CS 217 Data Management and Information Processing

GROUP BY and INNER JOINs

SQL Group By

GROUP BY

- GROUP BY combines multiple rows into one row in the result.
  - Rows with the same value for the *grouping criterion* are grouped.
  - An aggregation function will be applied within each group.

```
SELECT CategoryID, COUNT(*) AS category_count,
       MAX(RetailPrice) AS most_expensive_price
FROM Products GROUP BY CategoryID;
```

ProductName	RetailPrice	CategoryID
Trek 9000 Mountain Bike	1200	2
Eagle FS-3 Mountain Bike	1800	2
Dog Ear Cyclecomputer	75	1
Victoria Pro All Weather Tires	54.95	4
Dog Ear Helmet Mount Mirrors	7.45	1
Viscount Mountain Bike	635	2
Viscount C-500 Wireless Bike Computer	49	1

CategoryID	category_count	most_expensive
1	18	179
2	4	1800
3	4	85
4	9	189
5	2	180
6	3	34

The GROUP BY expression

"GROUP BY **x**" means:

- Each row in the output will represent many aggregated rows having the same value for **x**.
- Thus, the number of rows in the result is the number of distinct values taken by **x** (after the WHERE filtering).
- Usually it's just the name of a column, but it can be an arbitrary expression.

SELECT category, AVG(price) FROM product **GROUP BY category;**

Table “product”

id	name	price	category
1	Quart Skim Milk	2.49	3
2	Rye Bread	1.99	1
3	1lb Butter	5.99	3
4	32oz Yogurt	4.99	3
5	Navel Orange (each)	0.89	2
6	Pineapple (each)	1.99	2
7	English Muffins	3.99	1
8	Spinach (bunch)	1.49	2
9	Carrots (lb bag)	0.99	2
10	Dozen Eggs	2.49	3

Output

category	AVG(price)
1	2.99
2	1.34
3	3.99

This is a typical GROUP BY example.

SELECT price, COUNT(*)
FROM product **GROUP BY price** ORDER BY price;

Table “product”

id	name	price	category
1	Quart Skim Milk	2.49	3
2	Rye Bread	1.99	1
3	1lb Butter	5.99	3
4	32oz Yogurt	4.99	3
5	Navel Orange (each)	0.89	2
6	Pineapple (each)	1.99	2
7	English Muffins	3.99	1
8	Spinach (bunch)	1.49	2
9	Carrots (lb bag)	0.99	2
10	Dozen Eggs	2.49	3

Output

price	COUNT(*)
0.89	1
0.99	1
1.49	1
1.99	2
2.49	2
3.99	1
4.99	1
5.99	1

This is a typical GROUP BY example.

SELECT category, price
FROM product **GROUP BY category;**

Table “product”

id	name	price	category
1	Quart Skim Milk	**2.49**	3
2	Rye Bread	**1.99**	1
3	1lb Butter	5.99	3
4	32oz Yogurt	4.99	3
5	Navel Orange (each)	**0.89**	2
6	Pineapple (each)	1.99	2
7	English Muffins	3.99	1
8	Spinach (bunch)	1.49	2
9	Carrots (lb bag)	0.99	2
10	Dozen Eggs	2.49	3

Output

category	price
1	1.99
2	0.89
3	2.49

Uncommon use of GROUP BY

It’s missing an aggregation function (like SUM, MIN, etc.).
It prints a random price for each category.

SELECT id, name FROM product **GROUP BY id;**

Table “product”

id	name	price	category
1	Quart Skim Milk	2.49	3
2	Rye Bread	1.99	1
3	1lb Butter	5.99	3
4	32oz Yogurt	4.99	3
5	Navel Orange (each)	0.89	2
6	Pineapple (each)	1.99	2
7	English Muffins	3.99	1
8	Spinach (bunch)	1.49	2
9	Carrots (lb bag)	0.99	2
10	Dozen Eggs	2.49	3

Output

id	name
1	Quart Skim Milk
2	Rye Bread
3	1lb Butter
4	32oz Yogurt
5	Navel Orange (each)
6	Pineapple (each)
7	English Muffins
8	Spinach (bunch)
9	Carrots (lb bag)
10	Dozen Eggs

This GROUP BY is not very useful because **id** is always different.

