Databases - 5

Problems with the relational model
Functions and sub-queries

Problems (1)

To store information about real life entities, we often have to cut them up into separate tables

Page *1* — ***DreamHome*** Customer Rental Details — Date *7-Oct-98*

Customer Name *John Kay* Customer Number *CR76*

Property Number	Property Address	Rent Start	Rent Finish	Rent	Owner Number	Owner Name
PG4	6 Lawrence St, Glasgow	1-Jul-94	31-Aug-96	350	CO40	Tina Murphy
PG16	5 Novar Dr, Glasgow	1-Sep-96	1-Sep-98	450	CO93	Tony Shaw

Problems (1)

To store this information efficiently we'd use 4 tables

Customer

Customer No	***Cname***
CR76	John Kay
CR56	Aline Stewart

Rental

Customer No	***Property No***	***RentStart***	***RentFinish***
CR76	PG4	1-Jul-94	31-Aug-96
CR76	PG16	1-Sep-96	1-Sep-08
CR56	PG4	1-Sep-92	10-Jun-94
CR56	PG36	10-Oct-94	1-Dec-95
CR56	PG16	1-Jan-96	10-Aug-96

Property

Property No	***Paddress***	***Rent***	***Owner_No***
PG4	6 Lawrence St, Glasgow	350	CO40
PG36	2 Manor Rd, Glasgow	375	CO93
PG16	5 Novar Dr, Glasgow	450	CO93

Owner

Owner_No	***Oname***
CO40	Tina Murphy
CO93	Tony Shaw

Problems (1)

To answer any questions we have to put them back together again!

select *
from customer, rental, property, owner
where .....

Customer

Customer No	*Cname*
CR76	John Kay
CR56	Aline Stewart

Rental

Customer No	*Property No*	*RentStart*	*RentFinish*
CR76	PG4	1-Jul-94	31-Aug-96
CR76	PG16	1-Sep-96	1-Sep-08
CR56	PG4	1-Sep-92	10-Jun-94
CR56	PG36	10-Oct-94	1-Dec-95
CR56	PG16	1-Jan-96	10-Aug-96

Property

Property No	*Paddress*	*Rent*	*Owner_No*
PG4	6 Lawrence St, Glasgow	350	CO40
PG36	2 Manor Rd, Glasgow	375	CO93
PG16	5 Novar Dr, Glasgow	450	CO93

Owner

Owner_No	*Oname*
CO40	Tina Murphy
CO93	Tony Shaw

Problems (1)

select *
from customer, rental, property, owner
where .....

Cartesian Product is the most 'expensive' operation

- Takes time
- Requires lots of memory
- Requires lots of processor power

Note

Thankfully, lots of research work in this area to improve the efficiency

Solution - Object Oriented Databases

Store information as a whole item - called an Object (in the Object data model)

Page *1* — ***DreamHome*** Customer Rental Details — Date *7-Oct-98*

Customer Name *John Kay* Customer Number *CR76*

Property Number	Property Address	Rent Start	Rent Finish	Rent	Owner Number	Owner Name
PG4	6 Lawrence St, Glasgow	1-Jul-94	31-Aug-96	350	CO40	Tina Murphy
PG16	5 Novar Dr, Glasgow	1-Sep-96	1-Sep-98	450	CO93	Tony Shaw

Problems (2) - Temporal information

The relational model is not very good at storing many 'states' of information (unless explicitly structured)

student

kuID	name	term time address
k0699345	Fred Tester	17 Penrhyn Road
k0545646	Martin Qwerty	20 Richmond Road
k0665665	Mary Smith	34 Kingston Road

Problems (2) - Temporal information

What happens when Martin moves in with Fred?

student

kuID	name	term time address
k0699345	Fred Tester	17 Penrhyn Road
k0545646	Martin Qwerty	17 Penrhyn Road
k0665665	Mary Smith	34 Kingston Road

Problems (2) - Temporal information

Query: Show all the places Martin has ever lived

Problem: We can't

student

kuID	name	term time address
k0699345	Fred Tester	17 Penrhyn Road
k0545646	Martin Qwerty	17 Penrhyn Road
k0665665	Mary Smith	34 Kingston Road

Problems (2) - Temporal information

Query: When did Martin move house?

Problem: Can't answer

student

kuID	name	term time address
k0699345	Fred Tester	17 Penrhyn Road
k0545646	Martin Qwerty	17 Penrhyn Road
k0665665	Mary Smith	34 Kingston Road

Problems (2) - Temporal information

Query: When did Martin tell us that he'd move house?

