

## Lecture 6: The Relational Model: Updates and Integrity

### A more careful look at the relational data model

- Tips on your own PostgreSQL installation
- Domains, Relation Schemas and Database Schemas
- Keys and Integrity Constraints

## Quick Feedback

I do not want to wait till the end of the year to find out how you found my teaching. Give me some feedback now, and perhaps we can improve things *during* the course.

## Quick Feedback

I do not want to wait till the end of the year to find out how you found my teaching. Give me some feedback now, and perhaps we can improve things *during* the course.

Please write answers to the following questions on a sheet of paper and place it in the envelope when it is passed around.

(yes, no, maybe)

- 1 Was it helpful to overview the whole course in the first 3 lectures?
- 2 Do feel you know what to expect from the course now?
- 3 Were these first 3 lectures too simplistic?
- 4 Did you understand most of lecture 5 (on basic maths)?
- 5 What would be most helpful for you to learn the content if this course?

# Cooperation and Plagiarism

Executive summary: cooperation yes, plagiarism no.

# Cooperation and Plagiarism

Executive summary: cooperation yes, plagiarism no.

I encourage you to discuss the course with other students, in labs, as you move around campus, in study groups.

# Cooperation and Plagiarism

Executive summary: cooperation yes, plagiarism no.

I encourage you to discuss the course with other students, in labs, as you move around campus, in study groups.

However *work handed in for assessment must be your own.*

We have ways of detecting plagiarism, and the consequences can be severe.

## Installing PostgreSQL on Linux/Unix

Discussion on the forum led to creating your own PostgreSQL installation. Here are the short installation instructions for the Unix versions of PostgreSQL 8.2.4

```
./configure
gmake
su
gmake install
adduser postgres
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su - postgres
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/dat
/usr/local/pgsql/bin/postgres -D /usr/local/pgsql/d
>logfile 2>&1 &
/usr/local/pgsql/bin/createdb test
/usr/local/pgsql/bin/psql test
```

## Other PostgreSQL Installation Strategies

- 1 forget it, use what the DCS IT guys provide in the labs!

## Other PostgreSQL Installation Strategies

- 1 forget it, use what the DCS IT guys provide in the labs!
- 2 use Linux packages: .rpm .deb etc

## Other PostgreSQL Installation Strategies

- 1 forget it, use what the DCS IT guys provide in the labs!
- 2 use Linux packages: .rpm .deb etc
- 3 non-root unix installation works  
`./configure --prefix=$HOME`

## Other PostgreSQL Installation Strategies

- 1 forget it, use what the DCS IT guys provide in the labs!
- 2 use Linux packages: .rpm .deb etc
- 3 non-root unix installation works  
`./configure --prefix=$HOME`
- 4 the Unix/Linux way would work on a Macintosh, so long as you have the xcode developer tools installed

## Other PostgreSQL Installation Strategies

- 1 forget it, use what the DCS IT guys provide in the labs!
- 2 use Linux packages: .rpm .deb etc
- 3 non-root unix installation works  
`./configure --prefix=$HOME`
- 4 the Unix/Linux way would work on a Macintosh, so long as you have the xcode developer tools installed
- 5 a Windows version with graphical installation is available, see instructions at  
<http://pginstaller.projects.postgresql.org>

## Datatypes and Domains

A domain is a datatype, named and constrained to represent a particular kind of real world value.

## Datatypes and Domains

A domain is a datatype, named and constrained to represent a particular kind of real world value.

For Example, we could create a domain `sex` and constrain it to the values `'M'` and `'F'`.

```
CREATE DOMAIN sex AS char
CHECK ( VALUE = 'M' or VALUE = 'F' );
```

## Datatypes and Domains

A domain is a datatype, named and constrained to represent a particular kind of real world value.

For Example, we could create a domain `sex` and constrain it to the values `'M'` and `'F'`.

```
CREATE DOMAIN sex AS char  
CHECK ( VALUE = 'M' or VALUE = 'F' );
```

- A datatype is just a set

## Datatypes and Domains

A domain is a datatype, named and constrained to represent a particular kind of real world value.

