X THE PLACE OF USEFUL LEARNING

University of Strathclyde Glasgow

Database Fundamentals – CS990

Database and Web Systems Development - CS952

Image source: https://www.strath.ac.uk/branding/

ERD Mapping
Relational Algebra
SQL

Course Content

1. Introduction to Relational Databases *(Introduction + Relational Model)*
2. Data Modelling - *(Entity Relationship Modelling + The Enhanced Entity Relationship Model)*
3. Database Design and SQL - *(Logical modelling + Introduction to SQL)*
4. Further SQL - *(Advanced SQL queries + Creating tables with SQL)*
5. Normalisation - *(Normalisation to second normal form + Third normal form)*

Contents

Data Modelling

- Logical design
  - *Design rules*
  - *Example*
  - *Design decisions involving sub-types and supertypes*

Relational Algebra

SQL

To do…

- Course work 1
- Self Assessment
- Class Test 1

Logical design rules

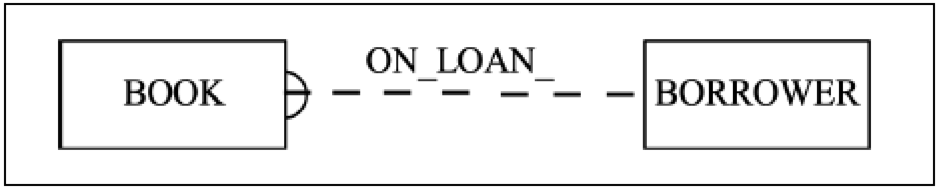
The basis of logical design is to **minimise the number of separate tables** needed to represent a database design without the need to use **null values** to indicate values that are not relevant.

STEP 1: DERIVING SKELETON TABLES

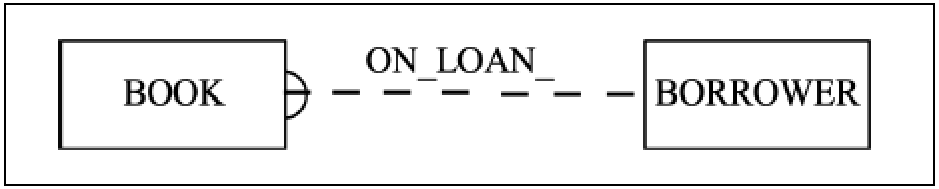
1.1 Each entity from E - R diagram becomes a table.

1.2 Relationships

The objective of this step is to represent the relationships using as few tables as possible but being careful not to require the storage of null values.



...Some borrowers have no books on loan and some books are not on loan...



One to many

One possible approach: Each relationship becomes a relation with **two domains**: the **identifiers** of the entities related.

BOOK

NO	TITLE
001	…….
002	…….
003	…….
004	…….

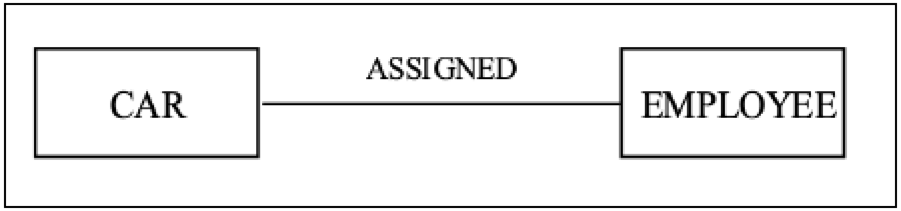
ON_LOAN_TO

BOOK	BOR
001	A12
002	A13
003	A14
004	A14

BORROWER

NO	NAME
A12	…….
A13	…….
A14	…….
A15	…….

This borrower has no books on loan, so is an optional relationship on borrower.



One to One

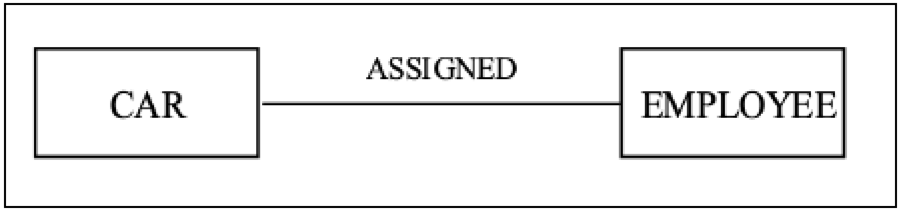
...**every car** in the pool is assigned to an employee. **Each employee** is provided with a car.

CAR

<u>REG</u>	MAKE
EK70 VWB	FORD
DR69 NLM	VW
FG70 WMO	HONDA

EMPLOYEE

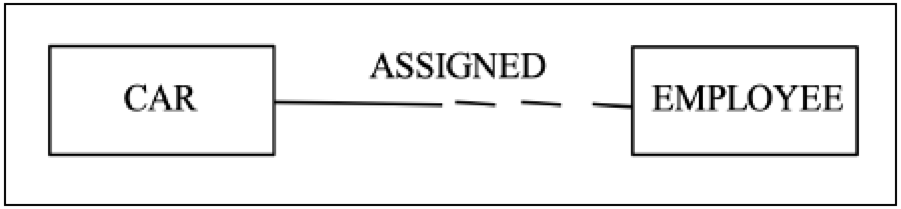
<u>NO</u>	NAME
0035	BILL
0124	FRED
0103	JIM



...**every car** in the pool is assigned to an employee. Each employee is provided with a car.

EMP_CAR

<u>NO</u>	**NAME**	**REG**	**MAKE**
0035	BILL	EK70 VWB	FORD
0124	FRED	DR69 NLM	VW
0103	JIM	FG70 WMO	HONDA



One to One
(optional)

...**every car** in the pool is assigned to a **specific employee**. Most employees are **not** provided with cars.

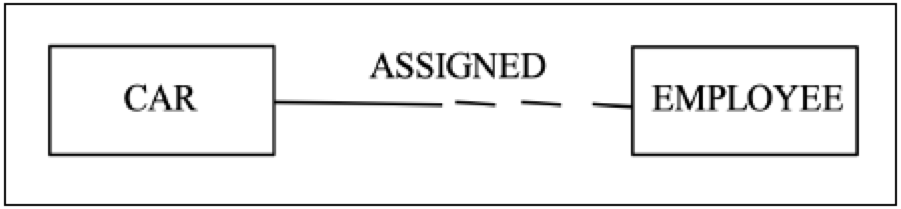
CAR

REG	**MAKE**
EK70 VWB	FORD
DR69 NLM	VW

EMPLOYEE

NO	**NAME**
0035	FRED
0005	PAT
0111	MIKE
0124	BILL
0103	JIM

Problem: Null Values



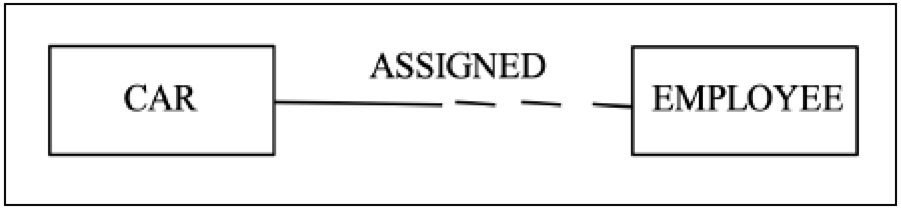
...**every car** in the pool is assigned to a **specific employee**.

Most employees are **not** provided with cars.

EMP_CAR

NO	**NAME**	**REG**	**MAKE**
0035	FRED	EK70 VWB	FORD
0005	PAT	DR69 NLM	VW
0111	MIKE	**NULL**	**NULL**
0124	BILL	**NULL**	**NULL**
0103	JIM	**NULL**	**NULL**

Solution



...every car in the pool is assigned to a specific employee. Most employees are not provided with cars.

A better solution: post the identifier of the **non-obligatory entity** to the **obligatory entity** and maintain **two tables.**

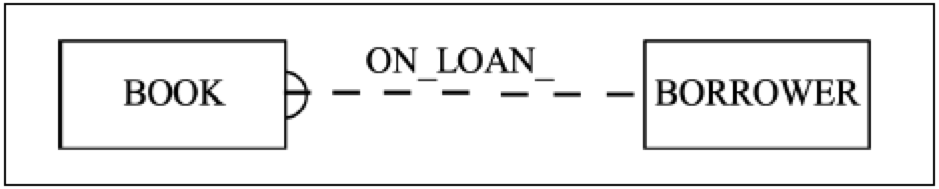
CAR

REG	**MAKE**
EK70 VWB	FORD
DR69 NLM	VW

EMP

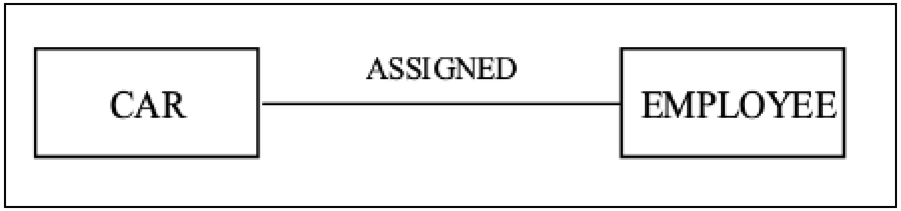
NO	**NAME**
0035	FRED
0005	PAT
0001	MIKE
0124	BILL
0103	JIM

Guidance Table



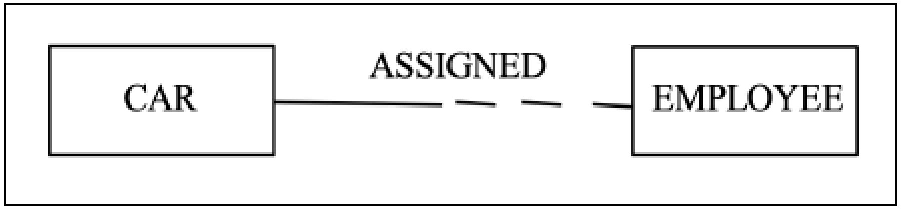
	1:1	1:N	N:M
Obligatory on neither	New table to represent relationship. Post identifiers as candidate key	New table to represent relationship. Post identifiers as candidate key	New table to represent relationship. Post identifiers as candidate key
Obligatory on one	Post identifier of non-obligatory to obligatory table	New table to represent relationship. Post identifiers as candidate key	-
Obligatory on many	-	Post identifier of 'one' table to 'many' table	New table to represent relationship. Post identifiers as candidate key
Obligatory on both	Post all attribute into one table	Post identifier of 'one' table to 'many' table	New table to represent relationship. Post identifiers as candidate key

Guidance Table



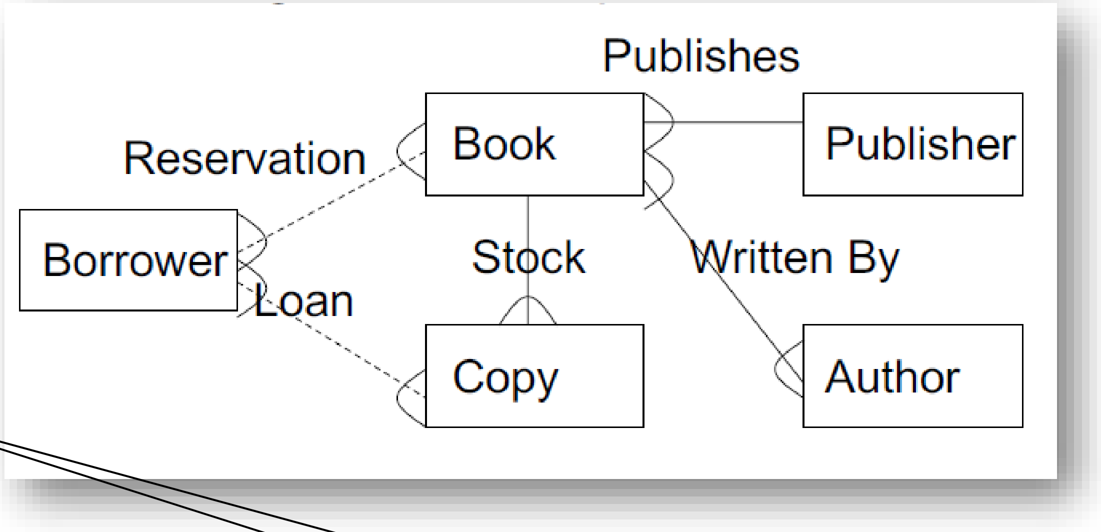
	1:1	1:N	N:M
Obligatory on neither	New table to represent relationship. Post identifiers as candidate key	New table to represent relationship. Post identifiers as candidate key	New table to represent relationship. Post identifiers as candidate key
Obligatory on one	Post identifier of non-obligatory to obligatory table	New table to represent relationship. Post identifiers as candidate key	-
Obligatory on many	-	Post identifier of 'one' table to 'many' table	New table to represent relationship. Post identifiers as candidate key
Obligatory on both	Post all attribute into one table	Post identifier of 'one' table to 'many' table	New table to represent relationship. Post identifiers as candidate key

Guidance Table



	1:1	1:N	N:M
Obligatory on neither	New table to represent relationship. Post identifiers as candidate key	New table to represent relationship. Post identifiers as candidate key	New table to represent relationship. Post identifiers as candidate key
Obligatory on one	Post identifier of non-obligatory to obligatory table	New table to represent relationship. Post identifiers as candidate key	-
Obligatory on many	-	Post identifier of 'one' table to 'many' table	New table to represent relationship. Post identifiers as candidate key
Obligatory on both	Post all attribute into one table	Post identifier of 'one' table to 'many' table	New table to represent relationship. Post identifiers as candidate key

ERD to Relations - Design rules example



Identifiers

- **Entities**:
  - Borrower (Borrower #, ...)   Book (ISBN, ...)
  - Copy (Accession #, ...)   Publisher (Pub-Code, ...)
  - Author (Author #, ...)

