Topic 02: The Relational Model

Murdoch University

ICT285 Databases
Dr Danny Toohey

Topic Learning Outcomes

After completing this topic you should be able to:

- Describe the characteristics of the relational database model
- Define and give examples of the different types of keys used in the relational database model
- Explain and give examples of the relational model's integrity constraints
- Use the fundamental operators of the relational algebra (restrict, project, Cartesian product, join, intersection, difference, union, and division) to define simple queries

Resources for this topic

READING

- Kroenke & Auer, 13th ed: Chapter 3 p.150-156 (RM Terminology; Alternative Terminology); p161-163 (Keys)
- Kroenke & Auer, 14th ed: Chapter 3 p.168-172 (RM Terminology; Alternative Terminology); p177-180 (Keys)

My Unit Readings:

- Connolly & Begg: Relational Algebra
- Codd's original paper "A relational model of data for shared data banks": http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf (for interest)

Topic Outline

1. Relational model characteristics
2. Keys
3. Constraints
4. Relational algebra

Topic 02: Part 01

Relational model characteristics

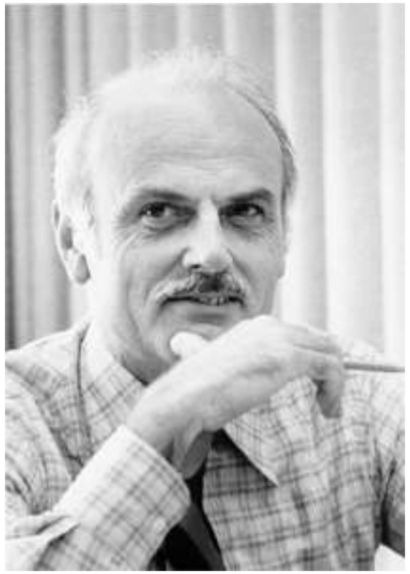
The Relational Model (RM)

- In Topic 1 we discussed a number of different data models used in databases, including hierarchical, network, object-oriented, noSQL, etc
- For the majority of this unit we will concentrate on the *relational* database model:
  - It is currently very widely used (and is expected continue to be) as the basis for commercial database systems
  - It has a strong theoretical base
  - It is widely understood

The Relational Model

The relational model was created by E.F. Codd in the late 1960’s

- RM was developed in response to problems of inflexibility associated with the navigational data models
- Relationships in RM represented through *values*, not predefined links
- This enables queries to be much more simply expressed, plus great flexibility in querying data sets
- Many relational DBMSs today; use SQL as standard query language



https://www-03.ibm.com/ibm/history/exhibits/builders/builders_codd.html
https://www-03.ibm.com/ibm/history/ibm100/us/en/icons/reldb/

Relational Model concepts

We said that a data model has 3 features:

Data Structure

- In the RM, the structure is the *relation*, made up of *tuples*, *attributes, domains*

Operators

- In the RM, the operators are defined by the relational calculus and *relational algebra*

Constraints

- In the RM, constraints are provided by *keys* and *integrity constraints*

RM concepts

Conceptually, a **relation** is a table of values:

- Each ***tuple*** (row) in the table represents a collection of related data values
- Each ***attribute*** (column) of the table specifies how to interpret the data values in each row
- The type of data (or set of allowable values) that can appear in each attribute is called the ***domain***
- Each tuple is unique and is identified by a ***key***

StudentID	FamilyName	Degree	Major	GPA
12345678	WELLS	BSc	ISD	3.00
12456789	NORBERT	BSc	CS	2.70
23456789	KENDALL	BSc	GT	3.50

RM concepts

EmpNo	FamilyName	GivenName	DeptNo
12345678	Smith	John	5
23456789	Wong	Franklin	2
34567890	Zelaya	Alicia	
45678901	Wallace	Jennifer	2

- Relations are linked through matching values of *primary keys and foreign keys*

- This is what gives the relational model great flexibility and the ability to ask ad-hoc queries

DeptNo	DeptName
1	Research
2	Admin
3	HQ
5	Youth

Properties of a relation

A valid relation has the following properties:

- A name that is distinct from all other relations
- Each **attribute** in a relation has a distinct name
- All cell values are **atomic** (multi-valued attributes are not allowed) – i.e., each row/column intersection represents a single data value
- Values in attributes are from the same *domain*
  - The **attribute domain** is the set of all possible values it may take. Definition covers both physical (data type) and logical (semantic) values
- All **tuples** must be **unique** – i.e., there must be an attribute or set of attributes that uniquely identifies each row
- **Attributes** are **not ordered**
- **Tuples** are **not ordered**

Example relation

StudentID	FamilyName	Degree	Major	GPA
12345678	WELLS	BSc	BIS	3.00
12456789	NORBERT	BSc	CS	2.70
23456789	KENDALL	BSc	GT	3.50

Note the following features of the **Student** relation above:

1. Each *tuple* is unique
2. Each *tuple* is about ONE student
3. Each *attribute* contains data from the same *domain*
4. Each *attribute* has a unique name
5. Each *cell* is *atomic*

Is this a valid relation?

