'

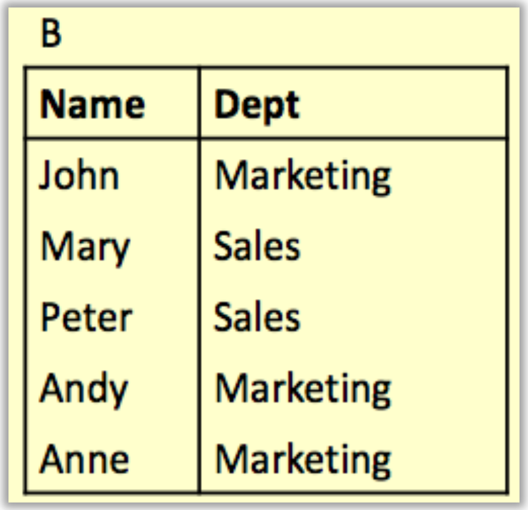
Employee A



A

Name	Dept
John	Marketing
Mary	Sales
Peter	Sales
Andy	Marketing
Anne	Marketing

Employee B

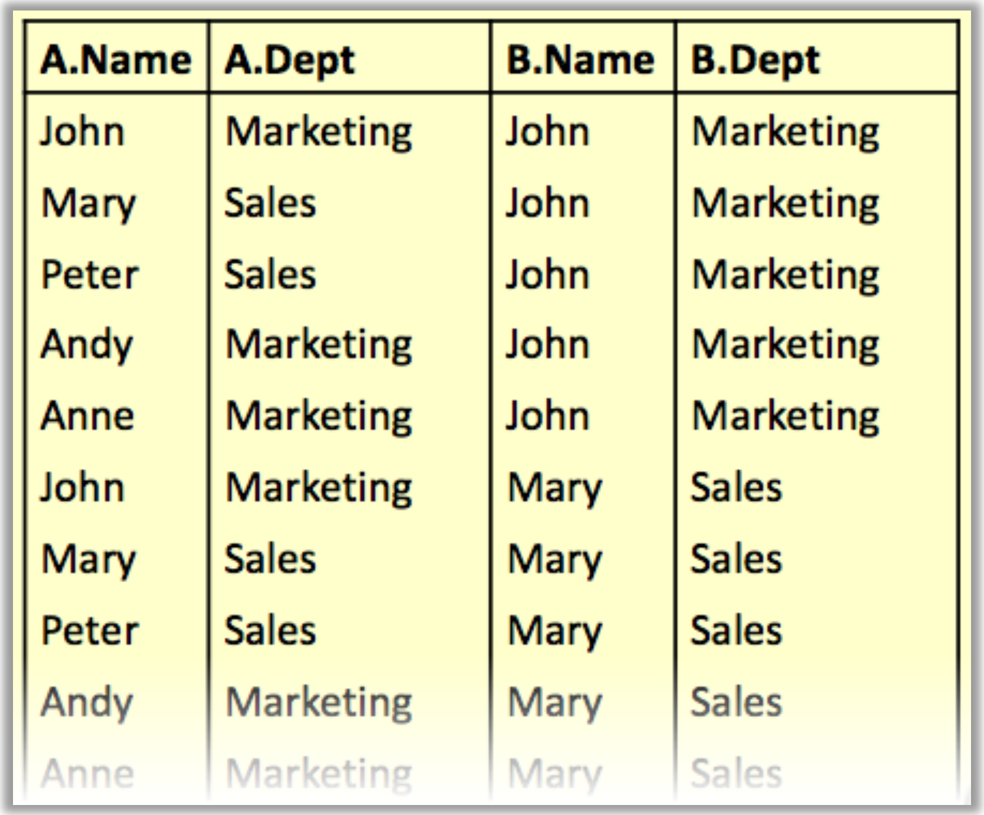


B

Name	Dept
John	Marketing
Mary	Sales
Peter	Sales
Andy	Marketing
Anne	Marketing

Aliases and ‘Self-Joins’

