Student-course-professor details

Sid	Sname	cid	Cname	Grade	Cloc	Sec	Pid	pname
1	X	31	Java	A	C1	1	P4	John
1	X	45	Oracle	B	C2	1	P6	David
5	Z	45	Oracle	A	C2	1	P6	David

Student details

<u>**Sid**</u>	Sname
1	X
5	Z

Course details

<u>cid</u>	Cname	Cloc
31	Java	C1
45	Oracle	C2

Professor details

<u>Pid</u>	pname
P4	John
P6	David

Student-course details

<u>**Sid**</u>	<u>**cid**</u>	Grade
1	31	A
1	45	B
5	45	A

Course-professor details

cid	**Sec**	Pid
31	1	P4
45	1	P6

BCNF(Boyce Codd Normal Form)

- A relation is said to be in Boyce-Codd Normal Form if all its FDs are either trivial FDs or key FDs.

A FD X→Y is said to be trivial if Y $\subseteq$ X

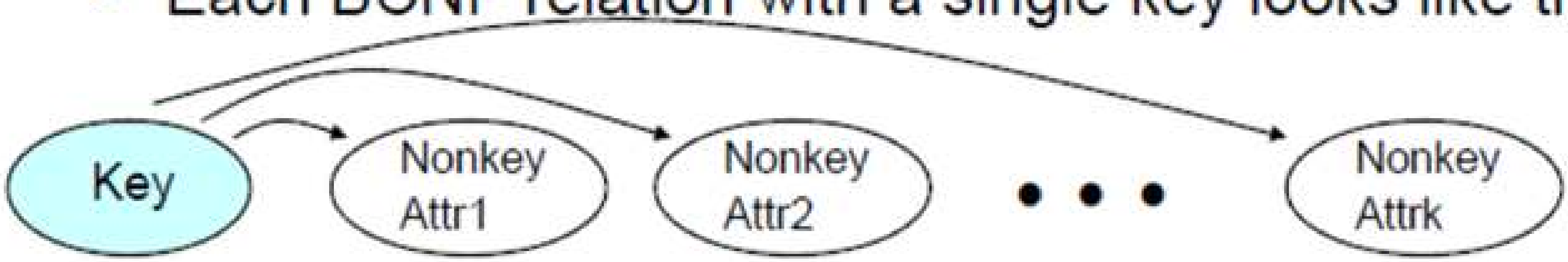
- Which of these relations is BCNF?

  EmpDept(<u>EID</u>, Name, DeptName)

  Assigned(<u>EmpID, JobID</u>, EmpName, percent)

  EnrollStud(<u>StudID, ClassID</u>, grade)

- Each BCNF relation with a single key looks like this



BCNF(Boyce Codd Normal Form)

- Here is an algorithm for decomposing an arbitrary relation R into a collection of BCNF relations:

1. If R is not in BCNF and **X→A** is a non-key FD, then decompose R into R – A and XA.
2. If R – A and/or XA is not in BCNF, recursively apply step 1.

i.e;

If AB→C and C→B then

Decompose the table into relations having columns (A,C) and (C,B).

Decomposing to BCNF

- Given the schema

  EnrollStud(<u>StudID, ClassID</u>, Grade, ProfID, StudName)

including its natural FDs, decompose it into BCNF relations.

Begin with the non-key FD **StudID→StudName**. This results in the decomposition

EnrollProf(<u>StudID,ClassID</u>,Grade,ProfID) Stud(<u>StudID</u>,StudName)

Stud is BCNF, but EnrollProf is not. The FD **ClassID→ProfID** gives the further decomposition

Enroll(<u>StudID,ClassID</u>,Grade)

Prof(<u>ClassID</u>,ProfID)

Stud(<u>StudID</u>,StudName)

in which all schemas are BCNF.

We could have begun with the FD **ClassID→ProfID** and first got EnrollStud1(<u>StudID, ClassID</u>, Grade, StudName), Prof(<u>ClassID</u>,ProfID)

Then apply the FD **StudID→StudName** to EnrollStud1 and get the same BCNF tables as above

BCNF example

courseno,subjname→profno. And profno→subjname

Q.Display the professor's name who teaches OS.

Prob: P4 is displayed twice.

Q.List the subjects taken by prof P4.

Prob: OS is displayed twice.

