Dr Greg Wadley

THE UNIVERSITY OF MELBOURNE

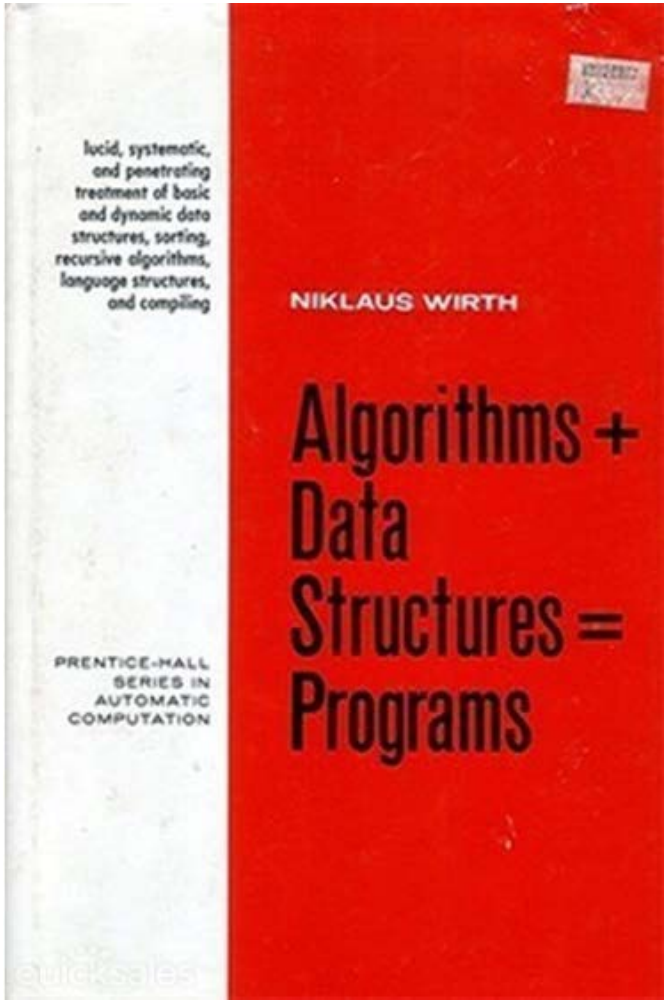
INFO 90002 Database Systems & Information Modelling

Week 01

Introduction to Databases

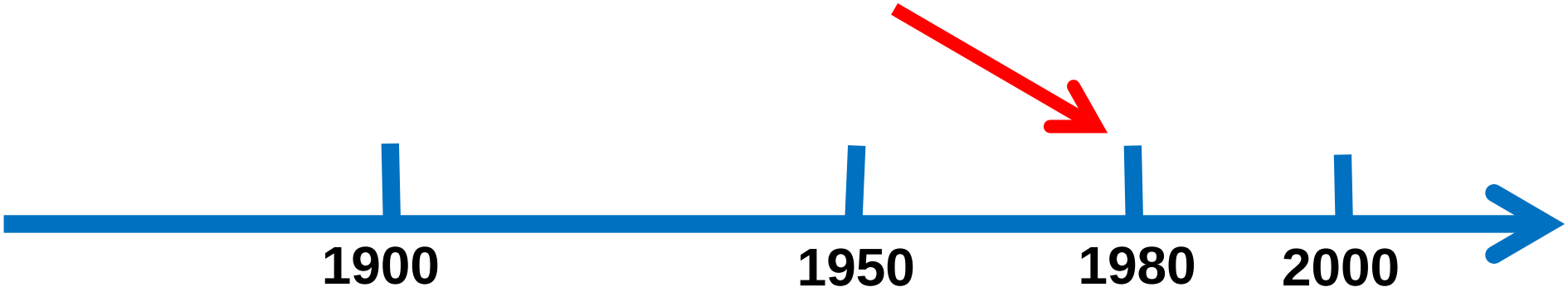
Context: software and data

- Computer systems consist of *software* (algorithms) working to process *data*.
- You will learn about creating software and algorithms in other subjects.
- This subject is about *data*.



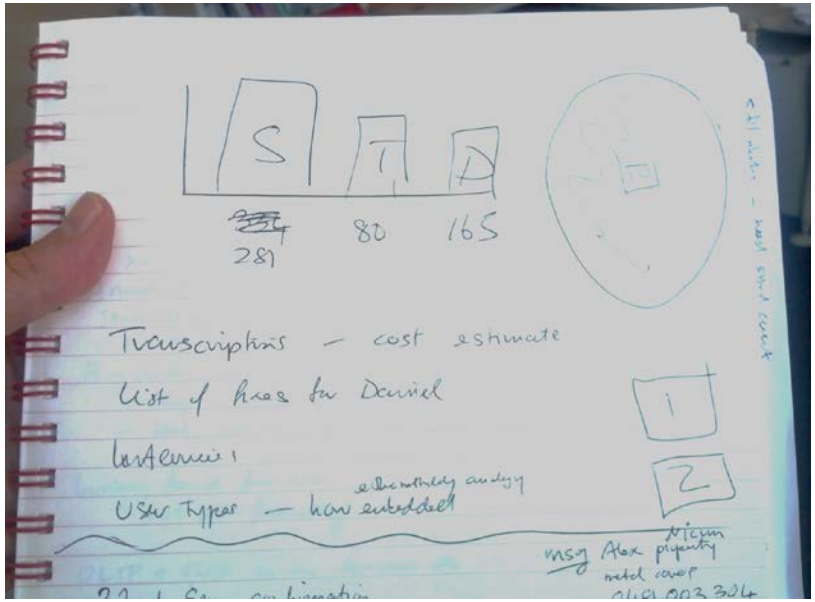
© The University of Melbourne 2018

Introduction to databases



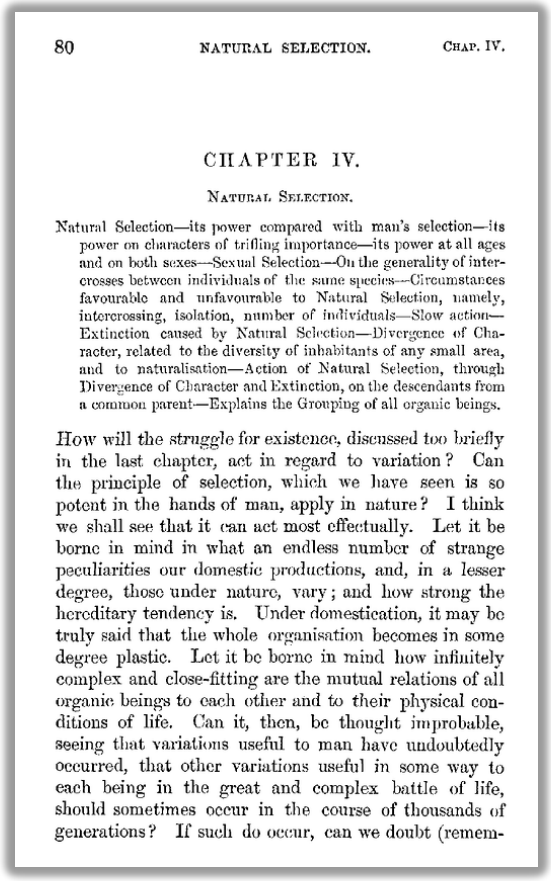
© The University of Melbourne 2018

Unstructured data



- handwritten notes
- printed books
- spreadsheets etc

	A	B	C	D	E
1	My Holiday Budget ----->				
2					
3	Flights	$10,000.00			
4	Accommodation	$5,000.00	<- can this be reduced?		
5	Food	$1,000.00			
6					
7		$16,000.00	TOTAL		
8					
9					
10					



80 NATURAL SELECTION. CHAP. IV.

CHAPTER IV.

NATURAL SELECTION.

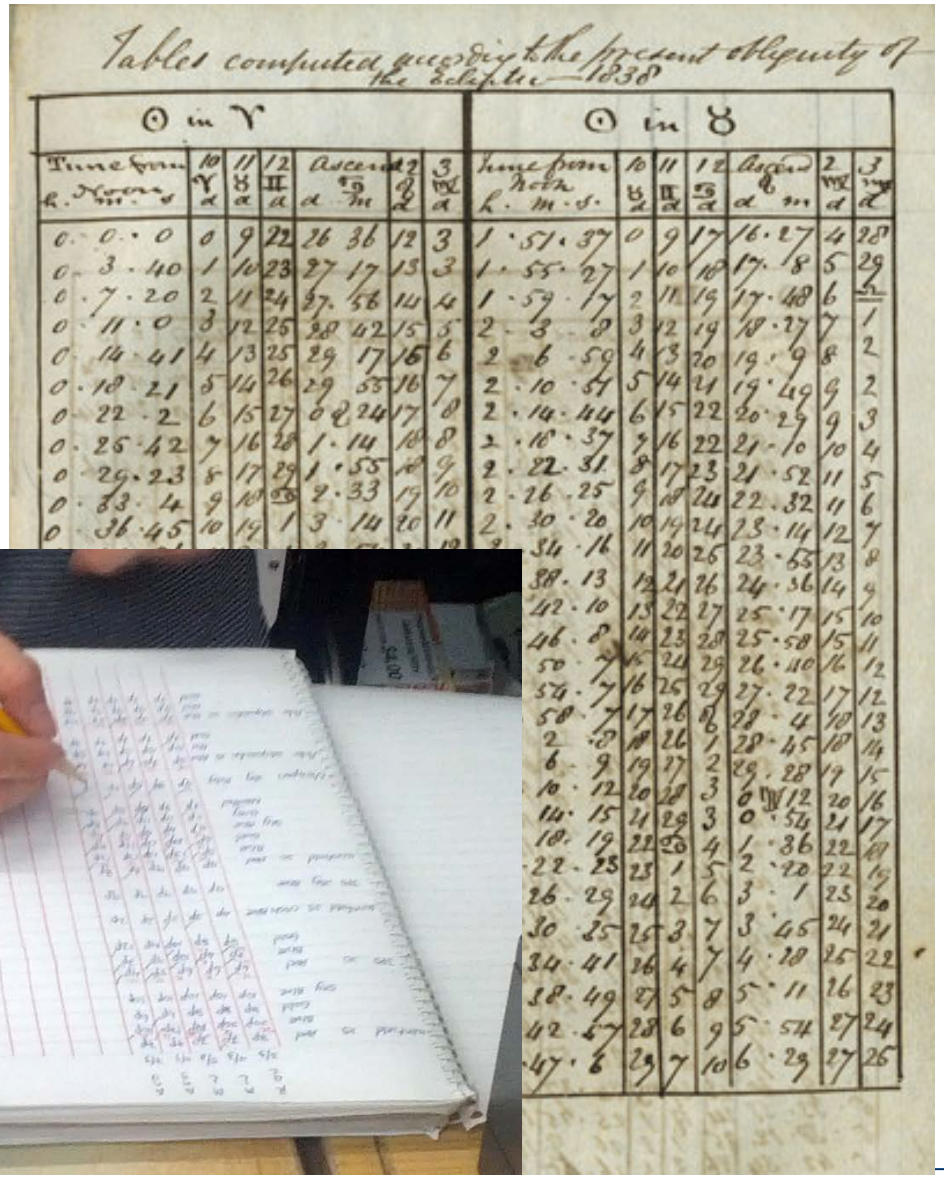
Natural Selection—its power compared with man's selection—its power on characters of trifling importance—its power at all ages and on both sexes—Sexual Selection—On the generality of intercrosses between individuals of the same species—Circumstances favourable and unfavourable to Natural Selection, namely, intercrossing, isolation, number of individuals—Slow action—Extinction caused by Natural Selection—Divergence of Character, related to the diversity of inhabitants of any small area, and to naturalisation—Action of Natural Selection, through Divergence of Character and Extinction, on the descendants from a common parent—Explains the Grouping of all organic beings.

