Appendices

P.J. McBrien

Imperial College London

Appendix 1: RDF Data Model and OWL Schemas

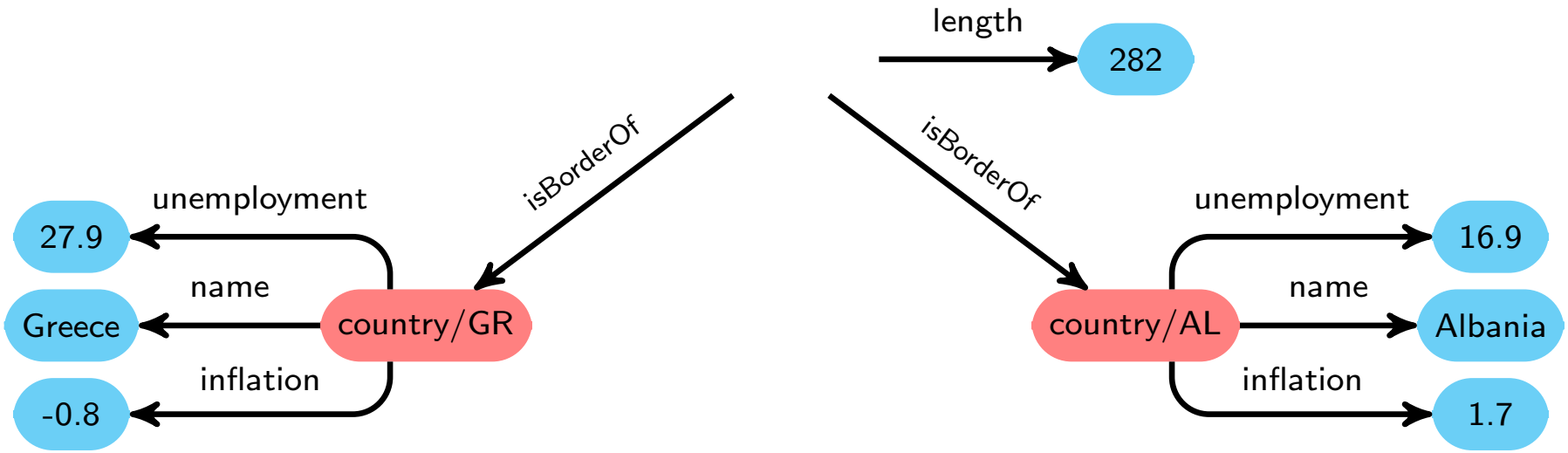
P.J. McBrien

Imperial College London

RDF Graph Model

Triples

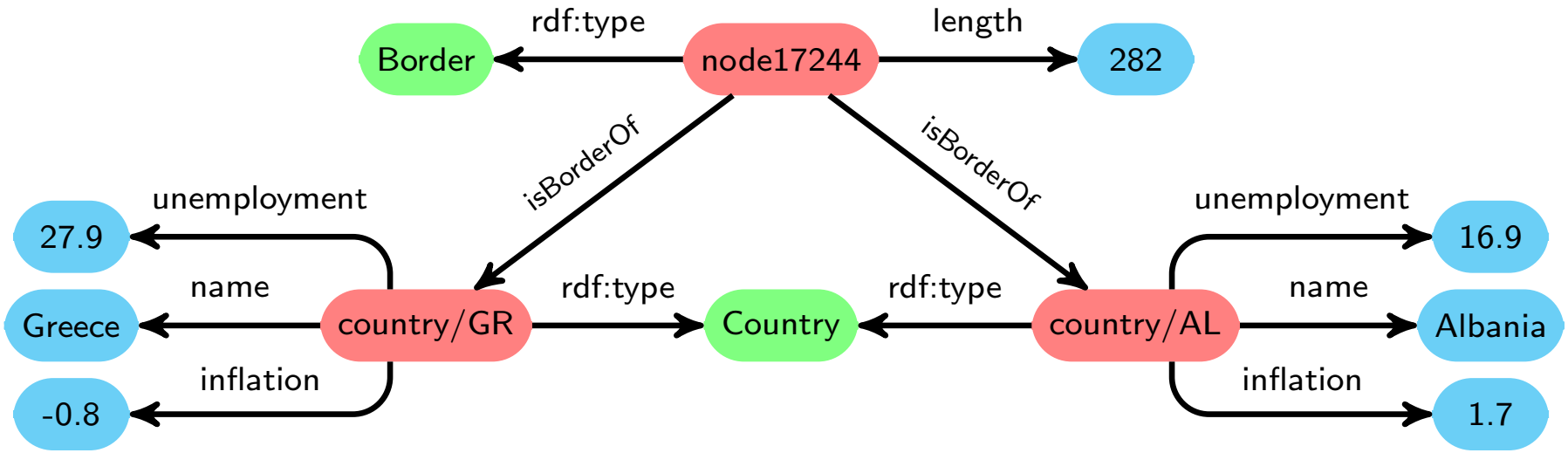
Resource Description Framework (RDF):
data (nodes) are connected by **properties** (directed edges)
A **triple** is the combination of two data items linked by a property.



RDF Graph Model

Triples

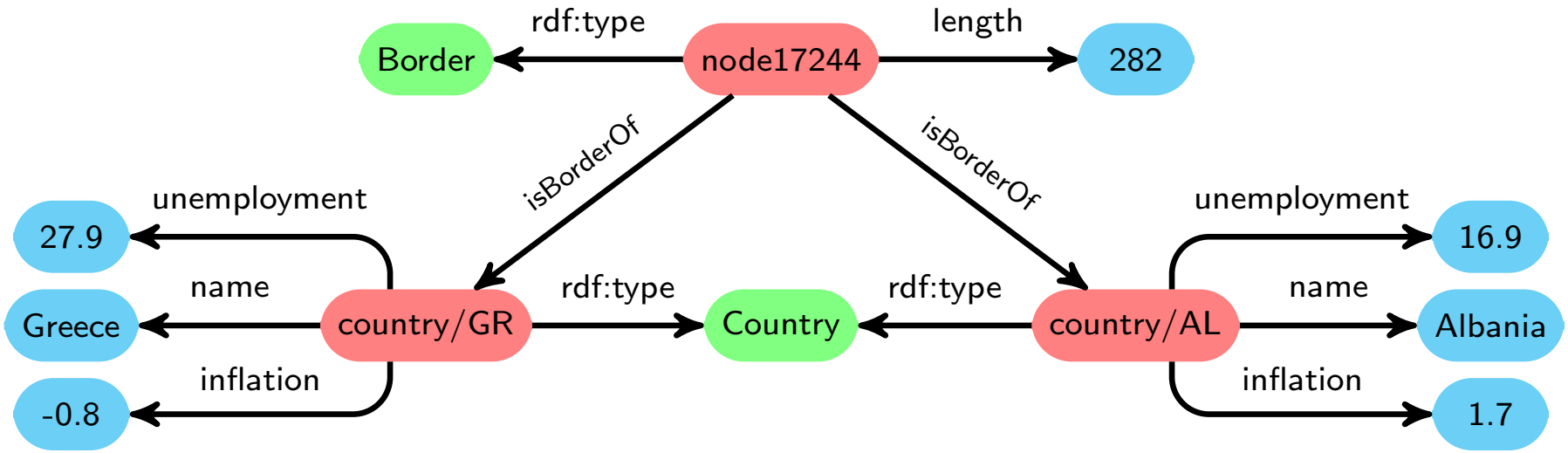
Resource Description Framework (RDF):
data (nodes) are connected by **properties** (directed edges)
A **triple** is the combination of two data items linked by a property.



RDF Graph Model

Triples

Resource Description Framework (RDF):
data (nodes) are connected by **properties** (directed edges)
A **triple** is the combination of two data items linked by a property.



Turtle/N3 representation of RDF

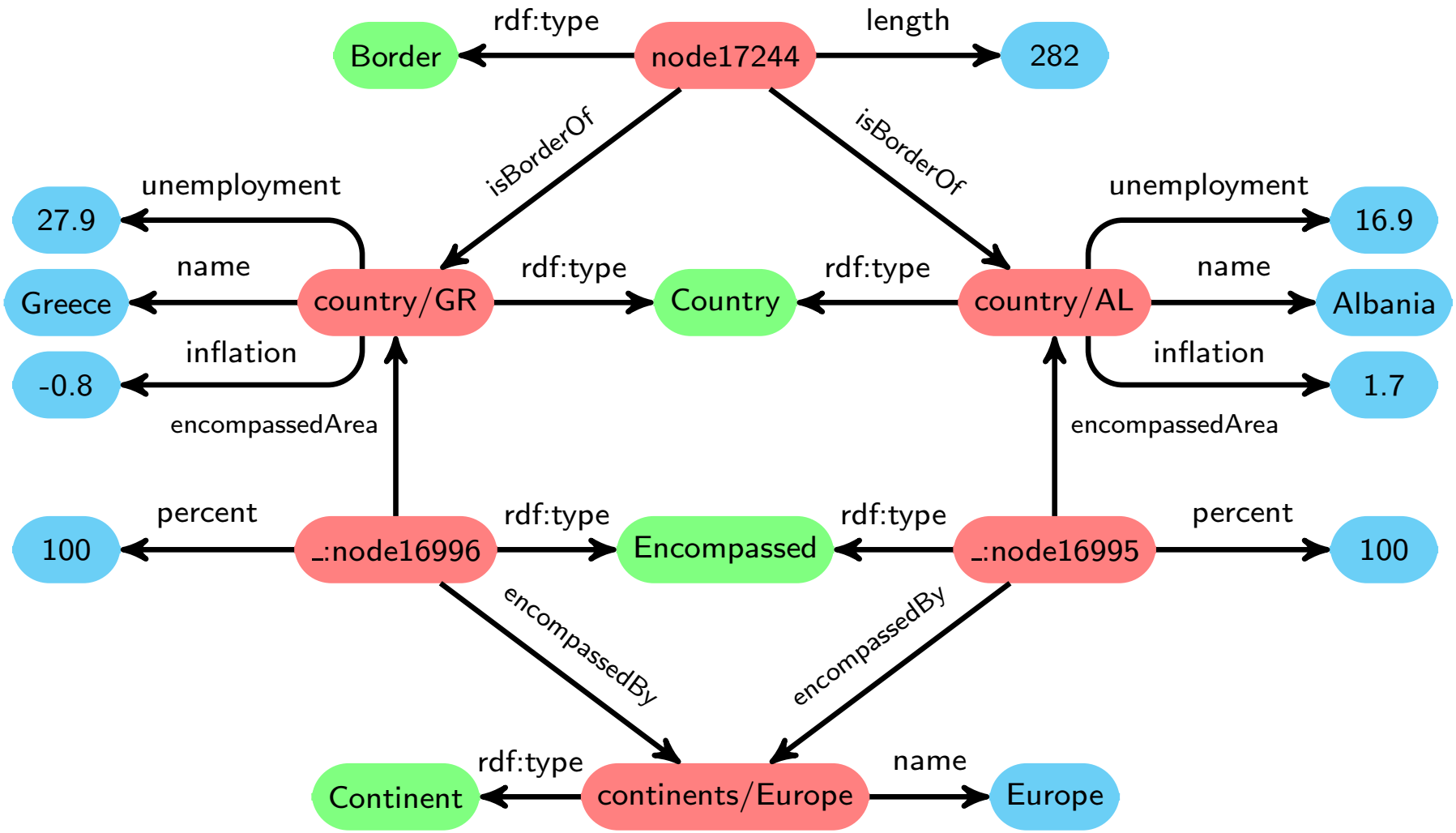
Listing properties of a node

```
<country/AL> rdf:type :Country ;
  :name "Albania" ;
  :inflation 1.7 ;
  :unemployment 16.9 .
```

