Relational Databases - Comp2400 / Comp6240

Lecture 2: SQL and the Relational Data Model

A quick overview of the relational data model and SQL

- Data in tables
- SQL basics
- Relational Algebra basics
- Keys and Joins

This material is covered in Chapter 5 of Elmasri and Navathe.

News and Administration

See `http://cs.anu.edu.au/student/comp2400`.

- Please be clear about what you are registering in: there are two groups for Mondays at 5pm:

News and Administration

See `http://cs.anu.edu.au/student/comp2400`.

- Please be clear about what you are registering in: there are two groups for Mondays at 5pm:
  - a normal lab session group, which meets for the first time on 4 August

News and Administration

See `http://cs.anu.edu.au/student/comp2400`.

- Please be clear about what you are registering in: there are two groups for Mondays at 5pm:
  - a normal lab session group, which meets for the first time on 4 August
  - a one-time-only computer skills session which meets on 29 July

News and Administration

See `http://cs.anu.edu.au/student/comp2400`.

- Please be clear about what you are registering in: there are two groups for Mondays at 5pm:
  - a normal lab session group, which meets for the first time on 4 August
  - a one-time-only computer skills session which meets on 29 July
    - that's Monday, before our next lecture

News and Administration

See `http://cs.anu.edu.au/student/comp2400`.

- Please be clear about what you are registering in: there are two groups for Mondays at 5pm:
  - a normal lab session group, which meets for the first time on 4 August
  - a one-time-only computer skills session which meets on 29 July
    - that's Monday, before our next lecture
    - if registration fills up, I will add another session on Wednesday morning

News and Administration

See `http://cs.anu.edu.au/student/comp2400`.

- Please be clear about what you are registering in: there are two groups for Mondays at 5pm:
  - a normal lab session group, which meets for the first time on 4 August
  - a one-time-only computer skills session which meets on 29 July
    - that's Monday, before our next lecture
    - if registration fills up, I will add another session on Wednesday morning
- Some students can not get into a group they can attend. Please consider changing groups if you are in a group that has 20 people already. Here are some possible incentives:

News and Administration

See `http://cs.anu.edu.au/student/comp2400`.

- Please be clear about what you are registering in: there are two groups for Mondays at 5pm:
  - a normal lab session group, which meets for the first time on 4 August
  - a one-time-only computer skills session which meets on 29 July
    - that's Monday, before our next lecture
    - if registration fills up, I will add another session on Wednesday morning
- Some students can not get into a group they can attend. Please consider changing groups if you are in a group that has 20 people already. Here are some possible incentives:
  - in a smaller group, you get more of the tutors time

News and Administration

See `http://cs.anu.edu.au/student/comp2400`.

- Please be clear about what you are registering in: there are two groups for Mondays at 5pm:
  - a normal lab session group, which meets for the first time on 4 August
  - a one-time-only computer skills session which meets on 29 July
    - that's Monday, before our next lecture
    - if registration fills up, I will add another session on Wednesday morning
- Some students can not get into a group they can attend. Please consider changing groups if you are in a group that has 20 people already. Here are some possible incentives:
  - in a smaller group, you get more of the tutors time
  - Greg will *probably* be taking the Wednesday and Thursday groups
    (some people prefer to be in the lecturer's tute group)

Relations and Relation Schemas

A relation is *like* a table, with column headings. [E&N §5.1]

Relations and Relation Schemas

A relation is *like* a table, with column headings. [E&N §5.1]

employee

name	**department**
Abbott	Marketing
Burns	Executive
Costello	Accounts
Downer	Accounts

Relations and Relation Schemas

A relation is *like* a table, with column headings. [E&N §5.1]

employee

name	**department**
Abbott	Marketing
Burns	Executive
Costello	Accounts
Downer	Accounts

- as our company changes, the table can change, but the column headings remain the same

Relations and Relation Schemas

A relation is *like* a table, with column headings. [E&N §5.1]

employee

name	**department**
Abbott	Marketing
Burns	Executive
Costello	Accounts
Downer	Accounts

- as our company changes, the table can change, but the column headings remain the same
- the column headings correspond to a *relation schema*

Relations and Relation Schemas

A relation is *like* a table, with column headings. [E&N §5.1]

employee

name	**department**
Abbott	Marketing
Burns	Executive
Costello	Accounts
Downer	Accounts