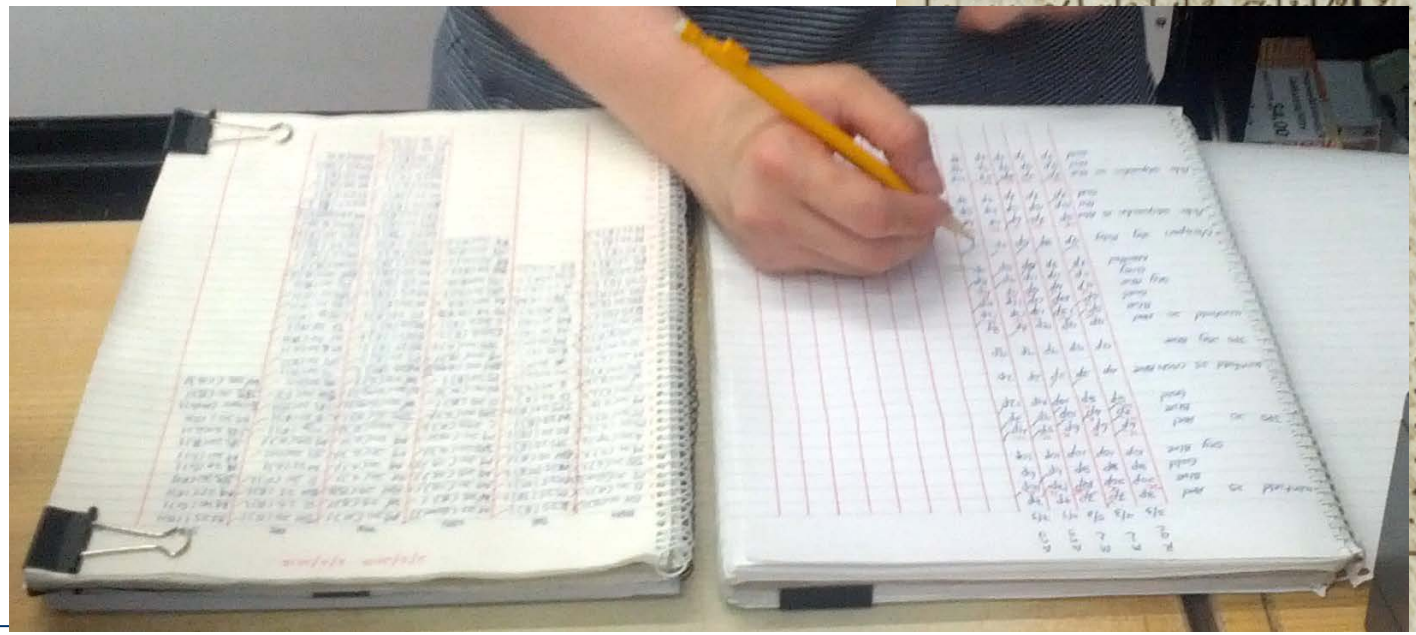
How will the struggle for existence, discussed too briefly in the last chapter, act in regard to variation? Can the principle of selection, which we have seen is so potent in the hands of man, apply in nature? I think we shall see that it can act most effectually. Let it be borne in mind in what an endless number of strange peculiarities our domestic productions, and, in a lesser degree, those under nature, vary; and how strong the hereditary tendency is. Under domestication, it may be truly said that the whole organisation becomes in some degree plastic. Let it be borne in mind how infinitely complex and close-fitting are the mutual relations of all organic beings to each other and to their physical conditions of life. Can it, then, be thought improbable, seeing that variations useful to man have undoubtedly occurred, that other variations useful in some way to each being in the great and complex battle of life, should sometimes occur in the course of thousands of generations? If such do occur, can we doubt (remem-

© The University of Melbourne 2018

Tables

- Each row represents an instance of a set of entities
- Entities might be people, things, events, transactions…
- You can extract useful knowledge via simple, repetitive processes
- A natural fit for business and scientific data.

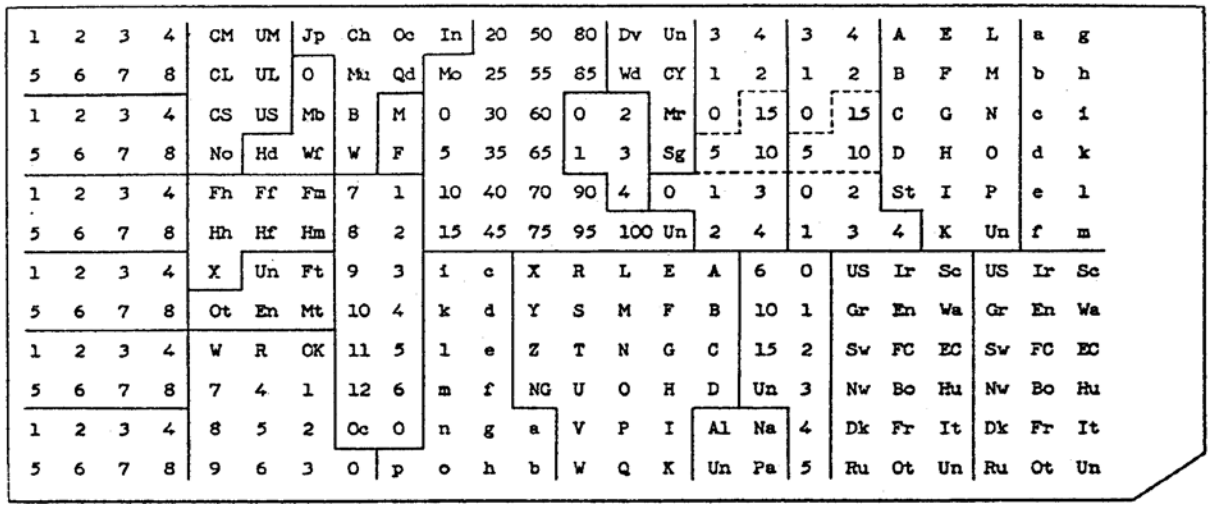




© The University of Melbourne 2018

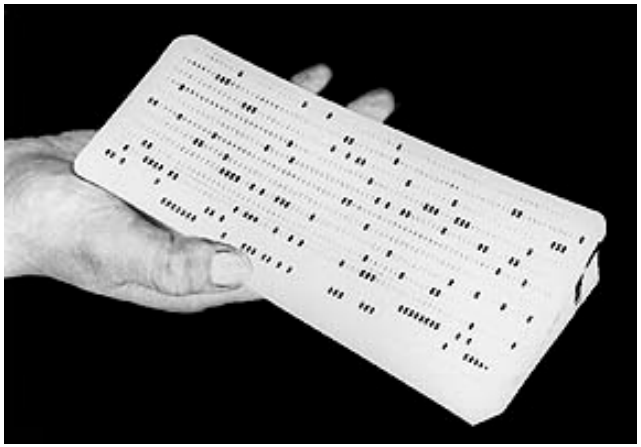
Table of rows -> Deck of cards

1890 US census



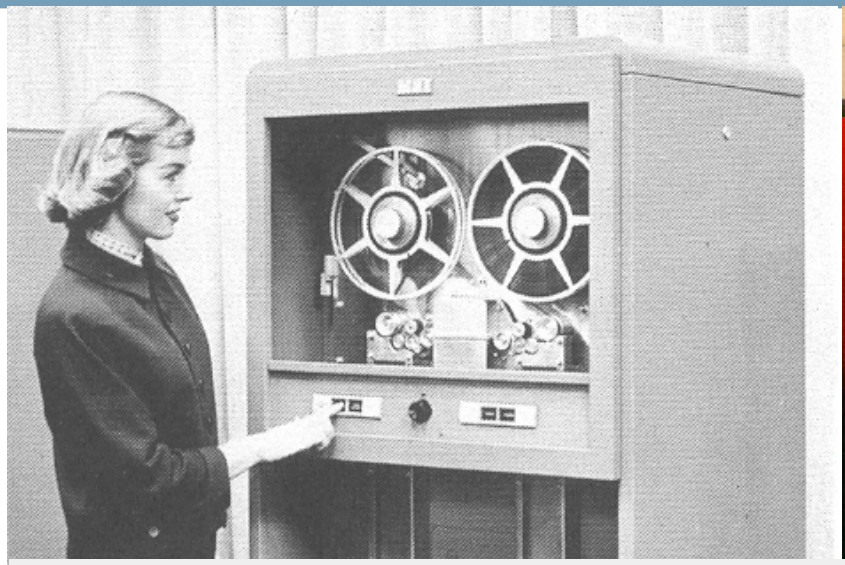
1	2	3	4	CM	UM	Jp	Ch	Oc	In	20	50	80	Dv	Un	3	4	3	4	A	E	L	a	g
5	6	7	8	CL	UL	O	Mu	Qd	Mo	25	55	85	Wd	CY	1	2	1	2	B	F	M	b	h
1	2	3	4	CS	US	Mb	B	M	O	30	60	O	2	Mr	O	15	O	15	C	G	N	c	i
5	6	7	8	No	Hd	Wf	W	F	5	35	65	1	3	Sg	5	10	5	10	D	H	O	d	k
1	2	3	4	Fh	Ff	Fm	7	1	10	40	70	90	4	O	1	3	O	2	St	I	P	e	l
5	6	7	8	Hh	Hf	Hm	8	2	15	45	75	95	100	Un	2	4	1	3	4	K	Un	f	m
1	2	3	4	X	Un	Ft	9	3	i	c	X	R	L	E	A	6	O	US	Ir	Sc	US	Ir	Sc
5	6	7	8	Ot	En	Mt	10	4	k	d	Y	S	M	F	B	10	1	Gr	En	Wa	Gr	En	Wa
1	2	3	4	W	R	OK	11	5	l	e	Z	T	N	G	C	15	2	Sw	FC	EC	Sw	FC	EC
5	6	7	8	7	4	1	12	6	m	f	NG	U	O	H	D	Un	3	Nw	Bo	Hu	Nw	Bo	Hu
1	2	3	4	8	5	2	Oc	O	n	g	a	V	P	I	Al	Na	4	Dk	Fr	It	Dk	Fr	It
5	6	7	8	9	6	3	O	p	o	h	b	W	Q	K	Un	Pa	5	Ru	Ot	Un	Ru	Ot	Un