StudentID	FamilyName	Degree	Major	GPA
12345678	WELLS	BSc	BIS	3.0
			CS	3.5
12456789	NORBERT	BSc	CS GT	D
23456789	KENDALL	BSc	CFIS	3.5
23456789	KENDALL	BSc	CFIS	3.5

Why? Why not?

Relation schema

A relation can be *described* by its name and attributes

- this is called the **relation schema**

e.g. STUDENT (<u>StudentNo</u>, StudentName, Email, **Course**)

We can also show the *values* of each tuple in a populated relation

- this is usually shown in table form:

StudentNo	StudentName	Email	Course
12345678	WELLS	wells@murdoch...	B1317
12456789	NORBERT	norbert@yahoo...	B1317
23456789	KENDALL	kendall@gmail …	B1317

How do we write relation schemas?

There is no real standard, but the following convention is often used:

- The relation name is uppercase (STUDENT)
- Attributes are initial uppercase (StudentName)
- Primary keys are underlined (StudentNo)
- Foreign keys are bolded (**Course**) or sometimes italic (*Course*)
- The schema is written with the relation name followed by the attributes in brackets:

STUDENT (StudentNo, StudentName, Email, **Course**)

The takeaways…

- The relational database model consists of *relations* of *attributes* and *tuples* (=tables of columns and rows)
- A valid relation has a number of properties:
  - Relation names are unique, and attribute names are unique within a relation
  - Tuples are unique
  - Attribute values are from the same domain
  - Each cell is single valued (atomic)
  - The order of the rows and columns is unimportant

Topic 02: Part 02

Keys

Keys

- Recall that each *tuple* in a relation must be unique for it to be a valid relation
- Therefore, there must be *an attribute or set of attributes that is unique* and so can be used to identify each tuple
- This attribute or set of attributes is called a **key**
- There are several types of key…
  - Superkey
  - Candidate Key
  - Primary Key
  - Alternate Key
  - Foreign Key

Superkeys

A superkey is …

- …any attribute or combination of attributes containing unique values for each tuple.
    - The combination of attributes containing ALL attributes in a relation is always a superkey

- Consider the Student relation:
    - STUDENT (StudentNumber, FamilyName, Degree, Major, GPA)

• *What are the superkeys of the STUDENT relation?*

Candidate Keys

A candidate key is...

- A **minimal** superkey
  - A superkey is minimal if the removal of an attribute results in the loss of uniqueness

- Consider the relation:
  - MOTOR VEHICLE (EngineNo, RegistrationNo, Colour, Model)

- *List all the superkeys of the relation.*
- *Which of those superkeys are also **candidate keys**?*

Candidate Keys

- MOTOR VEHICLE (EngineNo, RegistrationNo, Colour, Model)
- *Assuming both EngineNo and RegistrationNo are unique,*
- *the superkeys are:*

- E, R, C, M
- E, R, M
- R, C, M
- E, C, M
- E, R, C
- R, M
- E, M
- E, R
- R, C
- E, C
- R
- E

And EngineNo and RegistrationNo are both candidate keys of the relation

Primary Keys

A Primary Key is

- *The candidate key that is chosen to be the key for the relation*
- A relation can only have **one** primary key
- The value of the primary key:
    - MUST be UNIQUE
    - MUST NOT be NULL
- If a primary key is made up of > 1 attribute, it is known as a **compound, composite** or **concatenated** primary key
    - TUTORIAL (<u>TutorialDay, TutorialStartTime</u>, TutorName)

Question

What might be the primary keys of the following relations?

- STUDENT
- LECTURE
- AIRLINE TICKET
- BOOK

How do we find keys?

- Look at the data set - if you know that it is representative
- Formally – from the *functional dependencies* among the data
  - We will look more at this in Topic 4, Normalisation
- In practice – from the meaning of the data in the real world
  - e.g. your student number is designed to be a unique identifier
  - Phone numbers and email addresses must be unique to be useful

Alternate Keys

An Alternate key is:

- Simply, any candidate key that is not chosen as the primary key of the relation

Examples???