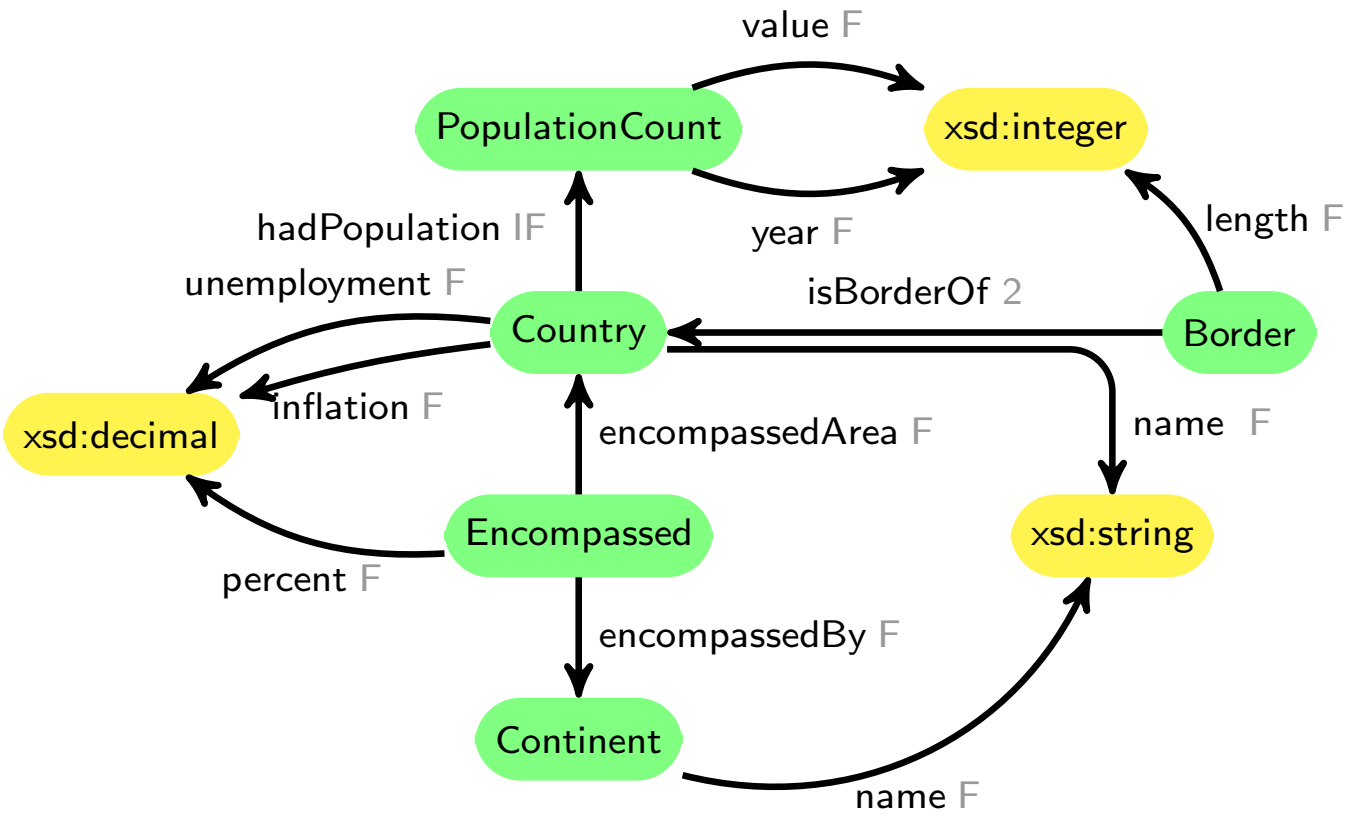
Listing properties of an anonymous node

```
[ rdf:type :Border ;
  :length 282 ;
  :isBorderOf <country/AL> ,
              <country/GR> ] .
```

Example RDF Graph Model



OWL Schemas



Web Ontology Language (OWL)

Allows more schema information to be specified

- Properties can be **functional**, **inverse functional**, or have **cardinality**
- Class properties link to classes
- Data properties link a class to a datatype.

Appendix 2: SPARQL Query Language for RDF Data

P.J. McBrien

Imperial College London

SPARQL

```
SELECT ?country ?inflation ?unemployment
WHERE {
   ?c :name ?country ;
      :inflation ?inflation ;
      :unemployment ?unemployment
}
ORDER BY ?country
```

SPARQL query

Basis of SPARQL query is a SELECT...WHERE statement

- WHERE contains a Turtle pattern; with variables prefixed by ?
- SELECT lists variables

SPARQL

```
SELECT ?country ?inflation ?unemployment
WHERE {
   ?c rdf:type :Country;
      :name ?country ;
      :inflation ?inflation ;
      :unemployment ?unemployment
}
ORDER BY ?country
```

SPARQL query

Basis of SPARQL query is a SELECT...WHERE statement

- WHERE contains a Turtle pattern; with variables prefixed by ?
- SELECT lists variables

Better to restrict the schema of the nodes matched.

SPARQL: Querying more than one class

```
SELECT  ?country ?continent ?pc
WHERE {
     ?e rdf:type :Encompassed ;
        :encompassedArea ?c ;
        :encompassedBy ?con ;
        :percent ?pc .
     ?c rdf:type :Country ;
        :name ?country .
     ?con rdf:type :Continent ;
        :name ?continent .
}
ORDER BY ?country ?continent
```

SPARQL: Use BIND to perform calculations.

```
SELECT  ?country ?continent ?pc ?area
WHERE {
     ?e rdf:type :Encompassed ;
        :encompassedArea ?c ;
        :encompassedBy ?con ;
        :percent ?pc .
     ?c rdf:type :Country ;
        :name ?country ;
        :area ?totalarea .
     ?con rdf:type :Continent ;
        :name ?continent .
     BIND ( ?pc * ?totalarea / 100.0 AS ?area )
}
ORDER BY ?country ?continent
```

SPARQL: Care when properties are not functional

```
SELECT ?country ?length ?neighbour
WHERE {
  ?c rdf:type :Country;
     :name ?country ;
     :inflation ?inflation ;
     :unemployment ?unemployment .
  ?b rdf:type :Border;
     :isBorderOf ?c ;
     :isBorderOf ?n ;
     :length ?length .
  ?n rdf:type :Country;
     :name ?neighbour .
  FILTER (?c = ?n)ORDER BY ?country DESC(?length)
```

Appendix 3: SQL: Rules

P.J. McBrien

Imperial College London

Mondial Database: borders and encompasses Tables

encompasses		
country	continent	percentage
A	Europe	100
AL	Europe	100
AND	Europe	100
B	Europe	100
CH	Europe	100
CY	Europe	100
CZ	Europe	100
D	Europe	100
DK	Europe	100
E	Europe	100
F	Europe	100
GB	Europe	100
GR	Europe	100
I	Europe	100
IRL	Europe	100
L	Europe	100
LT	Europe	100
LV	Europe	100
M	Europe	100
N	Europe	100
NL	Europe	100
P	Europe	100
PL	Europe	100
R	Asia	80
R	Europe	20
RSM	Europe	100
S	Europe	100
SF	Europe	100
SK	Europe	100
TR	Asia	68
TR	Europe	32
V	Europe	100

borders		
country1	country2	length
A	H	366.0
A	I	430.0
A	D	784.0
AND	E	65.0
AND	F	60.0
B	NL	450.0
B	F	620.0
B	D	167.0
CH	D	334.0
CH	F	573.0
CH	I	740.0
CH	A	164.0
D	F	451.0
DK	D	68.0
E	F	623.0
FL	A	37.0
FL	CH	41.0
I	F	488.0
IRL	GB	360.0
L	B	148.0
L	D	138.0
L	F	73.0
MC	F	4.4
N	R	167.0
N	S	1619.0
N	SF	729.0
NL	D	577.0
P	E	1214.0
RSM	I	39.0
SLO	I	235.0
SYR	TR	822.0
V	I	3.2

Mondial Database: borders and encompasses Tables

encompasses		
country	continent	percentage
A	Europe	100
AL	Europe	100
AND	Europe	100
B	Europe	100
CH	Europe	100
CY	Europe	100
CZ	Europe	100
D	Europe	100
DK	Europe	100
E	Europe	100
F	Europe	100
GB	Europe	100
GR	Europe	100
I	Europe	100
IRL	Europe	100
L	Europe	100
LT	Europe	100
LV	Europe	100
M	Europe	100
N	Europe	100
NL	Europe	100
P	Europe	100
PL	Europe	100
R	Asia	80
R	Europe	20
RSM	Europe	100
S	Europe	100
SF	Europe	100
SK	Europe	100
TR	Asia	68
TR	Europe	32
V	Europe	100

borders		
country1	country2	length
A	H	366.0
A	I	430.0
A	D	784.0
AND	E	65.0
AND	F	60.0
B	NL	450.0
B	F	620.0
B	D	167.0
CH	D	334.0
CH	F	573.0
CH	I	740.0
CH	A	164.0
D	F	451.0
DK	D	68.0
E	F	623.0
FL	A	37.0
FL	CH	41.0
I	F	488.0
IRL	GB	360.0
L	B	148.0
L	D	138.0
L	F	73.0
MC	F	4.4
N	R	167.0
N	S	1619.0
N	SF	729.0
NL	D	577.0
P	E	1214.0
RSM	I	39.0
SLO	I	235.0
SYR	TR	822.0
V	I	3.2

Countries bordering Italy

```
SELECT borders.*
FROM   borders
WHERE  'I' IN (country1,country2)
```

country1	country2	length
A	I	430.0
I	F	488.0
SLO	I	235.0
CH	I	740.0
V	I	3.2
RSM	I	39.0

Common Table Expressions

Common Table Expressions

The WITH keyword allows a table to be defined that can be used multiple times in a SELECT query.

Finding countries with a common neighbour

```
WITH    uo_border(country1,country2) AS
        (SELECT country1,country2
         FROM   borders
         UNION ALL
         SELECT country2,country1
         FROM   borders
        )
SELECT  uo_border.country1,
        neighbour.country2
FROM    uo_border
        JOIN uo_border AS neighbour
        ON   uo_border.country2=neighbour.country1
WHERE   NOT uo_border.country1=neighbour.country2
ORDER BY uo_border.country1,
        neighbour.country2
```

country1	country2
E	AND
E	B
E	CH
E	D
E	F
E	I
E	L
E	MC
I	A
I	AND
I	B
I	CH
I	CZ
I	D
I	E
I	F
I	FL
I	H
I	HR
I	L
I	MC
I	SK
I	SLO
	.
	.

Recursive SQL

WITH RECURSIVE allows a table to refer to itself

Finding all countries reachable via land borders

```
WITH RECURSIVE
        uo_border(country1,country2) AS
        (SELECT country1,country2
         FROM   borders
         UNION ALL
         SELECT country2,country1
         FROM   borders
        ),
        land_link(country1,country2) AS
        (SELECT *
         FROM   uo_border
         UNION
         SELECT land_link.country1,
                uo_border.country2
         FROM   land_link
                JOIN uo_border
                ON   land_link.country2=uo_border.country1
                AND  NOT land_link.country1=uo_border.country2
        )
SELECT *
FROM   land_link
ORDER BY country1,country2
```

reachable by land

country1	country2
DK	A
DK	AFG
DK	AL
DK	AND
DK	ANG
DK	ARM
DK	AZ
DK	B
⋮	⋮
GB	IRL
⋮	⋮
IRL	GB
⋮	⋮

SQL Functions

FUNCTION

- Most SQL implementations support some variant of ANSI SQL FUNCTION
- Details vary ...