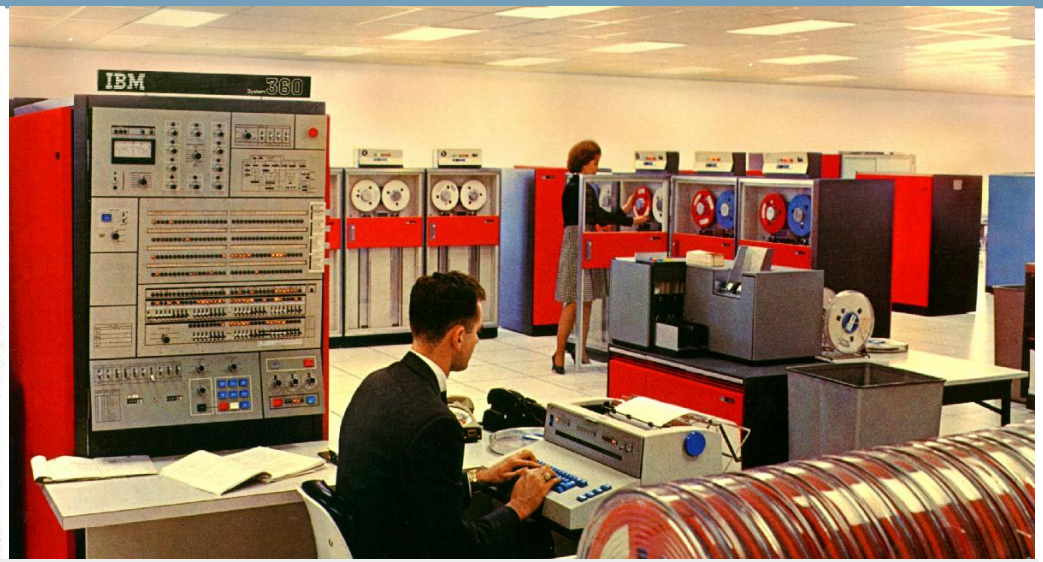
https://www.census.gov/history/www/innovations/technology/the_hollerith_tabulator.html



© The University of Melbourne 2018

Tape and Disc





EMP_RECORD...	EMP_ID	EMP_REGION	EMP_DEPT	EMP_HIRE_D...	E...	E...	EMP_...	EMP_SALARY	EMP_NAME	EMP_SKIL
68	3715	4	153	09061987	9	6	1987	14000000	IRENE HIRSH	041085
62	39412	1	650	03119590	3	11	9590	167000000	ANN FAHEY	031099
56	1939	2	265	09281988	9	28	1988	21300000	EMILY WILM...	021077
50	3502	2	165	07041985	7	4	1985	19500000	CATHEZINE ...	011015
44	4435	2	117	05141989	5	14	1989	17000000	AGNES KING	00
68	1673	3	138	07021985	7	2	1985	16800000	MARTIN XU	041033
62	4181	3	161	02031988	2	3	1988	15900000	JOHN DURN	030045
56	1443	1	265	12028900	12	2	8900	6000000	PAT DUNN	021055
50	3607	3	127	08072000	8	7	2000	18300000	ANDREA HIN...	011014
44	1775	3	288	02051989	2	5	1989	2700000	PETER JONES	00
68	1209	2	165	05121986	5	12	1986	17300000	DIDRA WILK...	041065

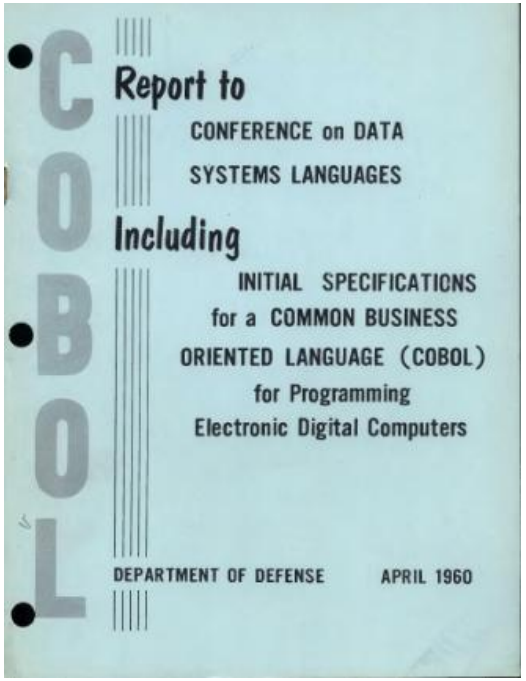
http://www.computerhistory.org/timeline/memory-storage/

© The University of Melbourne 2018

File Processing

http://groups.engin.umd.umich.edu/CIS/course.des/cis400/cobol/cobol.html

```
DATA DIVISION.
FILE SECTION.
FD StudentFile.
01 StudentRec.
   88  EndOfStudentFile  VALUE HIGH-VALUES.
   02  StudentId         PIC 9(7).
   02  StudentName.
       03 Surname        PIC X(8).
       03 Initials       PIC XX.
   02  DateOfBirth.
       03 YOBirth        PIC 9(4).
       03 MOBirth        PIC 9(2).
       03 DOBirth        PIC 9(2).
   02  CourseCode        PIC X(4).
   02  Gender            PIC X.
```



Problems with flat-files:

- data access routines must be programmed in detail
- each program must include full detail of data structure
- multiple users cannot simultaneously access data
- multiple copies of data - not centrally managed

