CT230 DATABASE SYSTEMS

Introductions

Semester 1

2022

WELCOME!

Fáilte roimh go léir!

Module Code:
CT230
Module Name:
Database Systems

LECTURE TODAY ... INTRODUCTIONS

- Me
- You
- CT230:
  - Learning outcomes and course outline
  - Systems and tools we will use
  - Some information on how lectures, labs, assessment and exam will work this year
  - Introduction to Database Systems

ME!

Dr Josephine Griffith **(She/Her)**

Room 405 Computer Science Building (formerly IT building)

School of Computer Science

College of Science & Engineering

Josephine.Griffith@universityofgalway.ie

YOU … 140 students taking this class at the latest count !

- 2BCT
- 2BA, 2BDA, 2BDS, 2BFD, 2BFS, 2BGM, and possibly others once registration is finished
- 3CSM
- 3BP, 3BLE
- Erasmus Visiting Students: 1EM

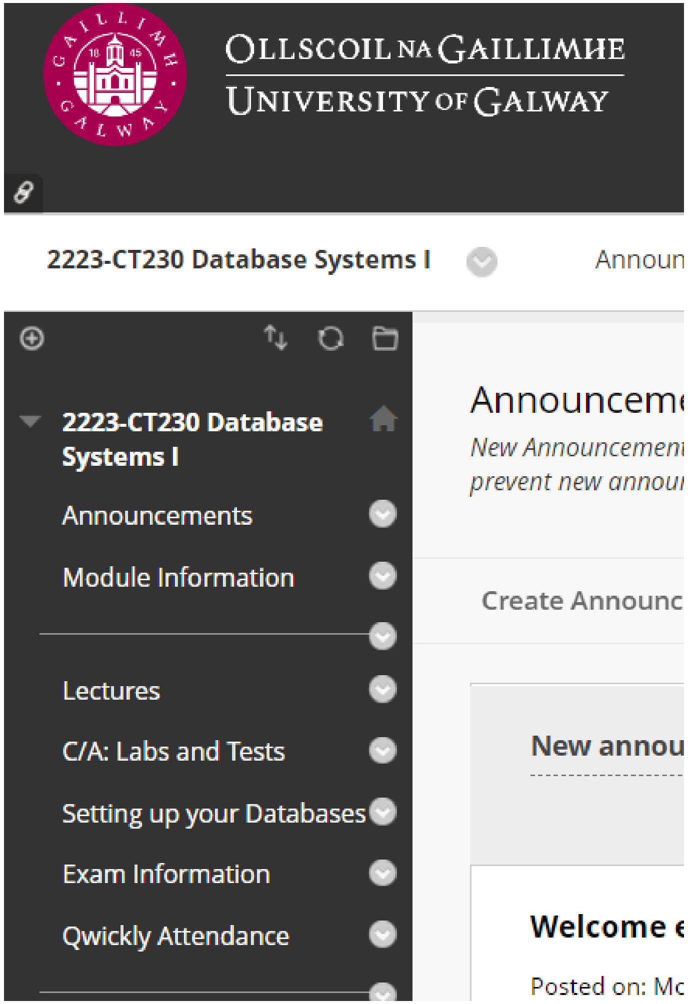
You all belong here!

HOW ARE "WE" GOING TO DO THIS ....

- We are required to exclusively deliver on-campus lectures and on-campus labs
- Attendance at lectures will be logged – though won't be correct until probably week 3 when registration is hopefully finalised - please only sign in if you really are here!
- I generally provide summary videos of core concepts as a study/revision aid.

BLACKBOARD "QWICKLY" SIGN IN …

- Need to be registered and on Blackboard to sign in
- I know not everyone will be registered fully today – that's ok.
- Go to Blackboard and CT230 and click on "Quickly Attendance" in Content Area
- I will give code …



USING MENTI FOR QUESTIONS

We will use menti.com to ask questions (2-way!).

I will try take a regular break from lectures to try answer some of the questions but all valid questions will be answered eventually.

Note to treat each other (and me!) with respect and be careful of our tone. We have a diverse group – which is fantastic! – let's make sure everyone feels they belong and that we don't waste anyone's time.

FOR NON-PUBLIC QUESTIONS

I will try finish 5 minutes early for the first few weeks so I will have extra time for questions.

As we have to vacate the lecture theatre in time for the next class, I can wait outside the lecture theatre (downstairs) if we run out of time.

Email is always an option:

josephine.Griffith@universityofgalway.ie

You can arrange to come talk to me in person by setting up an appointment.

I will also attend some labs and you can ask me questions there.

LET'S NOW START TALKING ABOUT THE MODULE ...

CT230 DATABASE SYSTEMS

LEARNING OUTCOMES
– A folder for each of these on Blackboard
– A number of lectures associated with each

LO1	Define and explain terms, concepts, properties and constraints of Relational Database Systems
LO2	Identify the theoretical and practical issues in the storage, manipulation, organisation and indexing of large quantities of data
LO3	Program a database management system for database creation and manipulation
LO4	Use Relational Algebra for relational database retrieval
LO5	Program using SQL for relational database retrieval and manipulation
LO6	Create and apply Entity Relationship Diagrams (ERD) as part of database development
LO7	Specify functional dependencies and differentiate between relations in 1st Normal Form, 2NF, 3NF. Apply the process of normalization
LO8	Define and explain the process of query processing and optimisation. Apply query optimisation heuristics to develop a query tree that represents an efficient evaluation strategy for a given query.

COURSE TOPICS

(not the order we will follow)

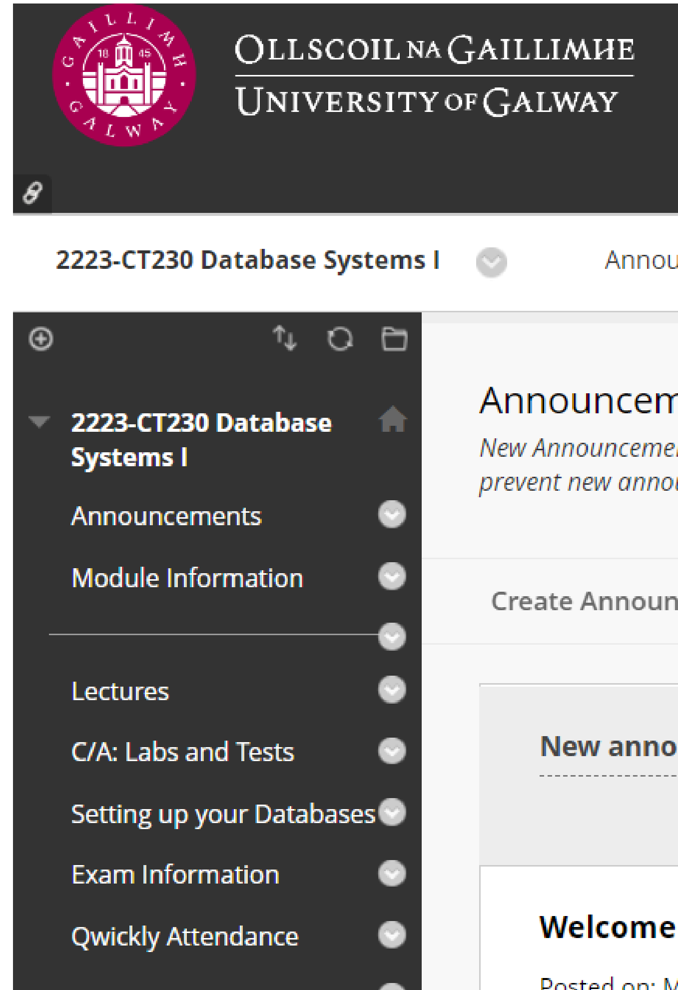
- LO1: Database fundamentals:
- Data, Information & Database Systems
- The Relational Model
- LO2: Database fundamentals: File Organisations
- L03: Database programming: SQL for database creation and manipulation (DDL)
- LO4: Database fundamentals: Relational Algebra
- L05: Database programming: SQL for database retrieval and manipulation (DML)
- LO6: Database design: Entity-Relationship Modelling
- LO7: Database design: Normalisation
- LO8: Database programming: Query Processing and Optimisation

Lectures

<u>Lecture materials will be posted before lectures on Blackboard</u>

In person lectures for 12 weeks in AM200 (Fottrell theatre)

- Tuesday 2-2.50pm
- Wednesday 12-12.50pm



LECTURE NOTES

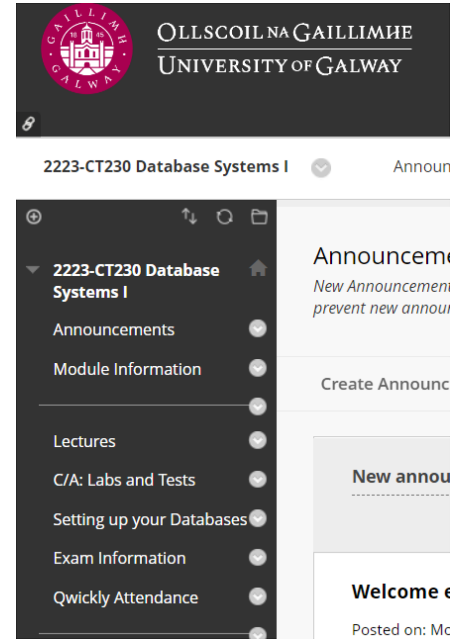
CT230 module on Blackboard will contain all the materials.

Note that there will be a mixture of notes, videos, worked examples, code, etc. for this module.

Ideally use a (paper or electronic) notebook for the course or have a good process for working on exercises on laptop.

Notepad++ is a good editor (rather than using MS word or equivalent)

C/A: Labs and Tests
Starting week 3 (19/09/2022)



Each lab session will have a lab tutor

3BP, 3BLE: Mon 4-6 *IT101*

2BCT: Tue 3-5 *IT101*

3CSM: Thur 10-12 *IT106*

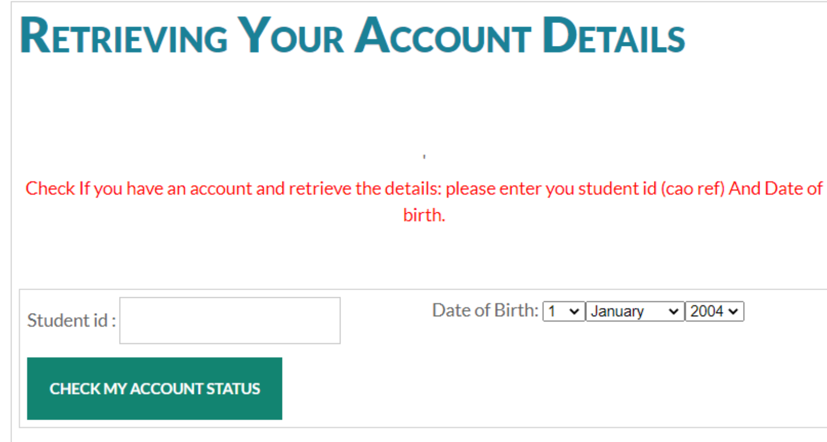
2BA and other BA programmes: Thur 10-2 *IT106* – *any 2 hours – to be organised*

Visiting Students: Pick any time that suits and let me know please *(by email)*

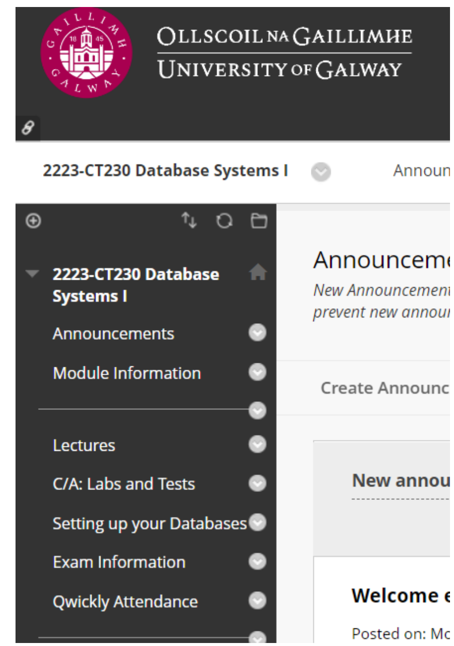
IMPORTANT RE LABS

- Physical Labs will not start until week 3.
- You can start getting prepared for labs based on our lectures.
- It is important that you have your CS account set-up and are able to connect to the CS server before labs begin.
- A schedule of labs and tests will be available soon on Blackboard and I will discuss in detail once available.

CS ACCOUNTS



- Everyone should have an ISS account which you use to access Blackboard, Library, Computer Suites, etc.
- For this module you also need a CS account to access the CS intranet to get your own mySQL database:
- http://www2.it.nuigalway.ie/accounts/
- All information will be in "Setting up your Databases"



ASSESSMENT INFORMATION

CT230 is assessed via a Semester 1 written exam and C/A throughout the semester.

The breakdown of the final mark is:

- 20% C/A
- 80% Exam

EXAM

- Examined in December 2022 (Exams office will generate exam timetable in November)
- Exam will be two hours duration* and will be in person (not online).
- Will discuss the format of the exam closer to the time - many past exam papers exist as examples.

* unless you have a LENS report

ATTENDANCE AT LECTURES, LABS, COMPLETION OF ASSIGNMENTS AND TESTS

In addition to gaining up to 20% of the final mark, some exam questions become much easier if you have completed the assignment work

Plagiarism is not acceptable and will result in a 0 mark

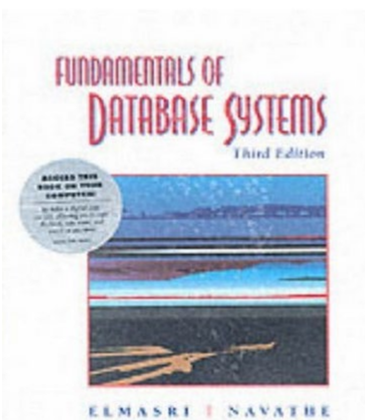
RECOMMENDED BOOK

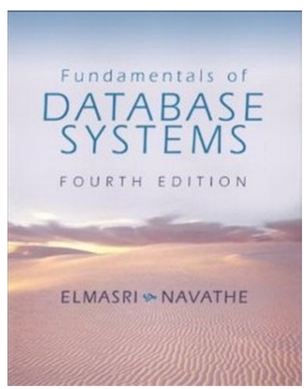
Fundamentals of database systems

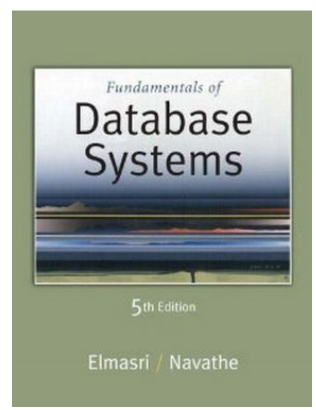
By Ramez Elmasri and Sham Navathe

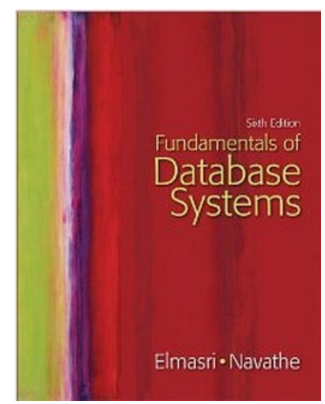
- Any edition is fine
- Editions 3, 5 and 6 available in library at

  Main Library Open Access (005.74 ELM)









SOME GENERAL COMMENTS ON THE YEAR

It starts now!

If you have problems, the sooner I (and in general "we") find out the more we can help.

Spend some time now:

- thinking of your priorities and time commitments and how you will manage these for the semester.
- Thinking of your triggers – how you will know if things are going well or not going well – and what you will do.
- Thinking about how to organise your notes and materials and lecture sessions and meet your deadlines.

TOMORROW'S LECTURE

- 12 noon here in AM200
- Topic: Introduction to Database Systems
- Have a great day!

TOPIC: THE RELATIONAL MODEL

CT230 Database Systems

Recall …
why learn about relational DBMS?

90% of industry/enterprise/business applications are STILL Relational DBMS or Relational DBMS with extensions (e.g. OO Relational).

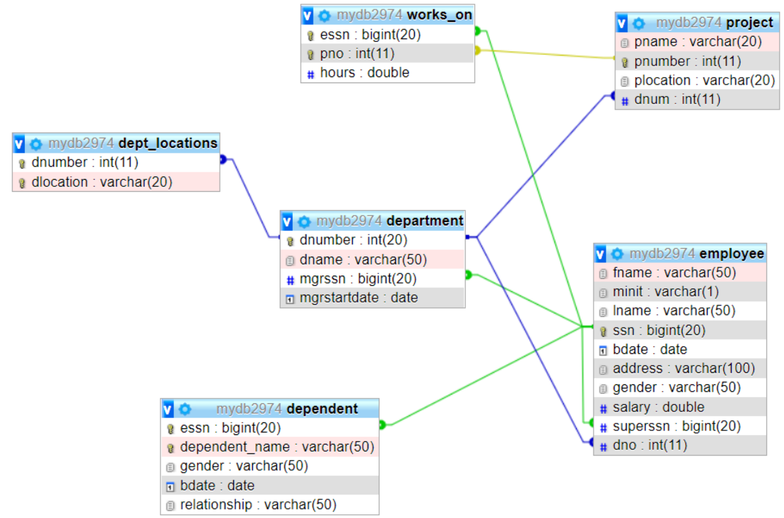
Majority of industry applications require:

- Correctness
- Completeness
- Efficiency (Complex optimisation techniques and complex Indexing structures).

Relational DBMS provide this.

OUR NOTATION

case **not** significant; spaces not allowed



DATABASE SCHEMA

employee(fname, minit, lname, ssn, bdate, address, gender, salary, superssn, dno)

department(dname, dnumber, mgrssn, mgrstartdate)

dept_locations(dnumber, dlocation)

project(pname, pnumber, plocation, dnum)

works_on(essn, pno, hours)

dependent(essn, dependent_name, gender, bdate, relationship)

SETTING UP YOUR DATABASE …

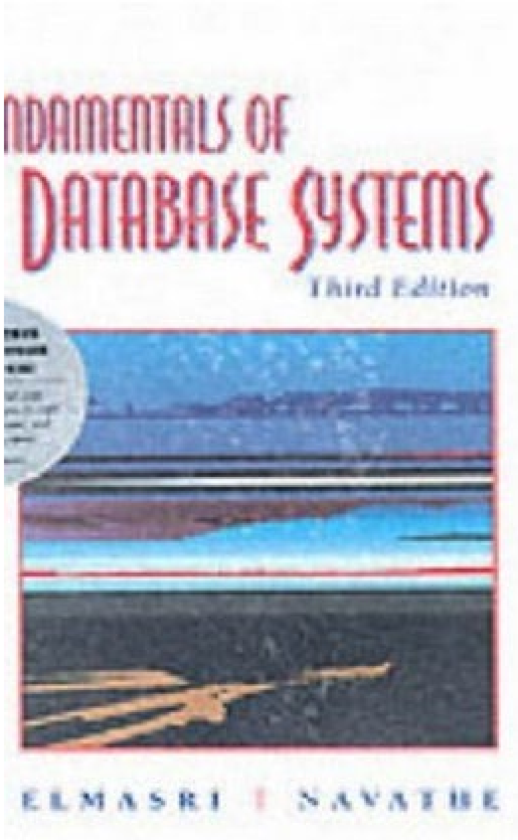
See supplemental notes and video will be added before labs next week ….

TOPIC:
Defining and working with the Relational Model

See

Elmasri and Navathe book

Chapter 7



RELATIONAL DATA MODEL

- Collection of relations (often called *tables*) where each relation contains tuples (rows) and attributes (columns).
- Closely related to file system model at (we use in our own programming)
- Relations are named: e.g., relation 'employee':

employee(fname, minit, lname, ssn, bdate, address, gender, salary, superssn, dno)

fname	minit	lname	ssn	bdate	address	gender	salary	superssn	dno
John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	5
Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	Woman	44183	333445555	5
Ramesh	K	Narayan	666884444	1995-09-15	975 Fire Oak, Humble, TX	Man	60000	333445555	5
James	E	Borg	888665555	1997-11-10	450 Stone, Houston, TX	Man	94199	*NULL*	1
Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	4
Ahmad	V	Jabbar	987987987	2000-03-29	980 Dallas, Houston, TX	Man	44183	987654321	4
Alicia	J	Zelaya	999887777	1998-07-19	3321 Castle, Spring, TX	Non-binary	44183	987654321	4

- Relation = table
- Attributes = columns and these are (mostly always) fixed (e.g., fname, minit, lname …) and do not change

* The number of attributes of a relation is referred to as its grade or degree

- Tuples = rows which contain the data and there is variable number of these

* The number of tuples of a relation is referred to as its cardinality.

ATTRIBUTES/COLUMNS

Each attribute belongs to **one** *domain* and has a single:

- name
- data type
- format

e.g.,

Column	Type
fname	varchar(50) *NULL*
minit	varchar(1) *NULL*
lname	varchar(50) *NULL*
ssn	bigint(20)
bdate	date *NULL*
address	varchar(100) *NULL*
gender	varchar(50) *NULL*
salary	double *NULL*
superssn	bigint(20) *NULL*
dno	int(11) *NULL*

`Name:     bDate`

`Type:     date`

`Format:   yyyy/mm/dd`

NAMING COLUMNS (ATTRIBUTES)

Column	Type
fname	varchar(50) *NULL*
minit	varchar(1) *NULL*
lname	varchar(50) *NULL*
ssn	bigint(20)
bdate	date *NULL*
address	varchar(100) *NULL*
gender	varchar(50) *NULL*
salary	double *NULL*
superssn	bigint(20) *NULL*
dno	int(11) *NULL*

- case **not** significant in SQL
- no spaces allowed
- no reserved keywords (e.g. date) allowed
- as usual, if picking names yourself - choose meaningful variable name
- if given the names of relations and attributes, use **exactly** what you are given

DATA TYPES

As with many programming languages must specify the data type of all attributes (columns) defined

Common data types used are:

- varchar(*N*), N an integer (for strings)
- date
- int
- double

Often specify the sizes especially for integers and strings

Will discuss in more detail when we start to create tables

Column	Type
fname	varchar(50) *NULL*
minit	varchar(1) *NULL*
lname	varchar(50) *NULL*
ssn	bigint(20)
bdate	date *NULL*
address	varchar(100) *NULL*
gender	varchar(50) *NULL*
salary	double *NULL*
superssn	bigint(20) *NULL*
dno	int(11) *NULL*

NULL

Null valued-attributes: values of some attribute within a particular tuple may be unknown or may not apply to a particular tuple … null value is used for these cases.

NULL is a special marker used in SQL to denote the absence of a value

Column	Type
fname	varchar(50) *NULL*
minit	varchar(1) *NULL*
lname	varchar(50) *NULL*
ssn	bigint(20)
bdate	date *NULL*
address	varchar(100) *NULL*
gender	varchar(50) *NULL*
salary	double *NULL*
superssn	bigint(20) *NULL*
dno	int(11) *NULL*

- In some cases we wish to allow the possibility of a NULL value although they will often require extra handling (e.g. checking for =NULL).
- In other cases we want to prevent NULL being entered as a value and specify NOT NULL as a constraint on data entry.

ATOMIC ATTRIBUTES

Column	Type
fname	varchar(50) *NULL*
minit	varchar(1) *NULL*
lname	varchar(50) *NULL*
ssn	bigint(20)
bdate	date *NULL*
address	varchar(100) *NULL*
gender	varchar(50) *NULL*
salary	double *NULL*
superssn	bigint(20) *NULL*
dno	int(11) *NULL*

An **atomic attribute** is an attribute which contains a single value of the appropriate type. Generally meaning, “no repeating values of the same type”

The relational model should **only** have atomic values

Example: Attribute address of type varchar(100) *Null*

Should only contain one address “3 Cherry Road, Carlow”

Rather than “3 Cherry Road, Carlow; Apt 12 Corrib Village, Galway”

COMPOSITE ATTRIBUTES

A **composite attribute** is an attribute that is composed of several more basic/atomic attributes.

Example:

- Name = FirstName, Middle Initial, Surname

Column	Type
fname	varchar(50) *NULL*
minit	varchar(1) *NULL*
lname	varchar(50) *NULL*
ssn	bigint(20)
bdate	date *NULL*
address	varchar(100) *NULL*
gender	varchar(50) *NULL*
salary	double *NULL*
superssn	bigint(20) *NULL*
dno	int(11) *NULL*

We often want to decompose a composite attribute into atomic attributes unless there is a very good reason not to (e.g. why is address not decomposed in to street, city, county, etc.?)

MULTI-VALUED ATTRIBUTES

A **multi-valued attribute** is an attribute which has lower and upper bounds on the number of values for an individual entry.

(the opposite of an atomic attribute)

Example:

qualifications

phone numbers

Column	Type
fname	varchar(50) *NULL*
minit	varchar(1) *NULL*
lname	varchar(50) *NULL*
ssn	bigint(20)
bdate	date *NULL*
address	varchar(100) *NULL*
gender	varchar(50) *NULL*
salary	double *NULL*
superssn	bigint(20) *NULL*
dno	int(11) *NULL*

The relational model should **NOT** store multi-valued attributes – database design/re-design should be used to deal with this issue by creating more attributes (columns) or more tables.

DERIVED ATTRIBUTES

A **derived attribute** is an attribute whose value can be determined from another attribute

Example:

from bdate can derive age

Column	Type
fname	varchar(50) *NULL*
minit	varchar(1) *NULL*
lname	varchar(50) *NULL*
ssn	bigint(20)
bdate	date *NULL*
address	varchar(100) *NULL*
gender	varchar(50) *NULL*
salary	double *NULL*
superssn	bigint(20) *NULL*
dno	int(11) *NULL*

It is a good idea to not directly store attributes which can be derived from other attributes.

RECALL ....

- We said that the Relational Data Model consists of a **collection** of relations (*tables*)
- Tables are **cross-linked**

COLLECTION OF RELATIONS

A relational database usually contains many relations (tables) rather than storing all data in one single relation.

A relational database schema, S, is a definition of a set of relations that are to be stored in the database, i.e.,

$S = \{R_1, R_2, \ldots, R_n\}$

e.g., S = {employee, department, works_on, dept_locations, project, dependent}

Formal definition of “schema”

A relational schema *R* is the definition of a table in the database. It can be denoted by listing the table name and the attributes:

$$R(A_1, A_2, \ldots, A_n)$$

where A_i is an attribute.

e.g. with n=3, that is, 3 attributes:

works_on(essn, pno, hours)

RECALL:
Database schemas and instances

Similar to types and variables in programming languages.

Schema: the logical structure of a database.

Instance: the actual content of the database at some point in time

LINKING TABLES …

Two VERY (*very, very*) important concepts within the relational model which allow tables to be linked and cross-referenced are:

- PRIMARY KEY attributes
- FOREIGN KEY attributes

We will define and discuss these tomorrow!

QUESTIONS?/ISSUES?

PRIMARY KEYS

Fundamental concept of Primary Keys:

All tuples (row) in a relation must be distinct

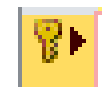
To ensure this must have:

- one of more attributes/columns whose data values will always be unique for each tuple - these attributes are called **key attribute(s)** and are used to uniquely identify a tuple in the relation.

There may be a few possibilities for primary key – these are called Candidate keys

One candidate key is ultimately chosen as the primary key as part of the Design stage

DEFINITION:
PRIMARY KEY



A primary key is defined as one or more attributes, per table where:

- there can be only one such primary key per table
- the primary key can never contain the NULL value
- all values entered for the primary key must be unique (no duplicates across rows)

- Often primary keys are used as indexes (*will discuss later)
- We use the convention (in writing) that attributes that form the primary key are underlined

EXAMPLES
(Company schema): Adminer

Table: employee

Select data | Show structure | Alter table | New item

Column	Type	Comment
fname	varchar(50) *NULL*	
minit	varchar(1) *NULL*	
lname	varchar(50) *NULL*	
ssn	bigint(20)	
bdate	date *NULL*	
address	varchar(100) *NULL*	
sex	varchar(1) *NULL*	
salary	double *NULL*	
superssn	bigint(20) *NULL*	
dno	int(11) *NULL*	

Indexes

PRIMARY	*ssn*

What is the primary key of these tables?

See menti.com

Table: dept_locations

Select data | Show structure | Alter table | N

Column	Type	Comment
dnumber	int(11)	
dlocation	varchar(20)	

Indexes

PRIMARY	*dnumber, dlocation*

MySQL » mysql1.it.nuigalway.ie » mydb2974 » Table:

Table: works_on

Select data | Show structure | Alter table | New item

Column	Type	Comment
essn	bigint(20)	
pno	int(11)	
hours	double *NULL*	

Indexes

PRIMARY	*essn, pno*

Consider the `works_on` table:

A table to hold details on which projects an employee works on and the number of hours worked on each project:

`works_on(essn, pno, hours)`

Primary Key?

“ one of more attributes/columns whose data values will always be unique for each tuple.”

SOME SAMPLE DATA FROM works_on TABLE ....

essn	pno	hours
123456789	1	32.5
123456789	2	7.5
333445555	2	10

An employee can work on more than one project

A project can contain more than one employee

ALL DATA FROM THE works_on TABLE ....

essn	pno	hours
123456789	1	32.5
123456789	2	7.5
123456789	3	3
333445555	2	10
333445555	3	10
333445555	10	10
333445555	20	10
453453453	1	20
453453453	2	20
666884444	3	40
888665555	20	0
987654321	20	15
987654321	30	20
987987987	10	35
987987987	30	5
999887777	30	30

works_on(essn, pno, hours)

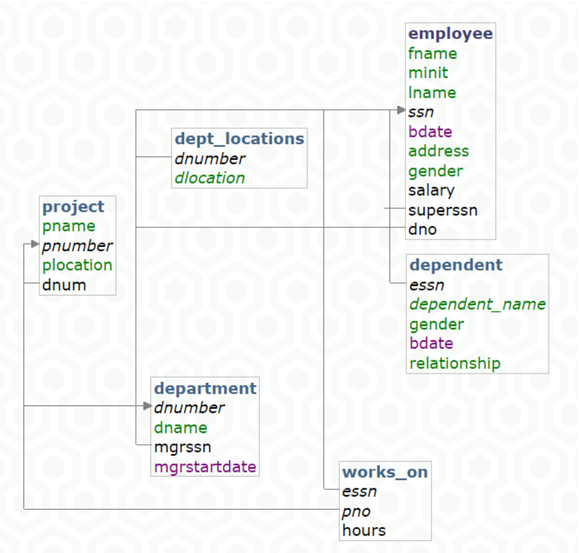
QUESTION: What are suitable primary keys for the following tables?

module(code, name, department, semester, exam_duration, ECTS)

student(ID, FirstName, LastName, HomeAddress, HomePhone)

car(EngineNo, CarReg, Make, Model, Year)

FOREIGN KEYS



Fundamental concept of Foreign Keys:

- Allows data in tables to be linked and cross-referenced by matching the same data values in both tables

Note:

- Matching must take place to primary or candidate keys
- There may be a few different links across the same tables

DEFINITION: FOREIGN KEY

A foreign key is an attribute, or set of attributes, within one table that matches or - links to - the candidate key of some other table (possibly the same table)

More formally - Given relations r_1 and r_2, a foreign key of r_2 is an attribute (or set of attributes) in r_2 where that attribute is a candidate key in r_1. relations r_1 and r_2 may be the same relations

FOREIGN KEY TERMINOLOGY

Often use the terminology of:

- parent, master or referenced table/relation for the relation containing the candidate key(s)
- child or referencing table/relation for the relation containing the foreign key

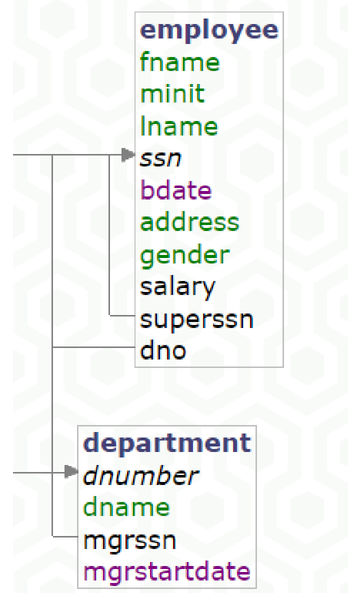
  For example:

In company schema, department is parent/master table (containing PK dnumber) and employee is child/referencing table (with FK dno)

Foreign keys

Source	Target	ON DELETE	ON UPDATE	
dno	department(*dnumber*)	RESTRICT	RESTRICT	Alter

EXAMPLE: FOREIGN KEY



employee

Modify	fname	minit	lname	ssn	bdate	address	gender	salary	superssn	dno
edit	John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	5
edit	Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5
edit	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	Woman	44183	333445555	5
edit	Ramesh	K	Narayan	666884444	1995-09-15	975 Fire Oak, Humble, TX	Man	60000	333445555	5
edit	James	E	Borg	888665555	1997-11-10	450 Stone, Houston, TX	Man	94199	*NULL*	1
edit	Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	4
edit	Ahmad	V	Jabbar	987987987	2000-03-29	980 Dallas, Houston, TX	Man	44183	987654321	4
edit	Alicia	J	Zelaya	999887777	1998-07-19	3321 Castle, Spring, TX	Non-binary	44183	987654321	4

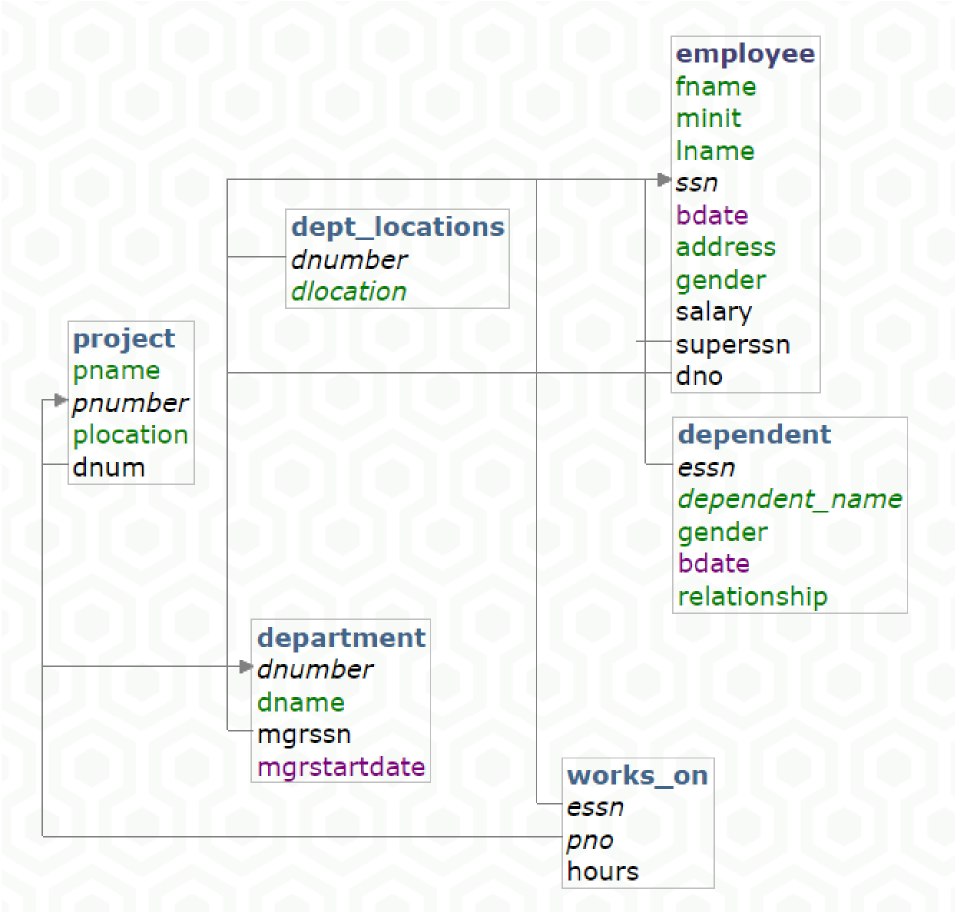
department

Modify	dnumber	dname	mgrssn	mgrstartdate
edit	1	Headquarters	888665555	2019-06-19
edit	4	Administration	987654321	2015-01-01
edit	5	Research	333445555	2018-05-22

dno is a foreign key in relation employee linking to dnumber in department

EXAMPLES (COMPANY SCHEMA): SEE menti.com

What is/are the foreign key(s) in the dependent table?
What is/are the foreign key(s) in the employee table?



SUMMARY: RELATIONAL MODEL

- Terminology and definitions associated with main concepts of the relational model **very important**
- Company schema will be used **extensively** for much of the course so a good understanding of it from these lectures is **very important**
- VERY important you get access to the CS Intranet and MySQL and import the company database this week if you are registered.
- Next … how to create tables and add data to tables…

INTRODUCTION TO SQL AND DATA DEFINITION LANGUAGE (DDL)

CT230
Database
Systems

LABS NEXT WEEK ….

Mon 19th 4-6 in IT106
Tue 20th 3-5 in IT101
Thur 23rd 10-2 in IT106 – will have assigned time before then

Please attend if you are able!

Nice working environment and you can get help if needed.

Main goals of the lab next week are:

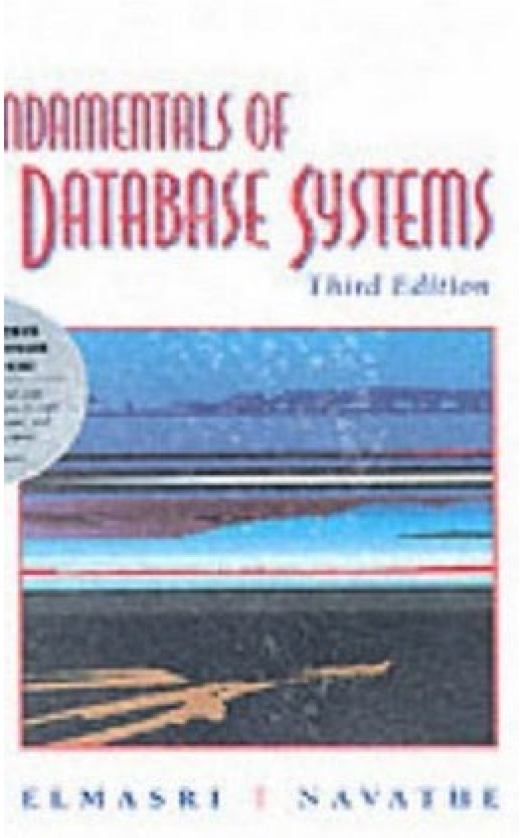
- Becoming familiar with phpMyAdmin and/or Adminer.
- Becoming familiar with the company database.
- Creating tables – GUI and DDL `CREATE TABLE`
- Adding data using `INSERT INTO`

SQL AND DDL

Relevant chapter in recommended book:

Elmasri and Navathe

Chapter 8 (3rd Edition)



QUESTIONS?

SQL:

- Structured
- Query
- Language

A special purpose programming language for relational database systems

FEATURES OF SQL:

- SQL is based on *relational algebra*:
  - All relational, set and hybrid operators are supported but SQL has additional operators to allow easier query development.
- SQL has been standardised since 1987 (SQL-86/SQL-87)
- The American National Standards Institute (ANSI) and the International Standards Organization (ISO) form SQL standards committees. Many vendors also take part.
- Recent standards include XML-related features in addition to many others (e.g., JSON data types etc.)

ANSI/ISO SQL

Despite standards there can be lack of portability between database systems due to:

- Complexity and size of standards (not all vendors will implement all of the standard).
- Vendor wants to keep syntax consistent with their other software products/OS or develop features to support user base.
- Want to maintain backward compatibility.
- Want to maintain “Vendor lock-in”.

ANSI/ISO SQL

We will concentrate on the standardised SQL syntax that should work across vendors:

Comprises three components:

DDL – data definition language

DCL – data control language

DML – data manipulation language

DCL: DATA CONTROL LANGUAGE

Used to control access to the database and to database relations.

Role of database administrator.

Very important in multi-user systems.

Typical commands:

- GRANT
- REVOKE

Each of these can be used to:

Grant/revoke access to database.

Grant/revoke access to individual relations.

DDL:
DATA DEFINITION LANGUAGE

Recall: what is schema?

Standardised language to **define** the **schema** of a database.

Back-end of "Design" options on Interface (e.g. Create options).

Typical tasks: create, modify, and remove database objects such as tables and indexes.

Common DDL keywords are:

CREATE

ALTER

DROP

ADD

CONSTRAINT

DML:
DATA MANIPULATION LANGUAGE

4 DML statements:

`INSERT`	insert data
`SELECT`	query data
`UPDATE`	update data
`DELETE`	delete data

BACK TO DDL COMMANDS:

We use the DDL commands to mostly create tables and add constraints to our database:

Common DDL keywords are:

CREATE

ALTER

DROP

ADD

CONSTRAINT

Create a table and it's indexes and constraints

Steps:

1. Specify table (relation) name.

2. For each attribute in the table specify:
   - Attribute Name (e.g., ssn)
   - Data Type (e.g., bigint).
   - Any constraints (e.g. not null).

3. Specify Primary key of table: choose one or more attributes.

4. Specify Foreign keys *if they exist* and assuming the attributes and table you are referencing exists (may have to return to this step).

** Steps 1-3 MUST be completed for all tables.

Recall: what is a primary key?

Recall: what is a foreign key?

DATA TYPES
3 MAIN TYPES: strings, numeric and date/time

The main ones you will use:

- char(size)
- varchar(size)
- bool/boolean
- tinyint, smallint(size), mediumint(size), int(size)/integer(size), bigint(size)
- double(size, d)
- float()
- decimal(size, d)
- date, datetime, timestamp, time, year

Important to pick a suitable data type and a suitable size (based on the sample data)

Strings	can contain letters, numbers, and special characters *size* parameter specifies the maximum column length in characters
`char(size)`	FIXED length. *size can* be from 0 to 255. Default is 1
`varchar(size)`	VARIABLE length. *size* can be from 0 to 65535
`text`	string

Date/time	
`date`	Format: YYYY-MM-DD
`time`	Format: hh:mm:ss
`datetime`	Format: YYYY-MM-DD hh:mm:ss
`year`	A year in four-digit format

… Important to pick a suitable data type and a suitable size (based on the sample data) *ctd.*

Numeric	*Max size value is 255* (mySQL supports UNSIGNED numeric types but not all DBMS do)
`Integers`	*See next slide*
`Bool/Boolean`	0 is False; non zero is True
`FLOAT`	Floating point number. 4 bytes, single precision
`DOUBLE`	Floating point number. 8 bytes, double precision
`DECIMAL(`*`size, d`*`)` `or dec(size,d)`	An exact fixed-point number. size = total number of digits (max 65, default 10) d = number of digits after the decimal point (max 30, default 0).

INTEGERS

Type	Bytes	Range
`tinyint`	1	-128 to 127
`smallint`	2	-32768 to 32767
`mediumint`	3	-8388608 to 8388607
`int`	4	-2147483648 to 2147483647
`bigint`	8	-9223372036854775808 to 9223372036854775807

Note:

Number in brackets (for integers) only refers to display not size

OTHERS

Unicode Char/String
Binary
Blob, Json etc.

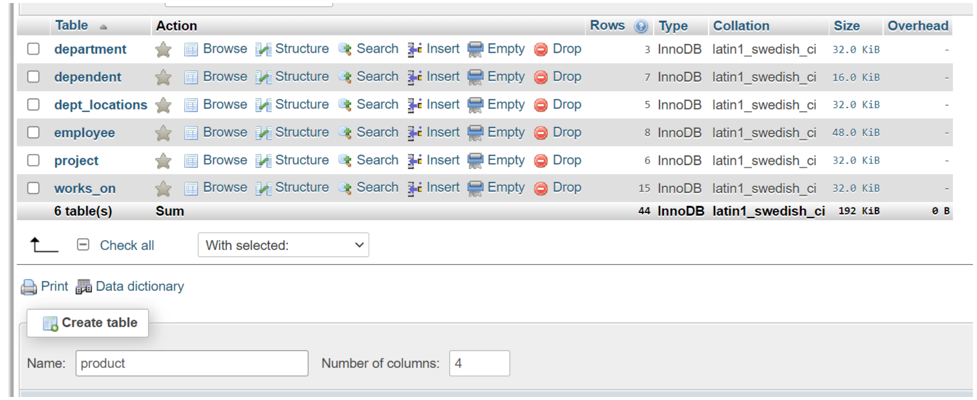
AUTONUMBER
AUTO_INCREMENT in mySQL

Specifying an attribute to be "AUTO-INCREMENT" tells the DBMS to generate a number automatically when a new tuple is inserted into a table.

Often this is used for an "artificial" primary key value which is needed to ensure we have a primary key but has no meaning for the data being stored – using auto-increment means that the DBMS takes care of inserting a unique value automatically every time a new tuple is inserted.

By default, **AUTO_INCREMENT** is 1, and is incremented by 1 for each new tuple inserted.

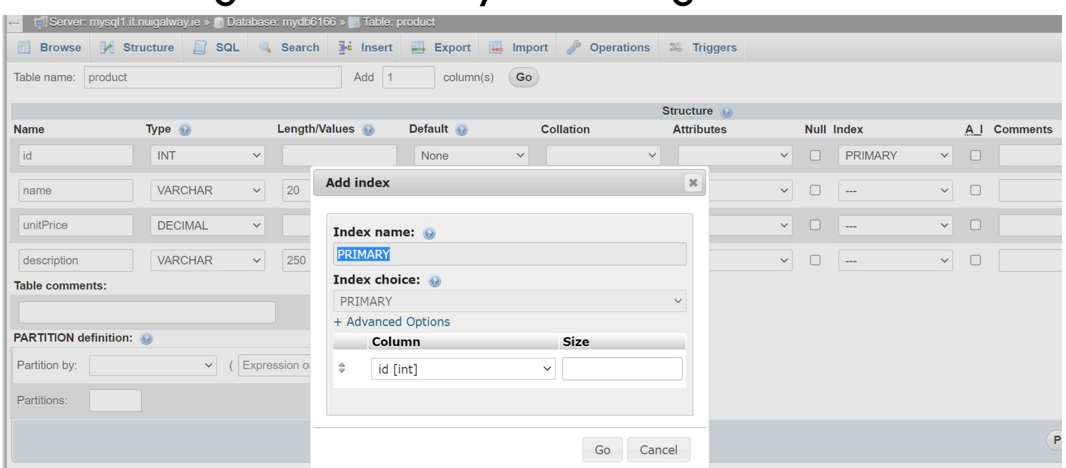
USING phpMyAdmin GUI to create a table and PK



Steps:

1: In the "Structure" view, in the "Create table" section, enter the new table name and number of columns and click the "Go" button.

2: In the new window, enter details of attributes (name and data types). Specify the keys in the Index option – "Primary" (for primary keys) and "Index" for Foreign keys (if they exist) and choose "Save". Note you may wish to view the SQL generated by choosing the "Preview SQL" option.

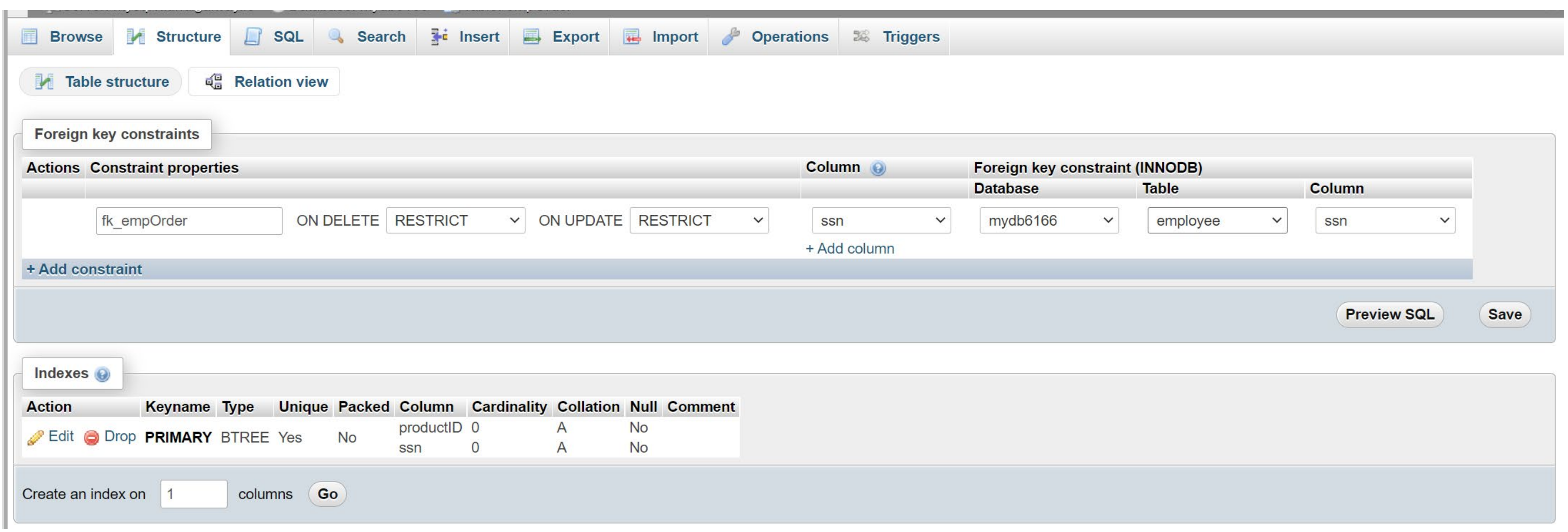


USING phpMyAdmin GUI to create Foreign keys

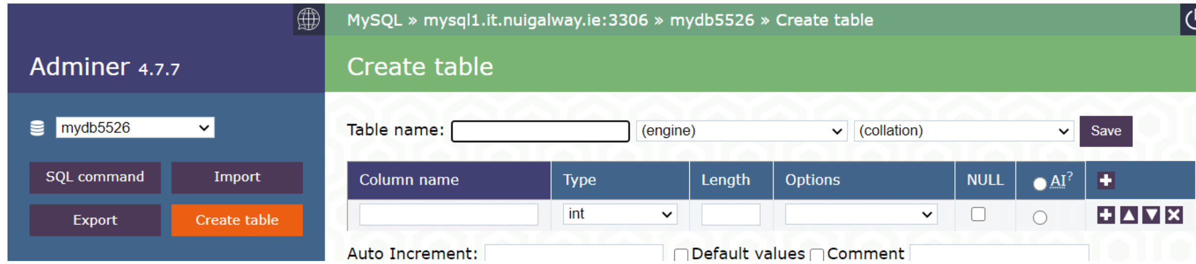
Steps:

3. Specify the FK by choosing the "Relation view" and choose the name, table and attribute that the FK references. Keep the ON DELETE and ON UPDATE as the default "RESTRICT" and choose save. (Note you might want to check "Preview SQL" again).

4. Look in Designer View to see the changes made.



USING GUI TO CREATE A TABLE
with Adminer

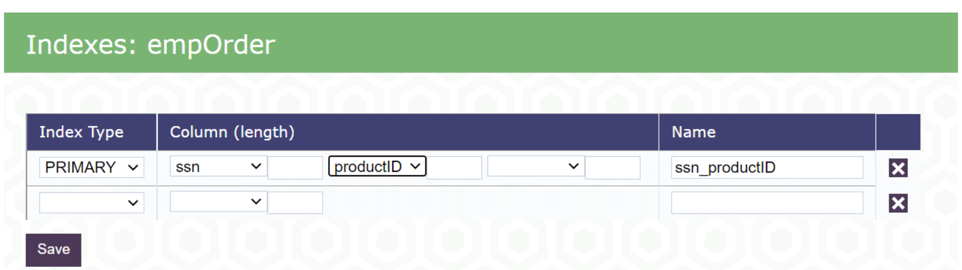


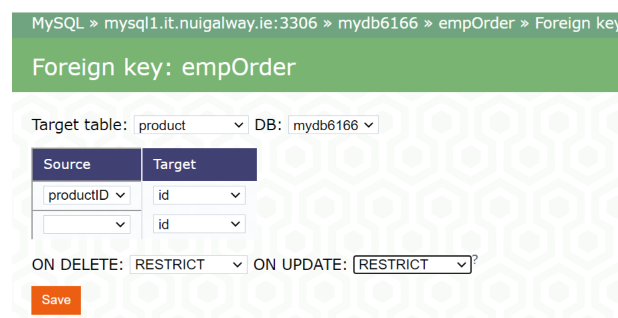
Steps:

1 and 2: Choose Create Table option and enter table name and details on attributes (name and data types). Choose the Save option.

3. Click on the table you created and choose the Alter Indexes option and specify Primary Key Index. Choose the Save option.

4. If there are foreign key(s) and the table being referenced exists, choose Add foreign key option and specify foreign keys. Choose the Save option. Else return to this step when other table(s) are created.





Using SQL DDL to create a table with index and constraints – when only one attribute is part of primary key

Syntax 1 (equivalent when only one Primary Key):

CREATE TABLE *tablename*

(attribute1 datatype [NOT NULL] [PRIMARY KEY],

attribute2 datatype [DEFAULT NULL],

attribute3 datatype,

….. ,

FOREIGN KEY (*attributename*) REFERENCES *tablename*(*attributename*)

);

Using SQL DDL to create a table with index and constraints – when more than one attribute is part of primary key

(See company2022.sql for examples!)