Course no.	**Subj name**	Prof no.
FE	Chem	P1
FE	Phy	P5
SE	OS	P4
TE	OS	P4
SE	DB	P3
TE	ADB	P2

BCNF example

Course no.	Prof no.
FE	P1
FE	P5
SE	P4
SE	P3
TE	P4
TE	P2

BCNF example

Prof no.	Subj name
P1	Chem
P2	ADB
P3	DB
P4	OS
P5	Phy

4NF(no multivalued dependencies) author→→subject and subject→→bookpublisher

List the authors working with TMH

Prob: Silberschatz is displayed twice.

List the subjects published by TMH

Prob:Database systems is displayed thrice

Display the publisher of silberschatz.

Prob:TMH is displayed twice.

<u>Subject</u>	**<u>Author</u>**	Book publisher
Database systems	Korth	Tata McGraw Hill
Database systems	Sudarshan	Tata McGraw Hill
Database systems	Silberschatz	Tata McGraw Hill
Database systems	Elmarsi	Pearson Education
Database systems	Navathe	Pearson Education
Operating systems	Silberschatz	Tata McGraw Hill

4NF(no multivalued dependencies)

Subject	**Author**
Database systems	Korth
Database systems	Sudarshan
Database systems	Silberschatz
Database systems	Elmarsi
Database systems	Navathe
Operating systems	Silberschatz

4NF(no multivalued dependencies)

Subject	**Book publisher**
Database systems	Tata McGraw Hill
Database systems	Pearson Education
Operating systems	Tata McGraw Hill

4NF(no multivalued dependencies)

Author	Book publisher
Korth	Tata McGraw Hill
Sudarshan	Tata McGraw Hill
Silberschatz	Tata McGraw Hill
Elmarsi	Pearson Education
Navathe	Pearson Education

Dependent_Relationship) and ... Dependent_Relationship) ... any
the attributes of (Dependent_Name, ... {EMPLOYEE – Employee_Name – Depen-
way to the values of the attributes in {EMPLOYEE

Figure 7.1 Unnormalized EMPLOYEE relation.

Employee_ Name	*Dependent*		*Positions*		*Home_ City*	*Home_ Phone*
	Name	*Relationship*	*Title*	*Date*		
Jill Jones	Bill Jones	spouse	J. Engineer	05/12/84	Lynn, MA	794-2356
			Engineer	10/06/86		
	Bob Jones	son	J. Engineer	05/12/84		
			Engineer	10/06/86		
Mark Smith	Ann Briggs	spouse	Programmer	09/15/83	Revere, MA	452-4729
			Analyst	06/06/86		
	Chloe Smith-Briggs	daughter	Programmer	09/15/83		
			Analyst	09/06/86		
	Mark Briggs-Smith	son	Programmer	09/15/83		
			Analyst	09/06/86		

Figure 7.2 Normalized EMPLOYEE relation.

Employee_ Name	*Dependent_ Name*	*Dependent_ Relationship*	*Position_ Title*	*Position_ Date*	*Home_ City*	*Home_ Phone#*
Jill Jones	Bill Jones	spouse	J. Engineer	05/12/84	Lynn, MA	794-2356
Jill Jones	Bill Jones	spouse	Engineer	10/06/86	Lynn, MA	794-2356
Jill Jones	Bob Jones	son	J. Engineer	05/12/84	Lynn, MA	794-2356
Jill Jones	Bob Jones	son	Engineer	19/06/86	Lynn, MA	794-2356
Mark Smith	Ann Briggs	spouse	Programmer	09/15/83	Revere, MA	452-4729
Mark Smith	Ann Briggs	spouse	Analyst	06/06/86	Revere, MA	45204729
Mark Smith	Chloe Smith-Briggs	daughter	Programmer	09/15/83	Revere, MA	452-4729
Mark Smith	Chloe Smith-Briggs	daughter	Analyst	06/06/86	Revere, MA	452-4729
Mark Smith	Mark Briggs-Smith	son	Programmer	09/15/83	Revere, MA	452-4729
Mark Smith	Mark Briggs-Smith	son	Analyst	06/06/86	Revere, MA	452-4729

dent). Similarly, the set of values for the attributes of (*Position_Title, Position_Date*)

Figure 7.4 Replacing the EMPLOYEE relation with three relations.