SELECT AVG(price)
FROM product **GROUP BY "hello";**

Table "product"

id	name	price	category
1	Quart Skim Milk	2.49	3
2	Rye Bread	1.99	1
3	1lb Butter	5.99	3
4	32oz Yogurt	4.99	3
5	Navel Orange (each)	0.89	2
6	Pineapple (each)	1.99	2
7	English Muffins	3.99	1
8	Spinach (bunch)	1.49	2
9	Carrots (lb bag)	0.99	2
10	Dozen Eggs	2.49	3

Output

AVG(price)
2.73

Uncommon use of GROUP BY

"hello" is the same for every row, so it always aggregates all rows to one output row.

AVG would have given the same result without any GROUP BY.

SELECT category=2, AVG(price) FROM product **GROUP BY category=2;**

Table “product”

id	name	price	category
1	Quart Skim Milk	2.49	3
2	Rye Bread	1.99	1
3	1lb Butter	5.99	3
4	32oz Yogurt	4.99	3
5	Navel Orange (each)	0.89	2
6	Pineapple (each)	1.99	2
7	English Muffins	3.99	1
8	Spinach (bunch)	1.49	2
9	Carrots (lb bag)	0.99	2
10	Dozen Eggs	2.49	3

Output

category=2	AVG(price)
0 *(false)*	3.6566667
1 *(true)*	1.34

This is an advanced GROUP BY example.

It divides the rows into two groups, those with category=2 in one group and everything else in the other group.

Prints the average price of fruits & vegetables vs the average price of other foods.

SELECT price, COUNT(*) FROM product GROUP BY price **HAVING COUNT(*)>1;**

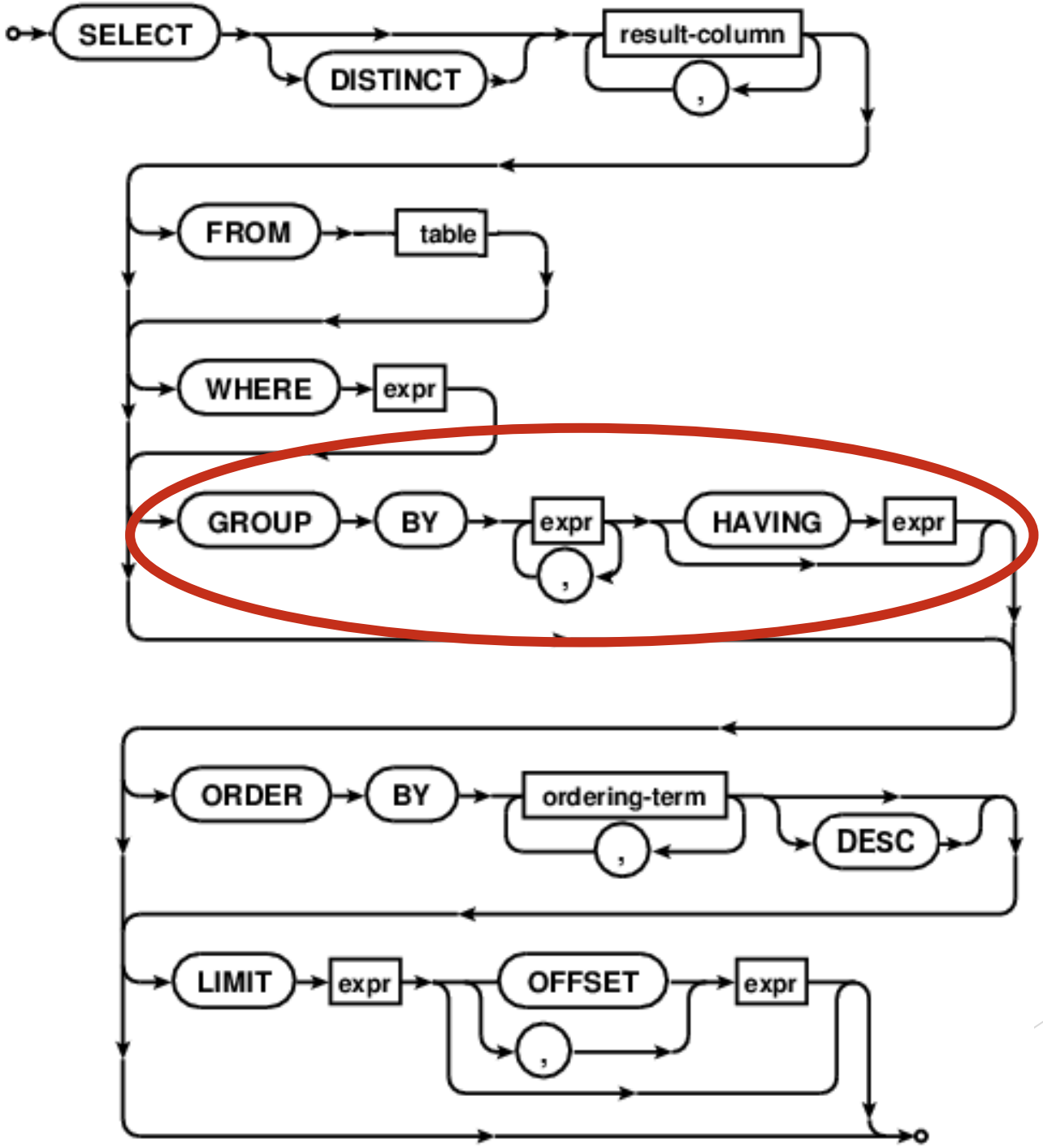
Table “product”