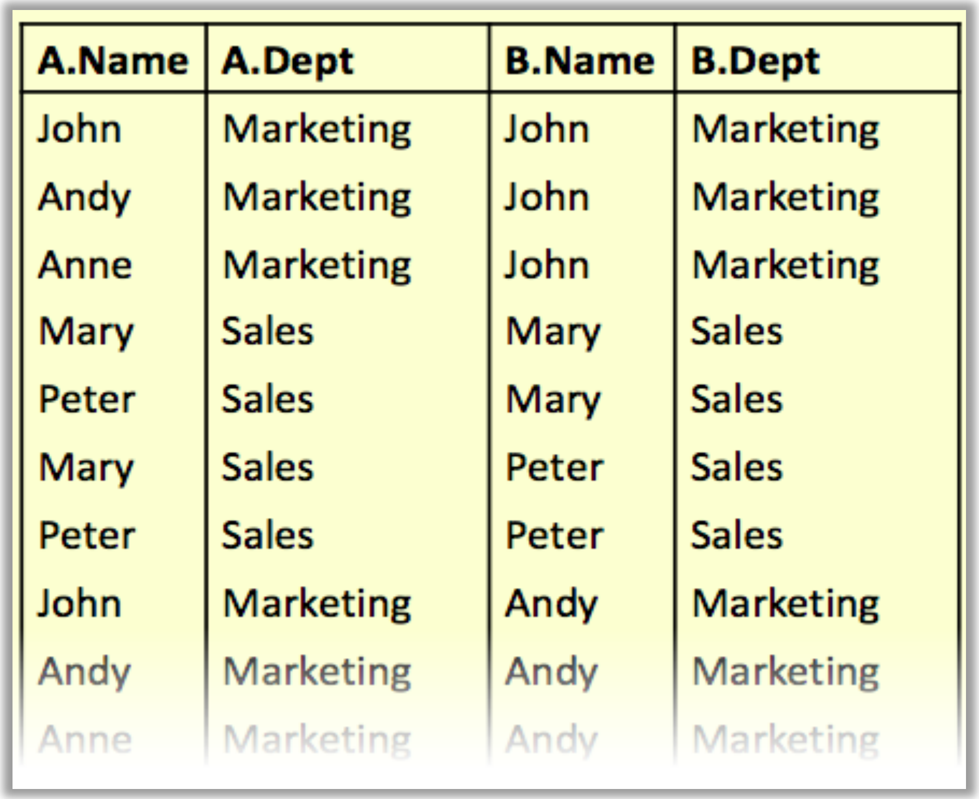
```
SELECT ... FROM Employee A, Employee B ...
```



A.Name	A.Dept	B.Name	B.Dept
John	Marketing	John	Marketing
Mary	Sales	John	Marketing
Peter	Sales	John	Marketing
Andy	Marketing	John	Marketing
Anne	Marketing	John	Marketing
John	Marketing	Mary	Sales
Mary	Sales	Mary	Sales
Peter	Sales	Mary	Sales
Andy	Marketing	Mary	Sales
Anne	Marketing	Mary	Sales

Aliases and ‘Self-Joins’

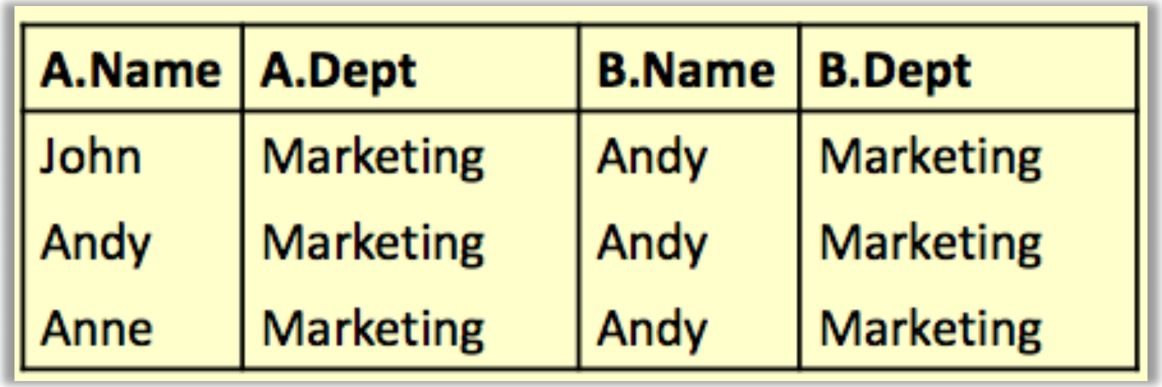
```
SELECT ... FROM Employee A, Employee B
      WHERE A.Dept = B.Dept ...
```