- as our company changes, the table can change, but the column headings remain the same
- the column headings correspond to a *relation schema*
- each row is a *fact* about an employee

Relations and Relation Schemas

A relation is *like* a table, with column headings. [E&N §5.1]

employee

name	**department**
Abbott	Marketing
Burns	Executive
Costello	Accounts
Downer	Accounts

- as our company changes, the table can change, but the column headings remain the same
- the column headings correspond to a *relation schema*
- each row is a *fact* about an employee
- the employee is identified by his/her *name*

SQL - create and populate table

We can use SQL to create this table (schema).

```
CREATE TABLE employee (
   name varchar(20),
   department varchar(20)
);
```

SQL - create and populate table

We can use SQL to create this table (schema).

```
CREATE TABLE employee (
   name varchar(20),
   department varchar(20)
);
```

and add the data rows

```
INSERT INTO employee (name, department)
VALUES ('Abbott', 'Marketing');
```

SQL - create and populate table

We can use SQL to create this table (schema).

```
CREATE TABLE employee (
   name varchar(20),
   department varchar(20)
);
```

and add the data rows

```
INSERT INTO employee (name, department)
VALUES ('Abbott', 'Marketing');
INSERT INTO employee (name, department)
VALUES ('Burns', 'Executive');
INSERT INTO employee (name, department)
VALUES ('Costello', 'Accounts');
INSERT INTO employee (name, department)
VALUES ('Downer', 'Accounts');
```

SQL Queries

show all data in the employee table

```
SELECT *
FROM employee;
```

SQL Queries

show all data in the employee table

```
SELECT *
FROM employee;
```

what departments do we have?

```
SELECT department
FROM employee;
```

SQL Queries

show all data in the employee table

```
SELECT *
FROM employee;
```

what departments do we have?

```
SELECT department
FROM employee;
```

who works for the accounts department?

```
SELECT name
FROM employee
WHERE department='Accounts';
```

How Relational is this “Relational” Database?

- A relation is a *set*.

How Relational is this “Relational” Database?

- A relation is a *set*.
- A given element is either in the set, or not.

How Relational is this “Relational” Database?

- A relation is a *set*.
- A given element is either in the set, or not.
- It can not be in the set twice.

How Relational is this “Relational” Database?

- A relation is a *set*.
- A given element is either in the set, or not.
- It can not be in the set twice.
- That is, $\{A, A\} = \{A\}$.

How Relational is this “Relational” Database?

- A relation is a *set*.
- A given element is either in the set, or not.
- It can not be in the set twice.
- That is, $\{A, A\} = \{A\}$.
- (Don’t panic if this is new to you, but do come along to the special tute in a couple of weeks)

How Relational is this “Relational” Database?

- A relation is a *set*.
- A given element is either in the set, or not.
- It can not be in the set twice.
- That is, $\{A, A\} = \{A\}$.
- (Don’t panic if this is new to you, but do come along to the special tute in a couple of weeks)

How Relational is this “Relational” Database?

- A relation is a *set*.
- A given element is either in the set, or not.
- It can not be in the set twice.
- That is, $\{A, A\} = \{A\}$.
- (Don’t panic if this is new to you, but do come along to the special tute in a couple of weeks)

However, let’s see what the Postgres relational DBMS does with this:

```
INSERT INTO employee (name, department)
VALUES ('Downer', 'Accounts');
```

But if you ask it nicely. . .

It takes a lot of work to remove duplicates from a big table, so the lazy DBMS doesn’t do it unless we ask explicitly.

```
SELECT DISTINCT *
FROM employee
```

But if you ask it nicely. . .

It takes a lot of work to remove duplicates from a big table, so the lazy DBMS doesn't do it unless we ask explicitly.

```
SELECT DISTINCT *
FROM employee
```

- Can one employee work for two departments?

But if you ask it nicely. . .

It takes a lot of work to remove duplicates from a big table, so the lazy DBMS doesn't do it unless we ask explicitly.

```
SELECT DISTINCT *
FROM employee
```

- Can one employee work for two departments?
- Don't ask me! Ask the personnel manager!

But if you ask it nicely. . .