Syntax 2:

CREATE TABLE *tablename*

(attribute1 datatype [NOT NULL],

attribute2 datatype [DEFAULT NULL],

attribute3 datatype,

….. ,

PRIMARY KEY(*attributename(s)*),

FOREIGN KEY (*attributename*) REFERENCES *tablename*(*attributename*)

);

Naming the constraints …

Syntax 3 (name the constraints):

CREATE TABLE *tablename*

(attribute1 datatype [NOT NULL],

attribute2 datatype [DEFAULT NULL],

attribute3 datatype,

….. ,

CONSTRAINT *constraintname* PRIMARY KEY (*attributename*),

CONSTRAINT *constraintname* FOREIGN KEY (*attributename*) REFERENCES *tablename*(*attributename*)

);

Looking at DDL code for department

```
CREATE TABLE `department` (
`dnumber` int(20) NOT NULL PRIMARY KEY,
`dname` varchar(50) DEFAULT NULL,
`mgrssn` bigint(20) DEFAULT NULL,
`mgrstartdate` date DEFAULT NULL)
ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

NOTE: CONSTRAINTS: FOREIGN KEYS:

FOREIGN KEY (*attributename*) REFERENCES *tablename*(*attributename*)

Need to specify:

* Keyword FOREIGN KEY to indicate it is a foreign key constraint and the attribute name or attribute names that will be the foreign key in current table. If there is more than one attribute they should be separated by commas. Attribute names should be enclosed in brackets.

* Keyword REFERENCES to specify attribute it references by specifying the table name and the attribute name. Again attribute name(s) should be in brackets. Table name is outside the bracket.

Constraint examples from COMPANY Schema for works_on table

```
CONSTRAINT pk_works_on PRIMARY KEY (essn, pno),

CONSTRAINT fk_works_on_employee FOREIGN KEY (essn)
REFERENCES employee(ssn),

CONSTRAINT fk_works_on_project FOREIGN KEY(pno)
REFERENCES project(pnumber)
```

Looking at DDL code in `company sql` file

Note that:

- For this SQL dump the Foreign Keys were created after the tables, and after the data was entered (using INSERT INTO commands).
- Generally, it is better to create ALL the structure first and only then enter the data.
- Sometimes you can only add Foreign keys *after* all the tables have been created

USING ALTER TO MODIFY DESIGN

Remember: Cannot create a foreign key link *unless* the attribute it is referencing already exists

If you want to create everything but foreign keys initially you can add a foreign key later using the ALTER TABLE command

SYNTAX FOR ALTER COMMAND:

To add a constraint:

```
ALTER TABLE tablename
ADD CONSTRAINT constraintname FOREIGN KEY
(attributename) REFERENCES
tablename(attributename);
```

To add an attribute (column) constraint:

```
ALTER TABLE tablename
ADD attributename DATATYPE;
```

Looking at DDL code for Foreign Key constraint in department

```
ALTER TABLE `department`
ADD KEY `mgrssn` (`mgrssn`),
ADD CONSTRAINT `department_ibfk_2`
        FOREIGN KEY (`mgrssn`) REFERENCES `employee` (`ssn`);
```

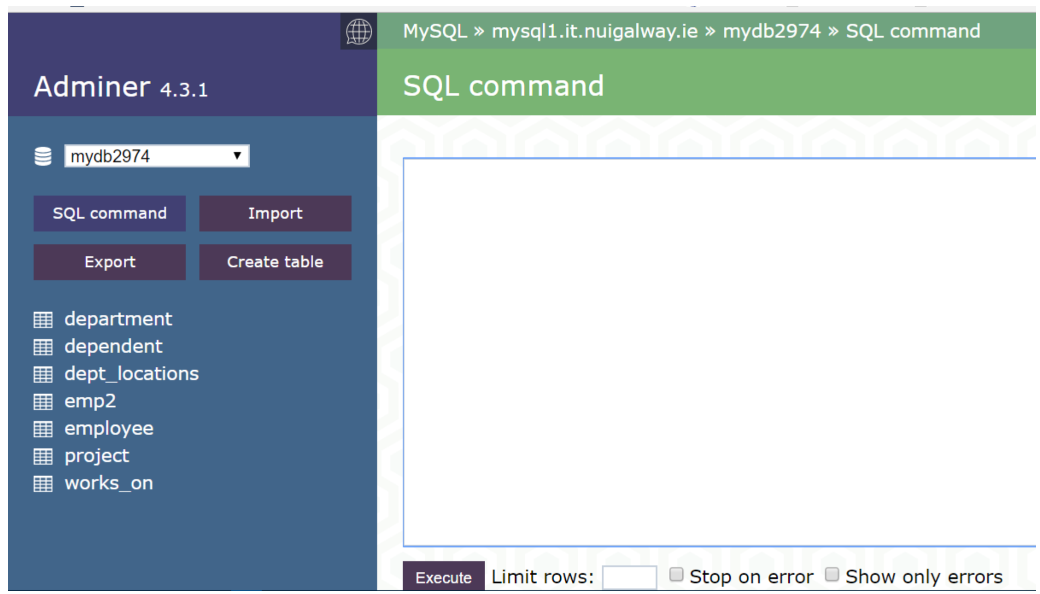
HOW TO WORK WITH DDL IN ADMINER?

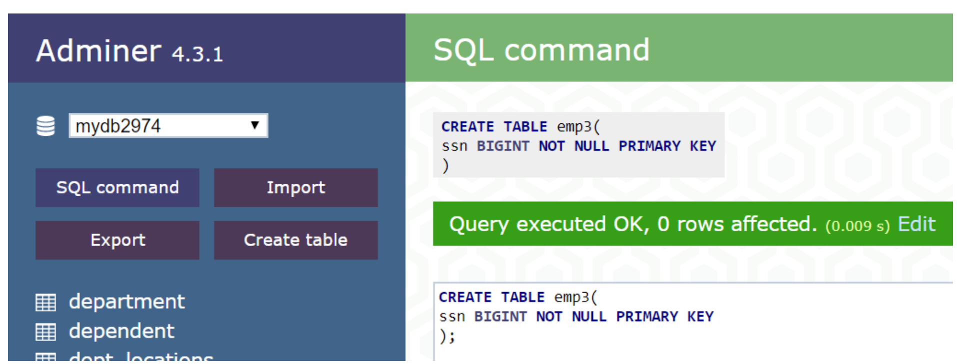
Choose:

1. Choose SQL command option

2. Once you have typed in the SQL in the displayed editor choose the Execute option

(or CTRL+Enter)





* Note you may want to save your query

HOW TO WORK WITH DDL IN phpmyadmin

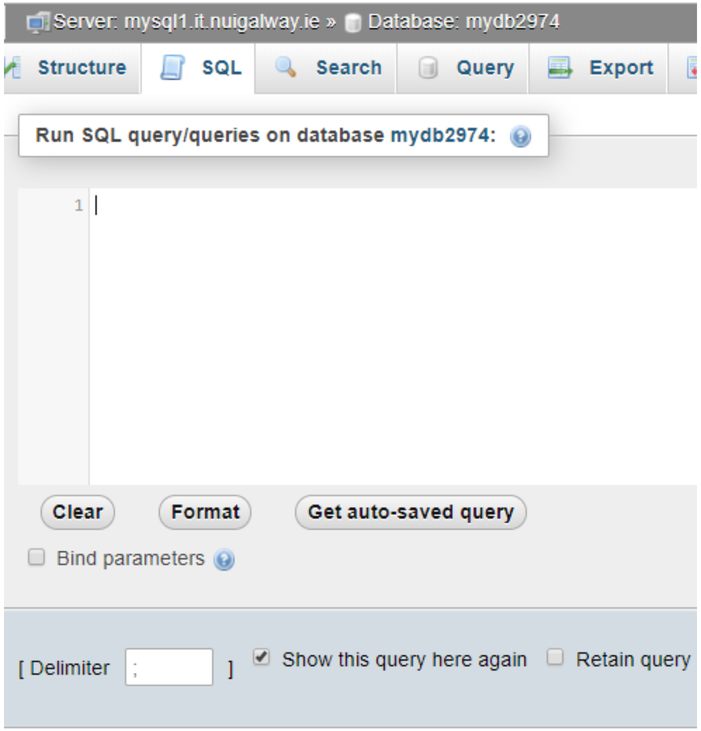
Choose:

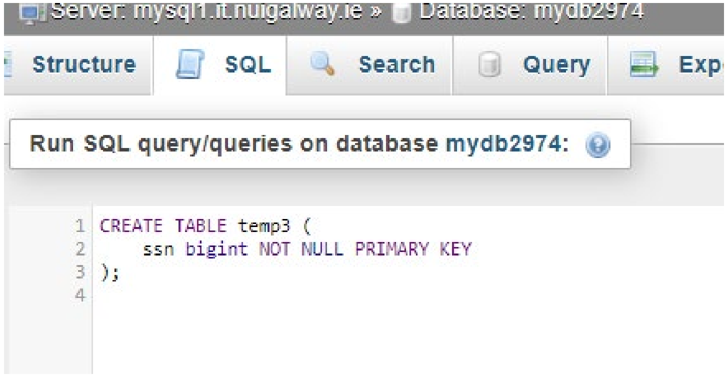
1. Choose SQL tab at the top

2. Type/Copy and Paste SQL in to the editor

3. Click "Go"

(or CTRL+Enter)





Looking at DML INSERT INTO code in `company2022.sql` file

Note that:

- Tuples are enclosed in brackets () and tuples are separated by commas
- Data type, format and order must correspond exactly to the data type, format and order specified when creating the tables.
- Strings, including dates, should be enclosed in single quotes
- Numbers are **not** enclosed in quotes

Looking at DML INSERT code for Foreign Key constraint in department

```
INSERT INTO `department`
(`dnumber`, `dname`, `mgrssn`, `mgrstartdate`) VALUES
(1, 'Headquarters', 888665555, '2019-06-19'),
(4, 'Administration', 987654321, '2015-01-01'),
(5, 'Research', 333445555, '2018-05-22');
```

IMPACT OF SETTING DATA TYPES, CONSTRAINTS (E.G., "NOT NULL), PRIMARY KEYS AND FOREIGN KEYS …

The DBMS has (Very!) strict checking of all constraints – and will not allow data to be entered if the data does not comply with the constraints set … this is one of the main advantages of a DBMS in terms of data correctness but it sometimes makes working with data entry difficult!

Consider the following examples ….

DOMAIN CONSTRAINTS

Definition: The value of each attribute A must be an atomic value from the domain dom(A).

- Can be tested easily by DBMS for data entry
- Queries can also be tested.
- Example attributes:
  - fname
  - minit
  - bdate

Column	Type
fname	varchar(50) *NULL*
minit	varchar(1) *NULL*
lname	varchar(50) *NULL*
ssn	bigint(20)
bdate	date *NULL*
address	varchar(100) *NULL*
gender	varchar(50) *NULL*
salary	double *NULL*
superssn	bigint(20) *NULL*
dno	int(11) *NULL*

Essentially: data types and formats must match to that specified

ENTITY INTEGRITY CONSTRAINTS
(PRIMARY KEY CONSTRAINTS)

Definition: The primary key should uniquely identify each tuple in a relation. This means:

- No duplicate values for primary key allowed
- Null values not allowed for primary key
- Example:
  - `ssn` in **employee** table
  - `essn` and `pno` in **works_on** table

Essentially: "no null or missing values for primary key"

NOTE:

As we already discussed, Null values may not be permitted for other attributes also. e.g., name of student may be constrained to be NOT NULL

- We often see this constraint when filling out forms online (*required) and the constraint is often necessary for many non-key attributes
- However, we should be careful of only adding 'NOT NULL' constraints in the databases in our own assignments when they are **really necessary**

REFERENTIAL INTEGRITY CONSTRAINTS

Definition: Specified between two relations and require the concept of a foreign key. The constraint ensures that the database must **not** contain any unmatched foreign keys.

Therefore a relationship involving foreign keys MUST be between attributes of the **same type and size**

In addition, a value for a foreign key attribute MUST exist already as a candidate key value.

Essentially: "no unmatched foreign keys"

EXAMPLE (AGAIN):

employee

Modify	fname	minit	lname	ssn	bdate	address	gender	salary	superssn	dno
edit	John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	5
edit	Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5
edit	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	Woman	44183	333445555	5
edit	Ramesh	K	Narayan	666884444	1995-09-15	975 Fire Oak, Humble, TX	Man	60000	333445555	5
edit	James	E	Borg	888665555	1997-11-10	450 Stone, Houston, TX	Man	94199	*NULL*	1
edit	Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	4
edit	Ahmad	V	Jabbar	987987987	2000-03-29	980 Dallas, Houston, TX	Man	44183	987654321	4
edit	Alicia	J	Zelaya	999887777	1998-07-19	3321 Castle, Spring, TX	Non-binary	44183	987654321	4

department

Modify	dnumber	dname	mgrssn	mgrstartdate
edit	1	Headquarters	888665555	2019-06-19
edit	4	Administration	987654321	2015-01-01
edit	5	Research	333445555	2018-05-22

Any referential integrity constraints problems with dno (a foreign key in relation employee) linking to dnumber in department?

SEMANTIC INTEGRITY CONSTRAINTS

Specified and enforced using a *constraint specification language*

Two types:

state constraints: e.g., “the maximum number of hours an employee can work on all projects per week is 39”

transition constraints: e.g., “the salary of an employee can only increase”; “the date entered for order delivery must not be in the past”

We will not use semantic integrity constraints

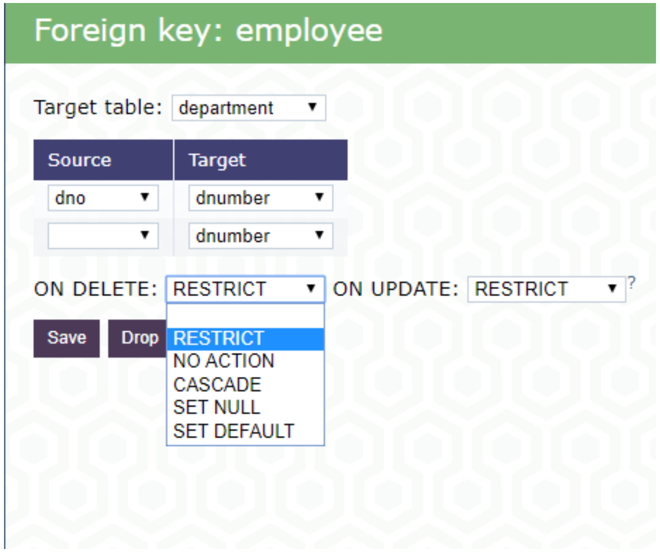
Consider the MySQL database and the associated data (company2022.sql):

Are there any unmatched foreign keys?

Are foreign and primary keys of same type and size?

SETTING CONSTRAINTS

- Domain constraints are set automatically once the data type is chosen
- Entity constrains are also set automatically once a primary key has been chosen
- Usually default constraints are set for foreign keys but these can be changed



UPDATE OPERATIONS AND CONSTRAINT VIOLATIONS

The DBMS must check that constraints are not violated whenever update operations are applied.

Three basic update operations on tables where constraints must be checked:

- insert
- delete
- modify

1. INSERT OPERATION

Provides a list of attribute values for a new tuple *t* that is to be inserted in to a relation *R*

This can happen directly via the interface or via a query

If a constraint is violated DBMS will reject insertion; usually with an explanation

Examples:

Using the company database state the problems, if any, with the following insertions to the database:

INSERT INTO employee VALUES

('Ciara', 'F', 'Smith', NULL, '1993-05-03', '2345 Tudor Heights, TX', 'Female', 40000, NULL, 4);

INSERT INTO employee VALUES

('Tony', 'D', 'Burns', 523523523, '1983-05-03', '34 Sycamore Drive, TX', ‘2000', 50000, NULL, 4);

INSERT INTO employee VALUES

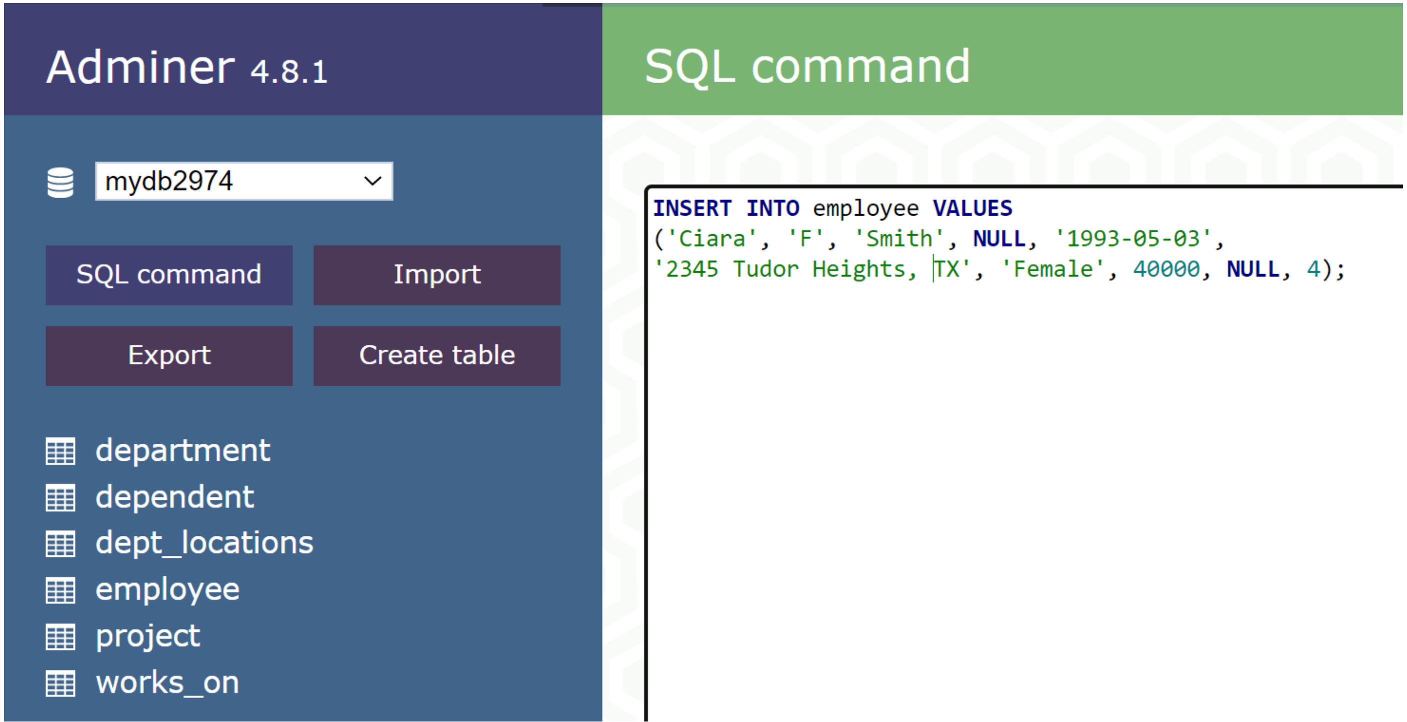
('Tony', 'D', 'Burns', 523523523, '1983-05-03', '34 Sycamore Drive, TX', 'Male', 50000, NULL, 14);

INSERT INTO employee VALUES

('Ciara', 'F', 'Smith', 4444555, '1993-05-03', '2345 Tudor Heights, TX', 'Female', 40000, NULL, 4);

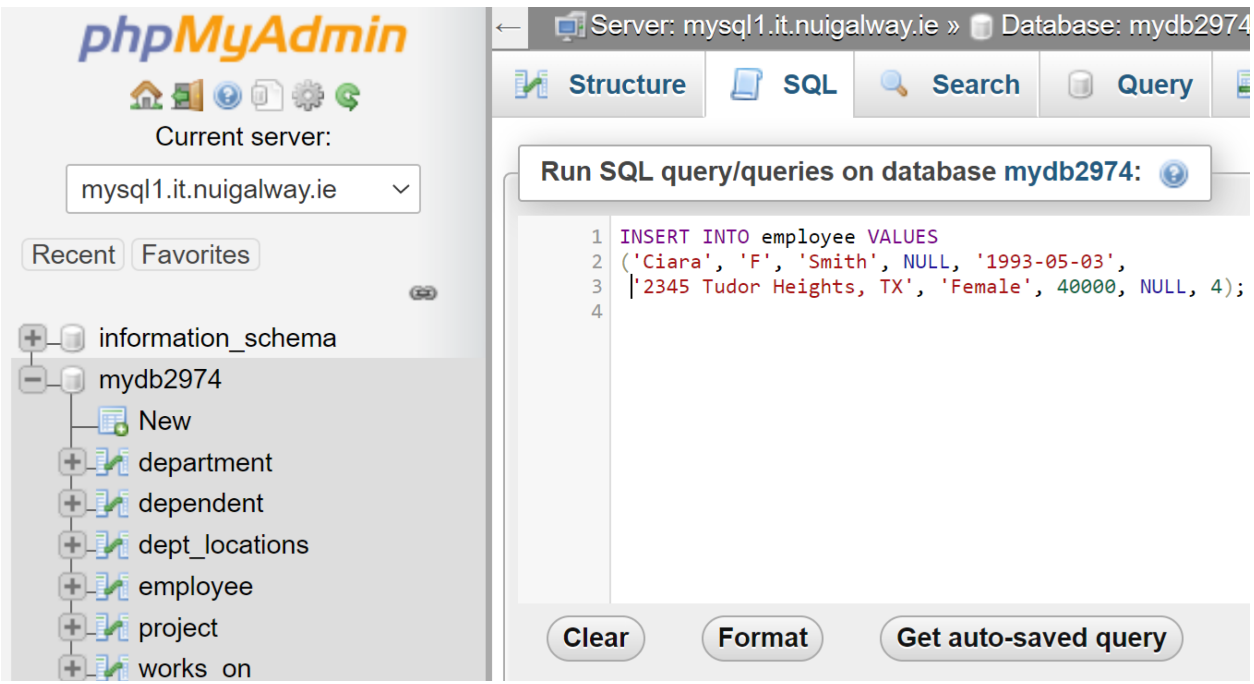
Trying this with Adminer:

1. Choose the “SQL command” button on LHS
2. A SQL editor is displayed on RHS
3. Type or copy and paste query in to editor
4. Choose “Execute” command



Trying this with phpMyAdmin

1. Choose the "SQL" tab on the top
2. A SQL editor is displayed in the middle of the screen
3. Type or copy and paste query in to editor
4. Choose "Go" button



2. DELETE OPERATION

A delete operation can only violate referential integrity constraints, i.e., if the tuple being deleted is referenced by the foreign keys from other tuples.

DBMS can:

reject deletion, with explanation

attempt to *cascade* deletion

modify referencing attribute

EXAMPLE: DELETE THE TUPLE JUST INSERTED (WITH SSN = 4444555)

```
DELETE FROM employee
WHERE ssn = 4444555;
```

3. UPDATE OPERATION

An update operation is used to change the values of one or more attributes in a tuple of a table

Issues already discussed with insert and delete could arise with this operation, specifically:

- if a primary key is modified … same as deleting one tuple and inserting another tuple in its place
- if a foreign key is modified … DBMS must ensure that new value refers to an existing tuple in the reference relation.

CASCADE UPDATE AND DELETE

Whenever tuples (rows) in the referenced (master) table are deleted (or updated), the respective tuples of the referencing (child) table with a matching foreign key column will be deleted (or updated) as well.

Note that if cascading DELETE is turned on there could be many deletions performed with the following query:

```
DELETE FROM employee

WHERE SSN = 123456789;
```

PROBLEM SHEETS/EXAM

- In problem sheet 1 you will practice DDL (and using the GUI (Create Table option) if you wish)
- In other assignments you will be asked to work with the DDL commands
- In exam, you will be asked for DDL commands but not any GUI questions

Therefore … it is important to know both approaches.

You try … Try adding these tables to the company database (choosing suitable data types):

These two tables keep track of products ordered by employees.

The product table contains a unique product id (the primary key of the table), name of the product, the unit price of the product and a description of the product).

The empOrder table contains the SSN of each employee who ordered a product, the ID of the product they ordered (productID) and the date they made the order. Note that ssn and productID are the primary keys, ssn is a foreign key to ssn in table employee and productID is a foreign key to id in table product:

```
product(id, name, unitPrice, description)
empOrder(ssn, productID, orderDate)
```

SQL DML STATEMENT

CT230
Database
Systems

QUESTIONS?

Recall: SQL:

- Structured
  - Query
    - Language

A special purpose programming language for relational database systems

Recall: ANSI/ISO SQL

Standardised SQL which comprises three components:

DDL – data definition language

DCL – data control language

DML – data manipulation language

DML: DATA MANIPULATION LANGUAGE

4 DML statements:

`INSERT`	insert data
`SELECT`	query data
`UPDATE`	update data
`DELETE`	delete data

DML SUPPORTS CRUD OPERATIONS

CRUD operations are the four basic functions we wish to perform on persistent data:

Create: insert a new tuple (INSERT)

Read: retrieve some data (SELECT)

Update: modify some data (UPDATE)

Delete: delete some data or a tuple (DELETE)

* we have already seen examples of INSERT, UPDATE and DELETE

SELECT

Basic syntax for an SQL select query to READ data consists of 6 clauses:

```
SELECT [DISTINCT] <attribute list>
FROM <table list>
WHERE <condition>
GROUP BY <group attributes>
HAVING <group condition>
ORDER BY <attribute list>
```

Notes:

- The order of the clauses cannot be changed
- SELECT and FROM are **always** required, other clauses are optional

NOTES ON SQL CLASS WORK:

For SQL SELECT work all examples with have a unique (!) number to ease cross-reference between lecture notes, your own attempts, and examples on Blackboard.

SELECT FROM WHERE

```
SELECT [DISTINCT] <attribute list>
FROM <table list>
WHERE <condition>
```

`<attribute list>` list of attribute (column) names (separated by commas) whose values will be retrieved by the query

`<table list>` list of table names (separated by commas) containing the attributes

`<condition>` Boolean expression that identifies the tuples to be retrieved by the query

WHERE clause: Boolean condition

For each tuple (row) in the table(s) which are part of query:

- tuple is checked to see if condition is true for this tuple
  - If true, tuple is part of the output
  - If not true, tuple is not part of the output

COMPARISON OPERATORS:

The comparison operators are:

=   <=   <   >   >=   !=

Conditions can be compounded by used of Boolean

AND, OR

Conditions can be negated with NOT

(Note: In some versions of SQL (e.g. in MS) the != operator is written as <>)

RECALL: SQL is case insensitive ...

But linux **is** case sensitive .... and web1.cs.nuigalway.ie is a linux server

Therefore need to be careful with table names in particular as

```
EMPLOYEE != employee
```

First SELECT Examples

Using the COMPANY relational database instance of the COMPANY SCHEMA develop SQL queries for the following:

attribute

table

1. List the names of all employees who earn more than 45000

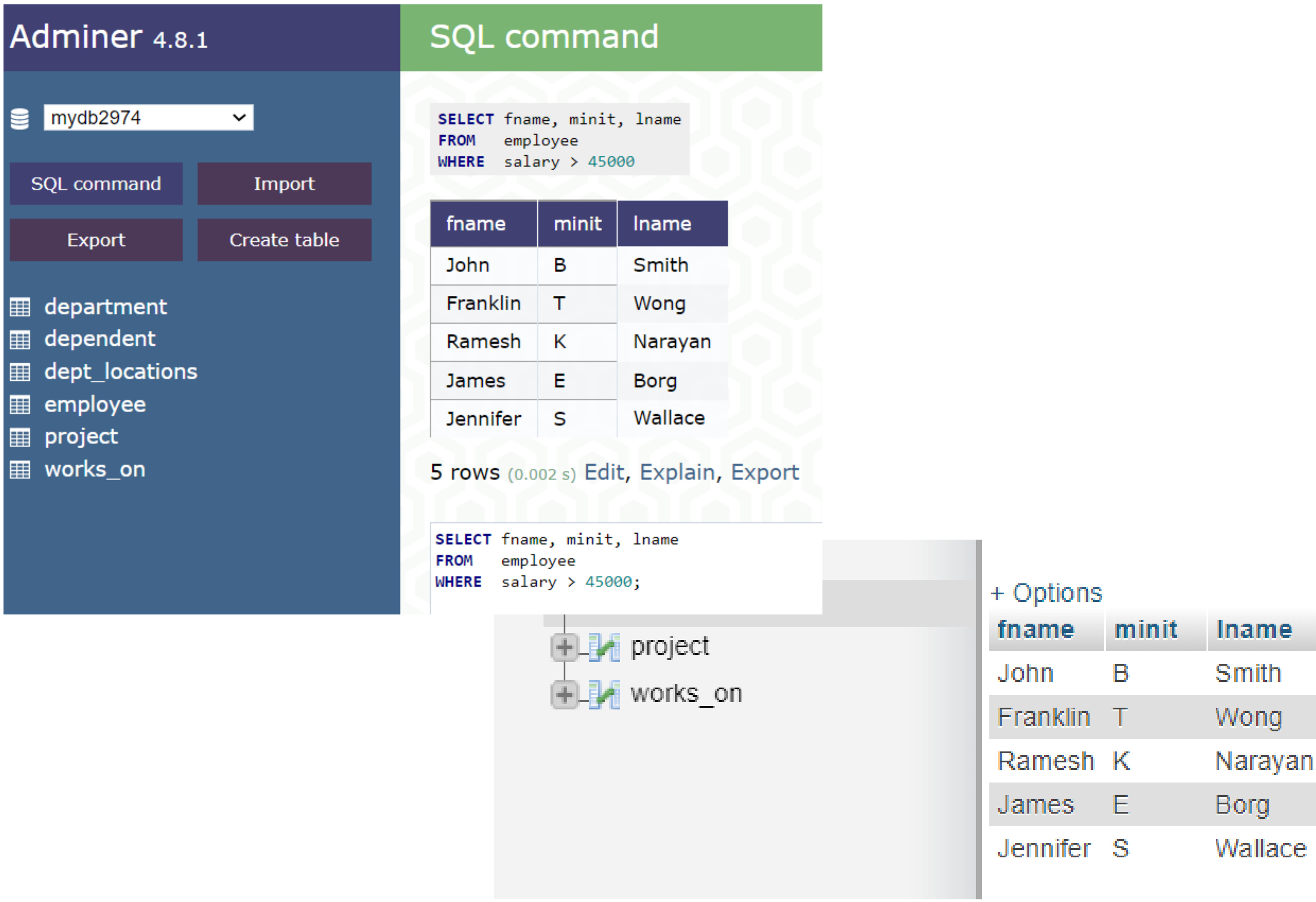
condition

employee(fname, minit, lname, ssn, bdate, address, gender, salary, superssn, dno)

```
SELECT    fname, minit, lname
FROM      employee
WHERE     salary > 55000;
```

What is output? … how many employees? … menti.com

mySQL ...


Adminer 4.8.1
SQL command
mydb2974
SQL command
Import
Export
Create table
department
dependent
dept_locations
employee
project
works_on
SELECT fname, minit, lname
FROM employee
WHERE salary > 45000
fname minit lname
John B Smith
Franklin T Wong
Ramesh K Narayan
James E Borg
Jennifer S Wallace
5 rows (0.002 s) Edit, Explain, Export
SELECT fname, minit, lname
FROM employee
WHERE salary > 45000;
project
works_on
+ Options
fname minit lname
John B Smith
Franklin T Wong
Ramesh K Narayan
James E Borg
Jennifer S Wallace

NOTE:

** Attribute names are separated by commas
** Numbers are **NOT** enclosed in quotes
** Strings are enclosed in quotes

SQL command

```sql
SELECT fname, minit, lname
FROM   employee
WHERE  salary > 45000
```

Using AND and OR ... SEE menti.com

What is the difference in output between these two versions of the query:

```
employee(fname, minit, lname, ssn, bdate, address, gender,
salary, superssn, dno)
```

```
SELECT    fname, minit, lname
FROM      employee
WHERE     dno != 5 AND salary > 45000;
```

```
SELECT    fname, minit, lname
FROM      employee
WHERE     dno != 5 OR salary > 45000;
```

Recall: BOOLEAN ALGEBRA:

In order for the Boolean AND of three conditions to be true, each individual condition (a, b, c) must be true.

Evaluation usually proceeds from Left to Right evaluating the TRUTH or each condition before returning True or False.

CODING STYLE

- Complying with coding style rules is crucial for a career in computing.
- Clean code is focused and understandable.
- Usually SQL keywords are capitalised and table and column names are mostly kept in lowercase unless combining words and not using an underscore
- Code should be organised *horizontally* and *vertically* (and not all written on one line).
- Code blocks are separated by a semi-colon.
- Use comments (#, --, /* and */) to explain code.

2 EXAMPLES TO TRY … menti.com

employee(fname, minit, lname, ssn, bdate, address, gender, salary, superssn, dno)

department(dname, dnumber, mgrssn, mgrstartdate)

dept_locations(dnumber, dlocation)

project(pname, pnumber, plocation, dnum)

works_on(essn, pno, hours)

dependent(essn, dependent_name, gender, bdate, relationship)

Example 2: Write a query to list the names of all projects located in Stafford

Example 3: Write a query to list the address and birth date of the employee with name John B Smith

Note: strings MUST BE enclosed in single quotes

Are these solutions correct?

#3: Write a query to list the address and birth date of the employee with name John B Smith

```
SELECT   bdate, address
FROM     employee
WHERE    fname = 'John B Smith';
```

```
SELECT   bdate, address
FROM     employee
WHERE    ssn = 123456789;
```

Be VERY careful of getting the "right" result using the "wrong" query

CALCULATED OR DERIVED FIELDS

Can specify an SQL expression in the SELECT clause which can involve numerical operations on numeric fields and counting operations on non-numeric fields

Example 4: Produce a list of monthly salaries for staff, showing staff ID and the salary details

```
employee(fname, minit, lname, ssn, bdate, address, gender,
salary, superssn, dno)
```

WILL THIS WORK?

Example 4: produce a list of monthly salaries for staff, showing staff ID (ssn) and the monrthly salary details

`employee(fname, minit, lname, ssn, bdate, address, gender, salary, superssn, dno)`

SELECT ssn, salary/12

FROM employee;

```
SELECT    ssn, salary/12
FROM      employee
```

ssn	salary/12
123456789	4604.16666666667
333445555	5416.66666666667
453453453	3681.9166666666665
666884444	5000
888665555	7849.91666666667
987654321	5770
987987987	3681.9166666666665
999887777	3681.9166666666665

8 rows (0.002 s) Edit, Explain, Export

TIDYING UP THE OUTPUT ….

1. Using Keywords CAST, AS and DECIMAL(x, y) to specify the total number of digits (x) and number of digits (y) after the decimal point when working with real numbers :

```
SELECT    ssn, CAST(salary/12.0 AS DECIMAL(8, 2))
FROM      employee;
```

2. Using Keyword AS to rename output:

```
SELECT    ssn, CAST(salary/12.0 AS DECIMAL(8, 2))
          AS mthlySalary
FROM      employee;
```

ssn	mthlySalary
123456789	4604.17
333445555	5416.67
453453453	3681.92
666884444	5000.00
888665555	7849.92
987654321	5770.00
987987987	3681.92
999887777	3681.92

USING KEYWORD DISTINCT

Keyword DISTINCT automatically removes duplicates from the returned result set.

Should be careful of using with large result sets as can be an expensive operation to perform (not a problem for our small examples).

QUESTION … how do you think DISTINCT could be implemented?

EXAMPLE 5:
Produce a list of all salaries

```
SELECT salary
FROM employee;
```

salary
55250
65000
44183
60000
94199
69240
44183
44183

8 rows (0.002 s) E

EXAMPLE 6:
Produce a list of DISTINCT salaries

```
SELECT DISTINCT salary
FROM employee;
```

SQL command

```
SELECT    DISTINCT salary
FROM      employee
```

salary
55250
65000
44183
60000
94199
69240

6 rows (0.002 s) Edit, Explain, Ex

NOTE:

To retrieve all attribute values of selected tuples, you do not have to explicitly list all the attribute names …. Instead can use SELECT *

May need to be careful of using this when you begin to join multiple tables or in real-world applications

```
SELECT *
FROM employee;
```

MORE EXAMPLES TO TRY: SEE menti.com

#7: Retrieve the address of the employee whose SSN is 123456789

#8: Retrieve all details stored on all employees in the employee table who work in department 4.

#9. List all locations where departments are (no need to list the department as well)

#10. Retrieve the salary and name of all employees working in department 5

SOME NEW OPERATORS:

BETWEEN : range search, including endpoints of range

IN : tests if a data value matches one of a list of values

(NOT IN)

LIKE : allows string comparison, when equality is too strict

IS NULL : allow an explicit search for NULL

Set Operators:

UNION, INTERSECTION,

MINUS/DIFFERENCE

EXAMPLE 11: Retrieve names of all employees whose salary is between 50000 and 80000

-- option1:

SELECT fname, minit, lname

FROM employee

WHERE salary >= 50000 AND salary <= 80000;

```
SELECT fname, minit, lname
FROM   employee
WHERE  salary BETWEEN 50000 AND 80000
```

fname	minit	lname
John	B	Smith
Franklin	T	Wong
Ramesh	K	Narayan
Jennifer	S	Wallace

4 rows (0.002 s) Edit, Explain, Export

-- option 2:

SELECT fname, minit, lname

FROM employee

WHERE salary BETWEEN 50000 AND 80000;

```
SELECT fname, minit, lname
FROM   employee
WHERE  salary BETWEEN 50000 AND 80000;
```

end points included in range

SUMMARY

The 3 most important keywords in Database Programming:

SELECT

FROM

WHERE

Practice with your own company database until questions 1-11 make sense to you!

MORE SQL OPERATORS, WORKING WITH STRINGS AND SUB-QUERIES

CT230

Database Systems 1

CONCATENATING STRINGS AND ORDERING OUTPUT

Although we want to store atomic attributes as much as possible we may not want to display string values in a way different to how they are stored

For example, for query #10. Retrieve the salary and name of all employees working in department 5, compare the outputs:

fname	minit	lname	salary
John	B	Smith	55250
Franklin	T	Wong	65000
Joyce	A	English	44183
Ramesh	K	Narayan	60000

name	salary
John B Smith	55250
Franklin T Wong	65000
Joyce A English	44183
Ramesh K Narayan	60000

KEYWORDS TO MODIFY OUTPUT …

AS

… Used to rename any output in SELECT

… can also be used to re-name (alias) tables in FROM

CONCAT

… concatenate strings

… similar usage to other programming languages

CAST

… CAST(*expression* AS *datatype*(*length*))

ORDER BY

… last clause in SQL to order output results

ORDERING THE OUTPUT WITH ORDER BY

Syntax:

```
ORDER BY <attribute list>
```

Allows the results of a query to be ordered by values of one or more attributes

Either ascending (**ASC**) or descending (**DESC**).

The default order is ascending.

** Must be last clause of the SELECT statement.

Note: ORDER is a reserved keyword!

```sql
SELECT    fname, minit, lname, salary
FROM      employee
WHERE     dno = 5
ORDER BY salary DESC
```

fname	minit	lname	salary
Franklin	T	Wong	65000
Ramesh	K	Narayan	60000
John	B	Smith	55250
Joyce	A	English	44183

```sql
SELECT    fname, minit, lname, salary
FROM      employee
WHERE     dno = 5
ORDER BY salary ASC
```

fname	minit	lname	salary
Joyce	A	English	44183
John	B	Smith	55250
Ramesh	K	Narayan	60000
Franklin	T	Wong	65000

TIDYING UP SQL CODE ... Example 11 again

EXAMPLE 11: Retrieve names of all employees whose salary is between 50000 and 80000

```
SELECT
    fname,
    minit,
    lname
FROM
    employee
WHERE
    salary BETWEEN 50000 AND 80000;
```

TIDYING UP OUTPUT… #11 again

```
SELECT
   CONCAT(fname, ' ', minit, ' ',   lname) AS Name
FROM
   employee
WHERE
   salary BETWEEN 50000 AND 80000
ORDER BY
   lname;
```

```
SELECT
   CONCAT(fname, ' ' , minit, ' ',   lname) AS Name
FROM
   employee
WHERE
   salary BETWEEN 50000 AND 80000
ORDER BY
   lname
```

Name
Tony D Burns
Ramesh K Narayan
John B Smith
Jennifer S Wallace
Franklin T Wong

EXAMPLE 12: Produce a list of salaries for all staff, produced in descending order of salary (outputting ssn, names and salary)

```
SELECT   CONCAT(fname, ' ', minit, ' ', lname) AS name, salary
FROM     employee
WHERE    dno = 5
ORDER BY salary DESC
```

name	salary
Franklin T Wong	65000
Ramesh K Narayan	60000
John B Smith	55250
Joyce A English	44183

TOP AND LIMIT (EXAMPLE 13)

SELECT TOP N clause is used to specify the number of tuples/rows (N) to return but it is not supported by mySQL. Instead mySQL supports a LIMIT N clause which has the same functionality. The LIMIT clause is listed at the end of the query.

Example 13: List the employees with the top 3 salaries

SELECT

ssn, CONCAT(fname, ‘ ‘ , lname) AS Name , salary

FROM

employee

ORDER BY

salary DESC

LIMIT 3;

```
SELECT   ssn, CONCAT(fname, ' ' , lname) AS Name, salary
FROM     employee
ORDER BY salary desc
LIMIT 3
```

ssn	Name	salary
888665555	James Borg	94199
987654321	Jennifer Wallace	69240
333445555	Franklin Wong	65000

NOTE: SINGLE AND DOUBLE QUOTES

MySQL usually allows single and double quotes to be used interchangeably.

Generally, single quotes should be used for strings (varchar(), text, etc.)

HOW TO DEAL WITH APOSTROPHES IN STRINGS ….

We must be careful because an opening quote could be accidently closed by an apostrophe.

To overcome this, if there is an apostrophe in a string it should be replaced by two apostrophes side-by-side (general rule for all special characters – have two of the character) or \

e.g., Find the salary for the employee with surname O'Grady

SELECT salary

FROM employee

WHERE lname = 'O''Grady';

N.B. Must also take care of this when inserting string data using INSERT INTO

Example from company database:

```
INSERT INTO employee VALUES
('Ciara', 'F', 'O'Reilly', 444555, '2002-05-03', '23 Tudor Lawn, Galway, IRL', 'Female', 44000,  NULL, 5);
```

```
Error in query (1064): Syntax error near 'Reilly', 444555, '2002-05-03', '23 Tudor Lawn, Galwa
```

```
INSERT INTO employee VALUES
('Ciara', 'F', 'O''Reilly', 444555, '2002-05-03', '23 Tudor Lawn, Galway, IRL', 'Female', 44000,  NULL, 5);
```

EXAMPLE 14: Using the operator Is Null retrieve names of all employees who **Do Not** have a supervisor (superssn IS NULL)

IS NULL : allow an explicit search for NULL

SELECT

FROM

WHERE

WORKING WITH STRINGS AND PATTERN MATCHING

SQL is case insensitive (apart from table names as mentioned if on linux server)

Case insensitivity also applies to string searching

However, *often* when working with strings we do not look for an exact match (i.e. an exact match using “=“)

To support partial matching often use pattern matching characters and `LIKE` with wildcard characters % and _

Symbol	Description	Example (fname)
%	Represents 0 or more characters	j% finds John, Joyce, James, Jennifer
_	Represents a single character	j___ finds John only

EXAMPLES (#15) … what is the difference?

```
SELECT    fname, lname
FROM      employee
WHERE     fname LIKE 'j%'
ORDER BY  fname
```

fname	lname
James	Borg
Jennifer	Wallace
John	Smith
Joyce	English

```
SELECT    fname, lname
FROM      employee
WHERE     fname LIKE 'j___'
ORDER BY  fname
```

fname	lname
John	Smith

```
SELECT    fname, lname
FROM      employee
WHERE     fname LIKE '%a%'
ORDER BY  fname
```

fname	lname
Ahmad	Jabbar
Alicia	Zelaya
Franklin	Wong
James	Borg
Ramesh	Narayan

CAN USE REGEXP FOR MORE COMPLICATED STRING MATCHING

Symbol	Description
^	Matches position at the **beginning** of the searched string
$	Matches position at the **end** of the searched string
[ ]	Matches any character inside the square brackets
[^]	Matches any character **not** inside the square brackets
*	Matches preceding character 0 or more times
+	Matches preceding character 1 or more times
|	Or
{n}	Matches preceding character n number of times

EXAMPLE 16a: Find the names of employees whose first names begin with *jo* or *ja*

```
SELECT fname, lname
FROM   employee
WHERE  fname REGEXP '^(jo|ja)'
```

fname	lname
John	Smith
Joyce	English
James	Borg

EXAMPLE 16b: Find the names of employees whose first names end with *n*

```
SELECT   fname, lname
FROM     employee
WHERE    fname REGEXP 'n$'
ORDER BY fname
```

fname	lname
Franklin	Wong
John	Smith

EXAMPLE 17: Find employees (name and address) who live in Houston

```sql
SELECT
    fname,
    lname,
    address
FROM
    employee
WHERE
    address REGEXP 'Houston'
ORDER BY
    fname
```

fname	lname	address
Ahmad	Jabbar	980 Dallas, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
James	Borg	450 Stone, Houston, TX
John	Smith	731 Fondren, Houston, Tx
Joyce	English	5631 Rice, Houston, TX

5 rows (0.002 s) Edit, Explain, Export

```sql
SELECT
    fname,
    lname,
    address
FROM
    employee
WHERE
    address LIKE '%Houston%'
ORDER BY
    fname
```

fname	lname	address
Ahmad	Jabbar	980 Dallas, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
James	Borg	450 Stone, Houston, TX
John	Smith	731 Fondren, Houston, Tx
Joyce	English	5631 Rice, Houston, TX

5 rows (0.002 s) Edit, Explain, Export

EXAMPLE 18:

Version 1: List the details (name and birth date) of the children of the employee with SSN 333445555

Version 2: List the details (name and birth date) of the children of Franklin T Wong

What is the difference?

For version 2, we need two tables and we need to explicitly link the two tables as part of the query (that is the `employee` and `dependent` tables) in order to meet this request or to use a sub-query

HOW TO ACCESS DATA ACROSS MULTIPLE TABLES?

3 potential approaches*:

- Joins
- Subqueries
- Union queries

* not all suitable for all problems

SUBQUERIES

- A subquery is a query within another query
  - Also called a ***nested*** query
- The subquery *usually* returns data that will be used in the main query
- Data returned from the subquery may be a set of values or a single value
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements

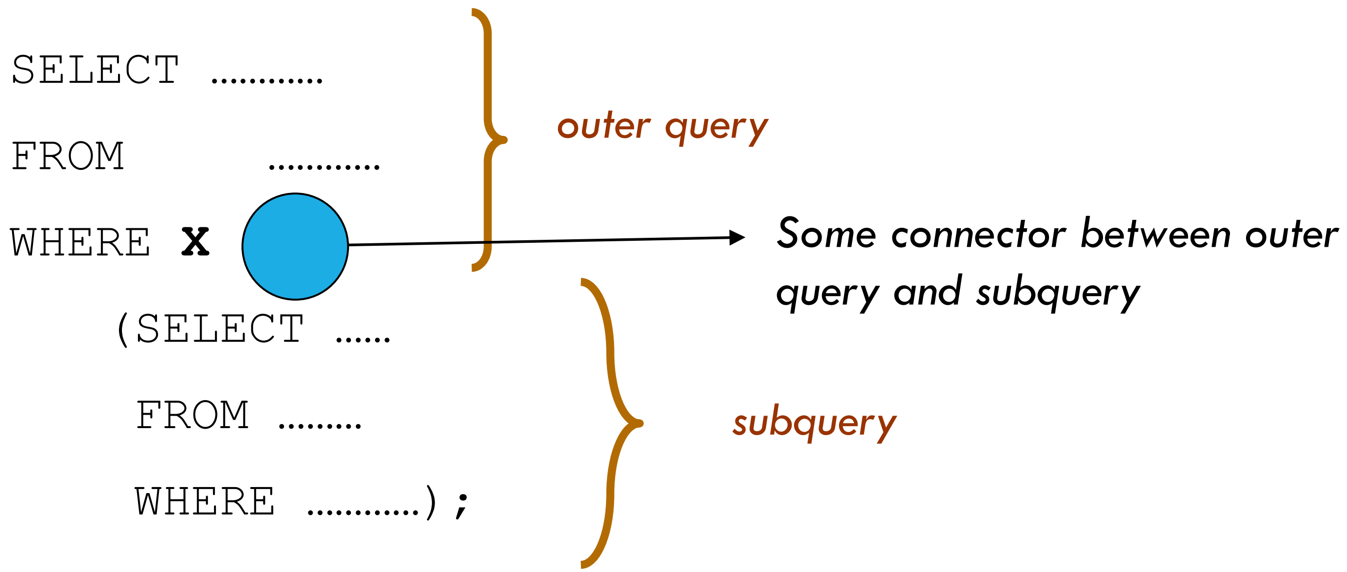
When to use a sub-query?

- Needed when an existing value from the database needs to be retrieved and used as part of the query solution.
- Needed when an aggregate function needs to be performed and used as part of a query solution.
- Can (sometimes) replace a join of tables (where appropriate).

Subqueries in SELECT

Subqueries can be used as part of the **WHERE** and **HAVING** clauses of an outer SELECT

SUBQUERY SAMPLE FORMAT:



Nested SELECT statement is called a *subquery*

SELECT statement which contains subquery is called an *outer query*

CONNECTING OUTER AND INNER QUERIES (1 OF 2)

If subquery returns only **one value** then can use operators such as:

=, !=, >, >=, <, <=

If subquery *could* return **more than one value (i.e., a list of values)** then need connectors such as:

IN, ANY, ALL to check through the values from the subquery.

CONNECTING OUTER AND INNER QUERIES (2 OF 2)

The keyword NOT can also be used where appropriate (often with `IN`, e.g., `NOT IN`)

In addition can have a more general condition using:

Exists: True if there exists at least one value in the result from a subquery

Not Exists: True if there is nothing in the result form a subquery (i.e. it is empty).

CONNECTORS: ANY, ALL

Used with basic mathematical operators: =, ! =, >, <, >=, <=

For example,

=ALL

>ANY

- ALL: the condition is true if the comparison is true for every (ALL) values returned by the subquery.
- ANY: the condition is true if the comparison is true for at least one (ANY) value returned by the subquery.

CONNECTOR: IN

Checks for equality.

Can be used for a list of values or a single value.

Does not require any additional mathematical operator.

The IN condition is true if the comparison is true for at least one value returned by the subquery, i.e. "a value is IN the subquery".

Returning to EXAMPLE 18:

Version 2: List the details (name and birth date) of the children of Franklin T Wong?

Using a sub-query:

- The sub-query should query the employee table to find the ssn of the employee Franklin T Wong.
- The outer query can then use the ssn returned by the subquery to check if the ssn exists (as an essn) in the dependent table. If/when a match is found return the name and birth date of the children.

EXAMPLE 18 *ctd.*

- “The sub-query should query the employee table to find the ssn of the employee Franklin T Wong”

```
SELECT  ssn
FROM    employee
WHERE   fname = 'Franklin' AND minit = 'T' AND lname = 'Wong';
```

- The outer query can then use the ssn returned by the subquery to check if the ssn exists (as an essn) in the dependent table. If/when a match is found return the name and birth date of the children (not spouse).

```
SELECT  dependent_name, bdate
FROM    dependent
WHERE   relationship != ‘spouse’ AND essn =
```

PUTTING THIS TOGETHER ....

```
SELECT  dependent_name, bdate
FROM    dependent
WHERE   relationship != 'spouse'
        AND essn =
        (SELECT  ssn
        FROM     employee
        WHERE  fname = 'Franklin' AND minit = 'T' AND lname = 'Wong')
```

dependent_name	bdate
Alice	2010-04-05
Theodore	2014-10-25

TRY …. EXAMPLE 19: Using a subquery method, list the staff (names) who work in department named 'headquarters'

EXAMPLE 20: Using subqueries, list the names of all employees who are in the same department as employee John B Smith

Steps:

1. Use a subquery to get John B Smith’s department (a single number)

2. Use outer query to find who else is in that department number

* Be careful not to return “John B Smith” in the answer – i.e. he is in his own department!

You try ....

#21 Retrieve the name and salary of all employees who work on a project for greater than 20 hours.

#22 Retrieve the names of employees who have no dependents (Hint: using NOT IN to connect the queries).

SUMMARY

- Working with strings is an important part of SQL coding.
- Writing code that is easy to read – and that produces easy-to-read output is also very important.
- We can nest queries so that we can access data across multiple tables (Sub-queries). It is very important to use the correct connector between outer and inner queries (often there is more than one suitable option).

SQL SELECT STATEMENT

Aggregate Functions
GROUP BY & HAVING clauses

CT230
Database Systems

AGGREGATE FUNCTIONS

Aggregate functions are only supported (can only be used) in **SELECT** clause and **HAVING** clause, even if we would like to use them elsewhere! (e.g as part of a condition in where clause)

- Keywords **`SUM, AVG, MIN, MAX`** work as expected and can only be applied to **numeric** data
- Keyword **`COUNT`** can be used to count the number of tuples/values/rows specified in a query
- Can also use mathematical operations as part of an aggregate function on **numeric** data (e.g., *, +, -, /).

USING SUM, MAX, MIN, AVG

Example 23: Find the total number of hours worked on projects in the company, the maximum and minimum hours worked by an employee on a project and the average number of hours worked.

```
SELECT    SUM(hours) AS 'Total Hrs Worked',
          MAX(hours) AS 'Max Hrs Worked',
          MIN(hours) AS 'Min Hrs Worked',
          ROUND(AVG(hours), 2) AS 'Avg Hrs Worked'
FROM      works_on;
```

Total Hrs Worked	Max Hrs Worked	Min Hrs Worked	Avg Hrs Worked
265	40	0	17.67

DOES THIS MAKE SENSE?

```
SELECT    ssn, SUM(salary) AS answer
FROM      employee;
```

EXAMPLE 24 What is the output?

```
SELECT
    SUM(salary)/12
FROM
    employee;
```

To Do: Tidy up the output …

WORKING WITH `COUNT()`

- Very useful aggregate function
- Counts the **number of tuples/rows** in a result
- Can only be used in SELECT and HAVING clauses, as with all aggregate functions
- Similar to count() and counta() in Excel and other spreadsheets

EXAMPLE 25:

How *many* employees earn over 60000

** *Note:*

- Do not want the employee names
- Want to count how many there are
- Want a number returned…so we use count()

```
SELECT
    COUNT(*) AS 'num earning > 60k'
FROM
    employee
WHERE
    salary > 60000;
```

NOTE:

Whatever is in the output it is the tuples/rows which are counted .... therefore it is not necessary to specify the attribute name

```
SELECT
    COUNT(*) AS 'num earning > 60k'
FROM
    employee
WHERE
    salary > 60000;
```

MORE COUNT() EXAMPLES:

Example 26: Using a sub-query find how many employees work on project with name 'ProductY'?

Example 27: Using a sub-query find how many children employee John Smith has?

Example 28: Find the yearly salary payments the company must make if everyone receives a 2% (.02) pay rise

Example 29: Find the number of employees working for the research department

USING A SUB-QUERY TO RETURN AN AGGREGATE VALUE

Example 30: Name the employees who earn greater than the average employee salary in the company

```
SELECT fname, lname
FROM   employee
WHERE  salary >
         (SELECT AVG(salary)
          FROM employee)
```

fname	lname
Franklin	Wong
Ramesh	Narayan
James	Borg
Jennifer	Wallace

4 rows (0.002 s) Edit, Explain, Export

Only a subquery will work here

EXAMPLE 30 VARIATIONS
Will these work?

```
SELECT fname, lname, AVG(Salary)
FROM   employee
```

```
SELECT fname, lname
FROM   employee
WHERE  salary > AVG(salary)
```

```
SELECT fname, lname
FROM   employee
WHERE  (SELECT AVG(salary)
        FROM employee) <= salary
```

YOU TRY …

Example 31:

How many employees earn the minimum salary in the company?

GROUP BY
HAVING

Recall:

```
SELECT [DISTINCT] <attribute list>
FROM <table list>
WHERE <condition>
GROUP BY <group attributes>
HAVING <group condition>
ORDER BY <attribute list>
```

GROUP BY

Syntax:

```
GROUP BY <group attributes>
```

- The GROUP BY clause allows the grouping (combining) of rows of a table together so that all occurrences within a specified group are collected together.
- Aggregate functions (min, max, avg, sum, count) can then be applied to the groups.

Example 32:
List the dno of each department

```
-- version 1
SELECT    dno
FROM      employee
GROUP BY dno;

-- version 2
SELECT    DISTINCT dno
FROM      employee;
```

```
SELECT   dno
FROM     employee
GROUP BY dno
```

dno
1
4
5

USING AGGREGATE FUNCTIONS WITH **GROUP BY** :

The GROUP BY clause specifies the group and the aggregate function is applied to the group.

- COUNT(*) can be used to *count* the number of rows (tuples) in the specified groups.
- AVG, SUM, MIN, MAX can be used to find average, sum, min and max of a *numerical value* in a specified group.

The aggregate function is not mentioned in the GROUP BY clause, but is specified in the SELECT clause.

* IMPORTANT *

You must GROUP BY ALL attributes mentioned in the SELECT clause *unless* they are involved in an aggregation.

EXAMPLE 33: List the department number and the number of employees in each department

```
SELECT      dno, COUNT(*) AS numEmps
FROM        employee
GROUP BY    dno;
```

dno	numEmps
1	1
4	3
5	4

EXAMPLE 34: List the department number and the total salary in each department

```
SELECT    dno, SUM(salary) AS sum_salary
FROM      employee
GROUP BY dno;
```

dno	sum_salary
1	94199
4	157606
5	224433

You try … EXAMPLE 35: For each department, retrieve the department number, the number of employees in the department, and the average salary of the department

```
SELECT

FROM

GROUP BY
```

EXAMPLE 36:

List the number of dependents of each employee who has dependents

Why is this wrong?

```
SELECT      dno,   salary
FROM        employee
GROUP BY    dno;
```

Error

SQL query:

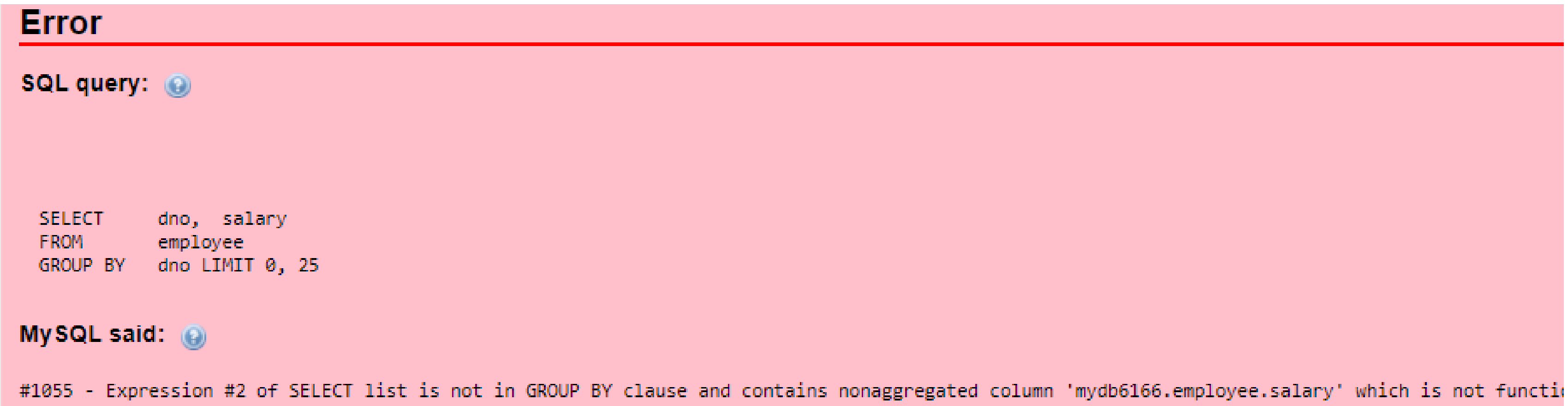
```
SELECT     dno,  salary
FROM       employee
GROUP BY   dno LIMIT 0, 25
```

MySQL said:

#1055 - Expression #2 of SELECT list is not in GROUP BY clause and contains nonaggregated column 'mydb6166.employee.salary' which is not functi

Recall:

- GROUP BY must contain all attributes in the SELECT clause that are not part of an aggregate function
- In the example, we cannot leave "salary" without a group



HAVING

Syntax:

```
HAVING <group condition>
```

The HAVING clause is used in conjunction with `GROUP BY` and allows specification of conditions on groups.

N.B. The column names used in the HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function, i.e., you cannot apply a HAVING condition to something that has not been calculated already.

Example 37: For each department that has more than 1 employee, retrieve the department number, the number of employees in the department and the average salary of the department.

```
SELECT      dno,
            COUNT(*) AS numEmps,
            AVG(salary) AS avgSalary
FROM        employee
GROUP BY    dno
HAVING      COUNT(*) > 1
```

Example 37: Tidying Output ...