A.Name	A.Dept	B.Name	B.Dept
John	Marketing	John	Marketing
Andy	Marketing	John	Marketing
Anne	Marketing	John	Marketing
Mary	Sales	Mary	Sales
Peter	Sales	Mary	Sales
Mary	Sales	Peter	Sales
Peter	Sales	Peter	Sales
John	Marketing	Andy	Marketing
Andy	Marketing	Andy	Marketing
Anne	Marketing	Andy	Marketing

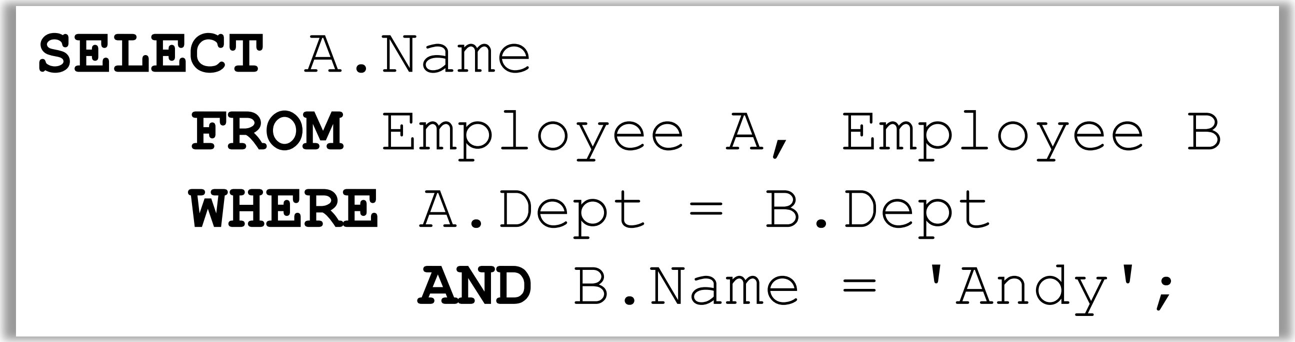
Aliases and 'Self-Joins'

```
SELECT ... FROM Employee A, Employee B
    WHERE A.Dept = B.Dept
    AND B.Name = 'Andy';
```



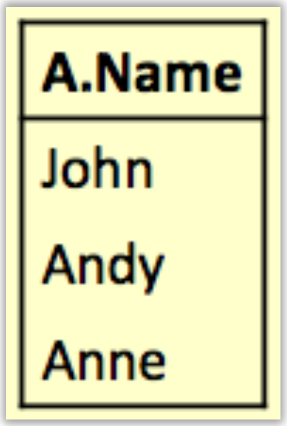
A.Name	A.Dept	B.Name	B.Dept
John	Marketing	Andy	Marketing
Andy	Marketing	Andy	Marketing
Anne	Marketing	Andy	Marketing

Aliases and ‘Self-Joins’



```
SELECT A.Name
    FROM Employee A, Employee B
    WHERE A.Dept = B.Dept
          AND B.Name = 'Andy';
```

Names of all employees who work in the same department as Andy.