It takes a lot of work to remove duplicates from a big table, so the lazy DBMS doesn't do it unless we ask explicitly.

```
SELECT DISTINCT *
FROM employee
```

- Can one employee work for two departments?
- Don't ask me! Ask the personnel manager!
- We want the database structure to be faithful to the *business rules*.

Relations for Real!

(Those with weak maths backgrounds, do not panic, special tutorial coming soon.)

- You probably already know one *cartesian product*, the x, y number plane.

Relations for Real!

(Those with weak maths backgrounds, do not panic, special tutorial coming soon.)

- You probably already know one *cartesian product*, the x, y number plane.
- Each point in the plane is a pair of numbers (x, y).

Relations for Real!

(Those with weak maths backgrounds, do not panic, special tutorial coming soon.)

- You probably already know one *cartesian product*, the x, y number plane.
- Each point in the plane is a pair of numbers (x, y).
- The plane itself is the set of all such pairs of numbers.

Relations for Real!

(Those with weak maths backgrounds, do not panic, special tutorial coming soon.)

- You probably already know one *cartesian product*, the x, y number plane.
- Each point in the plane is a pair of numbers (x, y).
- The plane itself is the set of all such pairs of numbers.
- If we write $\mathbb{R}$ for the set of real numbers, we write $\mathbb{R} \times \mathbb{R}$ (or $\mathbb{R}^2$) for the number plane.

Relations for Real!

(Those with weak maths backgrounds, do not panic, special tutorial coming soon.)

- You probably already know one *cartesian product*, the x, y number plane.
- Each point in the plane is a pair of numbers (x, y).
- The plane itself is the set of all such pairs of numbers.
- If we write $\mathbb{R}$ for the set of real numbers, we write $\mathbb{R} \times \mathbb{R}$ (or $\mathbb{R}^2$) for the number plane.
- We often draw figures on the plane to represent a subset of the points that interest us, eg those points where $y = 2x$.

Products and Relations

- A relation is a subset of a (cartesian) product of sets.

Products and Relations

- A relation is a subset of a (cartesian) product of sets.
- We can take the product of any sets, eg.

$$\{A, B\} \times \{1, 2\} = \{(A, 1), (A, 2), (B, 1), (B, 2)\}$$

Products and Relations

- A relation is a subset of a (cartesian) product of sets.
- We can take the product of any sets, eg.

$$\{A, B\} \times \{1, 2\} = \{(A, 1), (A, 2), (B, 1), (B, 2)\}$$

- Here is an example relation

$$\{(A, 1), (A, 2), (B, 2)\}$$

Products and Relations

- A relation is a subset of a (cartesian) product of sets.
- We can take the product of any sets, eg.

$$\{A, B\} \times \{1, 2\} = \{(A, 1), (A, 2), (B, 1), (B, 2)\}$$

- Here is an example relation

$$\{(A, 1), (A, 2), (B, 2)\}$$

- which is a subset of the cartesian product $\{A, B\} \times \{1, 2\}$

Relations and Tables

This relation

$$\{(A, 1), (A, 2), (B, 2)\}$$

can also be written as a table

Relations and Tables

This relation

$$\{(A, 1), (A, 2), (B, 2)\}$$

can also be written as a table

relationAsTable

letter	**number**
A	1
A	2
B	2

Relation and Planes

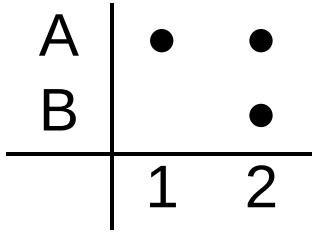
$$\{(A, 1), (A, 2), (B, 2)\}$$

It can also be written as an x, y plane

Relation and Planes

$$\{(A,1),(A,2),(B,2)\}$$

It can also be written as an x, y plane



Functions

- A function is a special kind of relation.

Functions

- A function is a special kind of relation.
- Take a relation $r \subseteq X \times Y$, and one value x from the set X.

Functions

- A function is a special kind of relation.
- Take a relation $r \subseteq X \times Y$, and one value x from the set X.
- How many y's can there be with the pair (x, y) in r?

Functions

- A function is a special kind of relation.
- Take a relation $r \subseteq X \times Y$, and one value x from the set X.
- How many y's can there be with the pair (x, y) in r?
- There might be none, there might be lots!