id	name	price	category
1	Quart Skim Milk	2.49	3
2	Rye Bread	1.99	1
3	1lb Butter	5.99	3
4	32oz Yogurt	4.99	3
5	Navel Orange (each)	0.89	2
6	Pineapple (each)	1.99	2
7	English Muffins	3.99	1
8	Spinach (bunch)	1.49	2
9	Carrots (lb bag)	0.99	2
10	Dozen Eggs	2.49	3

Output

price	COUNT(*)
1.99	2
2.49	2

HAVING filters records that work on summarized GROUP BY results.


SELECT
DISTINCT
result-column
,
FROM
table
WHERE
expr
GROUP
BY
expr
,
HAVING
expr
ORDER
BY
ordering-term
,
DESC
LIMIT
expr
OFFSET
,
expr

SQL JOINs

What if you need to combine data from multiple tables?

1. `FROM` chooses the table of interest
2. `WHERE` throws out irrelevant rows
3. `GROUP BY` identifies rows to combine
4. `SELECT` tells what values to return (allowing math and aggregation)
5. `HAVING` throws out irrelevant rows (after aggregation)
6. `ORDER BY` sorts
7. `LIMIT` throws out rows based on their position in the results

A subquery can draw data from another table, but JOINs are a more powerful way to use multiple tables.

List of SQL JOINs

- **INNER JOIN (default)**
- **NATURAL JOIN**
- LEFT JOIN
- RIGHT JOIN
- CROSS JOIN

Recall: Normalization

- Normalization split the staff directory to three tables
- It eliminates redundant information, but now we have to look in three different tables to answer some questions.

staff					
id	***name***	***department***	***building***	***room***	***faxNumber***
11	Bob	Industrial Eng.	Tech	100	1-1000
20	Betsy	Computer Sci.	Ford	100	1-5003
21	Fran	Industrial Eng.	Tech	101	1-1000
22	Frank	Chemistry	Tech	102	1-1000
35	Sarah	Physics	Mudd	200	1-2005
40	Sam	Materials Sci.	Cook	10	1-3004
54	Pat	Computer Sci.	Ford	102	1-5003

staff			
id	***name***	***room***	***departmentId***
11	Bob	100	1
20	Betsy	100	2
21	Fran	101	1
22	Frank	102	4
35	Sarah	200	5
40	Sam	10	7
54	Pat	102	2

department		
id	***name***	***buildingId***
1	Industrial Eng.	1
2	Computer Sci.	2
4	Chemistry	1
5	Physics	4
7	Materials Sci.	5

building		
id	***name***	***faxNumber***
1	Tech	1-1000
2	Ford	1-5003
4	Mudd	1-2005
5	Cook	1-3004
6	Garage	1-6001

What if we want to print the staff directory?

staff					
id	***name***	***department***	***building***	***room***	***faxNumber***
11	Bob	Industrial Eng.	Tech	100	1-1000
20	Betsy	Computer Sci.	Ford	100	1-5003
21	Fran	Industrial Eng.	Tech	101	1-1000
22	Frank	Chemistry	Tech	102	1-1000
35	Sarah	Physics	Mudd	200	1-2005
40	Sam	Materials Sci.	Cook	10	1-3004
54	Pat	Computer Sci.	Ford	102	1-5003