© The University of Melbourne 2018

Relational Databases

- the first relational databases from Oracle and IBM appear around 1980

ORACLE®

IBM DB2

- others appear later

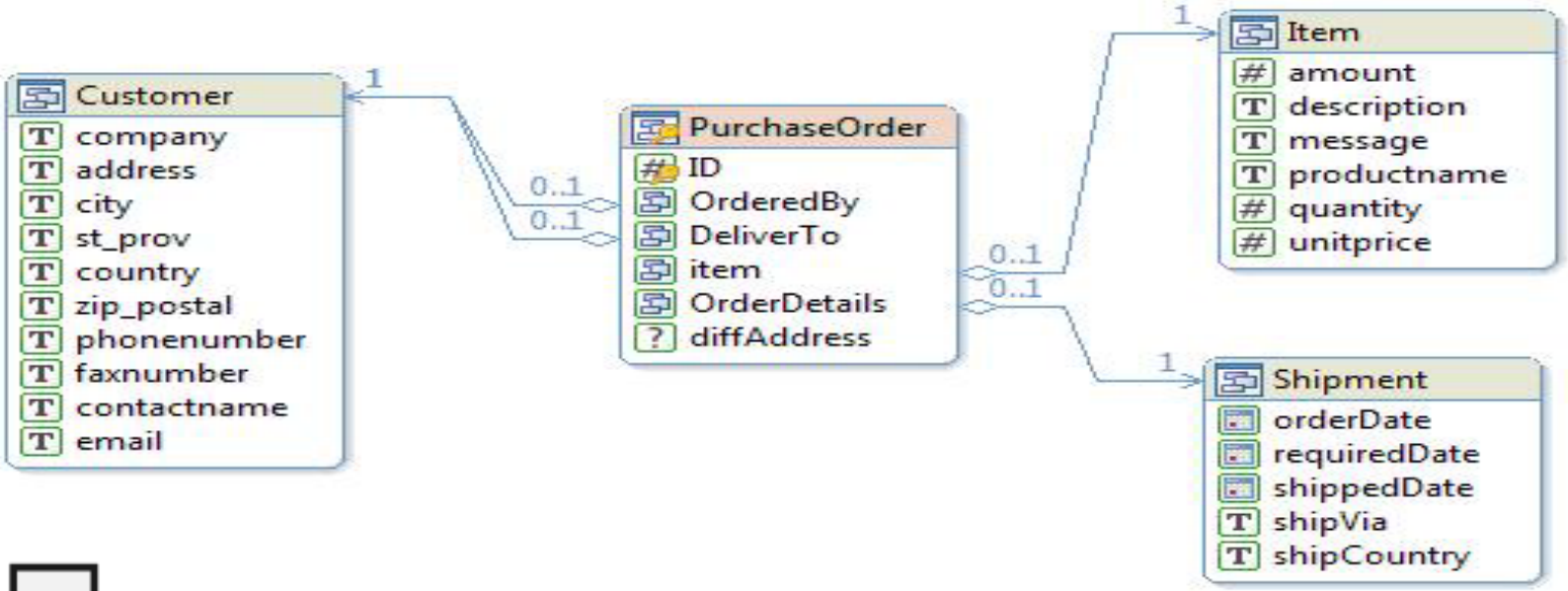
Microsoft® SQL Server®

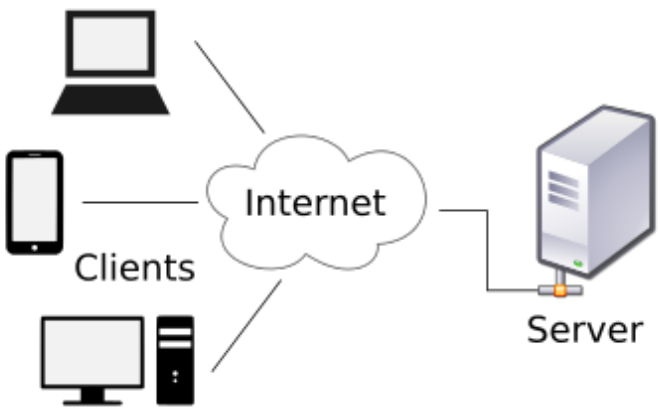
MySQL™

© The University of Melbourne 2018

Relational Databases

- entity-relationship diagrams, client-server architecture, SQL language

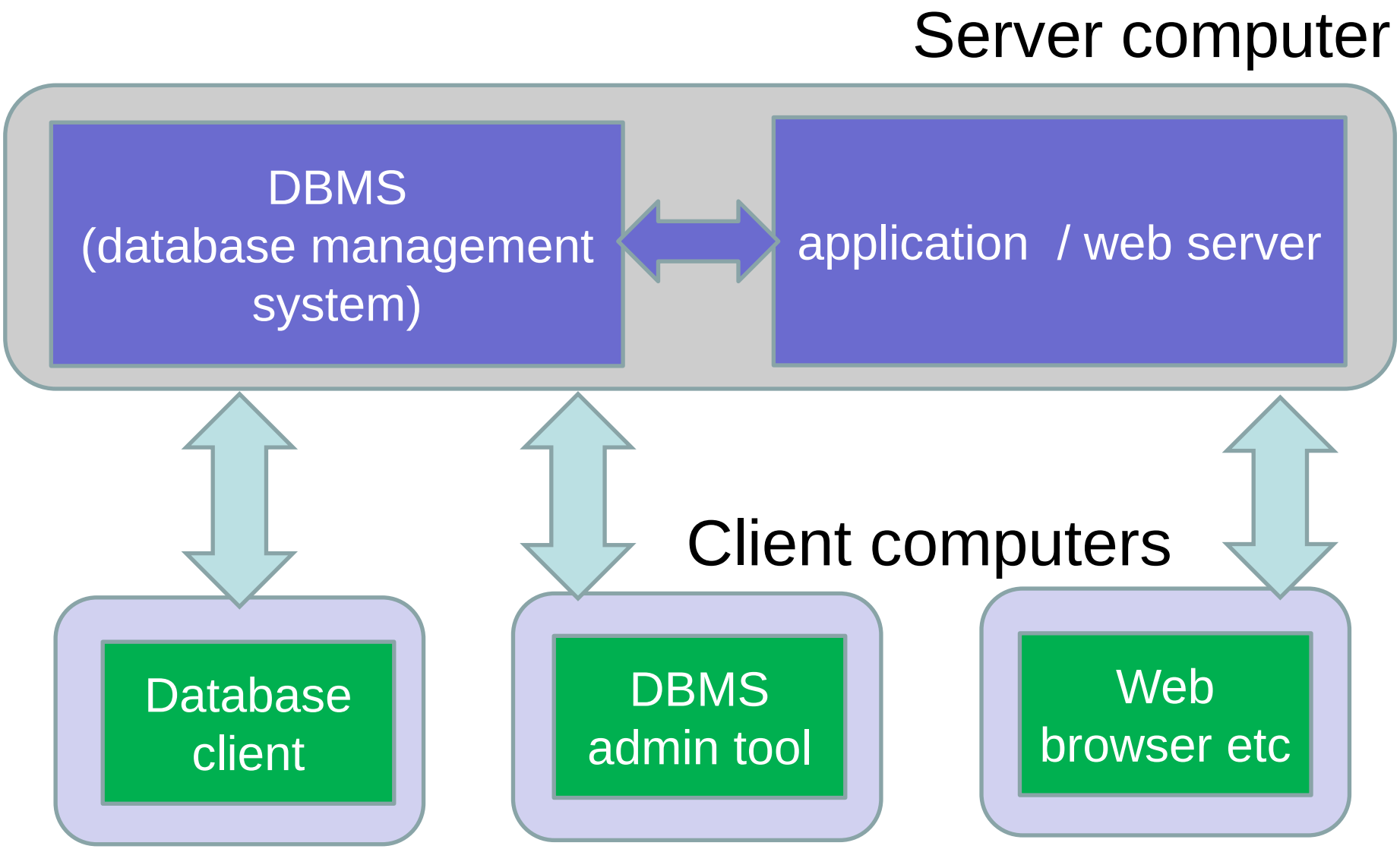




SELECT address
FROM Customer
WHERE Country = ‘USA’;

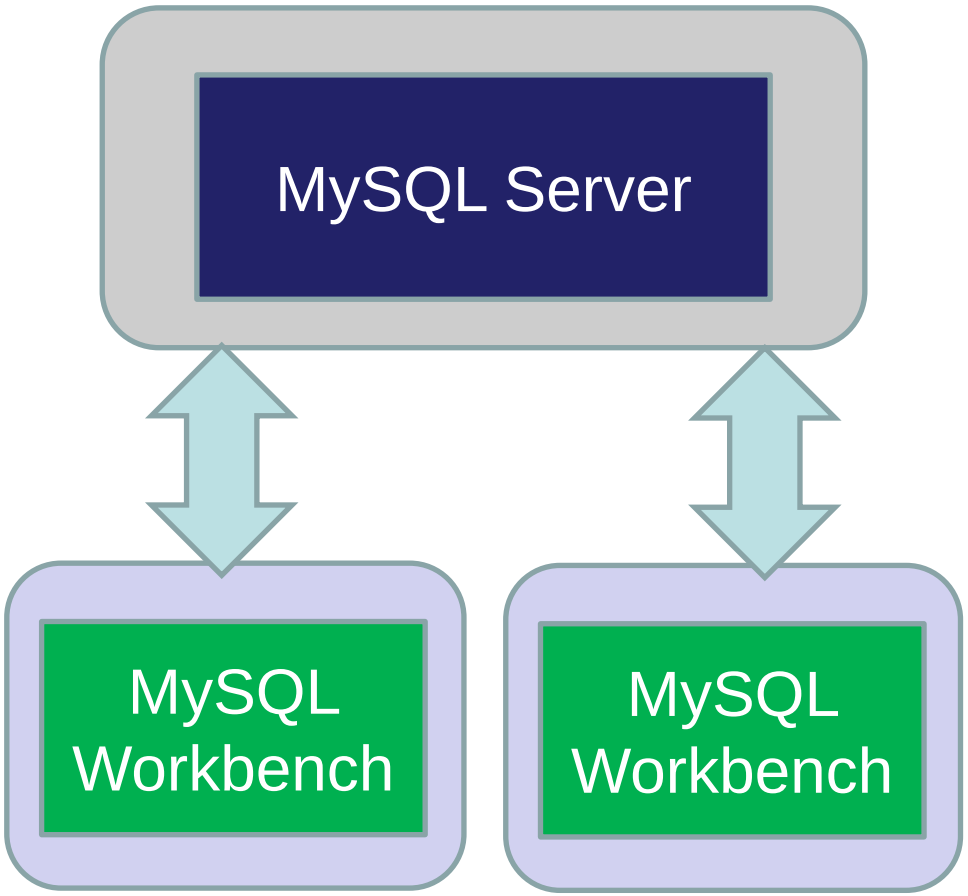
© The University of Melbourne 2018

Client-Server … in industry

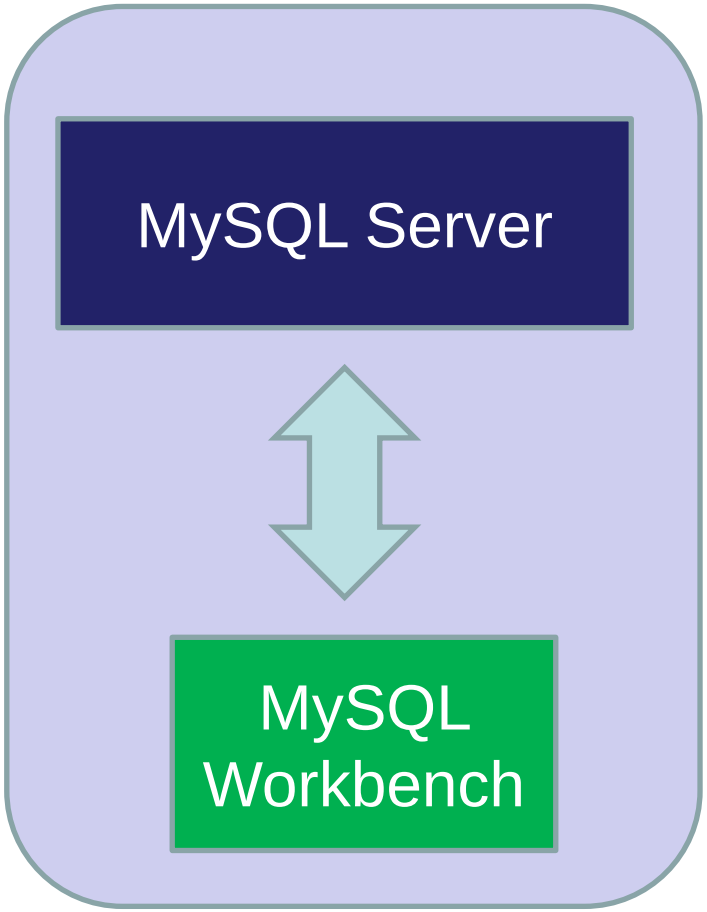


© The University of Melbourne 2018

… in teaching and development



Labs

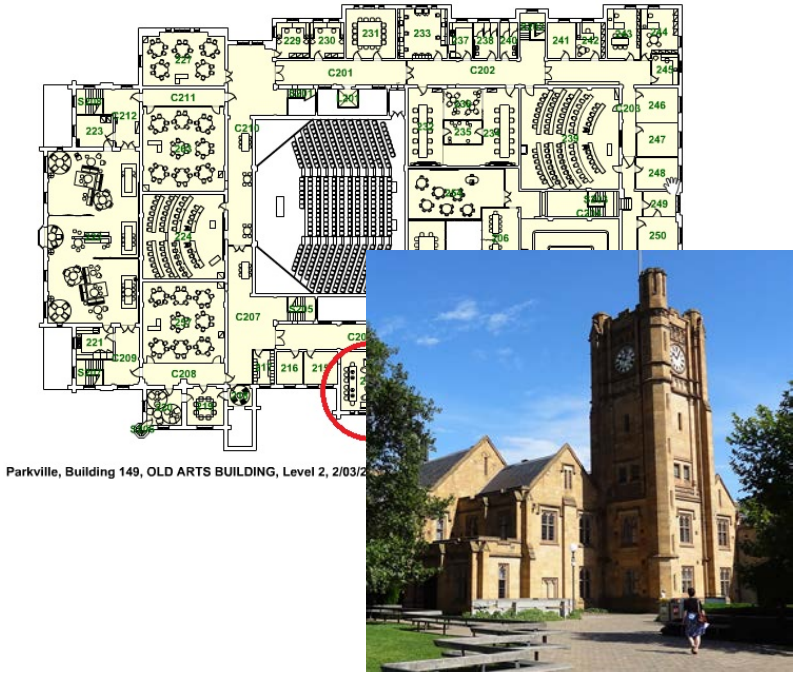


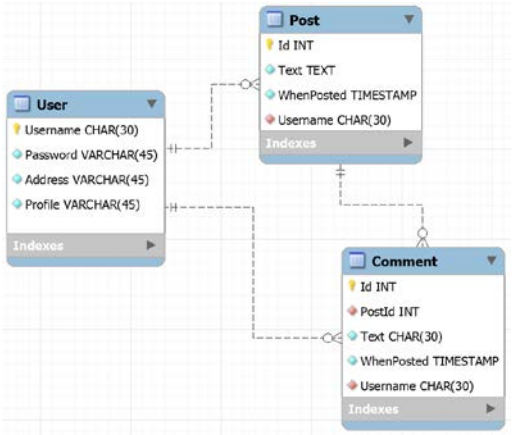
Your PC

© The University of Melbourne 2018

Let’s design a database …

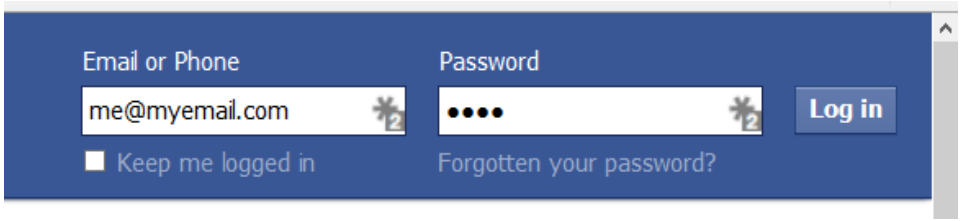
“What is unique about ICT ... In no other discipline is there such an emphasis on developing artefacts (e.g., computer and information systems) which are so abstract and complex and where modelling tools and methods are essential. The systems that ICT professionals deal with cannot be seen or handled in the same simple and direct manner as products of other applied disciplines …“ *from the Australian Computer Society CBOK*