```
SELECT     dno,
           COUNT(*) AS numEmps,
           CAST( AVG(salary) AS DECIMAL(10, 2)) AS avgSalary
FROM       employee
GROUP BY   dno
HAVING     COUNT(*) > 1
```

dno	numEmps	avgSalary
4	3	52535.33
5	4	56108.25

EXAMPLE 38: List the project number and the number of employees who work on the project for projects that have 2 or more employees

SELECT

FROM

GROUP BY

HAVING

ORDER BY

pno ▲ 1	Num Emps per Project
1	2
2	3
3	2
10	2
20	3
30	3

SUMMARY

Apart from Joins, have covered some of the most important aspects of SQL DDL and DML SELECT statements – with these you can build and query many databases.

Important to know:

- DDL `CREATE TABLE`
- DML `INSERT INTO`
- DML `SELECT`:
- Single table queries
- Multiple table queries with sub-queries *(To Do: Joins)*
- Aggregate functions
- Working with strings (LIKE, %, REGREP, etc.)
- Tidying Output (AS, CAST)

TOPIC:
ENTITY RELATIONSHIP MODELS

CT230
Database
Systems 1

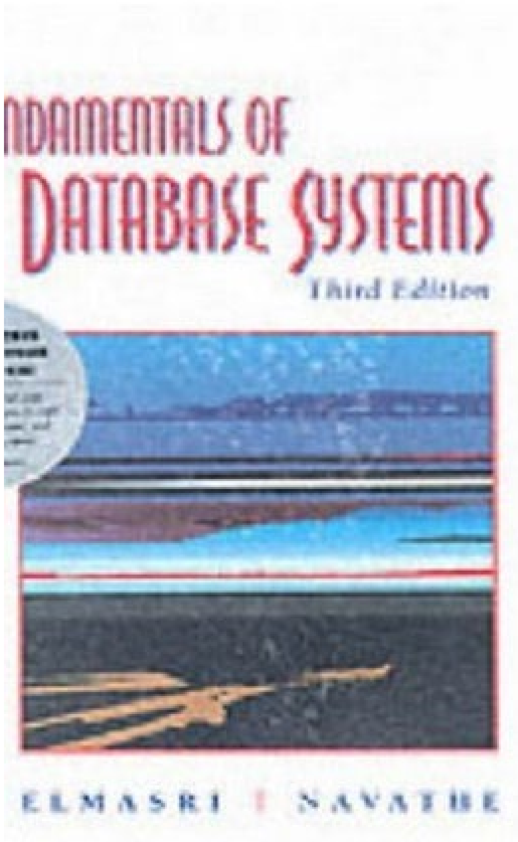
TOPIC:
Designing Tables with ER Models

See

Elmasri and Navathe book

Chapter 3 & Chapter 9

(3rd Edition)



DATA MODELS

Data models are concepts to describe the **structure** of a database. They comprise

- High level or logical models;
- Representational/Implementation data models;
- Physical Data models

Data models allow for **database abstraction**

DATA DESIGN and ENTITY RELATIONSHIP MODELS

Entity Relationship Models:

- Provide a way to **model the data** that will be stored in a system.
- The models are then used to **create tables** in the relational model.

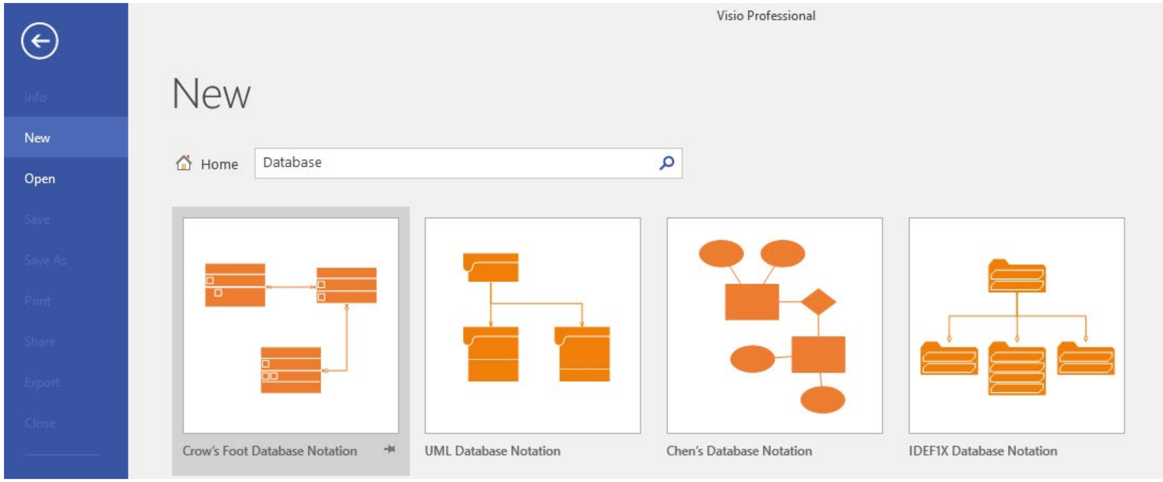
ENTITY RELATIONSHIP (ER) MODELS

ER models are a top-down approach to database design.

They are used to identify:

1. the important data to be stored in database called **entities.**
2. the **relationships** between the entities.
3. the **attributes** of entities.
4. the **constraints** of relationships and entities.

Software to Create ER Models

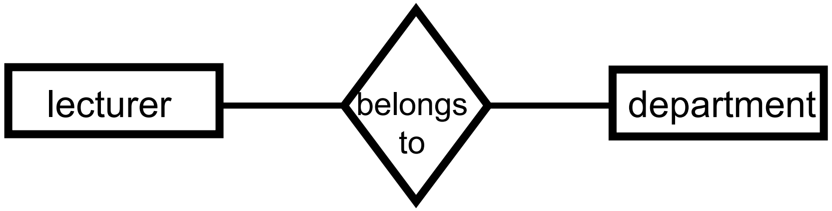


A comprehensive drawing package by Microsoft - MS Visio - supports the drawing of a large set of diagrams, including database ones. This is worth getting with your free Microsoft access.

Many other similar packages available:

- Edraw: https://www.edrawsoft.com/entity-relationship-diagrams.php
- Astah: http://astah.net/
- Lucidchart: www.lucidchart.com

NOTATION



A number of <u>different</u> notations can be used to represent the same model.

The original notation (Chen) uses diamonds, rectangles and ellipses.

It is easier to hand-draw so useful in an exam situation.

It is less implementation oriented than other notations.

ER MODEL NOTATIONS *CTD.*

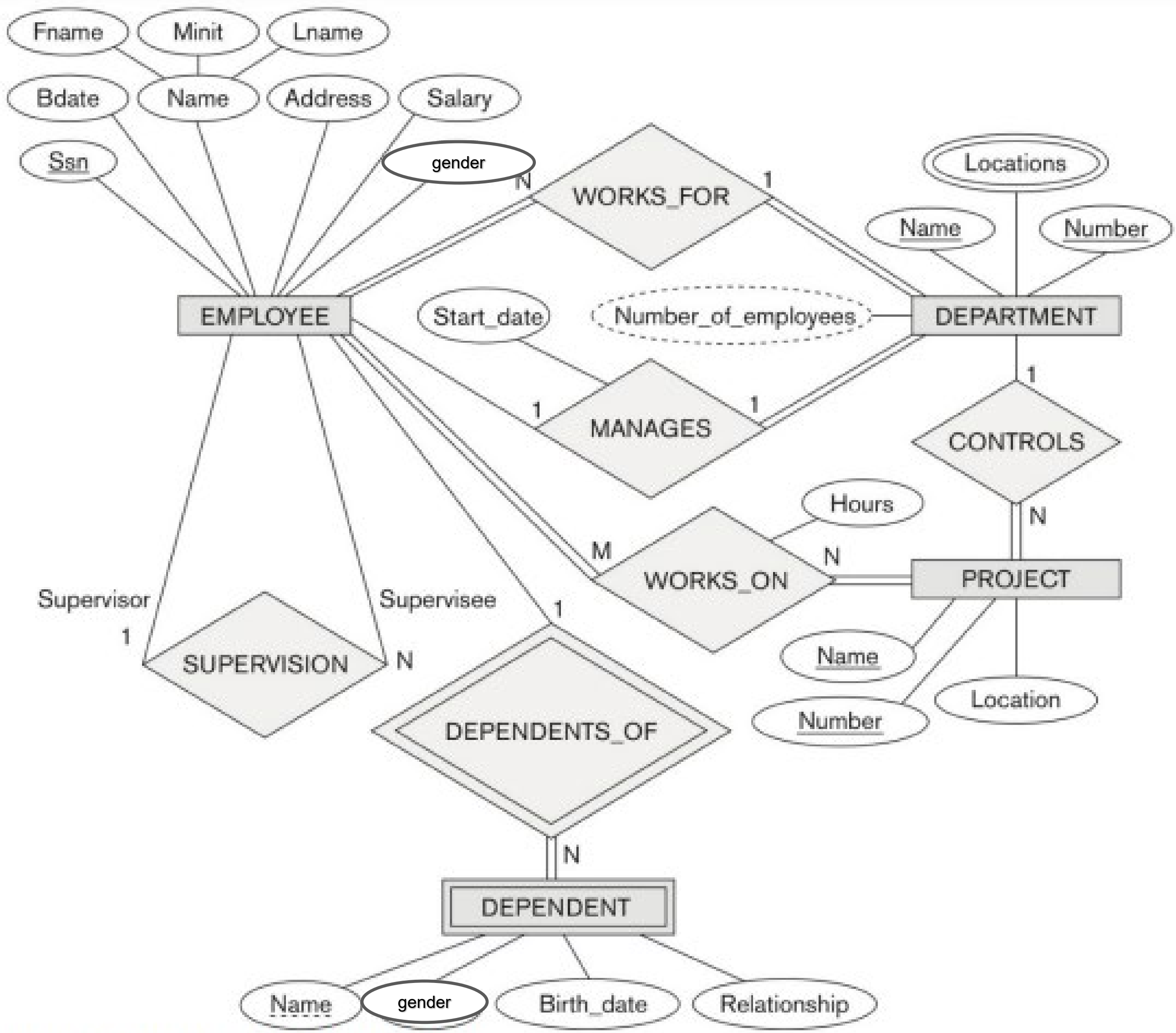
There are many notations in use, some of the more common:

- Chen Notation
- IE Crow's foot Notation
- UML
- Integrated Definition 1, Extended (IDEF1X)

Different software products often have their own minor variations of the above.

COMPANY ER MODEL EXAMPLE

Consider the ER diagram (Chen's notation) of the Company Schema ….


Fname
Minit
Lname
Bdate
Name
Address
Salary
Ssn
gender
N
WORKS_FOR
1
Locations
Name
Number
EMPLOYEE
Start_date
Number_of_employees
DEPARTMENT
1
1
1
MANAGES
CONTROLS
Hours
N
M
N
WORKS_ON
PROJECT
Supervisor
Supervisee
1
1
SUPERVISION
N
Name
Number
Location
DEPENDENTS_OF
N
DEPENDENT
Name
gender
Birth_date
Relationship

SOME DEFINITIONS:

Entity type: group of objects, with the same properties, which are identified as having an independent existence …

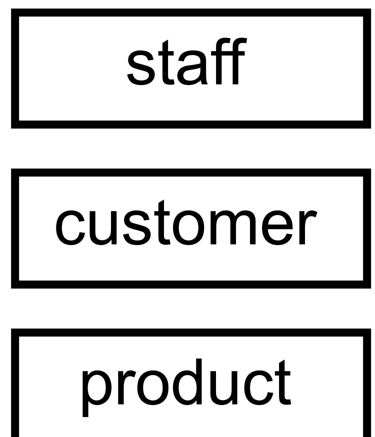
e.g.,

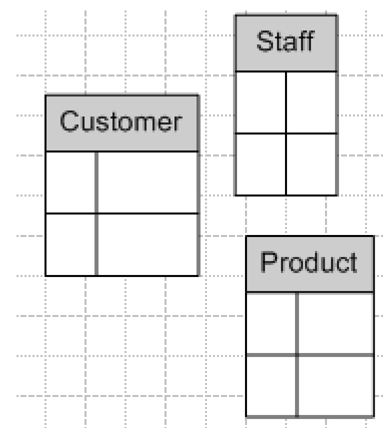
staff

customer

product

employee

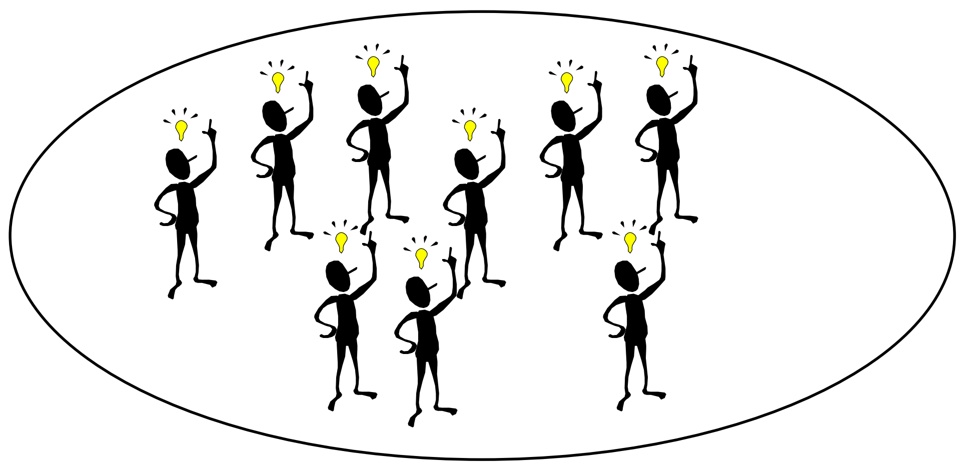




ENTITY INSTANCE AND ENTITY TYPE

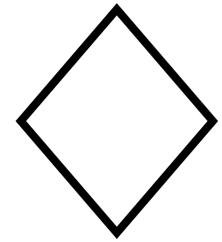
- An entity type is a collection of *entity instances* that share common properties or characteristics
- An entity instance or entity occurrence is a single uniquely identifiable occurrence of an entity type (e.g., row in a table).

Entity Instance: One employee, e.g. John B Smith



Entity Type: employee

RELATIONSHIP TYPE:

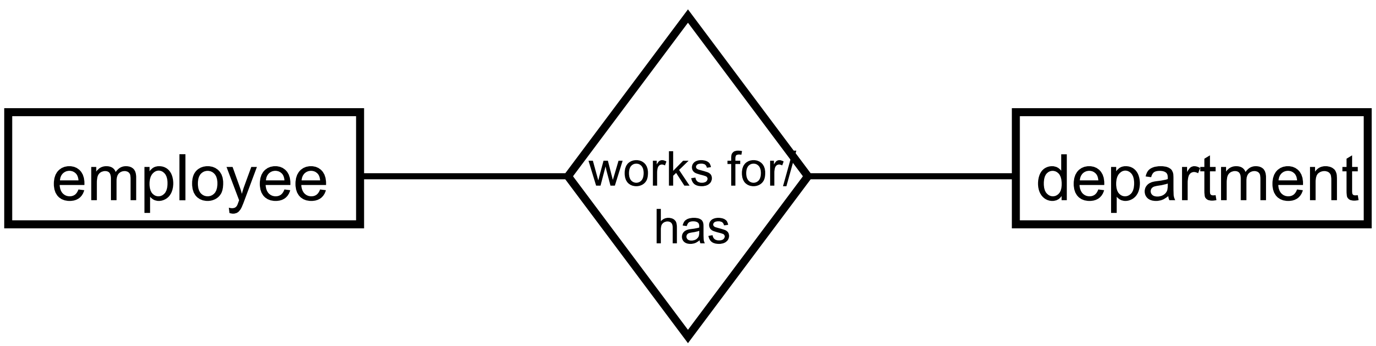


A set of meaningful relationships among entity types

e.g.,

employee “works for” department

department “has” employee

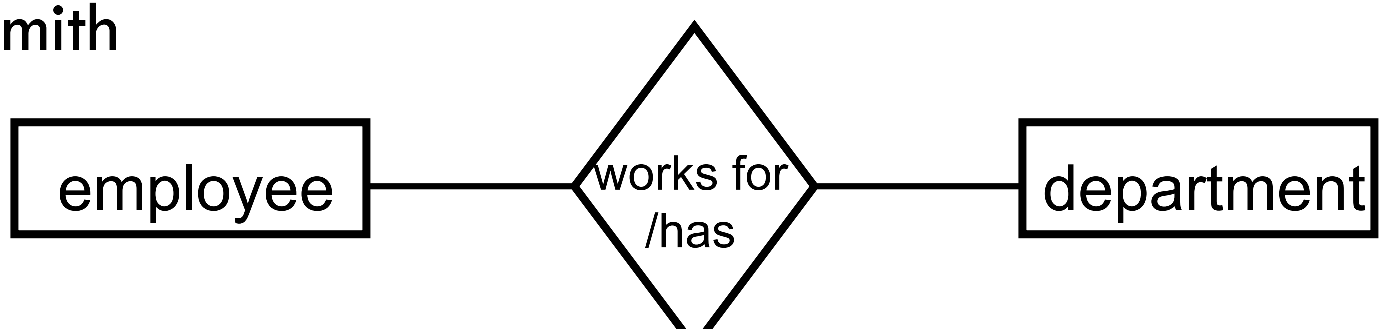


RELATIONSHIP OCCURRENCE (INSTANCE):

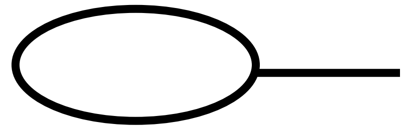
A uniquely identifiable association which includes one occurrence from each participating entity type; reading left to right and right to left.

e.g.

- Left-to-Right: John Smith "works for" Research department
- Right-to-left: Research department "has" John Smith



ATTRIBUTES



Attributes are a named property or characteristic of an entity.

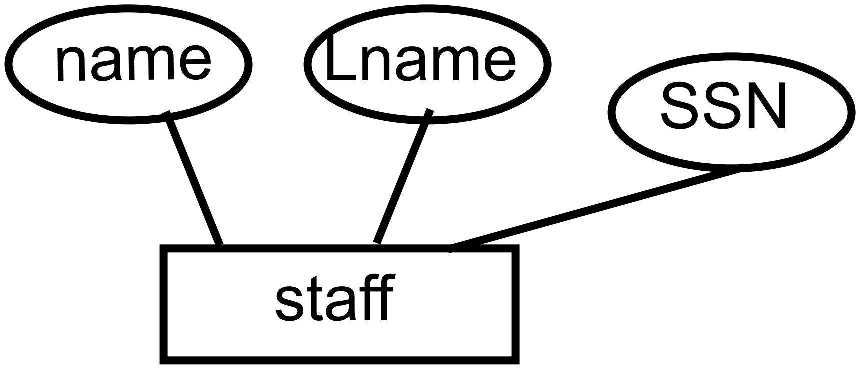
Each entity has a set of attributes associated with it.

Several types of attributes exist:

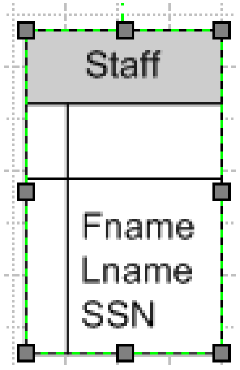
- Key
- Composite
- Derived
- Multi-valued

ATTRIBUTE NOTATION

Chen: An oval enclosing the name of the attribute



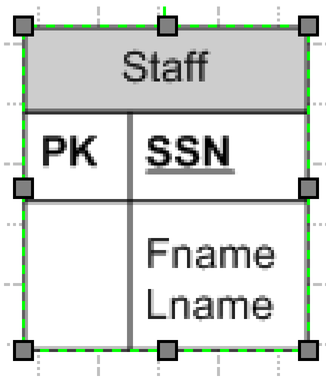
Crow: Listed in the entity box



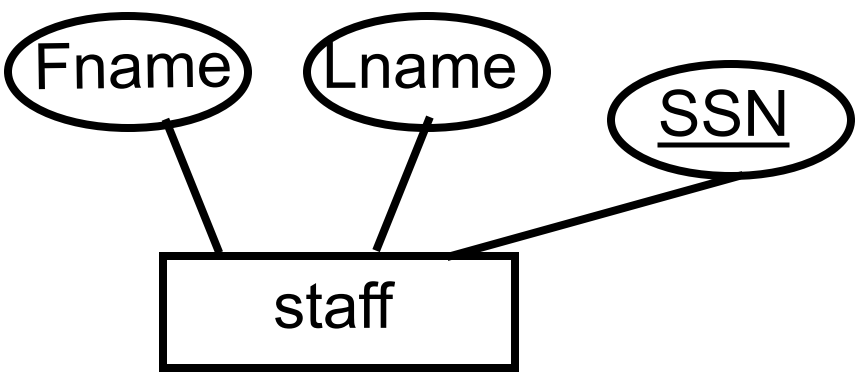
KEY ATTRIBUTES

- Each entity type must have an attribute or set of attributes that uniquely identifies each instance from other instances of the same type.
- A candidate key is an attribute (or combination of attributes) that uniquely identifies each instance of an entity type.
- A primary key (PK) is a candidate key that has been selected as the identifier for an entity type.
- **Notation:** Underline attribute name chosen as primary key

PK NOTATION: SSN PRIMARY KEY



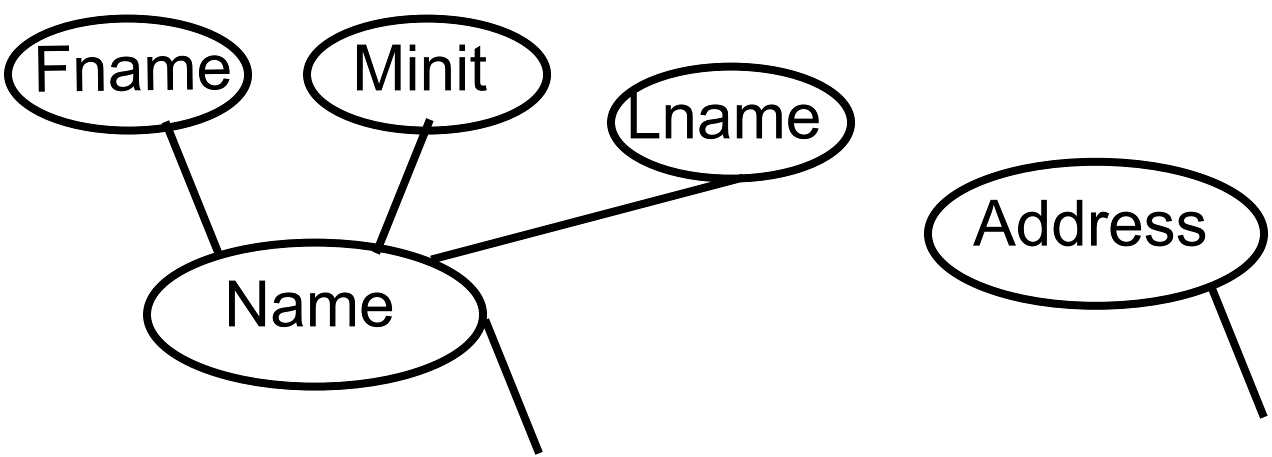
Physical Name	Data Type	Req'd	PK	
Fname	CHAR(10)	☐	☐	Fname is of Staff
Lname	CHAR(10)	☐	☐	Lname is of Staff
SSN	CHAR(10)	☑	☑	SSN identifies Staff
		☐	☐	

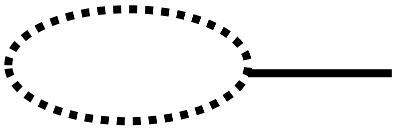


COMPOSITE AND SIMPLE (ATOMIC) ATTRIBUTES

A composite attribute is an attribute that is composed of several more basic/atomic attributes.

If the composite attribute is referenced as a whole only, then there is no need to subdivide it into component attributes, otherwise you should divide it:





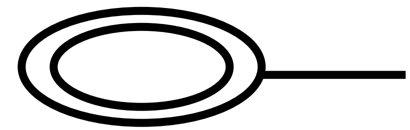
STORED AND DERIVED ATTRIBUTES

A **derived attribute** is an attribute whose value can be determined from another attribute.

For Chen's notation, the notation is a dotted oval.

For crow's foot notation, derived attributes can be represented by enclosing the attribute in [], e.g., [age].

MULTI-VALUED ATTRIBUTES

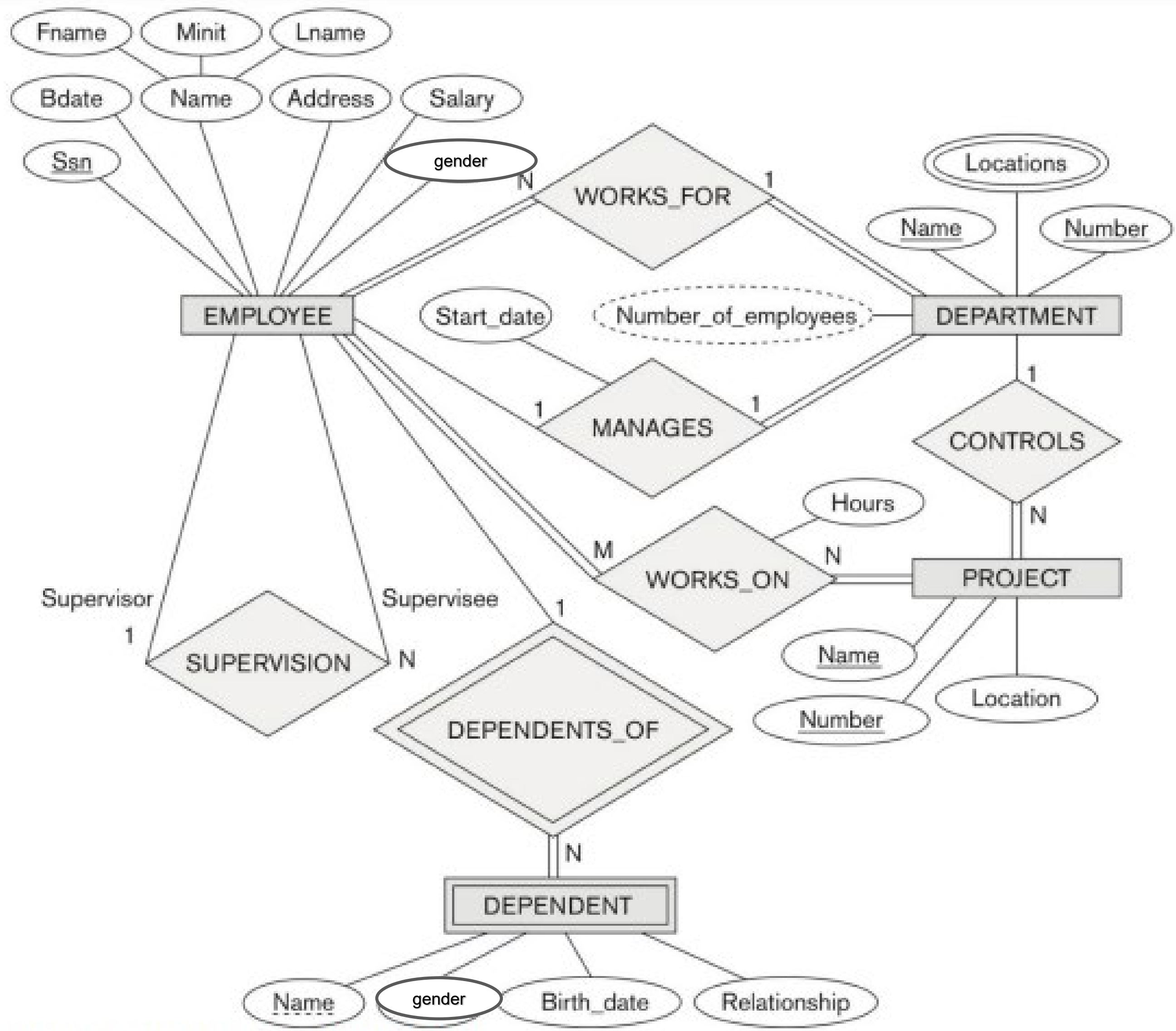


A **multi-valued attribute** is an attribute which has lower and upper bounds on the number of values for an individual entry.

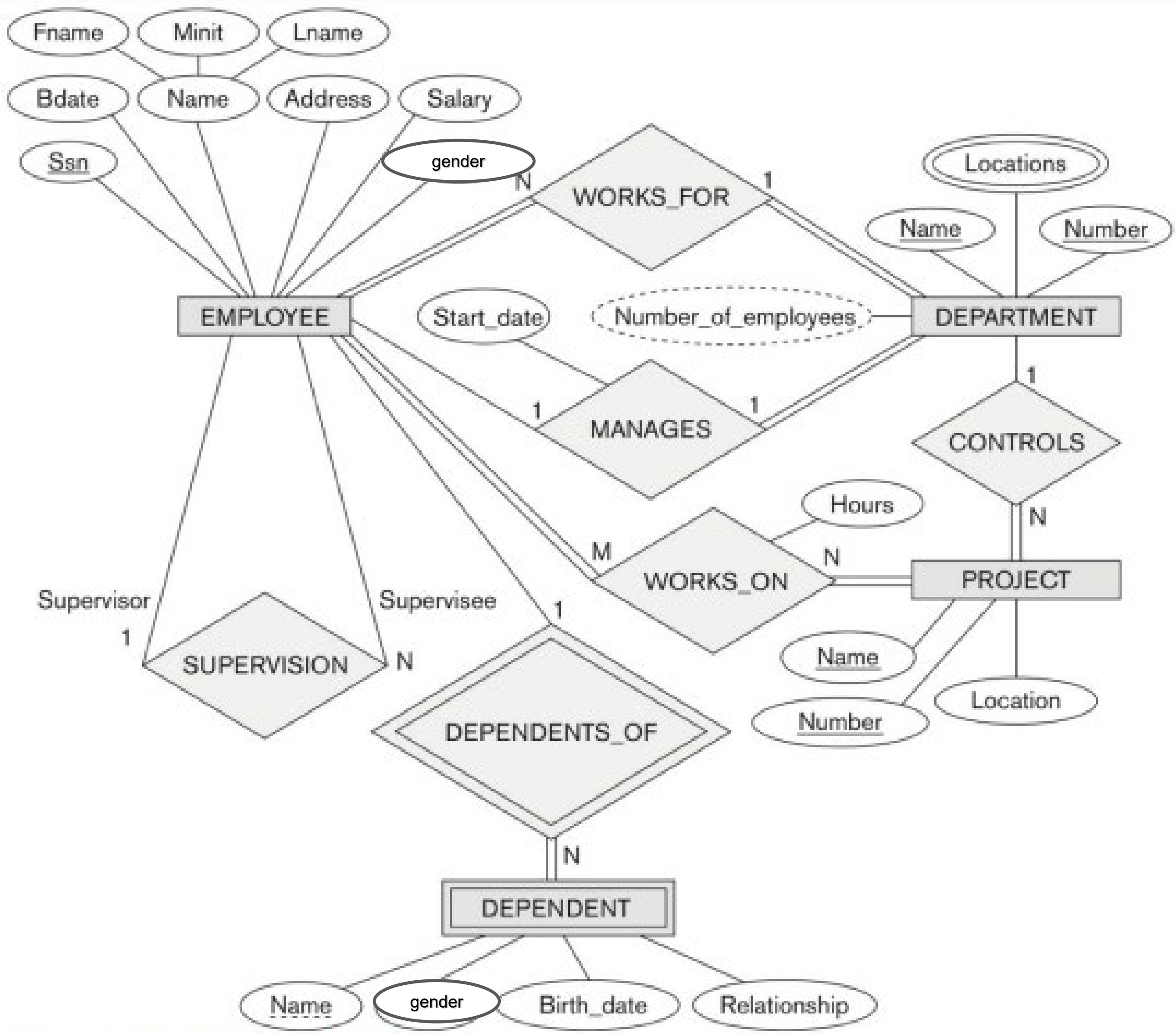
For Chen's notation, one oval inside another.

For crow's foot notation, multi-valued attributes can be represented by enclosing the attribute in {}, e.g., {skills}, {phoneNums}, etc.

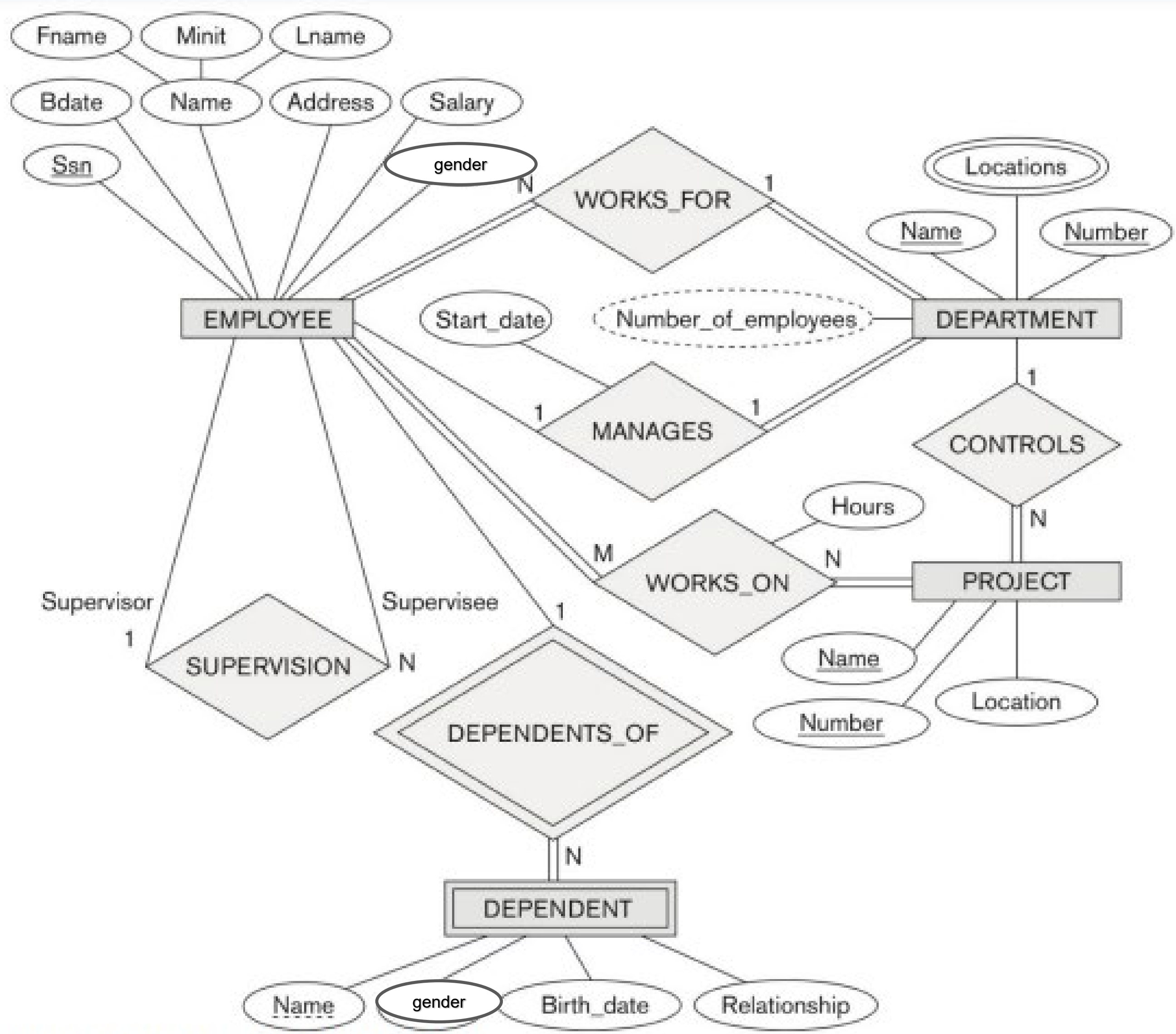
Can you identify …..



menti.com … list all multi-valued attributes?



menti.com … list all derived attributes?



NAMING

- The choice of names for entity types, attributes, relationship types and roles is not always straight-forward.
- Should choose names that convey as much as possible the meanings attached to the constructs.
- These names will subsequently be used as table names and attribute names in database so important to choose good names.
- Remember, should not use sql keywords (order, date, etc.)

QUESTION: What attributes might you have for these entities?

Subject/Module

Person

Exam ... see menti.com

Bank account

Book ... see menti.com

Film

MORE ON ENTITIES: STRONG AND WEAK ENTITIES

Strong: an entity type whose existence **is not** dependent on some other entity type.

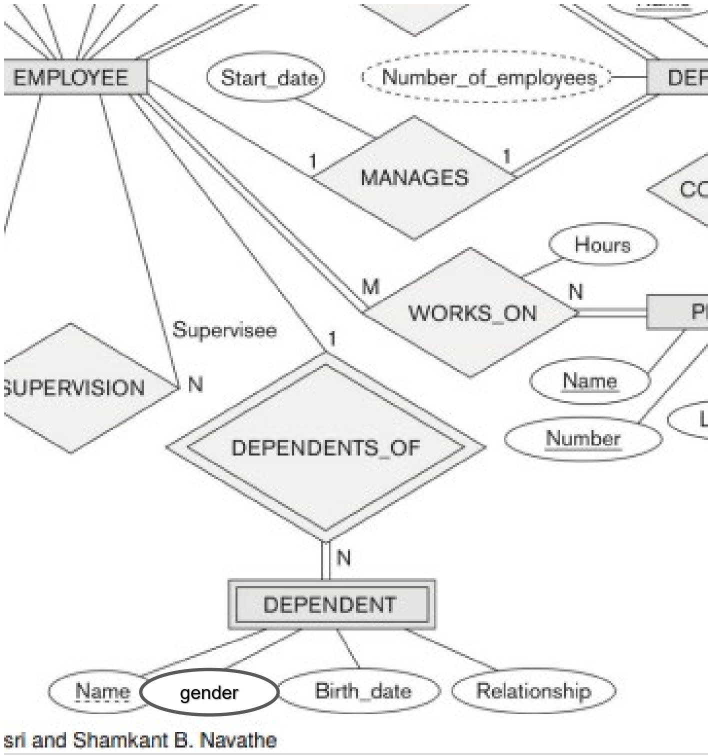
Weak: an entity type whose existence **is** dependent on some other entity type (does not have key attributes of its own)

EXAMPLE:

In the company schema the dependent relation contains data of dependents for each employee.

dependent is a weak entity because two tuples can only be distinguished based on employee SSN.

An alternative would be to have a unique ID for each dependent (e.g. their own SSN) and the dependents could be a strong entity



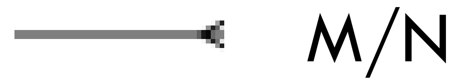
MORE ON RELATIONSHIPS

Whenever an attribute of one entity type refers to another entity type, some relationship exists.

The degree of a relationship type is the number of participating entity types.

Relationship types may have certain constraints.

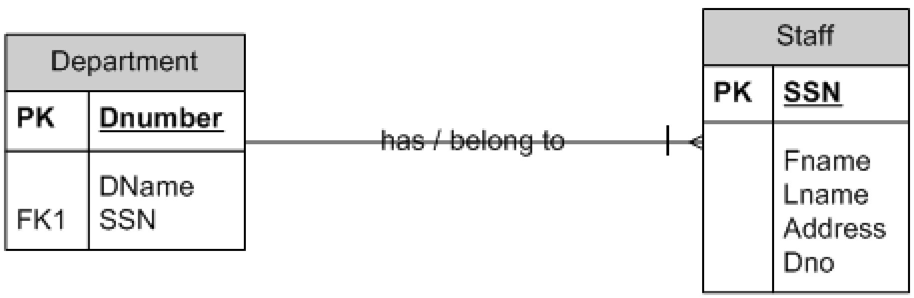
NOTATION

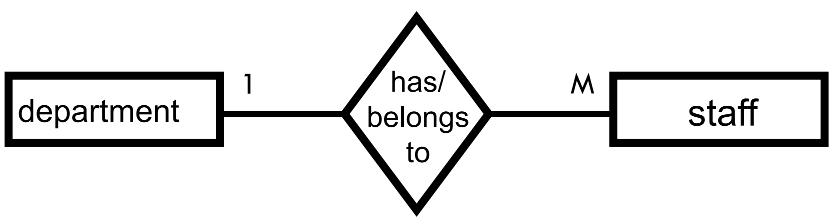


For Chen's notation: A Diamond shape is used to name the relationship. 1 and M/N are used for the "1" and "many" sides respectively.

For Crow's foot notation: The crow foot is used as the representation of "many", and one line is used for the representation of "1".

EXAMPLE: A department has many staff

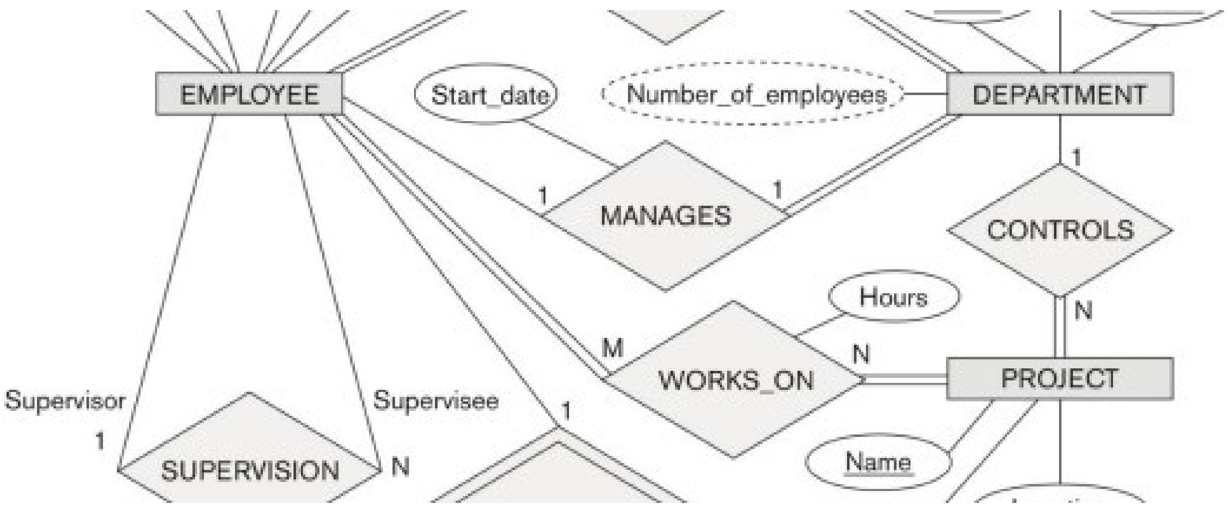




MORE ON RELATIONSHIPS

With Chen's notation, relationships may have attributes

Attributes are drawn "off" the diamond shape of the relationship.



CARDINALITY RATIO

Specifies the number of relationship instances that an entity can participate in.

The possible cardinality ratios for binary relationship types are:

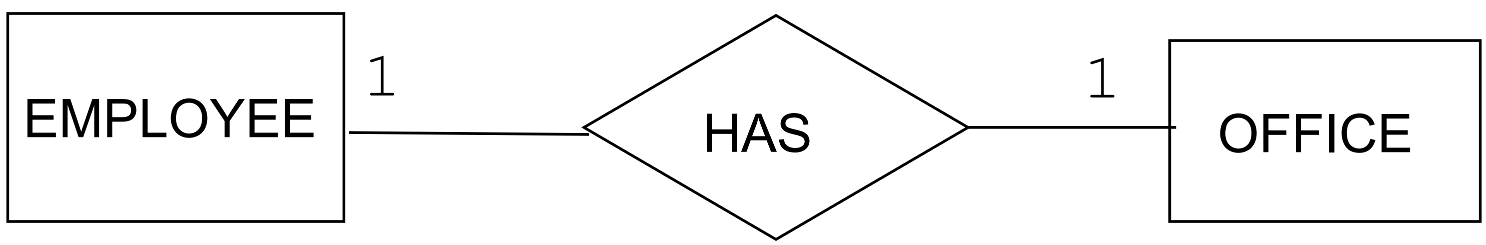
- 1:1, One to One
- 1:N, One to Many
- M:N, Many to Many

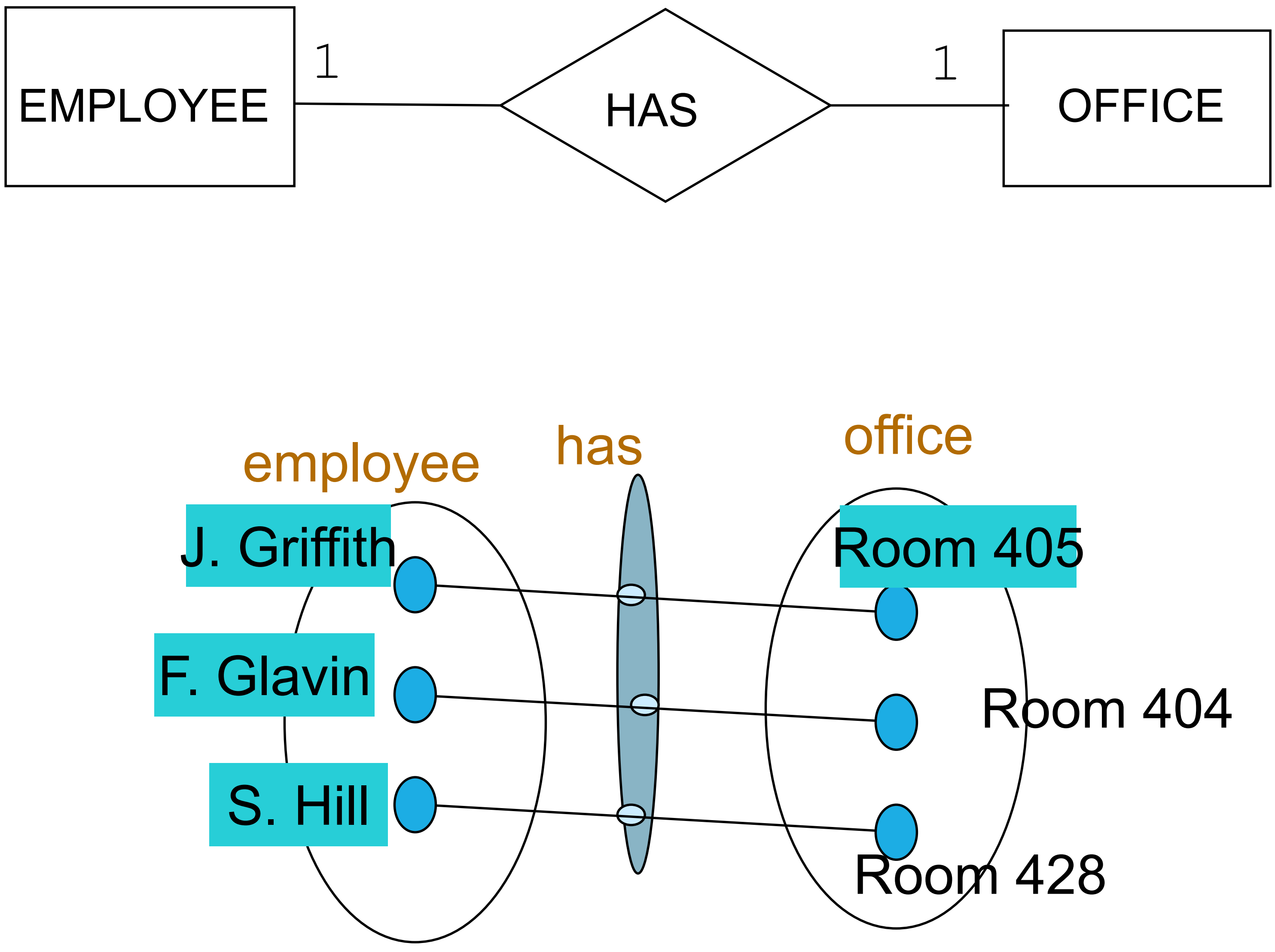
EXAMPLE: 1:1

At most one instance of entity A is associated with one instance of entity B

Example: One employee has one office

Chen notation:

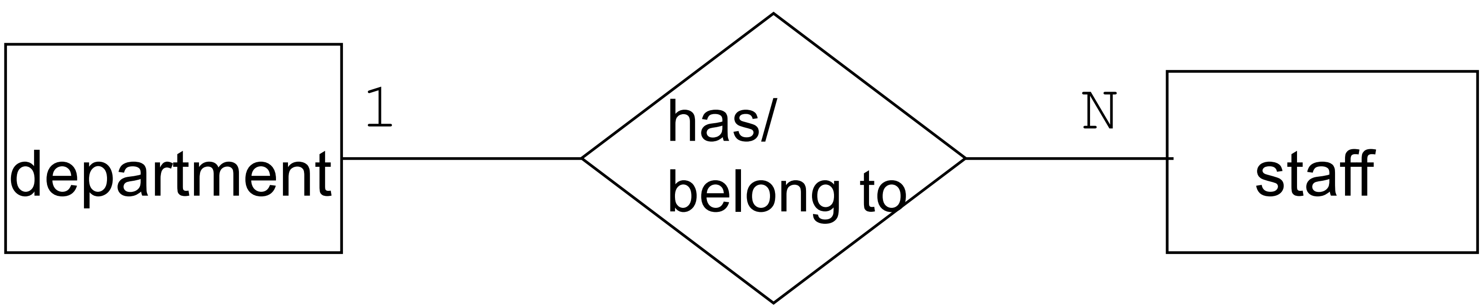



EMPLOYEE
1
HAS
1
OFFICE
employee
has
office
J. Griffith
F. Glavin
S. Hill
Room 405
Room 404
Room 428

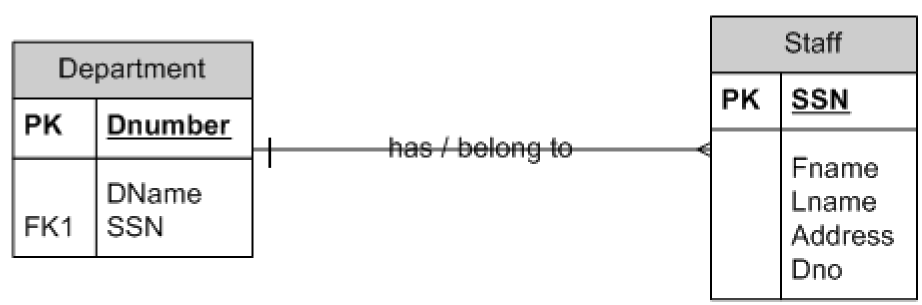
EXAMPLE: 1:N

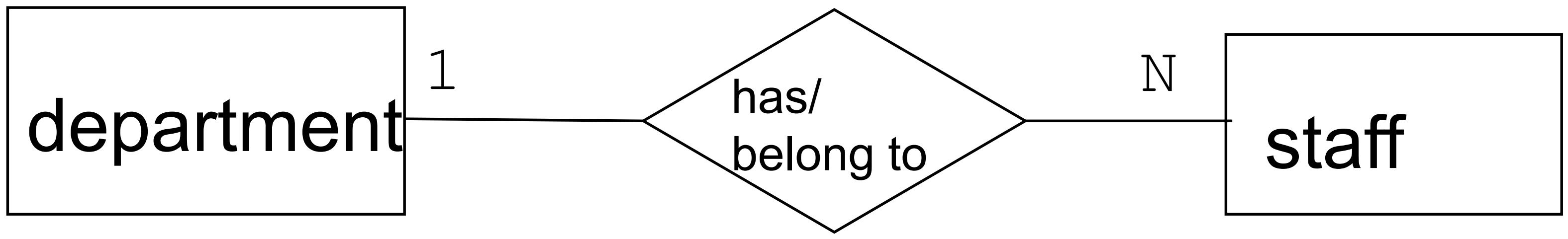
For one instance of entity A, there are 0, 1 or many instances of entity B

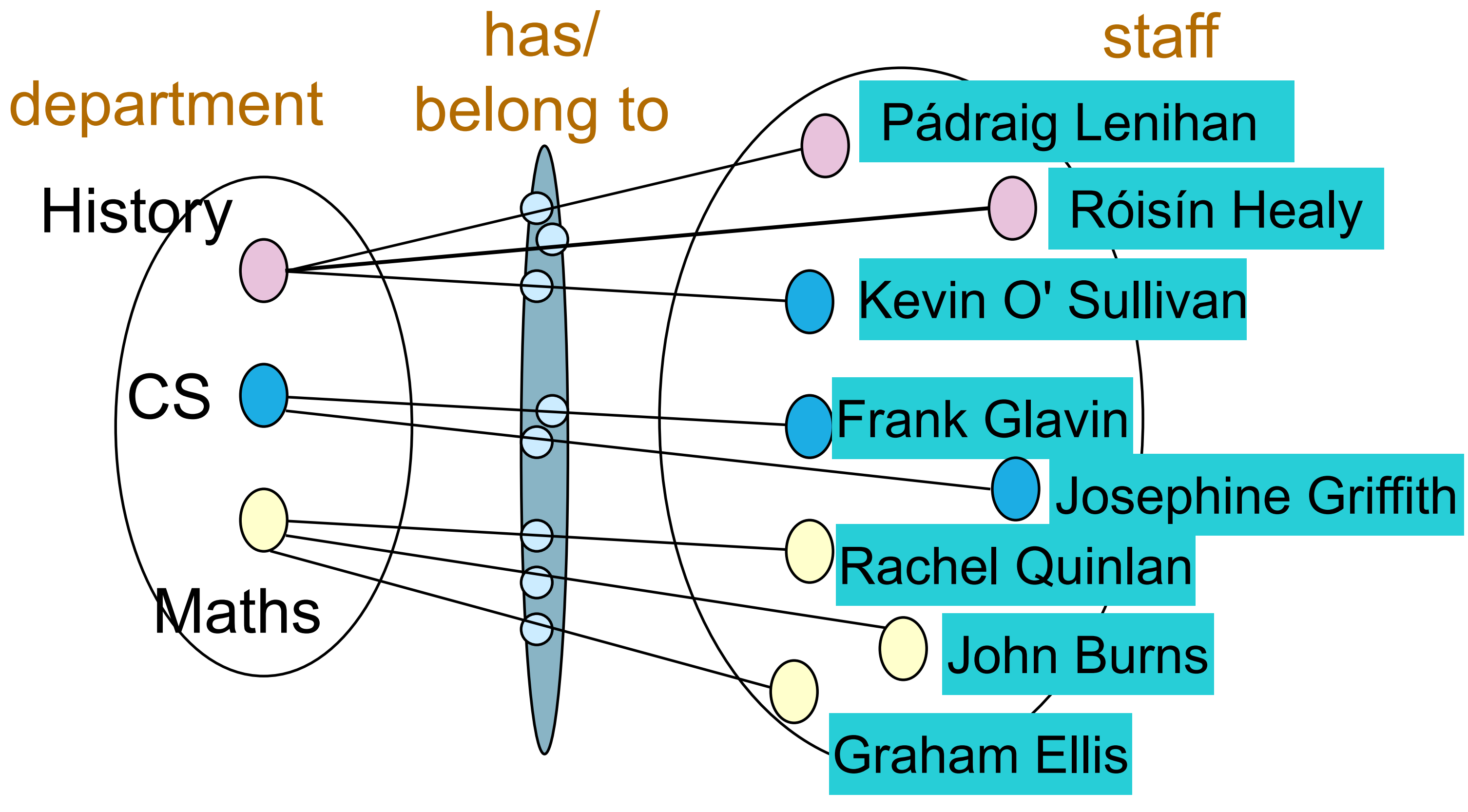
Chen Notation:



Crow's foot notation:



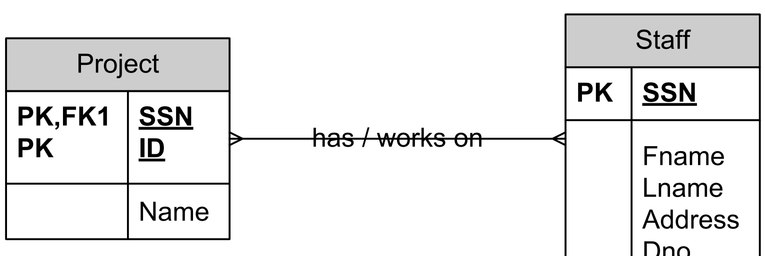

department
1
has/
belong to
N
staff

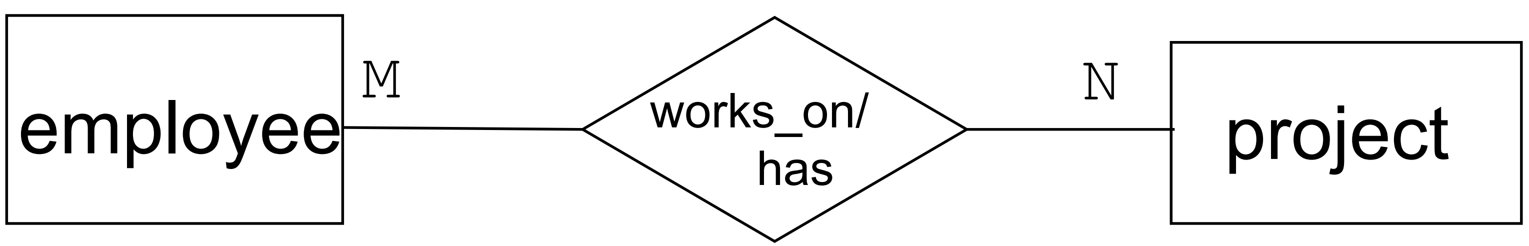

department
has/
belong to
staff
History
CS
Maths
Pádraig Lenihan
Róisín Healy
Kevin O' Sullivan
Frank Glavin
Josephine Griffith
Rachel Quinlan
John Burns
Graham Ellis

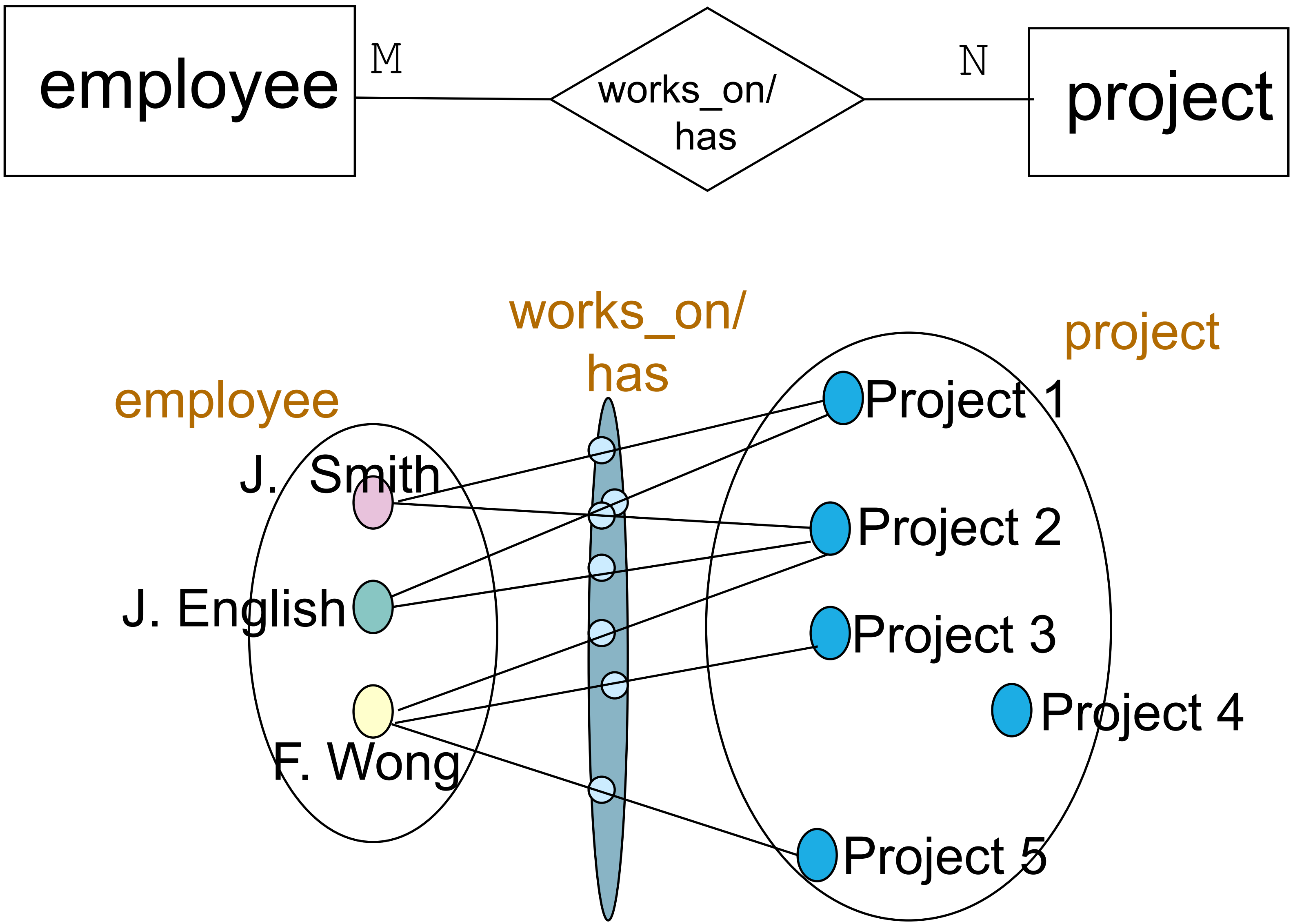
EXAMPLE: M:N

For one instance of entity A, there are 0, 1 or many instances of entity B and

For one instance of entity B, there are 0, 1 or many instances of entity A






employee
M
works_on/
has
N
project
works_on/
has
project
employee
J. Smith
J. English
F. Wong
Project 1
Project 2
Project 3
Project 4
Project 5

ASIDE: Structural constraints on relationships

Often we may know the min and max of the cardinalities

- e.g., limit to number of books which can be borrowed

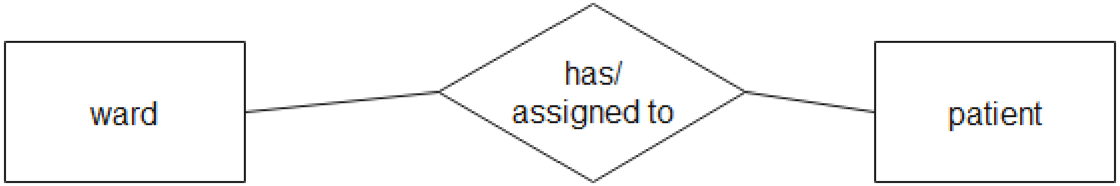
Structural constraints specify a pair of integer numbers *(min, max)* for each entity participating in a relationship

Examples: (0, 1) ,(1,1), (1, N), (1, 7)

We will not model this in our examples

CASS QUESTION –
See menti.com

In a hospital, patients are assigned to wards; wards have patients. What is the cardinality of the relationship?



TOTAL AND PARTIAL PARTICIPATION

Total Participation: all instances of an entity must participate in the relationship, i.e., **every** entity instance in one set **must** be related to an entity instance in the second set via the relationship.

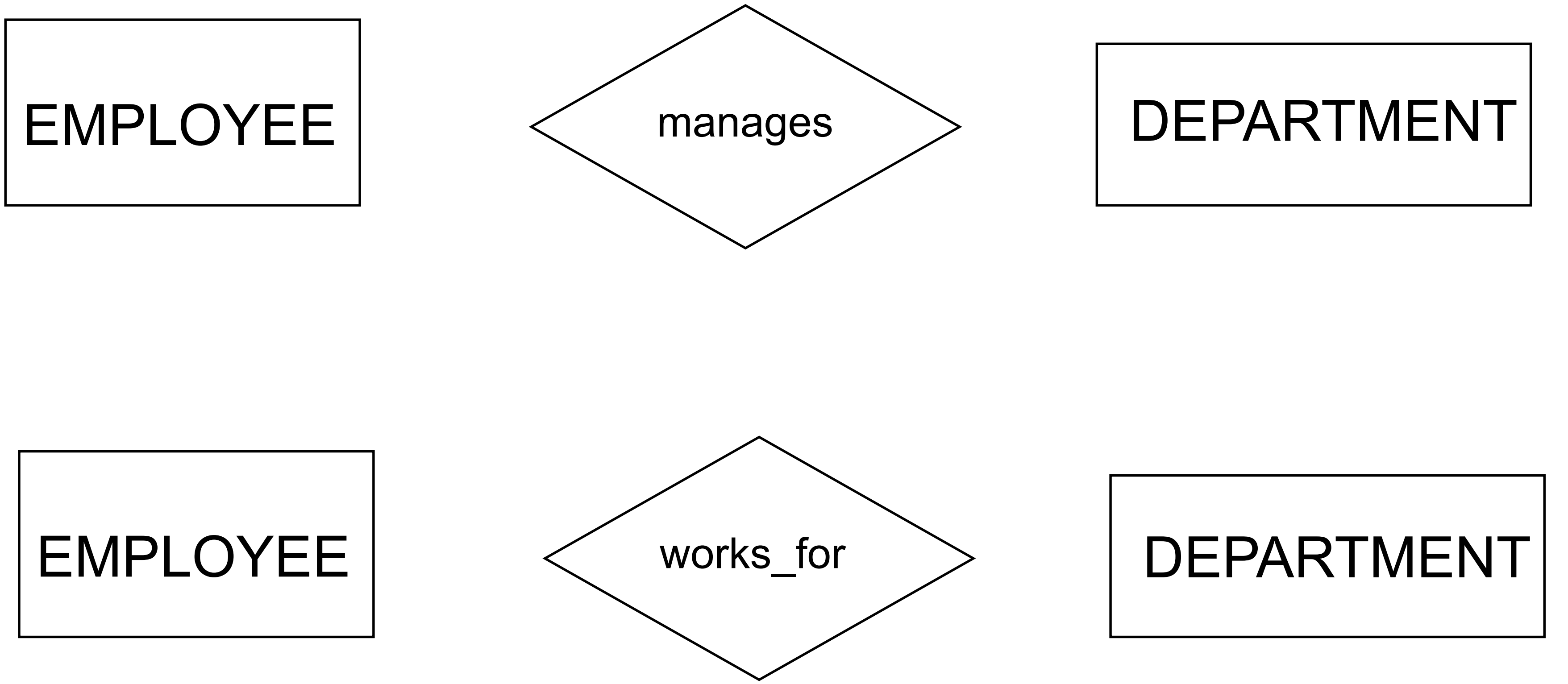
Partial Participation: some subset of instances of an entity will participate in relationship, but not all, i.e., **some** entity instances in one set are related to an entity instance in the second set via the relationship.

NOTATION FOR PARTICIPATION
CHEN'S NOTATION

- Double parallel lines for Total Participation
- Single line for Partial Participation

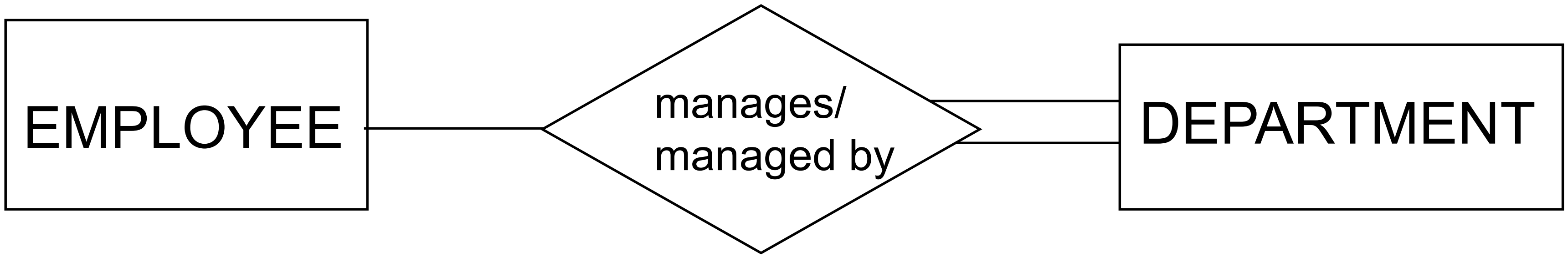
- In both cases, lines drawn from the participating entity to the relationship (the diamond) to indicate the participation of instance from that entity in the relationship

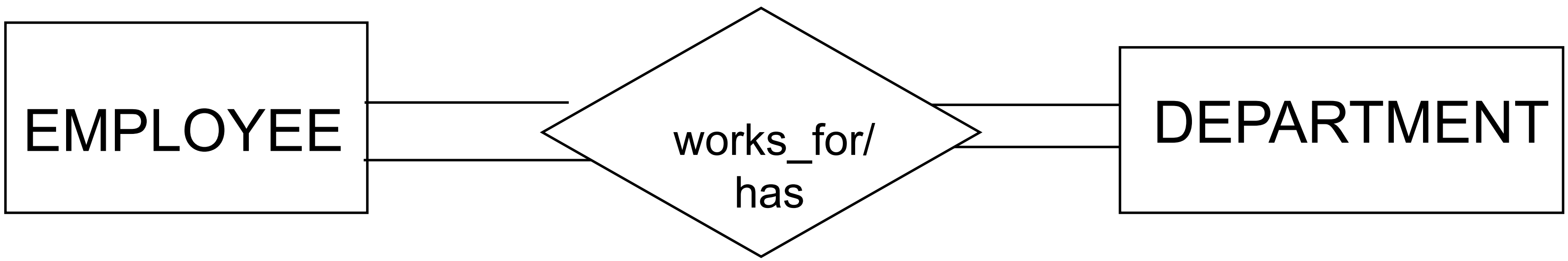
EXAMPLES



EMPLOYEE
manages
DEPARTMENT
EMPLOYEE
works_for
DEPARTMENT

EXAMPLES:
Total and partial participation





NOTATION FOR PARTICIPATION
CROW'S FOOT NOTATION

Use the idea of **Ordinality/Optionality**

- Optionality of 0: if an entity A has partial participation in a relationship to entity B then this means A is associated with 0 or more of the other entity so optionality sign goes beside B.
- Optionality of 1: if an entity A has full participation in a relationship to entity B then this means A is associated with at least 1 or more of B so optionality sign goes beside B.

(and vice versa when looking at participation of B in relationship)

CROW’S FOOT NOTATION

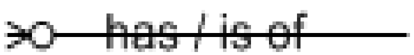
Bar for Optionality of 1: |

Circle (or ‘o’) for Optionality of 0 ○

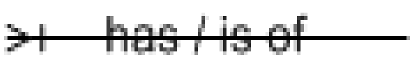
In Crow’s foot notation, there is no diamond so there is a direct relationship line between the entities. On this line:

- The optionality drawn beside entity A refers to how an instance of entity B is related to entity A.
- That is, whether B can be involved partially (0) or not (1)

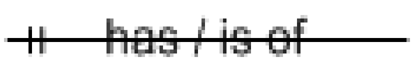
Example in Following *Right to Left* Relationships:



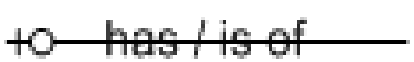
is of 0 or more



is of 1 or more



is of 1 and only 1

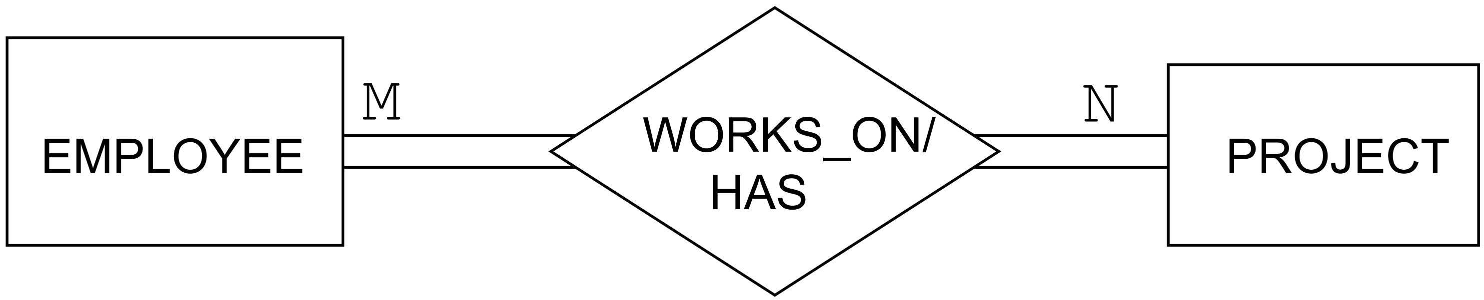


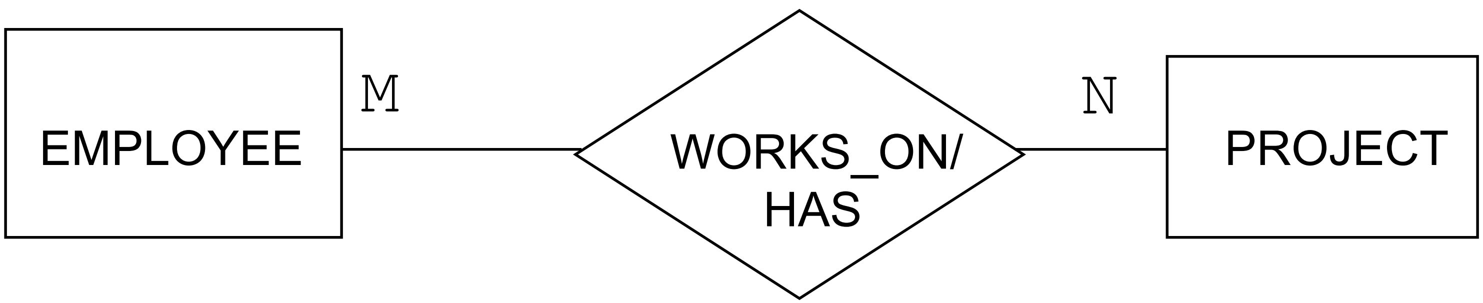
is of 0 or 1

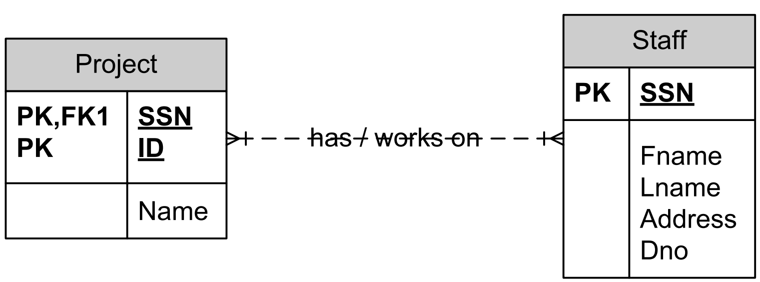
WHICH IS CORRECT FOR THIS RELATIONSHIP?

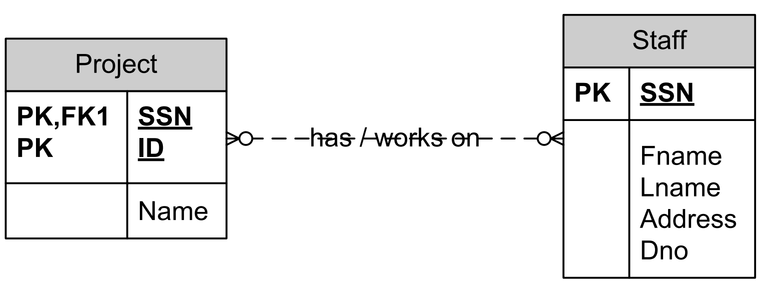
Total or partial participation?

See menti.com

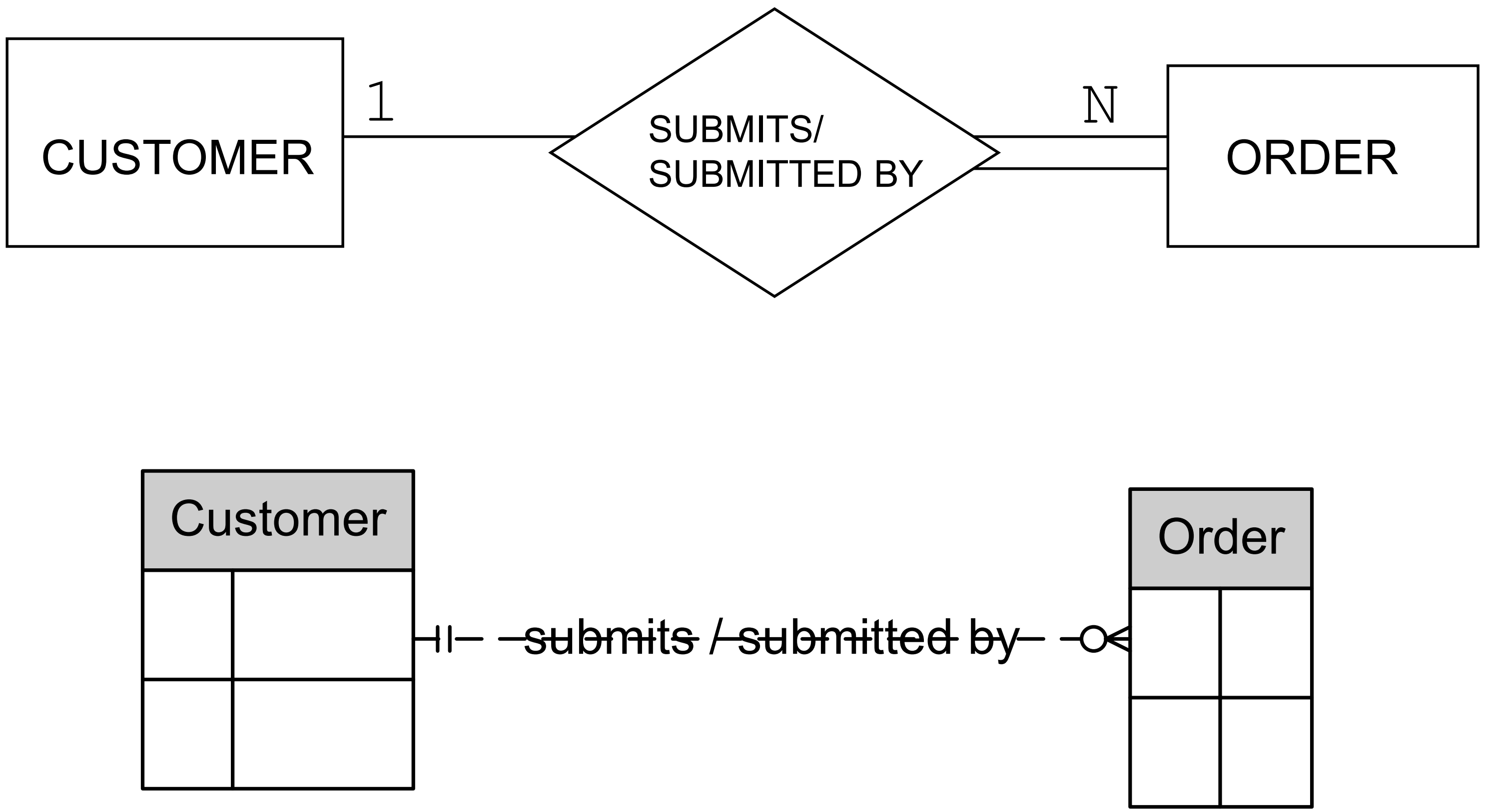






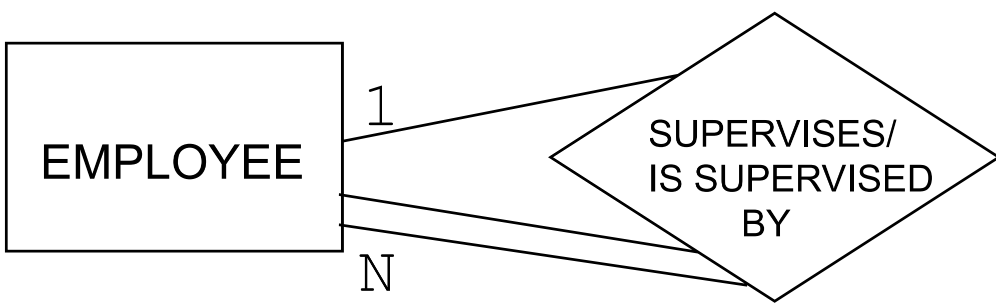


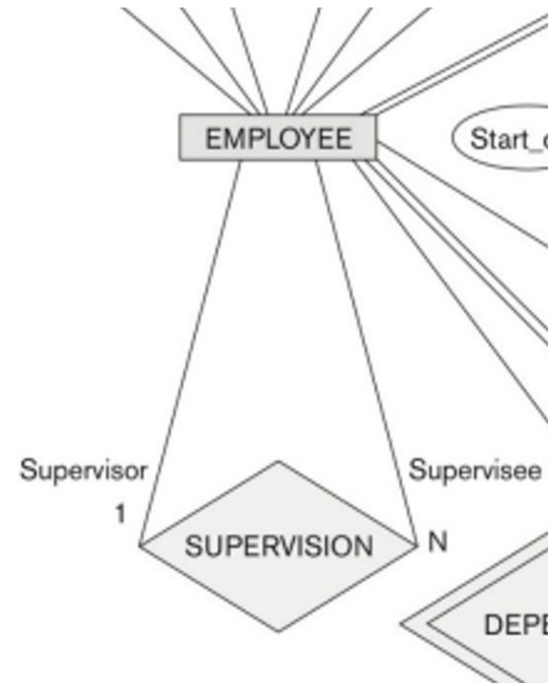
Describe the relationship in words in the following: See menti.com



See menti.com

Describe the relationship in words in the following:
Does it look correct?
How would you fix it?





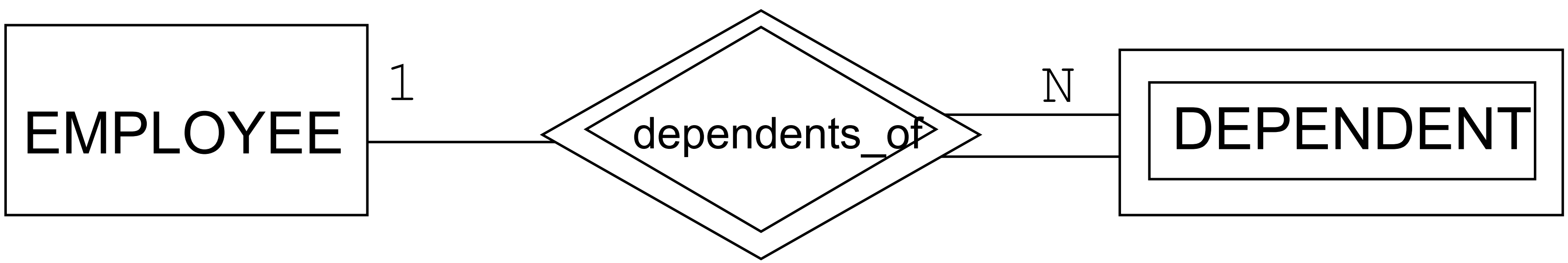
See menti.com
What is the relationship between these entities?

- Cars and people
- Students and library seats
- Students and subjects
- Exams and Locations
- Customers and Bank accounts
- Books and Authors
- Cinema and films/movies

NOTE:

A weak entity type always has a total participation constraint

Need to show the "identifying relationship"

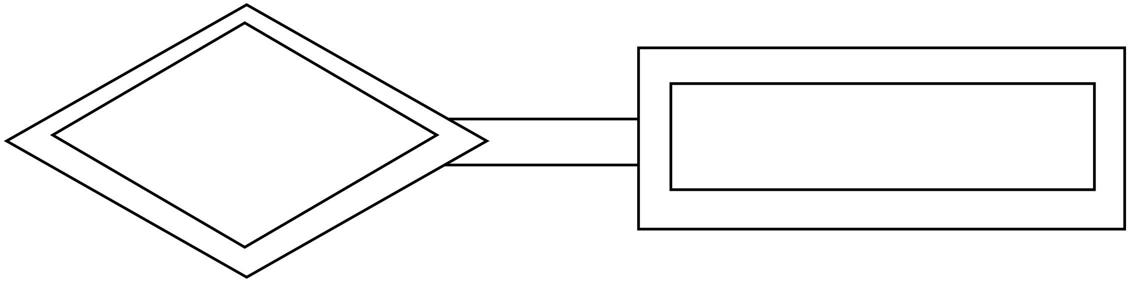


CHEN'S NOTATION FOR WEAK ENTITY

Double rectangle for Entity

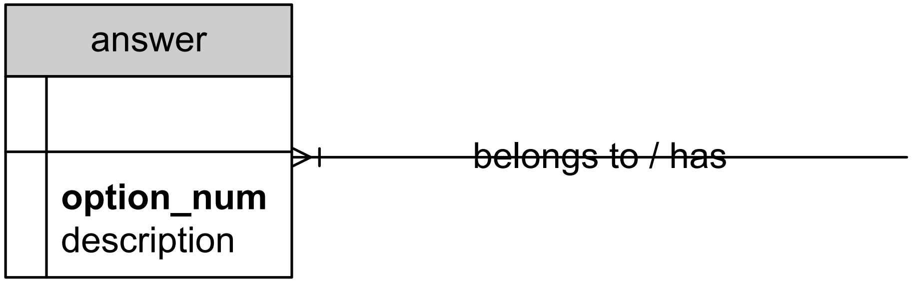
Double diamond for Relationship

Weak entity has full participation in the relationship



CROW'S FOOT NOTATION FOR WEAK ENTITY:

- Can represent the Weak Entity as a normal entity but do not choose any attributes as primary keys.
- For an attribute that partially determines the entity instances, choose the 'required' option
- Represent the relationship between entities with a **solid** line (usually)
- This indicates it is an "identifying" relationship



In general, with entities:

There may be two valid solutions, one with a weak entity and one without.

There is not a huge difficulty if you do not identify weak entities in a solution as long as all entities have **primary attributes.**

May be slightly non-optimal in terms of introducing an additional primary key that is not needed but not a huge problem for us at this level.

Entities or multi-valued attributes?

Sometimes it may not be clear whether something should be modelled as a multi-valued attribute or an Entity.

Both may be equally correct as long as you have represented all the information you were asked to.

When you map either case to tables in a database you might see very little difference between the two approaches.

CLASS EXAMPLE 1

A database is to be created to hold information on lecturers, departments, courses and modules.

Lecturers are associated with only one department. Each lecturer in addition has an associated staff id, title, name, office number and building. Each lecturer teaches a number of modules and a number of lecturers may teach one module.

Each module has an associated unique code (e.g. CT230), name, semester taught, semester examined, ECTs and zero or more prerequisites (which are modules). For example, CT103 and CT102 may be a prerequisite for CT2101.

Each module is part of one or more course instances (e.g. 2BA, 2BCT, 2BFS, 3BP). Each course has an associated name and code.

Each course is controlled by a department, and a department can control a number of courses. Each department has an associated name, and may have a number of different locations; each department has one head of department.

CLASS QUESTION:

Using Chen's notation, create an ER model to accurately model **the above information**. Show all entities, relationships, attributes, cardinalities, and total and partial participations. State any assumptions you make.

STEPS:

Identify entities.

Identify relationships between entities.

Draw entities and relationships.

Add attributes to entities (and relationships if appropriate).

Add cardinalities to relationships.

Add participation constraints (total or partial) to relationships.

Check all entities have primary keys identified.

MAPPING ER MODELS TO TABLES IN THE RELATIONAL MODEL

Once you have your ER diagram you now need to convert this into a set of tables so that you can implement this in a relational model (e.g. as MySQL tables using `CREATE TABLE` commands)

This stage is called **Mapping ER Models to Tables in the Relational Model** and it specifies a set of rules that must be followed in a certain order.

The rules specified here are based on Chen's notation.

STEPS … Mapping ER models to tables in the relational model

1. For each entity create a table R that includes all the **simple** attributes of the entity.

2. For strong entities, choose a key attribute as primary key of the table.

STEPS … Mapping ER models to tables in the relational model

3. For weak entities R, include as foreign key attributes of R the primary key attributes of the table that corresponds to the owner. The primary key of R is a combination of the primary key of owner and the partial key of the weak entity type.

> The relationship of the weak and strong entity is generally taken care of by this step

STEPS *CTD*.... mapping ER models to tables in the relational model

4. **For each binary 1:1 relationship**, identify entities S and T that participate in relation.

- If applicable, choose the entity that has total participation in the relation. Include as foreign key in this table the primary key of other relation. Include any attributes of the relationship as attributes of chosen table.
- If both entities have total participation in the relationship, you can choose either for the foreign key and proceed as above or can map 2 entities, and their associated attributes and relationship attributes into 1 table.

STEPS *CTD*.... mapping ER models to tables in the relational model

5. **For each binary 1:N relationship**, identify the table S that represents the N-side and T the table that represents the 1-side.

- Include as a foreign key in S the primary key of table T such that each entity on the N-side is related to at most one entity instance on the 1-side. Include any attributes of the relationship as attributes of S.
- For recursive 1:N relationships, choose the primary key of the table and include it as a foreign key in the same table (with a different name).

STEPS *CTD*.... mapping ER models to tables in the relational model

6. **For each M:N relationship**, create a new table S to represent the relationship.

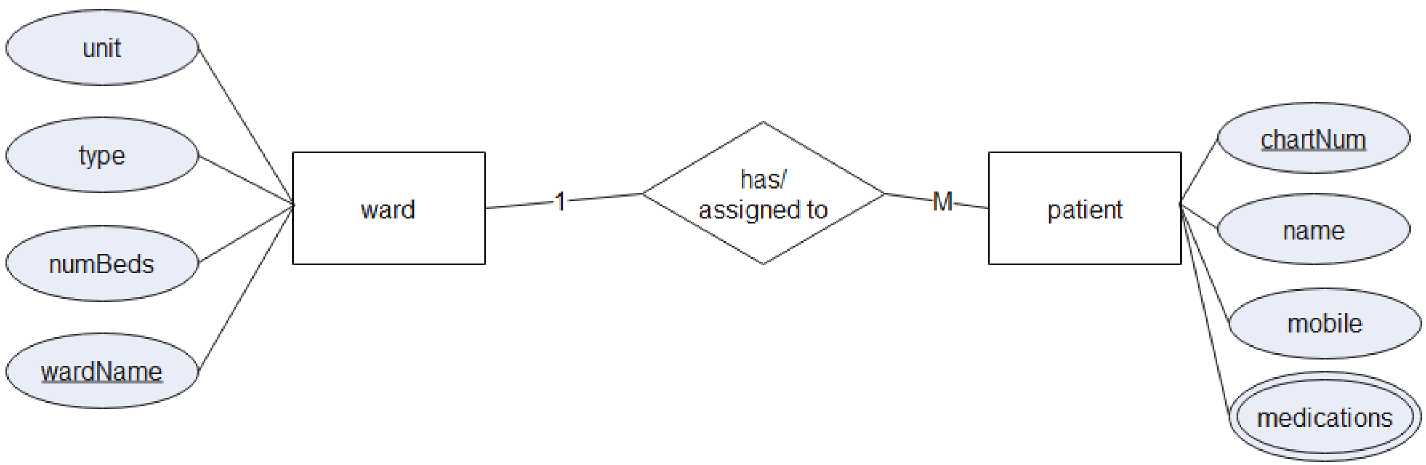
- Include as foreign key attributes in S the primary keys of the tables that represent the participating entity types – their combination will form the primary key of S. Also include in S any attributes of the relationship.
- For a recursive M:N relationship, both foreign keys come from the same table (give different name to each) and become the new primary key.

STEPS *CTD*…. mapping ER models to tables in the relational model

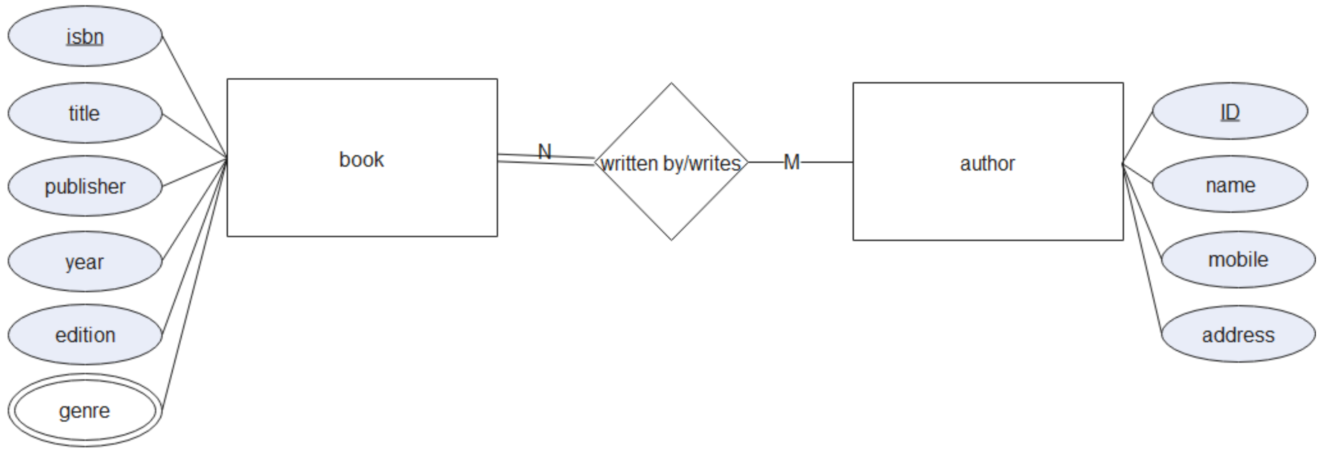
7. **For each multi-valued attribute A of an entity S,** create a new table R. R will include:

- an attribute corresponding to A,
- primary key of S which will be a foreign key in table R. Call this K.
- primary key of R is a combination of A and K

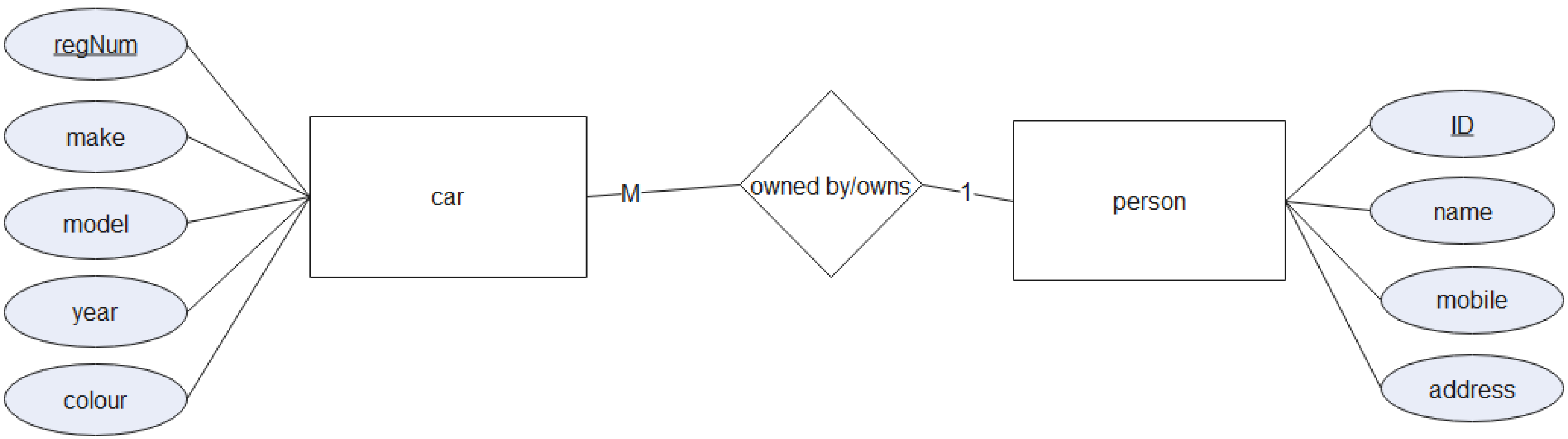
Map each of the following to tables in the relational model:
wards and patients



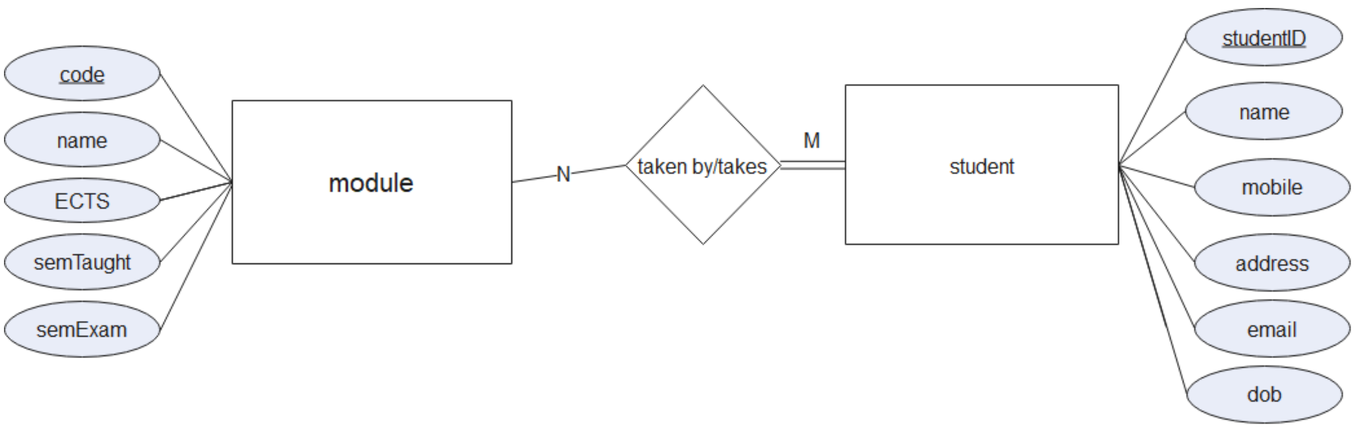
Map each of the following to tables in the relational model:
authors and books



Map each of the following to tables in the relational model:
cars and people



Map each of the following to tables in the relational model:
modules and students



CLASS WORK: Map the University model created (Example 1) to tables in the relational model

PROBLEM SHEET 4

An Irish holiday home rental company wishes to create an online database system to maintain information on home owners who own holiday houses which the rental company rents on their behalf; customers who rent the holiday homes, and the rental agreements. The data which should be stored is as follows:

Details stored on holiday houses are: a unique ID for each house, the address of the house (town, county and Eircode), the number of bedrooms and bathrooms in the house and the maximum number of people the house will accommodate. Two price details should be stored: low-season price per night and high-season/weekend price per night. In addition a short description of the house amenities and surrounding amenities should be stored.

Each house is owned by one home owner. A home owner may own many houses. Details stored on the home owners are: a unique id, a username and password to login to the system, their name, address and telephone number and their email address.

Customers can book one or more houses and a house can be booked many times. Details held on customers are: unique ID, customer name, address, email address and phone number.

Details held on a booking are the dates the booking begins and ends, and the number of people wishing to stay in the house as part of the booking. Any entered bookings must be confirmed by a company employee (via phone or email). When the confirmation takes place, data should be stored to indicate that the confirmation has taken place and to indicate the amount of money paid as a deposit. This database does not currently hold any information on the check-in process and the payment of the balance due.

SUMMARY:

Important to Know:

- Basic definitions of entity, relationship, attribute (and different types), cardinality and participation for Chen and Crow's foot notation.
- Create ER Model (in Chen's notation)
- Map from ER model in Chen notation to set of tables with associated primary and foreign keys.

Common Errors:

- Missing Primary Keys for Entities.
- Missing cardinalities in Relationships.
- Only mapping entities to tables; not mapping relationships or multi-valued attributes.

Returning to

SQL DML SELECT STATEMENT

Join and Union Queries

CT230

Database

Systems

RECALL EXAMPLE 18:

Version 1: List the details (name and birth date) of the children of the employee with SSN 333445555

Version 2: List the details (name and birth date) of the children of Franklin T Wong?

Now consider a 3rd version:

Version 3: List the details (name, birth date and address) of the children of Franklin T Wong (assuming the dependent's address is Franklin Wong's address)

RECALL sub-query solution to version 2:

List the details (name and birth date) of the children of Franklin T Wong?