We can generate a temporary table like this with (INNER) JOIN

staff			
id	***name***	***room***	***departmentId***
11	Bob	100	1
20	Betsy	100	2
21	Fran	101	1
22	Frank	102	4
35	Sarah	200	5
40	Sam	10	7
54	Pat	102	2

department		
id	***name***	***buildingId***
1	Industrial Eng.	1
2	Computer Sci.	2
4	Chemistry	1
5	Physics	4
7	Materials Sci.	5

ON tells how rows are matched

```
SELECT * FROM staff JOIN department ON staff.departmentId=department.id;
```

staff.***id***	staff.***name***	***staff.room***	staff.***departmentId***	department.***id***	department.***name***	department.***buildingId***
11	Bob	100	1	1	Industrial Eng.	1
20	Betsy	100	2	2	Computer Sci.	2
21	Fran	101	1	1	Industrial Eng.	1
22	Frank	102	4	4	Chemistry	1
35	Sarah	200	5	5	Physics	4
40	Sam	10	7	7	Materials Sci.	5
54	Pat	102	2	2	Computer Sci.	2

INNER JOIN

student	
id	***name***
123	Bob
124	Mary
125	Jane
126	Frank

enrollment	
id	***course***
123	DBS
124	PRG
124	DBS
125	PRG

```
SELECT * FROM student
JOIN enrollment
ON student.id=enrollment.id;
```

id	***name***	***id***	***course***
123	Bob	123	DBS
124	Mary	124	PRG
124	Mary	124	DBS
125	Jane	125	PRG

NATURAL JOIN

- A shorthand notation to make some JOINs shorter to express.
- NATURAL JOIN matches rows using whatever columns have identical names.

For example:

```
SELECT * FROM Orders JOIN Order_Details
ON Orders.OrderNumber=Order_Details.OrderNumber;
```

Very similar to:

```
SELECT * FROM Orders NATURAL JOIN
Order_Details;
```

NATURAL JOIN

student	
id	***name***
123	Bob
124	Mary
125	Jane
126	Frank

enrollment	
id	***course***
123	DBS
124	PRG
124	DBS
125	PRG

```
SELECT * FROM student
NATURAL JOIN enrollment;
```

id	***name***	***course***
123	Bob	DBS
124	Mary	PRG
124	Mary	DBS
125	Jane	PRG

Only one id column!

Trick 1: Just get the columns we need

```
SELECT staff.id, staff.name, staff.room,
       department.name, department.buildingId
FROM staff
JOIN department ON staff.departmentId=department.id;
```

staff.*id*	staff.*name*	staff.*room*	department.*name*	department.*buildingId*
11	Bob	100	Industrial Eng.	1
20	Betsy	100	Computer Sci.	2
21	Fran	101	Industrial Eng.	1
22	Frank	102	Chemistry	1
35	Sarah	200	Physics	4
40	Sam	10	Materials Sci.	5
54	Pat	102	Computer Sci.	2

Trick 2: Reorder and rename the columns

```
SELECT staff.id AS staffID, staff.name AS name,
       department.name AS department,
       department.buildingId AS buildingId,
       staff.room AS room
FROM staff
JOIN department ON staff.departmentId=department.id;
```

staffId	name	department	buildingId	room
11	Bob	Industrial Eng.	1	100
20	Betsy	Computer Sci.	2	100
21	Fran	Industrial Eng.	1	101
22	Frank	Chemistry	1	102
35	Sarah	Physics	4	200
40	Sam	Materials Sci.	5	10
54	Pat	Computer Sci.	2	102