© The University of Melbourne 2018

Let's design a database …



User

Username	Password	Address	Profile
Anne	pass1234	1 Anne st	hi I am Anne
Bill	petsname	2 Bill st	this is Bill's profile
Christine	mystreet	3 Christine st	hi everyone this is Christine!!!
David	childsname	4 David st	David's profile data

Post

Id	Text	WhenPosted	Username
1	Here's what I had for lunch	2015-01-24 20:09:02	Bill
2	What's everyone doing tonight?	2015-01-24 20:09:02	Anne
3	check out this great CAT VIDEOZ!!!	2015-01-24 20:09:02	David
4	now look what I had for dinner	2015-01-24 20:09:02	Bill
5	I love Game of Thrones	2015-01-24 20:09:02	Anne
6	I am posting a lot today…	2015-01-24 20:09:02	Anne

Comment

Id	PostId	Text	WhenPosted	Username
1	3	ha ha great video!!	2015-01-24 14:01:33	Christine
2	2	going to the movies	2015-01-24 14:01:33	Bill
3	2	oh that sound nice!	2015-01-24 14:01:33	Anne

© The University of Melbourne 2018

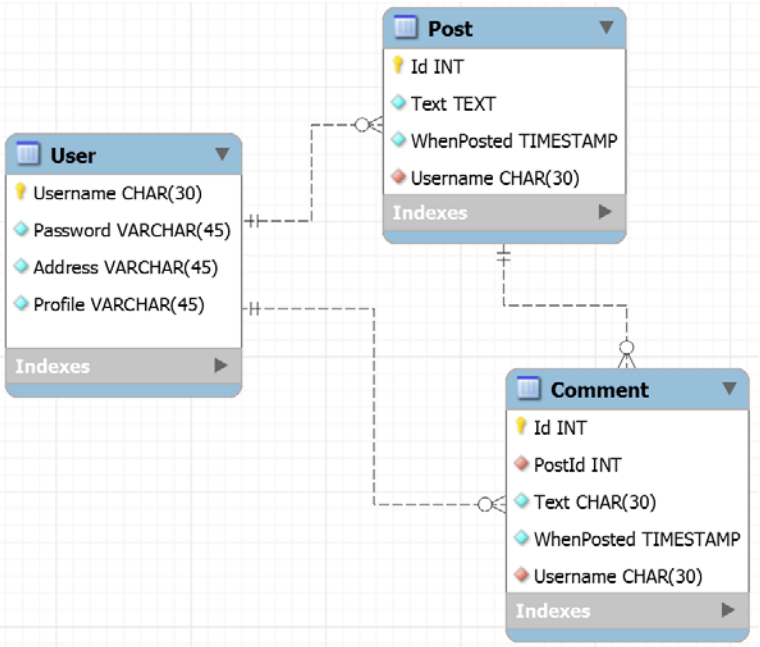
Let's design a database …

Username	Password	Address	Profile
Anne	pass1234	1 Anne st	hi I am Anne
Bill	petsname	2 Bill st	this is Bill's profile
Christine	mystreet	3 Christine st	hi everyone this is Christine!!!
David	childsname	4 David st	David's profile data

Id	Text	WhenPosted	Username
1	Here's what I had for lunch	2015-01-24 20:09:02	Bill
2	What's everyone doing tonight?	2015-01-24 20:09:02	Anne
3	check out this great CAT VIDEOZ!!!	2015-01-24 20:09:02	David
4	now look what I had for dinner	2015-01-24 20:09:02	Bill
5	I love Game of Thrones	2015-01-24 20:09:02	Anne
6	I am posting a lot today…	2015-01-24 20:09:02	Anne

Id	PostId	Text	WhenPosted	Username
1	3	ha ha great video!!	2015-01-24 14:01:33	Christine
2	2	going to the movies	2015-01-24 14:01:33	Bill
3	2	oh that sound nice!	2015-01-24 14:01:33	Anne

Entity-Relationship diagram:



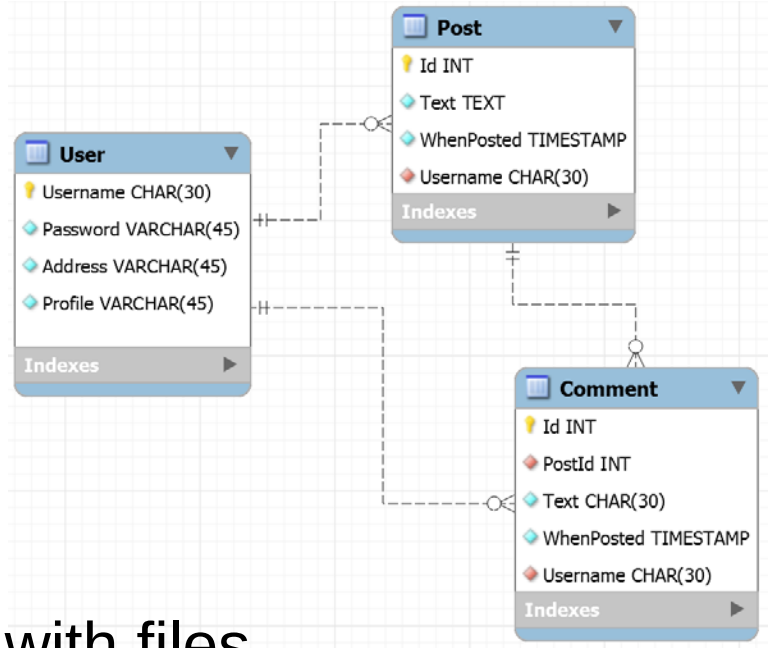
Relational notation:

User(<u>username</u>, password, address, profile)
Post(<u>id</u>, text, whenposted, *username*)
Comment(<u>id</u>, *postid*, text, whenposted)

© The University of Melbourne 2018

Maintaining structure of database

What commands do we need to manipulate the *structure* of our tables?



Working with tables is like working with files – there are 4 things you can do:

CREATE a table

DROP (i.e. delete) a table

ALTER a table (e.g. add a column)

RENAME a table

© The University of Melbourne 2018

Maintaining contents of database

What commands do we need to manipulate the *contents* of our tables?

User

Username	Password	Address	Profile
Anne	pass1234	1 Anne st	hi I am Anne
Bill	petsname	2 Bill st	this is Bill's profile
Christine	mystreet	3 Christine st	hi everyone this is Christine!!!
David	childsname	4 David st	David's profile data

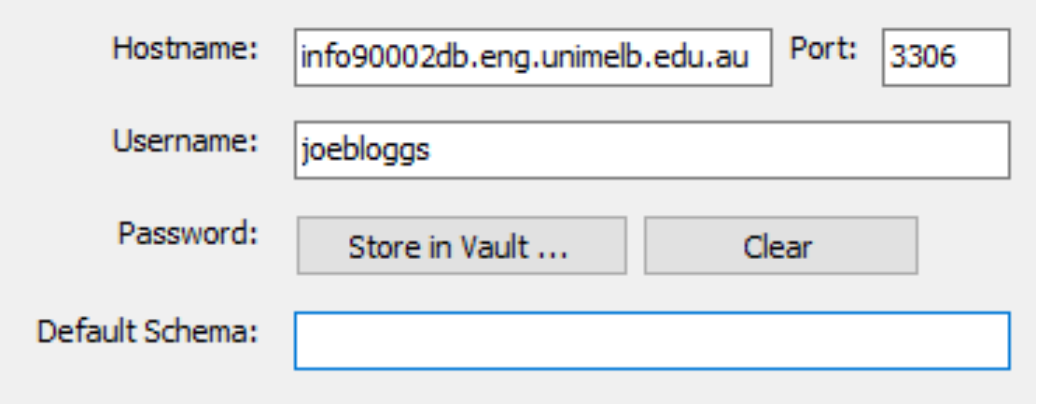
You simply need to be able to:
SELECT, or read, data from the table
INSERT new rows into the table
DELETE existing rows from the table
UPDATE existing rows in the table

© The University of Melbourne 2018

Maintaining users' access to database

What commands do we need to control *users' access* to our tables?

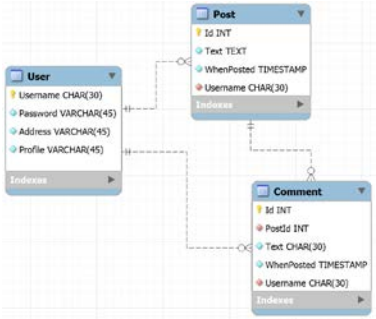
(users = I.T. staff here)



There are 4 things you can do:
CREATE a user
DROP a user
GRANT a user access rights to a table
REVOKE those rights

© The University of Melbourne 2018

Database lifecycle



- Design the database
  - data modelling, E-R diagrams
- Implement the database
  - data definition language DDL
- Data access / programming
  - data manipulation language DML
- Database administration
  - data control language DCL

•Create
•Drop
•Alter
•Rename

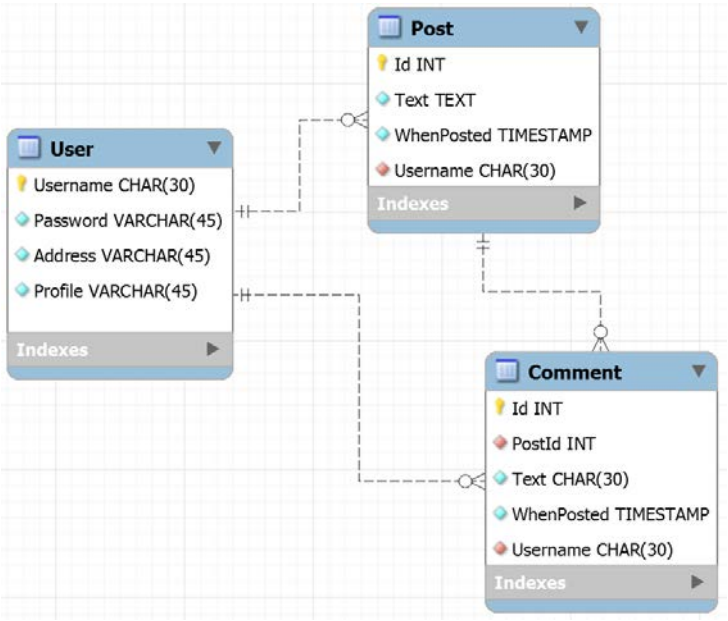
•Select
•Insert
•Update
•Delete

•Grant
•Revoke

© The University of Melbourne 2018

Data Modelling

- What are the entities that need to be tracked?
- What attributes will be recorded about each entity?
- What are the relationships between entities?
- What are the cardinalities of relationships?