Foreign Keys

A Foreign Key is:

- An attribute in one relation that is used to reference the primary key in another relation
- This allows us to determine which records are *related*

STUDENT

StudentNo	LastName
20123456	Wells
20987654	Kendall
20876567	Norbert

UNIT

UnitCode	UnitName
ICT285	Databases
ICT292	IS Management
ICT301	Enterprise Architectures

ENROLMENT

StudentNo	**UnitCode**
20123456	ICT285
20123456	ICT292
20876567	ICT301
20876567	ICT285
20987654	ICT285

Foreign keys

- A well designed relational database will be able to link all its tables through primary keys and foreign keys in a way that represents the meaning in the data
- This gives us great flexibility in formulating queries to retrieve combinations of information
- We'll look at this formally when we cover **normalisation**, but for now, notice how primary keys and foreign keys are used in the example tables you are provided with

Other keys that are used in practice

Secondary Key:

- An attribute or set of attributes used for data retrieval purposes NOT required to be unique
- (however, sometimes you see 'secondary' used to mean the same as alternate, which gets confusing)

•Surrogate Key:

- An artificial primary key created to simplify retrieval
- e.g. if you have a very long concatenated candidate key
- Only used for implementation, usually created automatically by the DBMS

The takeaways…

- **Keys** are an important concept in the relational model
- Keys provide the ability to identify and locate individual tuples and relationships among tuples
  - Superkey, candidate key, **primary key**, alternate key identify a tuple uniquely
  - **Foreign keys** are used to define relationships
  - Secondary keys are non-unique and used for data retrieval
  - Surrogate keys are substitutes for primary keys used for convenience of implementation

Topic 02: Part 03

Integrity Constraints

Data integrity constraints

- Data integrity means that the data held in the database must ***make sense:***
- In other words, it is *consistent* and reflects the real world *correctly*
- We ensure data integrity by enforcing **constraints** on the data
- In RM we are primarily concerned with the following constraints:
  - Domain Constraints
  - Entity Integrity Constraint
  - Referential Integrity Constraint
  - Enterprise Constraints

Domain constraint

- The **domain constraint** applies to the *values of an attribute*
- It specifies that:
  - Each attribute within a relation must be from a single domain
  - The domain of an attribute limits its data to particular set of *allowable values*
  - Domains are more than just data type, as they indicate the *meaning* of the data
    - GPA must be a numeric value between 0-4
    - Final grade must be one of {HD, D, C, P, N}

Entity Integrity constraint

The **Entity Integrity** constraint applies to a SINGLE relation

It specifies that:

- The **primary key** value cannot be **NULL**
  - The primary key value is used to identify individual tuples in a relation. If the value is NULL, we cannot identify some tuples
- The **primary key** value must be **UNIQUE**
  - By definition, each tuple in a relation must be unique. If a tuple is not unique, then tuples cannot be individually identified

Referential Integrity constraint

The **Referential Integrity** constraint applies to a TWO relations

It specifies that:

- If a **foreign key** exists in a relation, its value must either refer to an existing record in the relation it references (i.e. it must match a **primary key** value in that relation),
  or be wholly null

•Referential integrity maintains consistency between the information in different relations

Referential integrity: example

Is referential integrity violated here? Why or why not?

<u>EmpNo</u>	FamilyName	GivenName	**DeptNo**
12345678	Smith	John	5
23456789	Wong	Franklin	2
34567890	Zelaya	Alicia	3
45678901	Wallace	Jennifer	2

<u>DeptNo</u>	DeptName
1	Research
2	Admin
3	HQ
5	Youth

Referential integrity: example

Is referential integrity violated here? Why or why not?

<u>EmpNo</u>	FamilyName	GivenName	**DeptNo**
12345678	Smith	John	5
23456789	Wong	Franklin	2
34567890	Zelaya	Alicia	
45678901	Wallace	Jennifer	2

<u>DeptNo</u>	DeptName
1	Research
2	Admin
3	HQ
5	Youth

Referential integrity: example

Is referential integrity violated here? Why or why not?

EmpNo	FamilyName	GivenName	**DeptNo**
12345678	Smith	John	5
23456789	Wong	Franklin	2
34567890	Zelaya	Alicia	4
45678901	Wallace	Jennifer	2

DeptNo	DeptName
1	Research
2	Admin
3	HQ
5	Youth

Example:
Are entity integrity and referential integrity constraints met here?

Table name: CUSTOMER
Primary key: CUS_CODE
Foreign key: AGENT_CODE

Database name: CH2_INSURE_CO

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_RENEW_DATE	AGENT_CODE
10010	Ramas	Alfred	A	Friday, March 12, 1999	502
10011	Dunne	Leona	K	Tuesday, May 23, 2000	501
10012	Smith	Kathy	W	Tuesday, January 05, 1999	502
10013	Olowski	Paul	F	Monday, September 20, 1999	
10014	Orlando	Myron		Monday, December 04, 2000	501
10015	O'Brian	Amy	B	Tuesday, August 29, 2000	503
10016	Brown	James	G	Wednesday, March 01, 2000	502
10017	Williams	George		Friday, June 23, 2000	503
10018	Farriss	Anne	G	Tuesday, November 09, 1999	501
10019	Smith	Olette	K	Friday, February 18, 2000	503

Table name: AGENT
Primary key: AGENT_CODE
Foreign key: none

AGENT_CODE	AGENT_AREACODE	AGENT_PHONE	AGENT_LNAME	AGENT_YTD_SLS
501	713	228-1249	Alby	$1,735.00
502	615	882-1244	Hahn	$4,967.00
503	615	123-5589	Okon	$3,093.00