For Example, we could create a domain sex and constrain it to the values 'M' and 'F'.

```
CREATE DOMAIN sex AS char  
CHECK ( VALUE = 'M' or VALUE = 'F' );
```

- A datatype is just a set
- eg varchar(20) is the set of all strings of characters of length 20 or less

## Datatypes and Domains

A domain is a datatype, named and constrained to represent a particular kind of real world value.

For Example, we could create a domain `sex` and constrain it to the values `'M'` and `'F'`.

```
CREATE DOMAIN sex AS char  
CHECK ( VALUE = 'M' or VALUE = 'F' );
```

- A datatype is just a set
- eg `varchar(20)` is the set of all strings of characters of length 20 or less
- A domain is a subset of its underlying datatype

## Datatypes and Domains

A domain is a datatype, named and constrained to represent a particular kind of real world value.

For Example, we could create a domain sex and constrain it to the values 'M' and 'F'.

```
CREATE DOMAIN sex AS char  
CHECK ( VALUE = 'M' or VALUE = 'F' );
```

- A datatype is just a set
- eg varchar(20) is the set of all strings of characters of length 20 or less
- A domain is a subset of its underlying datatype
- $\{ 'M', 'F' \} \subseteq \{ '#', '$', \dots, 'a', \dots, 'z', 'A', \dots, 'Z', '0', \dots, '9' \}$

# Datatypes and Domains

## Question

Why is this

```
CREATE TABLE employee (  
    name varchar(20);  
    gender sex  
);
```

a better table definition than this

```
CREATE TABLE employee (  
    name varchar(20);  
    gender char  
);
```

## Relation

[E&N §5.1]

A relation is a set, but a table is a list.

# Relation

[E&N §5.1]

A relation is a set, but a table is a list.

- the tuples in a relation are not arranged in any particular order

$$\{1, 2, 3\} = \{3, 2, 1\}$$

(if we rearrange the rows in a table, we get a different table)

# Relation

[E&N §5.1]

A relation is a set, but a table is a list.

- the tuples in a relation are not arranged in any particular order  
 $\{1, 2, 3\} = \{3, 2, 1\}$   
(if we rearrange the rows in a table, we get a different table)
- the relation can not contain distinct but equal tuples  
 $\{1, 1\} = \{1\}$   
(a table with one row is different to the table that has that row twice)

# Relation

[E&N §5.1]

A relation is a set, but a table is a list.

- the tuples in a relation are not arranged in any particular order  
 $\{1, 2, 3\} = \{3, 2, 1\}$   
(if we rearrange the rows in a table, we get a different table)
- the relation can not contain distinct but equal tuples  
 $\{1, 1\} = \{1\}$   
(a table with one row is different to the table that has that row twice)

# Relation

[E&N §5.1]

A relation is a set, but a table is a list.

- the tuples in a relation are not arranged in any particular order  
 $\{1, 2, 3\} = \{3, 2, 1\}$   
(if we rearrange the rows in a table, we get a different table)
- the relation can not contain distinct but equal tuples  
 $\{1, 1\} = \{1\}$   
(a table with one row is different to the table that has that row twice)

As we saw, PostgreSQL is not truly relational in this sense, but now we are talking about the relational model, not (imperfect) implementations.

## Relation Schema

A relation schema is analagous to the column headings of a table.

# Relation Schema

A relation schema is analagous to the column headings of a table. It has

- a name

# Relation Schema

A relation schema is analagous to the column headings of a table. It has

- a name
- a list of **attributes**, each of which has

# Relation Schema

A relation schema is analagous to the column headings of a table. It has

- a name
- a list of attributes, each of which has
  - a name

# Relation Schema

A relation schema is analagous to the column headings of a table. It has

- a name
- a list of attributes, each of which has
  - a name
  - a domain

# Relation Schema

A relation schema is analagous to the column headings of a table. It has

- a name
- a list of attributes, each of which has
  - a name
  - a domain

# Relation Schema

A relation schema is analagous to the column headings of a table. It has

- a name
- a list of attributes, each of which has
  - a name
  - a domain

We could write this as  $(A_1, \dots, A_n)$  where  $N_i$  is the name of attribute  $A_i$  and  $D_i$  is its domain.