You will be given requirements such as:

"We have many users.
Users may enter posts into the system.
Each post is entered by exactly one user.
Users can comment on posts.
A post can be commented on many times."

© The University of Melbourne 2018

Noun-verb analysis

- An investment bank has a number of branches. Within each branch a number of departments operate and are structured in a hierarchical manner. The bank employs around 3000 staff who are assigned to work in the various departments across the branches. There are essentially three types of special employees where extra details required by the system. There are dealers who carry out investments who have limits imposed upon them for how much they can spend. There are IT compliance managers who's Basel2 role is required to be stored and there are HR managers that need have their assessment number recorded (along with other details not specified here).
- We need a database to record staff details including which department and branch they are assigned...

© The University of Melbourne 2018

Homework: Noun-verb analysis

- available on LMS
- do in your own time
- we'll discuss next week

© The University of Melbourne 2018

Dr Greg Wadley

THE UNIVERSITY OF MELBOURNE

INFO 90002 Database Systems & Information Modelling

Week 01

Introduction to Subject

Introduction to Relational Databases

Introduction to MySQL

Structure of today's lectures

- first hour: Introductions and admin
  - subject overview
  - staff and students
  - learning resources
  - assessment
- second hour: Introduction to databases
  - database technology, past present and future
  - how databases are designed, implemented and used
- third hour: Introduction to the *MySQL* RDBMS
  - client and server software
  - how to download and install
  - how to use in labs and at home

© The University of Melbourne 2018

Welcome!

- Welcome to INFO 90002
  Database Systems & Information Modelling

  Why this subject matters
  - database = key building block in many technology careers
  - database = one of the most widely-used technologies
    - embedded within most of the interesting ICT of today
      – social media, apps, websites, banking, scientific research ...
  - database have come to influence our culture
    - "The database is the major cultural form of the 21st century in much the same way as the novel was for the 19th and the film for the 20th. … While retaining the visual and temporal aspects of film, the modality of hypertext or of computer games eschews its linear modality for the modality of the database, in which objects are linked together but their assembly into a narrative experience is in the hands of the audience."
      Dourish and Mazmanian (2011), discussing Manovich (2002) *The Language of New Media*

© The University of Melbourne 2018

What are employers looking for?

Forbes (IT)	Forbes (NACE)	AIIA	Youth Central (Generic)	Dept of Employment
SQL	Teamwork	Cross-Cultural Competency	Communication	(ICT Sector) Lack of;
Mentoring	Solve Problems	Social Intelligence	Problem Solving	Industry Knowledge
ORACLE	Make Decisions	Novel & Adaptive Thinking	Teamwork	Technical Skills
Collaboration	Communicate	Virtual Collab.	Learning	Other Soft Skills
Process Improvement	Plan, Organise, Prioritise	New Media Literacy	Initiative and Enterprise	Communication Skills
Business Devt.	Obtain and Process Info	Computational Thinking	Self-management	
Decision Making	Quantitative Data Analysis	Design Mindset	Planning and Organising	
Data Analysis	Technical Skills	Resilience	Technical Skills	
SAP	Influencing	Transdisciplinarity		

© The University of Melbourne 2018

About the teaching staff

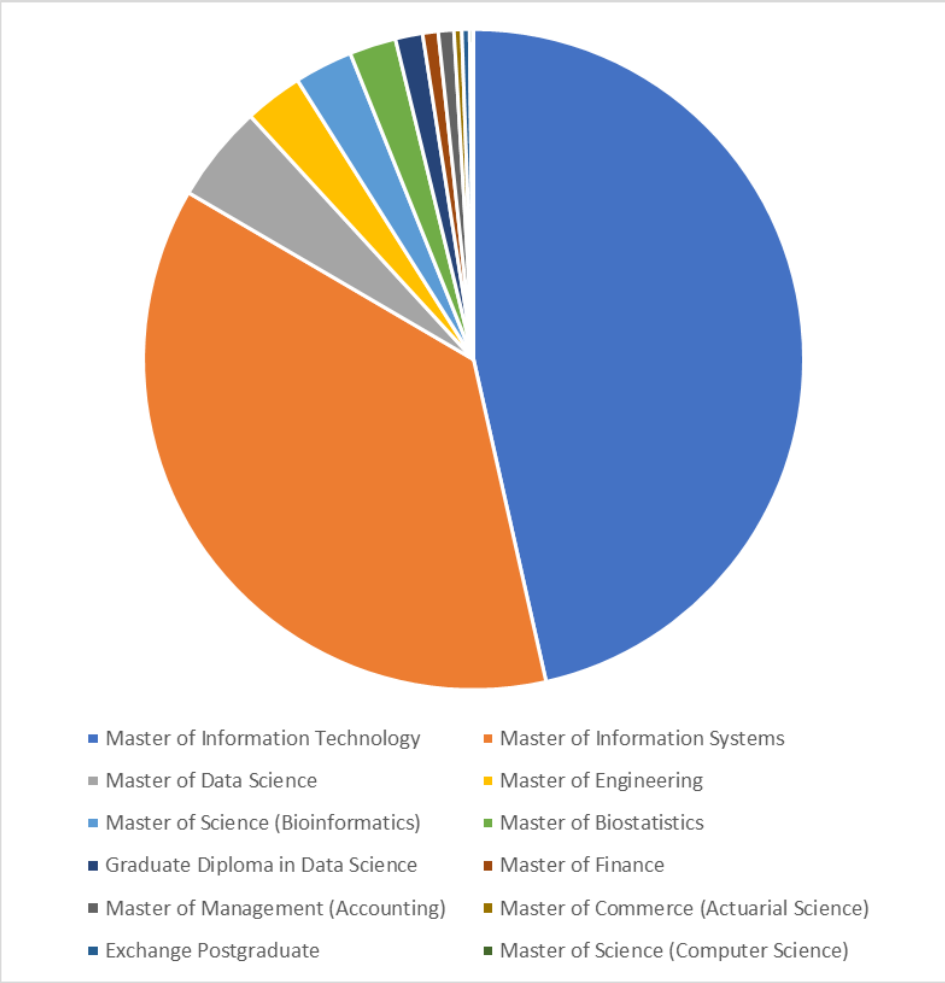
- Subject coordinators
  - Greg Wadley (lectures) and David Eccles (lectures and labs)
- Lab demonstrators
  - Neven, Imairi, Andreas, Ibrahim, Nick, Kat
- Student representative
  - (you? we are seeking a volunteer)
- Interacting with staff
  - in class
  - office hours (choose a time today) room 9.08 DMcD building
  - LMS discussion forum
  - email for *personal* questions only

© The University of Melbourne 2018

About you

- Your degree

Master of Information Technology	244
Master of Information Systems	194
Master of Data Science	25
Master of Engineering	15
Master of Science (Bioinformatics)	15
Master of Biostatistics	12
Graduate Diploma in Data Science	7
Master of Finance	4
Master of Management (Accounting)	4
Master of Commerce (Actuarial Science)	2
Exchange Postgraduate	2
Master of Science (Computer Science)	1



- Range of backgrounds
  - existing IT knowledge
  - academic and work history
  - career directions
  - local and international

© The University of Melbourne 2018

About the subject

- Prerequisites, not-allowed subjects, credit for experience
  - Have you studied DB already? Don't study it again - get credit!
- Semester schedule: the big picture
  - weeks 1 to 6: designing and using a db (data modelling, SQL)
  - weeks 7 to 10: advanced topics
  - weeks 11 to 12: industry trends, NoSQL, discuss assessment
- Assessment
  - assignment 1: data modelling (20%) .. groups of 3
  - assignment 2: SQL (10%) .. individual work
  - end of semester exam (70%, includes data modelling and SQL)
- How to succeed in this subject
  - practice especially DM and SQL skills
  - use all the learning resources provided

© The University of Melbourne 2018

Semester schedule

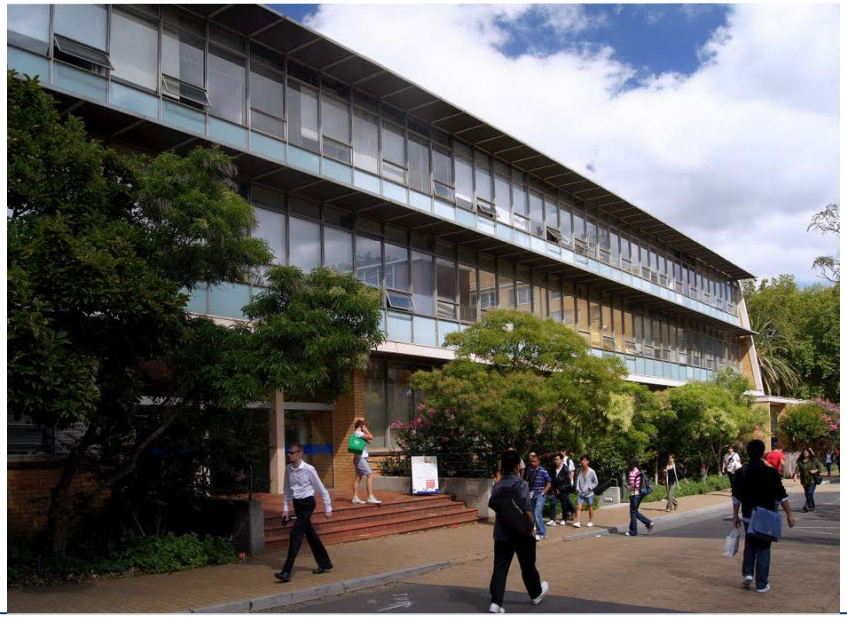
"Spiral" pedagogical pattern, see http://csis.pace.edu/~bergin/PedPat1.3.html#spiral