FIGURE 2.4 AN ILLUSTRATION OF INTEGRITY RULES

Figure 2.4 in Rob, P. & *systems: design, implementation and management.* 4th Ed. Thomson Learning. p.69

Enterprise Constraints

Enterprise constraints or **business rules**

- These are additional constraints that apply to the particular system being modelled
- They are specified by the users of the system, rather than the requirements of the relational model
    - A student must have passed the prerequisite for a unit before enrolling in it
    - A student must have passed 18 points at Part 1 before enrolling in a Part 2 unit
- Also known as **general constraints**

The takeaways…

The relational database model has a number of *constraints* that keep the data correct and consistent:

- The **domain constraint** states that the value of a particular attribute always comes from the same (specified) domain
- The **entity integrity constraint** states that the value of the primary key must be unique and not null
- The **referential integrity constraint** states that the value of a foreign key must match an existing primary key, or be null
- **Enterprise constraints** specify constraints relating to business rules that must hold true, sometimes across multiple attributes or relations

Topic 02: Part 04

Relational Algebra

Operations on the relational model

As part of its definition, the relational model includes a set of **operations** that define the way in which relations can be manipulated

These operations can be expressed in two logically equivalent ways:

- The **relational calculus** (non-procedural)
- The **relational algebra** (procedural)

We will look at the relational algebra because:

- It’s simpler to understand
- It’s more useful when we come to examine query optimisation

Relational Algebra

Codd defined the relational algebra as part of the relational model. It:

- is a **theoretical** language: there are no commercial implementations of the relational algebra
- assists with understanding the basic **operations** that can be performed on a relational database
- Is a **procedural** language – you need to specify the order in which the operations are carried out
- always transforms one or two relations into a new relation (**closure** property)
- is used in DBMSs for internal representation of query plans for optimisation

Relational Algebra operators

The relational algebra operators can be classified into:

- Relation specific operators
    - RESTRICT, PROJECT, CARTESIAN PRODUCT, JOIN, DIVISION
- Traditional set operators
    - UNION, INTERSECT, MINUS (DIFFERENCE)
- Extended operators

Relation Specific Operators

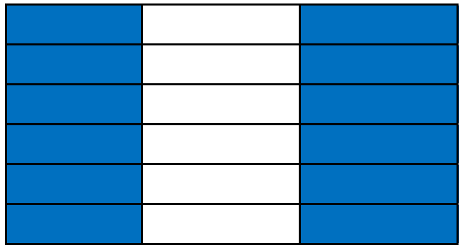
- RESTRICT σ
- PROJECT Π
- CARTESIAN PRODUCT X
- JOIN *
  - (various flavours)
- DIVISION ÷

Operations on a single relation: Restrict and Project

- Restrict and Project are similar in that the result of operations using them is a ***subset*** of the original relation

Restrict

Project



The Restrict Operator

Restrict σ

- Operates on **one** relation
- Produces a subset of the *tuples* of a relation
- Uses a condition or logical expression to restrict the tuples in the result relation
- The resulting relation has the same attributes as the original relation
- Referred to in some texts as *Select*

EMPLOYEE (E#, Name, Age, Salary)

'Restrict to Employees whose age is less than 30'

$$\sigma_{\text{Age} < 30} \text{ (EMPLOYEE)}$$

restriction condition *relation name*

Original Relation			
E #	Name	Age	Salary
1	Smith	20	1000
2	Jones	35	3000
3	Tan	25	2500

Result Relation			
E#	Name	Age	Salary
1	Smith	20	1000
3	Tan	25	2500

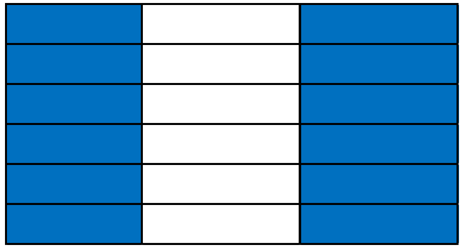
The restriction condition may be any Boolean expression

e.g. $\sigma_{\text{Age<30 and Salary<=1000}} \text{ (EMPLOYEE)}$

The Project Operator

Project Π

- Operates on **one** relation
- Used to select a subset of **attributes** of a relation
- The result of a project is a relation with only the attributes specified, and any duplicate tuples removed.
  - (WHY are the duplicates removed?)