```
SELECT  dependent_name, bdate
FROM    dependent
WHERE   relationship != 'spouse'
        AND essn =
        (SELECT  ssn
        FROM     employee
        WHERE  fname = 'Franklin' AND minit = 'T' AND lname = 'Wong')
```

dependent_name	bdate
Alice	2010-04-05
Theodore	2014-10-25

CAN WE MODIFY THIS TO GET THE SOLUTION TO VERSION 3?

List the details (name, birth date and address) of the children of Franklin T Wong (assuming the dependent's address is Franklin Wong's address)

```
SELECT  dependent_name, bdate
FROM    dependent
WHERE   relationship != 'spouse'
        AND essn =
        (SELECT  ssn
        FROM     employee
        WHERE  fname = 'Franklin' AND minit = 'T' AND lname = 'Wong')
```

dependent_name	bdate
Alice	2010-04-05
Theodore	2014-10-25

No – because we need information from two tables –we need to use a *join* to join or *combine* the two tables so that the information from both is accessible and can be displayed as the output

JOINS

Joins combine multiple tables in to one table. This new (temporary) table is then queried to return results so we can return values from any of the tables which were joined.

Tables are joined by specifying links (or joins) across attributes in the tables.

Joins are carried out on 2 tables at a time but many tables can be joined, i.e., a third table can be joined to the table that results from joining two tables.

SPECIFYING JOINS

1. In SQL must specify all the tables which are part of join in the FROM clause
2. There are many different types of joins – all may not be supported in the DBMS you are using – we will mostly use an *inner join* which will always be supported.
3. Must then specify the join condition: for an inner join the condition is *foreign_key = primary_key/candidate_key*.
4. The join condition can be specified in the FROM or WHERE clause.

INNER JOINING TABLES:

The result of an inner join operation between two tables:

$R(A_1, A_2, \ldots, A_n)$ and

$S\ (B_1, B_2, \ldots, B_m)$

is a table $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$ where:

Q has one tuple for each combination of tuples (one from R and S) whenever the combination satisfies the join condition – the join will retrieve ALL attributes in each table

CONSIDER:
INNER JOIN CONDITION FOR employee AND dependent TABLES

Join condition: `ssn = essn`

Full query retrieving all employees and their dependents (when they have dependents):

```
SELECT *
FROM    employee INNER JOIN dependent
        ON ssn = essn;
```

Result from joining `employee` and `dependent`:

fname	minit	lname	ssn	bdate	address	gender	salary	superssn	dno	essn	dependent_name	gender	bdate	relationship
John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	5	123456789	Alice	Woman	2008-12-30	Daughter
John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	5	123456789	Elizabeth	Woman	1976-05-05	Spouse
John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	5	123456789	Michael	Man	2011-01-04	Son
Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5	333445555	Alice	Woman	2010-04-05	Daughter
Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5	333445555	Joy	Woman	1981-05-03	Spouse
Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5	333445555	Theodore	Man	2014-10-25	Son
Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	4	987654321	Abner	Woman	1992-02-28	Spouse

EXAMPLE 18 VERSION 3 JOIN SOLUTION

List the details (name, birth date and address) of the children of Franklin T Wong

```
SELECT  dependent_name, dependent.bdate, address
FROM    employee INNER JOIN dependent ON
        ssn = essn
WHERE   relationship != ‘spouse’
        AND  fname = 'Franklin’
        AND  minit = 'T’
        AND  lname = 'Wong’;
```

dependent_name	bdate	address
Alice	2010-04-05	638 Voss, Houston, TX
Theodore	2014-10-25	638 Voss, Houston, TX

NOTE:

When attributes with the same name, but from different tables, are used in a join query, you need to specify the table name to avoid ambiguity with respect to the attribute names.

Example: bdate in employee and dependent relations.

Can refer to both of these unambiguously as:

```
employee.bdate
dependent.bdate
```

If you do not do this, the DBMS does not know which one you are referring to and gives an error:

Error in query (1052): Column 'bdate' in field list is ambiguous

EXAMPLE 39: Using an inner join, retrieve the names and addresses of all employees who work for the administration department

```
SELECT fname, lname, address
FROM   ???
WHERE  dname = 'administration';
```

CONSIDER THE INNER JOIN CONDITION FOR `employee` AND `department` USING DEPARTMENT NUMBER

Join condition is: `dno = dnumber`

Full query retrieving all employees and their departments:

```
SELECT *
FROM    employee INNER JOIN department
        ON dno = dnumber;
```

fname	minit	lname	ssn	bdate	address	gender	salary	superssn	dno	dnumber	dname	mgrssn	mgrstartdate
John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	5	5	Research	333445555	2018-05-22
Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5	5	Research	333445555	2018-05-22
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	Woman	44183	333445555	5	5	Research	333445555	2018-05-22
Ramesh	K	Narayan	666884444	1995-09-15	975 Fire Oak, Humble, TX	Man	60000	333445555	5	5	Research	333445555	2018-05-22
James	E	Borg	888665555	1997-11-10	450 Stone, Houston, TX	Man	94199	*NULL*	1	1	Headquarters	888665555	2019-06-19
Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	4	4	Administration	987654321	2015-01-01
Ahmad	V	Jabbar	987987987	2000-03-29	980 Dallas, Houston, TX	Man	44183	987654321	4	4	Administration	987654321	2015-01-01
Alicia	J	Zelaya	999887777	1998-07-19	3321 Castle, Spring, TX	Non-binary	44183	987654321	4	4	Administration	987654321	2015-01-01

EXAMPLE 39: Using a join, retrieve the names and addresses of all employees who work for the administration department

```
SELECT fname, lname, address
FROM   employee INNER JOIN department
       ON  employee.dno = department.dnumber
WHERE  dname = 'administration';
```

+ Options

fname	lname	address
Jennifer	Wallace	291 Berry, Bellaire, TX
Ahmad	Jabbar	980 Dallas, Houston, TX
Alicia	Zelaya	3321 Castle, Spring, TX

Class Question: Can this be done with a sub-query?

Class Question: Can this be done with a sub-query? (EXAMPLE 39: Retrieve the names and addresses of all employees who work for the administration department)

EXAMPLE 40: Retrieve the names and addresses of all employees who work for the administration department and the ssn of the manager of the administration department

```
SELECT   fname, lname, address, mgrssn
FROM     employee INNER JOIN department
         ON employee.dno = department.dnumber
WHERE    dname = 'administration';
```

fname	lname	address	mgrssn
Jennifer	Wallace	291 Berry, Bellaire, TX	987654321
Ahmad	Jabbar	980 Dallas, Houston, TX	987654321
Alicia	Zelaya	3321 Castle, Spring, TX	987654321

IMPLICIT AND EXPLICIT JOINS

The join condition can be specified implicitly or explicitly as follows:

- An explicit join is specified in the FROM clause where the tables to be joined are listed. The keyword INNER JOIN is used for inner joins and the **join condition** is listed also using keyword ON
- An implicit join is specified in the WHERE clause without using the keyword ON. It is referred to as a **join condition.** The tables must be listed in the FROM clause, separated by commas. Other conditions can also be specified in the WHERE clause as well as the join condition.

IMPLICIT JOIN CONDITION IN `WHERE` CLAUSE:

- No additional syntax to learn.
- All tables involved *MUST* be listed in FROM clause.
- Condition to join tables is contained in the WHERE clause. If there are other conditions, the join condition is appended on with AND
- This is an **INNER JOIN**: all rows from both tables will be returned whenever there is a match between the attributes in the join condition

EXPLICIT JOIN CONDITION IN FROM CLAUSE

Syntax for joining 2 tables:

```
SELECT [DISTINCT] <attribute list>
FROM   <table>
        [INNER/LEFT/RIGHT] JOIN <table>
        ON <join condition>
WHERE  <condition>
```

* Will mostly use INNER JOIN

EXAMPLE 18 AGAIN …. USING AN IMPLICT JOIN

List the details (name, birth date and address) of the children of Franklin T Wong

EXAMPLE 39 again: Retrieve the names and addresses of all employees who work for the administration department (using an implicit join)

```
SELECT fname, lname, address

FROM   ??

WHERE  dname = 'administration';
```

Syntax of **explicit join** with 3 tables

```
SELECT [DISTINCT] <attribute list>
FROM   (<table>
        [INNER/LEFT/RIGHT] JOIN <table>
        ON <join condition>)
        [INNER/LEFT/RIGHT] JOIN <table>
        ON <join condition>
WHERE <condition>
```

Syntax of **implicit join** with 3 tables

```
SELECT [DISTINCT] <attribute list>
FROM  <table>,<table>,<table>
WHERE <join condition> AND
      <join condition> AND
      <condition>
```

Syntax of **explicit join** with 4 tables

```
SELECT [DISTINCT] <attribute list>
FROM   ((<table>
         [INNER/LEFT/RIGHT] JOIN <table>
         ON <join condition>)
         [INNER/LEFT/RIGHT] JOIN <table>
         ON <join condition>)
         [INNER/LEFT/RIGHT] JOIN <table>
         ON <join condition>
WHERE <condition>
```

Syntax of **implicit join** with 4 tables

```
SELECT [DISTINCT] <attribute list>
FROM  <table>,<table>,<table>,<table>
WHERE <join condition> AND
      <join condition> AND
      <join condition> AND
      <condition>
```

EXAMPLE 41

For every project located in Stafford, list the project number, the controlling department name, and the department manager's surname, address and birth date.

EXAMPLE 41

```
SELECT pnumber, dname, lname, address, bdate
FROM   project INNER JOIN department
       ON project.dnum = department.dnumber
       INNER JOIN employee
       ON department.mgrssn = employee.ssn
WHERE  plocation = ‘stafford’;
```

pnumber	dname	lname	address	bdate
10	Administration	Wallace	291 Berry, Bellaire, TX	1991-06-20
30	Administration	Wallace	291 Berry, Bellaire, TX	1991-06-20

CLASS QUESTION:

> Re-write solution to example 41 using implicit joins?
> Can we re-write this using sub-queries?

DIFFERENT TYPES OF JOINS:

- Inner Join is the default when using Implicit Join
- An `INNER JOIN` includes the tuples from the first (left) of the two tables **only** when they satisfy the join condition and tuples from the second (right) table are **only** included when they also satisfy the join condition
- For explicit joins, should explicitly state the join used:

For example joining employee and department on ssn and mgrssn:

```
SELECT *
FROM     employee INNER JOIN department ON
         employee.ssn = department.mgrssn;
```

LEFT JOINS

Left (outer) joins include all of the tuples from the first (left) of two tables – when they satisfy the join condition and <u>even when they don't.</u> Tuples from the second (right) table are only included when they satisfy the join condition. Example:

```
SELECT *
FROM     employee LEFT JOIN department ON
         employee.ssn = department.mgrssn;
```

fname	minit	lname	ssn	bdate	address	gender	salary	superssn	dno	dnumber	dname	mgrssn	mgrstartdate
James	E	Borg	888665555	1997-11-10	450 Stone, Houston, TX	Man	94199	*NULL*	1	1	Headquarters	888665555	2019-06-19
Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	4	4	Administration	987654321	2015-01-01
Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5	5	Research	333445555	2018-05-22
John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	5	*NULL*	*NULL*	*NULL*	*NULL*
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	Woman	44183	333445555	5	*NULL*	*NULL*	*NULL*	*NULL*
Ramesh	K	Narayan	666884444	1995-09-15	975 Fire Oak, Humble, TX	Man	60000	333445555	5	*NULL*	*NULL*	*NULL*	*NULL*
Ahmad	V	Jabbar	987987987	2000-03-29	980 Dallas, Houston, TX	Man	44183	987654321	4	*NULL*	*NULL*	*NULL*	*NULL*
Alicia	J	Zelaya	999887777	1998-07-19	3321 Castle, Spring, TX	Non-binary	44183	987654321	4	*NULL*	*NULL*	*NULL*	*NULL*

RIGHT JOINS

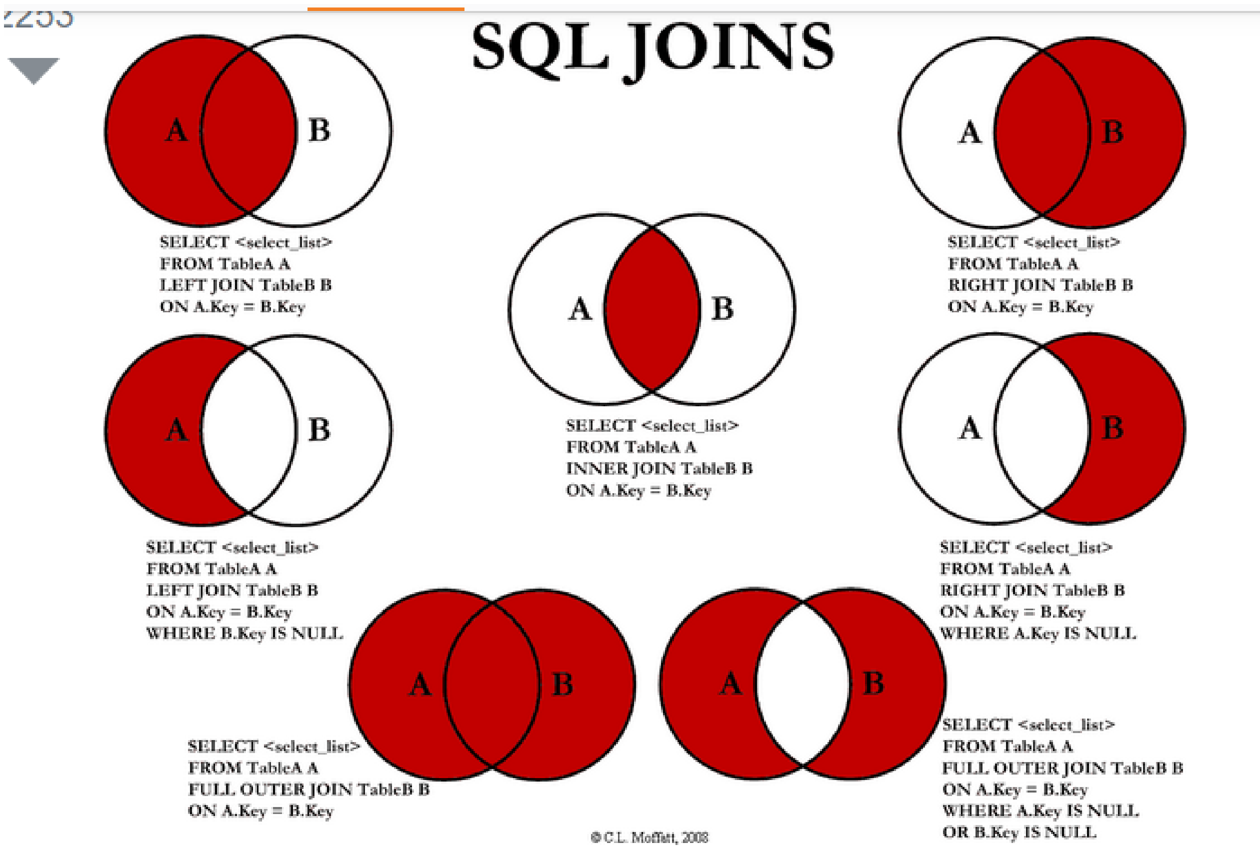
Right outer joins include **all** of the tuples from the second (right) of two tables, even if there are no matching values for records in the first (left) table. Tuples from the first (left) table are included **only** if they satisfy the join condition. Example:

```
SELECT    *
FROM      employee RIGHT JOIN department ON
          employee.ssn = department.mgrssn;
```

fname	minit	lname	ssn	bdate	address	gender	salary	superssn	dno	dnumber	dname	mgrssn	mgrstartdate
James	E	Borg	888665555	1997-11-10	450 Stone, Houston, TX	Man	94199	*NULL*	1	1	Headquarters	888665555	2019-06-19
Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	4	4	Administration	987654321	2015-01-01
Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5	5	Research	333445555	2018-05-22

Graphical representation of different types of joins (C.L. Moffat, 2008)

In MySQL only INNER, LEFT and RIGHT joins are supported



EXAMPLE 42: What is the difference in the output produced using INNER, LEFT and RIGHT joins in the following?

```
SELECT  *
FROM    employee [INNER/LEFT/RIGHT] JOIN dependent
        ON employee.ssn = dependent.essn;
```

SELF-JOINS AND ALIASES

A self-join is a normal SQL join that joins a table to itself.

This is accomplished by using aliases to give each "instance" of the table a separate name – the keyword AS is used.

EXAMPLE 43: For each employee, retrieve the employee's name and the name of the employee's supervisor

Consider:

1. How to write the query if asked for the employee's name and supervisor's SSN?

2. What should output look like? e.g., for John Smith:

fname	lname	fname	lname
John	Smith	Franklin	Wong

First consider joining employee to itself …

Need two “copies” or instances of table employee…

Call them E (for employee) and S (for supervisor)

```
SELECT *
FROM    employee AS e, employee AS s
WHERE   e.superssn = s.ssn;
```

```
SELECT *
FROM    employee AS e INNER JOIN employee AS s
        ON  e.superssn = s.ssn;
```

fname	minit	lname	ssn	bdate	address	gender	salary	superssn	dno	fname	minit	lname	ssn	bdate	address	gender	salary	superssn	dno
John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	5	Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5
Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5	James	E	Borg	888665555	1997-11-10	450 Stone, Houston, TX	Man	94199	*NULL*	1
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	Woman	44183	333445555	5	Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5
Ramesh	K	Narayan	666884444	1995-09-15	975 Fire Oak, Humble, TX	Man	60000	333445555	5	Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5
Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	4	James	E	Borg	888665555	1997-11-10	450 Stone, Houston, TX	Man	94199	*NULL*	1
Ahmad	V	Jabbar	987987987	2000-03-29	980 Dallas, Houston, TX	Man	44183	987654321	4	Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	4
Alicia	J	Zelaya	999887777	1998-07-19	3321 Castle, Spring, TX	Non-binary	44183	987654321	4	Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	4

Why is this version better?

"For each employee, retrieve the employee's name and the name of the employee's supervisor"

```
SELECT *
FROM    employee AS e LEFT JOIN employee AS s
        ON  e.superssn = s.ssn;
```

fname	minit	lname	ssn	bdate	address	gender	salary	superssn	dno	fname	minit	lname	ssn	bdate	address	gender	salary	superssn	dno
John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	5	Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5
Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5	James	E	Borg	888665555	1997-11-10	450 Stone, Houston, TX	Man	94199	*NULL*	1
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	Woman	44183	333445555	5	Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5
Ramesh	K	Narayan	666884444	1995-09-15	975 Fire Oak, Humble, TX	Man	60000	333445555	5	Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5
James	E	Borg	888665555	1997-11-10	450 Stone, Houston, TX	Man	94199	*NULL*	1	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	4	James	E	Borg	888665555	1997-11-10	450 Stone, Houston, TX	Man	94199	*NULL*	1
Ahmad	V	Jabbar	987987987	2000-03-29	980 Dallas, Houston, TX	Man	44183	987654321	4	Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	4
Alicia	J	Zelaya	999887777	1998-07-19	3321 Castle, Spring, TX	Non-binary	44183	987654321	4	Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	4

8 rows (0.002 s) Edit, Explain, Export

EXAMPLE 43: For each employee, retrieve the employee's name and the name of the employee's supervisor

```
SELECT  CONCAT(e.fname, ' ' , e.lname) AS employee,
        CONCAT(s.fname, ' ' , s.lname) AS supervisor
FROM    employee AS e LEFT JOIN employee AS s
        ON  e.superssn = s.ssn;
```

+ Options

employee	supervisor
John Smith	Franklin Wong
Franklin Wong	James Borg
Joyce English	Franklin Wong
Ramesh Narayan	Franklin Wong
James Borg	*NULL*
Jennifer Wallace	James Borg
Ahmad Jabbar	Jennifer Wallace
Alicia Zelaya	Jennifer Wallace

EXAMPLE 44: For each department, list the department name, and the names, addresses and the start date of all managers, ordered by department name

```
SELECT

FROM

WHERE

ORDER BY          ;
```

CAN SUB-QUERIES AND JOINS BE USED INTERCHANGEABLY?

In some cases, yes, can replace a join of tables (where appropriate) with a sub-query

But recall …

- Joins are needed when values across multiple tables must be displayed.
- Sub-queries are needed when an existing value from a table needs to be retrieved and used as part of the query solution.
- Sub-queries are needed when an aggregate function needs to be performed and used as part of a query solution.

EXAMPLE 45: JOINS AND GROUP BY

List the employee name, and number of dependents of each employee who has dependents

essn	fname	lname	numDeps
123456789	John	Smith	3
333445555	Franklin	Wong	3
987654321	Jennifer	Wallace	1

```
SELECT      essn, fname, lname,
            COUNT(*) AS numDeps
FROM        employee INNER JOIN dependent
            ON ssn = essn
GROUP BY    essn, fname, lname;
```

Why won't this work?