Employee_Name	*Dependent_Name*	*Dependent_Relationship*
Jill Jones	Bill Jones	spouse
Jill Jones	Bob Jones	son
Mark Smith	Ann Briggs	spouse
Mark Smith	Chloe Smith-Briggs	daughter
Mark Smith	Mark Briggs-Smith	son

Employee_Name	*Position_Title*	*Position_Date*
Jill Jones	J. Engineer	05/12/84
Jill Jones	Engineer	10/06/86
Mark Smith	Programmer	09/15/83
Mark Smith	Analyst	06/06/86

Employee_Name	*Home_City*	*Home_Phone#*
Jill Jones	Lynn, MA	794-2356
Mark Smith	Revere, MA	452-4729

for the EMPLOYEE relation of Figure 7.2.[1] Such a scheme avoids the necessity of multiple storage of the same information.

tional but multivalued. Similarly, the dependency between a course and a day is not simply fun... which it meets is multivalued. ... the dependency between a course and the room

These multivalued dependencies can be indicated as follows:

$$Course \twoheadrightarrow RoomDayTime$$

The SCHEDULE relation.

Prof	*Course*	*Room*	*Max_Enrollment*	*Day*	*Time*
Smith	353	A532	40	mon	1145
Smith	353	A534	40	wed	1245
Clark	355	H942	300	tue	115
Clark	355	H940	300	thu	115
Turner	456	B278	45	mon	845
Turner	456	B279	45	wed	845
Jamieson	459	D111	45	tue	1015
Jamieson	459	D110	45	thu	1015

Definition of Decomposition

Let R be a relation schema

A set of relation schemas { R1, R2,…, Rn } is a decomposition of R if

- R = R1 U R2 U …..U Rn
- each Ri is a subset of R (for i = 1,2…,n)

Example of Decomposition

For relation R(x,y,z) there can be 2 subsets:

R1(x,z) and R2(y,z)

If we union R1 and R2, we get R

R = R1 U R2

Goal of Decomposition

- Eliminate redundancy by decomposing a relation into several relations in a higher normal form.
- It is important to check that a decomposition does not lead to bad design

Decomposition

- Consider our original "bad" attribute set

Stuff(<u>sid</u>, name, <u>serno</u>, subj, cid, exp-grade)

- We could decompose it into

Student(<u>sid</u>, name)
Course(<u>serno</u>, cid)
Subject(<u>cid</u>, subj)

- But this decomposition loses information about the relationship between students and courses. Why?

Lossless Join Decomposition

$R_1, \ldots R_k$ is a *lossless join decomposition* of R w.r.t. an FD set F if for every instance r of R that satisfies F,

$$\Pi_{R_1}(r) \bowtie \ldots \bowtie \Pi_{R_k}(r) = r$$

Consider:

sid	name	serno	subj	cid	exp-grade
1	Sam	570103	AI	570	B
23	Nitin	550103	DB	550	A

What if we decompose on

(sid, name) and (serno, subj, cid, exp-grade)?

Example of Lossy Decomposition

S	P	D
S1	P1	D1
S2	P2	D2
S3	P1	D3

instance r

S	P
S1	P1
S2	P2
S3	P1

$\pi SP(r)$

P	D
P1	D1
P2	D2
P1	D3

$\pi PD(r)$

S	P	D
S1	P1	D1
S2	P2	D2
S3	P1	D3
S1	P1	D3
S3	P1	D1

$\pi SP(r) \times \pi PD(r$

Problem with Decomposition

- Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation – information loss

Lossy decomposition

- In previous example, additional tuples are obtained along with original tuples
- Although there are more tuples, this leads to less information
- Due to the loss of information, decomposition for previous example is called lossy decomposition or lossy-join decomposition

Lossless Decomposition

A decomposition {R1, R2,…, Rn} of a relation R is called a lossless decomposition for R if the natural join of R1, R2,…, Rn produces exactly the relation R.