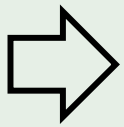
TransactSQL function to return cnames reformatted

```
CREATE FUNCTION cname_to_initial_first(@cname VARCHAR(20))
       RETURNS VARCHAR(20) AS
BEGIN
   DECLARE @ifcname VARCHAR(20)

   SELECT @ifcname=
            SUBSTRING(@cname,CHARINDEX(' ',@cname)+2,LEN(@cname))+
            SUBSTRING(@cname,1,CHARINDEX(' ',@cname)-1)

   RETURN @ifcname
END


SELECT no,
       dbo.cname_to_initial_first(
           account.cname) AS cname
FROM   account
```



no	cname
100	P.McBrien
101	P.McBrien
103	M.Boyd
107	A.Poulovassilis
119	A.Poulovassilis
125	J.Bailey

SQL Procedures

PROCEDURE

- No specific PROCEDURE construct in Postgres
- TransactSQL supports PROCEDURE definition, and generally refers to them a **stored procedure**

TransactSQL Procedure to move cash between branches

```
CREATE PROCEDURE move_cash
(   @from_branch INTEGER,
    @to_branch INTEGER,
    @total DECIMAL(10,2)
) AS
BEGIN
    UPDATE branch
    SET cash=cash-@total
    WHERE sortcode=@from_branch

    UPDATE branch
    SET cash=cash+@total
    WHERE sortcode=@to_branch
END
```

Maintaining Logical Rules in an RDBMS

Logical Rule

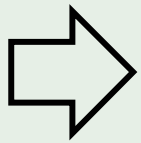
A statement of a logical fact that must always be (possibly) true

Logical rule to define possbile members of current account

IF account(No, CN, 'current', _, SC) THEN current_account(No, CN, SC)

Use logical rule to create a new table

```
SELECT  no,
        cname,
        sortcode
INTO    current_account
FROM    account
WHERE   type='current'
```



current_account		
no	cname	sortcode
100	McBrien, P.	67
103	Boyd, M.	34
107	Poulovassilis, A.	56
125	Bailey, J.	56

Maintaining Logical Rules in an RDBMS

Logical Rule

A statement of a logical fact that must always be (possibly) true

Logical rule to define possbile members of current account

IF account(No, CN, 'current', _, SC) THEN current_account(No, CN, SC)

Operation Rules

A statement of how to make a logical fact true over time in a DBMS

- Implement as a constraint
  If we insert/update an account to be a current account, then check that account number also appears in the current_account table, and prevent the insert/update if not.

Maintaining Logical Rules in an RDBMS

Logical Rule

A statement of a logical fact that must always be (possibly) true

Logical rule to define possbile members of current account

IF account(No, CN, 'current', _, SC) THEN current_account(No, CN, SC)

Operation Rules

A statement of how to make a logical fact true over time in a DBMS

- Implement as a constraint
  If we insert/update an account to be a current account, then check that account number also appears in the current_account table, and prevent the insert/update if not.
- Implement as a view
  The extent of current_account is derived from the extent of account.

Maintaining Logical Rules in an RDBMS

Logical Rule

A statement of a logical fact that must always be (possibly) true

Logical rule to define possbile members of current account

IF account(No, CN, 'current', _, SC) THEN current_account(No, CN, SC)

Operation Rules

A statement of how to make a logical fact true over time in a DBMS

- Implement as a constraint
  If we insert/update an account to be a current account, then check that account number also appears in the current_account table, and prevent the insert/update if not.
- Implement as a view
  The extent of current_account is derived from the extent of account.
- Implement as an active database
  Inserts/updates to the account table that match the rule condition cause insert/updates to the current_account table.

SQL Constraints

$\forall No, Rate.account(No, \_, \_, Rate, \_) \rightarrow Rate \geq 0.00$

```
ALTER TABLE account
ADD CONSTRAINT check_account_rate
CHECK (rate >=0.00)
```

IF account(No, CN,'current', _, SC) THEN current_account(No, CN, SC)

```
CREATE FUNCTION is_in_current_account(@NO INT,@CN VARCHAR(20),@SC INT)
RETURNS BIT AS
BEGIN
   IF EXISTS (SELECT *
              FROM   current_account
              WHERE  no=@NO
              AND    cname=@CN
              AND    sortcode=@SC)
      RETURN 1

   RETURN 0
END;

ALTER TABLE account
ADD CONSTRAINT check_current_account
CHECK (type<>'current' OR dbo.is_in_current_account(no,cname,sortcode)=1);
```

Operational Semantics: Views

VIEW

- Implements Datalog rules
- Basic ANSI SQL CREATE VIEW well supported across platforms
- Variations in details

Views defining current account and deposit account

```
CREATE VIEW current_account AS
SELECT no,
       cname,
       sortcode
FROM   account
WHERE  type='current'

CREATE VIEW deposit_account AS
SELECT no,
       cname,
       rate,
       sortcode
FROM   account
WHERE  type='deposit'
```

Quiz 3.1: Updates to SQL Views

account				
no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

```
CREATE VIEW current_account AS
SELECT no,
       cname,
       sortcode
FROM   account
WHERE  type='current'
```

Which SQL view update does not work?

A

```
UPDATE current_account
SET    sortcode=56
WHERE  no=125
```

B

```
UPDATE current_account
SET    sortcode=56
```

C

```
DELETE FROM current_account
WHERE  no=125
```

D

```
INSERT INTO current_account
VALUES (129,'Jones, F.',34)
```

SQL View Updates

SQL restricts View updates to view definitions

- on just one table
- containing no aggregates
- no computed columns
- for INSERT: all non-NULLable columns without defaults being included in view

Ambiguous view updates

```
CREATE VIEW active_account AS
SELECT no,
       cname,
       sortcode
FROM   account JOIN movement ON account.no=movement.no

DELETE FROM active_account
WHERE  no=100
```

The DELETE could be fulfilled by either (a) deleting account 100 or (b) deleting all movements for account 100

SQL Materialised Views

Materialised Views

- cache the result of the view query
- add MATERIALIZED to view creation
- use some sort of REFRESH to repopulate view ...
  not standardised or supported accross all platforms

Materialised Views

```
CREATE MATERIALIZED VIEW current_account AS
SELECT no,
       cname,
       sortcode
FROM   account
WHERE  type='current'

CREATE MATERIALIZED VIEW deposit_account AS
SELECT no,
       cname,
       rate,
       sortcode
FROM   account
WHERE  type='deposit'
```

Operational Semantics: Constraints

General purpose constraints

- PRIMARY KEY and FOREIGN KEY are examples of constraint rules
- CHECK allows any SQL code to be run after a table is updated

IF account(No, _, _, Rate, _) THEN Rate $\geq$ 0.00

```
ALTER TABLE account
ADD CONSTRAINT check_account_rate
CHECK (rate >=0.00)
```

Cascading UPDATE and DELETE operations

- Can cascade updates on one column to other columns that reference that column

```
CONSTRAINT account_fk FOREIGN KEY (sortcode)
REFERENCES branch ON UPDATE CASCADE
```

- Can cascade deletes on one column to other columns that reference that column

```
CONSTRAINT movement_fk FOREIGN KEY (no)
REFERENCES account ON DELETE CASCADE
```

Operational Semantics: Triggers

Trigger

A **trigger** causes a change on some table to cause a block of SQL to be executed. Implementations vary, but the SQL Standard says:

- Cause execution of procedure when an INSERT, DELETE, or UPDATE occurs
- Can be FOR EACH ROW or FOR EACH STATEMENT
- Can execute BEFORE or AFTER the change is made

IF account(No, CN, 'current', _, SC) THEN current_account(No, CN, SC) Implemented Using SQL Server Triggers

```
CREATE TRIGGER insert_current_account
ON account AFTER INSERT AS
INSERT INTO current_account
SELECT INSERTED.no,
       INSERTED.cname,
       INSERTED.sortcode
FROM   INSERTED
WHERE  INSERTED.type='current'
```

Triggers: Implementing Constraints

```
CREATE FUNCTION delete_only_cleared_accounts() RETURNS TRIGGER
AS
DECLARE account_balance DECIMAL(10,2);
BEGIN
      SELECT INTO account_balance SUM(amount)
      FROM   movement
      WHERE  movement.no=OLD.no;

      IF (account_balance<>0 AND account_balance IS NOT NULL) THEN
        RAISE EXCEPTION 'Unable to delete account % since it has balance of %',
                        OLD.no,
                        account_balance;
      END IF;

      RETURN OLD;
END
LANGUAGE plpgsql;

CREATE TRIGGER check_no_balance
BEFORE DELETE ON account FOR EACH ROW
EXECUTE PROCEDURE delete_only_cleared_accounts();
```

Triggers: Implementing Active Databases

```
CREATE FUNCTION delete_redundant_account() RETURNS TRIGGER
AS BEGIN
      DELETE FROM account
      WHERE  account.cname=OLD.cname
      AND NOT EXISTS (SELECT cname
                      FROM   account
                      WHERE  account.cname=OLD.cname
                      AND    type='current');

      RETURN NULL;
   END
LANGUAGE plpgsql;

CREATE TRIGGER check_no_other_accounts
AFTER DELETE ON account FOR EACH ROW
EXECUTE PROCEDURE delete_redundant_account();
```

Case Study: Transitive Closure

Transitive Closure in a Directed Graph

If edge(a,b) and edge(b,c) then edge(a,c)

edge	
node1	node2
a	b
c	d
c	f
e	b

Case Study: Transitive Closure

Transitive Closure in a Directed Graph

If edge(a,b) and edge(b,c) then edge(a,c)

edge	
node1	node2
a	b
c	d
c	f
e	b

```
INSERT INTO edge
VALUES('b','c');
```

edge	
node1	node2
a	b
a	c
a	d
a	f
b	c
b	d
b	f
c	d
c	f
e	b
e	c
e	d
e	f

Case Study: Transitive Closure

Transitive Closure in a Directed Graph

If edge(a,b) and edge(b,c) then edge(a,c)

edge	
node1	node2
a	b
c	d
c	f
e	b

```
INSERT INTO edge
VALUES('b','c');
```

edge	
node1	node2
a	b
a	c
a	d
a	f
b	c
b	d
b	f
c	d
c	f
e	b
e	c
e	d
e	f

```
INSERT INTO edge
VALUES('f','a');
```

edge	
node1	node2
a	b
a	c
a	d
a	f
b	a
b	c
b	d
b	f
c	a
c	b
c	d
c	f
e	a
e	b
e	c
e	d
e	f
f	a
f	b
f	c
f	d

Case Study: Implementing Transitive Closure using Triggers

```
CREATE FUNCTION edge_implies_transitive_edge() RETURNS TRIGGER
AS BEGIN
      INSERT INTO edge
      SELECT edge.node1,NEW.node2
      FROM   edge
      WHERE  edge.node2=NEW.node1
      AND    NOT EXISTS (SELECT *
                         FROM   edge AS old_edge
                         WHERE  edge.node1=old_edge.node1
                         AND    NEW.node2=old_edge.node2)
      AND    NOT edge.node1=NEW.node2;

      INSERT INTO edge
      SELECT NEW.node1,edge.node2
      FROM   edge
      WHERE  edge.node1=NEW.node2
      AND    NOT EXISTS (SELECT *
                         FROM   edge AS old_edge
                         WHERE  NEW.node1=old_edge.node1
                         AND    edge.node2=old_edge.node2)
      AND    NOT NEW.node1=edge.node2;


      RETURN NULL;
    END
LANGUAGE plpgsql;

CREATE TRIGGER edge_implies_edge
AFTER INSERT ON edge FOR EACH ROW
EXECUTE PROCEDURE edge_implies_transitive_edge();
```

Worksheet: Active Databases (1)

```
CREATE TABLE site
(       sitecode CHAR(4) NOT NULL,
        area INTEGER NOT NULL,
        postcode VARCHAR(20) NOT NULL,
        CONSTRAINT site_pk PRIMARY KEY (sitecode)
)
CREATE TABLE branch
(       sortcode CHAR(9) NOT NULL,
        name VARCHAR(20) NOT NULL,
        cash DECIMAL(10,2) NOT NULL,
        CONSTRAINT branch_pk PRIMARY KEY (sortcode)
)
```

Worksheet: Bank Branch Database

branch		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

account				
no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

key branch(sortcode)

key branch(bname)

key movement(mid)

key account(no)

movement(no) $\stackrel{fk}{\Rightarrow}$ account(no)

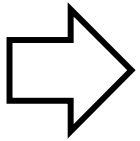
account(sortcode) $\stackrel{fk}{\Rightarrow}$ branch(sortcode)

Worksheet: Active Databases (1)

```
CREATE TABLE site
(       sitecode CHAR(4) NOT NULL,
        area INTEGER NOT NULL,
        postcode VARCHAR(20) NOT NULL,
        CONSTRAINT site_pk PRIMARY KEY (sitecode)
)
CREATE TABLE branch
(       sortcode CHAR(9) NOT NULL,
        name VARCHAR(20) NOT NULL,
        cash DECIMAL(10,2) NOT NULL,
        sitecode CHAR(4) NOT NULL,
        CONSTRAINT branch_pk PRIMARY KEY (sortcode),
        CONSTRAINT branch_fk FOREIGN KEY (sitecode)
            REFERENCES site ON DELETE CASCADE
)
```

Worksheet: Active Databases (2)

```
SELECT no,
       SUM(amount) AS balance
FROM   movement
GROUP BY no
```



no	balance
100	2086.78
101	5230.00
103	145.50
107	245.56
119	5600.00

Quiz 3.2: Execution of Triggers

```
DELETE FROM account WHERE no=107
```

What occurs in the database when the above is executed?