```
SELECT      essn, fname, lname,  COUNT(*) AS numDeps
FROM        employee INNER JOIN dependent
            ON ssn = essn
GROUP BY    essn;
```

Error in query (1055): Expression #2 of SELECT list is not in GROUP BY clause and contains nonaggregated column 'mydb2974.employee.salary' which is not functionally dependent on columns in GROUP BY clause; this is incompatible with sql_mode=only_full_group_by

EXAMPLE 46: List the project name and the number of employees who work on the project for projects that have 2 or more employees

SELECT pname,

COUNT(*) AS numEmps

FROM

GROUP BY

HAVING

pname	numEmps
ProductX	2
ProductY	3
ProductZ	3
Computerization	2
Reorganization	3
Newbenefits	3

UNION QUERIES

The keyword UNION is used to combine the results of two or more queries or tables

MySQL does not support minus or intersection (intersect) operators but the same functionality can be built using joins

For union queries, tables must be **union compatible**

UNION COMPATIBLE

Two relations are **union compatible** if the schemas of the two relations match, i.e.,

same number of attributes in each relation and each pair of corresponding attributes have the same domain

Example 47: Using both subqueries and union queries (no joins) list all project numbers for projects that involve a worker whose last name is 'Wallace' or a manager, of the department that controls the project, with last name 'Wallace'

Steps:

First, consider two queries on their own and these can be combined with a Union query:

Query 1. Finding the employees 'Wallace' working on projects …

Query 2. Finding the manger 'Wallace' of a department that controls project

Example 47: Using both subqueries and union queries (no joins) list all project numbers for projects that involve a worker whose last name is 'Wallace' or a manager, of the department that controls the project, with last name 'Wallace'

```sql
-- employee
SELECT pno
FROM  works_on
WHERE essn IN
 (SELECT ssn
  FROM employee
  WHERE lname =
 'Wallace');
```

```sql
-- manager
SELECT pnumber
FROM project
WHERE dnum IN
 (SELECT dnumber
  FROM department
  WHERE mgrssn IN
      (SELECT ssn
       FROM employee
       WHERE lname =
 'Wallace'));
```

EXAMPLE 47 Full solution

```
(SELECT pno
 FROM   works_on
 WHERE  essn IN
  (SELECT ssn  FROM employee
   WHERE lname = 'Wallace'))
UNION
(SELECT pnumber
 FROM   project
 WHERE  dnum IN (SELECT dnumber FROM department
 WHERE  mgrssn IN (SELECT ssn   FROM employee
                  WHERE  lname = 'Wallace')));
```

MORE EXAMPLES

Example 48

Using a join, list all the locations of the research department

Example 49

For all projects located in 'Houston' list the name of the project and the department which controls the project

Example 50

List the names of employees, and the number of hours they work, for employees who work greater than the average number of hours

SUMMARY: JOINS AND UNION QUERIES

Important to know:

- How joins work in general
- How implicit and explicit inner joins work
- How left and right joins work
- When to use sub-queries and joins
- How Union queries work

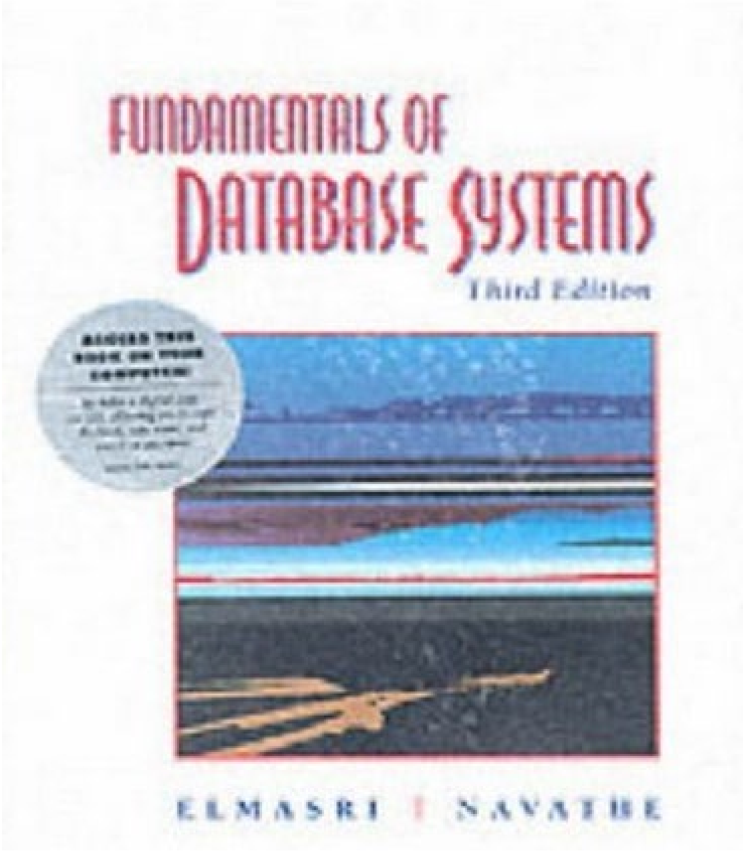
TOPIC: *NORMALISATION PART 1*

C230 Database Systems

FUNDAMENTALS OF DATABASE SYSTEMS
ELMASRI AND NAVATHE BOOK

See Chapter 14

(in 3rd Edition)



MOTIVATIONS

- We can see from ER examples and mappings why we get a particular grouping of tables.

However:

- What if different assumptions were made in the ER model that leads to different – maybe larger (more attributes/columns) tables?
- What happens over time as we need to add more attributes to our tables to capture information that was not part of the original requirements when creating the ER model?

For example, what if:

The employee entity had extra attributes to represent the department information?

fname	minit	lname	ssn	bdate	address	gender	salary	superssn	dname	dnumber	mgrssn	mgrstartdate
John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	Research	5	333445555	2018-05-22
Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	Research	5	333445555	2018-05-22
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	Woman	44183	333445555	Research	5	333445555	2018-05-22
Ramesh	K	Narayan	666884444	1995-09-15	975 Fire Oak, Humble, TX	Man	60000	333445555	Research	5	333445555	2018-05-22
James	E	Borg	888665555	1997-11-10	450 Stone, Houston, TX	Man	94199	*NULL*	Headquarters	1	888665555	2019-06-19
Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	Administration	4	987654321	2015-01-01
Ahmad	V	Jabbar	987987987	2000-03-29	980 Dallas, Houston, TX	Man	44183	987654321	Administration	4	987654321	2015-01-01
Alicia	J	Zelaya	999887777	1998-07-19	3321 Castle, Spring, TX	Non-binary	44183	987654321	Administration	4	987654321	2015-01-01

For example, what if:

The employee entity had the dependent information stored as attributes?

fname	minit	lname	ssn	bdate	address	gender	salary	superssn	dno	dependent_name	gender	bdate	relationship
John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	5	Alice	Woman	2008-12-30	Daughter
John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	5	Elizabeth	Woman	1976-05-05	Spouse
John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	5	Michael	Man	2011-01-04	Son
Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5	Alice	Woman	2010-04-05	Daughter
Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5	Joy	Woman	1981-05-03	Spouse
Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	5	Theodore	Man	2014-10-25	Son
Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	4	Abner	Woman	1992-02-28	Spouse

NORMALISATION

Normalisation rules gives us a formal measure of why one grouping of attributes in a relation schema may be better than another.

Normalised and un-normalised databases

We can distinguish between normalised and un-normalised databases

Both normalised and un-normalised databases have advantages and disadvantages

If database is normalised:

No (or very little) redundancy.

No anomalies when inserting, deleting or modifying data.

If database is normalised:

More tables.

More foreign and primary keys to link tables

=> more complex queries (joins etc.)

DEFINITION: **Redundancy**

<u>Unnecessary</u> duplication of data in the database

e.g. if we included department details in `Employee`?

fname	minit	lname	ssn	bdate	address	gender	salary	superssn	dname	dnumber	mgrssn	mgrstartdate
John	B	Smith	123456789	1975-01-09	731 Fondren, Houston, Tx	Man	55250	333445555	Research	5	333445555	2018-05-22
Franklin	T	Wong	333445555	1980-12-08	638 Voss, Houston, TX	Man	65000	888665555	Research	5	333445555	2018-05-22
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	Woman	44183	333445555	Research	5	333445555	2018-05-22
Ramesh	K	Narayan	666884444	1995-09-15	975 Fire Oak, Humble, TX	Man	60000	333445555	Research	5	333445555	2018-05-22
James	E	Borg	888665555	1997-11-10	450 Stone, Houston, TX	Man	94199	*NULL*	Headquarters	1	888665555	2019-06-19
Jennifer	S	Wallace	987654321	1991-06-20	291 Berry, Bellaire, TX	Woman	69240	888665555	Administration	4	987654321	2015-01-01
Ahmad	V	Jabbar	987987987	2000-03-29	980 Dallas, Houston, TX	Man	44183	987654321	Administration	4	987654321	2015-01-01
Alicia	J	Zelaya	999887777	1998-07-19	3321 Castle, Spring, TX	Non-binary	44183	987654321	Administration	4	987654321	2015-01-01

CONSEQUENCES OF REDUNDANCY:

Space is wasted (due to duplication)

Data can become inconsistent due to potential problems with update, insert and delete operations

DEFINITION: **Duplication**

Duplicated data can naturally be present in a database and is not necessarily redundant.

For example, an attribute can have two identical values.

e.g., In company schema, `ESSN` in `works_on` may be duplicated across many projects.

** Data is duplicated rather than redundant if when deleting data, information is lost.

EXAMPLE 1:

For the company schema, consider the following alternative schema for department which was initially created when each department had only one location:

`department(dnumber,  dname, mgrssn, dlocation)`

However, over time as the company grew, departments were located in multiple locations:

dnumber	dname	mgrssn	dlocation
1	Headquarters	888665555	Houston
4	Administration	987654321	Stafford
5	Research	333445555	Bellaire
5	Research	333445555	Houston
5	Research	333445555	Sugarland

Problems:

dnumber	dname	mgrssn	dlocation
1	Headquarters	888665555	Houston
4	Administration	987654321	Stafford
5	Research	333445555	Bellaire
5	Research	333445555	Houston
5	Research	333445555	Sugarland

1. What can be used as the primary key?

dnumber and dlocation

2. What happens if a new manager is appointed to the department with dnumber = 5?

3 tuples will need to be modified in this case

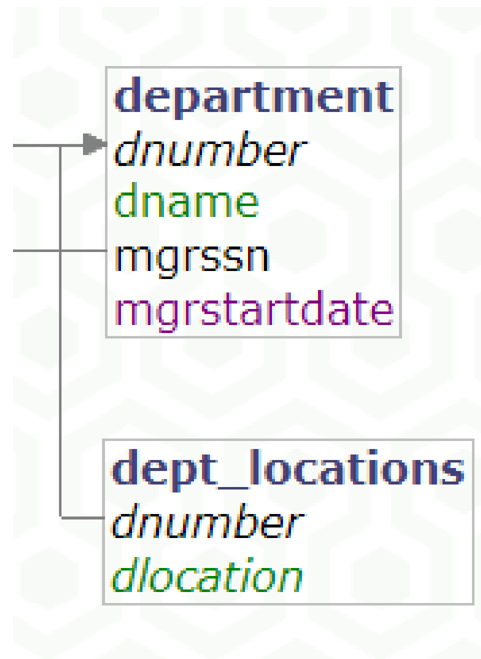
3. What happens if we add a new department, say "Development" with dnumber = 7?

Cannot be added unless we know where the department will be located.

FIXING THESE PROBLEMS?

This does not seem a good grouping of attributes …

We have seen, and worked with, a better one which stores `location` in a new table and uses `dnumber` as a foreign key to link to the other department information



EXAMPLE 2:

For the company schema, consider the following alternative schema to store information on employees and the projects they work on:

```
employee(ssn, fname, lname, address,  bdate, salary,
 pno, pname, plocation)
```

And the following (partial) instance:

ssn	fname	lname	address	bdate	salary	pno	pname	plocation
123456789	John	Smith	731 Fondren, Houston, Tx	1975-01-09	55250	1	ProductX	Bellaire
453453453	Joyce	English	5631 Rice, Houston, TX	1972-07-31	44183	1	ProductX	Bellaire
123456789	John	Smith	731 Fondren, Houston, Tx	1975-01-09	55250	2	ProductY	Sugarland
333445555	Franklin	Wong	638 Voss, Houston, TX	1980-12-08	65000	2	ProductY	Sugarland
453453453	Joyce	English	5631 Rice, Houston, TX	1972-07-31	44183	2	ProductY	Sugarland
333445555	Franklin	Wong	638 Voss, Houston, TX	1980-12-08	65000	3	ProductZ	Houston
666884444	Ramesh	Narayan	975 Fire Oak, Humble, TX	1995-09-15	60000	3	ProductZ	Houston
333445555	Franklin	Wong	638 Voss, Houston, TX	1980-12-08	65000	10	Computerization	Stafford
987987987	Ahmad	Jabbar	980 Dallas, Houston, TX	2000-03-29	44183	10	Computerization	Stafford
333445555	Franklin	Wong	638 Voss, Houston, TX	1980-12-08	65000	20	Reorganization	Houston

Problems?

1. What can be used as the key?

ssn and pno

2. What happens if we want to update the database when a new employee, Maria Browne, of 24 Cherry Drive, Voss, Houston, joins the company (with ssn = 34343434343)

cannot be added unless she is given a project to work on

ssn	fname	lname	address	bdate	salary	pno	pname	plocation
123456789	John	Smith	731 Fondren, Houston, Tx	1975-01-09	55250	1	ProductX	Bellaire
453453453	Joyce	English	5631 Rice, Houston, TX	1972-07-31	44183	1	ProductX	Bellaire
123456789	John	Smith	731 Fondren, Houston, Tx	1975-01-09	55250	2	ProductY	Sugarland
333445555	Franklin	Wong	638 Voss, Houston, TX	1980-12-08	65000	2	ProductY	Sugarland
453453453	Joyce	English	5631 Rice, Houston, TX	1972-07-31	44183	2	ProductY	Sugarland
333445555	Franklin	Wong	638 Voss, Houston, TX	1980-12-08	65000	3	ProductZ	Houston
666884444	Ramesh	Narayan	975 Fire Oak, Humble, TX	1995-09-15	60000	3	ProductZ	Houston
333445555	Franklin	Wong	638 Voss, Houston, TX	1980-12-08	65000	10	Computerization	Stafford
987987987	Ahmad	Jabbar	980 Dallas, Houston, TX	2000-03-29	44183	10	Computerization	Stafford
333445555	Franklin	Wong	638 Voss, Houston, TX	1980-12-08	65000	20	Reorganization	Houston

ssn	fname	lname	address	bdate	salary	pno	pname	plocation
123456789	John	Smith	731 Fondren, Houston, Tx	1975-01-09	55250	1	ProductX	Bellaire
453453453	Joyce	English	5631 Rice, Houston, TX	1972-07-31	44183	1	ProductX	Bellaire
123456789	John	Smith	731 Fondren, Houston, Tx	1975-01-09	55250	2	ProductY	Sugarland
333445555	Franklin	Wong	638 Voss, Houston, TX	1980-12-08	65000	2	ProductY	Sugarland
453453453	Joyce	English	5631 Rice, Houston, TX	1972-07-31	44183	2	ProductY	Sugarland
333445555	Franklin	Wong	638 Voss, Houston, TX	1980-12-08	65000	3	ProductZ	Houston
666884444	Ramesh	Narayan	975 Fire Oak, Humble, TX	1995-09-15	60000	3	ProductZ	Houston
333445555	Franklin	Wong	638 Voss, Houston, TX	1980-12-08	65000	10	Computerization	Stafford
987987987	Ahmad	Jabbar	980 Dallas, Houston, TX	2000-03-29	44183	10	Computerization	Stafford
333445555	Franklin	Wong	638 Voss, Houston, TX	1980-12-08	65000	20	Reorganization	Houston

3. Update the database when ProductX and ProductY are completed and details on the projects should be removed

If we delete the relevant tuples, then all details on John Smith will be lost

4. Update the database with a new address for Franklin Wong

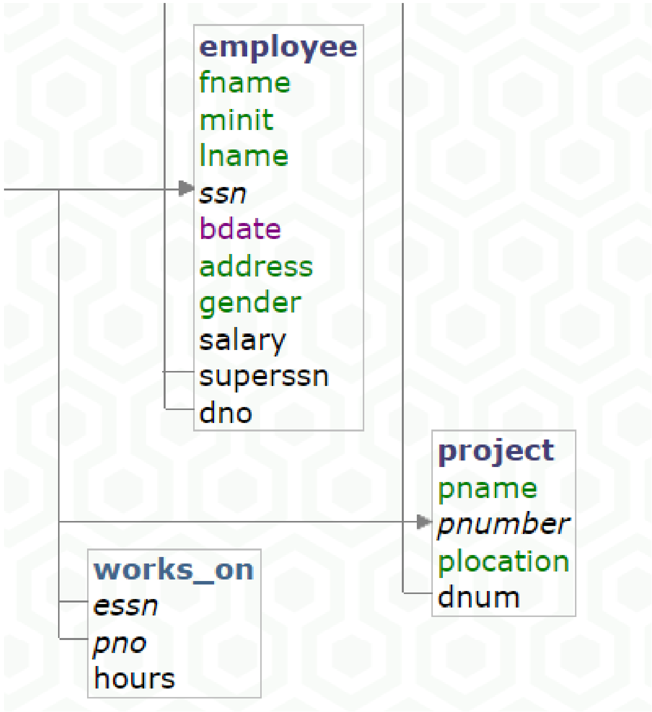
In this case, 4 tuples must be updated with the new address

FIXING THESE PROBLEMS?

This does not seem a good grouping of attributes …

We have seen, and worked with, a better one **involving 3 tables**

Note however the repetition of `ssn` (as `essn`) and `pnumber`/`pno`



NORMALISATION

Developed by Codd, 1972

- Takes each table through a series of tests to “verify” whether or not it belongs to a certain **normal form**
- Normal forms to check:
  - 1st, 2nd and 3rd normal forms (NF)
  - Boyce-Codd normal form – strong 3NF
  - 4th and 5th Normal Forms
- *We will consider 1NF, 2NF and 3NF only in detail*

NORMALISATION PROVIDES:

1. Formal framework for analysing relation schemas based on keys and functional dependencies among attributes.

2. Series of tests so that a database can be normalised to any degree (e.g., from 1NF to 5NF).

3. ***But*** does not necessarily provide a good design if considered in isolation to everything else.

WHY NORMALISE?

- Redundancy will be reduced or eliminated.
- Storage space will be reduced as a result.
- Task of maintaining data integrity is made easier.

However with normalisation, tables are usually added to the schema and linked with foreign keys. Thus queries become more complex as they often require data from multiple tables (requiring joins or subqueries).

ALTERNATIVES?

Retain redundant data and maintain data integrity by means of code consistency checks

In some applications the number of insertions may be very small or non-existent (e.g. analysing past logs, transaction data, weather data etc.) and in such cases the overhead of normalised tables is generally **not** required.

DE-NORMALISATION

A process of making compromises to the normalised tables by introducing intentional redundancy for performance reasons (querying performance).

Typically, de-normalisation will improve query times at the expense of data updates (insert, delete, update).

DEFINITION: Functional Dependency

Functional dependency is one of the main concepts associated with normalisation and describes the *relationship between attributes.*

If A and B are attributes of a relation R, then **B is functionally dependent (FD) on A** if each value of A is associated with exactly one value of B.

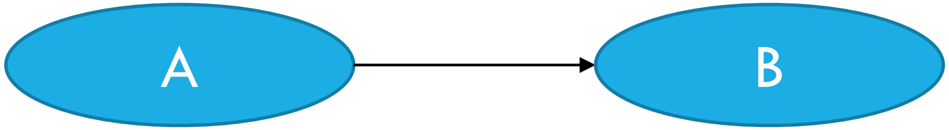
i.e., values in B are uniquely determined by values of A

TERMINOLOGY:
FUNCTIONAL DEPENDENCY (FD)

A $\longrightarrow$ B :

FD from A to B

B is FD on A



NOTES ON NOTATION:

A → B does not necessarily imply B → A

A ↔ B denotes A → B and B → A

A → {B, C} denotes A → B and A → C

{A, B} → C denotes that it is the **combination** of A and B that uniquely determines C.

TERMINOLOGY:
CANDIDATE KEY (CK)

Every relation has one or more candidate keys. A candidate key (CK) is one or more attribute(s) in a relation with which you can determine all the attributes in the relation.

Recall we pick one such candidate key as the primary key of a relation.

EXAMPLE 3: FINDING THE FUNCTIONAL DEPENDENCIES – GIVEN THE PRIMARY KEY

For the company schema, consider the following alternative schema to hold information on employees and projects:

`emp_proj(ssn, pnumber, hours, ename, pname, plocation)`

What are the functional dependencies?

- Think of this question as … "which attribute can be uniquely determined from another attribute"
- Begin with any known PK or CK

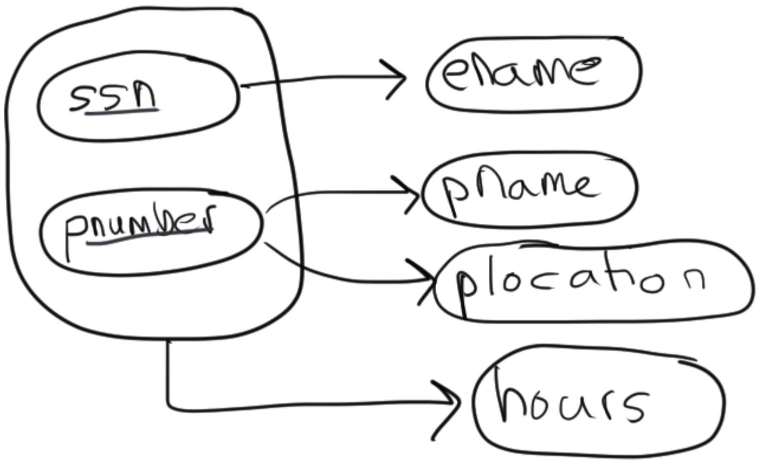
Can represent these FDs graphically:

emp_proj(ssn, pnumber, hours, ename, pname, plocation)

ssn → ename

pnumber → {pname, plocation}

{ssn, pnumber} → hours



IMPORTANT TO NOTE:

A functional dependency is a property of a relation schema *R* and cannot be inferred automatically but instead must be defined explicitly by someone who knows the **semantics** of *R*

i.e.

You will either be:

- explicitly given all FDs.
- given enough information about the attributes and the domain to *reasonably* infer the FDs (perhaps having to make certain assumptions).

TYPES OF FUNCTIONAL DEPENDENCIES

1. Full Functional Dependency:

A functional dependency {X,Y} → Z is a **full** functional dependency if when some attribute (either X or Y) is removed from the LHS the dependency **does not hold**.

Note: There may be any number of attributes on LHS

2. Partial Functional Dependency:

A functional dependency {X,Y} → Z is a **partial** functional dependency if some attribute (either X or Y) can be removed from the LHS and the dependency **still holds.**

Note: There may be any number of attributes on LHS

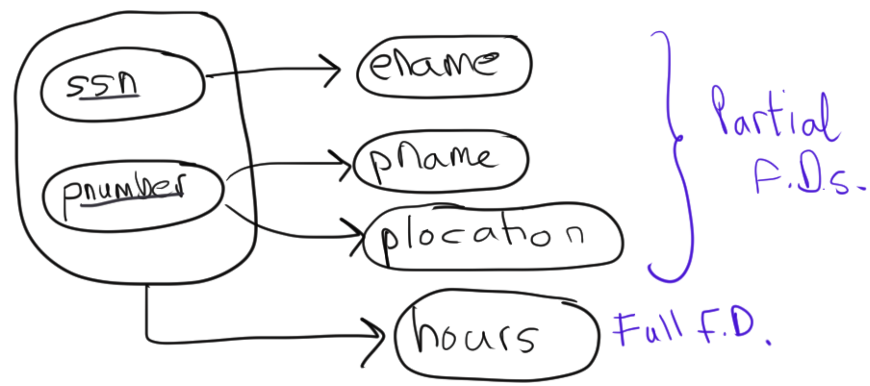
CONSIDER EXAMPLE 3 AGAIN:

emp_proj(ssn, pnumber, hours, ename, pname, plocation)

Are the following Full or Partial Functional Dependencies?

{ssn, pnumber} → hours

{ssn, pnumber} → ename



TYPES OF FUNCTIONAL DEPENDENCIES

3. Transitive Dependency:

A functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$ is a transitive dependency in the table/relation R if there is a set of attributes **Z** that is neither a candidate key nor a subset of any key of R and both:

$\mathbf{X} \rightarrow \mathbf{Z}$ and

$\mathbf{Z} \rightarrow \mathbf{Y}$

hold.

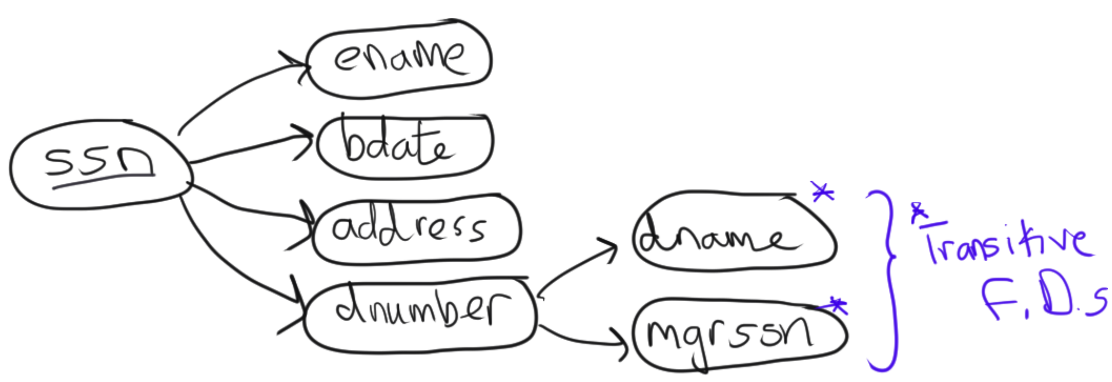
EXAMPLE 4:
Consider information on employees and departments

emp_dept(ename, ssn, bdate, address, dnumber, dname, dmgrssn)

The functional dependencies are:

ssn → {ename, bdate, address, dnumber}

dnumber → {dname, dmgrssn)



EXAMPLE 4:
An example of a transitive dependency

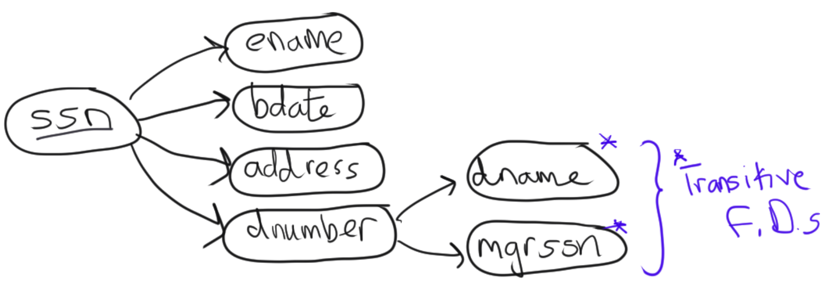
The dependency:

ssn → dmgrssn

is transitive through `dnumber` because both the following hold:

ssn → dnumber

dnumber → dmgrssn



But `dnumber` is neither a key or a subset of the key.

EXAMPLE 5:

Given the following table to hold student data:

`student(id, name, course, assocCollege, courseCoordinator)`

and the following Functional Dependencies:

id $\longrightarrow$ name

id $\longrightarrow$ course

course $\longrightarrow$ assocCollege

course $\longrightarrow$ courseCoordinator

EXAMPLE 5:
What is the candidate key?
What are the full dependencies?
What are the transitive dependencies?

Given the following table to hold student data:

```
student(id, name, course, assocCollege,
courseCoordinator)
```

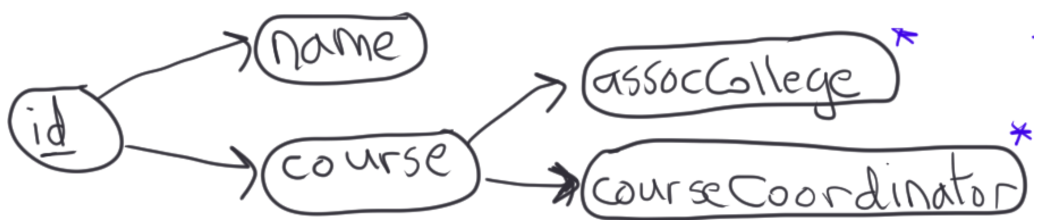
and the following Functional Dependencies:

id → name

id → course

course → assocCollege

course → courseCoordinator



EXAMPLE 6:
Draw the functional dependency diagram and find the candidate key

Consider the table R with 5 attributes

`R(A, B, C, D, E)`

and the following functional dependencies:

$A \rightarrow B$

$B \rightarrow A$

$B \rightarrow C$

$D \rightarrow A$

`R(A, B, C, D, E)`

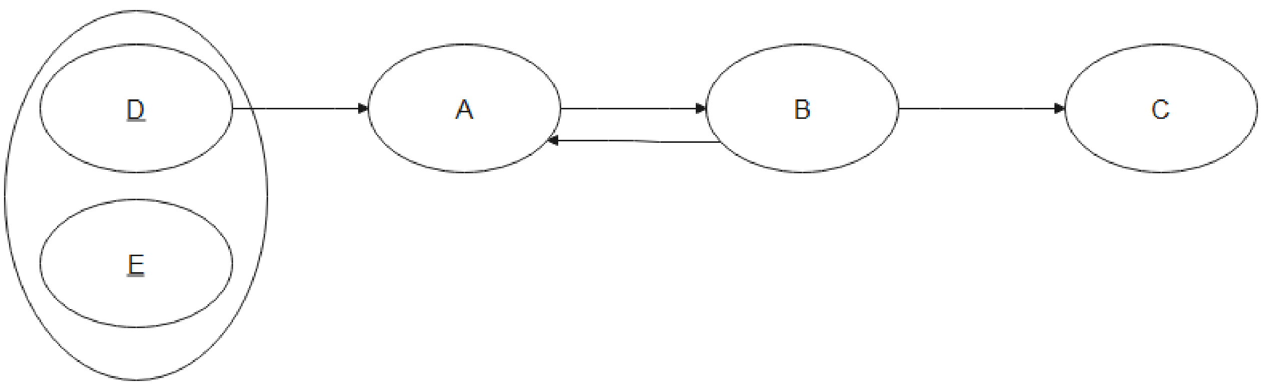
and the following functional dependencies:

$A \rightarrow B$

$B \rightarrow A$

$B \rightarrow C$

$D \rightarrow A$



Inference rules for Functional Dependencies

Typically the main **obvious** functional dependencies are specified for a schema

– call these F.

However many others can be inferred from F

– call these closure of F: F^+

FOR EXAMPLE:

$$F = \{ \quad A \rightarrow \{B, C, D, E\}$$
$$E \rightarrow \{F, G\} \quad \}$$

Some other FDs which can be inferred:

$A \rightarrow A$

$A \rightarrow \{F, G\}$

$E \rightarrow F$

etc.

Inference Rules for FDs:

1. Reflexive: Trivially, an attribute, or set of attributes, always determines itself.

2. Augmentation: if $X \rightarrow Y$ can infer $XZ \rightarrow YZ$

3. Transitive: if $X \rightarrow Y$ and $Y \rightarrow Z$ can infer $X \rightarrow Z$

4. Decomposition: if $X \rightarrow YZ$ can infer $X \rightarrow Y$

5. Union (additive): if $X \rightarrow Y$ and $X \rightarrow Z$ can infer if $X \rightarrow YZ$

6. Pseudotransitive: if $X \rightarrow Y$ and $WY \rightarrow Z$ can infer $WX \rightarrow Z$

*Note: Rules 1, 2 and 3 are together called **Armstrongs's Axioms**

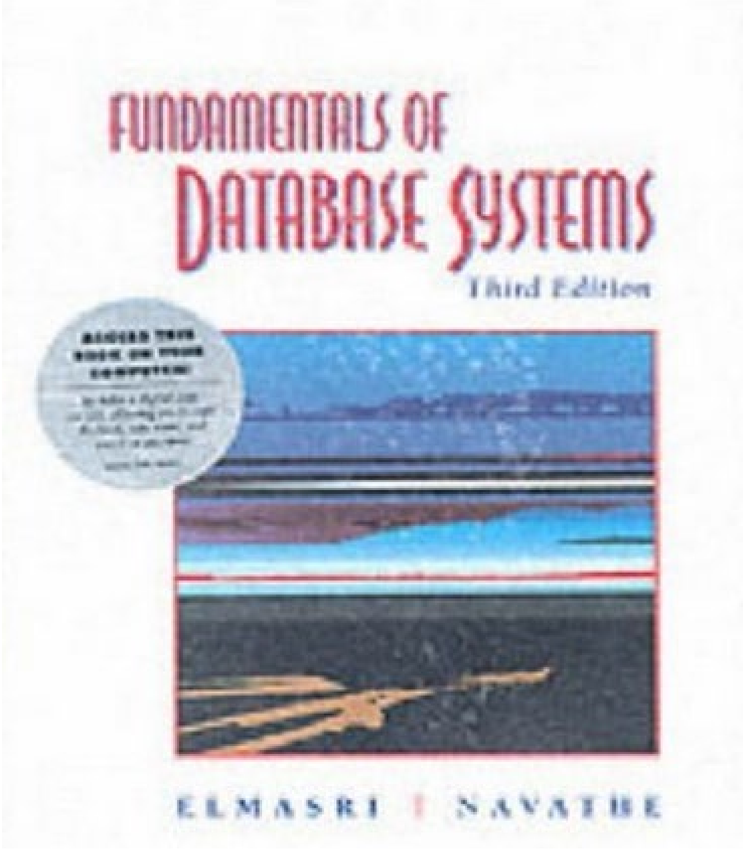
TOPIC: *NORMALISATION PART 2*

C230 Database Systems

FUNDAMENTALS OF DATABASE SYSTEMS
ELMASRI AND NAVATHE BOOK

See Chapter 14

(in 3rd Edition)



DEFINITION: Functional Dependency

Functional dependency is one of the main concepts associated with normalisation and describes the *relationship between attributes.*

If A and B are attributes of a relation R, then **B is functionally dependent (FD) on A** if each value of A is associated with exactly one value of B.

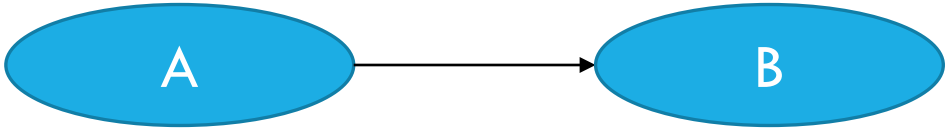
i.e., values in B are uniquely determined by values of A

TERMINOLOGY:
FUNCTIONAL DEPENDENCY (FD)

A → B :

FD from A to B

B is FD on A



NOTES ON NOTATION:

$A \longrightarrow B$ does not necessarily imply $B \longrightarrow A$

$A \leftrightarrow B$ denotes $A \longrightarrow B$ and $B \longrightarrow A$

$A \longrightarrow \{B, C\}$ denotes $A \longrightarrow B$ and $A \longrightarrow C$

$\{A, B\} \longrightarrow C$ denotes that it is the **combination** of A and B that uniquely determines C.

TERMINOLOGY:
CANDIDATE KEY (CK)

Every relation has one or more candidate keys. A candidate key (CK) is one or more attribute(s) in a relation with which you can determine all the attributes in the relation.

Recall we pick one such candidate key as the primary key of a relation.

EXAMPLE 3: FINDING THE FUNCTIONAL DEPENDENCIES – GIVEN THE PRIMARY KEY

For the company schema, consider the following alternative schema to hold information on employees and projects:

`emp_proj(ssn, pnumber, hours, ename, pname, plocation)`

What are the functional dependencies?

- Think of this question as … "which attribute can be uniquely determined from another attribute"
- Begin with any known PK or CK

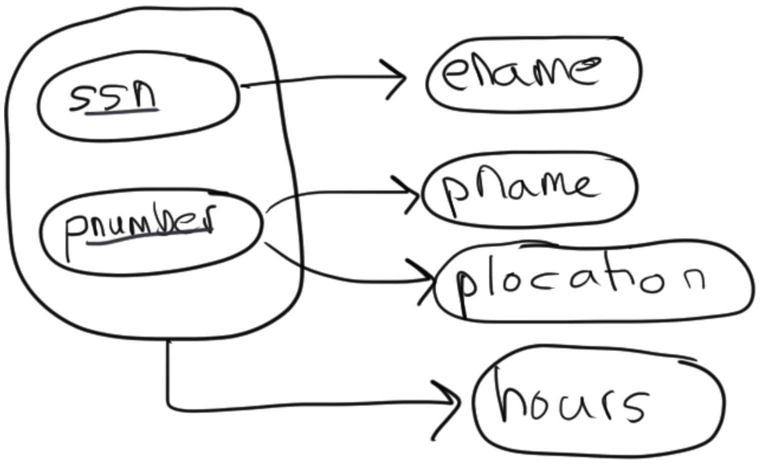
Can represent these FDs graphically:

`emp_proj(ssn, pnumber, hours, ename, pname, plocation)`

ssn → ename

pnumber → {pname, plocation}

{ssn, pnumber} → hours



IMPORTANT TO NOTE:

A functional dependency is a property of a relation schema *R* and cannot be inferred automatically but instead must be defined explicitly by someone who knows the **semantics** of *R*

i.e.

You will either be:

- explicitly given all FDs.
- given enough information about the attributes and the domain to *reasonably* infer the FDs (perhaps having to make certain assumptions).

TYPES OF FUNCTIONAL DEPENDENCIES

1. Full Functional Dependency:

A functional dependency {X,Y} → Z is a **full** functional dependency if when some attribute (either X or Y) is removed from the LHS the dependency **does not hold**.

Note: There may be any number of attributes on LHS

2. Partial Functional Dependency:

A functional dependency {X,Y} → Z is a **partial** functional dependency if some attribute (either X or Y) can be removed from the LHS and the dependency **still holds.**

Note: There may be any number of attributes on LHS

CONSIDER EXAMPLE 3 AGAIN:

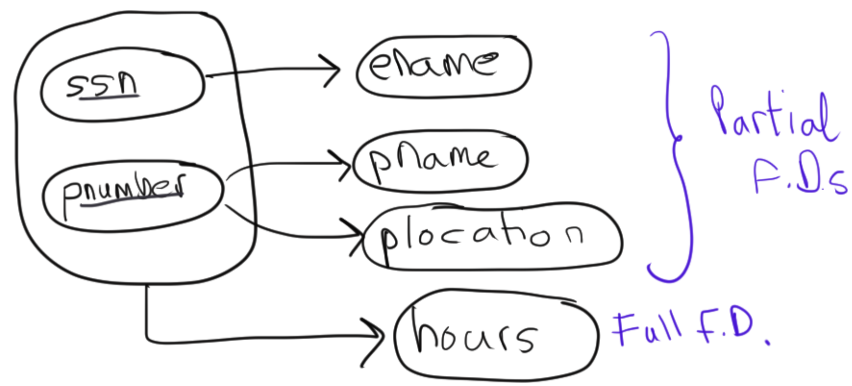
emp_proj(<u>ssn, pnumber</u>, hours, ename, pname, plocation)

Are the following Full or Partial Functional Dependencies?

See menti.com

{ssn, pnumber} → hours

{ssn, pnumber} → ename



TYPES OF FUNCTIONAL DEPENDENCIES

3. Transitive Dependency:

A functional dependency **X** $\rightarrow$ **Y** is a transitive dependency in the table/relation R if there is a set of attributes **Z** that is neither a candidate key nor a subset of any key of R and both:

X $\rightarrow$ **Z** and

Z $\rightarrow$ **Y**

hold.

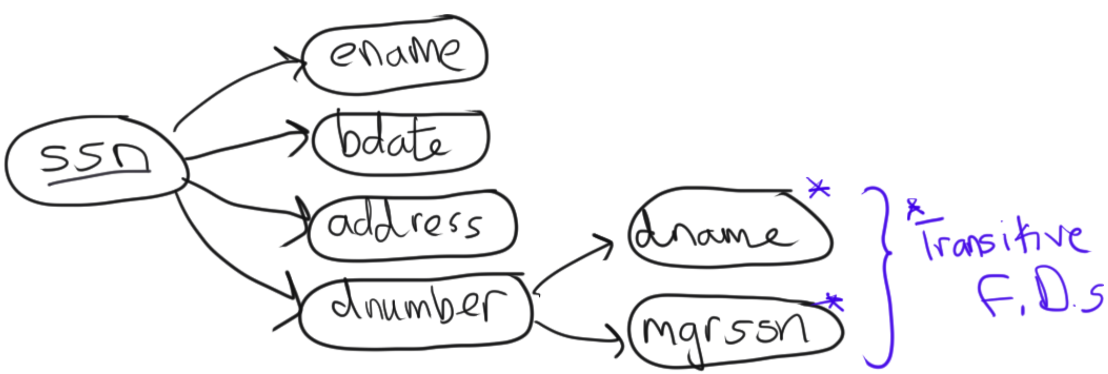
EXAMPLE 4:
Consider information on employees and departments

emp_dept(ename, ssn, bdate, address, dnumber, dname, dmgrssn)

The functional dependencies are:

ssn → {ename, bdate, address, dnumber}

dnumber → {dname, dmgrssn}



EXAMPLE 4:
An example of a transitive dependency

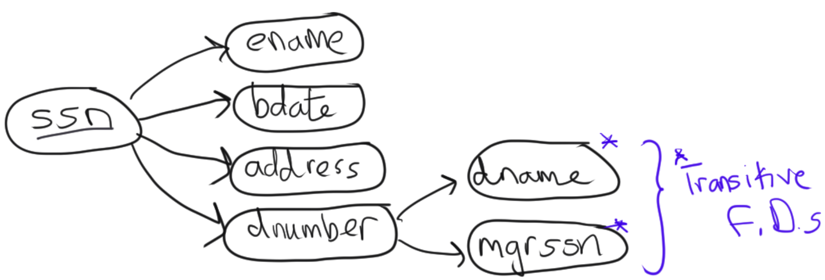
The dependency:

ssn → dmgrssn

is transitive through `dnumber` because both the following hold:

ssn → dnumber

dnumber → dmgrssn



But `dnumber` is neither a key or a subset of the key.

EXAMPLE 5:

Given the following table to hold student data:

`student(id, name, course, assocCollege, courseCoordinator)`

and the following Functional Dependencies:

id $\longrightarrow$ name

id $\longrightarrow$ course

course $\longrightarrow$ assocCollege

course $\longrightarrow$ courseCoordinator

EXAMPLE 5:
What is the candidate key?
What are the full dependencies?
What are the transitive dependencies?

Given the following table to hold student data:

```
student(id, name, course, assocCollege,
courseCoordinator)
```

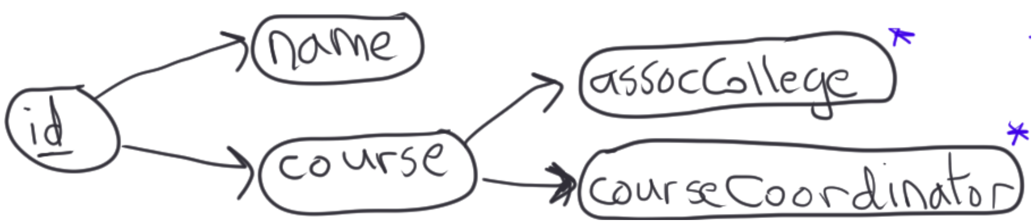
and the following Functional Dependencies:

id → name

id → course

course → assocCollege

course → courseCoordinator



EXAMPLE 6:
Draw the functional dependency diagram and find the candidate key

Consider the table R with 5 attributes

`R(A, B, C, D, E)`

and the following functional dependencies:

$A \rightarrow B$

$B \rightarrow A$

$B \rightarrow C$

$D \rightarrow A$

R(A, B, C, D, E)

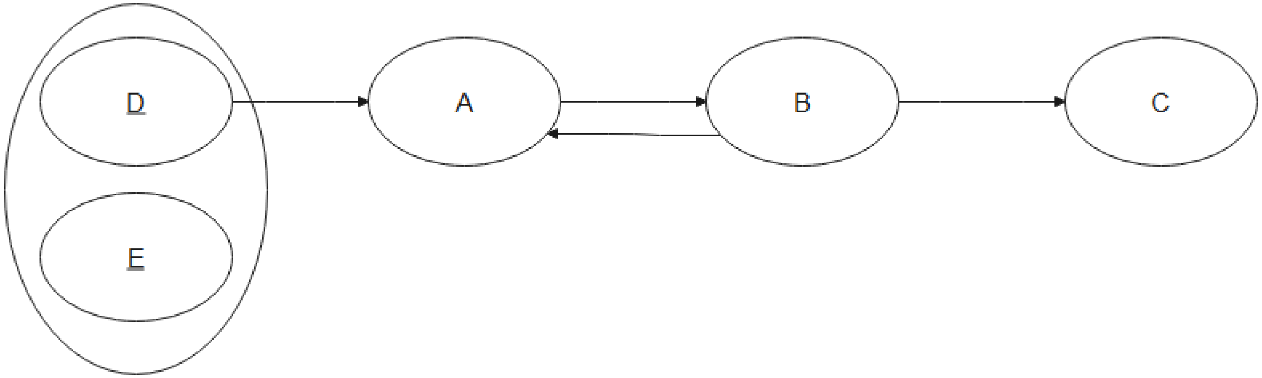
and the following functional dependencies:

$A \longrightarrow B$

$B \longrightarrow A$

$B \longrightarrow C$

$D \longrightarrow A$



Inference rules for Functional Dependencies

Typically the main **obvious** functional dependencies are specified for a schema

– call these F.

However many others can be inferred from F

– call these closure of F: F^+

FOR EXAMPLE:

$F = \{ \quad A \rightarrow \{B, C, D, E\}$

$\quad E \rightarrow \{F, G\} \quad \}$

Some other FDs which can be inferred:

$A \rightarrow A$

$A \rightarrow \{F, G\}$

$E \rightarrow F$

etc.

Inference Rules for FDs:

1. Reflexive: Trivially, an attribute, or set of attributes, always determines itself.

2. Augmentation: if $X \rightarrow Y$ can infer $XZ \rightarrow YZ$

3. Transitive: if $X \rightarrow Y$ and $Y \rightarrow Z$ can infer $X \rightarrow Z$

4. Decomposition: if $X \rightarrow YZ$ can infer $X \rightarrow Y$

5. Union (additive): if $X \rightarrow Y$ and $X \rightarrow Z$ can infer if $X \rightarrow YZ$

6. Pseudotransitive: if $X \rightarrow Y$ and $WY \rightarrow Z$ can infer $WX \rightarrow Z$

*Note: Rules 1, 2 and 3 are together called **Armstrongs's Axioms**

IMPORTANT CONCEPTS

Duplicated Data versus Redundant Data

Problems with un-normalised tables and maintaining redundant data

Trade off of un-normalised versus normalised tables

What is functional dependency – how to find it

What are full, partial and transitive dependencies – how to find them

DEFINITION:
FIRST NORMAL FORM (1NF)

A table is in 1NF if it satisfies the following:

The table must not have any **repeating groups**

Repeating groups: a group of attributes that occur a variable number of times in each record (non-atomic)

FIRST NORMAL FORM (1NF)

To ensure first normal form, choose an appropriate primary key (if one is not already specified) and if required, split table in to two or more tables to remove repeating groups

EXAMPLE 7:

Consider information on customers (unique number, name, address and their credit limit) and invoices issued to them (unique invoice number, date of invoice and amount in euros). Note that a customer can have many invoices issued to them.

```
customer(cNo, name, street, city,
credLim, invNo, invDate, amount)
```

Repeating Groups?

First Normal Form?

EXAMPLE 7

`customer(cno, name, street, city, credLim, invno, invDate, amount)`

To ensure 1NF, choose appropriate Primary Key ....

`cNo` **and** `invNo` **as primary key giving:**

customer(<u>cNo, invNo</u>, name, street, city, credLim, invDate, amount)

DEFINITION:
SECOND NORMAL FORM (2NF)

A relation in 2NF must be in 1NF and satisfy the following:

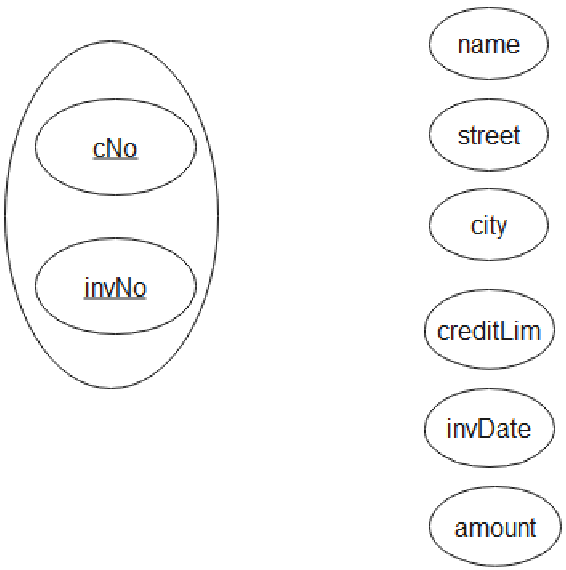
Where there is a composite primary key, all non-key attributes must be dependent on the **entire** primary key.

If partial dependencies exists create new relations to split the attributes such that the partial dependency no longer holds

check for partial dependencies and remove

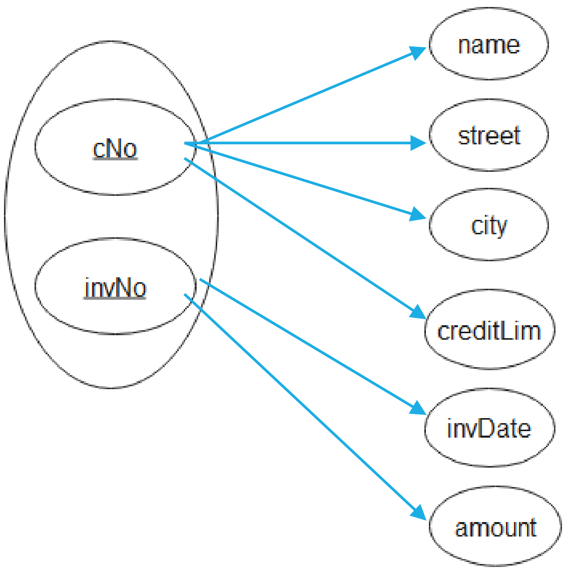
EXAMPLE 7:

customer(<u>cNo, invNo</u>, name, street, city, credLim, invDate, amount)



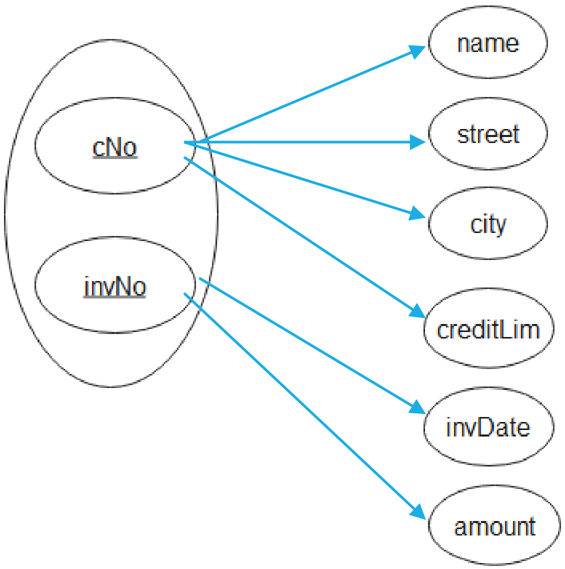
EXAMPLE 7:

customer(cNo, invNo, name, street, city, credLim, invDate, amount)



EXAMPLE 7:

customer(cNo, invNo, name, street, city, credLim, invDate, amount)



customerInvoice(cNo, invNo)

customer(cNo, name, street, city, credLim)

invoice(invNo, invDate, amount)

EXAMPLE 8:

Consider information on products that customers buy (e.g. the contents of their online basket). Information stored on customers is: unique customer number, name and address. The data stored on the products ordered is: unique product number, product description, unit price per product and quantity of each product required by the customer. The schema is:

```
purchase(CNo, ProdNo, cname, street, city, prodDesc,
price, quantity)
```

QUESTIONS:

`purchase(CNo, ProdNo, cname, street, city, prodDesc, price, quantity)`

- Is this table in first normal form?
- Draw a functional dependency diagram
- Is this table in second normal form?
- If not, what problems occur by the table not being in 2NF?
- If not, create a set of tables in 2NF

1NF?

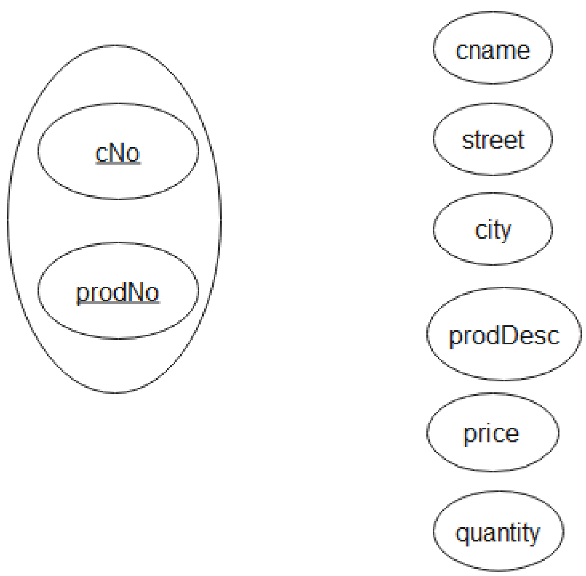
`purchase(CNo, ProdNo, cname, street, city, prodDesc, price, quantity)`

No primary key so not in 1NF.

A suitable primary key (using existing attributes) is a composite key of `CNo` and `ProdNo`

Draw the Functional Dependencies:

`purchase(CNo, prodNo, cname, street, city, prodDesc, price, quantity)`



Problems caused by `purchase` table not being in 2NF:

`purchase(cNo, prodNo, cname, street, city, prodDesc, price, quantity)`

Duplication of data:

- Every time a product is purchased by a customer the customer name, street etc. is stored again
- Every time a product is purchased, its description and price is stored again.

Create a set of tables in 2NF

Removing the partial dependencies means:

- Attributes that are partially dependent on the PK should move to a new table;
- The attribute on which they were dependent should be the PK of the new table but this attribute should not be removed from the original table

Giving the tables:

`purchase(cNo, prodNo, quantity)`

`customer(cNo, cname, street, city)`

`product(prodNo, prodDesc, price)`

N.B. Make sure each table has its own PK

DEFINITION:
THIRD NORMAL FORM (3NF)

A relation is in 3NF if it is in 2NF and there are no dependencies between attributes that are not primary keys. That is, no transitive dependencies exist in the table.

EXAMPLE 8 *extended*:

Consider the following information stored per product: unique product number (PK), product description and unit price and the number of the product in stock; also stored is the unique ID of the supplier of the product, and the supplier's details: name and address details:

```
product(prodNo, desc, price,
qty_in_stock, supplierNo, Sname,
Sstreet, Scity, SPostcode)
```

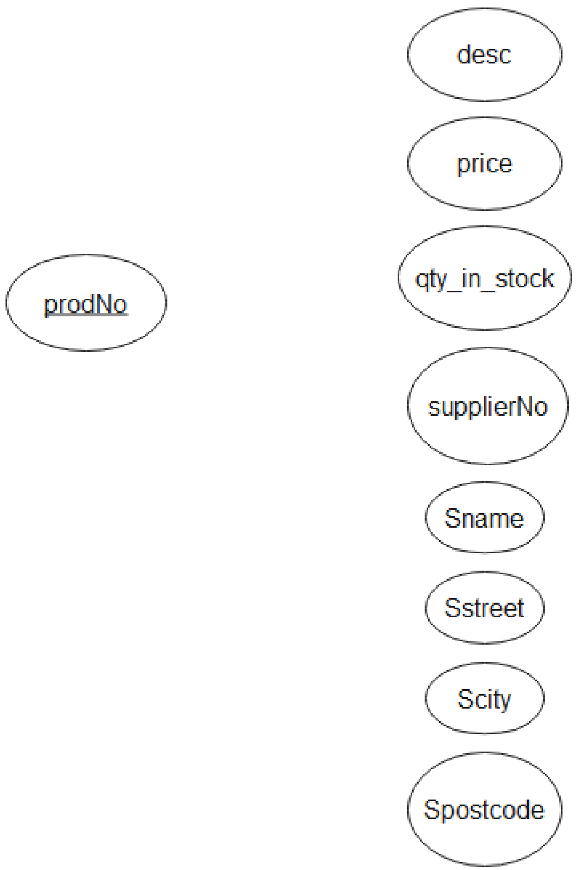
QUESTIONS:
EXAMPLE 8 extended

`product(prodNo, desc, price, qty_in_stock, supplierNo, Sname, Sstreet, Scity, SPostcode)`

(prodNo is underlined)

- Is this table in first normal form?
- Draw a functional dependency diagram
- Is this table in second and third normal form?
- If not, create a set of tables in 3NF

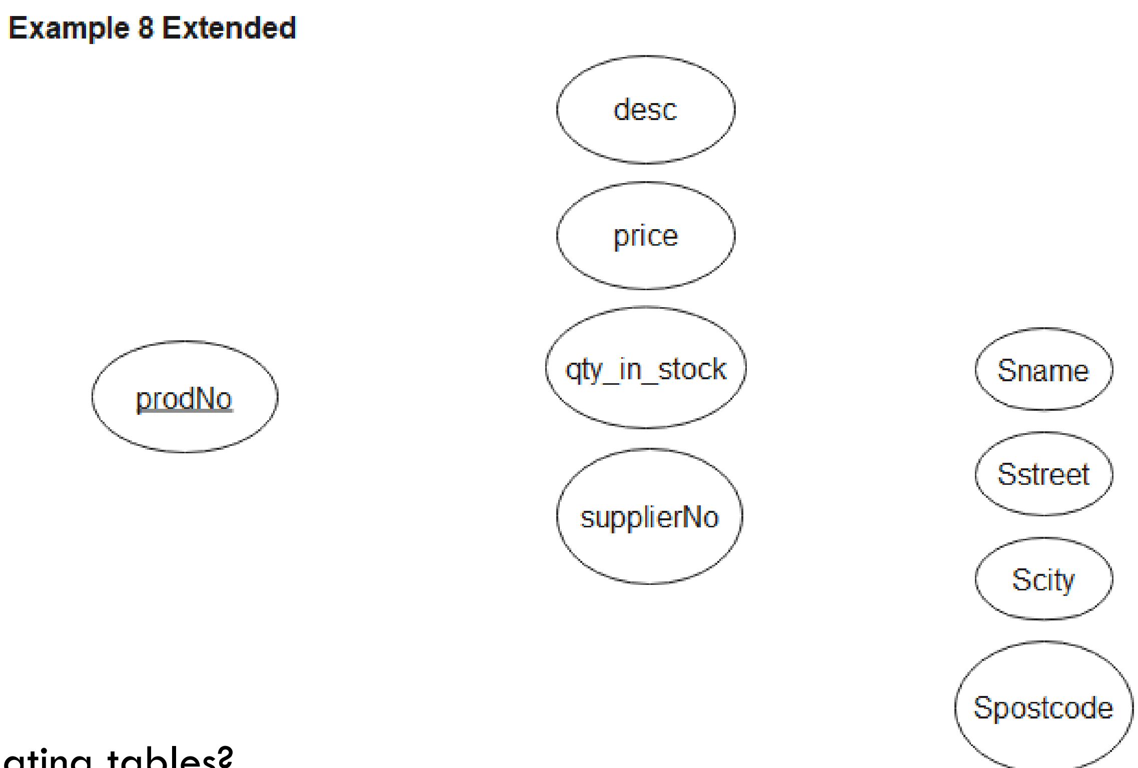
DEPENDENCY DIAGRAM FOR EXAMPLE 8 EXTENDED



Creating tables?

prodNo, desc, price, qty_in_stock, supplierNo, Sname, Sstreet, Scity, SPostcode

DEPENDENCY DIAGRAM FOR EXAMPLE 8 EXTENDED



Note: how we are creating links between the tables with Foreign Keys

Creating tables?

product(<u>prodNo</u>, desc, price, qty_in_stock, supplierNo)

supplier(<u>supplierNo</u>, Sname, Sstreet, Scity, Spostcode)

BOYCE-CODD NORMAL FORM (BCNF)

Only in rare cases does a 3NF table not meet the requirements of BCNF.

These cases are when a table has more than one candidate key - depending on the functional dependencies, a 3NF table with two or more overlapping candidate keys may or may not be in BCNF.

If a table in 3NF **does not** have multiple overlapping candidate keys then it is guaranteed to be in BCNF

SUMMARY: Steps to normalise to 3NF

- Identify appropriate Primary Key if not already given (this puts table in to 1NF)
- Draw diagram of Functional Dependencies from the primary key.
- Identify if dependencies are Full, Partial or Transitive.
- Using diagram of functional dependencies from previous step:
  - Normalise to 2NF by removing partial dependencies – creating new tables as a result. <u>Ensure all new tables have Primary Keys</u>
  - Normalise to 3NF by removing transitive dependencies (if they exist), creating new tables as a result. <u>Ensure any new tables have Primary Keys and are in 2NF</u>
  - Check that all resulting tables are themselves in 1NF, 2NF and 3NF (in particular, make sure they all have PKs of their own)

EXAMPLE 9:

An un-normalised staff relation has the following structure and description (next slide):

```
staff(sNo, sName, sAddress, deptNo,
deptName, managerNo, skilliD, skillName,
sCourseDate, sCourseDuration)
```

9.1. Where does duplication result from this relation design?

9.2. What is a suitable Primary Key to ensure the staff table is in 1NF?

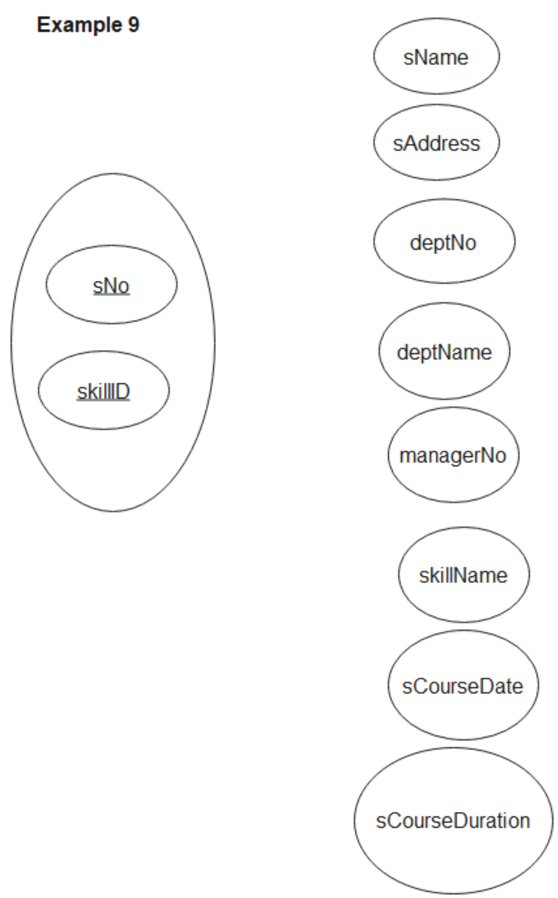
9.3. What attributes are fully functional dependent on the Primary Key?

Description 9(a):

`staff(sNo, sName, sAddress, deptNo, deptName, managerNo, skilliD, skillName, sCourseDate, sCourseDuration)`

A staff member has an associated number (sNo, which is unique for each staff member), a name and an address and works in a particular department. Each department has a number (unique), name and manager. A department has many staff but a staff member can only work for one department. A staff member can undertake a number of courses to gain new skills for their job. skilliD uniquely identifies the skill, which has also a name (skillName). **For each skill, courses are offered on a regular basis and staff can take the course at a date that suits them and complete the course at their own pace.** sCourseDate describes the date when a staff member undertakes the course for a particular skill and sCourseDuration describes the time that the staff member took to complete the course. A staff member cannot undertake more than one course to acquire a new skill.

FUNCTIONAL DEPENDENCIES



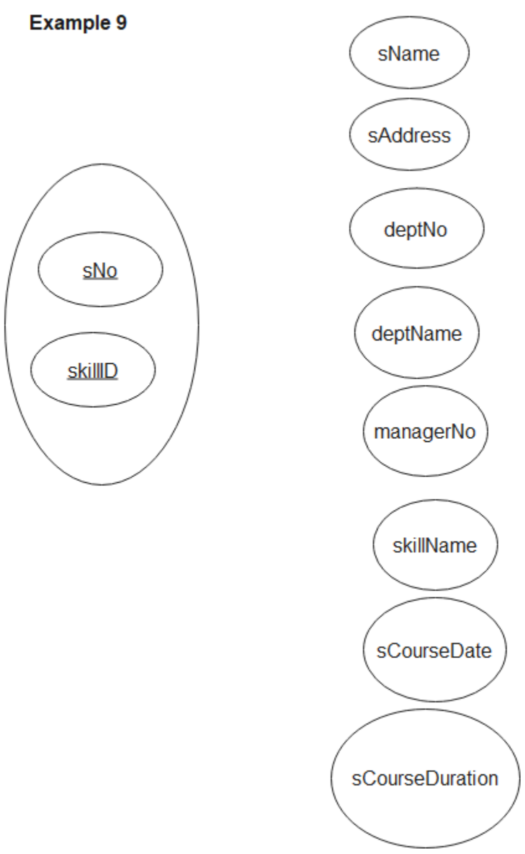
For each skill, courses are offered on a regular basis and staff can take the course at a date that suits them and complete the course at their own pace

Description 9(b)

`staff(sNo, sName, sAddress, deptNo, deptName, managerNo, skilliD, skillName, sCourseDate, sCourseDuration)`

A staff member has an associated number (sNo, which is unique for each staff member), a name and an address and works in a particular department. Each department has a number (unique), name and manager. A department has many staff but a staff member can only work for one department. A staff member can undertake a number of courses to gain new skills for their job. skilliD uniquely identifies the skill, which has also a name (skillName). **For each skill, courses are offered once at a certain date and for a certain duration and staff must take the course on that date:** sCourseDate describes the date of the course; sCourseDuration describes the length (in days) of the course. A staff member can undertake as many different courses as they wish.

FUNCTIONAL DEPENDENCIES



For each skill, courses are offered once at a certain date and for a certain duration and staff must take the course on that date

EXAMPLE 10: Winter 2019 Exam Paper question on Normalisation

A courier company keeps track of packages that are to be delivered to recipients, by couriers, in the following table:

`courier(packageID, recipientCode, recipientName, recipientAddr, recipientMobile, instructions, dateRec, dateDelivered, courierID, cName, cMobile)`

Stored in the `courier` table are: a unique package id (`packageID`) which is the primary key of the table, a code (`recipientCode`) which is unique to each recipient, and the name, address and mobile number of the recipient of the package (`recipientName`, `recipientAddr` and `recipientMobile`), delivery instructions (`instructions`), the date the package was received by the courier (`dateRec`), the date the courier delivers the package (`dateDelivered`), and details of the courier who delivers the package: an ID (`courierID`) which is unique to each courier, in addition to the courier's name (`cName`) and phone number (`cMobile`).