'List the names and salaries of all employees'

$$\Pi_{\text{Name, Salary}}\ (\text{EMPLOYEE})$$

attribute list *relation name*

Original Relation

E#	Name	Age	Salary
1	Smith	20	1000
2	Jones	35	3000
3	Smith	25	1000

Result Relation

Name	Salary
Smith	1000
Jones	3000

Notice that duplicates can arise in the result if non-key attributes only are projected

Sequences of operations

Since the result of a relational algebra operation is another relation, we can apply several operations in *sequence*

To represent this we can either:

- Write the operations as a **single expression** (sequence implied by brackets), OR
- Create, temporary, **intermediate relations** (you will need to name them appropriately)

Both are correct, so use whichever you find easiest

Example: single expression v intermediate relations

EMPLOYEE (E#, Name, Salary, Dept)

'Give the name and salary of all employees who work in department 5'

$$\Pi_{\text{Name, Salary}} (\sigma_{\text{Dept = 5}} (\text{EMPLOYEE}))$$

or:

$$\sigma_{\text{Dept = 5}} (\text{EMPLOYEE}) \rightarrow \text{Temp}$$

$$\Pi_{\text{Name, Salary}} (\text{Temp}) \rightarrow \text{Result}$$

Ensuring a sequence of operations is correct

- Because relational algebra is *procedural*, you need to make sure that each step in the sequence preserves attributes/tuples that will be needed in a later operation

EMPLOYEE (E#, Name, Salary, Dept)

'Give the name and salary of all employees who work in department 5'

$$\sigma_{\text{Dept} = 5} (\Pi_{\text{Name, Salary}} (\text{EMPLOYEE}))$$

Why won't this work??

Efficiency of a sequence of operations

- Some sequences will be more **efficient** than others in terms of the number of tuples returned by the each of the operations
- Finding an efficient sequence is the basis of **query optimisation** in the DBMS – the query optimiser takes the input query and works out the best way to solve it
- Efficiency is especially relevant when there are many relations to be **joined**
- However, when you are doing the relational algebra exercises for this topic you don't need to bother about creating an efficient query – just a correct one!

Cartesian Product operator

- **Cartesian Product X**
  - Applies to two relations
  - R1 X R2
  - Result is a relation with the combined attributes of the two relations and records consisting of *all possible combinations* of tuples from the two relations

Cartesian Product Example

E#	EName	D#
111	Fred	1
222	Jane	2
555	Ann	1

X

D#	DName
1	Admin
2	Research

E#	EName	EMP.D#	DEPT.D#	DName
111	***Fred***	***1***	***1***	***Admin***
111	Fred	1	2	Research
222	Jane	2	1	Admin
222	***Jane***	***2***	***2***	***Research***
555	***Ann***	***1***	***1***	***Admin***
555	Ann	1	2	Research

- Why is this result not very meaningful?
- What would we need to do next??

Natural Join Operator

R $*_{\textit{join condition}}$ **S**

- An operation on **two** relations, equivalent to a product followed by a restrict
- Usually, the select is on equality of related records from the two relations – equijoin or natural join
- Thus the join operation allows us to process relationships between relations, by *joining on primary key and foreign key*
  - $EMP *_{EMP.D\# = DEPT.D\#} DEPT$

- Note this is the same as:
  - EMP X DEPT → TEMP
  - $\sigma_{Emp.D\#=Dept.D\#}$ (TEMP) →Result

EMP* $_{EMP.D\# = DEPT.D\#}$ DEPT

EMP

E#	EName	D#
111	Fred	1
222	Jane	2
555	Ann	1

*$_{EMP.D\#=DEPT.D\#}$

DEPT

D#	DName
1	Admin
2	Research

E#	EName	EMP.D#	DEPT.D#	DName
111	Fred	1	Admin 1	Admin
222	Jane	2	Research 2	Research
555	Ann	1	Admin 1	Admin

Note how only the tuples from **both** relations that have the same D# are in the result relation.

The repeated attribute is removed from the result relation

Types of joins

- The **natural join *** (on primary and foreign key, eliminating duplicates) is usually the most useful
- Natural join is sometimes written ⋈

There is also:

⋈ Equijoin – joins on equality of attributes, doesn't eliminate duplicate common attribute from result

θ Theta join – joins on any comparison operator

--- you don't need to worry about equi- or theta joins here

Outer join – next slides

Outer Join Operator

A variation on the natural join:

- Natural join preserves *matching* tuples
- **Outer join** also preserves ***non-matching tuples***
  - Any missing values in the second relation are set to *null*
  - Useful in examples such as:
    - All units and the name of the Unit Coordinator, including units that do not have a Unit Coordinator
    - All students and the tutorials they are enrolled in, if any
- Outer joins can be:
  - **Full**
  - One-sided (**left outer join**, **right outer join**)

e.g. 'All employees, and the name of their next of kin, if they have one'

EMP **LEFT OUTER JOIN**$_{\mathbf{EMP.E\#=NOK.E\#}}$NOK

E#	EName	D#
111	Fred	1
222	Jane	2
555	Ann	1

NOK#	Name	E#
11	Liz	111
33	John	222

- This shows a left outer join – the tuples that fulfil the join condition are added to the result