A

Account 107 is Deleted

B

Account 107 is Deleted
Message 'Unable to delete account 107' printed

C

Account 107 is Deleted
Message 'Unable to delete account 107' printed
Transaction Rolled Back

D

Message 'Unable to delete account 107' printed
Transaction Rolled Back

Worksheet: Active Databases (3)

```
CREATE FUNCTION delete_redundant_account() RETURNS TRIGGER
AS 'BEGIN
      DELETE FROM account
      WHERE account.cname=OLD.cname
      AND NOT EXISTS (SELECT cname
          FROM account
          WHERE account.cname=OLD.cname
          AND NOT account.no=OLD.no
          AND type=\'current\');

      RETURN OLD;
END'
LANGUAGE plpgsql;

CREATE TRIGGER check_no_other_accounts
BEFORE DELETE ON account FOR EACH ROW
EXECUTE PROCEDURE delete_redundant_account();
```

Summary of Operational Semantics in RDBMS

At a logical level, we want to maintain rules of the form
IF x THEN y

Operation Semantics: Integrity Constraints

Keys, foreign keys and CHECK constrains all work to ensure the if a change to the database is attempted to break the logical rule, then a change is rejected.

Operation Semantics: Views

A VIEW means that the values of y can be derived by running a query that tests for x

Operation Semantics: Triggers

A TRIGGER can be written to ensure that if a change occurs that makes x true, then another change occurs that makes y true.

List Comprehensions

- RA is the foundation of relational databases
  - has well founded semantics
  - logic version **relational calculus**
- SQL is the relational database language used in practice
  - bag or set semantics
  - aggregates
  - can have arrays as attributes
- Other languages such as XPath
  - list semantics (*i.e.* order sensitive)
  - nested structures

List Comprehensions provide a combination of well founded semantics with list semantics and nested structures

Simple to represent RA (and SQL) in List Comprehensions

List Comprehension: Representation of Data

Most direct representation of relational data is as list of tuples.

branch

sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

$$\text{table:}\langle\langle \text{branch}/3 \rangle\rangle = [\langle 56, \text{'Wimbledon'}, 94340.45 \rangle, \langle 34, \text{'Goodge St'}, 8900.67 \rangle, \langle 67, \text{'Strand'}, 34005.00 \rangle]$$

movement

mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

$$\text{table:}\langle\langle \text{movement}/4 \rangle\rangle = [\langle 1000, 100, 2300.00, 5/1/1999 \rangle, \langle 1001, 101, 4000.00, 5/1/1999 \rangle, \langle 1002, 100, -223.45, 8/1/1999 \rangle, \langle 1004, 107, -100.00, 11/1/1999 \rangle, \langle 1005, 103, 145.50, 12/1/1999 \rangle, \langle 1006, 100, 10.23, 15/1/1999 \rangle, \langle 1007, 107, 345.56, 15/1/1999 \rangle, \langle 1008, 101, 1230.00, 15/1/1999 \rangle, \langle 1009, 119, 5600.00, 18/1/1999 \rangle]$$

List Comprehension: Generators and Filters

account				
no	type	cname	rate	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

table:⟪account/5⟫ =
[⟨100, 'current', 'McBrien, P.', NULL, 67 ⟩,
⟨101, 'deposit', 'McBrien, P.', 5.25, 67 ⟩,
⟨103, 'current', 'Boyd, M.', NULL, 34⟩,
⟨107, 'current', 'Poulovassilis, A.', NULL, 56⟩,
⟨119, 'deposit', 'Poulovassilis, A.', 5.50, 56 ⟩,
⟨125, 'current', 'Bailey, J.', NULL, 56 ⟩]

- **generators** read values from schemes

$$\text{table:}\langle\langle \text{account}/5 \rangle\rangle = [\langle x, y, z, w, v \rangle \mid \langle x, y, z, w, v \rangle \leftarrow \langle\langle \text{account}/5 \rangle\rangle]$$

- **predicates** filter the values in a list

$$\text{table:}\langle\langle \text{account}/5 \rangle\rangle \supseteq [\langle x, y, z, w, v \rangle \mid \langle x, y, z, w, v \rangle \leftarrow \langle\langle \text{account}/5 \rangle\rangle; w > 0]$$

List Comprehension: Project

account				
no	type	cname	rate	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

table:$\langle\langle$account/5$\rangle\rangle$ =
[$\langle$100, 'current', 'McBrien, P.', NULL, 67 $\rangle$,
$\langle$101, 'deposit', 'McBrien, P.', 5.25, 67 $\rangle$,
$\langle$103, 'current', 'Boyd, M.', NULL, 34$\rangle$,
$\langle$107, 'current', 'Poulovassilis, A.', NULL, 56$\rangle$,
$\langle$119, 'deposit', 'Poulovassilis, A.', 5.50, 56 $\rangle$,
$\langle$125, 'current', 'Bailey, J.', NULL, 56 $\rangle$]

- π_{sortcode}account

$$\text{distinct}[\langle v\rangle \mid \langle x, y, z, w, v\rangle \leftarrow \langle\langle \text{account}/5\rangle\rangle]$$

  - Use **distinct** when set semantics required → RA, or SQL with DISTINCT
  - No **distinct** when bag semantics required → SQL

List Comprehension: Select

account				
no	type	cname	rate	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

table:$\langle\langle$account/5$\rangle\rangle$ =
[$\langle$100, 'current', 'McBrien, P.', NULL, 67 $\rangle$,
$\langle$101, 'deposit', 'McBrien, P.', 5.25, 67 $\rangle$,
$\langle$103, 'current', 'Boyd, M.', NULL, 34$\rangle$,
$\langle$107, 'current', 'Poulovassilis, A.', NULL, 56$\rangle$,
$\langle$119, 'deposit', 'Poulovassilis, A.', 5.50, 56 $\rangle$,
$\langle$125, 'current', 'Bailey, J.', NULL, 56 $\rangle$]

- $\sigma_{\text{rate}=5.50}$account

$$[\langle x, y, z, w, v\rangle \mid \langle x, y, z, w, v\rangle \leftarrow \langle\langle \text{account}/5\rangle\rangle; w = 5.50]$$

which in IQL may also be written as

$$[\langle x, y, z, 5.50, v\rangle \mid \langle x, y, z, 5.50, v\rangle \leftarrow \langle\langle \text{account}/5\rangle\rangle]$$

List Comprehensions: Product

account				
no	type	cname	rate	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

table:$\langle\langle$account/5$\rangle\rangle$ =
[$\langle$100, 'current', 'McBrien, P.', NULL, 67 $\rangle$,
$\langle$101, 'deposit', 'McBrien, P.', 5.25, 67 $\rangle$,
$\langle$103, 'current', 'Boyd, M.', NULL, 34$\rangle$,
$\langle$107, 'current', 'Poulovassilis, A.', NULL, 56$\rangle$,
$\langle$119, 'deposit', 'Poulovassilis, A.', 5.50, 56 $\rangle$,
$\langle$125, 'current', 'Bailey, J.', NULL, 56 $\rangle$]

- branch $\bowtie$ account

$$[\langle x, y, z, r, s, t, v\rangle \mid \langle x, y, z\rangle \leftarrow \langle\langle branch/3\rangle\rangle; \langle r, s, t, v, w\rangle \leftarrow \langle\langle account/5\rangle\rangle; x = w]$$

which in IQL may also be written as

$$[\langle x, y, z, r, s, t, v\rangle \mid \langle x, y, z\rangle \leftarrow \langle\langle branch/3\rangle\rangle; \langle r, s, t, v, x\rangle \leftarrow \langle\langle account/5\rangle\rangle]$$

List Comprehensions: Union

account				
no	type	cname	rate	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

table:$\langle\langle$account/5$\rangle\rangle$ =
[$\langle$100, 'current', 'McBrien, P.', NULL, 67 $\rangle$,
$\langle$101, 'deposit', 'McBrien, P.', 5.25, 67 $\rangle$,
$\langle$103, 'current', 'Boyd, M.', NULL, 34$\rangle$,
$\langle$107, 'current', 'Poulovassilis, A.', NULL, 56$\rangle$,
$\langle$119, 'deposit', 'Poulovassilis, A.', 5.50, 56 $\rangle$,
$\langle$125, 'current', 'Bailey, J.', NULL, 56 $\rangle$]

- $\pi_{\text{sortcode as id}}\text{account} \cup \pi_{\text{no as id}}\text{account}$

$$\text{distinct}[\langle x\rangle \mid \langle x,y,z\rangle \leftarrow \langle\langle \text{branch}/3\rangle\rangle] +\!+ [\langle r\rangle \mid \langle r,s,t,v,w\rangle \leftarrow \langle\langle \text{account}/5\rangle\rangle]$$

List Comprehensions: Difference

account				
no	type	cname	rate	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

table:$\langle\langle$account/5$\rangle\rangle$ =
[$\langle$100, 'current', 'McBrien, P.', NULL, 67 $\rangle$,
$\langle$101, 'deposit', 'McBrien, P.', 5.25, 67 $\rangle$,
$\langle$103, 'current', 'Boyd, M.', NULL, 34$\rangle$,
$\langle$107, 'current', 'Poulovassilis, A.', NULL, 56$\rangle$,
$\langle$119, 'deposit', 'Poulovassilis, A.', 5.50, 56 $\rangle$,
$\langle$125, 'current', 'Bailey, J.', NULL, 56 $\rangle$]

- π_{no}account $-$ π_{no}movement

$$[\langle r\rangle \mid \langle r,s,t,v,w\rangle \leftarrow \langle\langle account/5\rangle\rangle] -\!\!- [\langle s\rangle \mid \langle r,s,t,v\rangle \leftarrow \langle\langle movement/4\rangle\rangle]$$

Worksheet: Translating Between SQL and IQL

```
SELECT cname, no
FROM   account
```

```
SELECT DISTINCT type, no
FROM   account JOIN movement ON account.no=movement.no
```

```
SELECT DISTINCT sortcode, bname
FROM   branch JOIN account ON branch.sortcode=account.sortcode
WHERE  account.rate>0
```

```
SELECT no
FROM   account
EXCEPT
SELECT no
FROM   movement
WHERE  amount<0
```