A.Name
John
Andy
Anne

Subqueries

Handling sets returned by subqueries

Subqueries

- A SELECT statement can be nested inside another query to form a subquery
- The results of the subquery are passed back to the containing query
- For example, retrieve a list of names of people who are in Andy’s department:

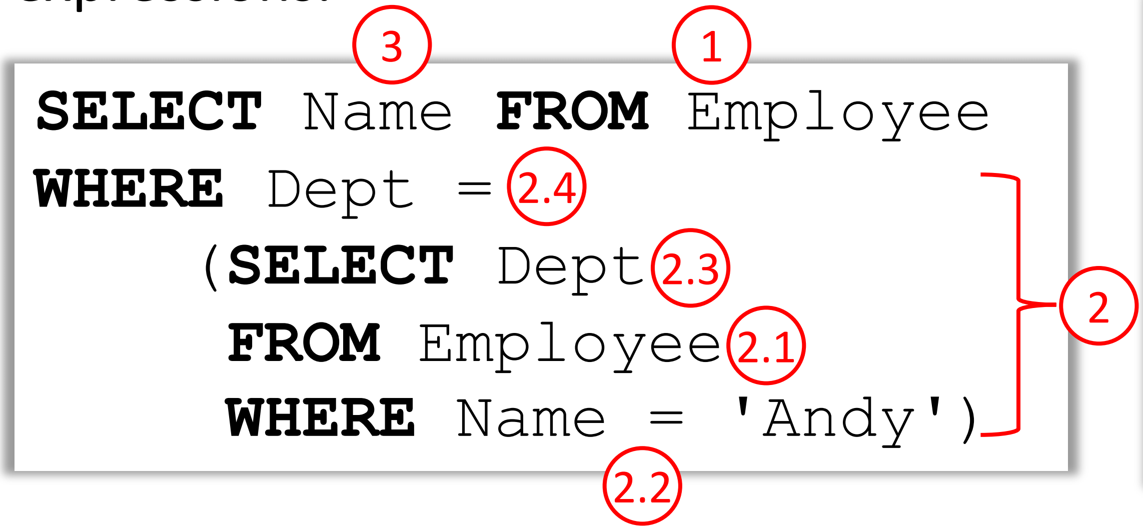
```
SELECT Name FROM Employee
    WHERE Dept =
        (SELECT Dept FROM Employee
          WHERE Name = 'Andy')
```

Subqueries

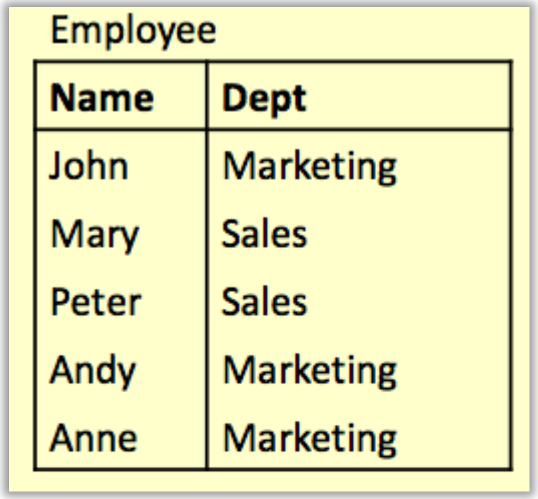
- The first `FROM` part is evaluated.
- For each row of Employee, we check whether `Dept` equals to the result of:

```
SELECT Dept FROM Employee WHERE Name = 'Andy')
```

- Then the columns will be filtered with the SELECT expressions.



```
SELECT Name FROM Employee
WHERE Dept =
      (SELECT Dept
       FROM Employee
       WHERE Name = 'Andy')
```



Employee

Name	Dept
John	Marketing
Mary	Sales
Peter	Sales
Andy	Marketing
Anne	Marketing

Subqueries and Aliases

- You can use a subquery between FROM and WHERE.
- But the result must be renamed:

```
SELECT * FROM
   (SELECT name, email FROM teachers) AS t
  WHERE t.email IS NOT NULL;
```

- This is because that the result of a subquery does not have a table name.