```
courier(packageID, recipientcode,
recipientname, recipientaddr, recipientmobile,
instructions, daterec, datedelivered,
courierid, cname, cmobile)
```

(i) By using the primary key given in the `courier` table, draw a functional dependency diagram showing the functional dependencies between all attributes and the key attribute. Clearly indicate on the diagram any full, partial or transitive dependencies and state any assumptions made. *(8)*

(ii) Normalise the `courier` table to third normal form, explaining the steps involved at each stage. *(8)*

QUERY PROCESSING AND RELATIONAL ALGEBRA

CT230
Database
Systems I

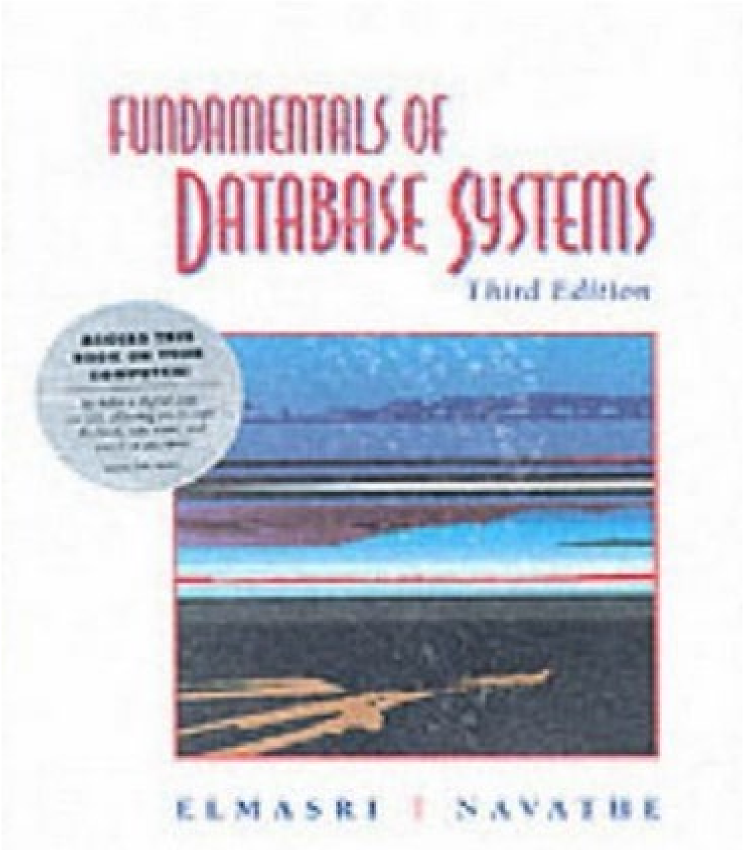
RECOMMENDED TEXT:

See:

Chapter 18

Elmasri & Navathe

(3rd Edition)



DEFINITION: Query Processing

Transforms SQL (high level language) in to a **correct** and **efficient** low level language representation of relational algebra.

Each relational algebra operator has code associated with it (a program) which, when run, performs the operation on the data specified, allowing the specified data to be output as the result.

Steps Involved in Processing a SQL Query:

- Process (Parse and Translate) and create an internal representation of the query – may be an Operator Tree, **Query tree** or Query graph (for more complicated queries).
- Optimise.
- Execute/Evaluate returning results.

How to Translate SQL to Relational Algebra?

Must have:

- a meaningful set of relational algebra operators (today's lecture).
- a mapping (translation) between SQL code and relational algebra expressions.

RELATIONAL ALGEBRA

Two formal languages exist for the relational model:

- Relational algebra (procedural)
- Relational calculus (non-procedural)

Both are logically equivalent

Note: the *practical/implementation* language of the relational model is SQL (as we have seen)

Relational Algebra Operations

π σ ρ ← → τ γ ∧ ∨ ¬ = ≠ ≥ ≤ ∩ ∪ ÷ - × ⋈ ⟕ ⟖ ⟗ ⋉ ⋊ ▷

- A basic set of operations exist for the relational model.
- These allow for the specification of basic retrieval requests.
- A sequence of relational algebra (RA) operations forms a relational algebra expression.
- RA operations are divided into two groups:
  - operations based on mathematical set theory (e.g., union, product etc.)
  - specific relational database operations.

RELATIONAL ALGEBRA versus SQL

The core operations and functions (i.e., programs) in the internal modules of most relational database systems are based on relational algebra.

SQL is a declarative language …. It allows you specify the results you require … not the order of the operations to retrieve those results.

Relational Algebra is *procedural* - must specify exactly *how* to retrieve results when using relational algebra.

RELATIONAL ALGEBRA EXPRESSIONS

- A valid relational algebra expression is built by connecting tables or expressions with defined unary and binary operators and their arguments (if applicable)
- Temporary relations resulting from a relational algebra expression can be used as input to a new relational algebra expression
- Expressions in brackets are evaluated first
- Relational Algebra operators are either Unary or Binary

Relational Algebra: UNARY OPERATORS

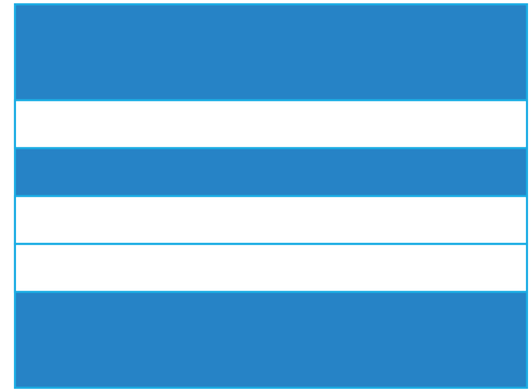
- Selection
- Projection
- Rename
- Order
- Group

Each operation:

- takes one relation (table) or expression as input
- gives a new relation as a result

Selection operator

σ (sigma)



Used to select certain tuples (rows) from a relation R

Notation: $\sigma_p R$

where:

p: selection predicate i.e., *a condition*

R: relation/table name

NOTE:

The Selection (σ) operator in relational algebra is NOT the same as the `SELECT` clause in an SQL query.

A SQL `SELECT` query could be equivalent to a combination of relational algebra operators (σ, π and `JOIN`)

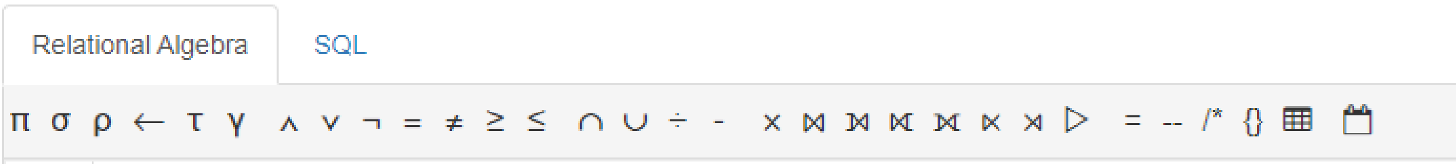
EXAMPLE 1 (using company schema): Find the projects with pno = 10 and hours worked < 20

$\sigma_{\text{(hours < 20 AND pno = 10)}}$ works_on

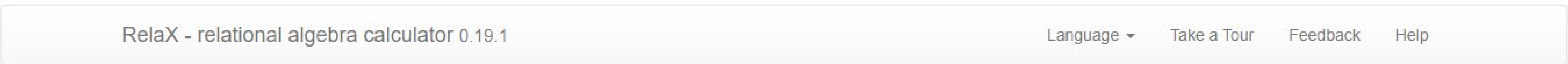
sigma$_{\text{(hours < 20 AND pno = 10)}}$ works_on

Returns the set:

{ (333445555, 10, 10.0), (999887777, 10, 10.0)}



WORKING WITH THE RelaX CALCULATOR



There is no standard language for relational algebra like there is for SQL.

One University group have developed a calculator that supports a *fairly* common standard.

Note that it is CASE SENSITIVE.

Provides a number of datasets with the option of also using your own dataset.

We will load in a version of the COMPANY schema

LOAD A DATASET:

Calculator: https://dbis-uibk.github.io/relax/calc/local/uibk/local/0

Go to “”Group Editor” Tab

Copy text from file on Blackboard and add

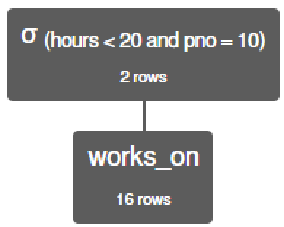
Then choose “Preview”

Then choose “Use group in Editor”

*Note: only stored temporarily

Example 1 in RelaX calculator:

Find the projects with pno = 10 and hours worked < 20

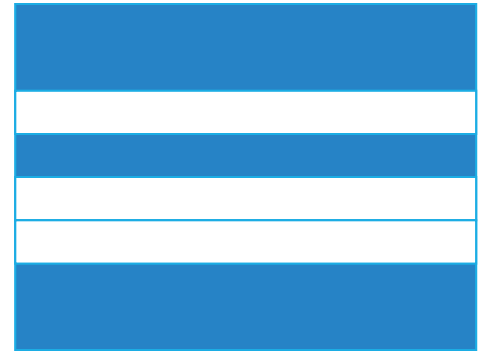


$\sigma_{(hours < 20 \text{ and } pno = 10)}$ works_on

Execution time: 0 ms

works_on.essn	works_on.pno	works_on.hours
333445555	10	10
999887777	10	10

NOTE:



- The **degree** of the relation resulting from a selection of table R is the same as the degree of R, e.g., *same number of attributes/columns*

The operation is **commutative**, i.e. a sequence of selects can be applied in any order,

e.g.

$\sigma_{(\texttt{hours < 20 and pno = 10})}$ `works_on`

$\sigma_{(\texttt{pno = 10 and hours < 20})}$ `works_on`

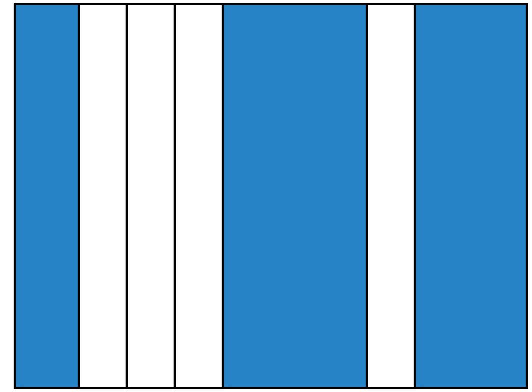
EXAMPLE 2: (Using company database):
List the department numbers of departments located in Houston

```
σ (dlocation = 'Houston') dept_locations
```

or can write as:

```
sigma (dlocation = 'Houston') dept_locations
```

PROJECTION OPERATOR
π *Pi*



Used to return certain attributes/columns

Notation: $\pi_{A1,\ A2,\ \ldots,\ Ak}$**(R)**

where:

$A_1 \ldots A_k$ attribute names

R: relation/table name

Result is a relation with the *k* attributes listed in same order as they appear in list. Duplicate tuples are removed from the result.

** NOTE: Commutativity does *not* hold.

EXAMPLE 3: (Company schema):

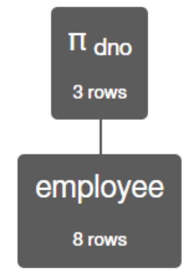
List all the department numbers where employees work

π dno employee

or can write as:

Pi dno employee

Returns:{5, 4, 1}

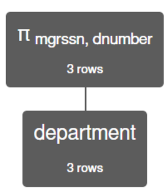


π_{dno} employee

employee.dno
5
4
1

EXAMPLE 4: List all managers (ssn) and the departments (number) they manage

π mgrssn, dnumber department



$\pi_{\text{mgrssn, dnumber}}$ department

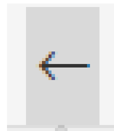
department.mgrssn	department.dnumber
333445555	5
987654321	4
888665555	1

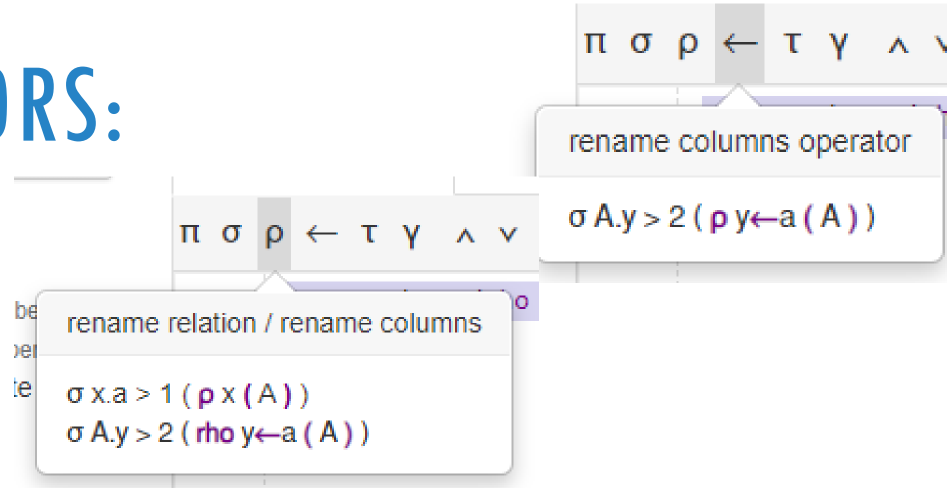
YOU TRY …

Example 5 Return all project locations which are in dept 5

Example 6 Return the names of all employees in department 5

Example 7. List the names of all employees whose salary is greater than 45000

RENAME OPERATORS: RHO *Ρ AND* 



Rename Operation (ρ)

Notation – ρ_{x} (E)

Where the result of expression **E** is saved with name of **x**

You might want to do this to save typing a table name,

e.g., for table dependent might want to rename it as dep as follows:

```
π dep.bdate (rho dep (dependent))
```

NOTE: ASSIGNMENT ALSO AVAILABLE

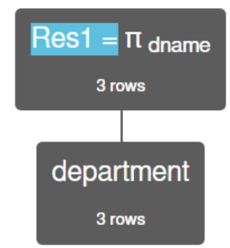
BUT NOT A RELATIONAL ALGEBRA OPERATOR

-- definition

Res1 = π dname department

-- execution

Res1



π_{dname} department

department.dname
'Research'
'Administration'
'Headquarters'

Order operator

τ (tau)

Used to order by certain columns from a relation R

Notation: $\tau_{A1,\ A2,\ \ldots,\ Ak}\ R$

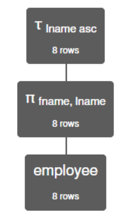
where:

`A1, A2, …, Ak` : are attributes with either asc or desc

R: relation/table name

EXAMPLE 8: (Company schema):
List all the employee first names and surnames, ordered by surname (asc)

τ lname asc (π fname, lname employee)



τ lname asc (π fname, lname employee)

employee.fname	employee.lname
'James'	'Borg'
'Joyce'	'English'
'Ahmad'	'Jabbar'
'Ramesh'	'Narayan'
'John'	'Smith'
'Jennifer'	'Wallace'
'Franklin'	'Wong'
'Alicia'	'Zelaya'

or can write as:

tau lname asc (π fname, lname employee)

asc is default ordering

Group By operator

γ (gamma)

Used to group by certain columns from a relation R

AGGREGATE FUNCTIONS SUPPORTED
(THOUGH NOT PART OF RELATIONAL ALGEBRA)

COUNT(*)

COUNT(column)

MIN(column)

MAX(column)

SUM(column)

AVG(column)

BINARY OPERATORS

General Syntax:

(child_expression) function argument (child_expression)

UNION OPERATOR: ∪

Notation: (R) ∪ (S)

where R and S are relations/tables

Returns all tuples from R and all tuples from S

Notes:

- No duplicates will be returned.

INTERSECTION OPERATOR: n

Notation: (R) n (S)

where R and S are relations/tables

Result: returns all tuples from R that are also in S.

SET DIFFERENCE: -

Notation: (R) – (S)

where R and S are relations/tables

Result:

returns tuples that are in relation R but not in S

Note: (R) – (S) and (S) – (R) are not the same

UNION COMPATIBILITY

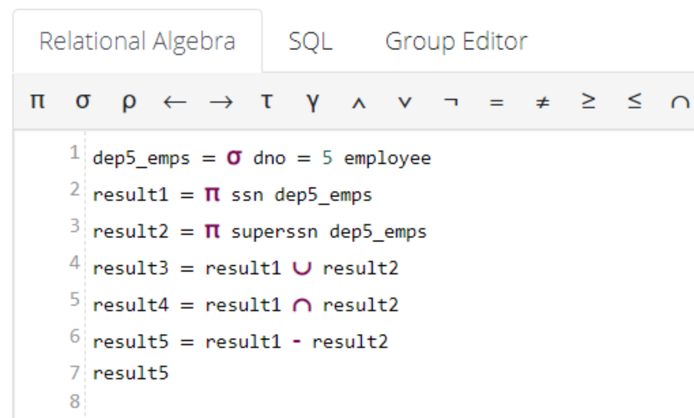
For union, intersection and minus, relations must be **union compatible**, that is:

- schemas of relations must match, i.e., same number of attributes and each corresponding attributes have the same domain

EXAMPLE 9:

What is displayed in the results relation following these operations?
(using ReLaX schema)

dep5_emps = σ dno = 5 employee
result1 = π ssn dep5_emps
result2 = π superssn dep5_emps
result3 = result1 ∪ result2
result4 = result1 ∩ result2
result5 = result1 - result2
result5



EXAMPLE 9: *ctd.*

`result1`

ssn
123456789
333445555
666884444
453453453

`result2`

superssn
333445555
888665555

`result1 u result2`

ssn
123456789
333445555
666884444
453453453
888665555

EXAMPLE 9 *ctd.*

`result1`

ssn
123456789
333445555
666884444
453453453

`result2`

superssn
333445555
888665555

`result1 n result2`

ssn
333445555

EXAMPLE 8 *ctd.*

`result1`

ssn
123456789
333445555
666884444
453453453

`result2`

superssn
333445555
888665555

`result1-result2`

ssn
123456789
666884444
453453453

CARTESIAN PRODUCT OPERATOR: X (cross join)

Notation: (R) x (S) where R and S are relations/tables

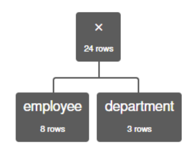
Returns: tuples comprising the concatenation (combination) of every tuple in R with every tuple in S

Note:

No condition is specified

Example:

```
employee x department
```



employee × department

Execution time: 2 ms

employee.fname	employee.minit	employee.lname	employee.ssn	employee.bdate	employee.address	employee.gender	employee.salary	empl
'John'	'B'	'Smith'	123456789	'1975-Jan-09'	'731 Fondren, Houston, TX'	'Man'	55250	
'John'	'B'	'Smith'	123456789	'1975-Jan-09'	'731 Fondren, Houston, TX'	'Man'	55250	
'John'	'B'	'Smith'	123456789	'1975-Jan-09'	'731 Fondren, Houston, TX'	'Man'	55250	
'Franklin'	'T'	'Wong'	333445555	'1980-Dec-08'	'638 Voss, Houston, TX'	'Man'	65000	
'Franklin'	'T'	'Wong'	333445555	'1980-Dec-08'	'638 Voss	'Man'	65000	

EXAMPLE 10:

Given relations: **R**(A, B) and **S**(C, D, E):

A	B
1	2
3	4

C	D	E
22	55	66
44	77	88
99	10	11

Then R x S is?

R

A	B
1	2
3	4

S

C	D	E
22	55	66
44	77	88
99	10	11

R x S =

A	B	C	D	E
1	2	22	55	66
1	2	44	77	88
1	2	99	10	11
3	4	22	55	66
3	4	44	77	88
3	4	99	10	11

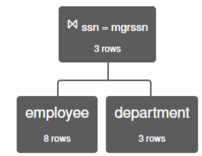
JOIN OPERATOR: ⋈

The Join operator is a hybrid operator – it is a combination of the Cartesian product operator (x) and a select operator (σ)

Tables are joined together based on the **condition** specified

Example:

`employee ⋈ ssn = mgrssn department`



employee ⋈ $_{ssn\ =\ mgrssn}$ department

employee.lname	employee.ssn	employee.bdate	employee.addre
'Wong'	333445555	'1955-Dec-08'	'638 Voss, Houston, TX'
'Wallace'	987654321	'1941-Jun-20'	'291 Berry, Bellaire, TX'
[illegible]	[illegible]	[illegible]	[illegible]

Cartesian product versus Join?

The main difference between a Cartesian product operator and a join operator is that with a join, only tuples **satisfying a condition** appear in the result (as we have already seen)

In a Cartesian product operator, all combinations of tuples are included in the result.

EQUI AND THETA JOINS

Notation: (**R1**) ⋈ p (**R2**)

where:

p: Join condition

R1 and R2: relations/tables

Result: The JOIN operation returns all combinations of tuples from relation R1 and relation R2 satisfying the join condition p

Note:

EQUI JOINS use only equality comparisons (=) in the join condition p

EXTRA EXAMPLES ....

11. Write the relational algebra expression to find the names of the employees in the Research department
12. Find the name(s) of Jennifer Wallace's dependents
13. Find the name(s) of employees who work on projects which are located in Houston

SUMMARY

Important to know:

- Unary relational algebra operators and how they work – especially, σ and π
- Binary relational algebra operators and how they work – especially x and $\bowtie$
- How to combine binary operators (where order is significant) to answer a question
- Using the ReLaX calculator

VERY Important <u>not</u> to confuse SQL and Relational Algebra

QUERY PROCESSING AND OPTIMISATION

CT230
Database
Systems I

RECALL: Definition of Query Processing

Transforms SQL (high level language) in to a **correct** and **efficient** low level language representation of relational algebra

Each relational algebra operator has code associated with it which, when run, performs the operation on the data specified, allowing the specified data to be output as the result

Representing the relational algebra solutions with a query tree

What is a tree?

A tree is a <u>collection of data</u> arranged as a finite set of elements - called **nodes** - such that:

The tree is empty or the tree contains a distinguished node, called the **root node**, and all other nodes are arranged in subtrees such that each node has a parent node. Nodes typically contain *data* and some pointers to other nodes

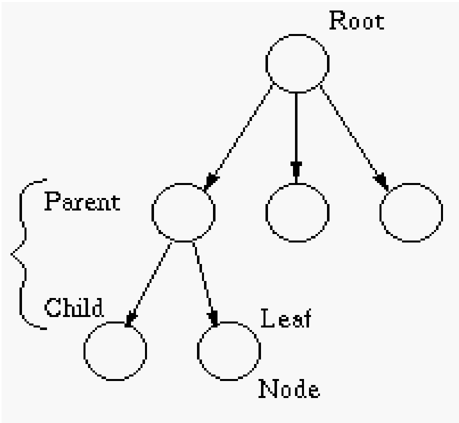


Figure: Tree data structure

TREES

Nodes may be:

root: no node points to it

inner: has parent and child nodes

leaves: has no child nodes

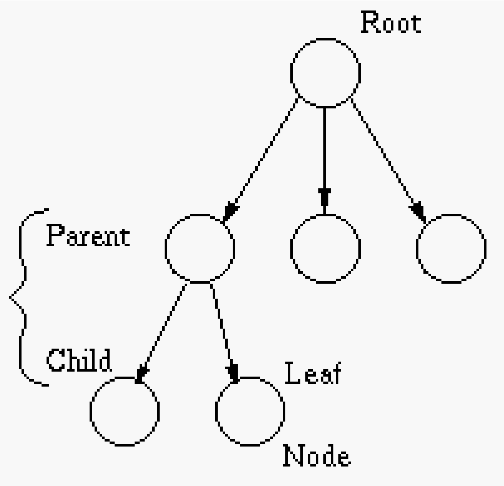


Figure: Tree data structure

Tree data structures (a grouping of data) are used frequently in computing allowing data to be stored in a non-linear (non-list) way.

They are often (but not always) **binary trees** where each node can have at most two child nodes

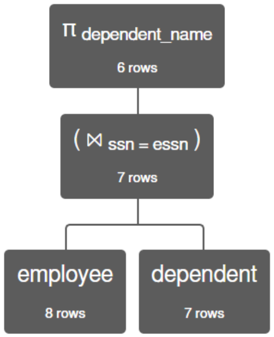
QUERY TREE

A **query tree** is a binary tree that corresponds to a relational algebra expression where:

- (input): tables are at the leaf nodes
- relational algebra operators are at internal nodes
- (output/result): the root of the tree returns the result (often with one final relational algebra operator)

The sequence of operations is directed from leaves to root and from left to right – e.g. the bottom-most, left-most side of tree is executed first

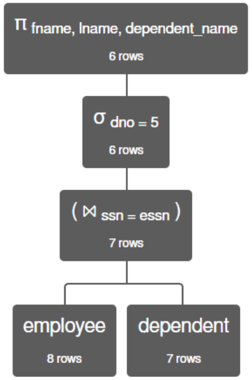
EXAMPLES: all dependent names



$\pi_{\text{dependent\_name}} (\text{employee} \bowtie_{\text{ssn = essn}} \text{dependent})$

dependent.dependent_name
'Michael'
'Alice'
'Elizabeth'
'Theodore'
'Joy'
'Abner'

EXAMPLES:
employees from department 5 and their dependents



$\pi_{\text{fname, lname, dependent\_name}} (\sigma_{\text{dno} = 5} (\text{employee} \bowtie_{\text{ssn} = \text{essn}} \text{dependent}))$

employee.fname	employee.lname	dependent.dependent_name
'John'	'Smith'	'Michael'
'John'	'Smith'	'Alice'
'John'	'Smith'	'Elizabeth'
'Franklin'	'Wong'	'Alice'
'Franklin'	'Wong'	'Theodore'
'Franklin'	'Wong'	'Joy'

How to Translate SQL to Relational Algebra?

- SELECT *attributes* corresponds to π
- Joins correspond to relational algebra joins $\bowtie$ with any join conditions specified as part of the join
- Any conditions in a WHERE clause correspond to a sigma σ relational algebra operator with associated conditions
- In addition, have rules for aggregate functions (sum, avg, count, etc.) and GROUP BY, HAVING and subqueries but we won't consider these

Executing query represented by query tree: one approach: Materialization Evaluation

Traverse tree from bottom to top, left to right. At each stage:

- Execute internal node operation whenever data for its child nodes are available
- Replace the internal node operation (and all child nodes) by the table resulting from executing the operation

Note: Results of operations are saved as temporary tables and are used as inputs to other operators

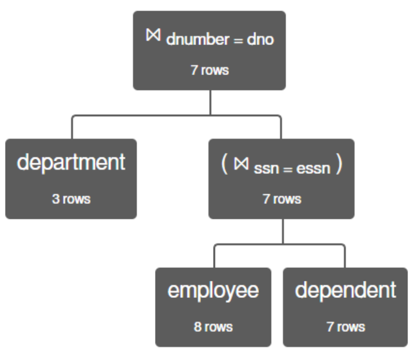
HOW TO DRAW A QUERY TREE?

Must remember the order of execution – from bottom to top, completing each level and then left to right of tree – therefore:

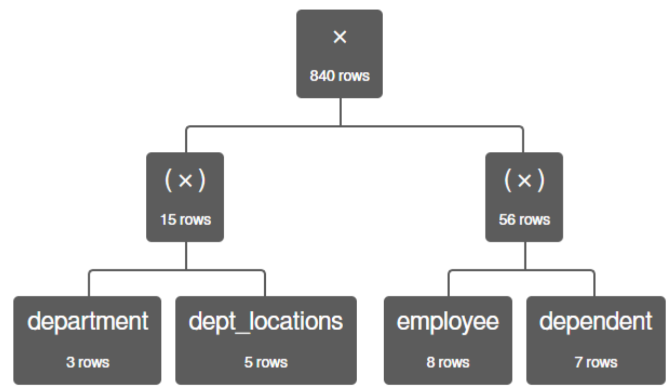
- the first operations – fetching tables – should be at the leaves of trees.
- the last operator – often π or aggregate functions - should be at the root of the table.
- joins must be applied to tables (2 at a time) and should be at internal nodes.
- any other operators should be at one or more internal nodes.

IMPORTANT

When Joining or multiplying more than two tables … operators can only be applied to 2 operands at a time



department $\bowtie_{dnumber = dno}$ (employee $\bowtie_{ssn = essn}$ dependent)



(department × dept_locations) × (employee × dependent)

ANNOTATING TREE

Each relation algebra operation can be evaluated using one of several different algorithms and each relational algebra expression can be evaluated in many ways.

** An **evaluation plan** is an annotated expression/query tree specifying the execution strategy for a query.

EXAMPLE 1

Consider the following SQL solution and relational algebra translation

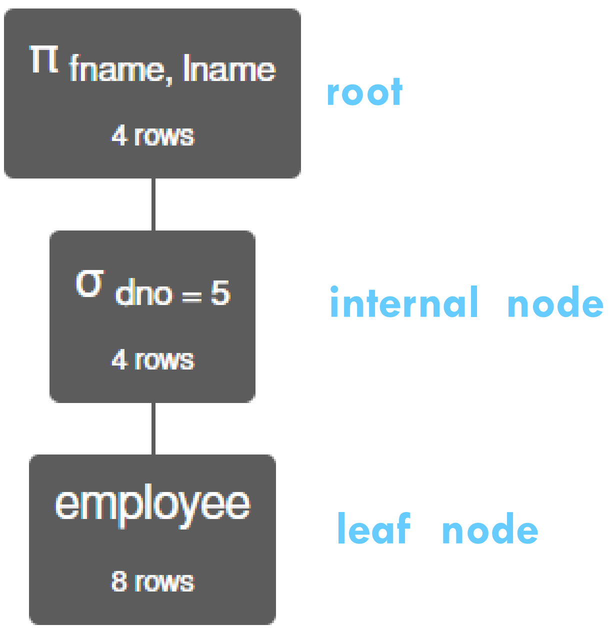
```
SELECT fname, lname
FROM   employee
WHERE  dno = 5;
```

$$\pi_{fname,\ lname}(\sigma_{dno\ =\ 5}\ employee)$$

Query tree representation

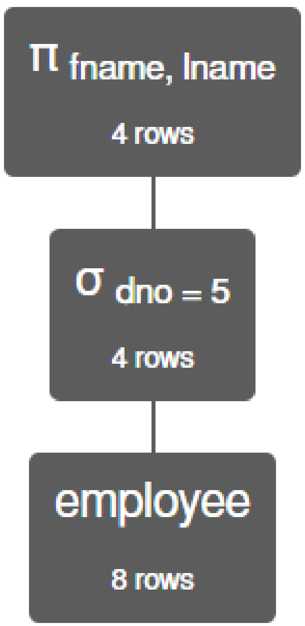
```sql
SELECT fname, lname
FROM   employee
WHERE  dno = 5;
```

$$\pi_{\text{fname, lname}} (\sigma_{\text{dno = 5}} \text{ employee})$$



employee.fname	employee.lname
'John'	'Smith'
'Franklin'	'Wong'
'Ramesh'	'Narayan'
'Joyce'	'English'

Query tree representation with evaluation plan

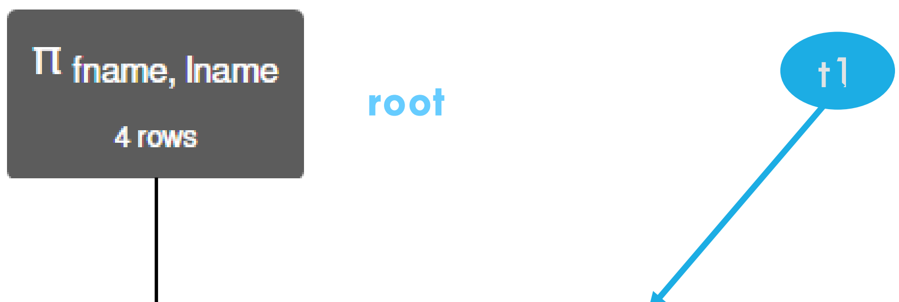


for each tuple in t1 retrieve fname, lname

linear search on condition. -write to t1

file scan: employee

How materialization evaluation works ...



employee.fname	employee.minit	employee.lname	employee.ssn	employee.bdate	employee.address	employee.sex	employee.salary	employee.superssn	employee.dno
'John'	'B'	'Smith'	123456789	'1965-Jan-09'	'731 Fondren, Houston, TX'	'M'	30000	333445555	5
'Franklin'	'T'	'Wong'	333445555	'1955-Dec-08'	'638 Voss, Houston, TX'	'M'	40000	888665555	5
'Ramesh'	'K'	'Narayan'	666884444	'1962-Sep-15'	'975 Fire Oak, Humble, TX'	'M'	38000	333445555	5
'Joyce'	'A'	'English'	453453453	'1972-Jul-31'	'5631 Rice, Houston, TX'	'F'	25000	333445555	5

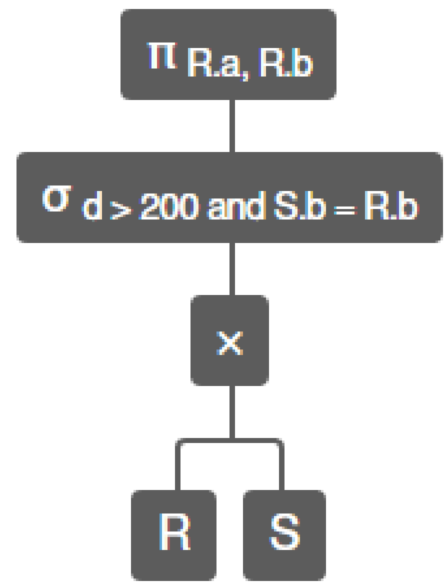
Example 2
UBIK database
https://dbis-uibk.github.io/relax/calc/local/uibk/local/0

Consider the following SQL query:

```
SELECT R.a, R.b
FROM   R, S
WHERE d > 200 AND S.b=R.b
```

And the relational algebra translation:

$$\pi_{R.a,\ R.b}\ \sigma_{d > 200 \text{ and } S.b = R.b}\ R \times S$$



R.a	R.b
3	c

Example 3 UBIK database

R.a	R.b
3	c

Consider the following SQL query:

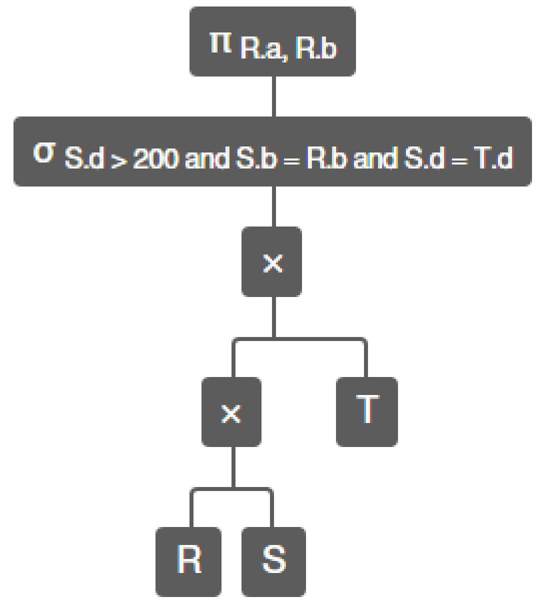
SELECT R.a, R.b

FROM R, S, T

WHERE S.d > 200 AND

S.b=R.b AND

S.d = T.d



And the relational algebra translation:

$$\pi_{R.a,\ R.b}\ \sigma_{S.d > 200 \text{ and } S.b = R.b \text{ and } S.d = T.d}\ R \times S \times T$$

EXAMPLE 4:
Translating SELECT FROM WHERE (with no subqueries) to Relational Algebra

Given a general SELECT statement of the form:

SELECT *attributeList*

FROM R1 INNER JOIN R2 ON *joinCondition*

WHERE *condition*

translates to:

$$\pi_{\texttt{attributeList}}\ (\sigma_{\texttt{condition}}(\texttt{R1 JOIN}_{\texttt{joinCondition}}\ \texttt{R2}))$$

NOTE: An SQL statement may have many equivalent relational algebra expressions.

Example 5: Consider the following (Company Schema):

List all salaries greater than 50000

The SQL solution:

```
SELECT salary
FROM  employee
WHERE salary > 50000;
```

Translating this SQL to Relational Algebra

```
SELECT salary
FROM employee
WHERE salary >50000;
```

Option 1:

$\pi_{salary} (\sigma_{(salary > 50000)} employee))$

retrieve tuples with salary > 50000

retrieve salary column

Option 2:

$\sigma_{(salary > 50000)} (\pi_{salary} employee)$

retrieve salary column

retrieve tuples with salary > 50000

DIFFERENCES BETWEEN THESE?

π_{salary} ($\sigma_{(salary> 50000)}$ employee))

$\sigma_{(salary > 50000)}$ (π_{salary} employee)

EXAMPLE 6:

Given the following problem based on the Company schema write the associated SQL code (using joins), a correct relational algebra expression translation and a query tree representing the relational algebra expression:

List the names of all employees who work on projects located in Stafford

EXAMPLE 7:

Given the following problem based on the Company schema write the associated SQL code (using joins), a correct relational algebra expression translation and a query tree representing the relational algebra expression:

List the location of all departments managed by manager Franklin Wong

ISSUES TO CONSIDER WITH QUERY TREES:

- Size of temporary tables
- Algorithms used for execution plan

OPTIMISATION

- Different query trees for a given query can have different *costs*
- Different evaluation plans for a given query can have different *costs*
- Optimisation techniques attempt to choose the best among a number of potential query trees

APPROACH 1:
Compare cost estimates across different solutions

- Cost is usually measured as the total elapsed time for answering a query
- One approach is to calculate cost estimates for each possible query tree
- The query tree with the lowest cost estimate should then be chosen

How to calculate cost estimates?

Cost factors include CPU speed, disk access time, network communication time, etc.

Disk access is typically the predominant cost and can be measured by number of blocks read/number of blocks written per query.

MAIN COST ESTIMATE USED:
Number of block transfers where each block contains a number of records

Number of blocks transferred from disk depends on:

- Size of buffer in main memory - having more memory reduces need for more disk accesses.
- Indexing structures used (primary, secondary, etc.)
- Whether all blocks of a file must be transferred or not
  - e.g., if search can be done on primary key of index file or on secondary index then only retrieve blocks that satisfy search condition

- As is typical in Computing, often use worst case estimates, knowing that any *actual* cost cannot exceed a worst case estimate.

DBMS CATALOG

The DBMS catalog stores statistical information about each table such as table sizes, indexes (and their depths) etc.

The statistical information on the tables and attributes used in a query, can be found in the *DBMS catalog* and these are used to calculate cost estimates also.

In DBMS catalog, for each table R information is stored on:

- Number of tuples/records in table R
- Number of blocks containing tuples of table R
- Size of a record in bytes
- Blocking factor
- Information on number of distinct values per attribute and number of values that would satisfy set of equality operations on attribute (by having averages, min, max, etc.)
- Information on indices (index types, index field values, etc.)

STEPS FOR APPROACH 1

1. Generate query trees and evaluation plans (maybe not all)

2. For each query tree get cost estimates using DBMS catalog

Resulting in a set of cost estimates such that the best can be chosen and the query tree with the lowest cost estimate can then be picked as the single best query tree and evaluation plan.

THEREFORE:

To choose among plans, the optimiser has to estimate cost of each evaluation plan.

Two aspects to this:

For each node of tree:

- estimate cost of performing associated operation
- estimate size of result and if it is sorted

APPROACH 1: SUMMARY

- Cost-based optimisation, while good, is expensive:

  As query complexity increases so does the different number of query trees and plans possible and each query tree requires its own cost estimates

N.B. It is important that the amount of time an optimiser spends on calculating the best solution (optimising) is not longer than the amount of time which would elapse if executing a solution picked at random

APPROACH 2:
Heuristic Optimisation

- Optimiser often uses heuristics to reduce the number of choices that must be made in a cost-based fashion.
- Heuristic optimisation transforms the query-tree by using a set of rules that typically (but not always) improve execution performance.
- Some cost based estimation is also performed – as part of the heuristic optimisation and to choose between a reduced set of trees and/or evaluation plans.

STEPS FOR APPROACH 2:

1. Create a *canonical query tree.*

2. Apply a *set of heuristics* to the tree to create a more efficient query tree.

3. Create cost estimates of this query tree, if appropriate, to ensure best evaluation plan.

DEFINITION:
Canonical query tree

A canonical query tree is an *inefficient* query tree representing relational algebra expressions which can be created directly from the SQL solution following a sequence of quick and easy steps:

- Uses CARTESIAN product instead of JOINS
- Keeps all conditions (σ) together in one internal node
- π becomes root node

Steps to create a canonical query tree with SELECT/FROM/WHERE clauses and no sub-queries:

1. All relations in FROM clause become leafs of the tree. They should be combined with a Cartesian product (x) of the relations.

* *Remember:* Only 2 relations can be involved in a Cartesian product at a time (binary tree)

2. All conditions in the WHERE clause and any JOIN conditions in WHERE or FROM clause become a sequence of relational algebra expressions in **one** inner node of the tree (with inputs from previous step)

3. All conditions from the SELECT clause become a relational algebra expression in the root node

EXAMPLE 8 *with implicit join*

List the names of employees in research department

```
SELECT fname, lname
FROM   employee, department
WHERE  dno = dnumber AND
       dname = 'Research';
```

Creating the **canonical query tree** …

EXAMPLE 8 *with explicit join*

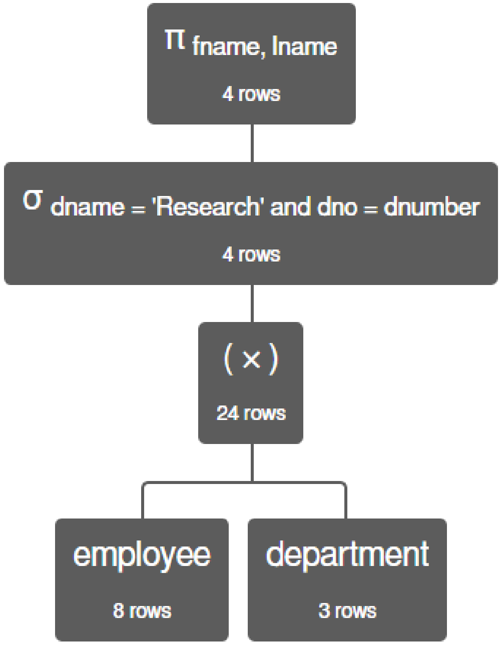
List the names of employees in research department

```
SELECT fname, lname
FROM   employee INNER JOIN department ON
       dno = dnumber
WHERE  dname = 'Research';
```

Creating the **canonical query tree** …

CANONICAL TREE REPRESENTATION:

```
SELECT  fname, lname
 FROM   employee INNER JOIN department
        ON dno = dnumber
 WHERE  dname = 'Research';
```



NOTE:

This would be very inefficient if executed directly because of the Cartesian product operations.

Recall Cartesian product:

R x S

Returns tuples comprising the concatenation of every tuple in R with every tuple in S

CONSIDER EXAMPLE 7 AGAIN

Draw the canonical query tree for the SQL query in Example 7:

List the location of all departments managed by manager Franklin Wong

HEURISTIC OPTIMISATION

Heuristic Optimisation **MUST** transform this canonical query tree into a final query tree that is efficient to execute:

- In general, heuristic optimisation tries to apply the most restrictive operators as early as possible in the tree (furthest down the tree) and to reduce the size of the temporary tables/results created that move "up" the tree.
- Heuristic Optimisation must include rules for equivalence among relational algebra expressions that can be applied to the initial tree.

HEURISTIC OPTIMISATION ALGORITHM:

Input: A canonical query tree

Process:

1. Decompose any σ with AND conditions into individual σ
2. Move each σ operator as far down the query tree as possible.
3. Rearrange the leaf nodes so that most restrictive σ can be applied first (using information from DBMS catalog) and so that future JOINS are possible.

Note: "most restrictive" means those operators that result in relations with the fewest tuples or with the smallest absolute size - these operations should happen first – that is – at the lowest level of the tree and on the left hand side of the tree.

4. Combine CARTESIAN PRODUCT operators with σ (sigma) to form JOIN operators where appropriate (replacing all x)

5. Decompose π and move each π as far down the tree as possible, possibly creating new π operators in the process.

(6. Identify subtrees that represent groups of operations that can be executed by a single algorithm.)

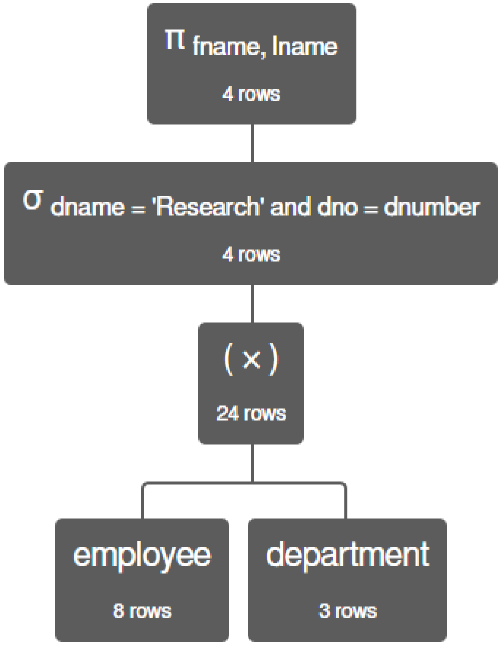
(7. Add evaluation plan)

Output: An efficient query tree

Back to EXAMPLE 8:

List the names of employees in research department

```
SELECT fname, lname
FROM   employee INNER JOIN department
       ON dno = dnumber
WHERE  dname = 'Research';
```

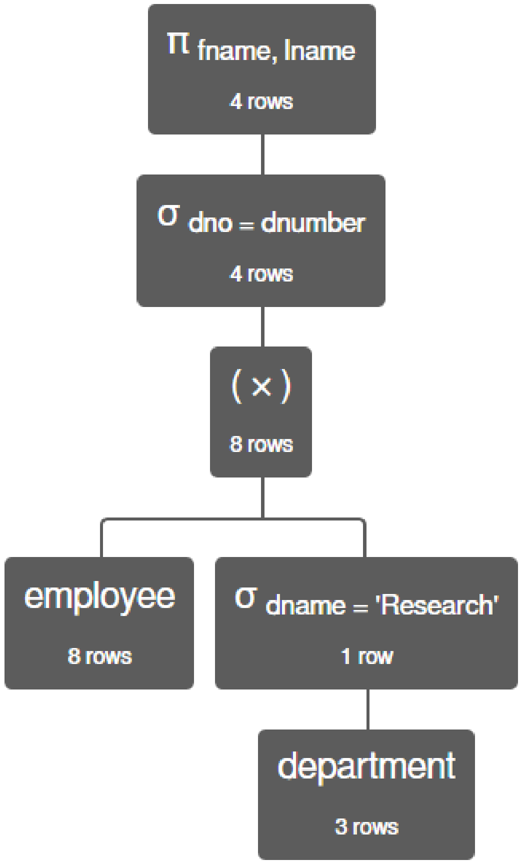


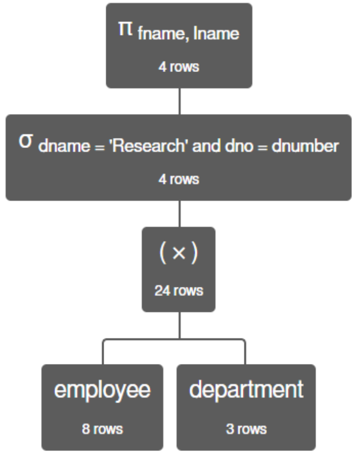
OPTIMISATION HEURISTIC 1 & 2:
Decompose conditions and apply sigma (σ) operators as early as possible

- **"Move σ down tree"** thus eliminating unwanted tuples.
- Heuristic 1 tries to reduce the size of the tables to be combined as much as possible:
- Therefore, if a selection operator (σ) occurs *after* a Cartesian product or a join, check to see if it can occur *before* these operations

Example 8:

Move (σ) sigma





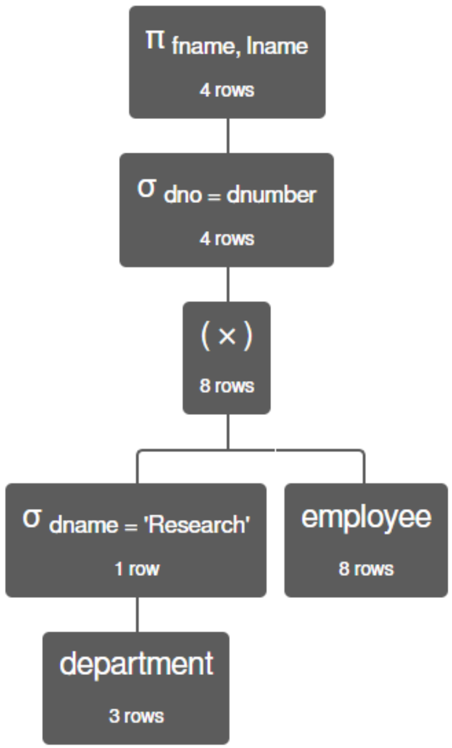
OPTIMISATION HEURISTIC 3:

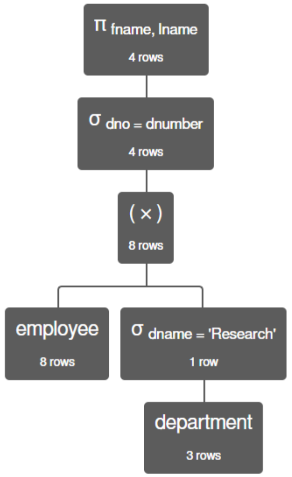
Rearrange the leaf nodes so that most restrictive sigma opeartors can be applied first

If we don't have any information from DBMS catalog

- we might leave nodes as they are
- Use database schema (number of columns) to make a good estimate
- Use sample data (number of rows) and database schema (number of columns) to make a good estimate

EXAMPLE 8:
REARRANGE LEAF NODES





OPTIMISATION HEURISTIC 4:
Replace Cartesian product (x) and appropriate selects (σ) with JOIN

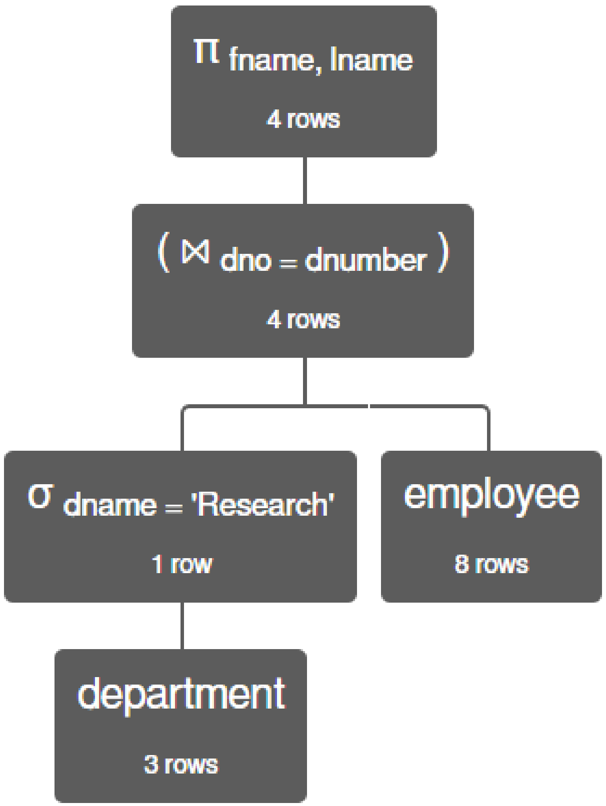
* First must ensure the leaf nodes are ordered such that this can happen – if not re-order leaf nodes and ensure to keep any select operators with the appropriate leaf node

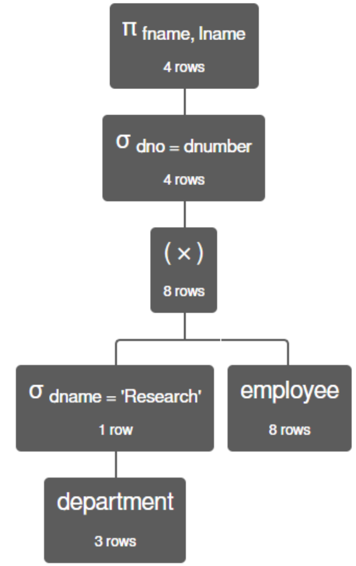
$$\sigma_{condition}(r1 \; X \; r2)$$

Is equivalent to:

$$R1 \; JOIN_{condition} \; R1$$

EXAMPLE 8:
REPLACE X





OPTIMISATION HEURISTIC 5:
Apply Pi (π) operators as early as possible

- **Motivation: "Move π down the tree"** (project) to eliminate unwanted columns

- The heuristic ensures that the size of the tables to be joined are as small as possible (reduces number of attributes/columns)

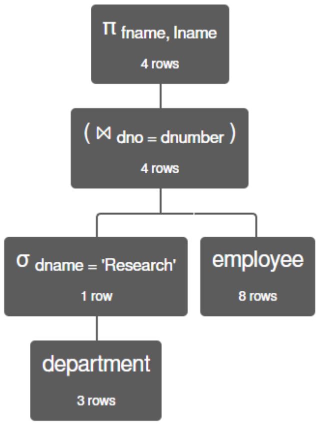
Therefore:

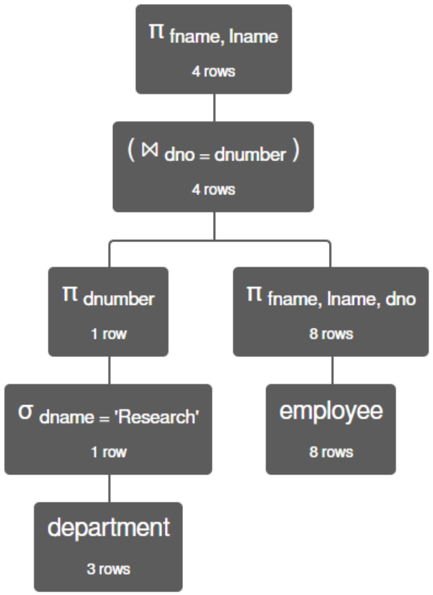
- for each π check if that π can be carried out before the join

- for each table check if additional π can be introduced (these may not be stated explicitly in the query)

N.B. MUST ensure that all needed columns further up in the tree are retained (even if they are not immediately necessary)

EXAMPLE 8:
Move Pi





$\pi_{\text{fname, lname}} ((\pi_{\text{dnumber}} \sigma_{\text{dname = 'Research'}} \text{department}) \bowtie_{\text{dno = dnumber}} \pi_{\text{fname, lname, dno}} \text{employee})$

EXAMPLE 9

Using the COMPANY relational schema and interpretation as defined in lectures develop an SQL query to satisfy the following information need:

"List the names of employees with salaries greater than 30000, who work on projects for greater than 25 hours where the projects are located in Houston or Bellaire"

Using query optimisation heuristics develop a query tree which represents an efficient evaluation strategy for the developed query.

SQL SOLUTION:

```
SELECT      fname, minit, lname
FROM        project, employee, works_on
WHERE       pno = pnumber AND essn = ssn AND
            hours > 25 AND salary > 30000 AND
            (plocation = ‘Houston’ OR
            plocation = ‘Bellaire’);
```

CANONICAL QUERY TREE SOLUTION

π fname, minit, lname

(σ pno = pnumber AND essn = ssn AND

hours > 25 AND salary > 30000 AND

plocation = 'Houston' OR plocation = 'Bellaire'

(project x employee x works_on)

)

OPTIMISATION HEURISTIC 1 & 2:

Decompose conditions and apply sigma (σ) operators as early as possible

OPTIMISATION HEURISTIC 3:

Rearrange the leaf nodes so that most restrictive sigma opeartors can be applied first and that future joins can be performed

OPTIMISATION HEURISTIC 4:

Replace Cartesian product (x) and appropriate selects (σ) with JOIN

OPTIMISATION HEURISTIC 5:

Apply Pi (π) operators as early as possible

EXAMPLE 10: (Winter 2017)

(Given the movie schema from the exam paper)

(c) Using joins, create a SQL query to answer the following information need. Using this SQL query, create a canonical query tree, explaining the steps you take in creating the tree and highlighting what parts of the SQL query are represented by the root, leaves and inner nodes of the tree.

For movies of genre 'Sci-Fi', released in 2016 or 2017, with an average rating greater than 7, list the movie title, movie category and the names of the actors who star in the movie.

(d) Using the canonical query tree from part (c), and with respect to *heuristic-based optimisation,* develop a query tree that represents an efficient evaluation strategy for the SQL query. Explain the steps taken, describing each heuristic used.

SCHEMA:

movie(id, title, relYear, category, runTime, director, studioName, description, rating)

actor(aID, fName, surname, gender)

stars(movieID, actorID)

movGenre(movieID, genre)

For movies of genre 'Sci-Fi', released in 2016 or 2017, with an average rating greater than 7, list the movie title, movie category and the names of the actors who star in the movie.

SQL SOLUTION:
(Note: can use implicit or explicit joins)