Worksheet: Translating Between SQL and IQL (1)

```
SELECT cname, no
FROM   account
```

$[\langle z, x\rangle \mid \langle x, y, z, w, v\rangle \leftarrow \langle\langle \text{account}/5\rangle\rangle]$

Worksheet: Translating Between SQL and IQL (2)

```
SELECT DISTINCT type, no
FROM   account JOIN movement ON account.no=movement.no
```

$$\mathsf{distinct}[\langle s, r\rangle \mid \langle r, s, t, v, u\rangle \leftarrow \langle\!\langle \mathsf{account}/5 \rangle\!\rangle; \langle x, r, y, z\rangle \leftarrow \langle\!\langle \mathsf{movement}/4 \rangle\!\rangle]$$

Worksheet: Translating Between SQL and IQL (3)

```
SELECT DISTINCT sortcode, bname
FROM   branch JOIN account ON branch.sortcode=account.sortcode
WHERE  account.rate>0
```

$$\text{distinct}[\langle x, y\rangle \mid \langle x, y, z\rangle \leftarrow \langle\langle \text{branch}/3\rangle\rangle; \langle r, s, t, v, x\rangle \leftarrow \langle\langle \text{account}/5\rangle\rangle; v > 0]$$

Worksheet: Translating Between SQL and IQL (4)

```
SELECT no
FROM   account
EXCEPT
SELECT no
FROM   movement
WHERE  amount<0
```

$$[\langle r \rangle \mid \langle r, s, t, v, u \rangle \leftarrow \langle\langle account/5 \rangle\rangle] --$$
$$[\langle y \rangle \mid \langle x, y, z, w \rangle \leftarrow \langle\langle movement/4 \rangle\rangle; z < 0]$$

Relational $\leftrightarrow$ ER

ER to relational mappings

- Design ER model for new DBMS system
- Map ER model to relations for DBMS implementation

Relational $\leftrightarrow$ ER

ER to relational mappings

- Design ER model for new DBMS system
- Map ER model to relations for DBMS implementation

Relational to ER mappings

- Have existing DBMS that one wishes to view the design of
- Map relations to ER model for design review, modification and integration

Mapping Relational to $\mathcal{ER}^{\mathcal{KLMOS}}$: (Assuming TPT)

- $\mathcal{R}$ If table R has just primary keys P_R that contains two columns which are foreign keys for $R_1, R_2 \rightarrow$ many-many ER relationship R between entities for R_1, R_2
- $\mathcal{E}$ Otherwise $R \rightarrow$ ER entity
  For each column A:

Mapping Relational to $\mathcal{ER}^{\mathcal{KLMOS}}$: (Assuming TPT)

- $\mathcal{R}$ If table R has just primary keys P_R that contains two columns which are foreign keys for $R_1, R_2 \rightarrow$ many-many ER relationship R between entities for R_1, R_2
- $\mathcal{E}$ Otherwise $R \rightarrow$ ER entity
  For each column A:
  1. If A is (part of) a candidate key:
     - $\mathcal{S}$ if candidate key is also a foreign key $\rightarrow$ subset
     - $\mathcal{K}$ otherwise $A \rightarrow$ ER key attribute

Mapping Relational to $\mathcal{ER}^{\mathcal{KLMOS}}$: (Assuming TPT)

- $\mathcal{R}$ If table R has just primary keys P_R that contains two columns which are foreign keys for $R_1, R_2 \rightarrow$ many-many ER relationship R between entities for R_1, R_2
- $\mathcal{E}$ Otherwise $R \rightarrow$ ER entity
  For each column A:
  1. If A is (part of) a candidate key:
     - $\mathcal{S}$ if candidate key is also a foreign key $\rightarrow$ subset
     - $\mathcal{K}$ otherwise $A \rightarrow$ ER key attribute
  2. If A is not (part of) a candidate key:
     - $\mathcal{R}$ if A is (part of) a foreign key $\rightarrow$ ER relationship
     - $\mathcal{O}$ otherwise if A is nullable $\rightarrow$ ER optional attribute
     - $\mathcal{M}$ otherwise $A \rightarrow$ ER mandatory attribute

Mapping Relational to $\mathcal{ER}^{\mathcal{KLMOS}}$ using TPT: Example

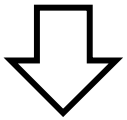
site(sitecode,sortcode?,name)
withdraw(sitecode,cname)
customer(cname,joined,salary?,address,phone)
web_customer(cname,username,password,email)
account(number,acname,cname,sitecode)

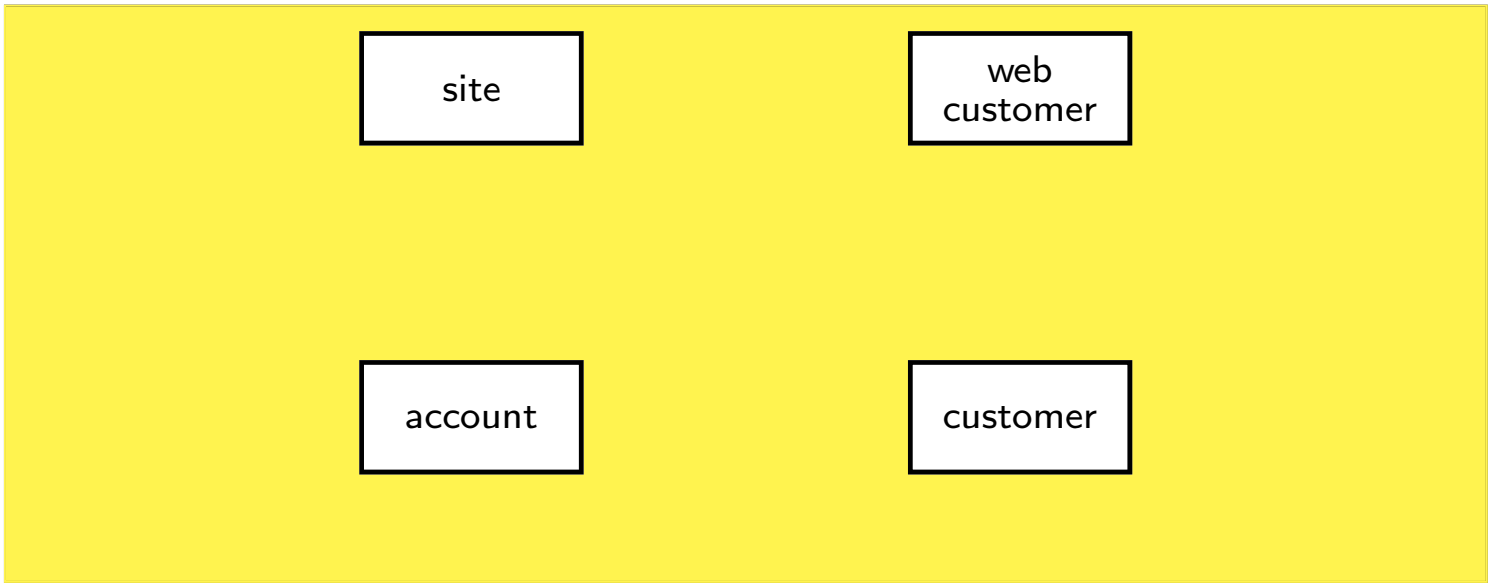
withdraw(sitecode) $\stackrel{fk}{\Rightarrow}$ site(sitecode)
withdraw(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
web_customer(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
account(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
account(sitecode) $\stackrel{fk}{\Rightarrow}$ site(sitecode)

Mapping Relational to $\mathcal{ER}^{\mathcal{KLMOS}}$ using TPT: Example

site(sitecode,sortcode?,name)
withdraw(sitecode,cname)
customer(cname,joined,salary?,address,phone)
web_customer(cname,username,password,email)
account(number,acname,cname,sitecode)

withdraw(sitecode) $\stackrel{fk}{\Rightarrow}$ site(sitecode)
withdraw(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
web_customer(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
account(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
account(sitecode) $\stackrel{fk}{\Rightarrow}$ site(sitecode)

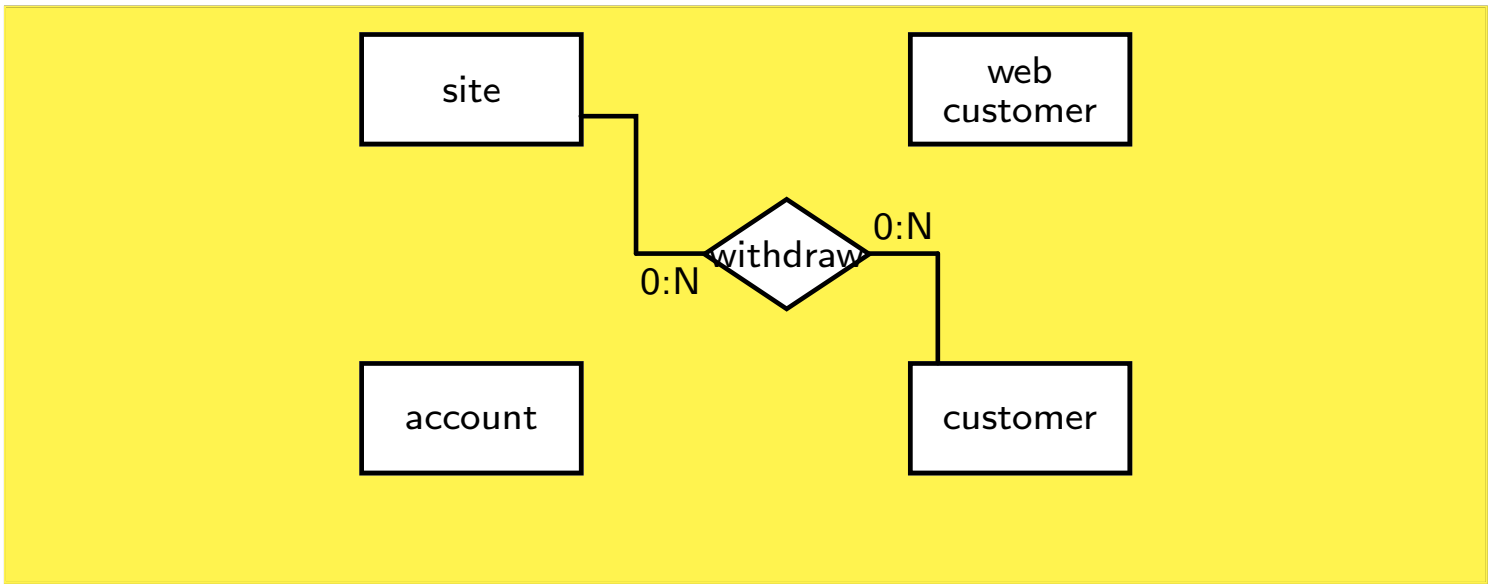




Mapping Relational to $\mathcal{ER}^{\mathcal{KLMOS}}$ using TPT: Example

site(sitecode,sortcode?,name)
withdraw(sitecode,cname)
customer(cname,joined,salary?,address,phone)
web_customer(cname,username,password,email)
account(number,acname,cname,sitecode)