Problem: Can't answer

student

kuID	name	term time address
k0699345	Fred Tester	17 Penrhyn Road
k0545646	Martin Qwerty	17 Penrhyn Road
k0665665	Mary Smith	34 Kingston Road

Solution - Temporal databases / temporal SQL

Databases explicitly structured to store old information

Databases explicitly structured to note when information changes

Problems (3) - Truth and probability

The relational model only stores information that is 100% true

student

kuID	name
k0699345	Fred Tester
k0545646	Martin Qwerty
k0665665	Mary Smith

By having these rows in the table...

...we are asserting that this information is true

Problems (3) - Truth and probability

student

kuID	name
k0699345	Fred Tester
k0545646	Martin Qwerty
k0665665	Mary Smith

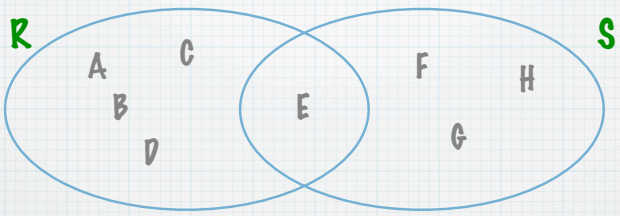
Fred is a student
Martin is a student
Mary is a student

All these statements are true - all other information is false

The Closed World Assumption (CWA)

The Closed World Assumption (CWA)

Unless explicitly included, everything else is false



Is P in R ? No.

Problems (3) - Truth and probability

All other information is false

Query: Is Robyn a student ?

student

kuID	name
k0699345	Fred Tester
k0545646	Martin Qwerty
k0665665	Mary Smith

Robyn isn't here

...so Robyn isn't a student

Problems (3) - Truth and probability

But what if we want to store negative or false information

Robyn is to NEVER be a student

student

kuID	name
k0699345	Fred Tester
k0545646	Martin Qwerty
k0665665	Mary Smith

Can't do it

Problems (3) - Truth and probability

But what if we want to store information that is probably true

Its 90% probable that Robyn is a student

student

kuID	name
k0699345	Fred Tester
k0545646	Martin Qwerty
k0665665	Mary Smith

Can't do it

Solution - Deductive databases

Databases explicitly structured to store rules and facts

These can store complex statements of truth and false

Problems (4) - Some queries can't be expressed in SQL

Find all the line managers for every employee

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
405	MARCH	ADMIN	938	13/06/1997	18000		2
535	BYRNE	SALES	734	15/08/1997	26000	300	3
557	BELL	SALES	734	26/03/2000	22500	500	3
602	BIRD	MANAGER	875	31/10/1997	39750		2
690	AHMAD	SALES	734	05/12/1997	22500	1400	3
734	COX	MANAGER	875	11/06/2002	38500		3
818	POLLARD	MANAGER	875	14/05/2000	34500		1
824	REES	ANALYST	602	05/03/2000	40000		2
875	PARKER	PRESIDENT		09/07/2002	60000		1
880	TURNER	SALES	734	04/06/2001	25000	0	3
912	HAYES	ADMIN	824	04/06/2001	21000		2
936	CASSY	ADMIN	734	23/07/2002	19500		3
938	GIBSON	ANALYST	602	05/12/1997	40000		2
970	BLACK	ADMIN	818	21/11/1997	23000		1

Black - Managers: Pollard, Parker
March - Managers: Gibson, Bird, Parker

Solution - Extensions to SQL

Extensions to the language to allow this kind of loop processing

Problem: Clutters the language

Functions

Transform a value or set of values using some rule

Built into the SQL standard

Functions Categories

String	concatenation, length, substring
Arithmetic	max, min, power, round, trunc
Date	add, subtract dates
Aggregate or group	average, sum, count

String Function - String Concatenation - concat

Combines fields and /or additional text

```
SELECT concat( "Employee: ", ename, " Sal: ", sal )
FROM emp
```

concat("Employee: ",ename, " Sal: ",sal)
Employee: BYRNE Sal: 26000
Employee: BELL Sal: 22500
Employee: BIRD Sal: 39750
Employee: AHMAD Sal: 22500
Employee: COX Sal: 38500
Employee: POLLARD Sal: 34500
Employee: REES Sal: 40000
Employee: PARKER Sal: 60000
Employee: TURNER Sal: 25000
Employee: HAYES Sal: 21000
Employee: CASSY Sal: 19500
Employee: GIBSON Sal: 40000
Employee: BLACK Sal: 23000
Employee: MARCH Sal: 18000