Lossless Decomposition

A decomposition is lossless if we can recover:

R(A, B, C)

Decompose

R1(A, B) R2(A, C)

Recover

R'(A, B, C)

Thus, R' = R

Lossless Decomposition Property

R : relation

F : set of functional dependencies on R

X,Y : decomposition of R

Decomposition is lossles if :

- $X \cap Y \rightarrow X$, that is: all attributes common to both X and Y functionally determine ALL the attributes in X **OR**
- $X \cap Y \rightarrow Y$, that is: all attributes common to both X and Y functionally determine ALL the attributes in Y

Testing for Lossless Join

R_1, R_2 is a lossless join decomposition of R with respect to *F* iff *at least one* of the following dependencies is in *F+*

$$(R_1 \cap R_2) \rightarrow R_1$$
$$(R_1 \cap R_2) \rightarrow R_2$$

So for the FD set:

sid → name

serno → cid, exp-grade

cid → subj

Is (sid, name) and (sid, serno, subj, cid, exp-grade) a lossless decomposition?

Lossless Decomposition Property

- In other words, if $X \cap Y$ forms a superkey of either X or Y, the decomposition of R is a lossless decomposition

Algorithm 15.2 Testing for the lossless (nonadditive) join property

Input: A universal relation R, a decomposition $D = \{R_1, R_2, ..., R_m\}$ of R, and a set F of functional dependencies.

1. Create an initial matrix S with one row i for each relation R_i in D, and one column j for each attribute A_j in R.
2. Set $S(i,j) := b_{ij}$ for all matrix entries.
   (* each b_{ij} is a distinct symbol associated with indices (i,j) *)
3. For each row i representing relation schema R_i
   {for each column j representing attribute A_j
   {if (relation R_i includes attribute A_j) then set $S(i,j) := a_j$;};};
   (* each a_j is a distinct symbol associated with index (j) *)
4. Repeat the following loop until a *complete loop execution* results in no changes to S
   {for each functional dependency $X \rightarrow Y$ in F
   {for all rows in S *which have the same symbols* in the columns corresponding to attributes in X
   {make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows: if any of the rows has an "a" symbol for the column, set the other rows to that *same* "a" symbol in the column. If no "a" symbol exists for the attribute in any of the rows, choose one of the "b" symbols that appear in one of the rows for the attribute and set the other rows to that same "b" symbol in the column ;};};};
5. If a row is made up entirely of "a" symbols, then the decomposition has the lossless join property; otherwise it does not.

Example

R1 (A1, A2, A3, A5)

R2 (A1, A3, A4)

R3 (A4, A5)

FD1: A1 $\rightarrow$ A3 A5

FD2: A5 $\rightarrow$ A1 A4

FD3: A3 A4 $\rightarrow$ A2

Example (con't)

	A1	A2	A3	A4	A5
R1	a(1)	a(2)	a(3)	b(1,4)	a(5)
R2	a(1)	b(2,2)	a(3)	a(4)	b(2,5)
R3	b(3,1)	b(3,2)	b(3,3)	a(4)	a(5)

Example (con't)

By FD1: A1 → A3 A5

	A1	A2	A3	A4	A5
R1	a(1)	a(2)	a(3)	b(1,4)	a(5)
R2	a(1)	b(2,2)	a(3)	a(4)	b(2,5)
R3	b(3,1)	b(3,2)	b(3,3)	a(4)	a(5)

Example (con't)

By FD1: A1 → A3 A5

we have a new result table

	A1	A2	A3	A4	A5
R1	a(1)	a(2)	a(3)	b(1,4)	a(5)
R2	a(1)	b(2,2)	a(3)	a(4)	a(5)
R3	b(3,1)	b(3,2)	b(3,3)	a(4)	a(5)

Example (con't)

By FD2: A5 → A1 A4

	A1	A2	A3	A4	A5
R1	a(1)	a(2)	a(3)	b(1,4)	a(5)
R2	a(1)	b(2,2)	a(3)	a(4)	a(5)
R3	b(3,1)	b(3,2)	b(3,3)	a(4)	a(5)

Example (con't)

FD2: A5 → A1 A4

we have a new result table

	A1	A2	A3	A4	A5
R1	a(1)	a(2)	a(3)	a(4)	a(5)
R2	a(1)	b(2,2)	a(3)	a(4)	a(5)
R3	a(1)	b(3,2)	b(3,3)	a(4)	a(5)

Conclusions

- Decompositions should always be lossless

  Lossless decomposition ensure that the information in the original relation can be accurately reconstructed based on the information represented in the decomposed relations.