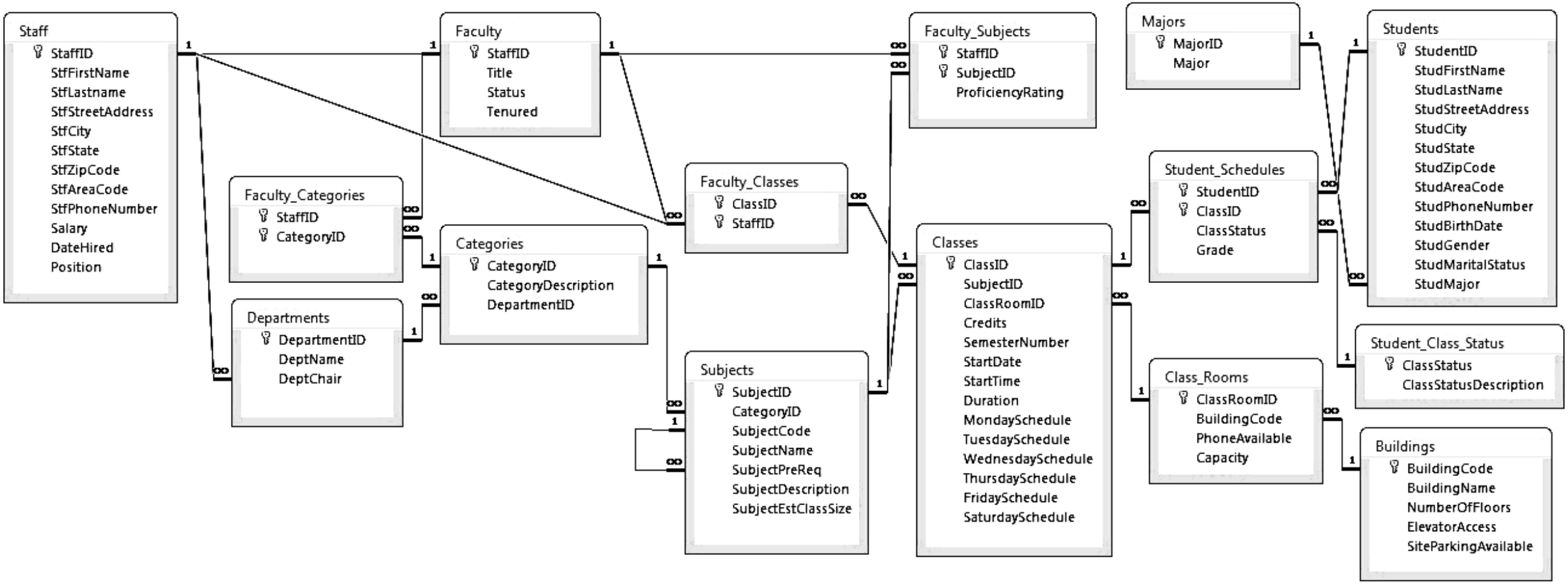
Trick 3: JOIN to the third table

```
SELECT staff.id AS staffId, staff,name, department.name AS department,
       building.name AS building, staff.room AS room,
       building.faxNumber AS faxNumber
FROM staff
JOIN department ON staff.departmentId=department.id
JOIN building ON department.buildingId=building.id;
```

staffId	*name*	*department*	*building*	*room*	*faxNumber*
11	Bob	Industrial Eng.	Tech	100	1-1000
20	Betsy	Computer Sci.	Ford	100	1-5003
21	Fran	Industrial Eng.	Tech	101	1-1000
22	Frank	Chemistry	Tech	102	1-1000
35	Sarah	Physics	Mudd	200	1-2005
40	Sam	Materials Sci.	Cook	10	1-3004
54	Pat	Computer Sci.	Ford	102	1-5003

Examples

SchoolScheduling.sqlite



SchoolScheduling.sqlite

- What is the average classroom capacity? Maximum?
- How much classroom capacity is there in each building?
- How many classes does each instructor teach on average?
- What is the average grade earned by students?

What is the average classroom capacity? Maximum?

How much classroom capacity is there in each building?

How many classes does each instructor teach on average?

What is the average grade earned by students?

SchoolScheduling.sqlite (answers)

- What is the average classroom capacity? Maximum?
  - `SELECT AVG(Capacity) FROM Class_Rooms;`
  - `SELECT MAX(Capacity) FROM Class_Rooms;`
- How much classroom capacity is there in each building?
  - `SELECT BuildingCode, SUM(Capacity) FROM Class_Rooms GROUP BY BuildingCode;`
- How many classes does each instructor teach on average?
  - `SELECT AVG(NumClasses) FROM (SELECT COUNT(*) AS NumClasses FROM Faculty_Classes GROUP BY StaffID);`
- What is the average grade earned by students?
  - `SELECT avg(Grade) FROM Student_Schedules WHERE ClassStatus = (SELECT ClassStatus FROM Student_Class_Status WHERE ClassStatusDescription="Completed");`

Who teaches the largest class & what is the average grade?