```
SELECT title, category, fname, surname
FROM   movie INNER JOIN movGenre ON id = movieGenre.movieID
       INNER JOIN stars ON id = stars.movieID
       INNER JOIN actor ON aid = actorID
WHERE  genre = 'Sci Fi' AND
       rating > 7 AND
       (relYear = 2016 OR relYear = 2017);
```

SUMMARY: IMPORTANT TO KNOW

- Basic relational algebra operators.
- Mapping between relational algebra operators and SQL.
- Mapping between relational algebra expression and query tree.
- Mapping from SQL to Canonical Query tree.
- Heuristic optimisation steps to map Canonical Query tree to efficient query tree.
- *N.B. Do not mix up SQL code and Relational Algebra expressions*

FILE ORGANISATIONS

CT230 Database Systems I

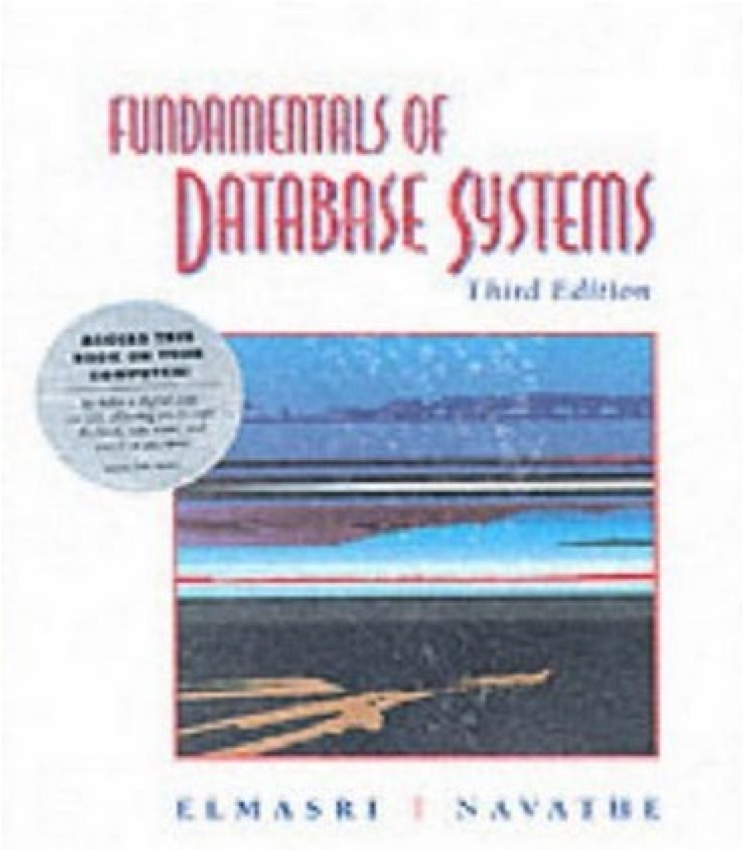
RECOMMENDED TEXT:

See:

Chapter 5

Elmasri & Navathe

(3rd Edition)



MOTIVATIONS

- Generally can assume for non-trivial relational databases, that the entire database will **not** fit in main memory (RAM)
- One of the DBMS's tasks is to manage the physical organisation (storage and retrieval) of the tuples (rows) in each table in the database
  - This is called File Organisation

NOTE:

Newer database system architectures, in-memory databases (such as SAP Hanna), manage their data through virtual memory, relying on the Operating System to manage the movement of data to and from main memory through the OS paging mechanism.

DEFINITION: FILE ORGANISATIONS

A database file organisation is the way tuples (records) from a table are physically arranged in secondary storage to facilitate storage of the data and read/write requests by users (via queries).

A number of factors to consider, including:

- Support of fast access of data – moving to/from secondary storage
- Cost
- Efficient use of secondary storage space
- Provision for table growth (when new tuples added)

Concerning the physical storage of tuples

- Options?
  - All stored together?
  - Separated in some way based on some *logical* grouping?

More definitions:

File = collection of data stored in bulk

In DBMS we have referred to these files as *tables* or *relations*

In DBMS we know that such tables contain a sequence of tuples, where each tuple contains a sequence of bytes and is subdivided into attributes or fields. Each attribute contains a specific piece of information. Associated with each attribute is a data type

In File Systems, we refer to these tuples as **records** containing **fields**

Size of records/tuples:

Fixed length: all records (tuples) in file (table) have exactly same size

Variable length: different records (tuples) in file (table) have different size

RECORDS

Each record often begins with a *header*, a fixed-length region which stores information about the record such as:

- Pointer to the database schema
- Length of the record
- Timestamp indicating the time the record was last modified or read
- Pointers to the fields of the record

File organisation issues:

How can these records be organised to:

- store in a compact manner on devices of limited capacity?
- provide convenient and quick access by programs

BLOCKS

- Different terminology used but generally,

  Block = Frame = Page

where records from a file are assigned to Blocks/Pages/Frames

- In relational DBMS use the terminology of a **block**
- Therefore, a table can also be defined as a collection of blocks where each block contains a collection of records.

DEFINITION: Blocks

- A block is the unit of data transfer between secondary storage and memory
- The block size **B** is fixed
- Records of a file must be allocated to blocks. Typically, the block size is larger than the record size, so each block will contain a number of records
- Some files may have very large records that cannot fit in one block so **span** records over a number of blocks
- A number of blocks is typically associated with a table

BLOCKS

Blocks also have header information holding information about the block such as:

- Links to one or more blocks associated with the table
- Which table (in the schema) the blocks belong to
- Timestamp of last access to block (read or write)

Example: Records assigned to blocks for the table with block header shown:

`dept_locations(dnumber, dlocation)`

Block 1

header	record 1	record 2	record 3	

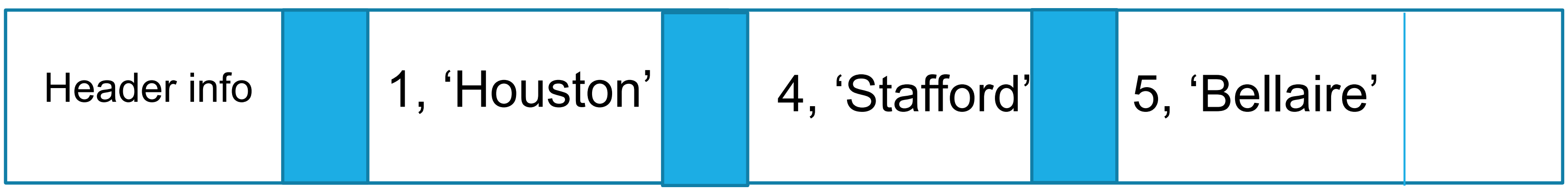
Block 2

header	record 4	record 5	

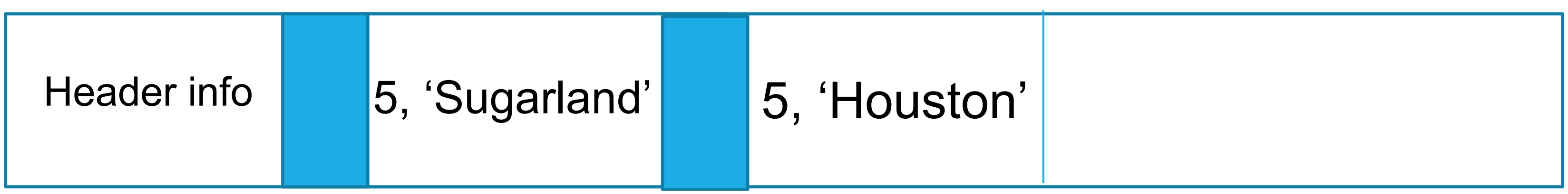
Example: Records assigned to blocks for the table with block and record header shown:

`dept_locations(dnumber, dlocation)`

Block 1



Block 2



DEFINITION: Blocking factor

- Blocking factor is the average number of records that fit per block
- Given block size B (in bytes), and record size R (in bytes), then with B >= R, can fit **floor(B/R)** records per block.
- Must ensure that the header information is also accounted for

Spanned vs Unspanned organisations:

Spanned organisation - records can span more than one block

Un-spanned - records are not allowed to cross block boundaries

So can only use when **B >= R**

(i.e., block size is greater than record size)

NOTE:

Block size and record size measured in bytes.

e.g., with unspanned memory organisation and

B = 1024 Bytes (once header information stored)

R = 100 Bytes and of fixed length

The blocking factor is:

```
floor(10.24) = 10
```

Why use blocking?

Say we need to retrieve a file with 1000 records …

- If not blocked then would need **1000 data transfers**
- If blocked with a blocking factor of 10, and records are stored one after another in blocks, then the same operation requires **100 data transfers**

EXAMPLE 1: A table has 20000 fixed-length STUDENT records

Schema:

```
student(name, studentID, address, mobphone,
 birthdate, gender, degreeCode, currentYear)
```

Each field is the following size:

```
name (30 bytes),
studentID (9 bytes),
address (40 bytes),
mobphone (10 bytes),
birthdate (10 bytes),
gender (1 byte),
degreeCode (8 bytes),
currentYear (4 bytes)
```

The file is stored on disk, in blocks, with 20 bytes required for header information per record.

EXAMPLE 1 QUESTIONS:

Each field is the following size:

```
name (30 bytes),
studentID (9 bytes),
address (40 bytes),
mobphone (10 bytes),
birthdate (10 bytes),
gender (1 byte),
degreeCode (8 bytes),
currentYear (4 bytes)
```

The file is stored on disk, in blocks, with 20 bytes required for header information per record.

What is the record size? (adding in the header information also)

30+9+40+10+10+1+8+4+20 = 132 bytes

Given a block size of 512 Bytes what is the blocking factor? (unspanned memory organisation)

512/132 = 3.87 ➡ blocking factor = 3

How many blocks are required to store all 20000 records if each block is filled before another block is used (remember records are fixed-length)

20000/3 = 6666.67 ➡ 6667 blocks needed

Operations performed on a file

All the operations we have been performing with SQL code:

- Scan or fetch all records
- Search records that satisfy an equality condition (i.e., find specific records)
- Search records where a value in the record is between a certain range
- Insert records
- Delete records

Steps to search for a record on a disk:

1. Locate relevant blocks
2. Move these blocks to main memory buffers
3. Search through block(s) looking for required record
4. At worst (*the worst case*), may have to retrieve and check through all blocks for the record

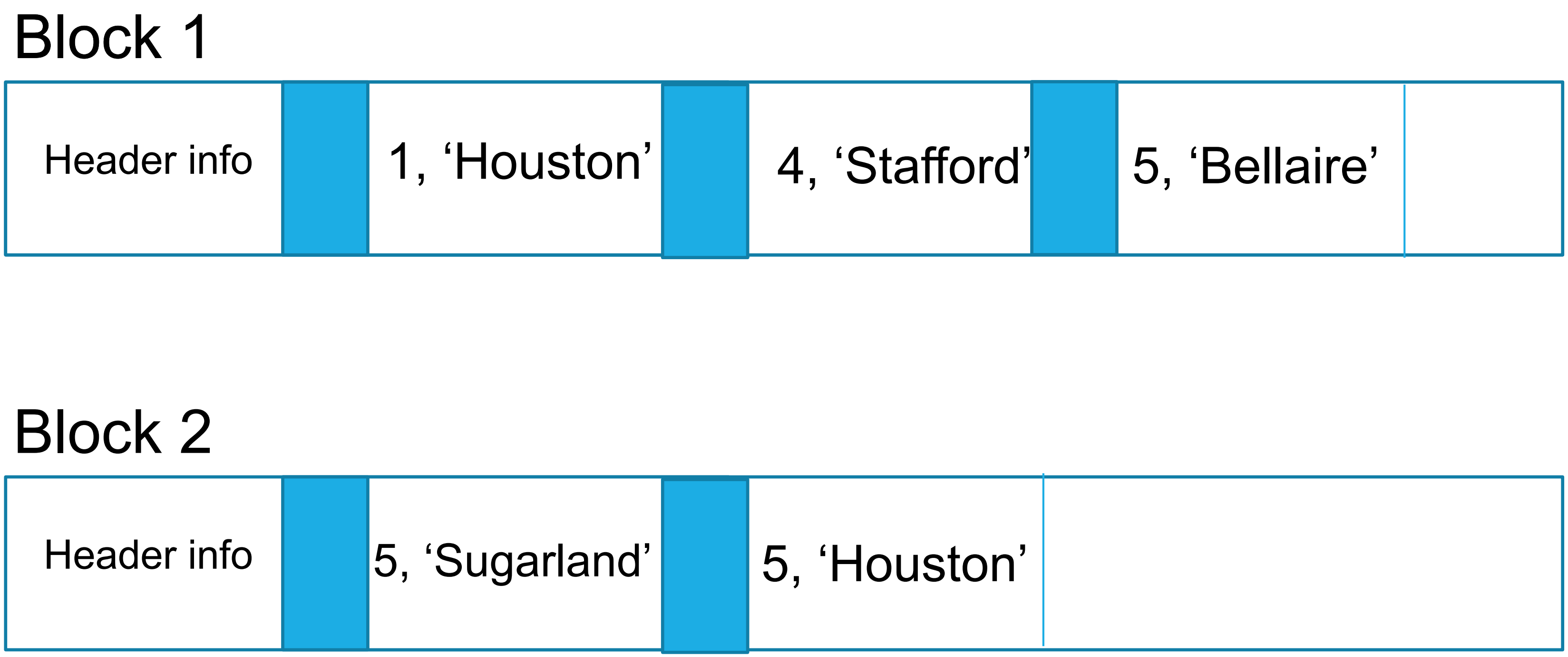
Generally, when accessing records:

To support record level operations, must:

- keep track of the *blocks* associated with a file
- keep track of *free space* on the blocks
- keep track of the *records* on a block

Recall example again: Records assigned to blocks for the table:

`dept_locations(dnumber, dlocation)`



Options for organising records?

- Heap file organisation (unordered)
- Sequential file organisation (ordered)
- Hashing/hashed file organisation
- Indexed file organisation (Primary, Clustered, B-Trees, B+ Trees)

HEAP FILE ORGANISATION

Approach: Any record can be placed in any block where there is space for the record (no ordering of records)

Insertion: last disk block associated with file (table) copied into buffer and record is added; block copied back to disk

Searching: must search all blocks (linear search)

Deletion: find block with record (linear search); delete link to record

EXAMPLE 2: Given a blocking factor of 2, and the student schema from example 1, sketch the placement of the following student records, in the order given, using heaped file organization

```
('Jane Casey', 111, '34 hazel park, newcastle, galway',
'087123456', '17-05-2001', 'F', 'GY101', 1)

('Jack Walsh ', 91, '13 college road, galway',
'086654321', '01-09-2000', 'M', 'GY350', 3)

('Sue Smyth ', 90, 'Maree, Oranmore, Co. Galway',
'087111222', '25-07-1999', 'F', 'GY406', 3)

('Gerard Kelly', 112, 'Main Street, Oughterard, Co.
Galway', '087121212', '30-12-2002', F, GY414, 1)
```

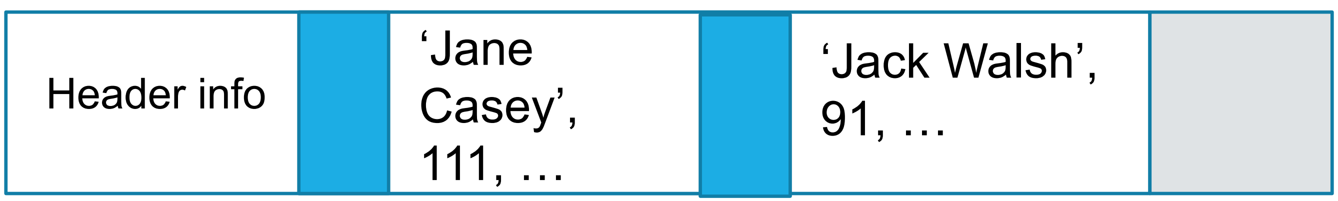
('Jane Casey', 111, '34 hazel park, newcastle, galway', '087123456', '17-05-2001', 'F', 'GY101', 1)

('Jack Walsh ', 91, '13 college road, galway', '086654321', '01-09-2000', 'M', 'GY350', 3)

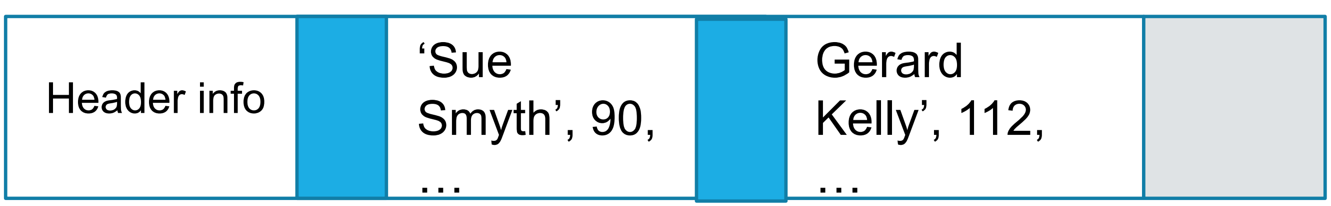
('Sue Smyth ', 90, 'Maree, Oranmore, Co. Galway', '087111222', '25-07-1999', 'F', 'GY406', 3)

('Gerard Kelly', 112, 'Main Street, Oughterard, Co. Galway', '087121212', '30-12-2002', F, GY414, 1)

Block 1



Block 2



How are the following supported in heaped file organisation (using example 2)?

1. Inserting a new tuple:

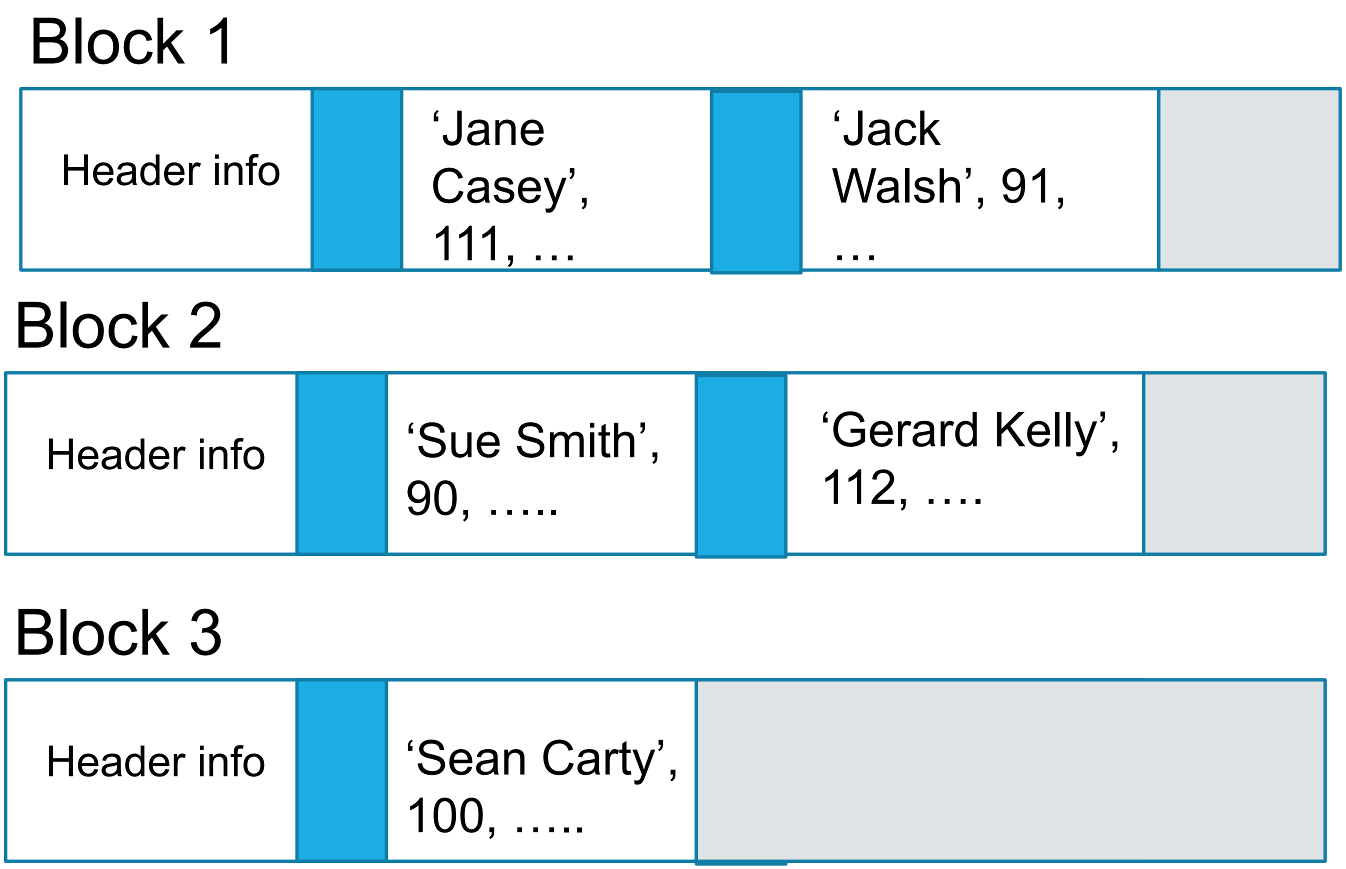
```
('Sean Carty', 100, '23 Ocean view, Salthill,
Galway', '087222333', '24-10-2002', 'M', 'GY101', 3)
```

2. Deleting an existing tuple:

```
('Jack Walsh ', 91, '13 College road, Galway',
'086654321', '01-09-2000', 'M', 'GY350', 3)
```

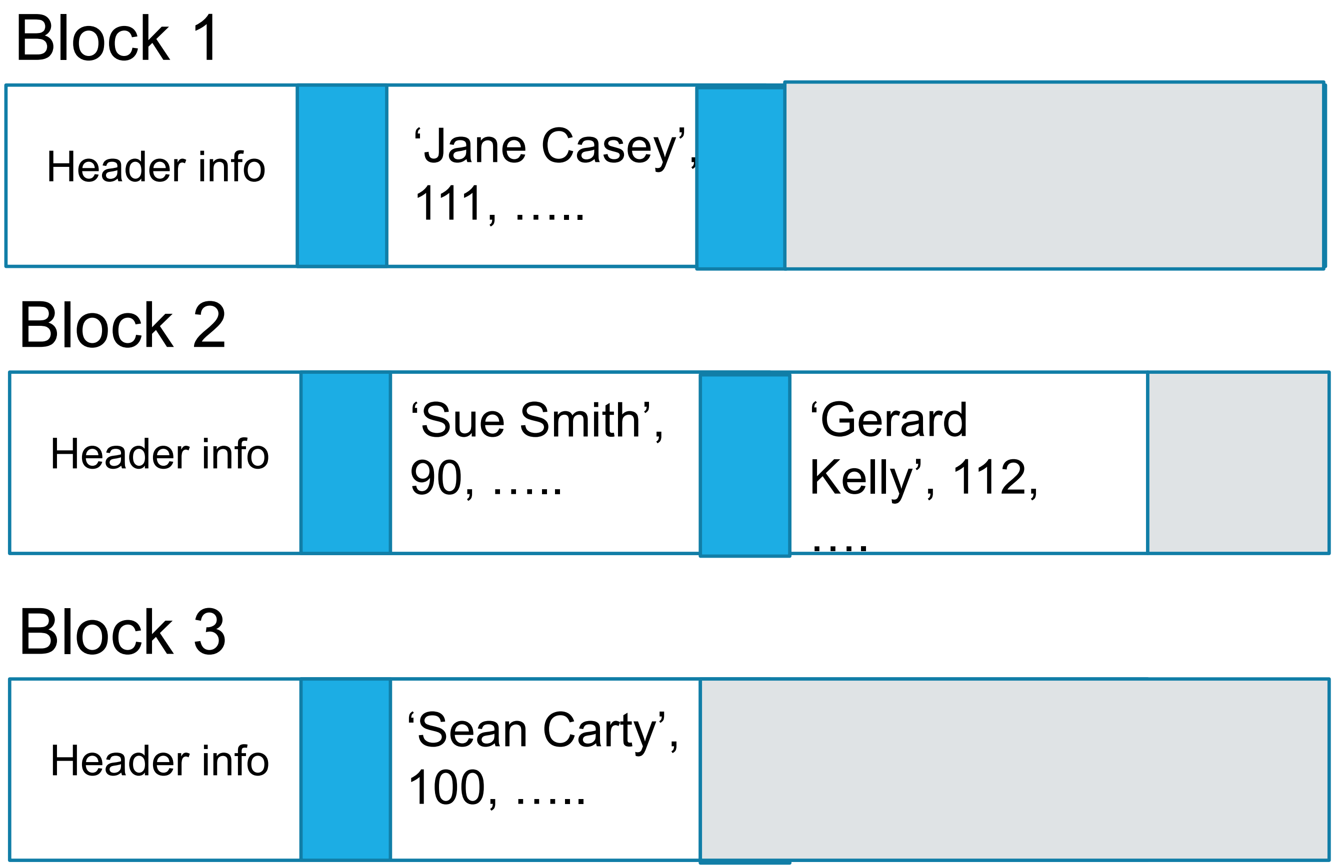
1. Inserting a new tuple:

```
('Sean Carty', 100, '23 Ocean view, Salthill, Galway',
'087222333', '24-10-2002', 'M', 'GY101', 3)
```



2. Deleting an existing tuple:

```
('Jack Walsh ', 91, '13 College road, Galway',
'086654321', '01-09-2000', 'M', 'GY350', 3)
```



HEAP FILE ORGANISATION

Advantages: Insertion efficient and easy - last disk block copied into buffer and record is added; block copied back to disk

Disadvantages:

1. Searching inefficient - must search all blocks (linear search)

2. Deleting inefficient - search first; delete and then leave unused space in block if using 'easy' insert approach

SEQUENTIAL FILE ORGANISATION

Approach: Records are stored in *sequential* order, based on the value of some key of each record – often primary key

Usually use an *index* with sequential file organisation

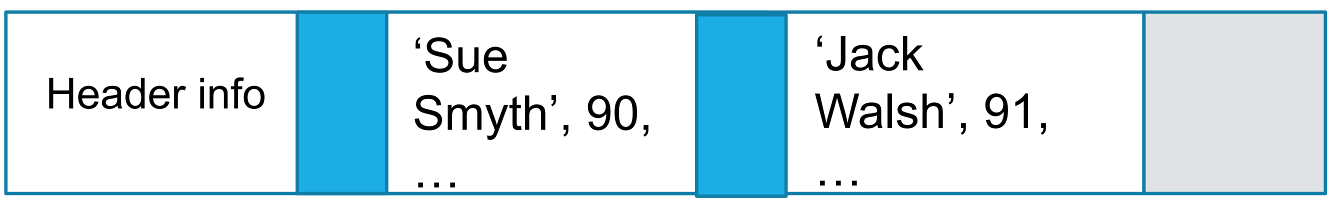
Allows records to be read in sorted order

EXAMPLE 3: Using a blocking factor of 2, and the schema from example 1, sketch the placement of the following student records using a <u>sequential file organisation</u> ordered on the `studentID`:

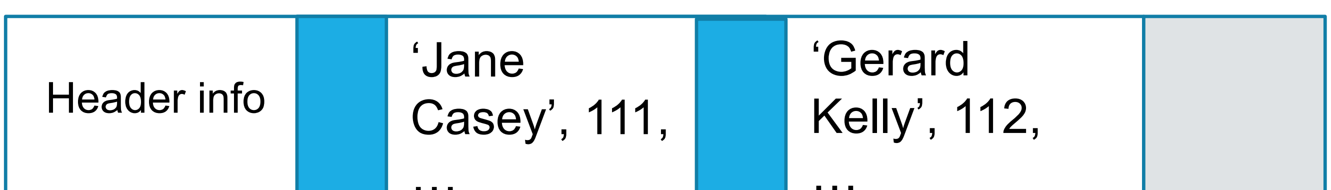
```
('Jane Casey', 111, '34 hazel park, Newcastle,
 Galway', '087123456', '17-05-2001', 'F', 'GY101', 1)

('Jack Walsh ', 91, '13 College road, Galway',
 '086654321', '01-09-2000', 'M', 'GY350', 3)

('Sue Smyth ', 90, 'Maree, Oranmore, Co. Galway',
 '087111222', '25-07-1999', 'F', 'GY406', 3)

('Gerard Kelly', 112, 'Main Street, Oughterard, Co.
 Galway', '087121212', '30-12-2002', F, GY414, 1)
```

```
('Jane Casey', 111, '34 hazel park, newcastle, Galway', '087123456', '17-05-2001', 'F', 'GY101', 1)

('Jack Walsh ', 91, '13 college road, Galway', '086654321', '01-09-2000', 'M', 'GY350', 3)

('Sue Smyth ', 90, 'Maree, Oranmore, Co. Galway', '087111222', '25-07-1999', 'F', 'GY406', 3)

('Gerard Kelly', 112, 'Main Street, Oughterard, Co. Galway', '087121212', '30-12-2002', F, GY414, 1)
```

Block 1



Block 2



How are the following supported in SEQUENTIAL file organisation (using results from example 3)?

1. Inserting a new tuple:

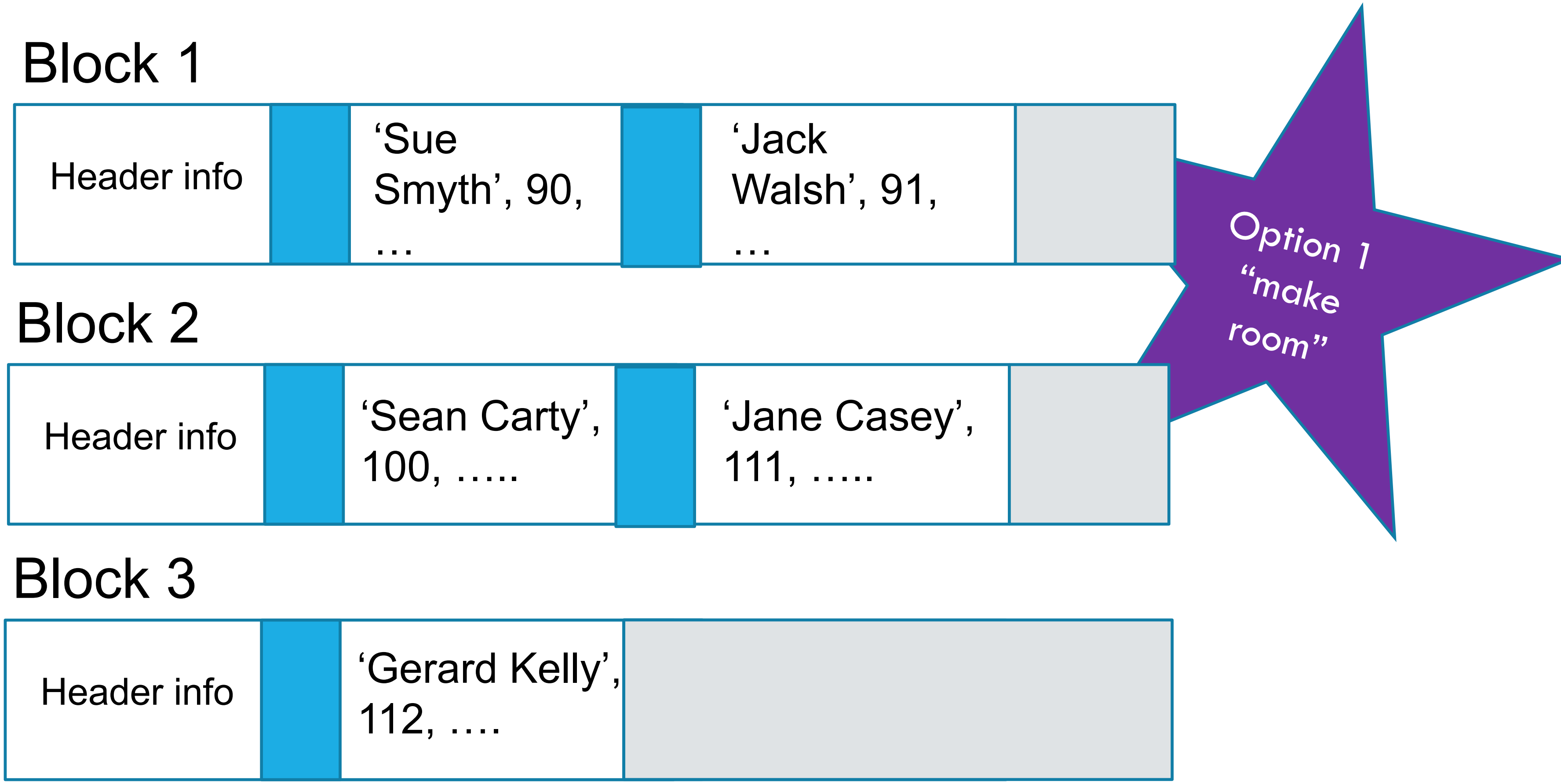
```
('Sean Carty', 100, '23 Ocean view,
Salthill, Galway', '087222333', '24-10-2002',
'M', 'GY101', 3)
```

2. Deleting an existing tuple:

```
('Jack Walsh ', 91, '13 College road,
Galway', '086654321', '01-09-2000', 'M',
'GY350', 3)
```

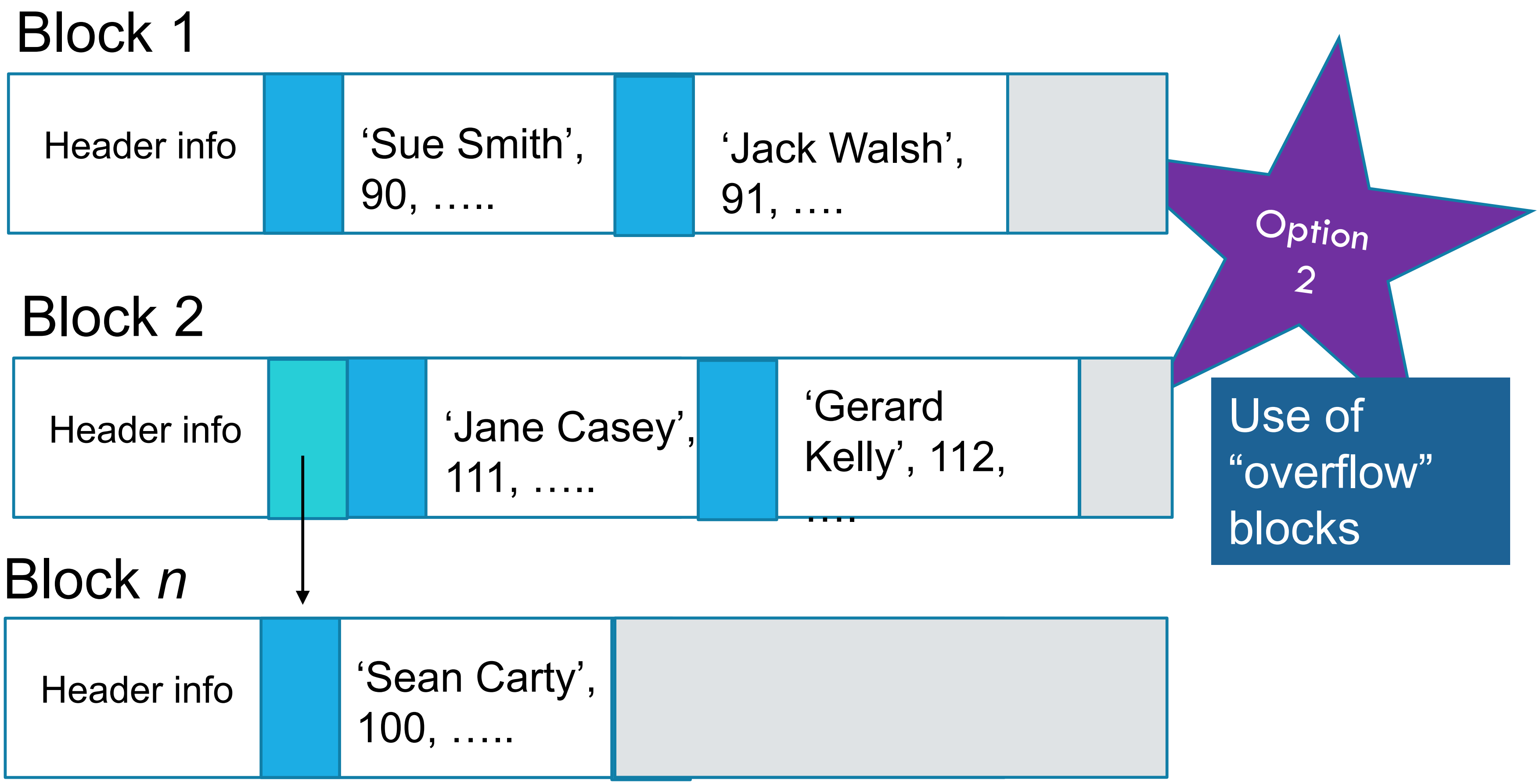
1. Inserting a new tuple:

```
('Sean Carty', 100, '23 Ocean view, Salthill, Galway',
'087222333', '24-10-2002', 'M', 'GY101', 3)
```



1. Inserting a new tuple:

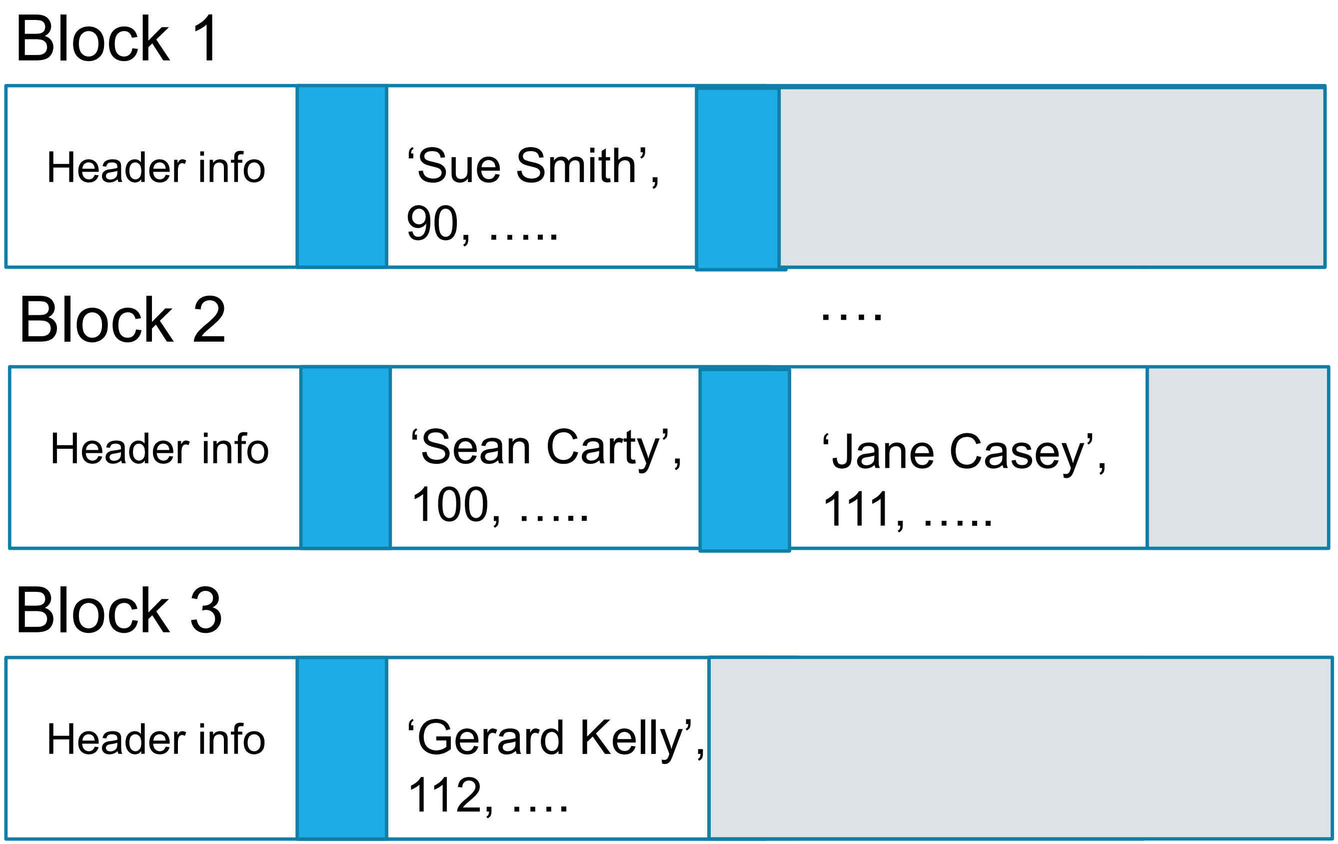
```
('Sean Carty', 100, '23 Ocean view, Salthill, Galway',
'087222333', '24-10-2002', 'M', 'GY101', 3)
```



2. Deleting an existing tuple (Option 1):

```
('Jack Walsh ', 91, '13 College road, Galway',
 '086654321', '01-09-2000', 'M', 'GY350', 3)
```

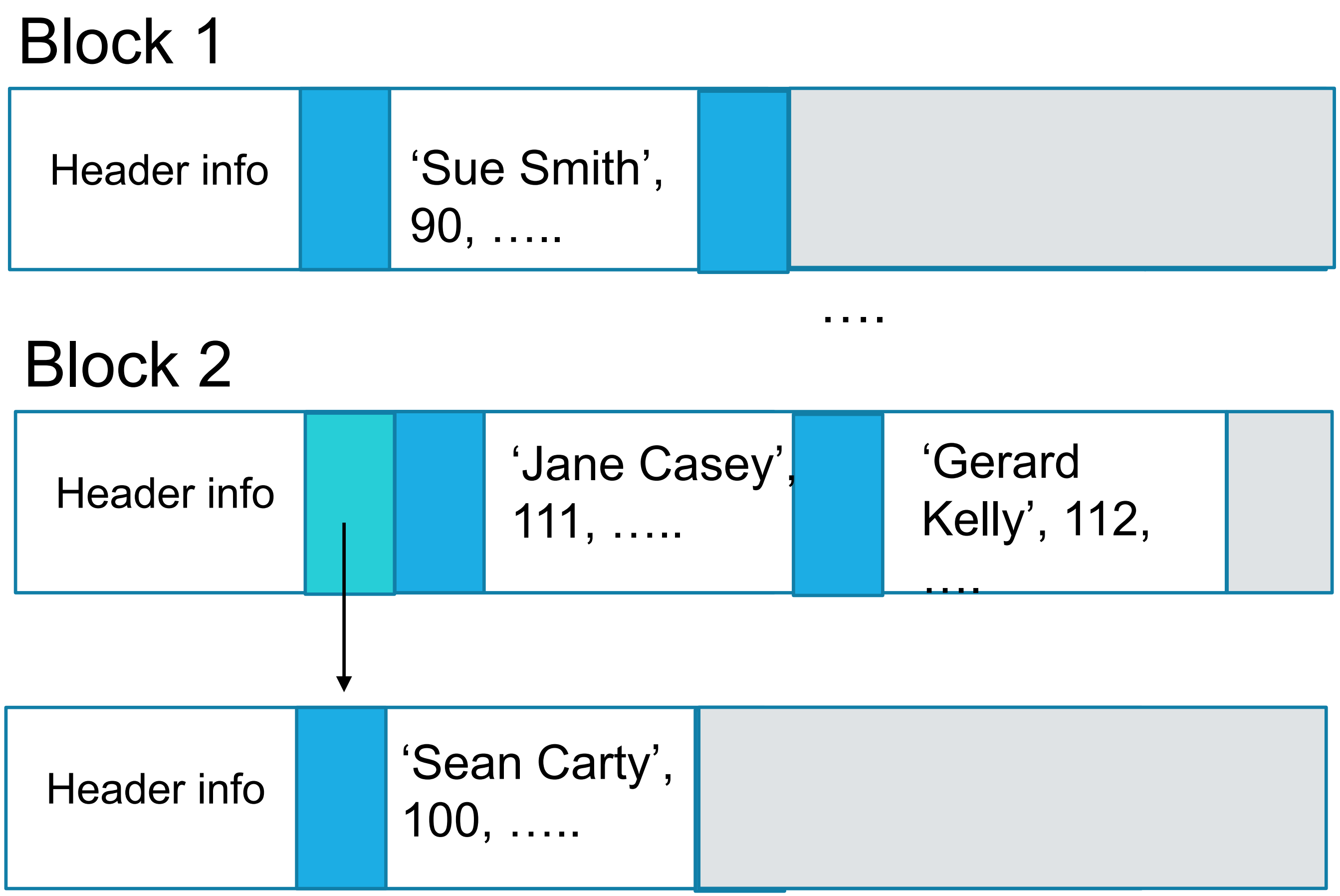
Result:



2. Deleting an existing tuple (Option 2):

```
('Jack Walsh ', 91, '13 College road, Galway',
 '086654321', '01-09-2000', 'M', 'GY350', 3)
```

Result:



SEQUENTIAL FILE ORGANISATION

Advantages:

- Reading records in order is efficient
- Searching is efficient on key field (binary search)
- Easy to find 'next record'

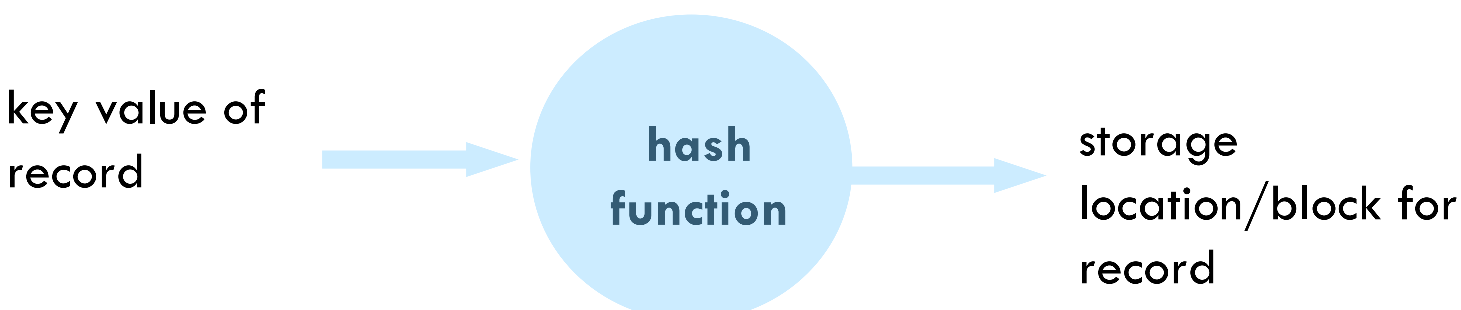
But …

- Insertion and deletion expensive as records must remain physically ordered. Pointer chains used (part of header information)
- What if searching on non-key field?

HASHING/HASHED FILE ORGANISATION

A *hash function* is computed on some attributes of each record (e.g., often key value)

The output of the hash function is the block address where the record should be placed



HASH FUNCTIONS

A common hash function is the MOD function where:

`a MOD b` or `a % b`

returns the remainder on dividing `a` by `b`, i.e. integer division.

Example:

```
20 MOD 7  = 6
100 MOD 5 = 0
```

where `b` should be a prime number – that is a number only evenly divisible by itself and 1

http://www.onlineconversion.com/prime.htm

EXAMPLE 4

Given the following records which should be stored in blocks based on user IDs and a hashed file organisation

The available blocks have IDs in the range 0-100 and have a blocking factor of 3

Assign the following records to blocks using user IDs:

1234

167

100

458

Example 4 steps:

1. Get prime number in the range 0-100 as close to 100 as possible - e.g., 97

2. For each key value of each record find the block number of where to place record by getting *keyvalue* mod *primenumber,* e.g., *keyvalue* mod 97

1234 MOD 97 = 70 *(97 divides in to 1234 12 times with remainder 70)*

167 MOD 97 = 70 (once)

100 MOD 97 = 3 (once)

458 MOD 97 = 70 (4 times)

Placing of the records:

Example 4 steps:

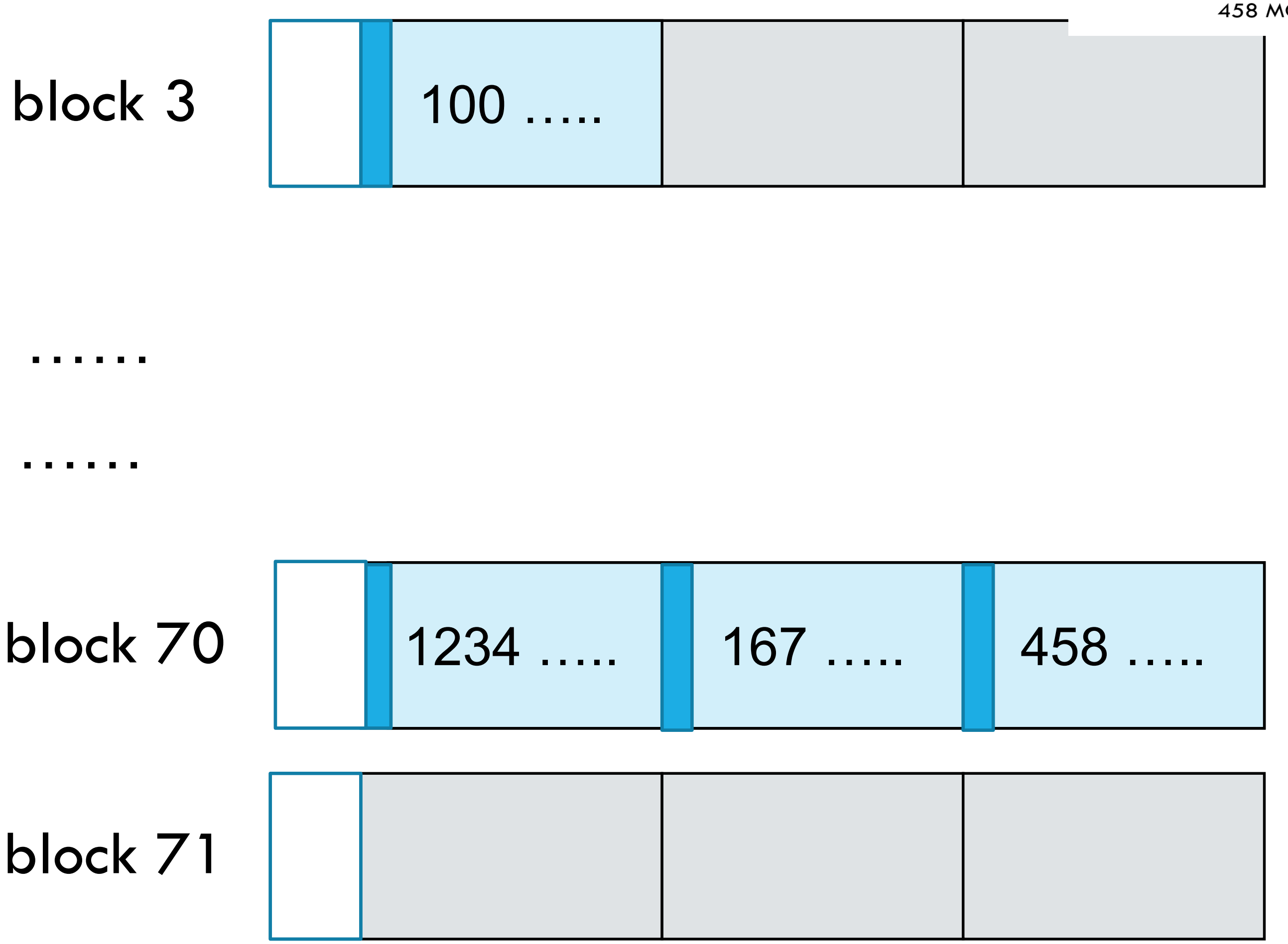
1. Get prime number in the range 0-100 as close to 100 as possible - e.g., *97*
2. For each key value of each record find the block number of where to place record by getting *keyvalue* mod *primenumber*, e.g., *keyvalue* mod 97

1234 MOD 97 = 70 *(97 divides in to 1234 12 times)*

167 MOD 97 = 70 (once)

100 MOD 97 = 3 (once)

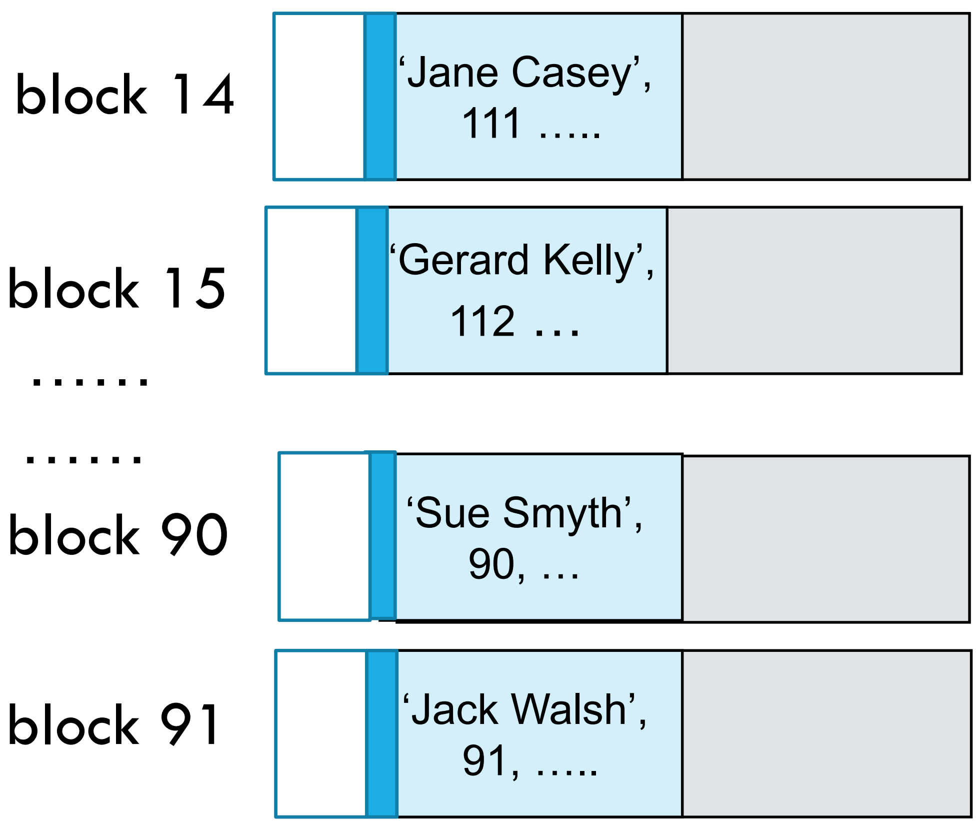
458 MOD 97 = 70 (4 times)



EXAMPLE 5: Using the student schema from example 1, and given a blocking factor of **2**, with mod function of 97, sketch the placement of the following student records using hashed file organisation:

```
('Jane Casey', 111, '34 hazel park, Newcastle, Galway',
 '087123456', '17-05-2001', 'F', 'GY101', 1)

('Jack Walsh ', 91, '13 College road, Galway',
 '086654321', '01-09-2000', 'M', 'GY350', 3)

('Sue Smyth ', 90, 'Maree, Oranmore, Co. Galway',
 '087111222', '25-07-1999', 'F', 'GY406', 3)

('Gerard Kelly', 112, 'Main Street, Oughterard, Co.
 Galway', '087121212', '30-12-2002', F, GY414, 1)
```

```
('Jane Casey', 111, '34 hazel park, Newcastle, Galway', '087123456', '17-05-2001', 'F', 'GY101', 1)

('Jack Walsh ', 91, '13 College road, Galway', '086654321', '01-09-2000', 'M', 'GY350', 3)

('Sue Smyth ', 90, 'Maree, Oranmore, Co. Galway', '087111222', '25-07-1999', 'F', 'GY406', 3)

('Gerard Kelly', 112, 'Main Street, Oughterard, Co. Galway', '087121212', '30-12-2002', F, GY414, 1)
```




```
111 mod 97 = 14

91 mod 97 = 91

90 mod 97 = 90

112 mod 97 = 15
```

How are the following supported in HASHED file organisation (using results from example 5)?

1. Inserting a new tuple:

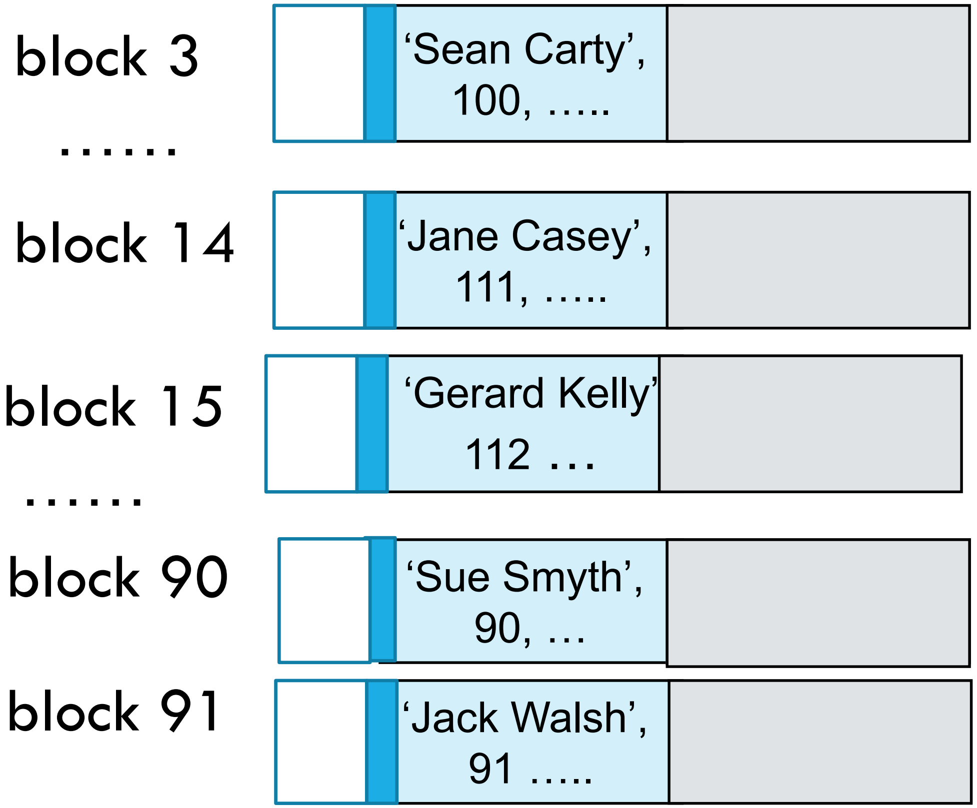
('Sean Carty', 100, '23 Ocean view, Salthill, Galway', '087222333', '24-10-2002', 'M', 'GY101', 3)

2. Deleting an existing tuple:

('Jack Walsh ', 91, '13 College road, Galway', '086654321', '01-09-2000', 'M', 'GY350', 3)

1. Inserting a new tuple:

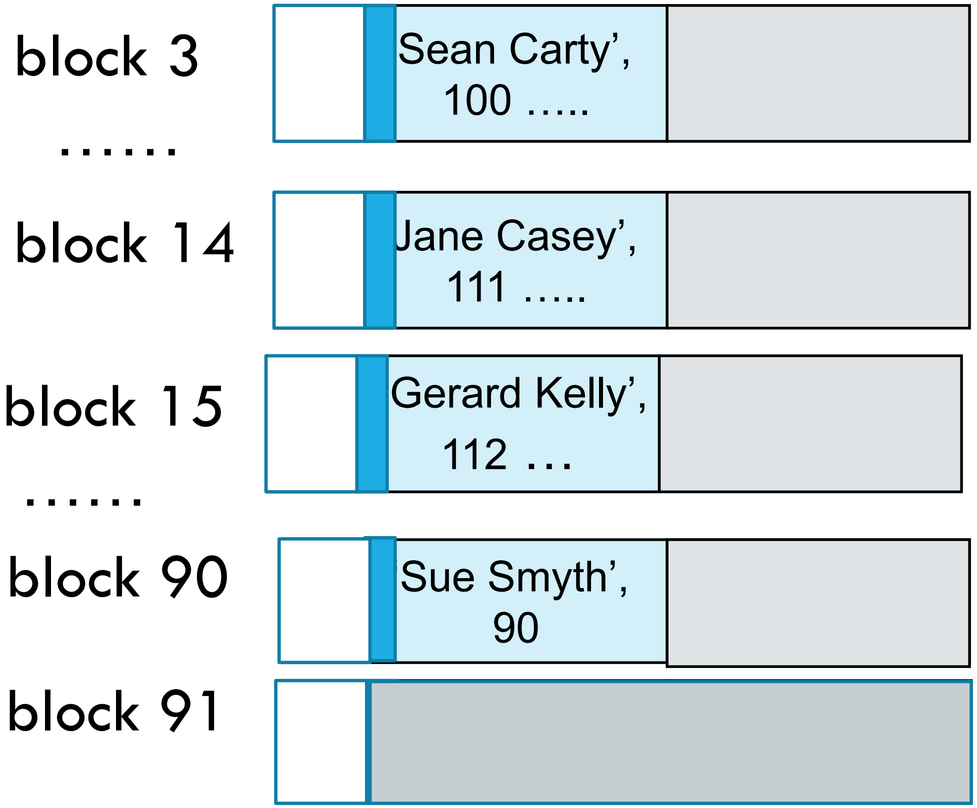
```
('Sean Carty', 100, '23 Ocean view, Salthill,
Galway', '087222333', '24-10-2002', 'M',
'GY101', 3)
```




```
100 mod 97 = 3
```

2. Deleting an existing tuple:

```
('Jack Walsh ', 91, '13 College road, Galway', '086654321',
 '01-09-2000', 'M', 'GY350', 3)
```