Name	Entities	Degree	Optionality
Loan	Copy,Borrower	N:M	Optional on both
Reservation	Book, Borrower	N:M	Optional on both
Stock	Book, Copy	1:N	Obligatory on both
Publishes	Publisher, Book	1:N	Obligatory on both
Written_By	Author, Book	N:M	Obligatory on both

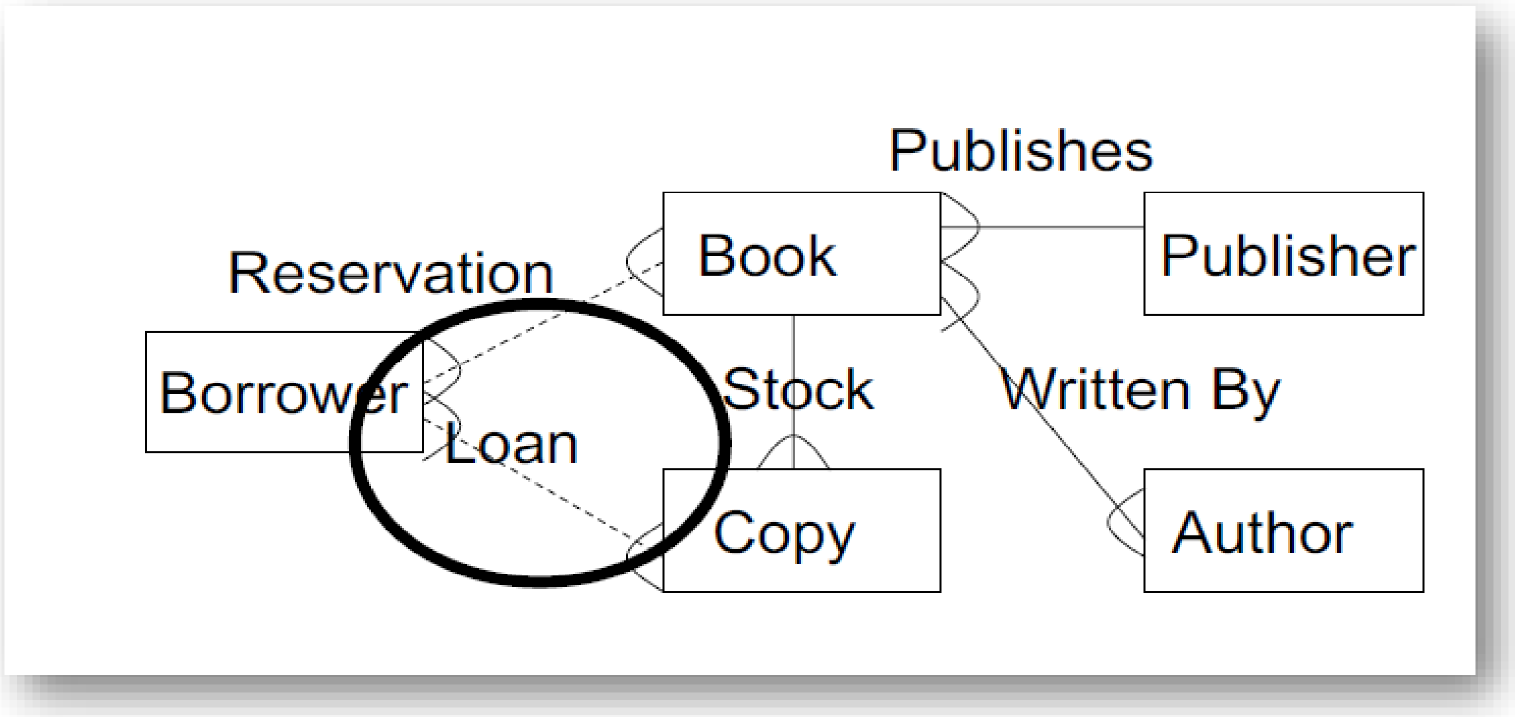


Table structures:

Borrower (<u>Borrower #</u>, ...)

Book (<u>ISBN</u>, ...)

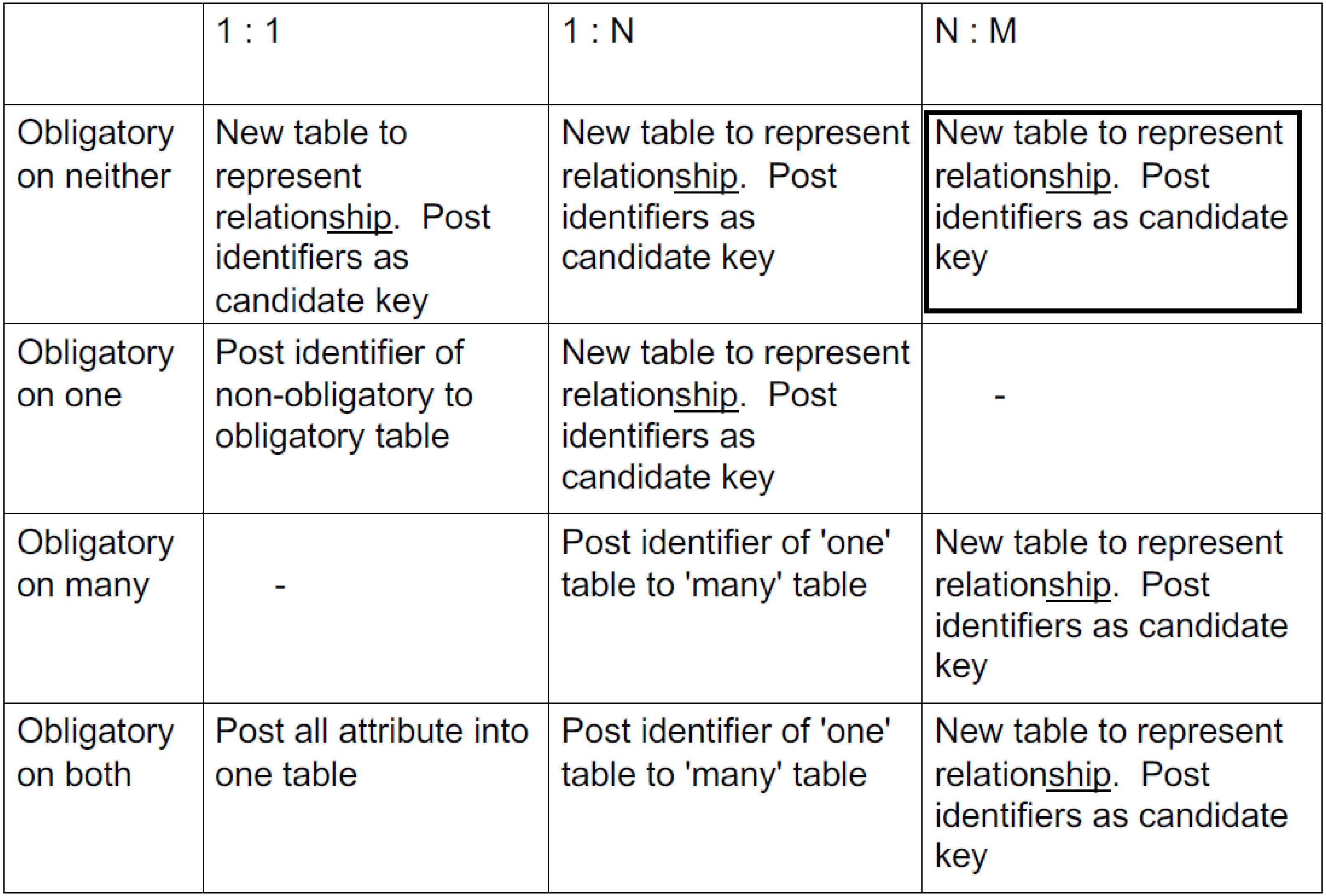
Copy (<u>Accession #</u>, ...)

Publisher (<u>Pub-Code</u>, ...)

Author (<u>Author #</u>, ...)

Primary Keys

Design rules...



	1 : 1	1 : N	N : M
Obligatory on neither	New table to represent relationship. Post identifiers as candidate key	New table to represent relationship. Post identifiers as candidate key	New table to represent relationship. Post identifiers as candidate key
Obligatory on one	Post identifier of non-obligatory to obligatory table	New table to represent relationship. Post identifiers as candidate key	-
Obligatory on many	-	Post identifier of 'one' table to 'many' table	New table to represent relationship. Post identifiers as candidate key
Obligatory on both	Post all attribute into one table	Post identifier of 'one' table to 'many' table	New table to represent relationship. Post identifiers as candidate key

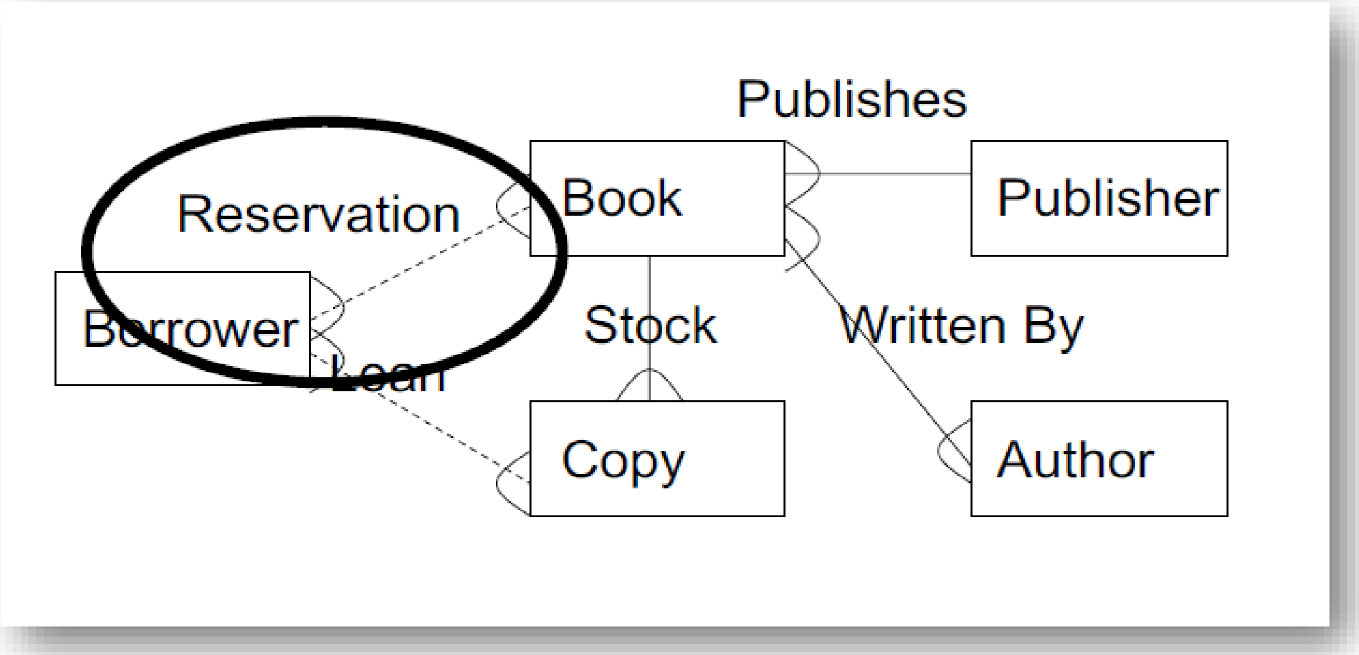


Table Structures

- **Borrower** (Borrower #, ...)
- **Book** (ISBN, ...)
- **Copy** (Accession#, ...)
- **Publisher** (Pub-Code, ...)
- **Author** (Author#, ...)
- **Loan** (Borrower#, Accession #)
- **Reservation** (Borrower#, ISBN)

