Relational Algebra and Calculus

Part 2

Decomposing complex operations

- We can decompose operations into a series of smaller relational algebra operations and give a name to the results of intermediate expressions. We use the assignment operation, denoted by ← to name the results of a relational algebra operations.

JOINs

- Types of joins:
  - Theta join
  - Equijoin
  - Natural join
  - Outer join
  - Semijoin.

Theta join (θ-join)

- $R \blacktriangleright\blacktriangleleft_F S$
  - The theta join operation defines a relation that contains tuples satisfying the predicate *F* from the Cartesian product of R and S.
  - The predicate F is of the form $R.a_i\ \theta\ S.b_i$ where θ may be one of the comparison operators (<, ≤, >, ≥, =, ≠).
- Or $R \bowtie_F S$
  - *To get this symbol, I used clipart, the one above uses symbols from the symbol table.*
- This can also be expressed as:
  - $R \bowtie_F S = \sigma_F(RXS)$

Equijoin operation

- The equijoin is a theta join where the comparison operator is =.
- E.g. Which customer bought corder no 1?
- Result ← Customer $\bowtie_{customer.customer\_id=corder.customer\_id}$ CORDER
- This result could also be represented as:
- Result ← $\sigma_{customer.customer\_id=corder.customer\_id}$(customerXcorder)

The natural join

- The Natural join is an equijoin of two relations R and S over all common attributes *x*. One occurrence of each common attribute is eliminated from the result.
- R⋈S represents the natural join.
- To return the customer name and address, the order no and the order date to relation P:

$$P \leftarrow (\pi_{customer\_id,\ customer\_address}(CUSTOMER) \bowtie \pi_{customer\_id,\ corderno,\ order\_date}(CORDER))$$

Outer join

- R ⟕ S The (left) outer join is a join in which tuples from R that do not have matching values in the common attributes of S are also included in the result relation.
  - Missing values in the second relation are set to null.
- E.g.
- $(\pi_{\text{customer\_id, customer\_name}}(\text{CUSTOMER})) \text{⟕ CORDER}$
- This returns all customers and any related orders.