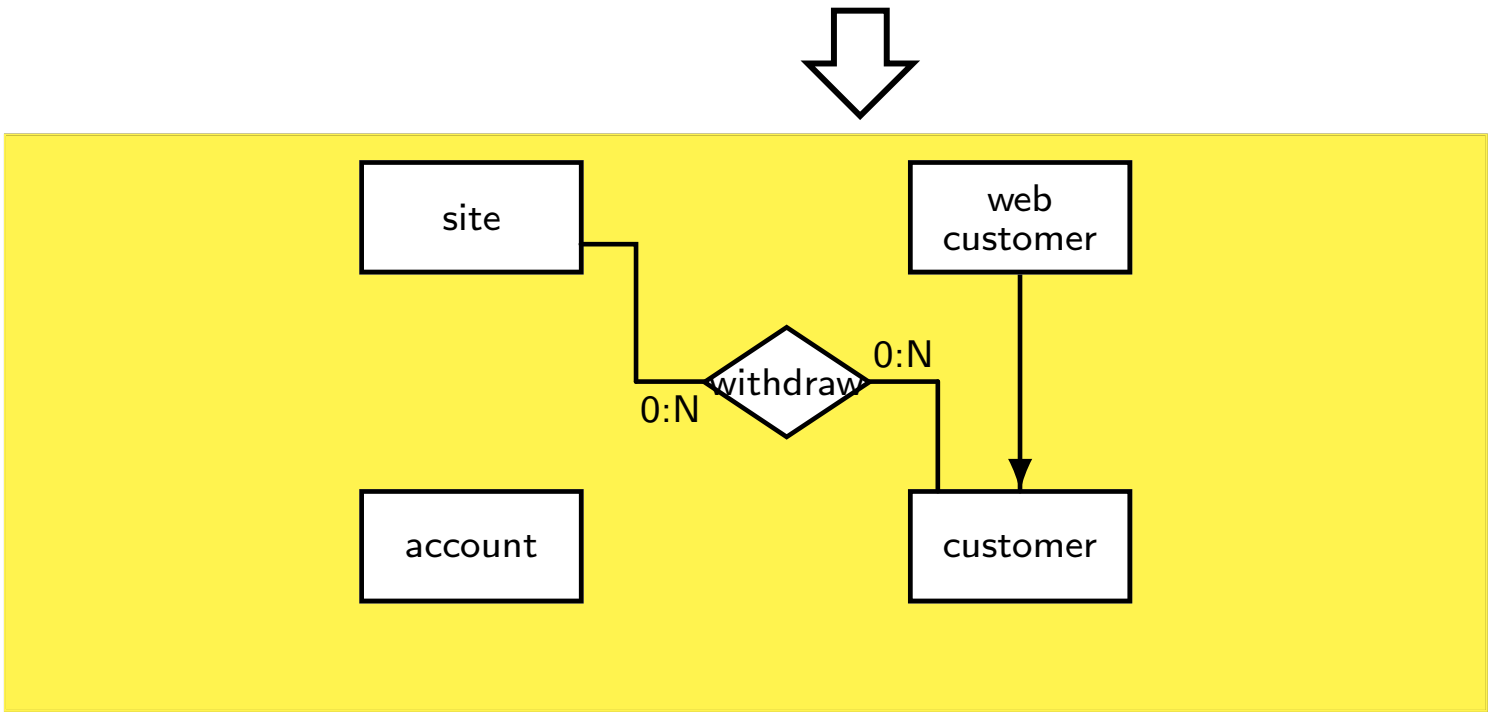
withdraw(sitecode) $\stackrel{fk}{\Rightarrow}$ site(sitecode)
withdraw(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
web_customer(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
account(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
account(sitecode) $\stackrel{fk}{\Rightarrow}$ site(sitecode)



Mapping Relational to $\mathcal{ER}^{KLMOS}$ using TPT: Example

site(sitecode,sortcode?,name)
withdraw(sitecode,cname)
customer(cname,joined,salary?,address,phone)
web_customer(cname,username,password,email)
account(number,acname,cname,sitecode)

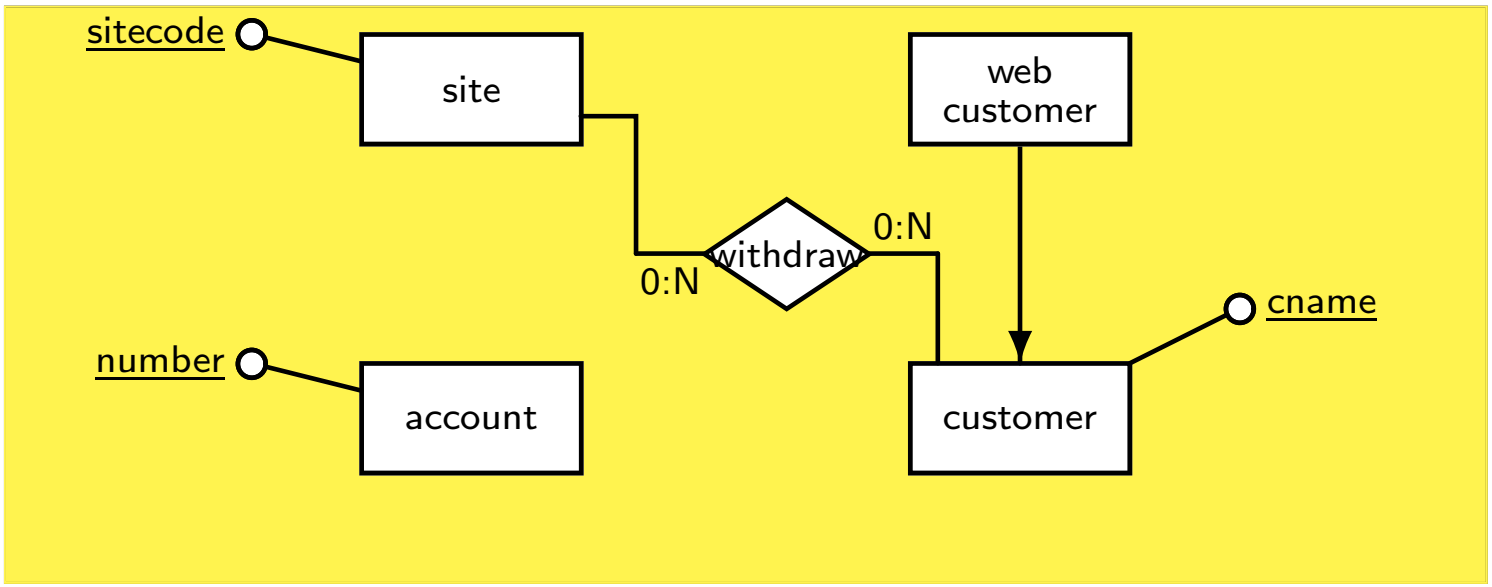
withdraw(sitecode) $\stackrel{fk}{\Rightarrow}$ site(sitecode)
withdraw(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
web_customer(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
account(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
account(sitecode) $\stackrel{fk}{\Rightarrow}$ site(sitecode)



Mapping Relational to $\mathcal{ER}^{\mathcal{KLMOS}}$ using TPT: Example

site(sitecode,sortcode?,name)
withdraw(sitecode,cname)
customer(cname,joined,salary?,address,phone)
web_customer(cname,username,password,email)
account(number,acname,cname,sitecode)

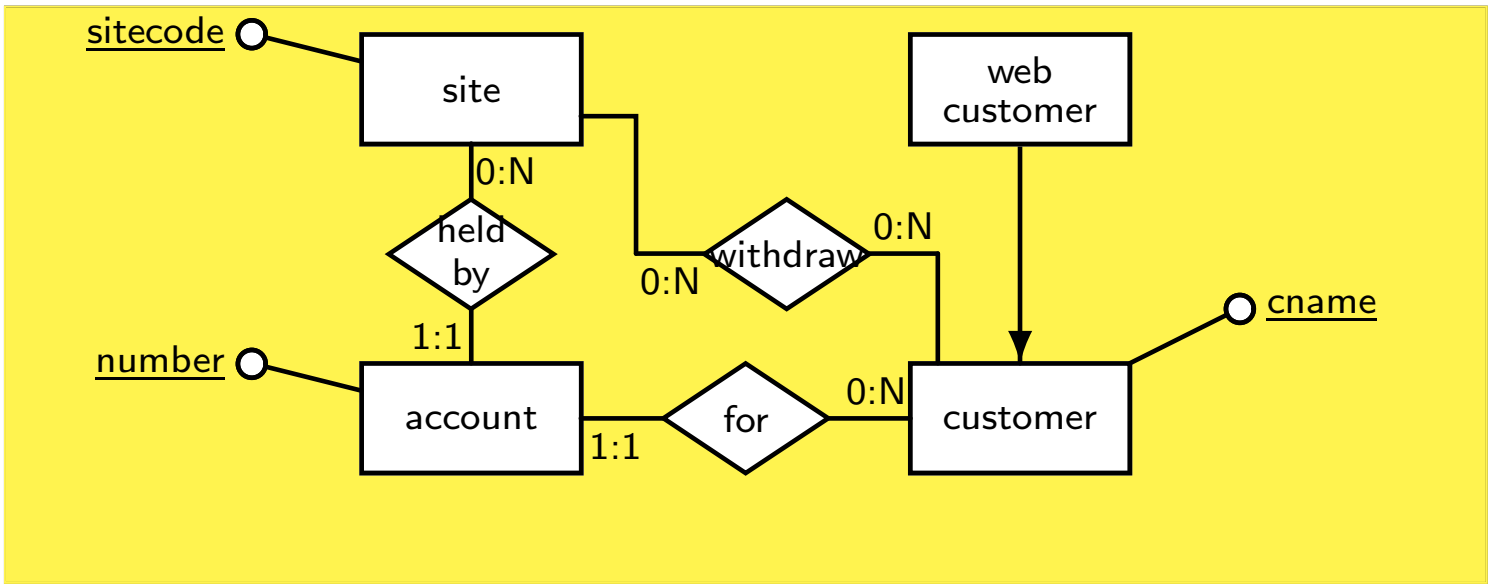
withdraw(sitecode) $\stackrel{fk}{\Rightarrow}$ site(sitecode)
withdraw(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
web_customer(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
account(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
account(sitecode) $\stackrel{fk}{\Rightarrow}$ site(sitecode)



Mapping Relational to $\mathcal{ER}^{KLMOS}$ using TPT: Example

site(<u>sitecode</u>,sortcode?,name)
withdraw(<u>sitecode</u>,<u>cname</u>)
customer(<u>cname</u>,joined,salary?,address,phone)
web_customer(<u>cname</u>,username,password,email)
account(<u>number</u>,acname,cname,sitecode)

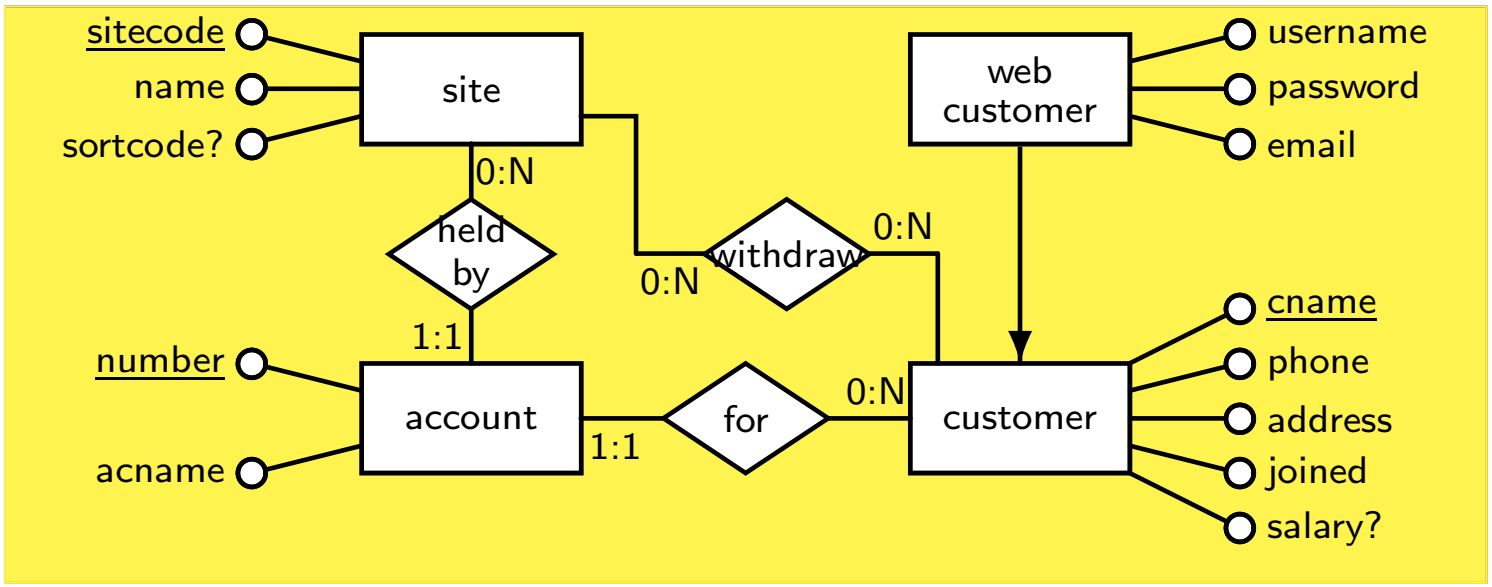
withdraw(sitecode) $\stackrel{fk}{\Rightarrow}$ site(sitecode)
withdraw(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
web_customer(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
account(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
account(sitecode) $\stackrel{fk}{\Rightarrow}$ site(sitecode)