This yields

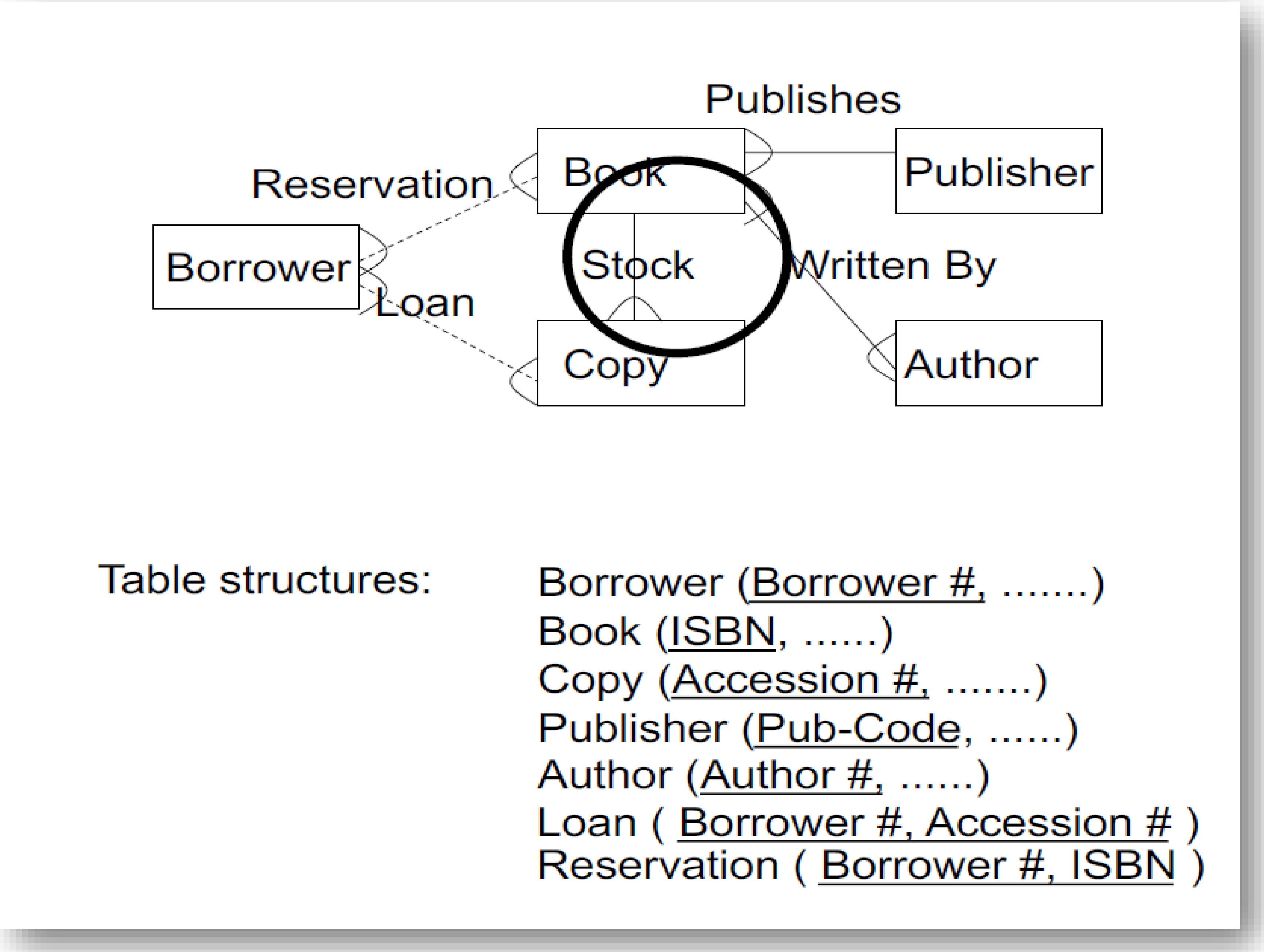


Table structures:

Borrower (Borrower #, ...)
Book (ISBN, ...)
Copy (Accession #, ...)
Publisher (Pub-Code, ...)
Author (Author #, ...)
Loan (Borrower #, Accession #)
Reservation (Borrower #, ISBN)

	1 : 1	1 : N	N : M
Obligatory on neither	New table to represent relationship. Post identifiers as candidate key	New table to represent relationship. Post identifiers as candidate key	New table to represent relationship. Post identifiers as candidate key
Obligatory on one	Post identifier of non-obligatory to obligatory table	New table to represent relationship. Post identifiers as candidate key	-
Obligatory on many	-	Post identifier of 'one' table to 'many' table	New table to represent relationship. Post identifiers as candidate key
Obligatory on both	Post all attribute into one table	Post identifier of 'one' table to 'many' table	New table to represent relationship. Post identifiers as candidate key

This yields

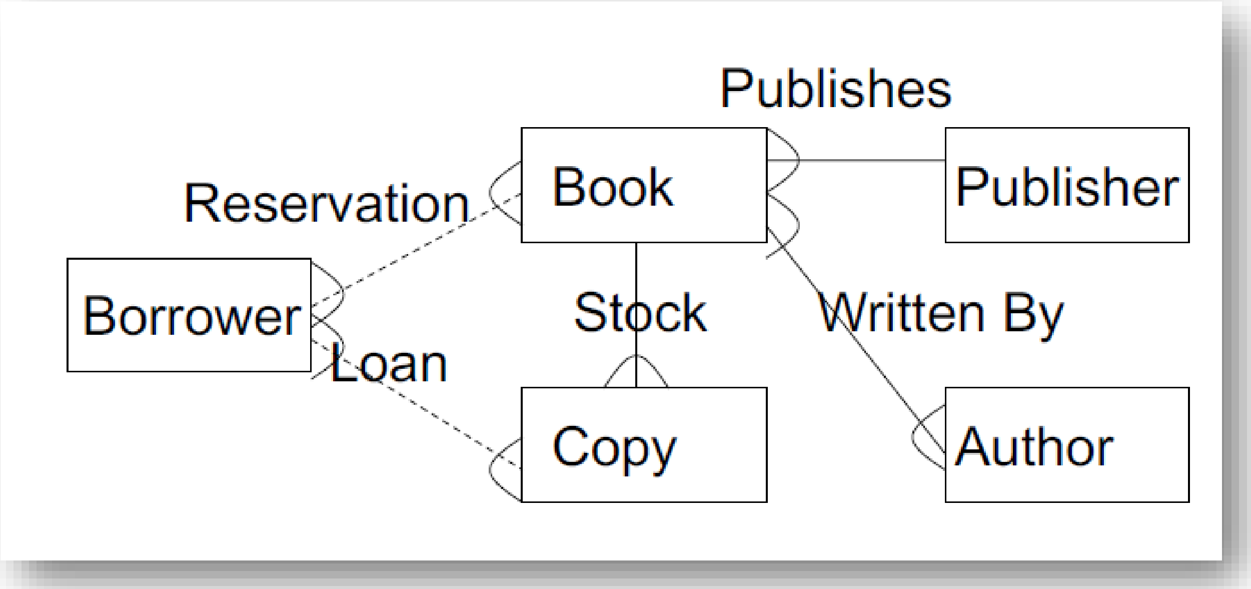


Table structures

Borrower (Borrower #, ...)

Book (ISBN, Pub-Code,......)

Copy (Accession#, ISBN..)

Publisher (Pub-Code, ...)

Author (Author #, ...)

Loan (Borrower#, Accession #)

Reservation (Borrower#, ISBN)

Written-by (Author #, ISBN)

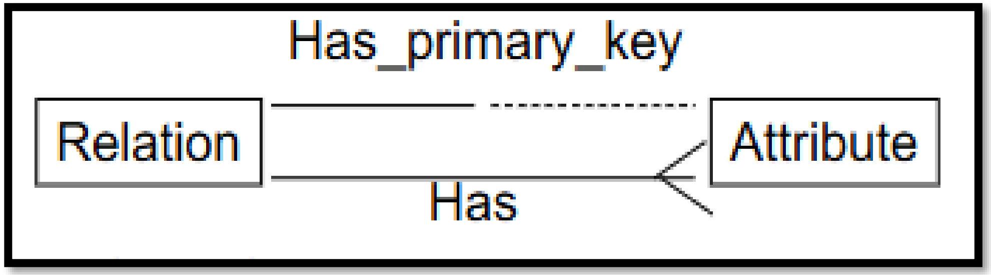
Finally: Adding the attributes

STEP : ASSIGN THE ATTRIBUTES

- **Borrower** (Borrower #, Borrower-Name, Borrower-Address)
- **Book** (ISBN, Title, Date, Pub-Code)
- **Copy** (Accession #, ISBN, Price)
- **Publisher** (Pub-Code, Pub-Name)
- **Author** (Author#, Author-Name)
- **Loan** (Borrower#, Accession #)
- **Reservation** (Borrower#, ISBN)
- **Written-by** (Author#, ISBN)

...each relation has a name, an owner and a number of attributes. Each attribute has a name and a data type. Every relation has one attribute that acts as the primary key for the relation

Conceptual model:



Entities: Relation (R_id, Name, Owner)
Attribute: (A_id, Name, Data_type)

Relationships:

Name	**Entities**	**Degree**	**Optionality**
Has	Relation, Attribute	1:N	Obligatory on both
Has_Primary_Key	Relation, Attribute	1:1	Optional on Attribute Obligatory on Relation

Logical model:

Tables: Relation (<u>R_id</u>, Name, Owner, A_id)
Attribute(<u>A_id</u>, Name, Data_type, R_id)

CS990

CS952

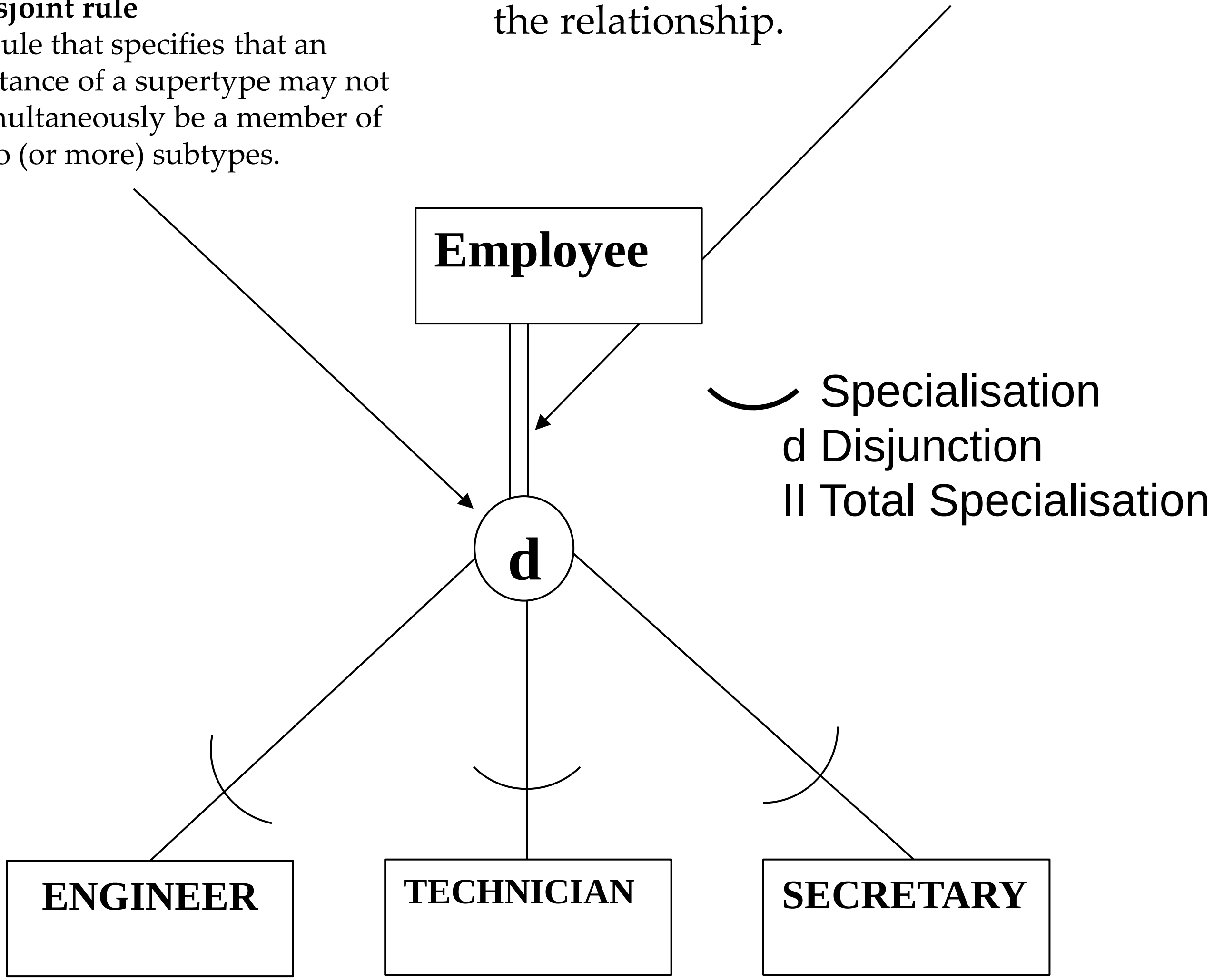
31pt1i

EERDs – Some more…

EER model provides more meaning than can be incorporated in the entity relationship model.