- Class enrollments are in **Student_Schedules** table
- Instructor→class assignments are in **Faculty_Classes** table
- Instructor names are in **Staff** table.
- Can use two subqueries to answer the first part of the question:
  - Get the largest class:

```
SELECT ClassID FROM Student_Schedules GROUP BY ClassID ORDER BY COUNT(*) DESC LIMIT 1;
```

  - Get the instructor ID of that class:

```
SELECT StaffID FROM Faculty_Classes WHERE ClassID=…
```

  - Get the instructor name:

```
SELECT StfFirstName, StfLastName FROM Staff WHERE StaffID=…
```

```
SELECT StfFirstName, StfLastName FROM Staff
WHERE StaffID=
(SELECT StaffID FROM Faculty_Classes WHERE ClassID=
  (SELECT ClassID FROM Student_Schedules
   GROUP BY ClassID ORDER BY COUNT(*) DESC LIMIT 1));
```

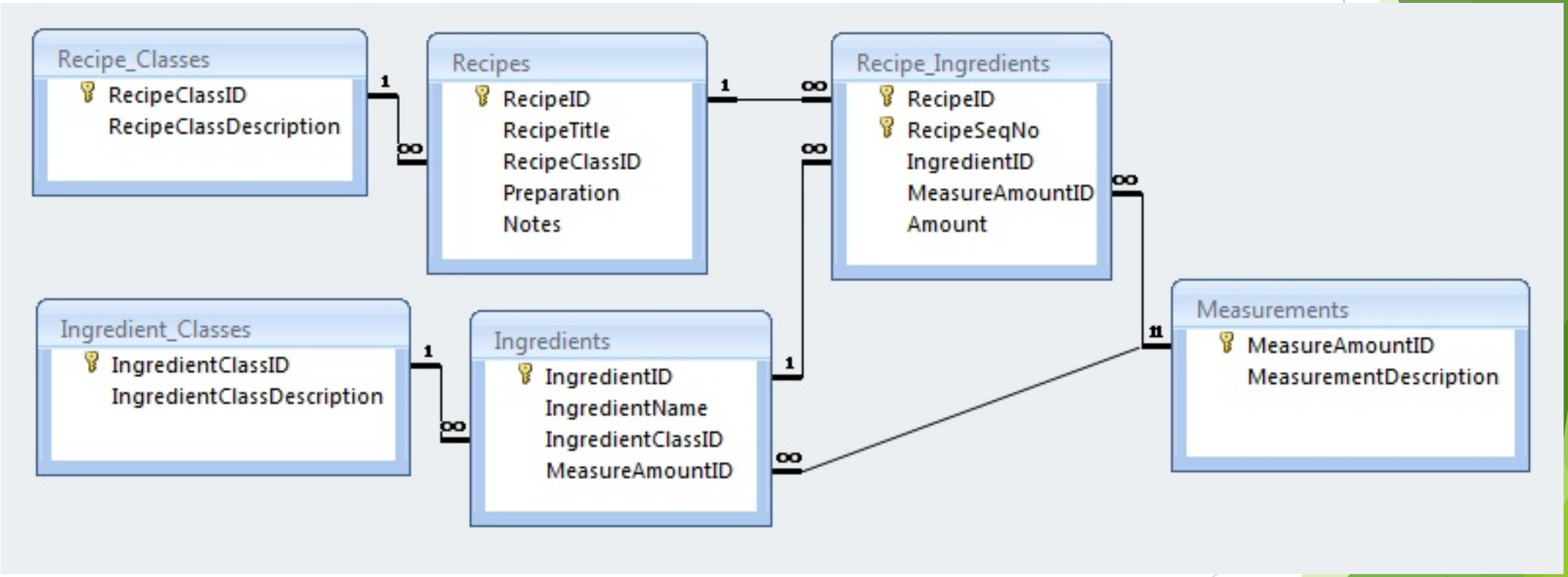
Who teaches the largest class & what is the average grade?

- *Alternative approach*:
  Use JOINs to create a composite table listing instructors, classes, and their average grades:

```
SELECT Student_Schedules.ClassID, StfLastname, AVG(Grade)
FROM Student_Schedules
JOIN Faculty_Classes ON
      Student_Schedules.ClassID=Faculty_Classes.ClassID
JOIN Staff ON
      Faculty_Classes.StaffID = Staff.StaffID
GROUP BY Student_Schedules.Clas
ORDER BY COUNT(*) DESC LIMIT 1;
```

General procedure: First get all information in one table using JOIN, then query on the JOIN.