Subqueries

- Often a subquery will return a set of values rather than a single value
- We cannot directly compare a single value to a set. Doing so will result in an error
- Options for handling sets
  - `IN` : checks to see if a value is in a set
  - `EXISTS` : checks to see if a set is empty
  - `ALL/ANY` : checks to see if a relationship holds for every/one member of a set
  - `NOT` : can be used with any of the above 4

Handling sets: IN

- Using `IN` we can see if a given value is in a set of values

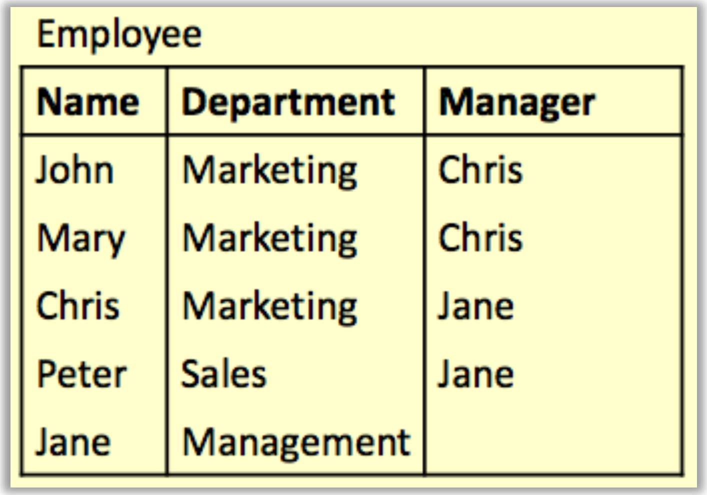
```
SELECT columns FROM tables
    WHERE col IN set;
```

- `NOT IN` checks to see if a given value is not in the set

```
SELECT columns FROM tables
    WHERE col NOT IN set;
```

- The set can be given explicitly or can be produced in a subquery

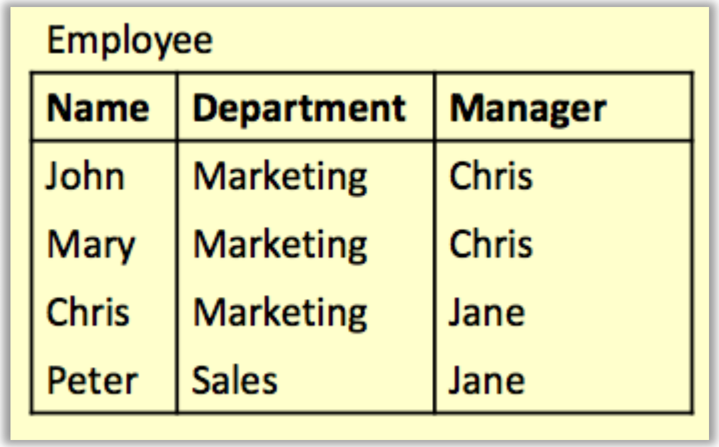
```
SELECT id FROM student
    WHERE id IN ('S103', 'S104');
```



Employee

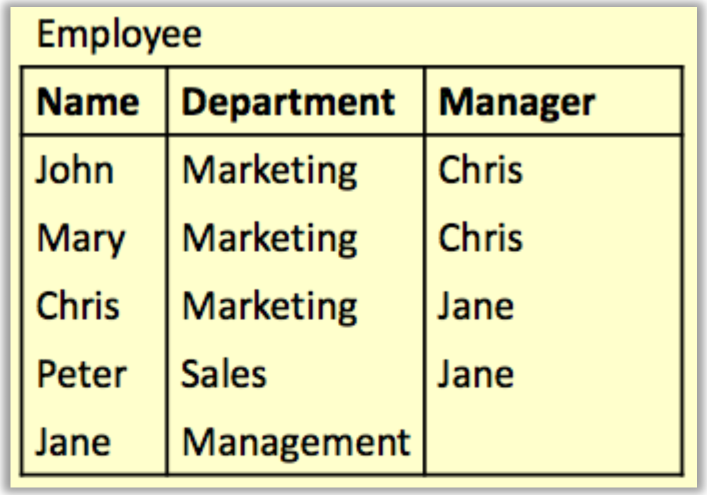
Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane
Jane	Management	

```
SELECT * FROM Employee
    WHERE Department IN ('Marketing', 'Sales');
```