OUTER JOIN Example 1

R

ColA	*ColB*
A	1
B	2
D	3
F	4
E	5

S

SColA	*SColB*
A	1
C	2
D	3
E	4

R LEFT OUTER JOIN R.ColA = S.SColA S

A	1	A	1
D	3	D	3
E	5	E	4
B	2	-	-
F	4	-	-

R RIGHT OUTER JOIN R.ColA = S.SColA S

A	1	A	1
D	3	D	3
E	5	E	4
-	-	C	2

OUTER JOIN Example 2

R

ColA	*ColB*
A	1
B	2
D	3
F	4
E	5

S

SColA	*SColB*
A	1
C	2
D	3
E	4

R FULL OUTER JOIN R.ColA = S.SColA S

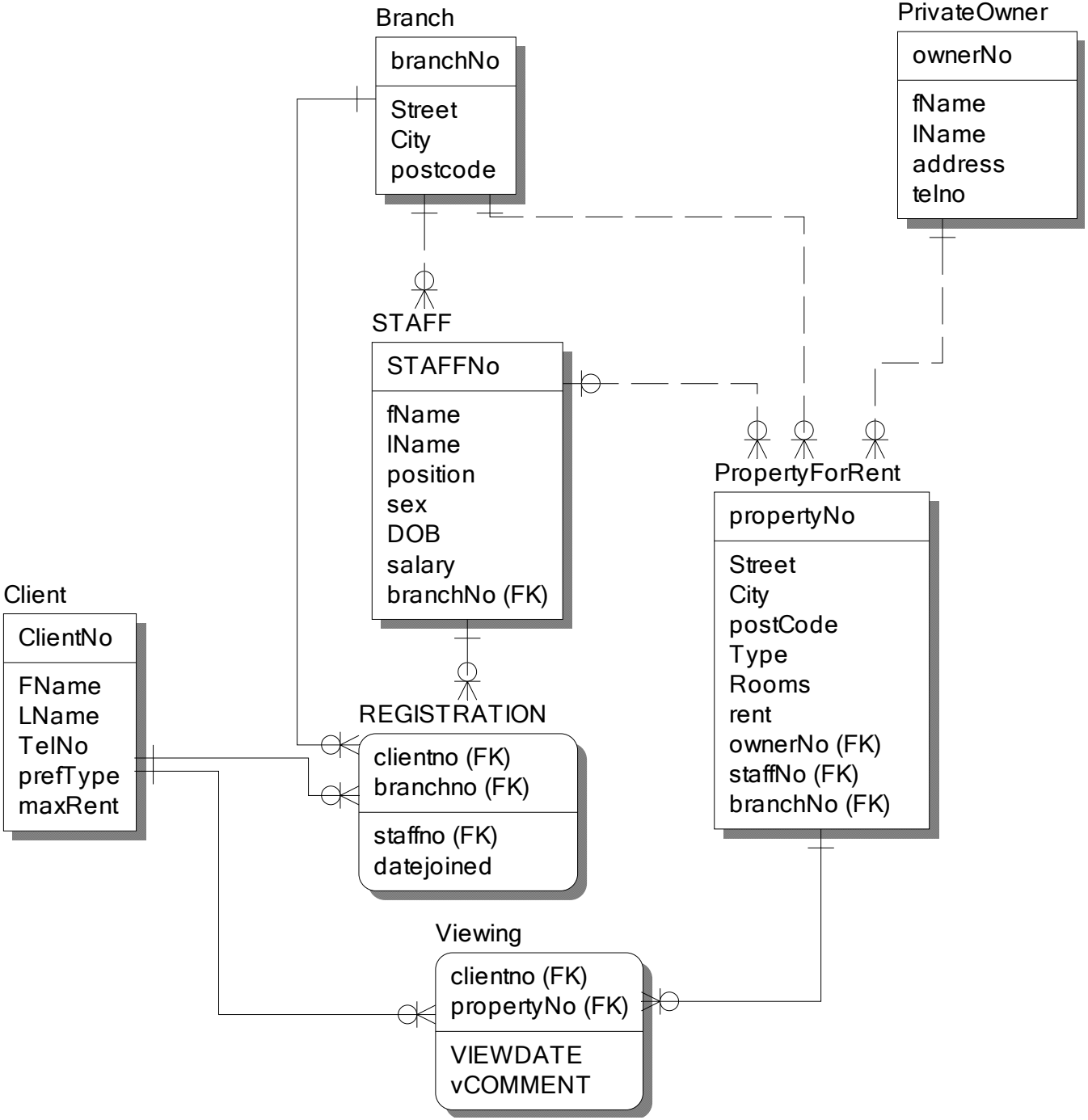
A	1	A	1
D	3	D	3
E	5	E	4
B	2	-	-
F	4	-	-
-	-	C	2

Semijoin

- $R \blacktriangleright_F S$ The semijoin operation defines a relation that contains the tuples of R that participate in the join of R with S.
- E.g. Complete details of all staff who have both issued and been paid for orders:

$$Staff \blacktriangleright_{customer.customer\_id = corder.customer\_id} (\sigma_{staff\_issued=staff\_paid)}(CORDER))$$

- This returns the details of the staff members only.



Connolly & Begg 'Dream Home'

Branch

Staff

propertyforrent

BRANCH NO	STREET	CITY	POSTCODE
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

STAFF NO	FNAME	LNAME	POSITION	SEX	DOB	SALARY	BRANCH NO
SL21	John	White	Manager	M	01-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	03-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jul-65	9000	B005

PROPERTYNO	STREET	CITY	POSTCODE	TYPE	ROOMS	RENT	OWNERNO	STAFFNO	BRANCHNO
PA14	16 Holhead	Aberdeen	AB7 SSU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40	-	B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

CLIENTNO	FNAME	LNAME	TELNO	PREFTYPE	MAXRENT
CR76	John	Kay	0207-774-5632	Flat	425
CR56	Aline	Stewart	0141-848-1825	Flat	350
CR74	Mike	Ritchie	01475-392178	House	750
CR62	Mary	Tregear	01224-196720	Flat	600

CLIENT

OWNERNO	FNAME	LNAME	ADDRESS	TELNO
CO46	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St,Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl,Glasgow G4 0QR	0141-225-7025

OWNER

CLIENT NO	BRANCH NO	STAFF NO	DATEJOINED
CR76	B005	SL41	02-Jan-04
CR56	B003	SG37	11-Apr-03
CR74	B003	SG37	15-Nov-02
CR62	B007	SA9	07-Mar-03

REGISTRATION

CLIENTNO	PROPERTYNO	VIEWDATE	VCOMMENT
CR56	PA14	24-May-04	Too small
CR76	PG4	20-Apr-04	Too remote
CR62	PA14	14-May-04	No dining room
CR56	PG4	26-May-04	-
CR56	PG36	28-Apr-04	-

VIEWING

Division $R \div S$

- The division operation defines a relation over the attributes C that consists of the set of tuples from R that match the combination of every tuple in S.
- $T_1 \leftarrow \pi_c(R)$
- $T_2 \leftarrow \pi_c((SXT_1) - R)$
- $T \leftarrow T_1 - T_2$