Week	Monday	Lecture 1	Lecture 2	Lab / Lecture 3	Hoffer	Extra reading	Homework	Assessment
1	23 Jul	Intro to subject	Intro to databases	Intro to MySQL	ch 1	Wikipedia, Hoffer video, HIstory of databases	noun-verb analysis	
2	30 Jul	Designing a database	Implementing a database	lab: data modelling, SQL , MySQL guide & install SQL script	2	SE Radio 'Relational Databases'	Real Estate model	assignment 1 released
3	6 Aug	Data modelling 1	SQL 1	lab: data modelling , SQL	2, 6	Simsion ch 1, Hoffer video	Uni Courses model	
4	13 Aug	Data modelling 2		lab: data modelling, SQL	3, 7	Simsion chapter 3, Hoffer video	how would you model an *invoice*?	
5	20 Aug	Normalization	Physical design	lab: SQL skills 1 & Solutions	4	Simsion chapter 4, Hoffer video, Kent (1983) Normalization		assignment 2 released
6	27 Aug	SQL 2		lab: SQL skills 2	5	podcast on 'SQL' MySQL data types		Asst 1 due
7	3 Sep	Databases in applications	Web apps	lab: SQL skills 3	3, 14	O'Reilly video: Intro to Web		
8	10 Sep	Transactions and concurrency (lecture by Dave Eccles)	Distributed databases (lecture by Dave Eccles)	lab: SQLReferential Integrity & scott.sql	11	podcast on 'CAP Theorem' Panel discussion on distributed		Asst 2 due
9	17 Sep	Database Administration (lecture by Dave Eccles)	Security and Ethics (lecture by Dave Eccles)	discuss Asst 1		MySQL database administration Oracle database administration		
	24 Sep	mid semester break - no classes						
10	1 Oct	NoSQL databases	... continued	discuss Asst 2		Martin Fowler NoSQL overview		
11	8 Oct	guest lecture to be announced	guest lecture to be announced	revision 1: you choose the topics	9			
12	15 Oct	Industry Trends	Wrapup, discuss exam	revision 2: you choose the topics		how Facebook stores data Information Week article		

© The University of Melbourne 2018

Labs

- Only run in weeks 2 through 8
- Not assessed, and attendance not recorded
- Demonstrator is there to help
- You can work in labs or at home
- Alice Hoy, Alan Gilbert, Doug McD, and Elec Eng buildings
- Tuesday - Friday
- Lab exercises on LMS



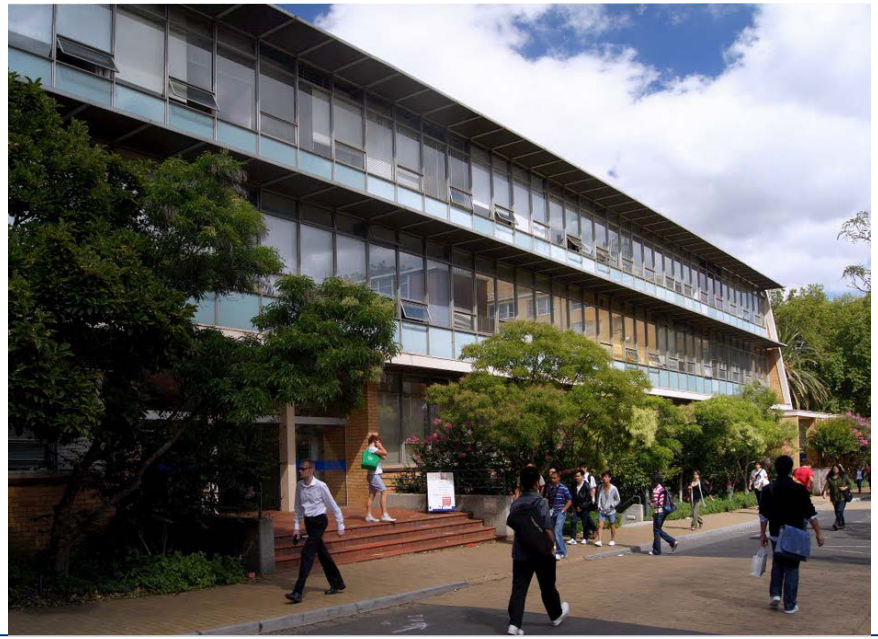
© The University of Melbourne 2018

Lab exercises

- Week 2 – 4 = data modelling, SQL

  (bring paper and pen for modelling!)

- Weeks 5 – 8 = SQL programming
  - a: learn by running example SQL provided
  - b: write SQL in response to a question



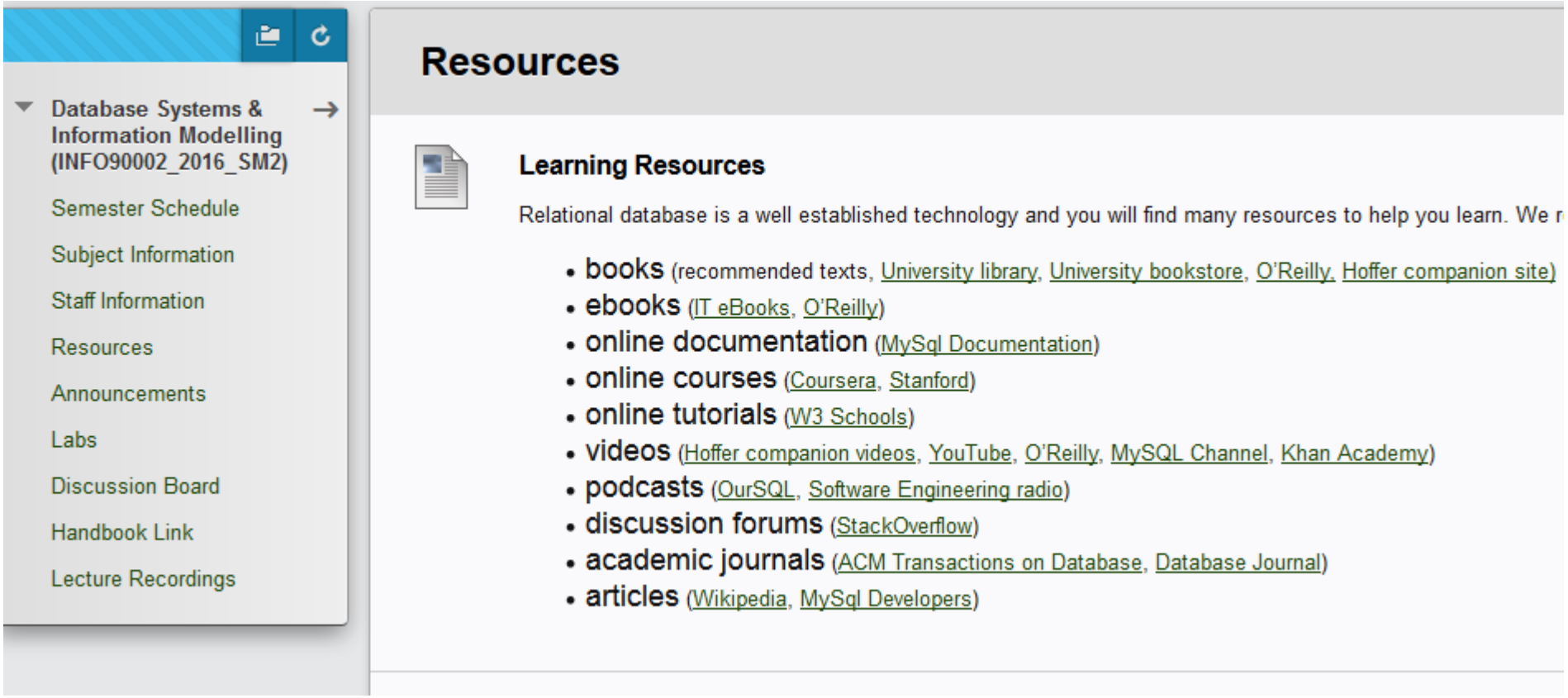
© The University of Melbourne 2018

Using database software

- We will use *MySQL* as server and client in this subject
- You can use either:
  - the University's database server
    - accessible from labs, or from home via VPN
    - your assignment 2 must be able to run on this server!
  - or, your own computer
    - do lab exercises and assignments at home
- Server address: info90002db.eng.unimelb.edu.au : port 3306
    - your username and password will be given out in first lab
    - not available outside the university without a VPN
- If you want to use your own computer …
  - download MySQL from http://dev.mysql.com/downloads/mysql/
  - get both Server and Workbench

© The University of Melbourne 2018

Teaching resources and LMS



© The University of Melbourne 2018

Dr Greg Wadley

THE UNIVERSITY OF MELBOURNE

INFO 90002 Database Systems & Information Modelling

Week 02

Designing and Implementing a Database

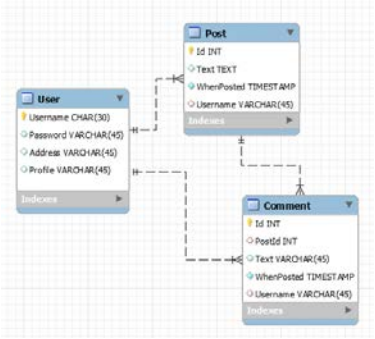
Structure of today's lecture

- first hour: **Designing Databases**
  - homework: noun-verb analysis
  - the database life-cycle
  - modelling a database for an example business
    - conceptual model
    - logical model
    - physical model
- second hour: **Implementing and Using Databases**
  - create the database and tables
  - populate tables with data
  - query data
  - change data

© The University of Melbourne 2018

Database Development Lifecycle

Recap week 1: Database lifecycle



- Design the database
  - data modelling, E-R diagrams
- Implement the database
  - data definition language (DDL)
- Data access / programming
  - data manipulation language (DML)
- Database administration
  - data control language (DCL)