## Relation Schema

We might also write the attribute  $A_i$  as  $N_i : D_i$

## Relation Schema

We might also write the attribute  $A_i$  as  $N_i : D_i$ , and the relation schema as

$$\mathbf{R}(N_1 : D_1, \dots, N_n : D_n)$$

where  $\mathbf{R}$  is the relation schema name.

## Relation Schema

We might also write the attribute  $A_i$  as  $N_i : D_i$ , and the relation schema as

$$\mathbf{R}(N_1 : D_1, \dots, N_n : D_n)$$

where  $\mathbf{R}$  is the relation schema name.

eg *employee*(*name* : *string*, *gender* : *sex*, *department* : *string*)

## Relation Schema

We might also write the attribute  $A_i$  as  $N_i : D_i$ , and the relation schema as

$$\mathbf{R}(N_1 : D_1, \dots, N_n : D_n)$$

where  $\mathbf{R}$  is the relation schema name.

eg *employee*(*name* : *string*, *gender* : *sex*, *department* : *string*)

We can indicate the primary key by underlining

*employee*(*name* : *string*, *gender* : *sex*, *department* : *string*)

# Schema States

A valid **state** of the relation schema is a relation whose tuples all match the schema.

## Schema States

A valid **state** of the relation schema is a relation whose tuples all match the schema.

That is, relation  $R$  is a valid state of relation schema  $\mathbf{R}(N_1 : D_1, \dots, N_n : D_n)$  iff

$$R \subseteq D_1 \times \dots \times D_n$$

## Keys can have more than one attribute

- Last week we only considered keys of a single attribute, however . . .

## Keys can have more than one attribute

- Last week we only considered keys of a single attribute, however . . .
- Consider a company with departments that do their own hiring and firing and HR stuff.

## Keys can have more than one attribute

- Last week we only considered keys of a single attribute, however . . .
- Consider a company with departments that do their own hiring and firing and HR stuff.
- Each department would probably allocate employee numbers to their staff too.

## Keys can have more than one attribute

- Last week we only considered keys of a single attribute, however . . .
- Consider a company with departments that do their own hiring and firing and HR stuff.
- Each department would probably allocate employee numbers to their staff too.
- To uniquely identify an employee in the company, we would need to specify both her employee number AND her department.

## Projection

A bit of notation from relational algebra will help us define keys properly.

## Projection

A bit of notation from relational algebra will help us define keys properly.

You now understand the SQL query

```
SELECT  $N_i$   
FROM  $R$ 
```

## Projection

A bit of notation from relational algebra will help us define keys properly.

You now understand the SQL query

```
SELECT  $N_i$   
FROM  $R$ 
```

We can also write this as

$$\pi_i(R)$$

or

$$\pi_{N_i}(R)$$

## Projection Example

*employee*(*name* : *string*, *gender* : *sex*, *department* : *string*)

The query

```
SELECT gender  
FROM employee
```

can be written

$\pi_{\text{gender}}(\textit{employee})$

or

$\pi_3(\textit{employee})$

# Projection

Similarly

```
SELECT  $N_i, N_j, N_k$   
FROM  $R$ 
```

can be written  $\pi_{\{i,j,k\}}(R)$  or  $\pi_{\{N_i,N_j,N_k\}}(R)$ .

# Projection

Similarly

```
SELECT  $N_i, N_j, N_k$   
FROM  $R$ 
```

can be written  $\pi_{\{i,j,k\}}(R)$  or  $\pi_{\{N_i, N_j, N_k\}}(R)$ .

If  $K = \{N_i, N_j, N_k\}$ , we can write the query as  $\pi_K(R)$ .

# Projection

Similarly

```
SELECT  $N_i, N_j, N_k$   
FROM  $R$ 
```

can be written  $\pi_{\{i,j,k\}}(R)$  or  $\pi_{\{N_i, N_j, N_k\}}(R)$ .

If  $K = \{N_i, N_j, N_k\}$ , we can write the query as  $\pi_K(R)$ .

Projections can also apply to single tuples rather than sets of them, eg  $\pi_2(a, b, c) = b$ .