It is based on developing the features for representing **supertype/subtype** relationships.

Specialisation is the process of defining a set of subclasses. There may be several specialisations of the same entity type



Disjoint rule
A rule that specifies that an instance of a supertype may not simultaneously be a member of two (or more) subtypes.

Total specialization rule
A rule that specifies that each entity instance of a supertype must be a member of some subtype in the relationship.

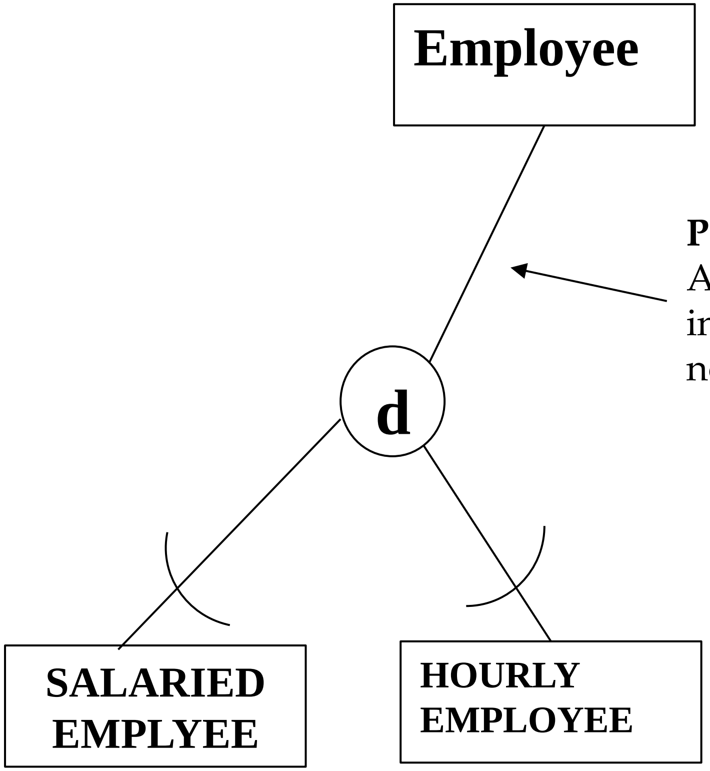
⌣ Specialisation
d Disjunction
II Total Specialisation

EERDs – Some more…

EER model provides more meaning than can be incorporated in the entity relationship model.

It is based on developing the features for representing **supertype/subtype** relationships.

Specialisation is the process of defining a set of subclasses. There may be several specialisations of the same entity type



Partial specialization rule
A rule that specifies that an entity instance of a supertype is allowed not to belong to any subtype

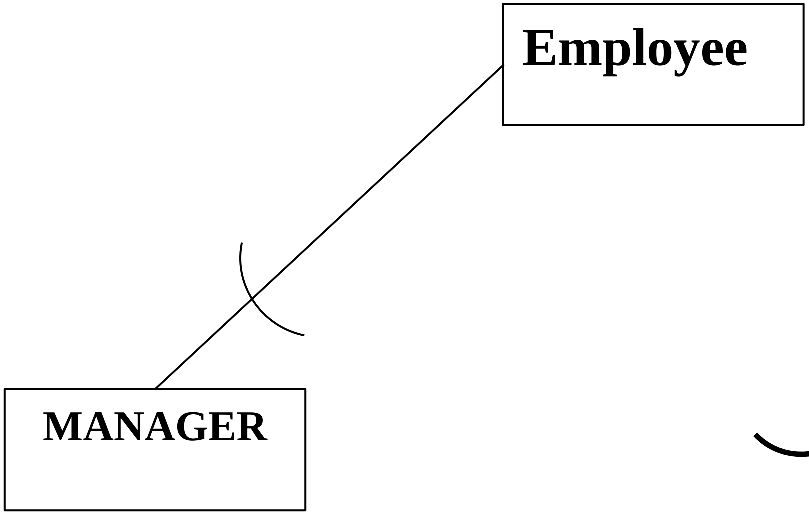
⌣ Specialisation
d Disjunction
II Total Specialisation

EERDs – Some more…

EER model provides more meaning than can be incorporated in the entity relationship model.

It is based on developing the features for representing **supertype/subtype** relationships.

Specialisation is the process of defining a set of subclasses. There may be several specialisations of the same entity type



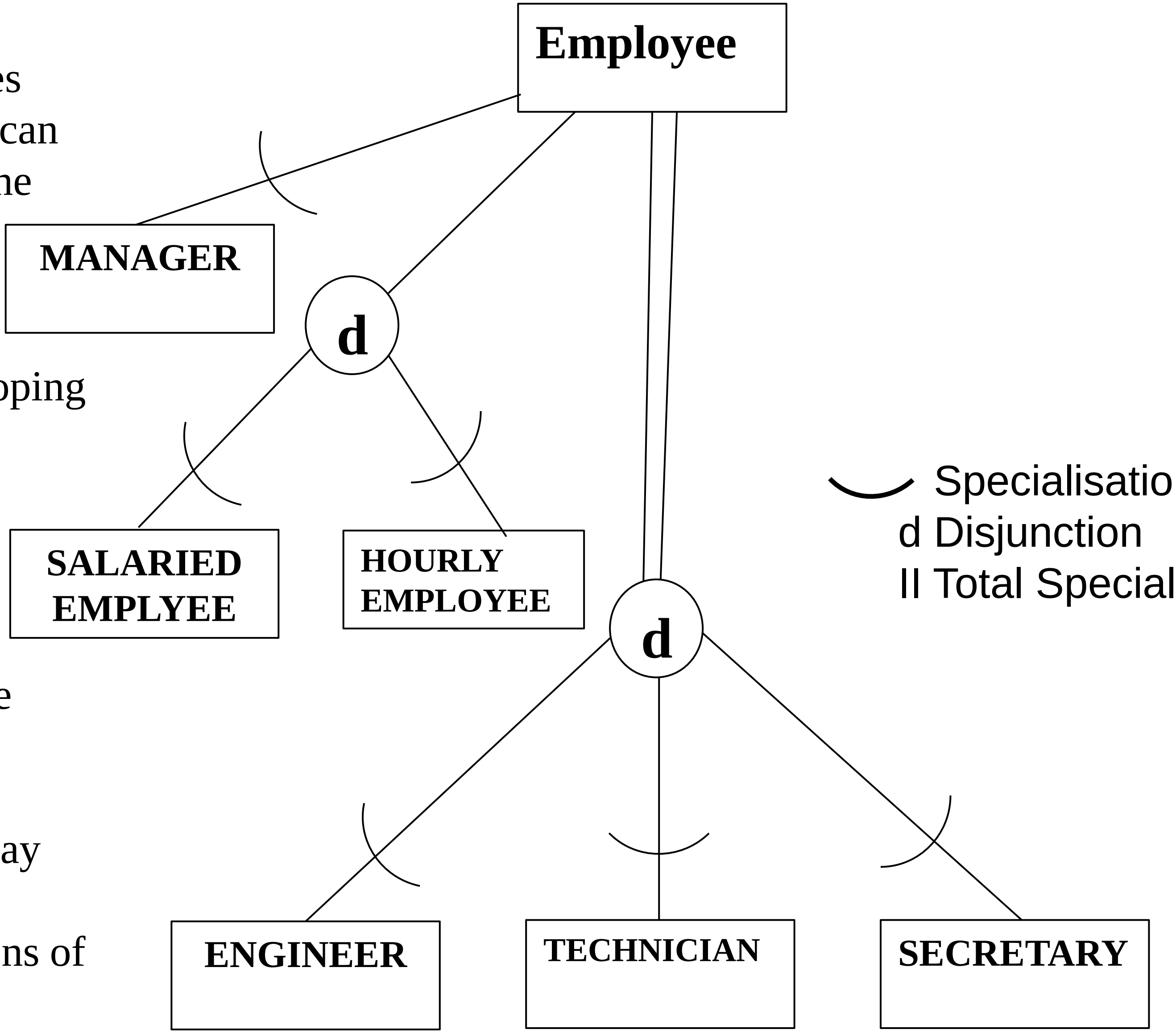
⌣ Specialisation
d Disjunction
II Total Specialisation

EERDs – Some more…

EER model provides more meaning than can be incorporated in the entity relationship model.

It is based on developing the features for representing **supertype/subtype** relationships.

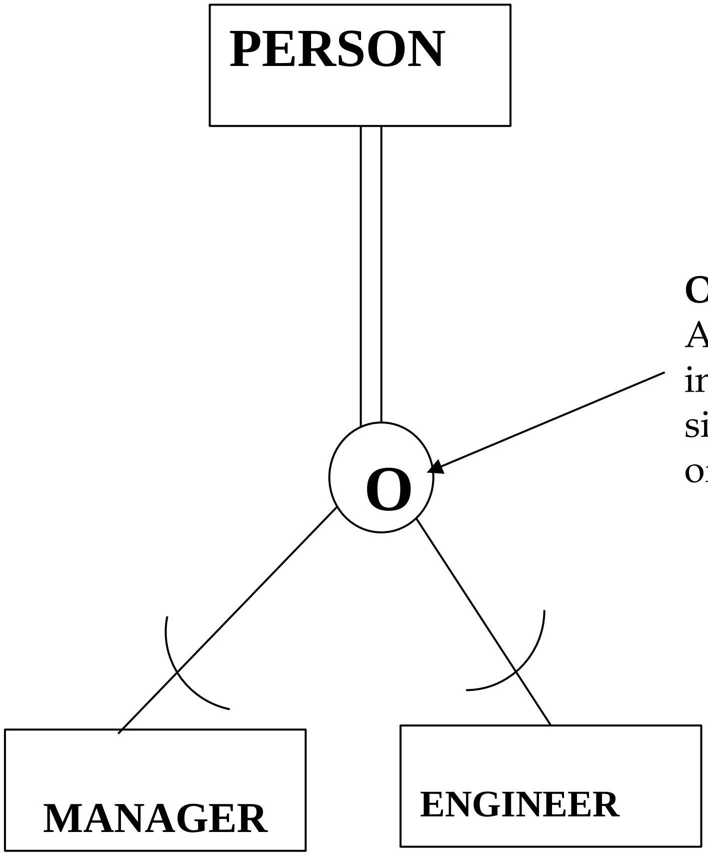
Specialisation is the process
of defining a set of subclasses. There may be
several specialisations of the
same entity type



◡ Specialisation
d Disjunction
II Total Specialisation

Subsets may overlap.

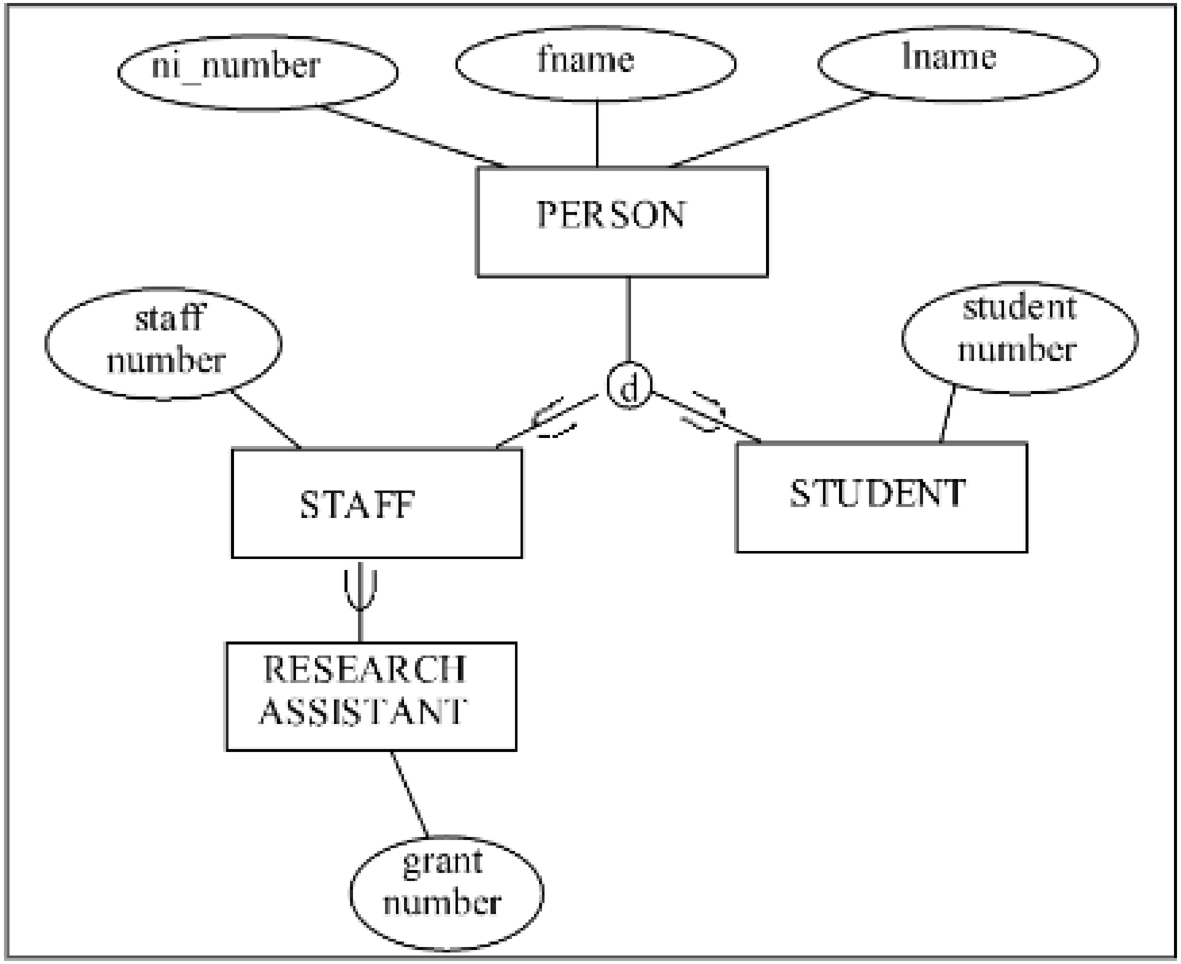
In this example **PERSONS** are specialized as either **ENGINEERS** or **MANAGERS** but some people are both engineers and managers.