Dependency Preservation

Ensures we can "easily" check whether a FD $X \rightarrow Y$ is violated during an update to a database:

- The *projection* of an FD set F onto a set of attributes Z, F_Z is

$$\{X \rightarrow Y \mid X \rightarrow Y \in F^+, X \cup Y \subseteq Z\}$$

  i.e., it is those FDs local to Z's attributes

- A decomposition $R_1, \ldots, R_k$ is *dependency preserving* if $F^+ = (F_{R_1} \cup \ldots \cup F_{R_k})^+$

The decomposition hasn't "lost" any essential FD's, so we can check without doing a join

Example of Lossless and Dependency-Preserving Decompositions

Given relation scheme

R(name, street, city, st, pin, item, price)

And FD set
name → street, city
street, city → st
street, city → pin
name, item → price

Consider the decomposition

R_1(name, street, city, st, pin) and R_2(name, item, price)

- *Is it lossless?*
- *Is it dependency preserving?*

What if we replaced the first FD by name, street → city?

ss and Dependency-Preserving
Decomposition

s algorithm we assume that we have a canonical cover $\mathbf{F_c}$ for the set of FDs he relation scheme **R** and that **K** is a candidate key of **R.** Algorithm 6.6 es a decomposition of **R** into a collection of relation schemes $\mathbf{R_1}, \mathbf{R_2}, \ldots,$ ch relation scheme $\mathbf{R_i}$ is in third normal form with respect to the projection of the scheme of **R**.

Example 6.31 below, we give a decomposition into 3NF relation schemes s both lossless and also dependency-preserving.

Find a lossless join and dependency-preserving decomposition of the following relation scheme with the given set of functional dependencies:

SHIPPING *(Ship, Capacity, Date, Cargo, Value)*
Ship $\rightarrow$ *Capacity,*
ShipDate $\rightarrow$ *Cargo,*
CargoCapacity $\rightarrow$ *Value*

Now use Algorithm 6.7 to find a lossless decomposition of **SHI
ING.** Since there is an FD *Ship* → *Capacity* and since *Ship* ↛ **SHIPPIN**
e replace **SHIPPING** with the relation $\mathbf{R}_1$ *(Ship, Capacity)* formed wi
e FD in question and $\mathbf{R}_2$*(Ship, Date, Cargo, Value)*. Consider the relati
$_2$: the FD *ShipDate* → *Cargo* is a nontrivial FD in the nonredundant cove
owever, since *ShipDate* → *ShipDateCargoValue*, the relation $\mathbf{R}_2$ is
CNF and we have completed the decomposition.

$\mathbf{R}_1$*(Ship, Capacity)* with the FD *Ship* → *Capacity*
$\mathbf{R}_2$*(Ship, Date, Cargo, Value)* with the FD *ShipDate* → *Cargo*

he decomposition of **SHIPPING** into $\mathbf{R}_1$ and $\mathbf{R}_2$ is lossless but not depe
ency preserving because the FD *CargoCapacity* → *Value* is not implied
e set of FDs {*Ship* → *Capacity*, *ShipDate* → *Cargo*}.

Another BCNF decomposition of **SHIPPING** is obtained when we co
der the FD *CargoCapacity* → *Value* first. This gives us the followi

ant FDs in the ...
e left-hand side. Hence the given set of FDs is ...
idate key of the relation is *ShipDate*.

Now use Algorithm 6.6 to find a lossless and depend
ecomposition of **SHIPPING.** Since all attributes appear i
over we need not form a relation for attributes not appear
here is no single FD in the canonical cover that contain
ttributes in **SHIPPING,** so we proceed to form a relation
he canonical cover.

$\mathbf{R}_1$(*Ship, Capacity*) with the FD *Ship* → *Capacity*
$\mathbf{R}_2$(*Ship, Date, Cargo*) with the FD *ShipDate* → *Cargo*
$\mathbf{R}_3$(*Cargo, Capacity, Value*) with the FD *CargoCapaci*

As a candidate key is included in the determinant of the FD
osed relation scheme $\mathbf{R}_2$, we need not include another relati
only a candidate key. The decomposition of SHIP...

Example of Decompositions in BCNF but not Dependency-Preserving

Given relation scheme

R(sailorid, boatid, date)

Sailorid, boatid->date

(a sailor can reserve a boat for atmost one day)

Date->boatid

(on a give day at most one boat can be reserved)

We cannot decompose into

(sailorid, date) and (date, boatid) since date is not a key.