String Function - Substring

mid (string, starting point, no of chars) - returns part of a string

```
select ename, mid (ename, 2, 4)
from emp
```

ename	mid(ename, 2, 4)
BYRNE	YRNE
BELL	ELL
BIRD	IRD
AHMAD	HMAD
COX	OX
POLLARD	OLLA
REES	EES
PARKER	ARKE
TURNER	URNE
HAYES	AYES
CASSY	ASSY
GIBSON	IBSO
BLACK	LACK
MARCH	ARCH

Arithmetic Function - min

min () - returns smallest value in a column

SAL
18000
26000
22500
39750
22500
38500
34500
40000
60000
25000
21000
19500
40000
23000

```
select min (sal)
from emp
```

min(sal)
18000

Arithmetic Function - max

max () - returns largest value in a column

SAL
18000
26000
22500
39750
22500
38500
34500
40000
60000
25000
21000
19500
40000
23000

```
select max (sal)
from emp
```

max(sal)
60000

Aggregate Function - avg

avg () - returns mean value in a column

SAL
18000
26000
22500
39750
22500
38500
34500
40000
60000
25000
21000
19500
40000
23000

```
select avg (sal)
from emp
```

avg(sal)
30732.1429

Aggregate Function - sum

sum () - returns total of all values in a column

SAL
18000
26000
22500
39750
22500
38500
34500
40000
60000
25000
21000
19500
40000
23000

```
select sum (sal)
from emp
```

sum(sal)
430250

Aggregate Function - count

count () - returns total number of values in a column

SAL
18000
26000
22500
39750
22500
38500
34500
40000
60000
25000
21000
19500
40000
23000

```
select count (sal)
from emp
```

count(sal)
14

Group by

Aggregate functions can be applied to subsets of the table by using the group by syntax

```
select * or expression
from relations
[where expression]
[group by expression]
```

Group by

Aggregate functions can be applied to subsets of the table by using the group by syntax

e.g.

Calculate avg()

Calculate avg()

Calculate avg()

ENAME	JOB	MGR	HIREDATE	SAL
BLACK	ADMIN	818	21/11/1997	23000
CASSY	ADMIN	734	23/07/2002	19500
HAYES	ADMIN	824	04/06/2001	21000
MARCH	ADMIN	938	13/06/1997	18000
GIBSON	ANALYST	602	05/12/1997	40000
REES	ANALYST	602	05/03/2000	40000
POLLARD	MANAGER	875	14/05/2000	34500
COX	MANAGER	875	11/06/2002	38500
BIRD	MANAGER	875	31/10/1997	39750
PARKER	PRESIDENT		09/07/2002	60000
TURNER	SALES	734	04/06/2001	25000
AHMAD	SALES	734	05/12/1997	22500
BELL	SALES	734	26/03/2000	22500
BYRNE	SALES	734	15/08/1997	26000

Group by

Aggregate functions can be applied to subsets of the table by using the group by syntax

```
select job, avg(sal)
from emp
group by job
```

job	avg(sal)
ADMIN	20375.0000
ANALYST	40000.0000
MANAGER	37583.3333
PRESIDENT	60000.0000
SALES	24000.0000

Group by

Find the highest salary for each job category

```
select job, max(sal)
from emp
group by job
```

job	max(sal)
ADMIN	23000
ANALYST	40000
MANAGER	39750
PRESIDENT	60000
SALES	26000

Nested sub-queries

When one of the conditions of a WHERE clause is a query itself, this is called a nested sub-query, i.e.,

```
SELECT select-list
FROM table(s)
WHERE object operator (SELECT select-list
            FROM table(s)
            [WHERE condition]);
```

Nested sub-queries example

Find all the employees who earn more than the average salary

Start by finding the average salary

```
select avg(sal)
from emp
```

avg(sal)
30732.1429

Nested sub-queries example

Find all the employees who earn more than the average salary

Now find all the employees that earn more than this:

```
select ename, sal
from emp
where sal >= ( select avg(sal)
               from emp )
```

ename	sal
BIRD	39750
COX	38500
POLLARD	34500
REES	40000
PARKER	60000
GIBSON	40000

Nested sub-queries example (2)

Typically, you need to use a nested sub-query where you need to use an aggregate function and an attribute at the same time

Find the employee who earns the least money

Attribute required

Aggregate function required

Nested sub-queries example (2)

Find the employee who earns the least money

First attempt may try something like this:

```
select ename, min(sal)
from ...
where ...
```

single value

many values

This can't work

Nested sub-queries example (2)

Find the employee who earns the least money

Better attempt:

```
select min(sal)
from emp
```

Find the minimum salary

Nested sub-queries example (2)

Find the employee who earns the least money

Better attempt:

```
select ename, sal
from emp
where sal= (select min(sal)
            from emp)
```

Find the person who has the minimum salary

ename	sal
MARCH	18000