Employee

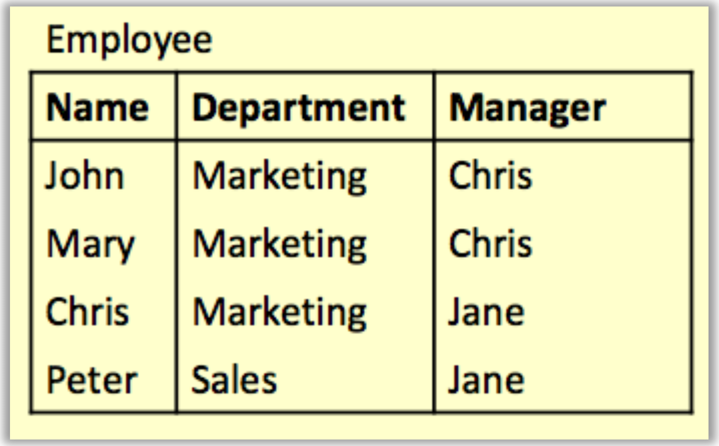
Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane



Employee

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane
Jane	Management	

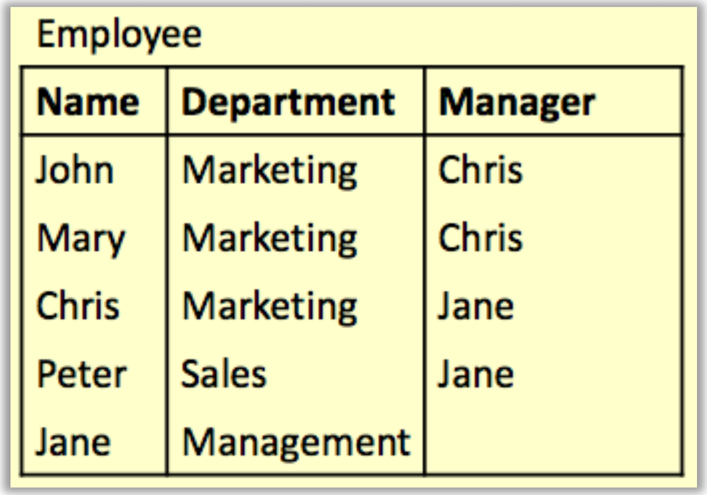
```
SELECT * FROM Employee
WHERE Department = 'Marketing'
   OR Department = 'Sales';
```



Employee

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane

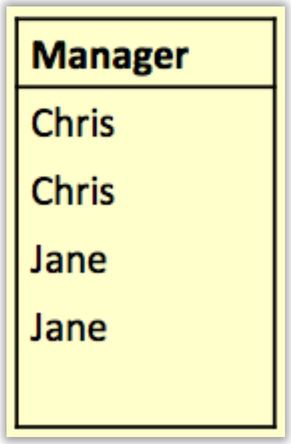
Handling sets: NOT IN



Employee

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane
Jane	Management	

```
SELECT * FROM Employee
WHERE Name NOT IN
       (SELECT Manager
        FROM Employee);
```



Manager
Chris
Chris
Jane
Jane

The query is equivalent to:

```
SELECT * FROM Employee
WHERE Name NOT IN
('Chris', 'Jane');
```

Handling sets: EXISTS

- Using `EXISTS` we can see whether there is at least one element in a given set.

```
SELECT columns
FROM tables
WHERE EXISTS set;
```

- `NOT EXISTS` is true if the set is empty

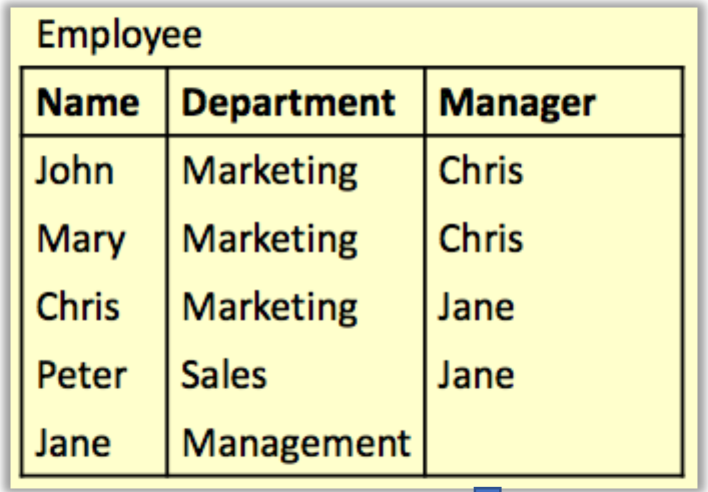
```
SELECT columns
FROM tables
WHERE NOT EXISTS set;
```