Overlap rule
A rule that specifies that an instance of a supertype may simultaneously be a member of two (or more) subtypes.

Converting extended entity relationship model to a relational schema (logical model)

There are three possibilities for converting super-type/sub-type entities to a relational schema (logical model).

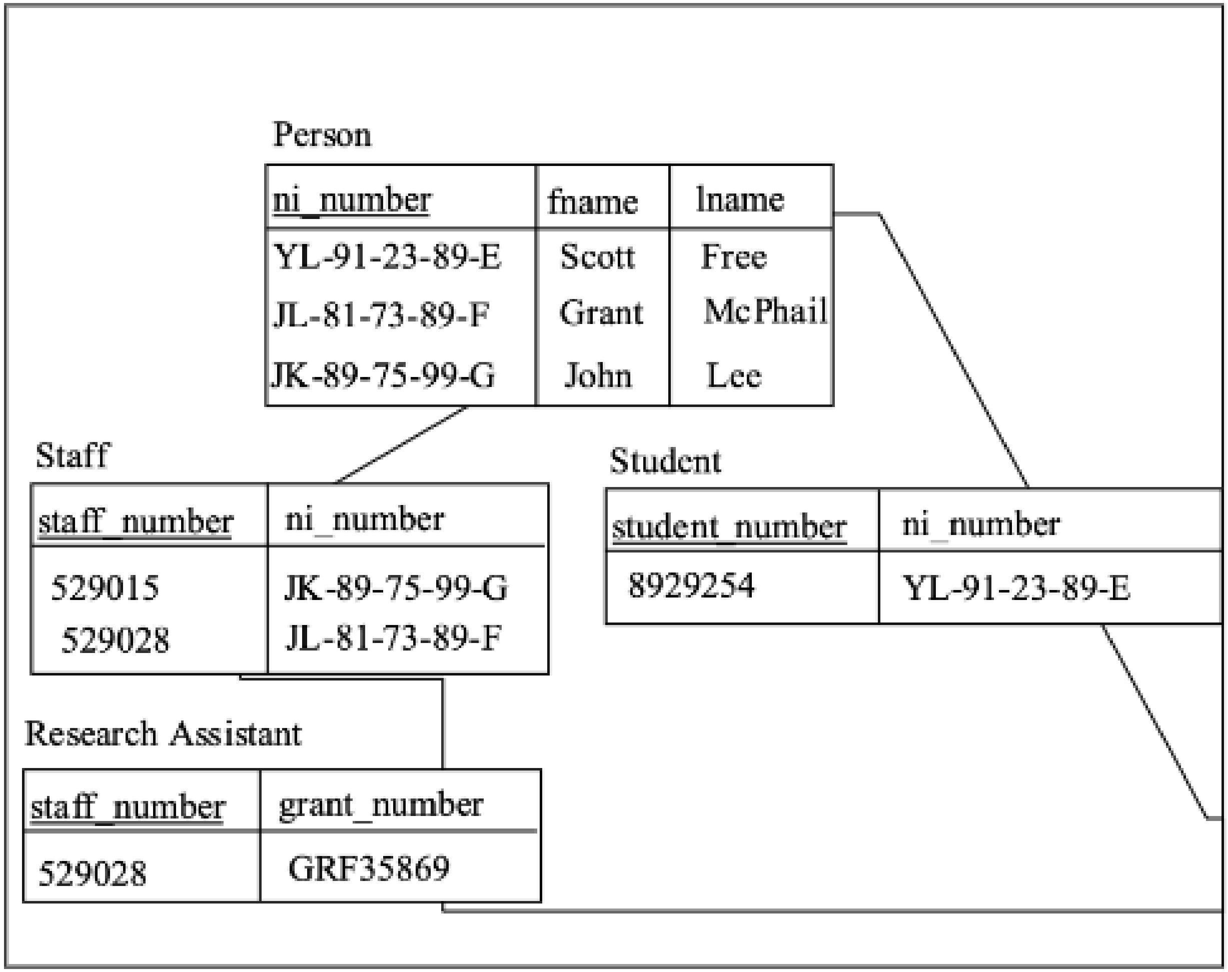


Separate Relations

- Create separate relations for the super-type and sub-type entities. Post the identifiers from the super-type to the sub-type

Advantage:
No null values

Disadvantage:
Generates many tables



Person

ni_number	fname	lname
YL-91-23-89-E	Scott	Free
JL-81-73-89-F	Grant	McPhail
JK-89-75-99-G	John	Lee

Staff

staff_number	ni_number
529015	JK-89-75-99-G
529028	JL-81-73-89-F

Student

student_number	ni_number
8929254	YL-91-23-89-E

Research Assistant

staff_number	grant_number
529028	GRF35869

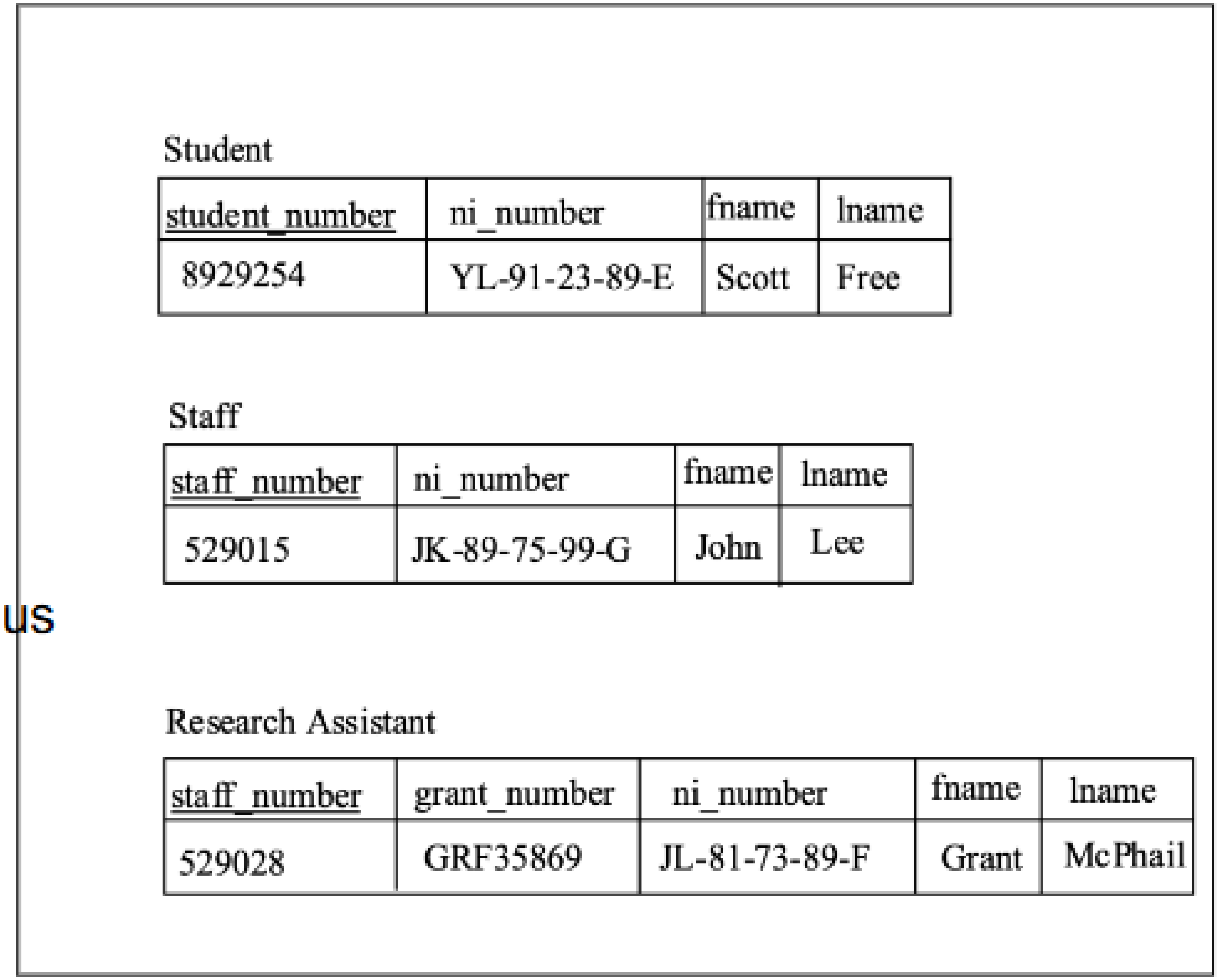
Single Relation

Create a single relation for each subtype. Post the attributes from the super-type to each of the sub-types concerned.

Advantage:
No null values

Disadvantage:
Many tables (one less that the previous method

us



Student

student_number	ni_number	fname	lname
8929254	YL-91-23-89-E	Scott	Free

Staff

staff_number	ni_number	fname	lname
529015	JK-89-75-99-G	John	Lee

Research Assistant

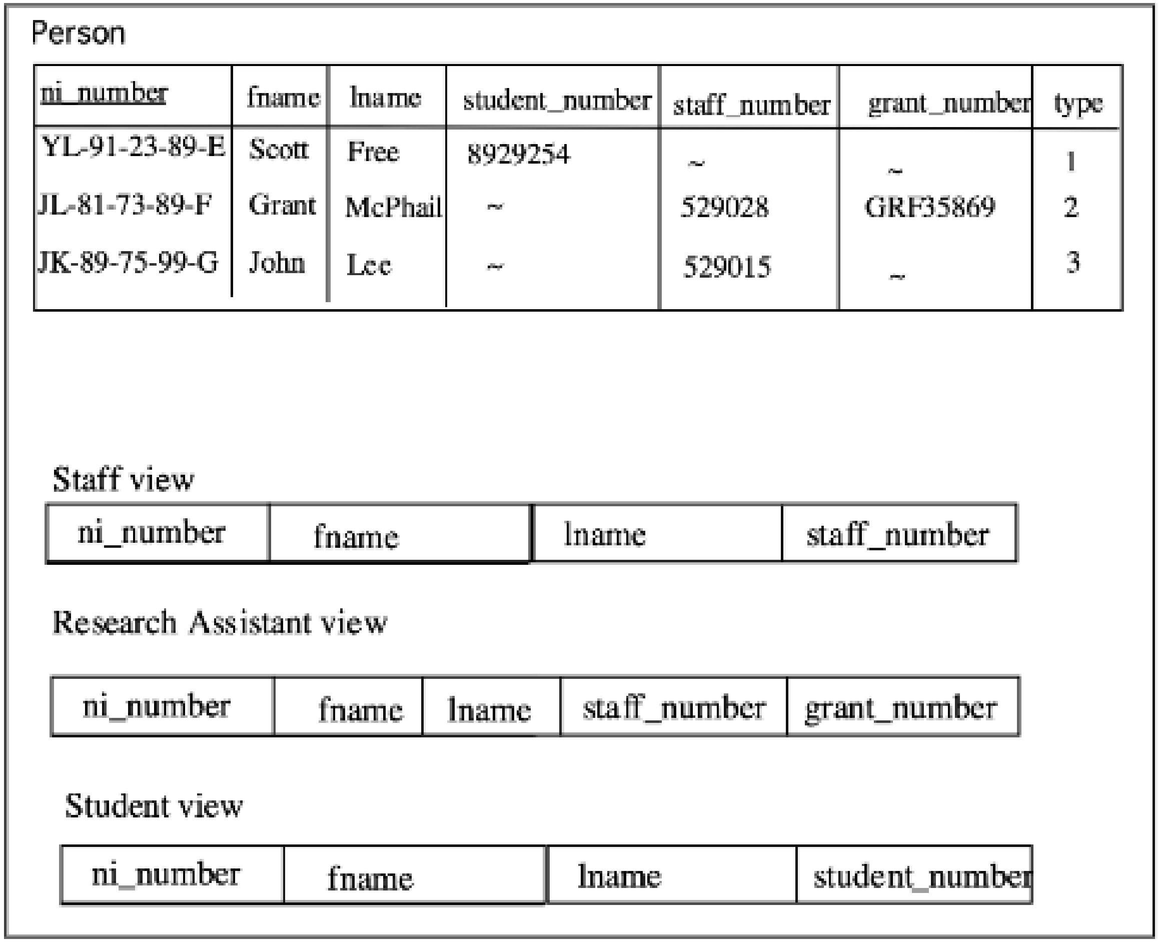
staff_number	grant_number	ni_number	fname	lname
529028	GRF35869	JL-81-73-89-F	Grant	McPhail