E#	EName	D#	NOK#	Name
111	Fred	1	11	Liz
222	Jane	2	33	John

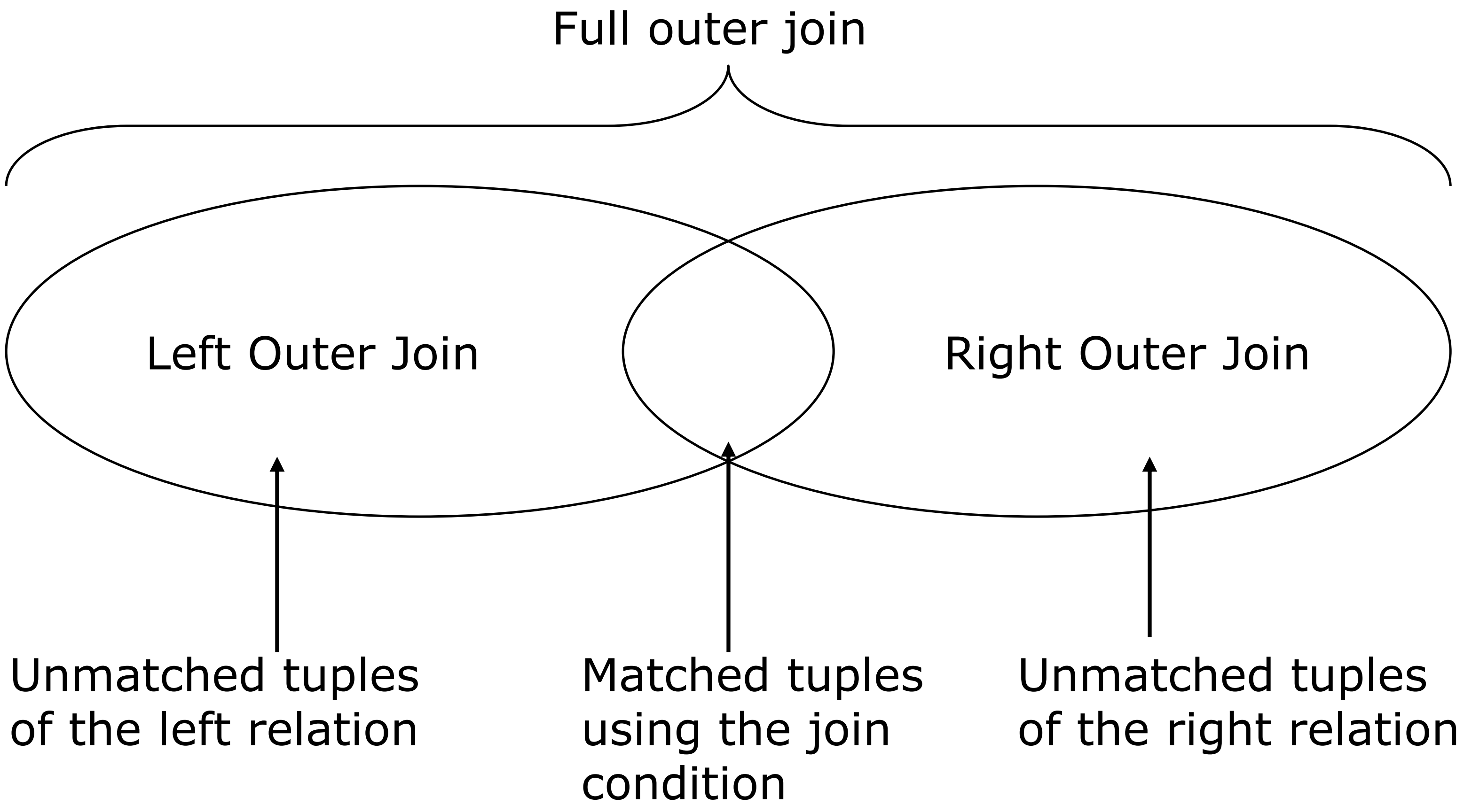
- The tuples from the left hand relation that do not fulfil the join condition are added to the result

E#	EName	D#	NOK#	Name
111	Fred	1	11	Liz
222	Jane	2	33	John
555	Ann	1		

- The "blank" values are padded with null

E#	EName	D#	NOK#	Name
111	Fred	1	11	Liz
222	Jane	2	33	John
555	Ann	1	**null**	**null**

Outer Join Operators



Full Outer Join

Faculty

FacSSN	FacName
111-11-1111	joe
222-22-2222	sue
333-33-3333	sara

Offering

Offerno	FacSSN
1111	111-11-1111
2222	222-22-2222
3333	111-11-1111
4444	

Outer Join of Offering and Faculty

FacSSN	FacName	OfferNo
111-11-1111	joe	1111
222-22-2222	sue	2222
111-11-1111	joe	3333
333-33-3333	sara	
		4444

The Division Operator

Division ÷

Match on a subset of values

- Suppliers who supply all parts
- Lecturers who teach every CS unit

Formally,

- A relation R with **two** attributes is divided by a relation S with **one** attribute, where S is a subset of R
- The result is a relation consisting of the attribute which was **not** in S
- Each record that appears in the result appears in R in combination with **every** tuple in S