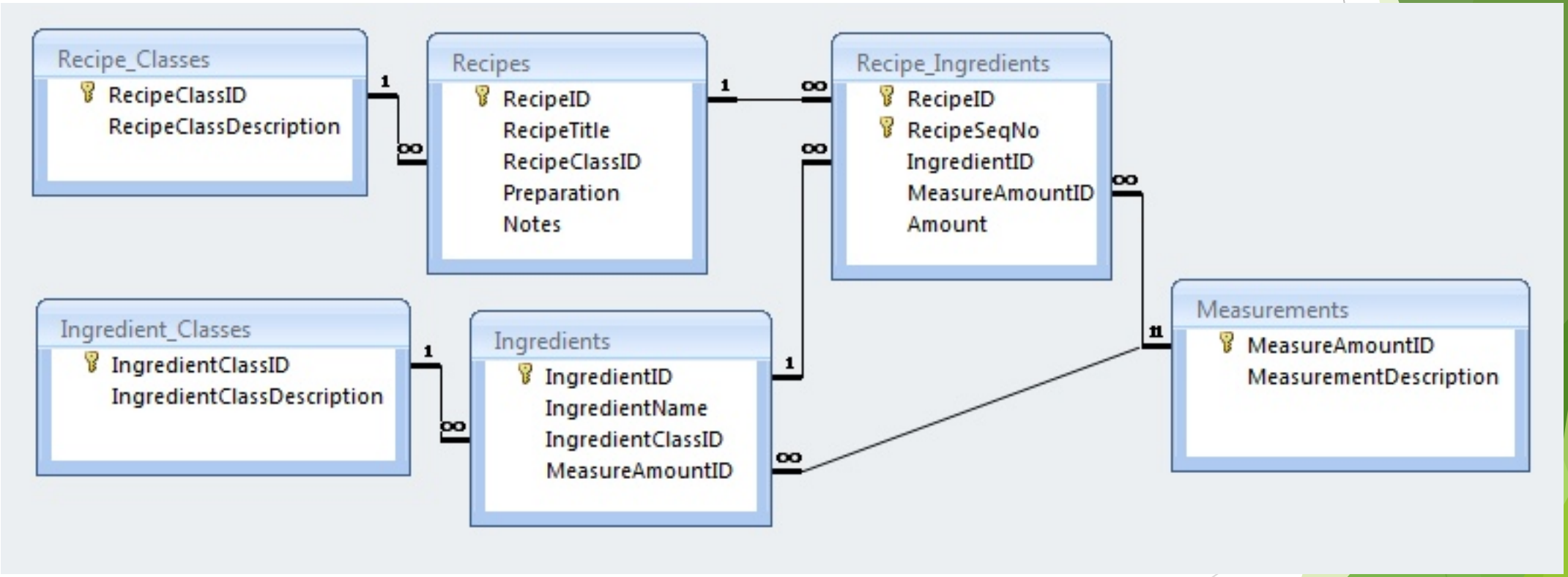
(Recipes.sqlite) Print the recipe for Irish Stew



(Recipes.sqlite) Print the recipe for Irish Stew

```
SELECT RecipeSeqNo, Amount,
MeasurementDescription, IngredientName

FROM Recipes JOIN Recipe_Ingredients

    ON Recipes.RecipeID = Recipe_Ingredients.RecipeID

JOIN Ingredients
    ON Recipe_Ingredients.IngredientId
       = Ingredients.IngredientID
JOIN Measurements
    ON Recipe_Ingredients.MeasureAmountID
       = Measurements.MeasureAmountID

WHERE RecipeTitle = "Irish Stew"

ORDER BY RecipeSeqNo;
```

What is the name of the recipe with the most ingredients?
(Can be done with either a subquery or a JOIN)



What is the name of the recipe with the most ingredients?

```
SELECT RecipeTitle,
       COUNT(*) AS numIngredients
FROM Recipe_Ingredients JOIN Recipes
ON Recipes.RecipeID = Recipe_Ingredients.RecipeID
GROUP BY Recipes.RecipeID
ORDER BY numIngredients DESC
LIMIT 1;
```