## Projection Example

Let  $K = \{name, department\}$ ,

## Projection Example

Let  $K = \{name, department\}$ ,

then  $\pi_K(employee)$  means

```
SELECT name, department  
FROM employee
```

## Projection Example

Let  $K = \{name, department\}$ ,

then  $\pi_K(employee)$  means

```
SELECT name, department  
FROM employee
```

We can set  $K$  to be the primary key like this

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

# Keys

[E&N §5.2.2]

Specifying a primary key imposes a further **constraint** on what relations are valid states of the relation schema:

*if two tuples have the same values for the primary key, then they are the same tuple*

# Keys

[E&N §5.2.2]

Specifying a primary key imposes a further **constraint** on what relations are valid states of the relation schema:

*if two tuples have the same values for the primary key, then they are the same tuple*

In “greek”, if  $K \subseteq \{N_1, \dots, N_n\}$  is the primary key of relation schema  $\mathbf{R}$ , then  $R \subseteq D_1 \times \dots \times D_n$  is a valid state of  $\mathbf{R}$  iff for every pair of tuples  $r, s \in R$  we have

$$\pi_K(r) = \pi_K(s) \longrightarrow r = s$$

## Primary Key Violation

For example, if **employeeNumber** is specified as the primary key, then the following relation violates the primary key constraint.

<b>name</b>	<b>department</b>	<b>employeeNumber</b>
Joe Smith	Deliveries	12345
Jane White	Sales	12345

## Primary Key Violation

For example, if **employeeNumber** is specified as the primary key, then the following relation violates the primary key constraint.

<b>name</b>	<b>department</b>	<b>employeeNumber</b>
Joe Smith	Deliveries	12345
Jane White	Sales	12345

However, if **department** and **employeeNumber** are specified as the primary key, then its fine.

# Entity Integrity

## Entity Integrity

Specifying a primary key also invokes the **entity integrity** constraint: *No primary key value can be NULL*

<b>name</b>	<b>department</b>	<b>employeeNumber</b>
Joe Smith	Deliveries	12345
Jane White	NULL	12346

## Entity Integrity

Specifying a primary key also invokes the **entity integrity** constraint: *No primary key value can be NULL*

<b>name</b>	<b>department</b>	<b>employeeNumber</b>
Joe Smith	Deliveries	12345
Jane White	NULL	12346

If **department** and **employeeNumber** are specified as the primary key, then this relation violates entity integrity.

## Entity Integrity

Specifying a primary key also invokes the **entity integrity** constraint: *No primary key value can be NULL*

<b>name</b>	<b>department</b>	<b>employeeNumber</b>
Joe Smith	Deliveries	12345
Jane White	NULL	12346

If **department** and **employeeNumber** are specified as the primary key, then this relation violates entity integrity.

But if **employeeNumber** is the primary key, its fine.

## Foreign Keys

When we use attributes to point to another tuple, we want that tuple to exist!

## Foreign Keys

When we use attributes to point to another tuple, we want that tuple to exist!

Specifying one or more attributes as a **foreign key** imposes a *referential integrity* constraint.

## Referential Integrity Violation

<b>name</b>	<b>department</b>	<b>employeeNumber</b>
Joe Smith	Deliveries	12345
Jane White	NULL	12346

<b>department</b>	<b>annualBudget</b>
Support	\$10,000
Sales	\$200,000

Here, if **department** is specified as a foreign key into the second table, then the first (!) row violates referential integrity, because there is no Deliveries department!

Note that the second row is fine in terms of referential integrity, though it may violate a *business rule*.

# Database Schema

A database schema consists of

- domains

# Database Schema

A database schema consists of

- domains
- relation schemas

# Database Schema

A database schema consists of

- domains
- relation schemas
- integrity constraints (ie primary and foreign key specifications)

# Database Schema

A database schema consists of

- domains
- relation schemas
- integrity constraints (ie primary and foreign key specifications)

# Database Schema

A database schema consists of

- domains
- relation schemas
- integrity constraints (ie primary and foreign key specifications)

A valid state of a database schema is a valid state (relation) for each relation schema, where this collection of relations satisfies the integrity constraints.

# Labs

- Make sure you enrol in a lab group before next week
- Make sure you attend your lab (2% per lab, and important learning experience)
- Make sure you *understand* what you are doing - if stuck ask your tutor for help
- Remember: *no lecture* next Wednesday 6th August