Mapping Relational to $\mathcal{ER}^{KLMOS}$ using TPT: Example

site(<u>sitecode</u>,sortcode?,name)
withdraw(<u>sitecode,cname</u>)
customer(<u>cname</u>,joined,salary?,address,phone)
web_customer(<u>cname</u>,username,password,email)
account(<u>number</u>,acname,cname,sitecode)

withdraw(sitecode) $\stackrel{fk}{\Rightarrow}$ site(sitecode)
withdraw(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
web_customer(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
account(cname) $\stackrel{fk}{\Rightarrow}$ customer(cname)
account(sitecode) $\stackrel{fk}{\Rightarrow}$ site(sitecode)



Worksheet: Reverse Engineering ER Schemas

Build an $\mathcal{ER}^{\mathcal{KLMOS}}$ schema representing the following relational schema

person(<u>name</u>,dcode,salary,age?)
person(dcode) $\stackrel{fk}{\Rightarrow}$ department(dcode)
manager(<u>name</u>,dcode,car)
manager(name) $\stackrel{fk}{\Rightarrow}$ person(name)
department(<u>dcode</u>,site)
sales_department(<u>dcode</u>,telephone)
sales_department(dcode) $\stackrel{fk}{\Rightarrow}$ department(dcode)
production_department(<u>dcode</u>)
production_department(dcode) $\stackrel{fk}{\Rightarrow}$ department(dcode)
department_handles_product(<u>dcode,pcode</u>)
department_handles_product(dcode) $\stackrel{fk}{\Rightarrow}$ department(dcode)
department_handles_product(pcode) $\stackrel{fk}{\Rightarrow}$ product(pcode)
product(<u>pcode</u>,price,weight)

Mapping Relational to $\mathcal{ER}^{\mathcal{ADHKLMNOSVW}}$ (Assuming TPT)

$\mathcal{H}$ For each table R that has primary key P_R that contains n columns which are foreign keys for $R_1, \ldots, R_n$ $(n \geq 2) \rightarrow$ many-many ER relationship R between the ER version of $R_1, \ldots, R_n$. For each column A of R:

- If column part of primary key, then already represented by many-many relationship
- $\mathcal{A}$ Otherwise, create attribute A of relationship

$\mathcal{V}$ Otherwise, if R has all columns as a key, and all but one column a foreign key, map R to an optional multi-valued attribute.

$\mathcal{W}$ Otherwise, if R has a set of columns as a key, and a proper subset of those columns as a foreign key of S, map R to a weak entity.

$\mathcal{E}$ Otherwise each table $R \rightarrow$ ER entity. For each column A on R.

1. If A is (part of) a candidate key:
   - $\mathcal{S}$ if candidate key is also a foreign key $\rightarrow$ subset
   - $\mathcal{K}$ otherwise $A \rightarrow$ ER key attribute
2. If A is not (part of) a candidate key:
   - $\mathcal{R}$ if A is (part of) a foreign key to $R_f \rightarrow$ ER relationship to ER version of R_f
   - $\mathcal{O}$ otherwise if A is nullable $\rightarrow$ ER optional attribute
   - $\mathcal{M}$ otherwise $A \rightarrow$ ER mandatory attribute

$\mathcal{D}$ Use additional domain knowledge to convert several subclasses of a single entity into members of a single generalisation hierarchy

Alternatives for mapping ER to Relational Form

- Table per Type (TPT)
  - Generates a large number of tables
  - Does not implement disjointness in subclasses (unless triggers are used . . .)

Alternatives for mapping ER to Relational Form

- Table per Type (TPT)
  - Generates a large number of tables
  - Does not implement disjointness in subclasses (unless triggers are used . . .)
- Table per Concrete Class (TPC)
  - Generates fewer tables, superclass instances now spread between subclass tables.
  - Does not implement disjointness in subclasses (unless triggers are used), but implement mandatory attributes or relationships.

Alternatives for mapping ER to Relational Form

- Table per Type (TPT)
  - Generates a large number of tables
  - Does not implement disjointness in subclasses (unless triggers are used . . .)
- Table per Concrete Class (TPC)
  - Generates fewer tables, superclass instances now spread between subclass tables.
  - Does not implement disjointness in subclasses (unless triggers are used), but implement mandatory attributes or relationships.
- Table per Hierarchy (TPH)
  - Generates even fewer tables, but with many nullable attributes
  - Implement disjointness in subclasses, but does not implement mandatory attributes or relationships (unless CHECK constraints are used)

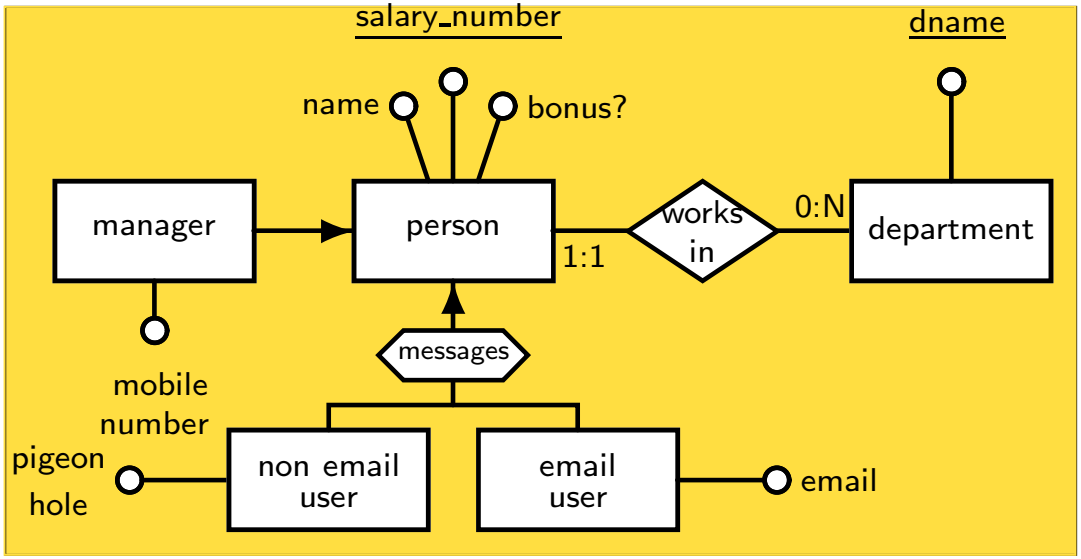
Mapping ER to Relational Form (Explicit Inheritance)

- Each entity maps to a distinct table, except superclass entities of total generalisations, which disappear!
- Each attribute maps to a column
- Each relationship
  - that is many-many maps to a table
  - that is one-many maps to a column in the table of the 'one' end and a foreign key pointing at the many table, provided the many table has no subclasses
- Each isa and generalisation causes the attributes of superclass entity to also appear as columns of the subclass table

Table per Concrete Type (TPC)

In ORM, this approach is called table per concrete type, since each non-virtual class type maps into a table, with the variables of super classes in the table.

Mapping ER to Relational Form (Explicit Inheritance)



manager(salary_number,name,bonus?,dname,mobile_number)

manager(dname) $\stackrel{fk}{\Rightarrow}$ department(dname)

department(dname)

non_email_user(salary_number,name,bonus?,dname,pigeon_hole)

non_email_user(dname) $\stackrel{fk}{\Rightarrow}$ department(dname)

email_user(salary_number,name,bonus?,dname,email)

email_user(dname) $\stackrel{fk}{\Rightarrow}$ department(dname)

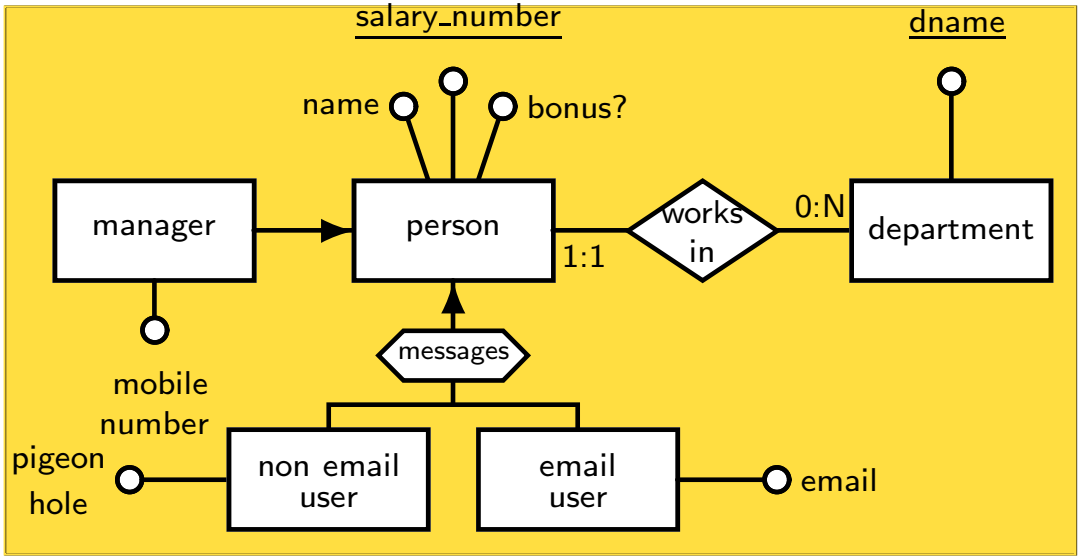
Mapping ER to Relational Form (ISA as Attribute)

- Each non-subclass entity maps to a distinct table
- Each attribute maps to a column in the superclass table (becoming nullable if moved up from a subclass)
- Each relationship
  - that is many-many maps to a table
  - that is one-many maps to a column in the table of the 'one' end and a foreign key pointing at the many table
- Each isa maps to a boolean flag
- Each generalisation maps to an attribute taking enumerated values

Table per Hierarchy (TPH)

In ORM, this approach is called table per hierarchy, since each class hierarchy maps into a single table

Mapping ER to Relational Form (ISA as Attribute)