Example division

- Identify all clients who have viewed all properties with three rooms.
- We have two relations:
  - Viewing(<u>clientNo</u>, <u>propertyNo*</u>, viewDate, Vcomment)
  - PropertyForRent(<u>propertyNo</u>, street, city, postcode, type, rooms, rent, ownerno, staffNo, branchNo)
- $(\pi_{\text{clientNo, propertyNo}}(\text{Viewing})) \div (\pi_{\text{propertyNo}}(\sigma_{\text{rooms=3}}(\text{PropertyForRent}))$

Continued…

clientNo	propertyNo
CR56	PA14
CR76	PG4
CR56	PG4
CR62	PA14
CR56	PG36

Result
CR56

($\pi_{clientNo,\ propertyNo}$ (Viewing))

propertyNo
PG4
PG36

($\pi_{propertyNo}(\sigma_{rooms=3}$ (PropertyForRent))

All clients who have viewed ALL 3-bedroom houses.

Aggregate and Grouping operations

- $\mathcal{I}_{AL}$ (R) Applies the aggregate function list, AL, to the relation R to define a relation over the aggregate list. AL contains one or more (<aggregate_function>, <attribute>) pairs.

Aggregate functions

- The main aggregate functions are:
  - Count
  - Sum
  - Avg
  - Max
  - Min

Example

- How many properties cost more than €350 per month to rent?
- ρ_{R} (myCount) $\mathcal{I}_{\text{COUNT propertyNo}}$ ($\sigma_{\text{rent} > 350}$ (PropertyForRent))

Grouping operation

- ${}_{GA}\mathcal{I}_{AL}$ (R) groups the tuples of relation R by the grouping attributes GA and then applies the aggregate function list AL to define a new relation. AL contains one or more (<aggregate_function>,<attribute>) pairs. The resulting relation contains the grouping attributes, GA, along with the results of each of the aggregate functions.

Example

- ρ_R(branchNo, myCount, mySum) $_{\text{branchNo}}\mathcal{I}_{\text{COUNT staffNo,SUM salary}}$ (Staff)

Existential and universal quantifiers

- There exists $\exists$
- For All $\forall$
- These functions also use and $\wedge$, not ~, or $\vee$.
- The existential quantifier is used in formulae that must be true for at least one instance.
  - E.g.

$$Staff(S) \wedge (\exists B)(Branch(B) \wedge (B.branchNo = S.branchNo) \wedge B.city = 'London)$$

  - There exists a branch tuple that has the same branchno of the current staff tuple, S, and is located in London.

The Universal qualifier ∀

- This is used in statements about every instance.
- To say:
  - For all Branch tuples, the address is not Paris

$$(\forall B)(B.city \neq 'Paris')$$

DeMorgan's laws

- 1 $(\exists X)(F(X) \equiv\sim (\forall X)(\sim(F(X)))$
- 2 $(\forall X)(F(X) \equiv\sim \exists(X)(\sim\exists(F(X)))$
- 3 $(\exists X)(F_1(X) \wedge F_2(X) \equiv\sim \forall X)(\sim(F_1(X))\vee \sim(F_2(X)))$
- 4 $(\forall X)(F_1(X) \wedge F_2(X)) \equiv\sim (\exists X)(\sim(F_1(X))\vee \sim(F_2(X)))$

DeMorgan's laws

$$(\exists X)(F(X) \equiv \sim (\forall X)(\sim (F(X)))$$

The first one of these means that, if there exist F(X) for X, then it is not true that for all X, there is no F(X)

- There exists a branch in London
- It is not true that for all branches, none of the locations is London.

DeMorgan's Laws

$$(\forall X)(F(X) \equiv\sim \exists(X)(\sim \exists(F(X))))$$

- If something is true for all X, then there is no instance of X for which it is not true.
- If I buy all parts worth > €60, then there does not exist a part worth > €60 that I have not bought.

DeMorgan's Laws

$$(\exists X)(F_1(X) \wedge F_2(X) \equiv \sim \forall X)(\sim (F_1(X)) \vee \sim (F_2(X)))$$

- If there exists an instance of X, where both F1 and F2 are true, then it is not true that for all instances of X F1 is not true or F2 is not true
  - If Paul bought a part worth €20 from John then it is not true that there is no part that john has not sold or that is worth €20.

DeMorgan's Laws

$$(\forall X)(F_1(X) \wedge F_2(X)) \equiv \sim (\exists X)(\sim (F_1(X)) \vee \sim (F_2(X)))$$

- If for all X, F1 and F2 are true, then there does not exist an X where F1 is not true or F2 is not true.
  - If all parts P have a price > 0 and a non null name, there does not exist a part P, where the price <= 0 or the name is null.