Functions

- A function is a special kind of relation.
- Take a relation $r \subseteq X \times Y$, and one value x from the set X.
- How many y's can there be with the pair (x, y) in r?
- There might be none, there might be lots!
- r is a *function* if there is exactly one y for each x

Functions

- A function is a special kind of relation.
- Take a relation $r \subseteq X \times Y$, and one value x from the set X.
- How many y's can there be with the pair (x, y) in r?
- There might be none, there might be lots!
- r is a *function* if there is exactly one y for each x
- This means we can use it to look up the unique y for any given x

Keys and Functions

Suppose that each row in the employees relation represents an employee, and that employees can only work for one department.

Keys and Functions

Suppose that each row in the employees relation represents an employee, and that employees can only work for one department. Then

employee

name	**department**
Downer	Accounts
Downer	Marketing

represents a situation that is not allowed.

Keys and Functions

Suppose that each row in the employees relation represents an employee, and that employees can only work for one department. Then

employee

name	**department**
Downer	Accounts
Downer	Marketing

represents a situation that is not allowed.

- The representational rule here is: one row per employee, who is identified by name.

Keys and Functions

Suppose that each row in the employees relation represents an employee, and that employees can only work for one department. Then

employee

name	**department**
Downer	Accounts
Downer	Marketing

represents a situation that is not allowed.

- The representational rule here is: one row per employee, who is identified by name.
- That is, the relation is a *function* from the names of our employees, to the names of our departments.

Keys and Functions

Suppose that each row in the employees relation represents an employee, and that employees can only work for one department. Then

employee

name	**department**
Downer	Accounts
Downer	Marketing

represents a situation that is not allowed.

- The representational rule here is: one row per employee, who is identified by name.
- That is, the relation is a *function* from the names of our employees, to the names of our departments.
- This table violates that rule, there are two rows for Downer.

Keys in SQL

first, let’s lose the previous version of the relation

```
DROP TABLE employee;
```

Keys in SQL

first, let's lose the previous version of the relation

```
DROP TABLE employee;
```

(*Warning:* think carefully before using this command!)

Keys in SQL

first, let's lose the previous version of the relation

```
DROP TABLE employee;
```

(*Warning:* think carefully before using this command!)

Now, create it again with a key.

```
CREATE TABLE employee (
   name varchar(20),
   department varchar(20),
   PRIMARY KEY (name)
);
```

A Key is a Constraint

```
INSERT INTO employee (name, department)
VALUES (’Downer’, ’Marketing’);

INSERT INTO employee (name, department)
VALUES (’Downer’, ’Accounts’);
```

A Key is a Constraint

```
INSERT INTO employee (name, department)
VALUES ('Downer', 'Marketing');

INSERT INTO employee (name, department)
VALUES ('Downer', 'Accounts');
```

- Postgres will not allow two rows with the same primary key value.

A Key is a Constraint

```
INSERT INTO employee (name, department)
VALUES ('Downer', 'Marketing');

INSERT INTO employee (name, department)
VALUES ('Downer', 'Accounts');
```

- Postgres will not allow two rows with the same primary key value.
- That is, it enforces that the relation is a function of its key values.

More Than One Kind of Fact!

We might also want to record some information about our departments.

More Than One Kind of Fact!

We might also want to record some information about our departments.

department

deptName	**budget**	**location**
Marketing	$250,000	Melbourne
Accounts	$150,000	Sydney

employee

name	**department**
⋮	⋮
Downer	Marketing

More Than One Kind of Fact!

We might also want to record some information about our departments.

department

deptName	**budget**	**location**
Marketing	$250,000	Melbourne
Accounts	$150,000	Sydney

employee

name	**department**
⋮	⋮
Downer	Marketing

- Where does Downer work?

More Than One Kind of Fact!

We might also want to record some information about our departments.

department

deptName	**budget**	**location**
Marketing	$250,000	Melbourne
Accounts	$150,000	Sydney

employee

name	**department**
⋮	⋮
Downer	Marketing

- Where does Downer work?
- How did you find that answer?