Thus R is not in BCNF

Since the decomposition do not preserve the dependency

Sailorid, boatid->date

Normal Forms Compared

- BCNF is preferable, but sometimes in conflict with the goal of dependency preservation
  - It's strictly stronger than 3NF
  - BCNF : lossless join decomposition
  - 3NF :lossless join, dependency preserving decomposition

Summary

- We can always decompose into 3NF and get:
  - Lossless join
  - Dependency preservation
- But with BCNF we are only guaranteed lossless joins
- BCNF is stronger than 3NF: every BCNF schema is also in 3NF
- The BCNF algorithm is nondeterministic, so there is not a unique decomposition for a given schema R

5NF(Projection join normal form)

q1.Display the list of employees having project p2.
prob: E2 is displayed twice.
q2. Display the projects under the manager M1
prob:P1 is displayed twice.
q3. Display the managers of project P2.
prob: M2 is displayed twice.
q4.Display the employees working under M1
prob:E3 is displayed twice

Employee no.	**Project no.**	**Manager no.**
E1	P1	M1
E1	P2	M2
E2	P2	M2
E2	P2	M4
E3	P1	M1
E3	P3	M1

5NF(Projection join normal form)

<u>Employee no.</u>	**<u>Project no.</u>**
E1	P1
E1	P2
E2	P2
E3	P1
E3	P3

5NF(Projection join normal form)

Project no.	**Manager no.**
P1	M1
P2	M2
P2	M4
P3	M1

5NF(Projection join normal form)

Employee no.	**Manager no.**
E1	M1
E1	M2
E2	M2
E2	M4
E3	M1

PROJECT_ASSIGNMENT relation.

Employee	*Project*	*Expertise*
Smith	Query Systems	Database Systems
Smith	File systems	Operating Systems
Lalonde	Database Machine	Computer Architecture
Lalonde	Database Machine	VLSI Technology
Evan	Database Machine	VLSI Technology
Evan	Database Machine	Computer Architecture
Drew	SQL + +	Relational Calculus
Drew	QUEL + +	Relational Calculus
Shah	SQL + +	Relational Calculus
Shah	QUEL +; +	Relational Calculus

Figure 7.7 Lossless decomposition of relation of Figure 7.6: (a) PROJECT_REQUIREMENT and (b) PROJECT_PREFERENCE.

Project	*Expertise*
Query Systems	Database Systems
File Systems	Operating Systems
Database Machine	Computer Architecture
Database Machine	VLSI Technology
SQL + +	Relational Calculus
QUEL + +	Relational Calculus

(a)

Employee	*Project*
Smith	Query Systems
Smith	File systems
Evan	Database Machine
Lalonde	Database Machine
Drew	SQL + +
Shah	QUEL + +
Drew	SQL + +
Shah	QUEL + +

(b)

..., can be carried out by B

NEW_PROJECT_ASSIGNMENT relation.

Employee	*Project*	*Expertise*
Brent	Work Station	User Interface
Brent	Work Station	Artificial Intelligence
Mann	Work Station	VLSI Technology
Smith	Work Station	Operating Systems
King	SQL 2	Relational Calculus
Ito	SQL 2	Relational Algebra
Ito	QBE++	Relational Calculus
Smith	Query Systems	Database Systems
Smith	File Systems	Operating Systems

7.9 Decomposition of relation of Figure 7.8.

Project	*Expertise*
Work Station	User interface
Work Station	Artificial Intelligence
Work Station	VLSI Technology
Work Station	Operating Systems
SQL 2	Relational Calculus
SQL 2	Relational Algebra
QBE++	Relational Calculus
Query Systems	Database Systems
File Systems	Operating Systems

(a)

Employee	*Expertise*
Brent	User Interface
Brent	Artificial Intelligence
Mann	VLSI Technology
King	Relational Calculus
Ito	Relational Algebra
Ito	Relational Calculus
Smith	Database Systems
Smith	Operating Systems

(b)

Employee	*Project*
Brent	Work Station
Mann	Work Station
King	SQL 2
Ito	SQL 2
Ito	QBE ++
Smith	File Systems
Smith	Query Systems
Smith	Work Station

(c)