‘Find the suppliers who supply ALL parts’

SuppPart

SuppNo	PartNo
S2	P3
S3	P1
S1	P1
S2	P1
S1	P3
S2	P2

Part

PartNo
P1
P2
P3

SuppPart DIVIDEBY Part

SuppNo:

‘Find the suppliers who supply ALL parts’

SuppPart

SuppNo	PartNo
S2	P3
S3	P1
S1	P1
S2	P1
S1	P3
S2	P2

Sort SuppPart by SuppNo

Part

PartNo
P1
P2
P3

SuppPart DIVIDEBY Part

SuppNo:

‘Find the suppliers who supply ALL parts’

SuppPart	
SuppNo	**PartNo**
S1	P1
S1	P3
S2	P1
S2	P2
S2	P3
S3	P1

S1 {P1, P3} does not contain {P1, P2, P3}, so is not included in the result

Part
PartNo
P1
P2
P3

SuppPart DIVIDEBY Part

SuppNo:

‘Find the suppliers who supply ALL parts’

SuppPart	
SuppNo	**PartNo**
S1	P1
S1	P3
S2	P1
S2	P2
S2	P3
S3	P1

S2 {P1, P2, P3} does contain {P1, P2, P3}, so S2 is included in the result

Part
PartNo
P1
P2
P3

SuppPart DIVIDEBY Part SuppNo:
{S2}

'Find the suppliers who supply ALL parts'

SuppPart	
SuppNo	**PartNo**
S1	P1
S1	P3
S2	P1
S2	P2
S2	P3
S3	P1

S3 {P1} does NOT contain {P1, P2, P3}, so is not included in the result

Part
PartNo
P1
P2
P3

SuppPart DIVIDEBY Part

SuppNo:

{S2}

Another Division…

'Find the employees who work on **all** projects'

Result?

E#	Project
1	ProductX
2	ProductY
3	ProductY
1	ProductY
1	ProductZ
2	ProductZ
3	ProductX

Project
ProductX
ProductY
ProductZ

Another Division…

'Find the employees who work on **all** projects'

Result

E#	Project
1	ProductX
2	ProductY
3	ProductY
1	ProductY
1	ProductZ
2	ProductZ
3	ProductX

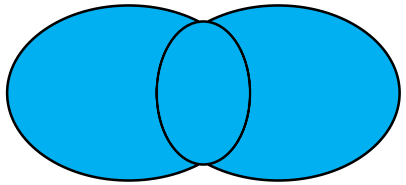
Project
ProductX
ProductY
ProductZ

E#
1

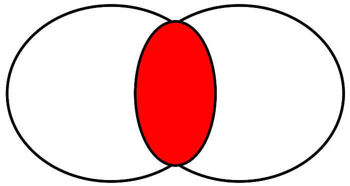
Set operators

- UNION
- INTERSECTION
- DIFFERENCE (MINUS)

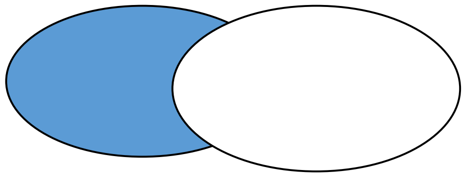
Set operators



A UNION B



A INTERSECT B



A MINUS B

(assume A is the set on the left hand side)

Union Compatibility

Unlike the relational algebra operators that compare on the join condition, the traditional set operators compare on the *whole relation*

To do this, we need UNION COMPATIBILITY

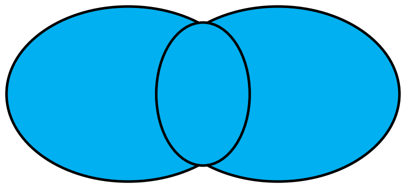
- Same number of attributes
- Each corresponding pair of attributes is compatible (*Positional correspondence*)

Often have to PROJECT the correct attributes first in order to get union compatible relations

Union

Union R U S

- Produces a relation that includes all the tuples **in R or S or both**
  - Duplicates are eliminated
  - By convention, the attributes in the result have the same names as those in the first relation



Union Example

'List the employees who work on Project X or Project Y, or both'

Proj_X

LastName
Smith
Jones
Tan

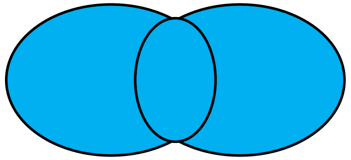
∪

Proj_Y

LName
Jones
Lee

→

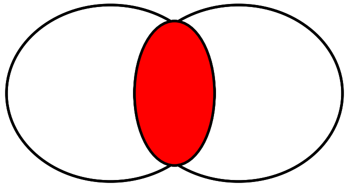
LastName
Smith
Jones
Tan
Lee



Intersection

Intersection **$R \cap S$**

- Produces a relation that includes all the tuples in **both R and S**



Intersection Example

'List the employees who work on both Project X **and** Project Y'