Single relation with UNION

Create a single relation that has the union of all the attributes from the super- type and each of the sub-types. Add in a separate attribute to denote the type of each entity instance

Advantage:
Single relation

Disadvantage:
Multiple null values



Person

ni_number	fname	lname	student_number	staff_number	grant_number	type
YL-91-23-89-E	Scott	Free	8929254	~	~	1
JL-81-73-89-F	Grant	McPhail	~	529028	GRF35869	2
JK-89-75-99-G	John	Lee	~	529015	~	3

Staff view

ni_number	fname	lname	staff_number

Research Assistant view

ni_number	fname	lname	staff_number	grant_number

Student view

ni_number	fname	lname	student_number

Relational Algebra

- The *relational algebra* is a theoretical language
  - It is the theoretical basis of query languages such as SQL.
  - It contains operators that work on one or more relations.
  - These operators give us the means to construct new relations from given ones.
  - It is similar in some ways to ordinary arithmetic (often called algebra when dealing with variables rather than explicit numbers).
- For example, with numbers we can write the following:

$5 + 9$

$(5 + 9) * 3$

-7

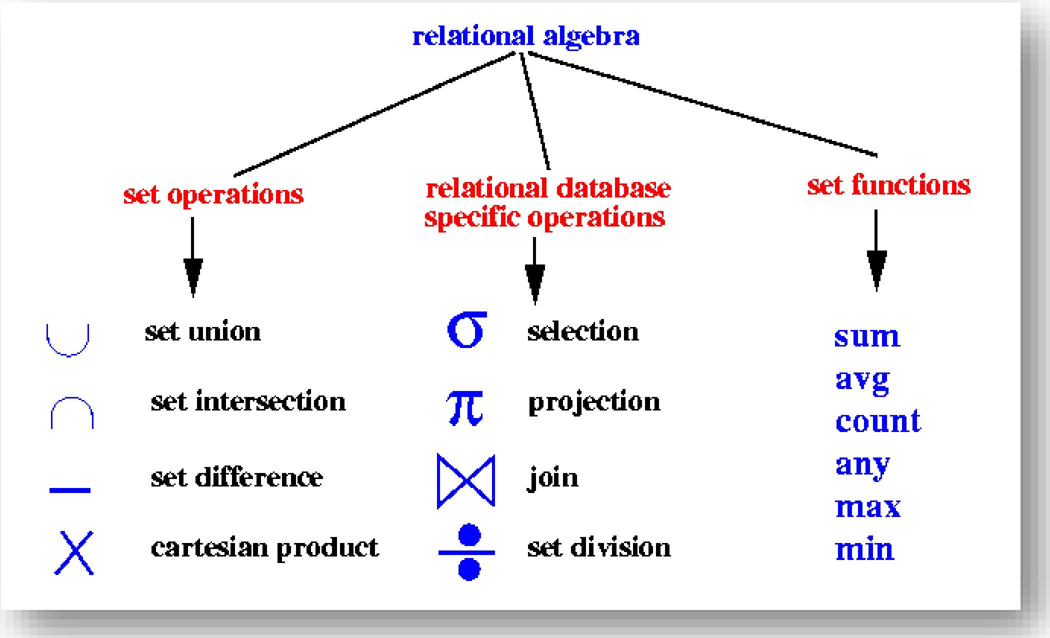
$(x + y) / 4$

Relational operators

- In 1972, Edward Codd proposed eight operators (though others have been introduced since then).
- All of these can be expressed in terms of five *fundamental* operators:
  - *Restriction*
  - *Projection*
  - *Union (or Addition)*
  - *Difference*
  - *Cartesian Product*
- The three other operators are really shortcuts for frequently used combinations of the fundamental operators. They are:
  - *Join*
  - *(Intersection and Division - these are not covered in this course)*

A note about notation

- There is no standard notation for the relational algebra operators
- The text books vary.
- Ritchie uses a rather verbose set of keywords, whereas Connolly and Begg stick to Codd’s original, very mathematical notation.
- You are allowed to use any recognised syntax in your own assessed work.



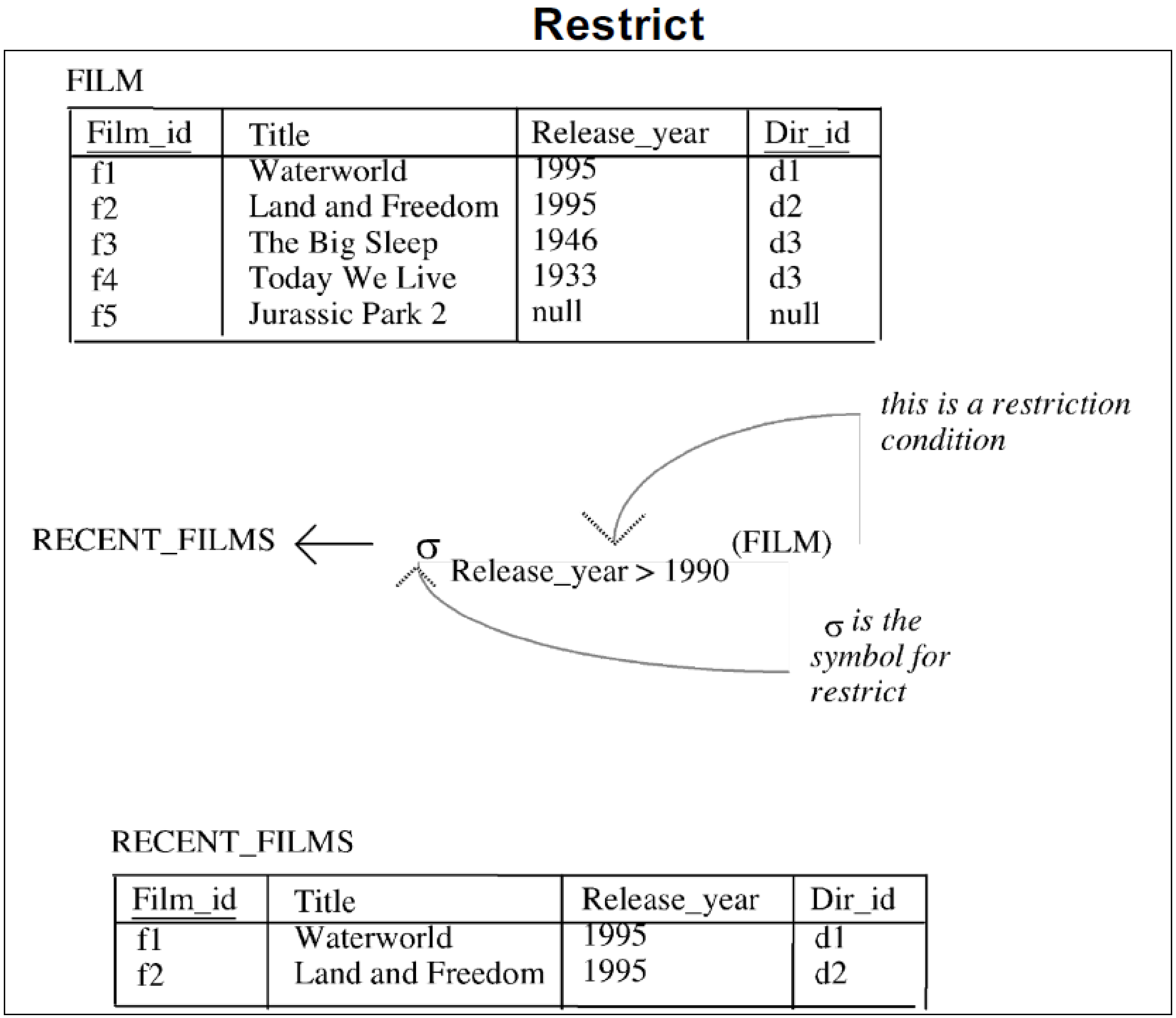
https://medium.com/@sahirnambiar/relational-algebra-the-underpinnings-of-sql-74959481231a

Data Manipulation

- Manipulation in the relational model is by a set of operators known as relational algebra.
- There are a number of operators available but three are of particular interest.
    - **RESTRICT** σ (Sigma)
    - **PROJECT** π (Pi)
    - **JOIN** ⋈ (Natural Join)
- Some other operators:
    - **UNION** ∪ (Union)
    - **DIFFERENCE** **-** (Minus)
    - **CAERTISAN PRODUCT** × (Cross Product)

Relational algebra helps us to understand SQL.

Restrict



Restrict

FILM

Film_id	Title	Release_year	Dir_id
f1	Waterworld	1995	d1
f2	Land and Freedom	1995	d2
f3	The Big Sleep	1946	d3
f4	Today We Live	1933	d3
f5	Jurassic Park 2	null	null

RECENT_FILMS ← $\sigma_{\text{Release\_year} > 1990}$ (FILM)

this is a restriction condition

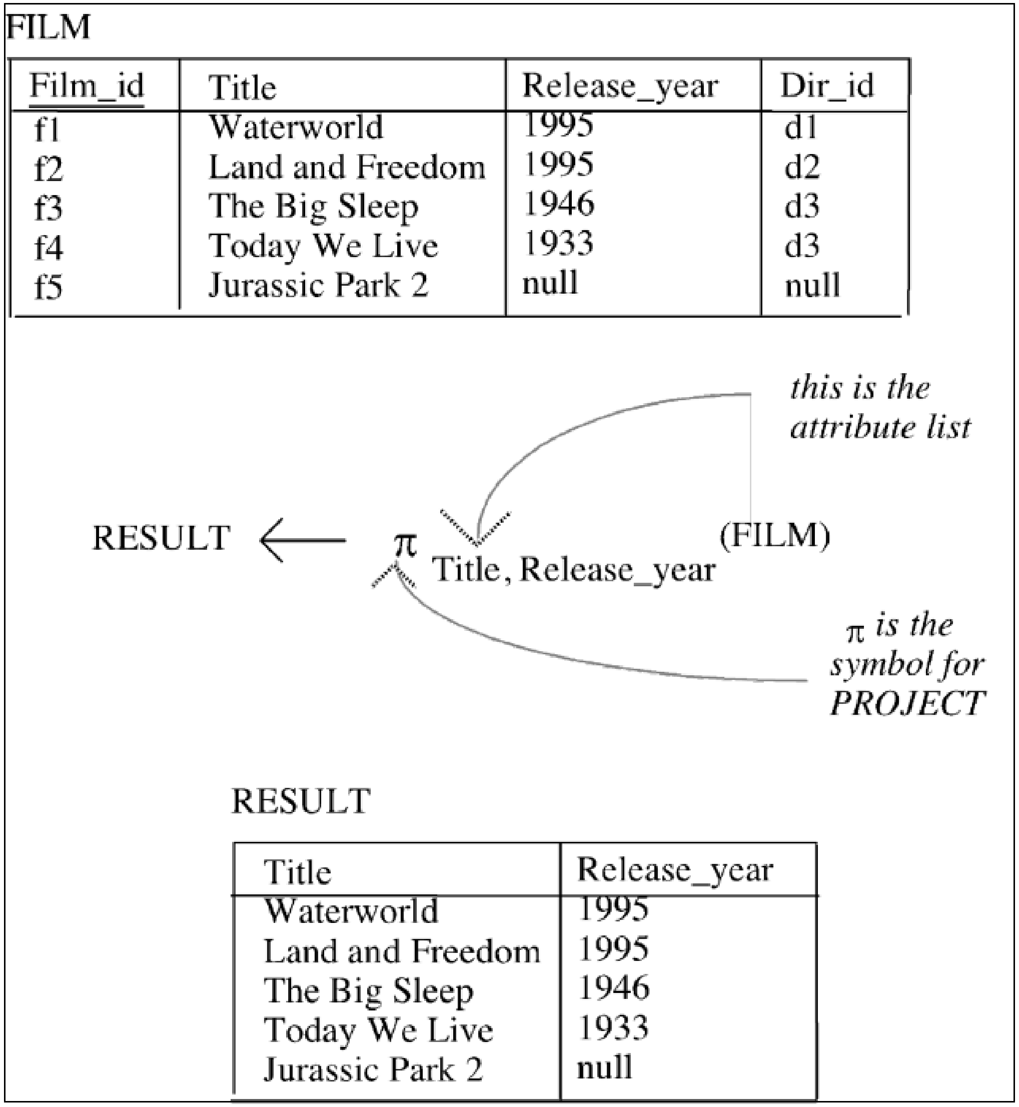
σ *is the symbol for restrict*

RECENT_FILMS

Film_id	Title	Release_year	Dir_id
f1	Waterworld	1995	d1
f2	Land and Freedom	1995	d2

Restrict: gives a horizontal 'slice' of a relation.