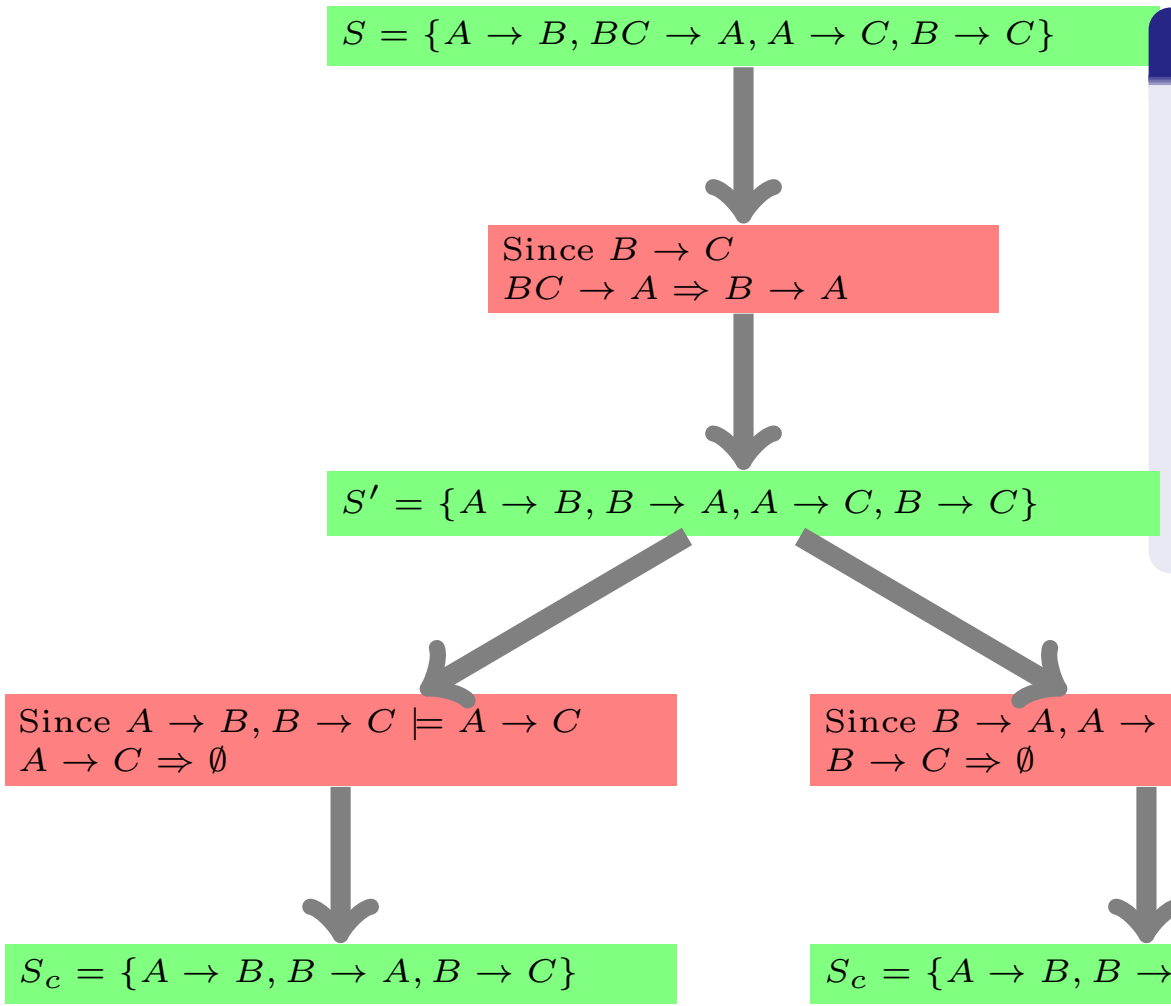


person(<u>salary_number</u>,name,bonus?,dname,
is_manager,mobile_number?,
messages,pigeon_hole?,email?)

person(dname) $\stackrel{fk}{\Rightarrow}$ department(dname)

department(<u>dname</u>)

Minimal cover of a set of FDs



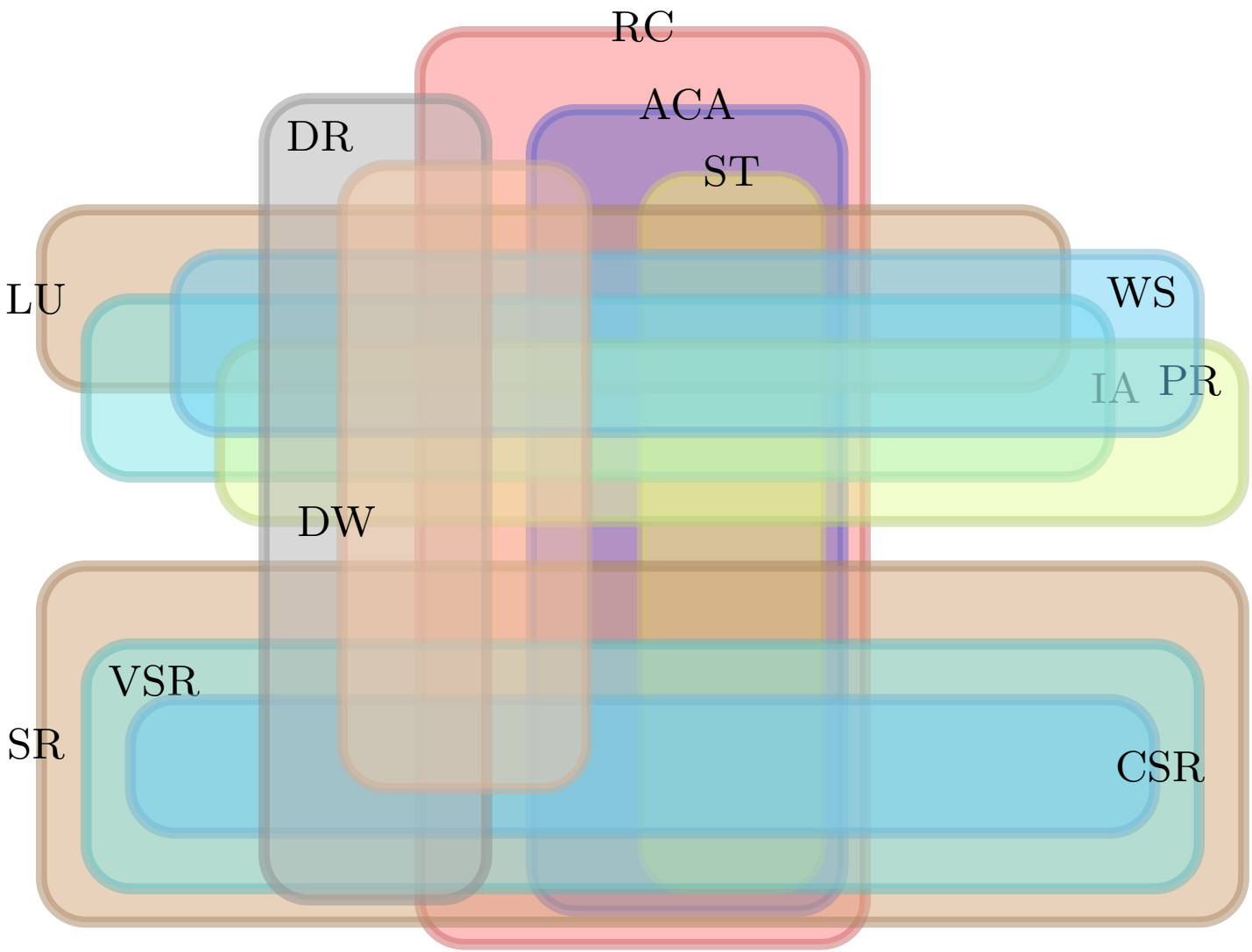
Minimal cover S_c of S

A minimal cover S_c of FD set S has the properties that:

- All the FDs in S can be derived from S_c (*i.e.* $S^+ = S_c^+$)
- It is not possible to form a new set S'_c by deleting an FD from S_c or deleting an attribute from an FD in S_c, and S'_c can derive all the FDs in S

In general, a set of FDs may have more than one minimal cover

Overview of Serialisable Histories Macro



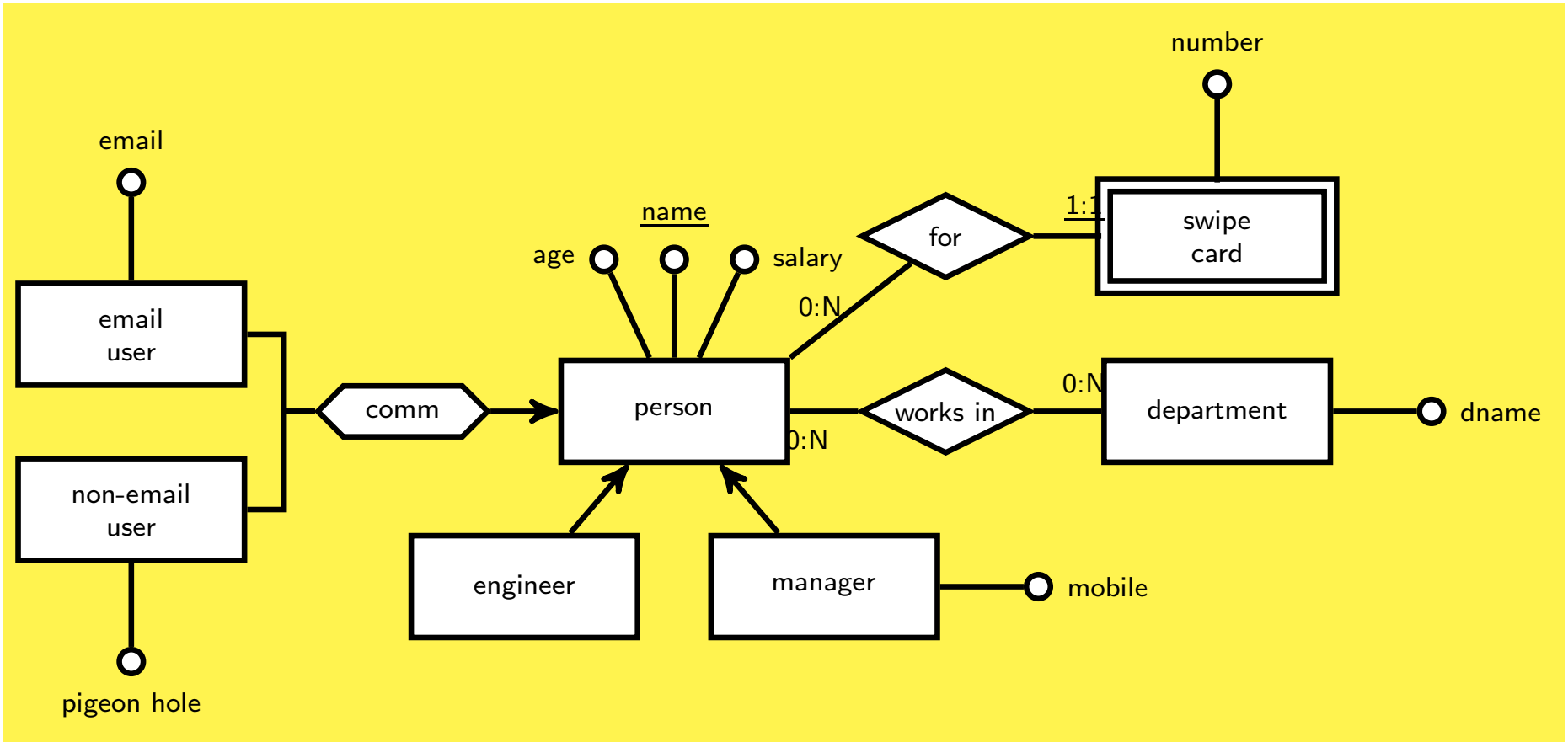
RC recoverable
ACA avoid cascading aborts
ST strict

DR dirty read
DW dirty write

SR serialisable
CSR conflict serialisable
VSR view serialisable

LU lost update
IA inconsistent analysis
WS write skew
PR phantom read

ER Diagram


number
email
name
age
salary
for
1:1
swipe card
email user
0:N
comm
person
works in
0:N
department
dname
non-email user
0:N
engineer
manager
mobile
pigeon hole