Proj_X

LastName
Smith
Jones
Tan

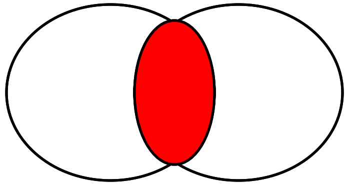
$\cap$

Proj_Y

LName
Jones
Lee

$\rightarrow$

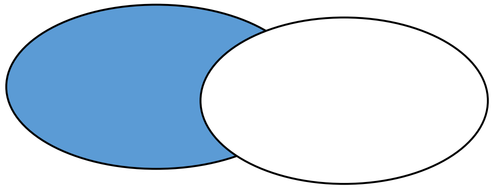
LastName
Jones



Difference (or Minus)

Difference **R – S**

Produces a relation that includes all the tuples that are **in R but not in S**



Difference Example

'List the employees who work on Project X **but not** Project Y'

Proj_X

LastName
Smith
Jones
Tan

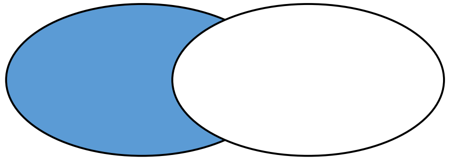
\-

Proj_Y

LName
Jones
Lee

→

LastName
Smith
Tan



Extended Operators

- Various authors have introduced extensions to the original relational algebra
- These are mainly aimed providing some computational capacity such as simple statistical functionality, similar to that found in SQL
  - E.g., Date (2005) includes the “Extend” and “Summarize” operators
- We won’t cover extended operators any further, but note that you may encounter them in other texts

The takeaways…

- The relational algebra provides the operators that can be used to query a set of relations
- The result of a relational algebra operation is another relation, so queries are constructed by applying one operation at a time, procedurally
- The **restrict** and **project** operators apply to a single relation
- The **join** operators enable related relations to be combined
- **Natural join** joins on primary key and foreign key
- Outer joins preserve non-matching tuples as well
- The **division** operator matches on a subset of values
- **Union**, **intersection** and **difference** enable set operations on union compatible relations

Topic 02: Part 05

Conclusion and RA Examples

Learning Outcomes

- **After completing this topic you should be able to:**
- Describe the characteristics of the relational database model
- Define and give examples of the different types of keys used in the relational database model
- Explain and give examples of the relational model's integrity constraints
- Use the fundamental operators of the relational algebra (restrict, project, Cartesian product, join, intersection, difference, union, and division) to define simple queries

Some examples …

- Have a go at these and ask your tutor if you have problems. Solutions will be posted on LMS

Example 1

CUSTOMER (CustomerNumber, CustomerName, DateOfBirth)

EMPLOYEE (EmployeeNumber, EmployeeName, DateOfBirth)

The following relational algebra query is incorrect:

π CustomerName, DateOfBirth (CUSTOMER)

UNION

π DateOfBirth, EmployeeName (EMPLOYEE)

Why is the relational algebra statement above incorrect?

Rewrite the statement to correct the error.

Example 2

Consider the following relations from a database that keeps track of business trips made by salespeople (SPN = Salesperson Number)

SALESPERSON (SPN, Name, StartYear, DeptNo)

- TRIP (TripID, ToCity, DepartDate, ReturnDate, **SPN**)
- EXPENSE (**TripID**, Account#, Amount)

1. Give all details of the Salesperson named ‘Bob’
2. Give the SPN and Name of salespeople who took trips to the city Sydney
3. Give the trip ID and destination city of all trips taken by the salesperson named ‘Dodgy’
4. Give the names of salespeople who have **not** travelled to Sydney

Example 3

Consider the following relations for a database that keeps track of student enrolment in units and the books adopted for each unit:

STUDENT (<u>StudID</u>, Name, Major, DoB)

UNIT (<u>UnitCode</u>, UnitName, School)

- ENROL (**<u>StudID</u>**, **<u>UnitCode</u>**, <u>Offering</u>, Grade)
- BOOKLIST (**<u>UnitCode</u>**, <u>Offering</u>, **ISBN**)

TEXT (<u>ISBN</u>, Title, Publisher, MainAuthor)

1. List the unit code of the units taken by the student with the ID “1234”
2. List the names of the units taken by all students named ‘John Smith’ in the offering Semester 2, 2014
3. Produce a list of the titles of the textbooks for units offered by the School of Information Technology
4. List the StudID of any students who are enrolled in ALL units offered by the Dodgy School of Business

Example 4

In terms of the following relations:

PROPERTY (PropertyNo, Address, NumberOfRooms, **OwnerNo**)
OWNER (OwnerNo, FamilyName, Given Names, Address)

Formulate the following relational algebra queries:

1. List the family and given names of the owners who own properties with more than three rooms.

2. List the family and given names of any owners that do not own a property.