Project



FILM

Film_id	Title	Release_year	Dir_id
f1	Waterworld	1995	d1
f2	Land and Freedom	1995	d2
f3	The Big Sleep	1946	d3
f4	Today We Live	1933	d3
f5	Jurassic Park 2	null	null

this is the attribute list

$$\text{RESULT} \longleftarrow \pi_{\text{Title, Release\_year}}(\text{FILM})$$

π *is the symbol for PROJECT*

RESULT

Title	Release_year
Waterworld	1995
Land and Freedom	1995
The Big Sleep	1946
Today We Live	1933
Jurassic Park 2	null

Project gives a vertical 'slice' of a relation

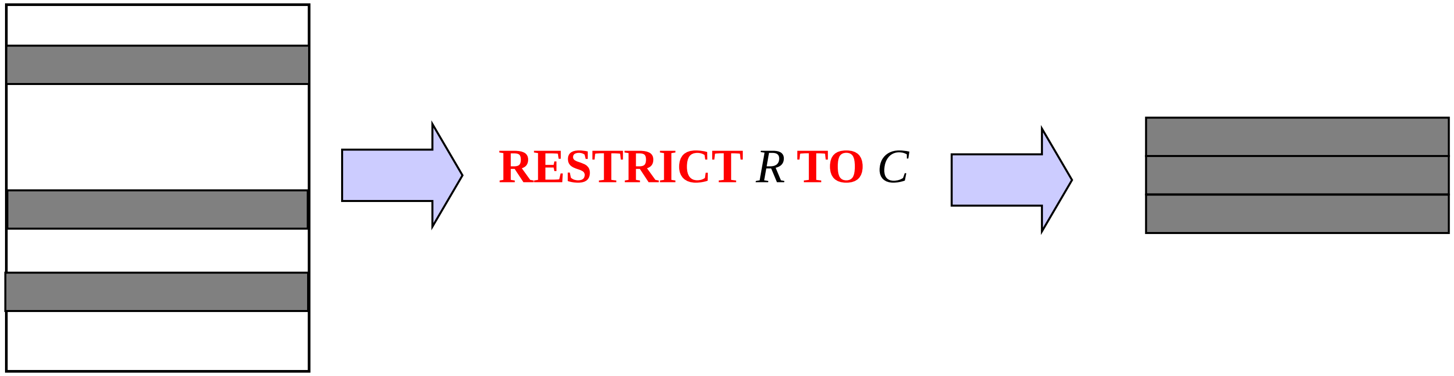
Restriction

- The restriction operator works on a single relation *R* and defines a new relation that contains only those rows of *R* that satisfy some specified condition, *C*.

- We denote the restriction operation as follows:

  RESTRICT *R* TO *C*

- The restriction operator effectively produces a subset of a relation as shown below. The shading denotes rows that satisfy condition *C*.



Restriction Example

- List all staff with a salary greater than £20,000

STAFF

Sno	*Name*	*Position*	*Sex*	*DOB*	*Salary*	*Bno*
SL21	John White	Manager	M	1-Oct-45	30000	B5
SG37	Ann Beech	Snr. Asst.	F	10-Nov-60	12000	B3
SG14	David Ford	Deputy	M	24-Mar-58	18000	B3
SA9	Mary Howe	Assistant	F	19-Feb-70	9000	B7
SG5	Susan Brand	Manager	F	3-Jun-40	24000	B3
SL41	Julie Lee	Assistant	F	13-Jun-65	9000	B5

Required information

RESTRICT STAFF **TO** Salary > 20000

Restriction operation

Sno	*Name*	*Position*	*Sex*	*DOB*	*Salary*	*Bno*
SL21	John White	Manager	M	01-Oct-45	30000	B5
SG5	Susan Brand	Manager	F	03-Jun-40	24000	B3

The result relation

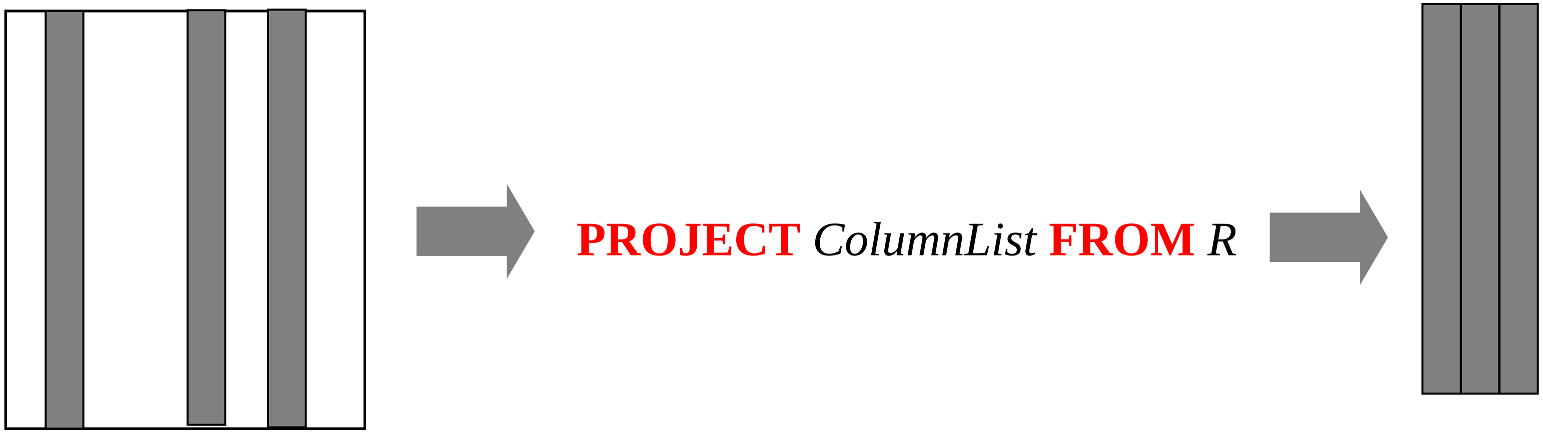
Projection

- The projection operator works on a single relation *R* and defines a relation that contains a subset of the columns of *R* (and eliminates any duplicate rows that may result).

- We denote a projection operation as follows:

  PROJECT *ColumnList* FROM *R*

- The projection operator works as follows. The shaded columns are the ones listed in *ColumnList*.



PROJECT *ColumnList* **FROM** *R*

Projection Example 1

- Produce a list of salaries for all staff, showing only the *Sno*, *Name*, and *Salary* details

STAFF

Sno	***Name***	***Position***	***Sex***	***DOB***	***Salary***	***Bno***
SL21	John White	Manager	M	1-Oct-45	30000	B5
SG37	Ann Beech	Snr. Asst.	F	10-Nov-60	12000	B3
SG14	David Ford	Deputy	M	24-Mar-58	18000	B3
SA9	Mary Howe	Assistant	F	19-Feb-70	9000	B7
SG5	Susan Brand	Manager	F	3-Jun-40	24000	B3
SL41	Julie Lee	Assistant	F	13-Jun-65	9000	B5

PROJECT *Sno*, *Name*, *Salary* **FROM** STAFF

Sno	***Name***	***Salary***
SL21	John White	30000
SG37	Ann Beech	12000
SG14	David Ford	18000
SA9	Mary Howe	9000
SG5	Susan Brand	24000
SL41	Julie Lee	9000

Projection Example 2

- Produce a list of the positions held within the organisation
  - Note that the result has fewer rows than the source relation. Why?

STAFF						
Sno	***Name***	***Position***	***Sex***	***DOB***	***Salary***	***Bno***
SL21	John White	Manager	M	1-Oct-45	30000	B5
SG37	Ann Beech	Snr. Asst.	F	10-Nov-60	12000	B3
SG14	David Ford	Deputy	M	24-Mar-58	18000	B3
SA9	Mary Howe	Assistant	F	19-Feb-70	9000	B7
SG5	Susan Brand	Manager	F	3-Jun-40	24000	B3
SL41	Julie Lee	Assistant	F	13-Jun-65	9000	B5

PROJECT *Position* **FROM** STAFF

Position
Manager
Snr. Asst.
Deputy
Assistant

Union

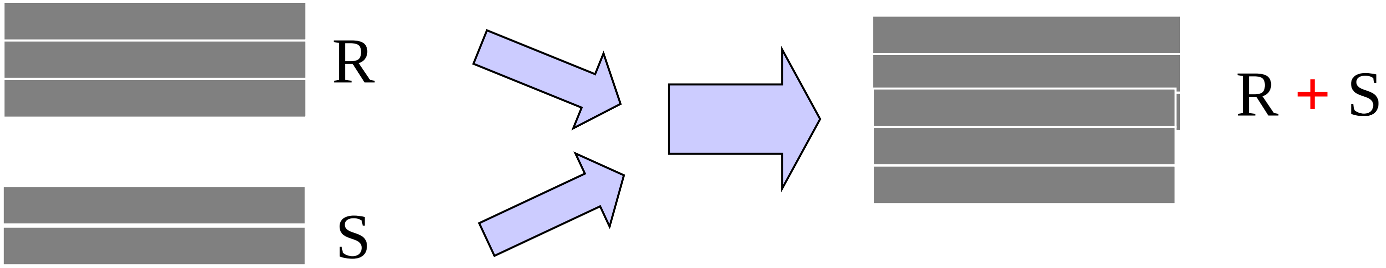
- The *union* of two relations *R* and *S* is obtained by pooling their rows into one relation, duplicate rows being eliminated.
  - It operates on two relations.

- The input relations *R* and *S* must be *union-compatible* (explained later).

- We denote a union operation as follows:

$$R + S$$

- The union operation works as follows:



Union Example

- Query: construct a list of all cities where there is **either** a Branch **or** a Property.

BRANCH

Bno	Street	Area	City	Postcode
B5	22 Deer St	Sidcup	London	SW1 4EH
B7	16 Argyll St	Dyce	Aberdeen	AB2 3SU
B3	163 Main St	Partick	Glasgow	G11 9QX
B4	32 Manse Rd	Leigh	Bristol	BS99 1NZ
B9	56 Clover Dr		London	NW10 6EU

PROPERTY

Pno	Street	Area	City	Rent
PA14	16 Holhead	Dee	Aberdeen	650.00
PL94	6 Argyll St	Kilburn	London	400.00
PG21	18 Dale Rd	Hyndland	Glasgow	600.00

(**PROJECT** City **FROM** BRANCH) **+** (**PROJECT** City **FROM** PROPERTY)

City
London
Aberdeen
Glasgow
Bristol

Union Compatibility

- The union operation can be applied only to relations which are *union compatible*.
  - Both relations must have the same degree (number of attributes).
  - Corresponding attributes must come from the same domain.
- For example, we cannot form the union of *BRANCH* and *PROPERTY*:
  - Both have 5 attributes, so that is ok.
  - But corresponding attributes do not all match (e.g. *Postcode* vs. *Rent*).
- We can sometimes solve union incompatibility problems by using *projection* (as in the previous example).

Difference

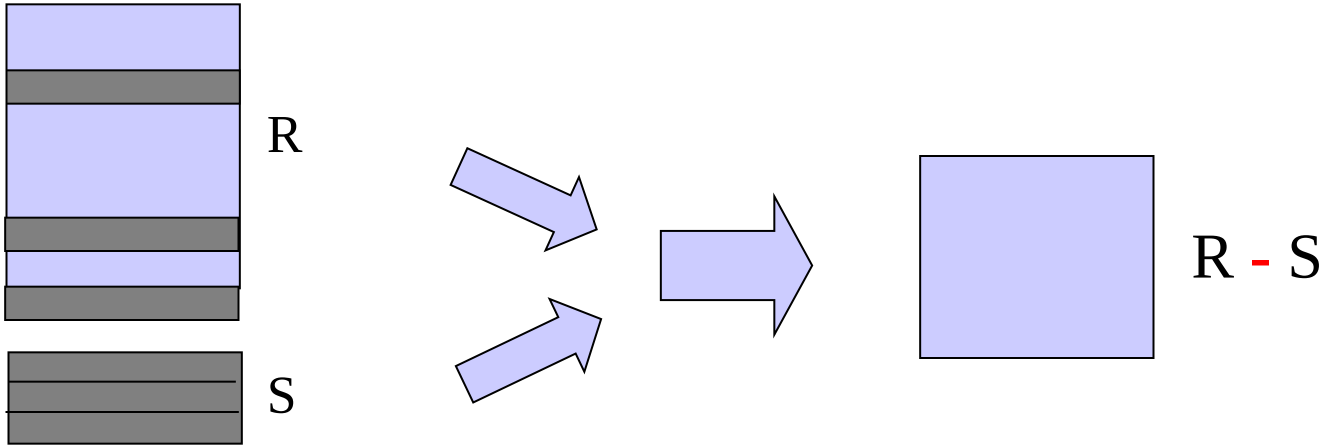
- The *difference* operator defines a relation consisting of all rows that are in relation *R*, but not in relation *S*.
  - It operates on two relations.