Tuple Relational Calculus

- This is based on the use of tuple variables.
- A tuple variable is one that ranges over a named relation.
  - i.e. its only permitted values are tuples of the relation.
  - The range of a tuple variable S as the Staff relation is Staff(S)
  - To find the set of all tuples S such that F(S) is true, we write {S|F(S)}
  - F is called a formula.

Expressions and Formulae in Tuple Relational Calculus

- A general expression of tuple relational calculus is of the form:
- $\{t_1.A_j, t_2.A_k, \ldots t_n.A_m | COND(t_1, t_2, \ldots t_n, t_{n+1}, t_{n+2}, \ldots t_{n+m})\}$
  - Where $t_1, t_2, \ldots t_n, t_{n+1}, t_{n+2}, \ldots t_{n+m}$ are tuple variables,
  - each A_j is an attribute of the relation on which t_i ranges.
  - COND is a condition or formula of the tuple relational calculus.
  - A formula is made up of atoms.

Atoms can be:

- An atom of the form $R(t_i)$ where
  - R is a relation name
  - t_i is a tuple variable.
  - Identifies the range of the tuple variable t_i as the relation whose name is R.
- An atom of the form $t_i.A$ op $t_j.B$, where
  - op is one of {=,<,>,≤,≥,≠},
  - t_i and t_j are tuple variables;
  - A is an attribute of the relation on which t_i ranges,
  - B is an attribute of the relation on which t_j ranges.
- An atom of the form $t_i.A$ op c or c op $t_j.B$,
  - Op is one of {=,<,>,≤,≥,≠},
  - t_i and t_j are tuple variables;
  - A is an attribute of the relation on which t_i ranges,
  - B is an attribute of the relation on which t_j ranges,
  - C is a constant value
- Each of these atoms evaluates to either TRUE or FALSE.

A formula (condition)

- Can be made up of one or more atoms connected via AND, OR and NOT.

SQL terms

- EXISTS
  - Checks for the existence of rows in the sub-query.
  - Find all staff who work in a London branch office.

```
Select staffno,fName, lName, position
FROM staff S
WHERE EXISTS
(SELECT *
 FROM Branch b
  WHERE s.branchNo = b.branchNo AND city =
   'London');
```

  - *Is there another way to do this?*

NOT EXISTS

- Checks for the non-existence of rows in the sub-query.
- As there is no data returned, the attribute list is often *.

IN and NOT IN

- This search condition can refer to a list or a sub-query:

  E.g. SELECT staffNo, fName, lName, position FROM staff

  WHERE position IN (‘Manager’,’Supervisor’);

  How else could you do this?

  E.g.2

  SELECT staffNo, fName, lName, position FROM staff

  WHERE position IN (SELECT position from POSITIONTBL);

DeMorgan's Laws interpreted

- NOT (A AND B) = (NOT A) OR (NOT B)
- NOT (A OR B) = (NOT A) AND (NOT B)

Using the universal quantifier

- E.g. List the names of all employees who work on all the projects controlled by department number 5.

{ *e*.Lname,*e*.Fname|EMPLOYEE(*e*) AND (∀*x*)(NOT(PROJECT(x))

OR NOT (*x*.Dnum = 5) OR ((∃*w*)(WORKS_ON(*w*) AND *w*.Essn=*e*.Ssn AND *x*.Pnumber=*w*.Pno)))} becomes:

{e.Lname, e.Fname | EMPLOYEE(e) AND F'}

F'=((∀x)(NOT(PROJECT(x)) OR F_1))

F_1 = NOT (x.Dnum=5) OR F_2

F_2=((∃w)(WORKS_ON(w) AND w.Essn=e.Ssn

And x.Pnumber=w.Pno

Transforming to the existential quantifier

{ *e*.Lname,*e*.Fname|EMPLOYEE(*e*) AND (NOT(∃ *x*)(PROJECT(x))
AND (*x*.Dnum = 5) AND (NOT(∃ *w*)(WORKS_ON(*w*)
AND *w*.Essn=*e*.Ssn AND *x*.Pnumber=*w*.Pno)))}

- SQL is based on tuple relational calculus.

Domain Relational Calculus

- Or domain calculus
- Uses different types of variables
- They range over values from domains of attributes, rather than tuples.
- An expression is of the form

$$\{x_1, x_2,\ldots,x_n|COND(x_1, x_2,\ldots, x_n,x_{n+1},x_{n+2},\ldots, x_{n+m}\}$$