“Joining” two rows

You probably did something like this

1. find the row in the employees table with the **name** Downer

“Joining” two rows

You probably did something like this

1. find the row in the employees table with the **name** Downer
2. get the **department** value from that row, Marketing

“Joining” two rows

You probably did something like this

1. find the row in the employees table with the **name** Downer
2. get the **department** value from that row, Marketing
3. find the row in the department table with **deptName** Marketing

“Joining” two rows

You probably did something like this

1. find the row in the employees table with the **name** Downer
2. get the **department** value from that row, Marketing
3. find the row in the department table with **deptName** Marketing
4. get the **location** value from that row

Joining two Tables

To find the location of *every* employee, it would be convenient to make a super-table, which adds the department information to each employee.

Joining two Tables

To find the location of *every* employee, it would be convenient to make a super-table, which adds the department information to each employee.

It would contain the following row:

$$\text{employee} \bowtie_{department=deptName} \text{department}$$

name	**department**	**deptName**	**budget**	**location**
⋮	⋮	⋮	⋮	⋮
Downer	Marketing	Marketing	$250,000	Melbourne

SQL for Joins: Product

We can include more than one table in our query.

```
SELECT *
FROM employee, department;
```

SQL for Joins: Product

We can include more than one table in our query.

```
SELECT *
FROM employee, department;
```

- The result is the *product* of the two relations.
- But many of these rows do not represent facts.
- Which ones?
- The useful ones are where *department* $=$ *deptName*.

SQL for Joins

```
SELECT *
FROM employee, department
WHERE department=deptName;
```

SQL for Joins

```
SELECT *
FROM employee, department
WHERE department=deptName;
```

But we just want to know where people work

```
SELECT name, location
FROM employee, department
WHERE department=deptName;
```

Where does he work?

It was only Downer we wanted to know about.

```
SELECT name, location
FROM employee, department
WHERE department=deptName
AND name='Downer';
```

Where does he work?

It was only Downer we wanted to know about.

```
SELECT name, location
FROM employee, department
WHERE department=deptName
AND name='Downer';
```

Now, what about Burns?

```
SELECT name, location
FROM employee, department
WHERE department=deptName
AND name='Burns';
```

Where does he work?

It was only Downer we wanted to know about.

```
SELECT name, location
FROM employee, department
WHERE department=deptName
AND name='Downer';
```

Now, what about Burns?

```
SELECT name, location
FROM employee, department
WHERE department=deptName
AND name='Burns';
```

Oops, we don't have an executive department!

Where does he work?

It was only Downer we wanted to know about.

```
SELECT name, location
FROM employee, department
WHERE department=deptName
AND name='Downer';
```

Now, what about Burns?

```
SELECT name, location
FROM employee, department
WHERE department=deptName
AND name='Burns';
```

Oops, we don't have an executive department!
This database is kind of broken, and should not be allowed.

Foreign Key Constraints

The values in **department** in the employee relation are meant to “point to” rows in the department table.

```
CREATE TABLE employee (
   name varchar(20),
   department varchar(20),
   PRIMARY KEY (name),
   FOREIGN KEY (department) REFERENCES department(de
);
```

This tells the DBMS that when this table “points to” a row in department, it must be there!

```
INSERT INTO employee (name, department)
VALUES (’Burns’,’Executive’);
```

Summary

That was a lot of material!

- Tables, with rows as “facts”

Summary

That was a lot of material!

- Tables, with rows as “facts”
- Keys, to identify the thing the “fact” is about

Summary

That was a lot of material!

- Tables, with rows as “facts”
- Keys, to identify the thing the “fact” is about
- Relations, products of sets

Summary

That was a lot of material!

- Tables, with rows as “facts”
- Keys, to identify the thing the “fact” is about
- Relations, products of sets
- Function, a kind of relation

Summary

That was a lot of material!

- Tables, with rows as “facts”
- Keys, to identify the thing the “fact” is about
- Relations, products of sets
- Function, a kind of relation
- Key constrains a relation to be a function

Summary

That was a lot of material!

- Tables, with rows as “facts”
- Keys, to identify the thing the “fact” is about
- Relations, products of sets
- Function, a kind of relation
- Key constrains a relation to be a function
- Combining relations using join

Summary

That was a lot of material!

- Tables, with rows as “facts”
- Keys, to identify the thing the “fact” is about
- Relations, products of sets
- Function, a kind of relation
- Key constrains a relation to be a function
- Combining relations using join
- Basics of SQL for all of the above!