- As for the union operation, R and S must be *union-compatible*. (Why?)

- We denote a *difference* operation as follows:

R - S

- The difference operation works as follows:



Difference Example

- Query: construct a list of all cities where there is a Branch **but no** Property.

BRANCH				
Bno	***Street***	***Area***	***City***	***Postcode***
B5	22 Deer St	Sidcup	London	SW1 4EH
B7	16 Argyll St	Dyce	Aberdeen	AB2 3SU
B3	163 Main St	Partick	Glasgow	G11 9QX
B4	32 Manse Rd	Leigh	Bristol	BS99 1NZ
B9	56 Clover Dr		London	NW10 6EU

PROPERTY				
Pno	***Street***	***Area***	***City***	***Rent***
PA14	16 Holhead	Dee	Aberdeen	650.00
PL94	6 Argyll St	Kilburn	London	400.00
PG21	18 Dale Rd	Hyndland	Glasgow	600.00

(**PROJECT** *City* **FROM** BRANCH) **-** (**PROJECT** *City* **FROM** PROPERTY)

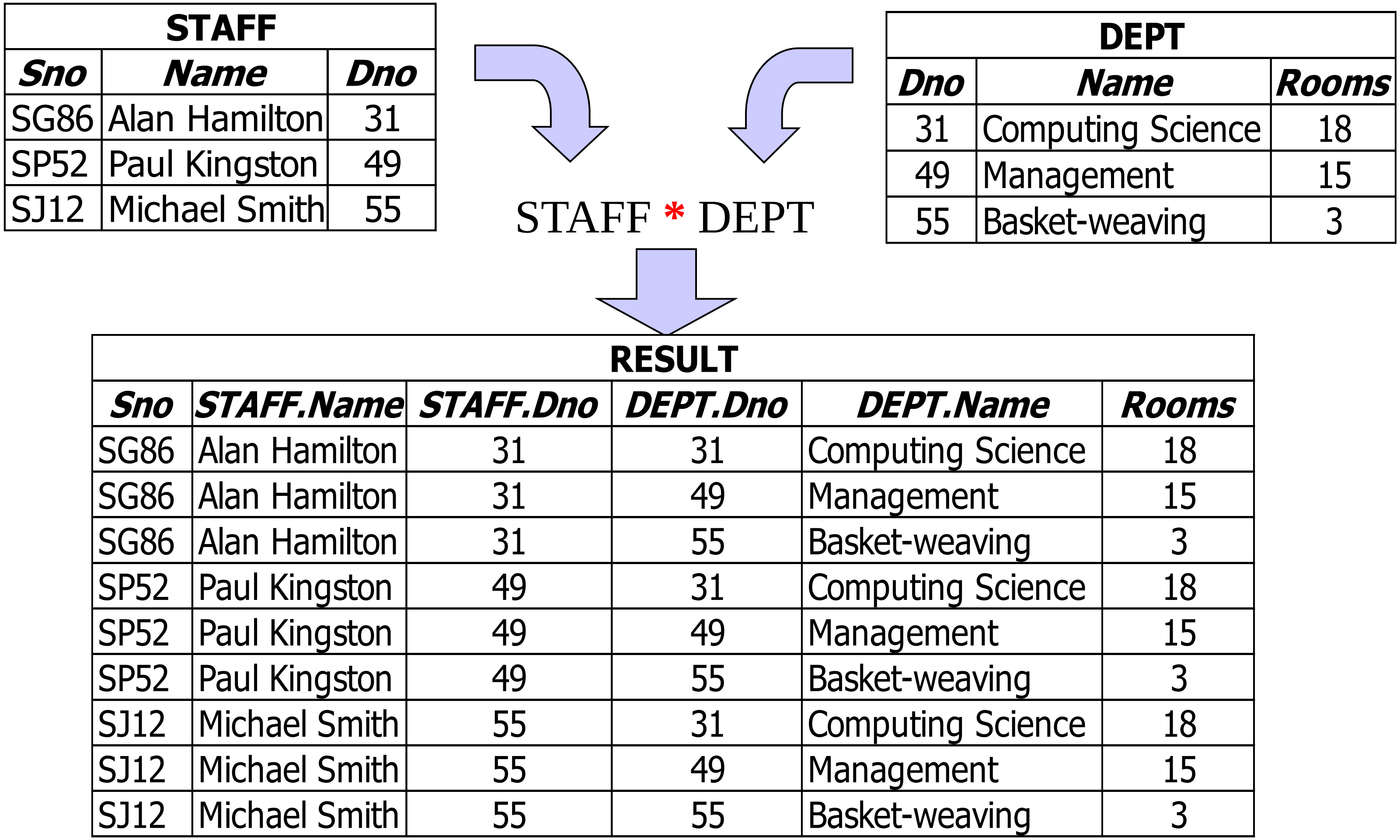
City
Bristol

Cartesian Product

- Restriction and Projection allow us to get information out of a *single* relation.
- Union and Difference allow us to manipulate *two* relations vertically (i.e. combine or remove rows).
- We often need to *combine the rows of two relations* in order to relate rows in one relation to the corresponding rows in another relation.
  - This is the purpose of the *Cartesian product* operator
- The Cartesian product operator defines a relation that includes the concatenation of *every* row of relation *R* with *every* row of relation *S*.
  - In other words, it produces every possible combination of the rows of R and S.
- We denote a Cartesian product operation as follows: $R * S$

Cartesian Product Example

- This example does not solve any particular query. It is intended to illustrate how the basic Cartesian product operation works.



STAFF		
Sno	***Name***	***Dno***
SG86	Alan Hamilton	31
SP52	Paul Kingston	49
SJ12	Michael Smith	55

DEPT		
Dno	***Name***	***Rooms***
31	Computing Science	18
49	Management	15
55	Basket-weaving	3

STAFF * DEPT

RESULT					
Sno	***STAFF.Name***	***STAFF.Dno***	***DEPT.Dno***	***DEPT.Name***	***Rooms***
SG86	Alan Hamilton	31	31	Computing Science	18
SG86	Alan Hamilton	31	49	Management	15
SG86	Alan Hamilton	31	55	Basket-weaving	3
SP52	Paul Kingston	49	31	Computing Science	18
SP52	Paul Kingston	49	49	Management	15
SP52	Paul Kingston	49	55	Basket-weaving	3
SJ12	Michael Smith	55	31	Computing Science	18
SJ12	Michael Smith	55	49	Management	15
SJ12	Michael Smith	55	55	Basket-weaving	3

Cartesian Product

- In effect, the Cartesian product operation joins the rows from the input relation to form *all possible combinations* of rows.

- How many rows and attributes is that?

- Since the two input relations may have attributes with the same name, it is necessary to avoid ending up with multiple like-named attributes in the result.

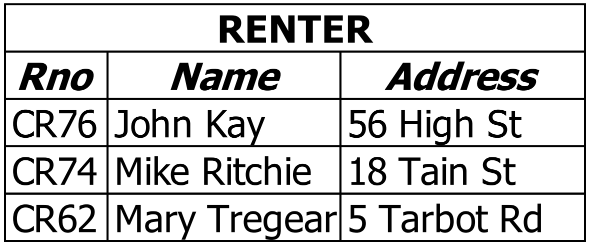
- This is avoided by prefixing the names of affected attributes with the names of the input relations:
  - Thus, in the previous example, we end up with *STAFF.Dno* and *DEPT.Dno*.
  - We also have *STAFF.Name* and *DEPT.Name.*

Using Cartesian Product

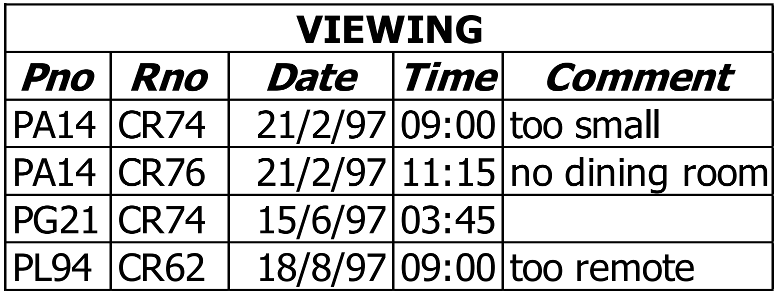
- We return to the example of the real estate agency.

- Query: list the names of renters who have viewed at least one property, together with the property number in question, and any comment.

- The input relations are shown below:



RENTER		
Rno	***Name***	***Address***
CR76	John Kay	56 High St
CR74	Mike Ritchie	18 Tain St
CR62	Mary Tregear	5 Tarbot Rd



VIEWING				
Pno	***Rno***	***Date***	***Time***	***Comment***
PA14	CR74	21/2/97	09:00	too small
PA14	CR76	21/2/97	11:15	no dining room
PG21	CR74	15/6/97	03:45	
PL94	CR62	18/8/97	09:00	too remote

- To perform this query we require renter names and numbers from the RENTER relation to be combined with the property number and comment information from the VIEWING relation

Using Cartesian Product - I

- Here is a first attempt at a solution:

R1 = **PROJECT** *Rno*, *Name* **FROM** RENTER

R2 = **PROJECT** *Pno*, *Rno*, *Comment* **FROM** VIEWING

RESULT1 = R1 ***** R2

- This is what the result looks like. What’s wrong with it?

RESULT1				
RENTER.Rno	***Name***	***Pno***	***VIEWING.Rno***	***Comment***
CR76	John Kay	PA14	CR74	too small
CR76	John Kay	PA14	CR76	no dining room
CR76	John Kay	PG21	CR74	
CR76	John Kay	PL94	CR62	too remote
CR74	Mike Ritchie	PA14	CR74	too small
CR74	Mike Ritchie	PA14	CR76	no dining room
CR74	Mike Ritchie	PG21	CR74	
CR74	Mike Ritchie	PL94	CR62	too remote
CR62	Mary Tregear	PA14	CR74	too small
CR62	Mary Tregear	PA14	CR76	no dining room
CR62	Mary Tregear	PG21	CR&4	
CR62	Mary Tregear	PL94	CR62	too remote

Using Cartesian Product - II

- The problem is that there are many rows in which *RENTER.Rno* and *VIEWING.Rno* do not match.
- We can solve this problem by using the restriction operator

  RESULT2 = RESTRICT RESULT1 TO *RENTER.Rno* = *VIEWING.Rno*

- The result of the restriction operation is shown below:

RESULT2				
RENTER.Rno	***Name***	***Pno***	***VIEWING.Rno***	***Comment***
CR76	John Kay	PA14	CR76	no dining room
CR74	Mike Ritchie	PA14	CR74	too small
CR74	Mike Ritchie	PG21	CR74	
CR62	Mary Tregear	PL94	CR62	too remote

- The effect of the restriction is to *eliminate* rows that have been formed by combining unrelated rows in the two original tables.
- Finally, a projection will give us the data we originally sought:

  RESULT = **PROJECT** *Name, Pno, Comment* **FROM** RESULT2

Questions

Home activity

- Read the following specification and design an enhanced entity relationship model for this data using the example from the video as guidance. The next lecture will start with a solution to the problems.
- An organisation keeps a register of individuals and companies that provide computing services. Individuals are identified by individual names and have an address, whereas companies are identified by trading names and have a company address. Both individuals and companies are classified as either consultants or hardware suppliers and some may be both. An average hourly rate is stored for each consultant. Consultants may be software consultants or hardware consultants or both. Internal sections of the organisation can also provide consultancy services. Each such internal section has a section name. The organisation uses its own identifiers to identify consultants.
- The organisation holds maintenance contracts with some of the hardware suppliers. Details held on maintenance contracts include start date and serial number of equipment covered.

Course Content

1. Introduction to Relational Databases *(Introduction + Relational Model)*
2. Data Modelling - *(Entity Relationship Modelling + The Enhanced Entity Relationship Model)*
3. Database Design and SQL - *(Logical modelling + Introduction to SQL)*
4. Further SQL - *(Advanced SQL queries + Creating tables with SQL)*
5. Normalisation - *(Normalisation to second normal form + Third normal form)*

X THE PLACE OF USEFUL LEARNING

University of Strathclyde Glasgow

Database Fundamentals – CS990

Database and Web Systems Development - CS952

Image source: https://www.strath.ac.uk/branding/