- The set is always given by a subquery

Handling sets: EXISTS

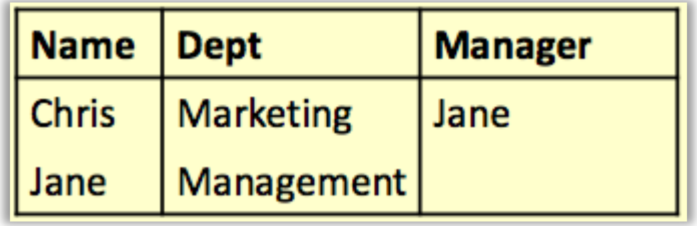
- Retrieve all the info for those employees who are also managers:

```
SELECT * FROM
  Employee AS E1
  WHERE EXISTS
    (SELECT * FROM
      Employee AS E2
      WHERE E1.Name = E2.Manager);
```



Employee

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane
Jane	Management	



Name	Dept	Manager
Chris	Marketing	Jane
Jane	Management	

Handling sets: ANY and ALL

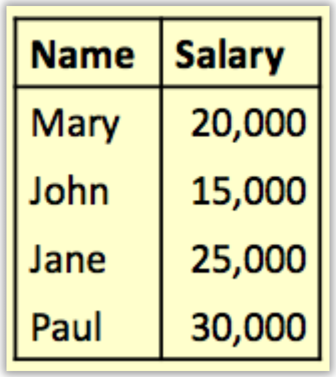
- ANY and ALL compare a single value to a set of values
- They are used with comparison operators like = , >, <, <>, >=, <=

- `val` **`= ANY (set)`**
  - is true if there is at least one member of the set equal to value
- `val` **`= ALL (set)`**
  - is true if all members of the set are equal to the value

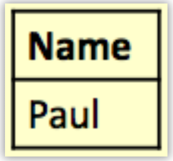
Handling sets: ALL

- Find the name(s) of the employee(s) who earn the highest salary
- Employee:

```
SELECT Name
    FROM Employee
    WHERE Salary >=
    ALL (
        SELECT Salary
        FROM Employee);
```



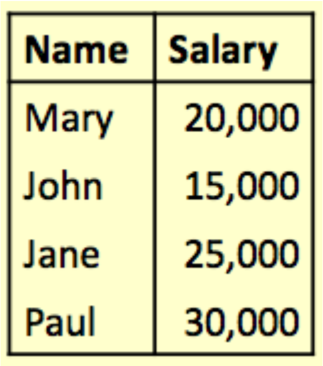
Name	Salary
Mary	20,000
John	15,000
Jane	25,000
Paul	30,000



Name
Paul

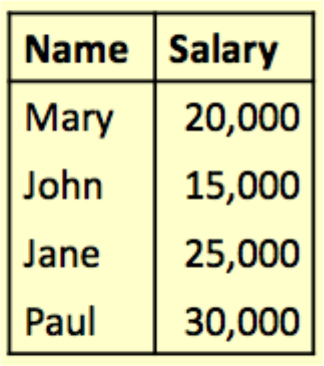
Handling sets: ANY

- Find the name(s) of the employee(s) who earn more than someone else

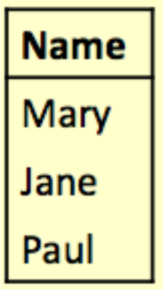


Name	Salary
Mary	20,000
John	15,000
Jane	25,000
Paul	30,000

Handling sets: ANY



Name	Salary
Mary	20,000
John	15,000
Jane	25,000
Paul	30,000



Name
Mary
Jane
Paul

- Find the name(s) of the employee(s) who earn more than someone else

```
SELECT Name
   FROM Employee
   WHERE Salary >
      ANY (
      SELECT Salary
      FROM Employee);
```