```
91 mod 97 = 91
```

QUESTION: Is 97 a good choice for this problem … with 20000 records?

No! Will use blocks from 0 to 96 (97 blocks)

With a blocking factor of 2, at most can fit 97x2 = 194 records

Need a much larger prime number and more blocks

Prime number close to 10000, e.g., 10009, but not much room for growth

Prime number close to 20000, e.g., 19751, would be much better

(http://www.onlineconversion.com/prime.htm)

Criteria for choosing hash function

- Easy and quick to compute (as mod function is)
- Should uniformly distribute hash addresses across the available space … Picking a prime number for the mod function helps with this … but cannot guarantee it
- Anticipate file growth (insertions and deletions) so only a fraction of each block is initially filled, thus leaving room to insert new records

COLLISIONS

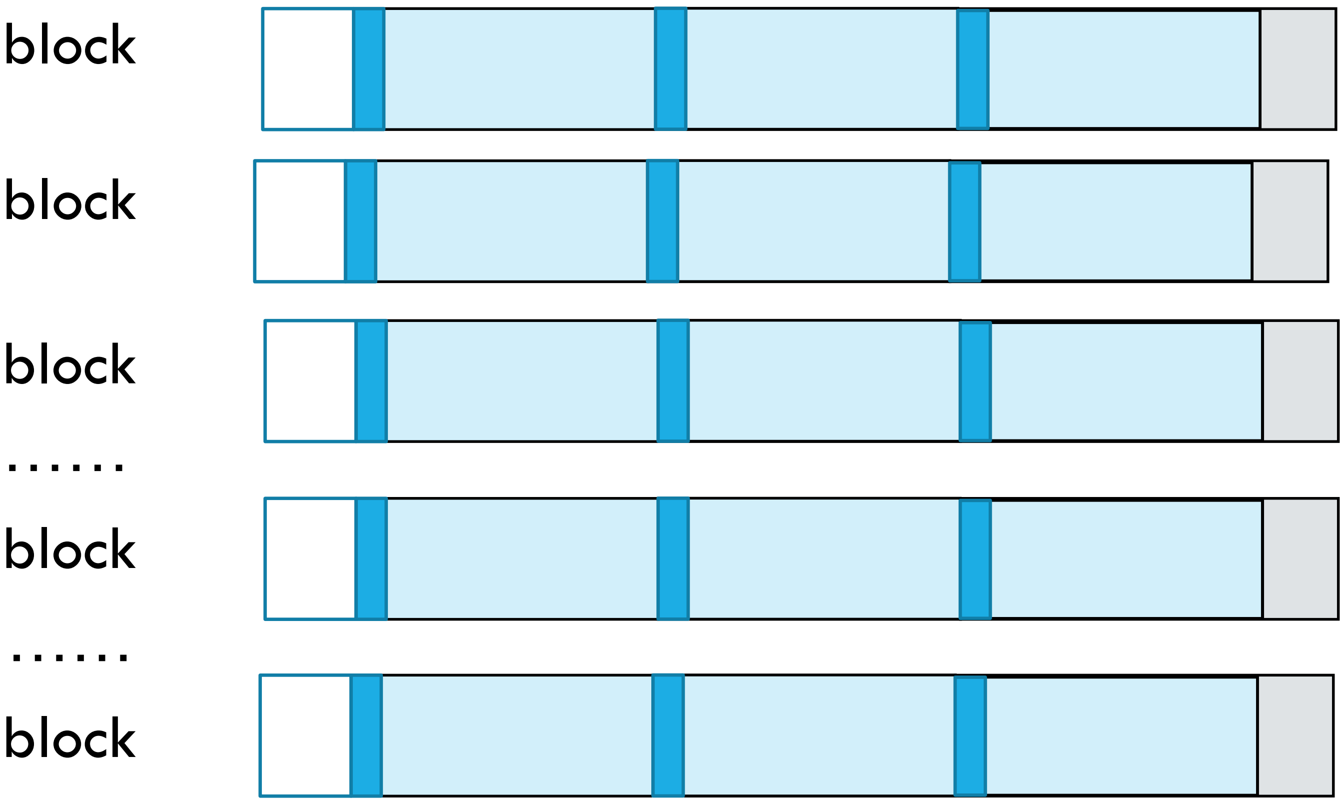
- However, at any stage, two or more key field values can hash to the same location … if there is no room to place record this is called a **collision**
- If a collision occurs, and there is no space in block for new record, then must find a new location … this is called **collision resolution**
- One approach to collision resolution is **Linear Probing**
  - Hash function returns block location i for record
  - If there is no room in block *i* check block *i+1*, *i+2* etc. until a block with room is found
  - Can degrade to a linear scan if load factor is high

EXAMPLE 6:

Given the following key field values of five records, show how the associated records are assigned to blocks using a *hashed file organisation* with the mod function (mod 7) where a blocking factor of 3 is being used and with linear probing collision resolution.

Key values of records: 24, 73, 20, 9, 10, 31

Placing of the records 24, 73, 20, 9, 10, 31 using mod 7 and a blocking factor of 3 and linear probing collision resolution



Calculating blocks IDs:

24 mod 7 =

73 mod 7 =

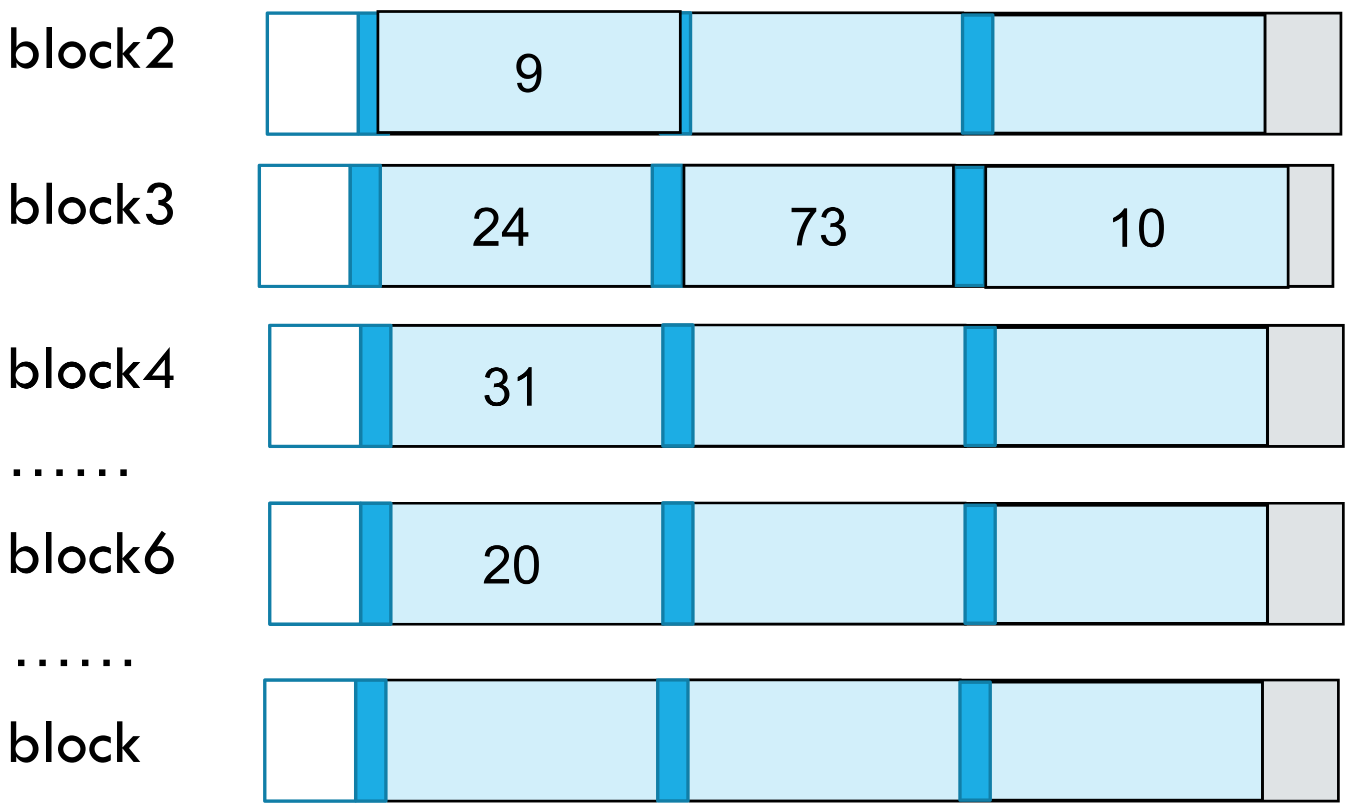
20 mod 7 =

9 mod 7 =

10 mod 7 =

31 mod 7 =

Placing of the records 24, 73, 20, 9, 10, 31 using mod 7 and a blocking factor of 3 and linear probing collision resolution



Calculating blocks IDs:

```
24 mod 7 = 3
73 mod 7 = 3
20 mod 7 = 6
9 mod 7 = 2
10 mod 7 = 3
31 mod 7 = 3
```

DATABASE INDEXES

Indexing speeds-up operations that are not efficiently supported by the basic file organisation.

Consists of ***index entries***

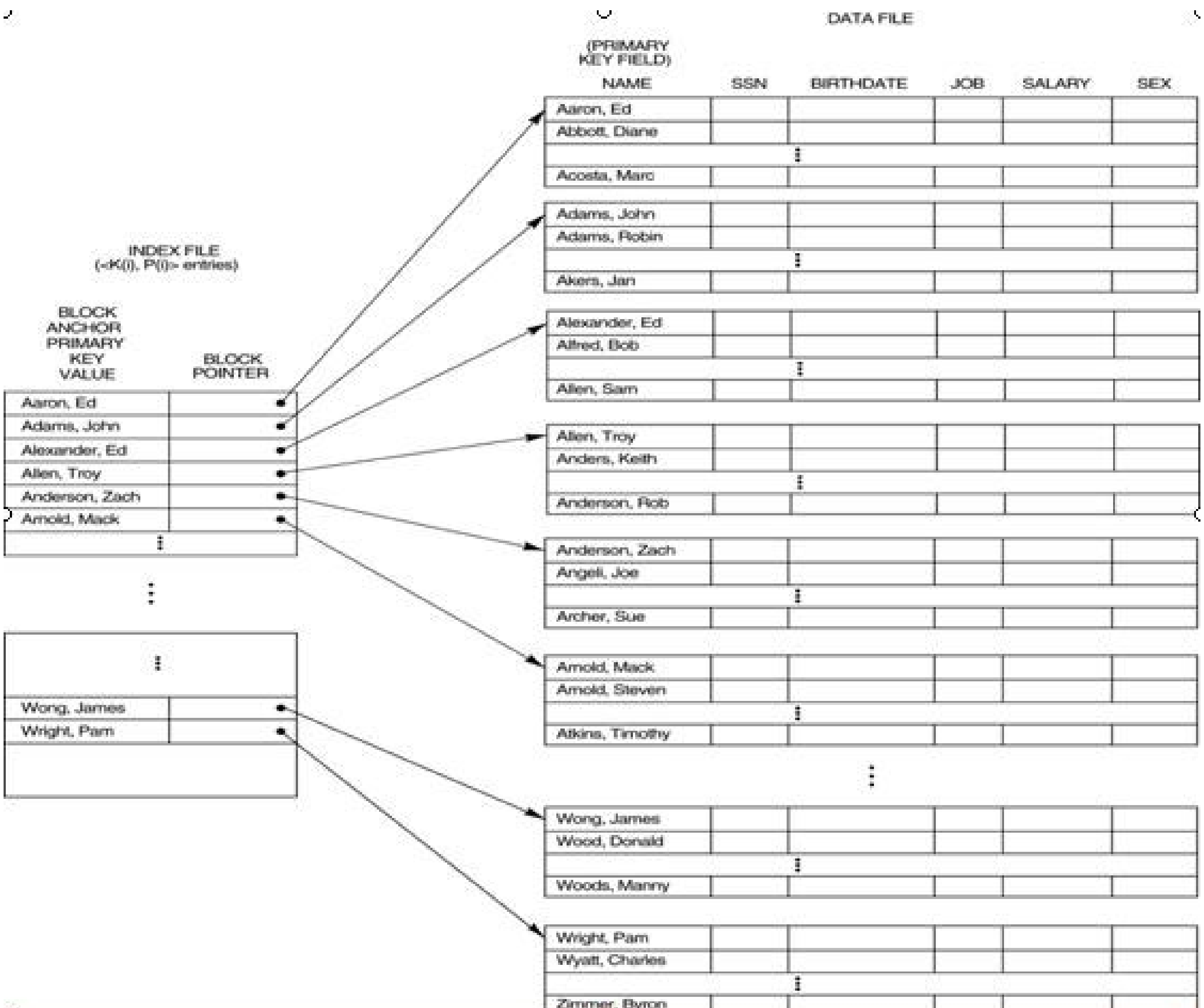
Each index entry consists of:

- index key
- record or block pointer

The index entries are placed on disk, either in sequential sorted order (ordered indexes) or hashed order.

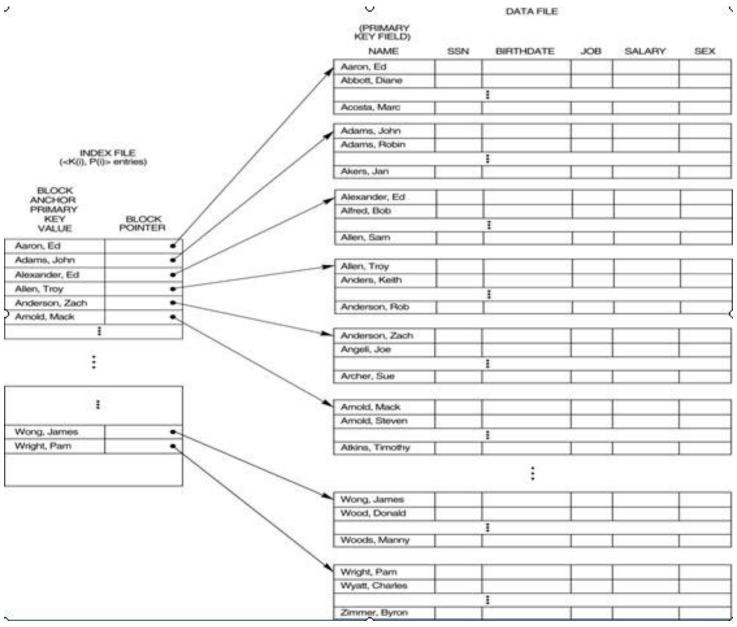
A complete index may be able to reside in main memory

Example of index file organisation of staff schema on name



To access a record using indexing key:

1. Retrieve index file
2. Search through it for required field (based on index key value)
3. Answer query or return to secondary storage for the block which contains the required record.



Dense vs sparse indexes

An index is dense if it contains an entry for every record in the file

A dense index may be created for any index key
A sparse/non-dense index contains an entry for each block rather than an entry for every record in the file and can only be used if the records are assigned to blocks in sorted (sequential) order based on the index key

- A sparse index is called a **primary index**

More on primary indexes

- The total number of entries in the index is the same as the number of **blocks** in the ordered file
- The first record in each block is called the **anchor record** of the block

Advantages:

- Fewer index entries than records so index file is smaller

Disadvantages:

- Insertions and deletions a problem - may have to change anchor record
- Searching may take longer

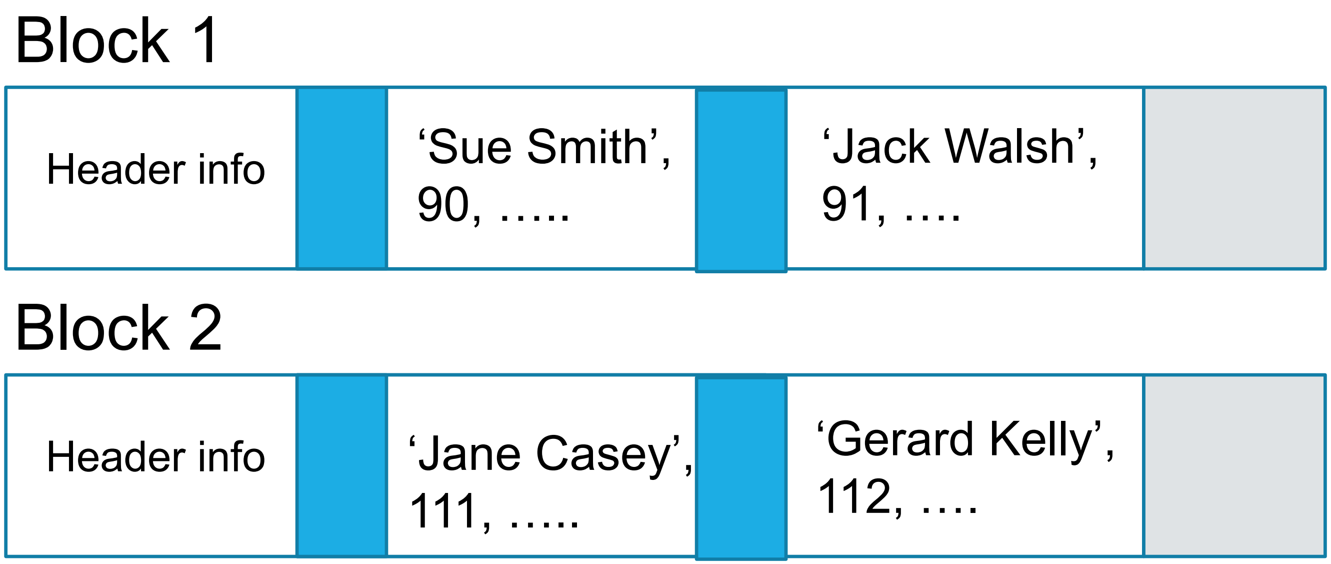
EXAMPLE 7: Indexed file Organisation

Given the student schema from Example 1, with primary key `studentID`. With the aid of a diagram, illustrate how a **dense** indexing file organisation might operate (with blocking factor of 2 and sequential file organisation)

e.g. for the examples:

```
('Jane Casey', 111, '34 hazel park, Newcastle, Galway',
 '087123456', '17-05-2001', 'F', 'GY101', 1)

('Jack Walsh ', 91, '13 College road, Galway',
 '086654321', '01-09-2000', 'M', 'GY350', 3)

('Sue Smyth ', 90, 'Maree, Oranmore, Co. Galway',
 '087111222', '25-07-1999', 'F', 'GY406', 3)

('Gerard Kelly', 112, 'Main Street, Oughterard, Co.
 Galway', '087121212', '30-12-2002', F, GY414, 1)
```

DENSE …. Index entry for each record



Index file

90	Block 1
91	Block 1
111	Block 2
112	Block 2

How are the following supported in dense indexed sequential file organisation (using example 7)?

1. Inserting a new tuple:

('Sean Carty', 100, '23 Ocean view, Salthill, Galway', '087222333', '24-10-2002', 'M', 'GY101', 3)

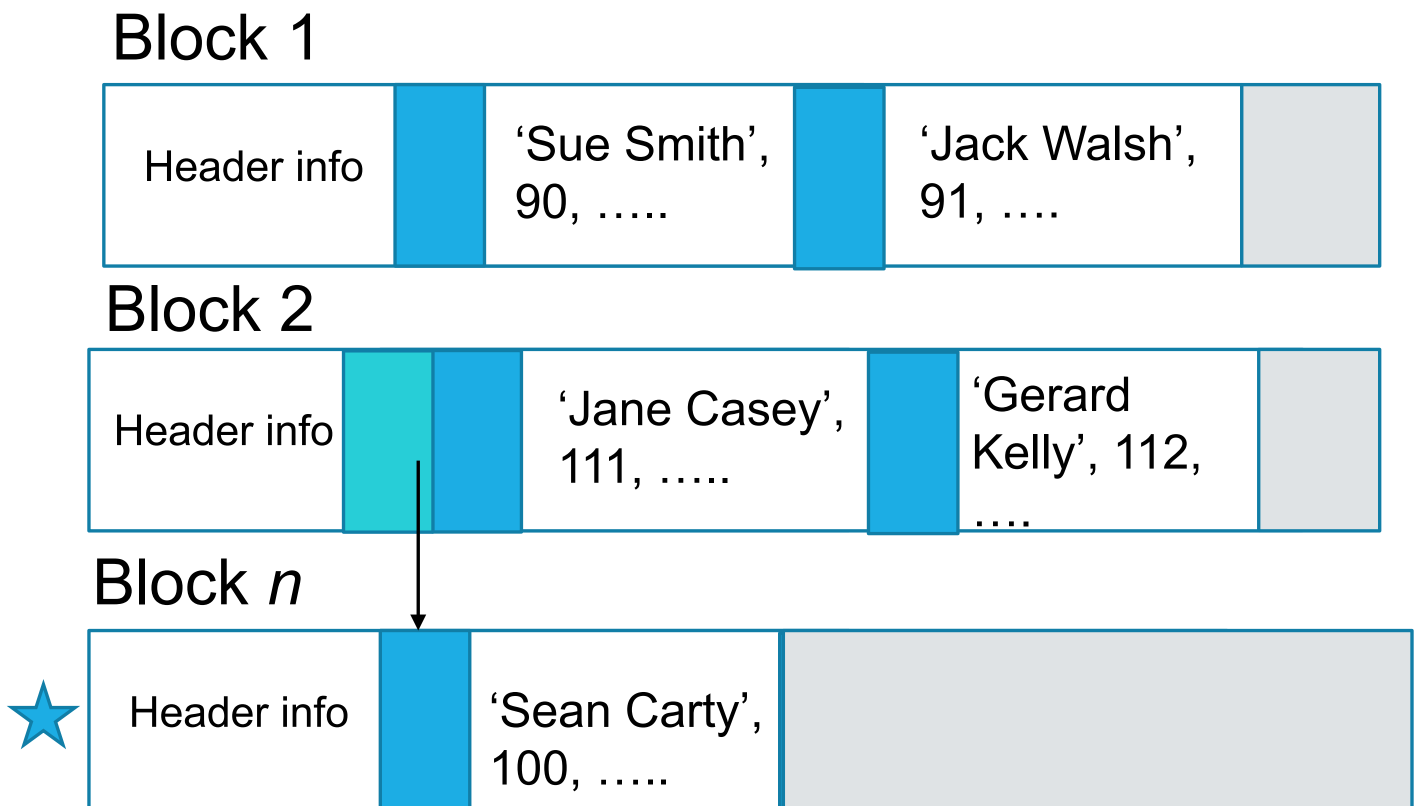
2. Deleting an existing tuple:

('Jack Walsh ', 91, '13 College road, Galway', '086654321', '01-09-2000', 'M', 'GY350', 3)

Inserting a tuple …
two updates needed

Index file

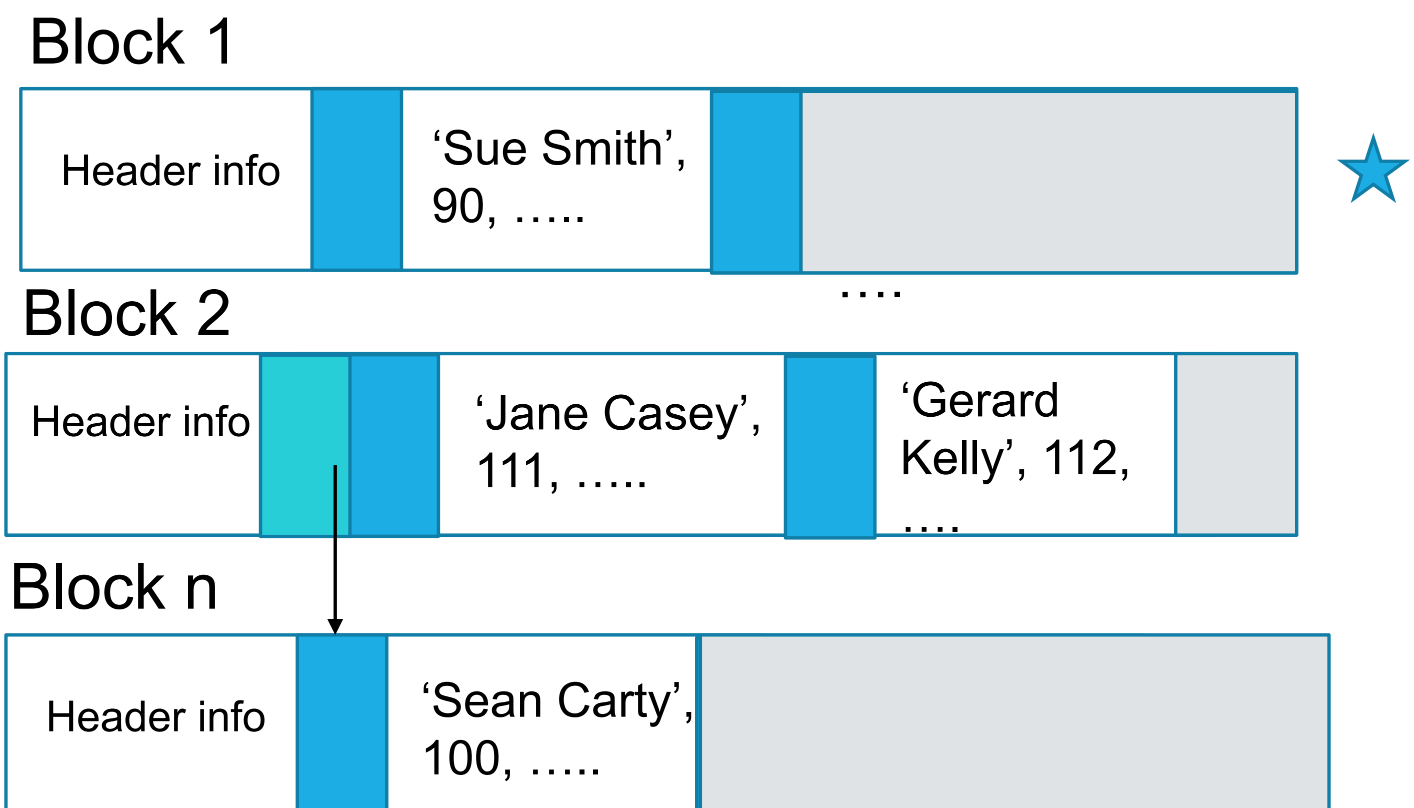
90	Block1
91	Block 1
100	Block n
111	Block 2
112	Block 2



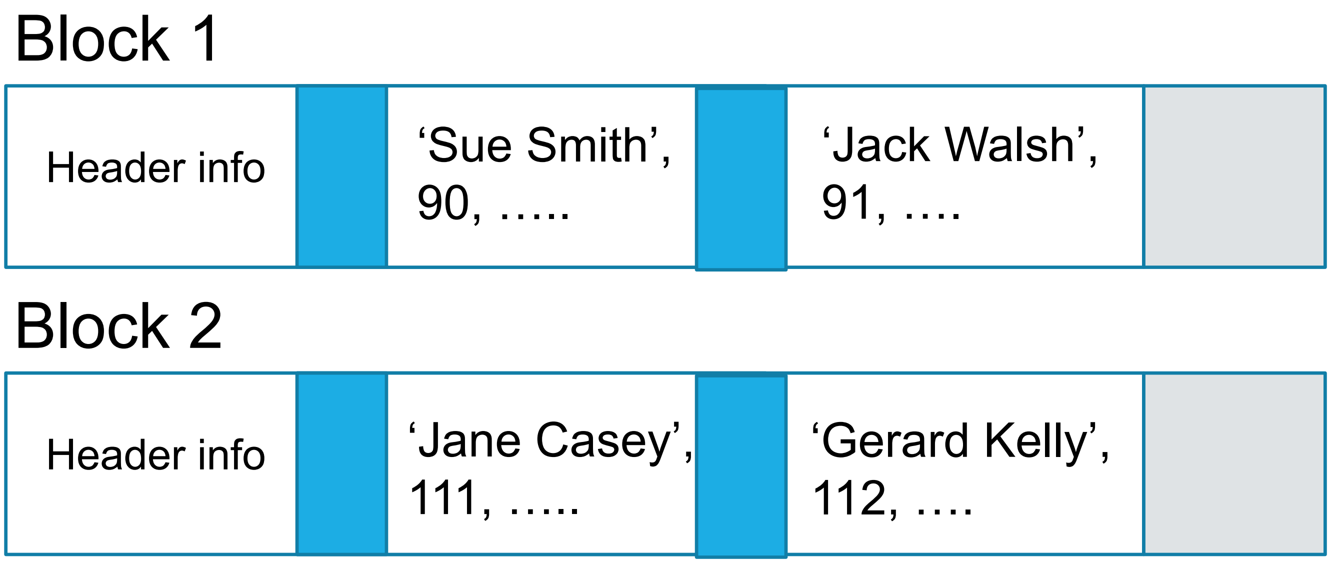
Deleting a tuple …
two deletions needed

Index file

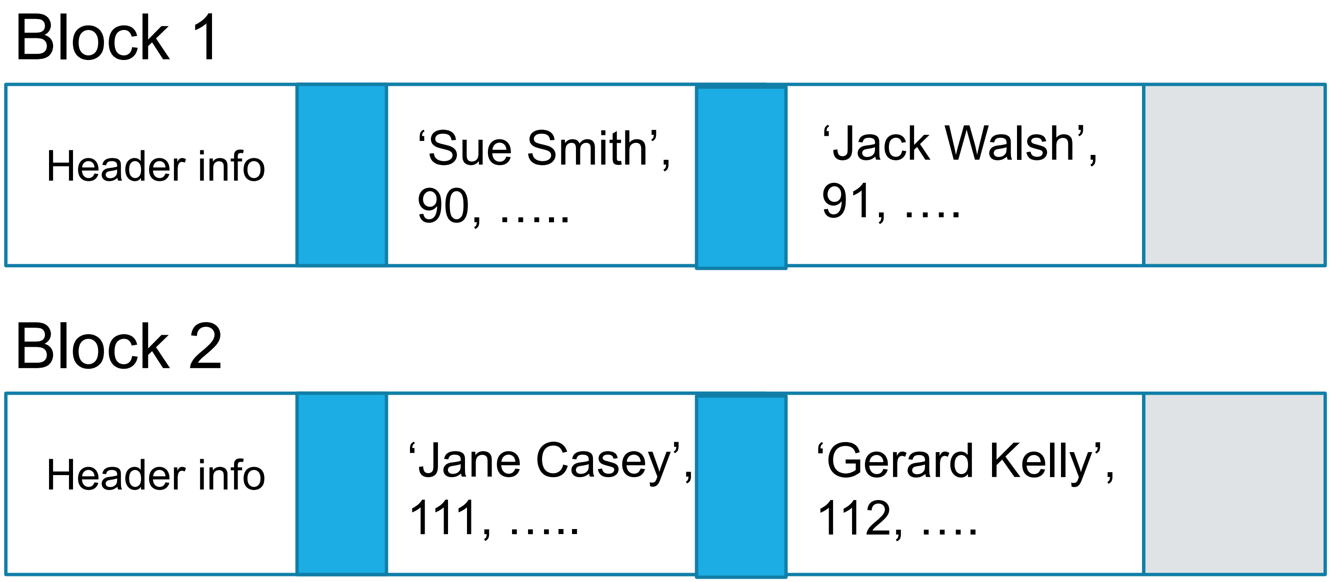
90	Block1
100	Block n
111	Block 2
112	Block 2



Example 8: Using the illustrated example from example 7, show how the organization of data looks for **non-dense** indexing (sequential organization)



Sparse/Non-dense ….
Index entry for each **block**



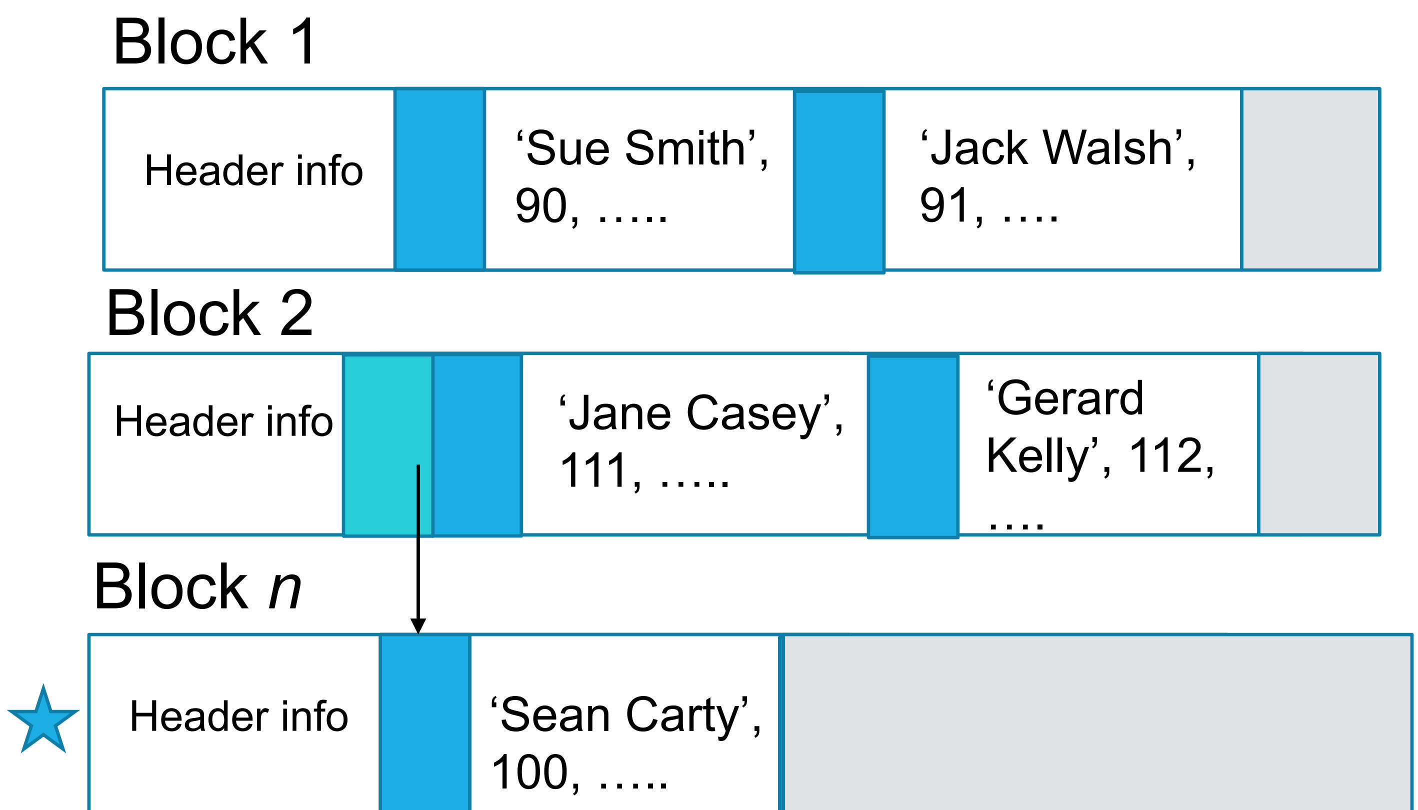
Index file

90	Block 1
111	Block 2

Inserting a tuple with sparse indexing

Index file

90	Block1
100	Block n
111	Block 2

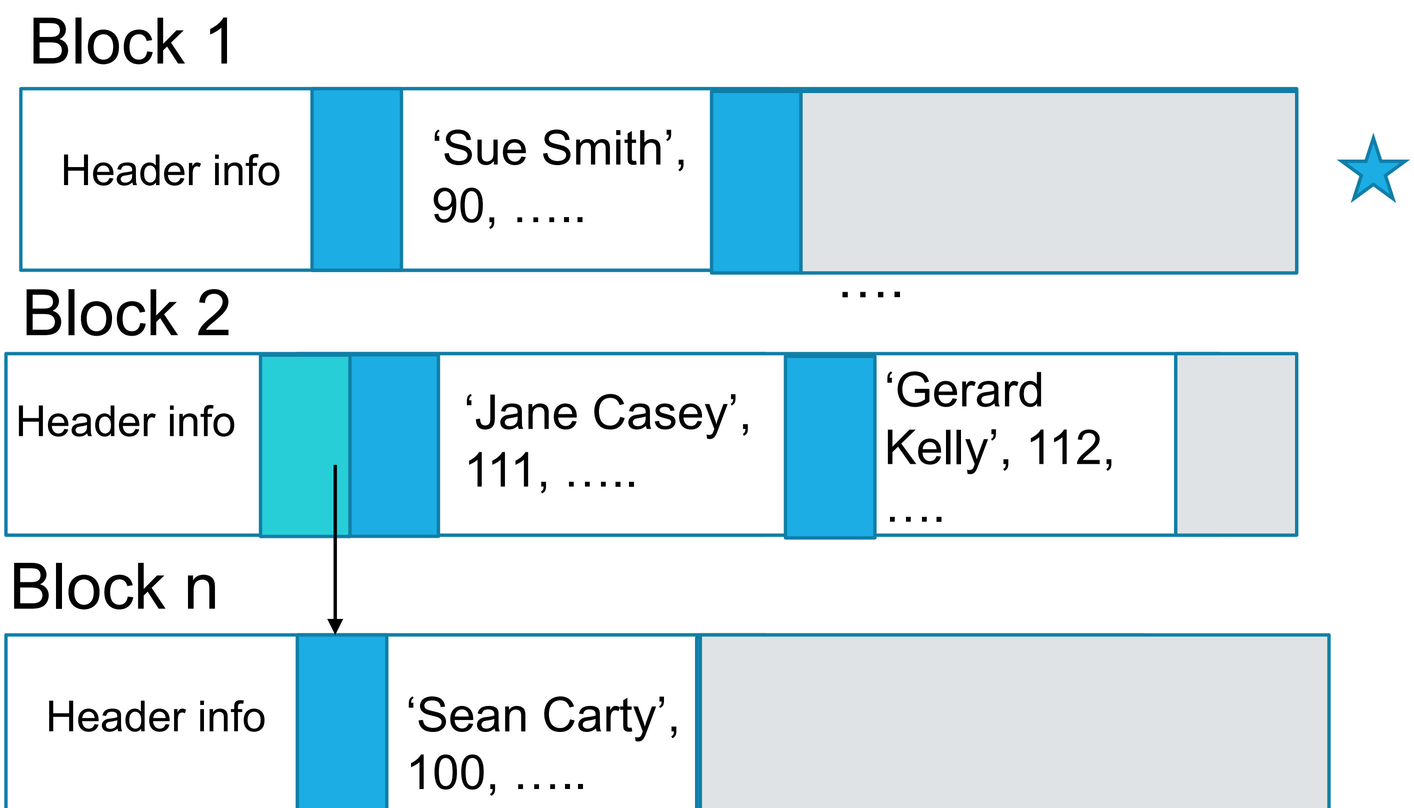


Deleting a tuple … with sparse indexing

Index file

90	Block1
100	Block n
111	Block 2

No change in index



CLUSTERED AND SECONDARY INDEXES

Records that are logically related are physically stored close together on the disk (i.e., in the same blocks or consecutive blocks)

Records are physically ordered on a non-key field that does not have a distinct value for each record

Clustering index consists of:

- clustering field value
- block pointer to first block that has a record with that value for clustering field

Advantages/disadvantages of clustering:

Quick access on clustering field but have to search all blocks in querying on non-clustering fields

Example 9:

Consider a file holding the employee information from the Company schema where each record contains a positive integer indicating the department where an employee works. Show how a clustering index on department number (DNO) might operate on such data – with blocking factor of 3

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

Option 1:
Fill all blocks

Index value (dno)	*block value*
1	b1
4	b1
5	b2

Index file

b1

James E Borg, …. , 888665555, …..	…, 1
Alicia Z Zelaya, …. 999887777, ……	…, 4
Jennifer S Wallace, …. 987654321, ….	…, 4

b2

Ahmad V Jabbar, ……, 987987987, …..	…., 4
John B Smith, ……… , 123456789, ….	….., 5
Franklin T Wong, ……., 333445555, ….	……, 5

b3

Ramesh K Narayan, …. 666884444, ….	……., 5
Joyce A English, …….., 453453453, ……	….., 5

bN

Option 2:

Leave 'space' for other records with that clustering field value

Index value (dno)	*block value*
1	**b1**
4	b2
5	b3

Index file

b1

James E Borg, …. , 888665555, …..	…, 1

b2

Alicia Z Zelaya, …. 999887777, ……	…, 4
Jennifer S Wallace, …. 987654321, ….	…, 4
Ahmad V Jabbar, ……, 987987987, …..	…., 4

b3

John B Smith, ……… , 123456789, ….	….., 5
Franklin T Wong, ……., 333445555, ….	……, 5
Ramesh K Narayan, …. 666884444, ….	…., 5

b4

Joyce A English, …….., 453453453, ……	….., 5

Option 3:

Use a *Secondary Index and sequential file organisation*

A secondary index is an index whose index (clustering) value specifies an order different to the underlying sequential order of the file.

Any attribute can be chosen as the clustering index value.

Any number of secondary indexes can be built with different clustering index values.

b1

John B Smith,... ,123456789, ...	...., 5
Franklin T Wong, ., 333445555, ...	...., 5
Joyce A English, ., 453453453,	. ...., 5

b2

Ramesh K Narayan, ...666884444,	...., 5
James E Borg, ... , 888665555,	..., 1
Jennifer S Wallace, ... 987654321,	....., 4

b3

Ahmad V Jabbar, ..., 987987987,	...., 4
Alicia Z Zelaya, ... 999887777, ...	..., 4

Option 3: *Secondary Index*

Clustering Index file

Index value (dno)	*block value*
1	**A**
4	**B**
5	**C**

Secondary Indexes

A

b2		

B

b2	b3	

C

b1	b2	

b1

John B Smith,… ,123456789, ….	…., 5
Franklin T Wong, ., 333445555, …	…., 5
Joyce A English, ., 453453453,	. …., 5

b2

Ramesh K Narayan, ….666884444,	…., 5
James E Borg, …. , 888665555,	…, 1
Jennifer S Wallace, …. 987654321,	….., 4

b3

Ahmad V Jabbar, ……, 987987987,	…., 4
Alicia Z Zelaya, …. 999887777, ……	…, 4

SECONDARY INDEXES

- Does not impact the actual storage of records (which blocks they reside in – which can be sequential)
- Can define multiple secondary indexes as well as a primary index

For example:

Clustering Index

1	**A**
4	**B**
5	**C**

Primary index

123456789	b1
666884444	b2
987987987	b3

Secondary Indexes

A

b2		

B

b2	b3	

C

b1	b2	

b1

John B Smith,… ,123456789, ….	…., 5
Franklin T Wong, ., 333445555, …	…., 5
Joyce A English, ., 453453453,	. …., 5

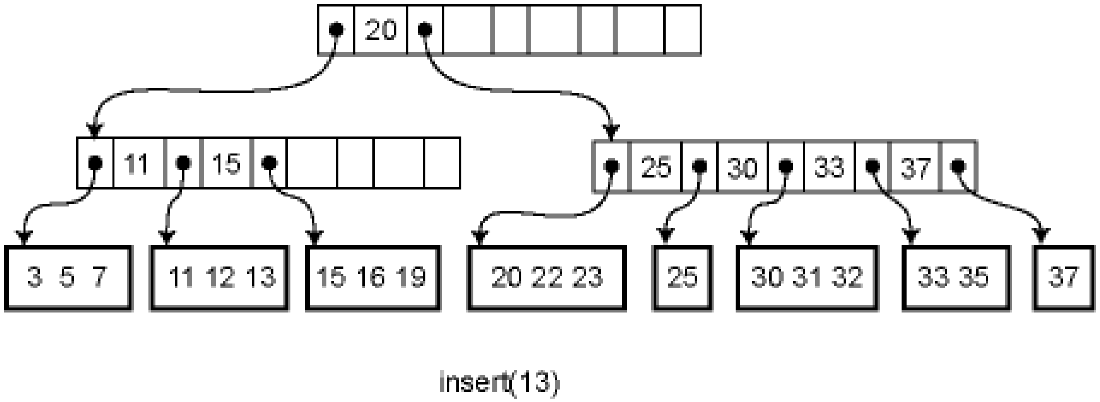
b2

Ramesh K Narayan, ….666884444,	…., 5
James E Borg, …. , 888665555,	…, 1
Jennifer S Wallace, …. 987654321,	….., 4

b3

Ahmad V Jabbar, ……, 987987987,	…., 4
Alicia Z Zelaya, …. 999887777, ……	…, 4

B-TREES



Most commercial systems use an indexing structure called B-trees, and specifically B+ trees.

B-trees allow as many levels of indexes as is appropriate for the file being indexed

B-trees manage the space in blocks so that every block is between half-used and completely full

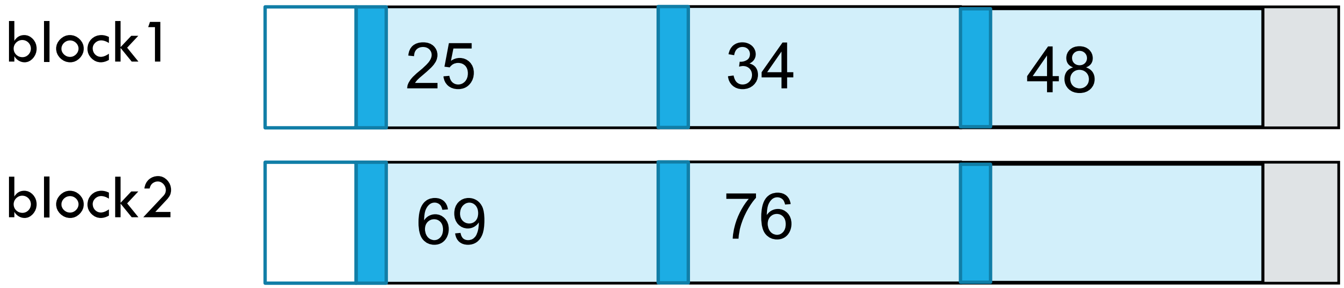
B-trees consist of sequences of pointers arranged in a tree data structure

CLASS WORK ….
WINTER 2017 QUESTION ON FILE ORGANISATIONS

(b)	**Given an unspanned memory organisation, fixed record length, a blocking factor of 3, and five records with the following primary keys:** **25, 34, 48, 69, 76**
(i)	With the aid of examples, outline the main advantages and disadvantages of placing the given records in blocks under a sequential file organisation. (5)
(ii)	With the aid of a diagram, and using sequential file organisation, differentiate between a dense and non-dense indexing of the given five records. (5)
(iii)	With the aid of an example, describe what is meant by secondary indexing. (5)
(iv)	(i) With the aid of a diagram, show where the given five records would be placed in blocks under a hashed file organisation. The mod function (mod 7) should be used in addition to linear probing. (5)

Given an unspanned memory organisation, fixed record length, a blocking factor of 3, and five records with the following primary keys:
25, 34, 48, 69, 76

(i)	With the aid of examples, outline the main advantages and disadvantages of placing the given records in blocks under a sequential file organisation. (5)

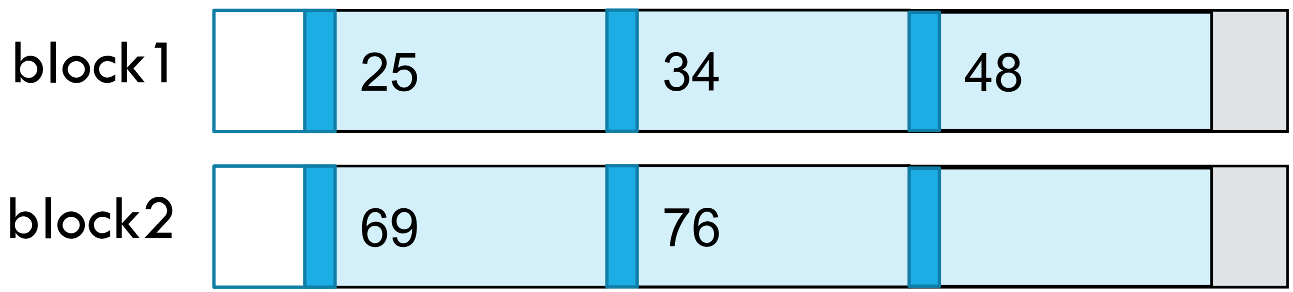


Advantages … reading on key field value (in order)
Disadvantages … maintaining sorted order when adding records

(ii)	With the aid of a diagram, and using sequential file organisation, differentiate between a dense and non-dense indexing of the given five records. (5)

Dense

25	block1
34	block1
48	block1
69	block2
76	block2



Non Dense (Primary Index)

25	block1
69	block2

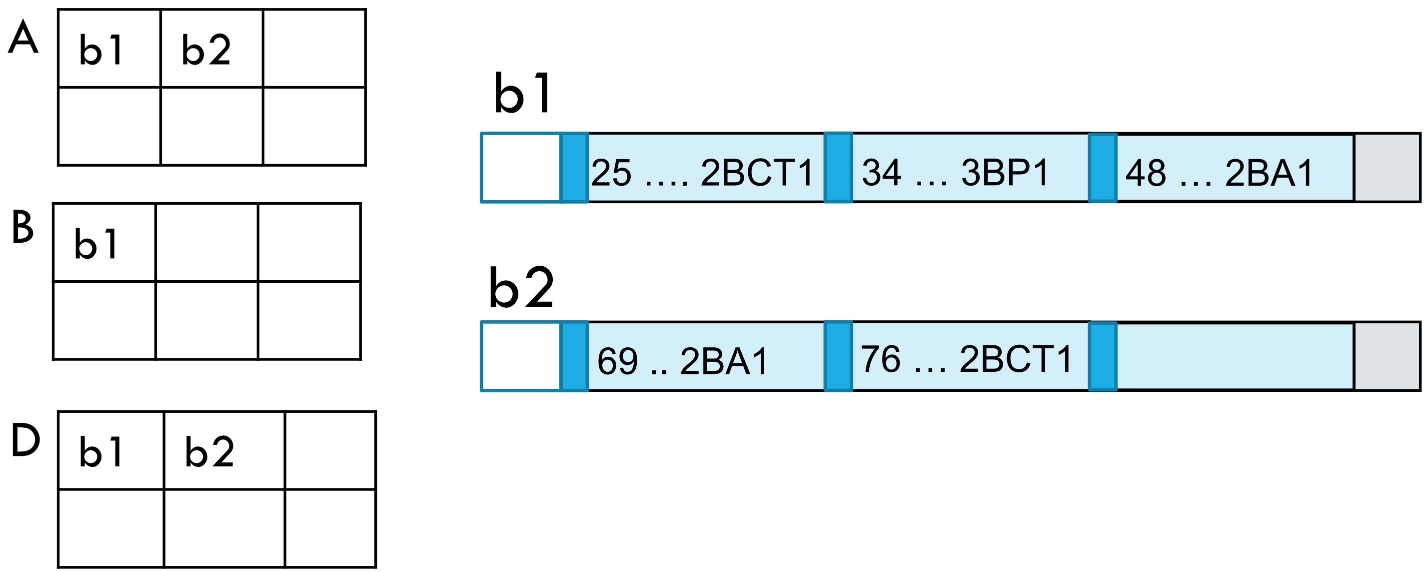
With dense indexing we should have an entry for every record; 5 records implies 5 index entries
With non-dense indexing we should have an entry for every block associated with the file; 2 blocks implies 2 index entries. The key value of the first record in each block is used as the index value.

(iii) With the aid of an example, describe what is meant by secondary indexing. (5)

Example: Assuming the primary keys are student IDs (e.g., 25, 34, 48) and we also store the course code for each student (e.g., 2BA, 3BP, etc.) as well as other student information (not shown). Records are assigned to blocks based on the primary key, with a blocking factor of 3.
Course code can be used as a clustering index and the actual references to the blocks holding the student records are stored in a secondary index.

Clustered Index

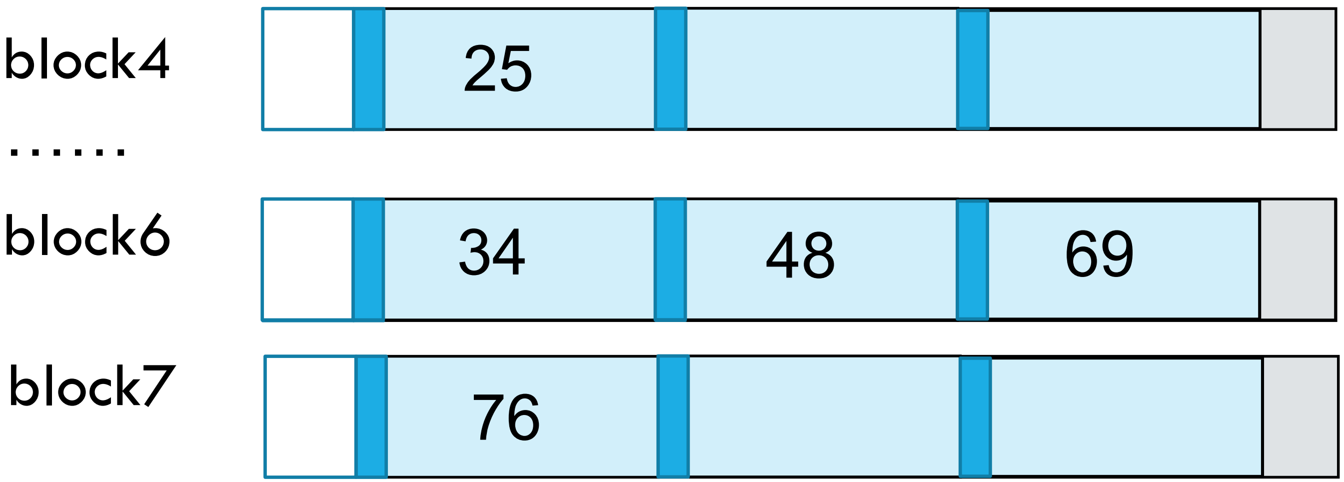
2BCT1	A
3BP1	B
3BLE1	C
2BA1	D
2BDS1	E



Secondary Indexes

(iv) (i) With the aid of a diagram, show where the given five records would be placed in blocks under a hashed file organisation. The mod function (mod 7) should be used in addition to linear probing. (5)

25, 34, 48, 69, 76



Calculating blocks IDs:

25 mod 7 = 4

34 mod 7 = 6

48 mod 7 = 6

69 mod 7 = 6

76 mod 7 = 6

SUMMARY: IMPORTANT TO KNOW

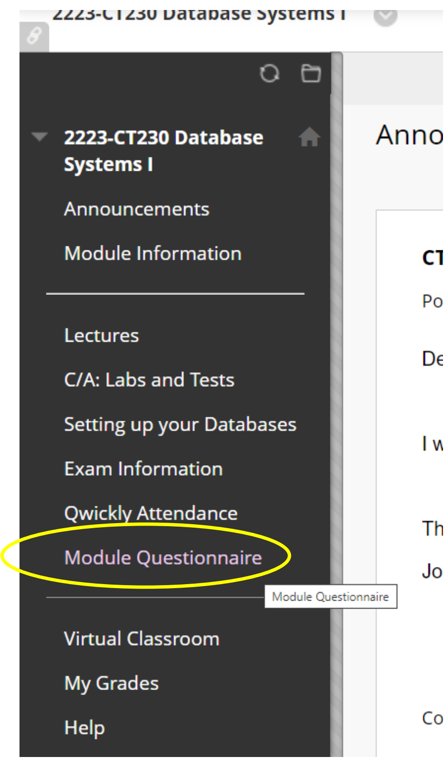
- Blocking factor
- Basic 3 organisations: Heaped, Sequential and Hashed (with collision resolution)
- Indexed – Dense and non-dense
- Clustered Index and secondary indexes (not B+ trees)

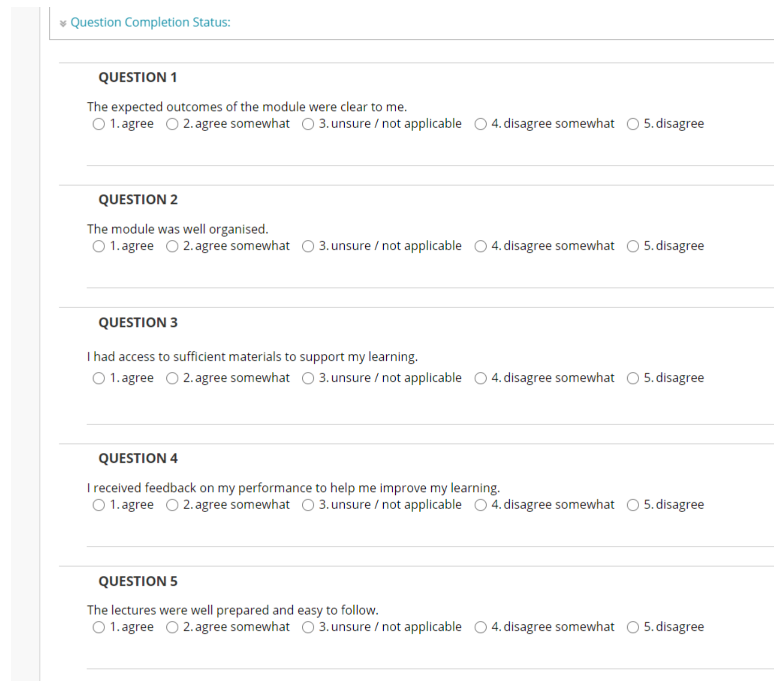
CT230
DATABASE SYSTEMS

Summary &
Exam
Information

EVALUATION FORM AVAILABLE ON BLACKBOARD

Please complete!





CT230 TOPICS

- Introductory material: Databases and Database Management Systems; File System approach Vs Database approach [no exam question]
- The Relational Model – definitions
- The Relational Model and Constraints
- SQL: DDL and DML SELECT
- ER Models
- Normalisation (1, 2 and 3 NF)
- Relational Algebra
- Query Processing and Cost Estimates and Heuristic Optimisation
- File Organisations: Heaped, Sequential, Hashed, Indexed – Dense, Non-dense, Clustered, Secondary

CT230 LEARNING OUTCOMES

On successful completion of this module the learner will be able to:

LO1	Define and explain terms, concepts, properties and constraints of Relational Database Systems
LO2	Identify the theoretical and practical issues in the storage, manipulation, organisation and indexing of large quantities of data
LO3	Program a database management system for database creation and manipulation
LO4	Use Relational Algebra for relational database retrieval
LO5	Program using SQL for relational database retrieval and manipulation
LO6	Create and apply Entity Relationship Diagrams (ERD) as part of database development
LO7	Specify functional dependencies and differentiate between relations in 1st Normal Form, 2NF, 3NF. Apply the process of normalization
LO8	Define and explain the process of query processing and optimisation. Apply query optimisation heuristics to develop a query tree that represents an efficient evaluation strategy for a given query.

EXAM: SEMESTER 1 2022

Name: CT230: Database Systems I

Time allowed: Two hours *

Date: 08/12/2022 at 13:00

(as of 21/11/2022 – double check closer to the exam time)

* unless you have a LENS report

EXAM FORMAT AND INSTRUCTIONS (as in previous years)

You will be given a description of a new (unseen) database (no data)

Exam will comprise 4 questions, Answer Question 1 and 2 others:

Question 1: SQL Questions based on database given. Compulsory. Answer all questions. (40 marks)

Answer any 2 questions from 3 (30 marks per question):

Question 2: Based on database given. Relational Algebra and Query Processing and Optimisation

Question 3: File Organisations and Normalisation

Question 4: ER Diagrams and Mapping to Tables.

PREVIOUS EXAM PAPERS AVAILABLE from exams database:

NOTES:

- There is no question specific to MySQL or Adminer or phpMyAdmin, ReLax calculator, MS Visio or equivalent or asking how to perform a task in these
- One database schema (tables and description only) will be used for the SQL, relational algebra and query processing questions. This will be a new database description (with no data given).
- A different database schema will be used for the normalisation question and a different example will be used for the ER model.

NOTES ON QUESTION 1 (SQL)

Only SQL questions (DDL and SQL SELECT)

- No sample data is given. Only code is important for exam, not the answer to the queries.
- Have to make *reasonable* guess about data types for any DDL question

MARKS

Exam paper: 80% of final mark

- Question 1: 40 marks
- 2 Questions (2, 3, 4): 30 marks per question

5 quizzes

- Worth 20% of final mark

STUDY AIDS

Lecture Notes

Code examples from lectures

Problem Sheets and sample solutions

Sample tests

Past exam papers

Elmasri & Navathe book and relevant chapters as highlighted in lectures

Note: You will not be expected to know any material outside of that which I presented

HINTS ON THE DAY …

- Decide on the questions you will answer
- Decide on the amount of time you will give to each question
- Take some time at the start to read through the database description a few times rather than starting to answer questions straight away
- More time should be given to Q1 but do not spend all your time on Q1
- Try not to get confused between Relational Algebra Syntax and SQL syntax
- Unless you find material easy and have extra time do not answer extra questions … rarely is an advantage

GENERAL EXAM HINTS

Note the amount of marks allocated to each question … unless you have time to spare, do not spend 20 minutes on a question worth 2 marks.

Try/attempt all required questions and all parts in each question

Answer what is asked … e.g. "*with the aid of examples*"; "*explain the approach taken*"

If short on time, try sketch down main points first, then return and add detail if any time remains.

EXAMINATION RESULTS

Except for visiting students, no **official** results will be available before the Examinations Office send results in summer.

For any official exams, lecturers will provide a provisional mark (date tbc by the Registrar but usually by the start of February)

EXAMINATION BOARDS

“Examination Boards will be held at the end of a Stage, normally Semester 2, and after the repeat examinations in August.”

“The Examination Board will determine the overall result and will apply compensation provisions.”

“Only those decisions approved by the Examinations Board will be formally recognized as official University examination results – relating to Passing, Progression, Determination of Honours, and Granting of Deferrals.”

CS USEFUL CONTACTS

Josephine.Griffith@universityofgalway.ie

School Administration: Deirdre King (**deirdre.king@nuigalway.ie**)

Help is available:

If you need help, especially coming up to exams, you can contact:

- DISC
- SUMS
- Your college office and student advisors
- Your lecturers and year tutors
- The Student Information Desk (SID)
- Student counsellors
- Chaplains

All will be ready to help…you just need to ask

Remember if are unable to sit your exams you should request a deferral