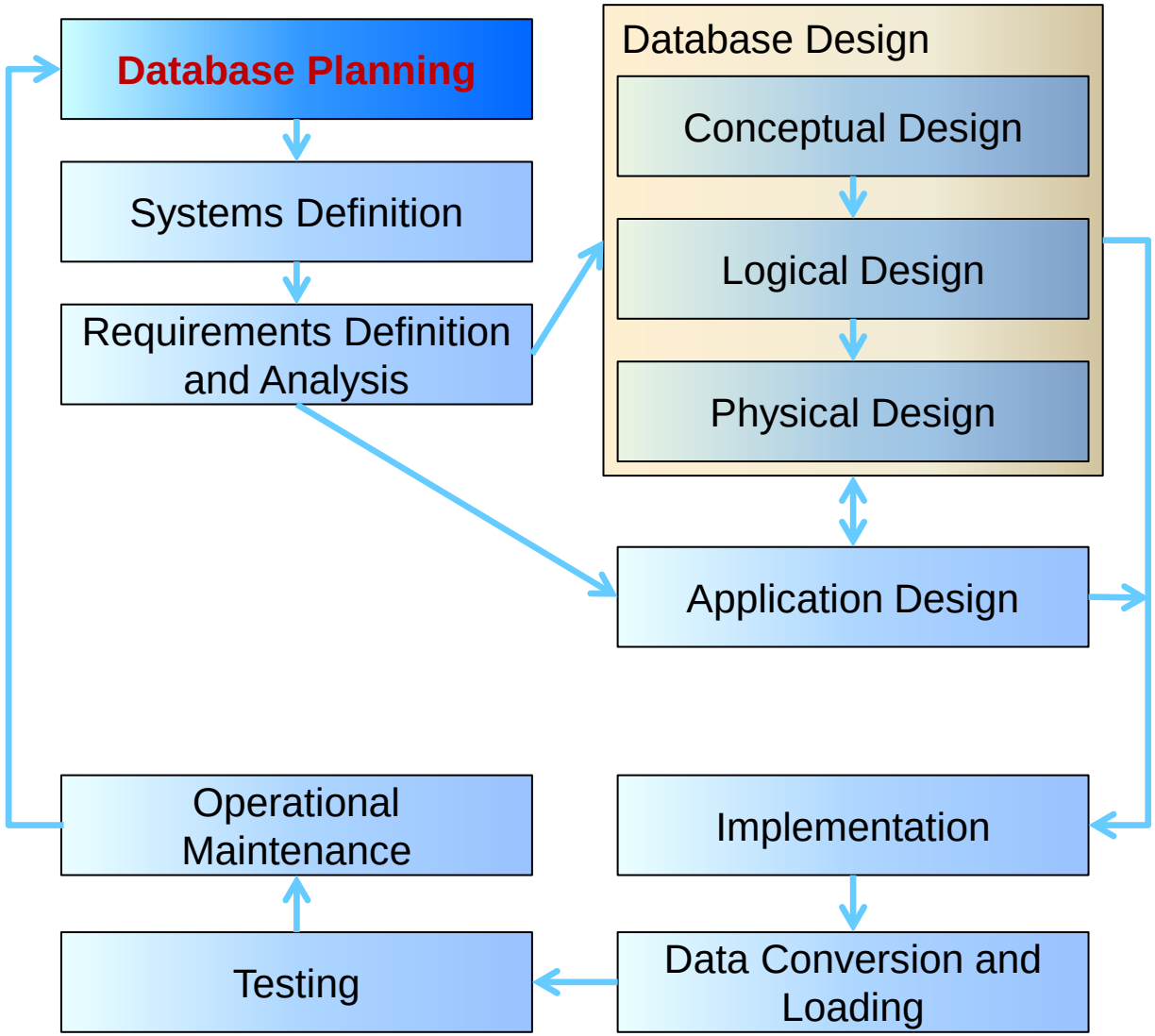
•Create
•Drop
•Alter
•Rename

•Select
•Insert
•Update
•Delete

•Grant
•Revoke

© The University of Melbourne 2018

Database Development Lifecycle

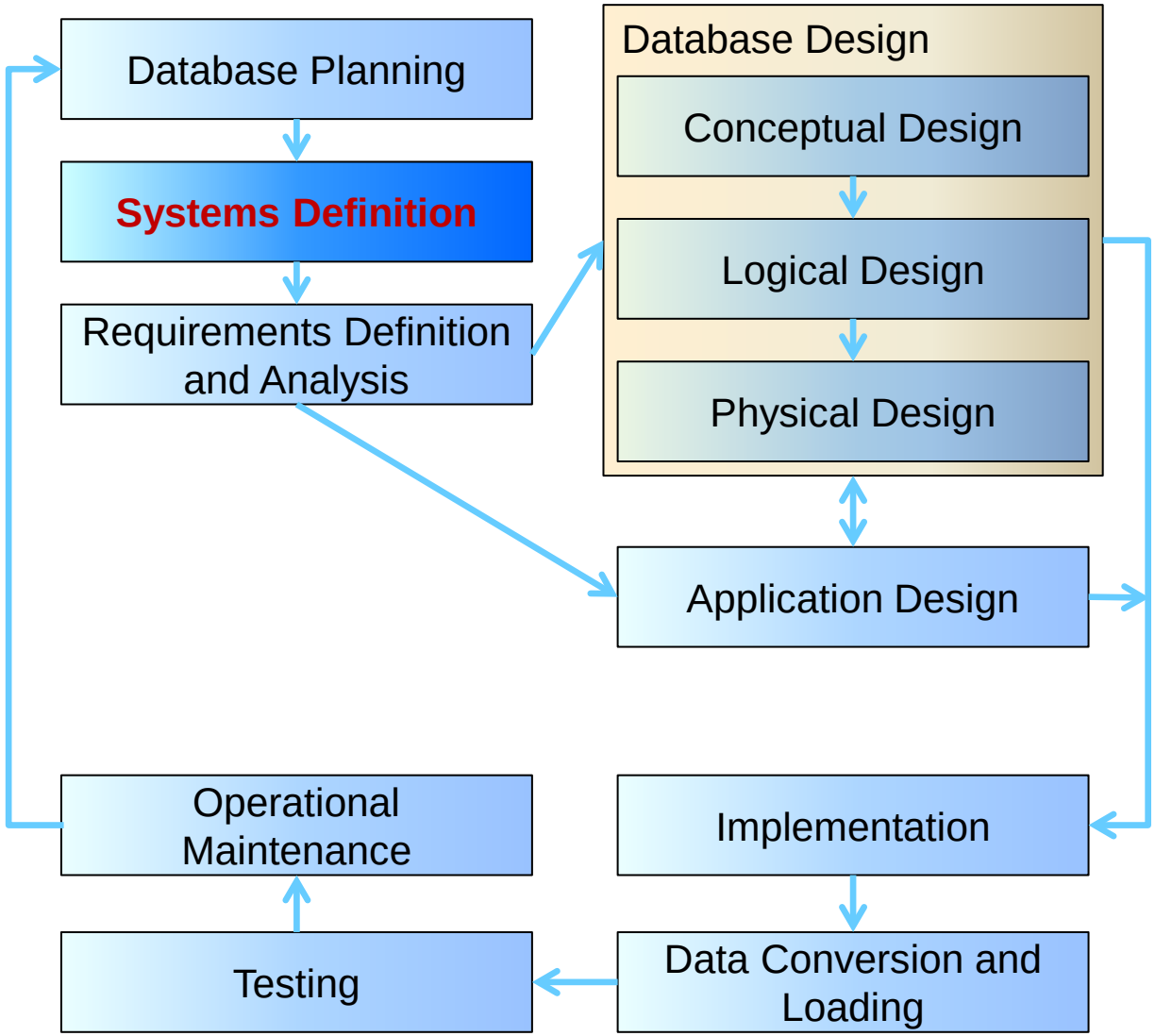


- Planning how to do the project.
  - How does the enterprise work
  - Enterprise data model
- How can the stages be completed efficiently and effectively.

- Outside scope of the course

© The University of Melbourne 2018

Database Development Lifecycle

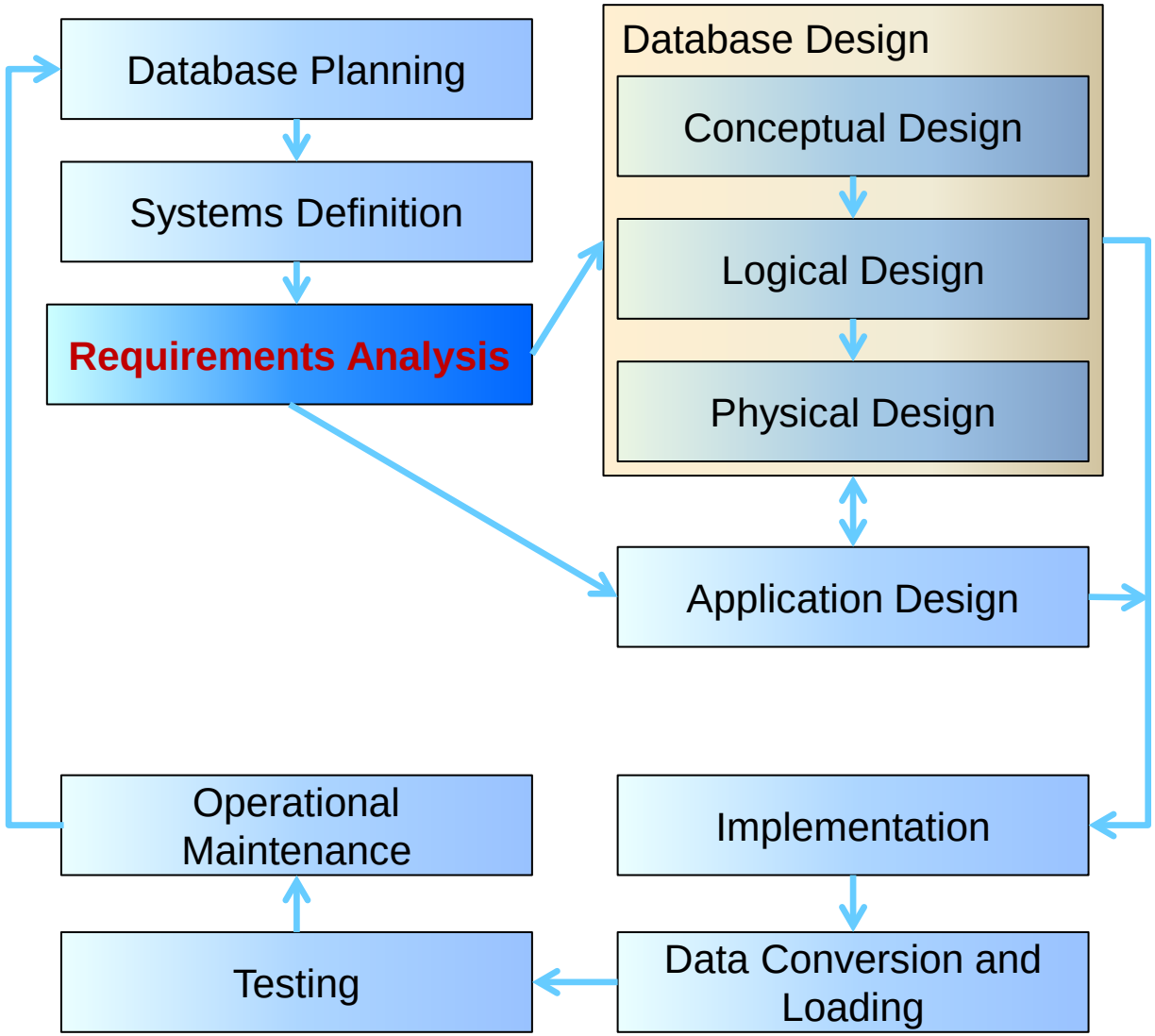


- Specifying scope and boundaries
  - Users
  - Major user views
  - Application areas
- How does it interact with other systems
- User views – how the system operates from differing perspectives

- Outside scope of the course (slightly)

© The University of Melbourne 2018

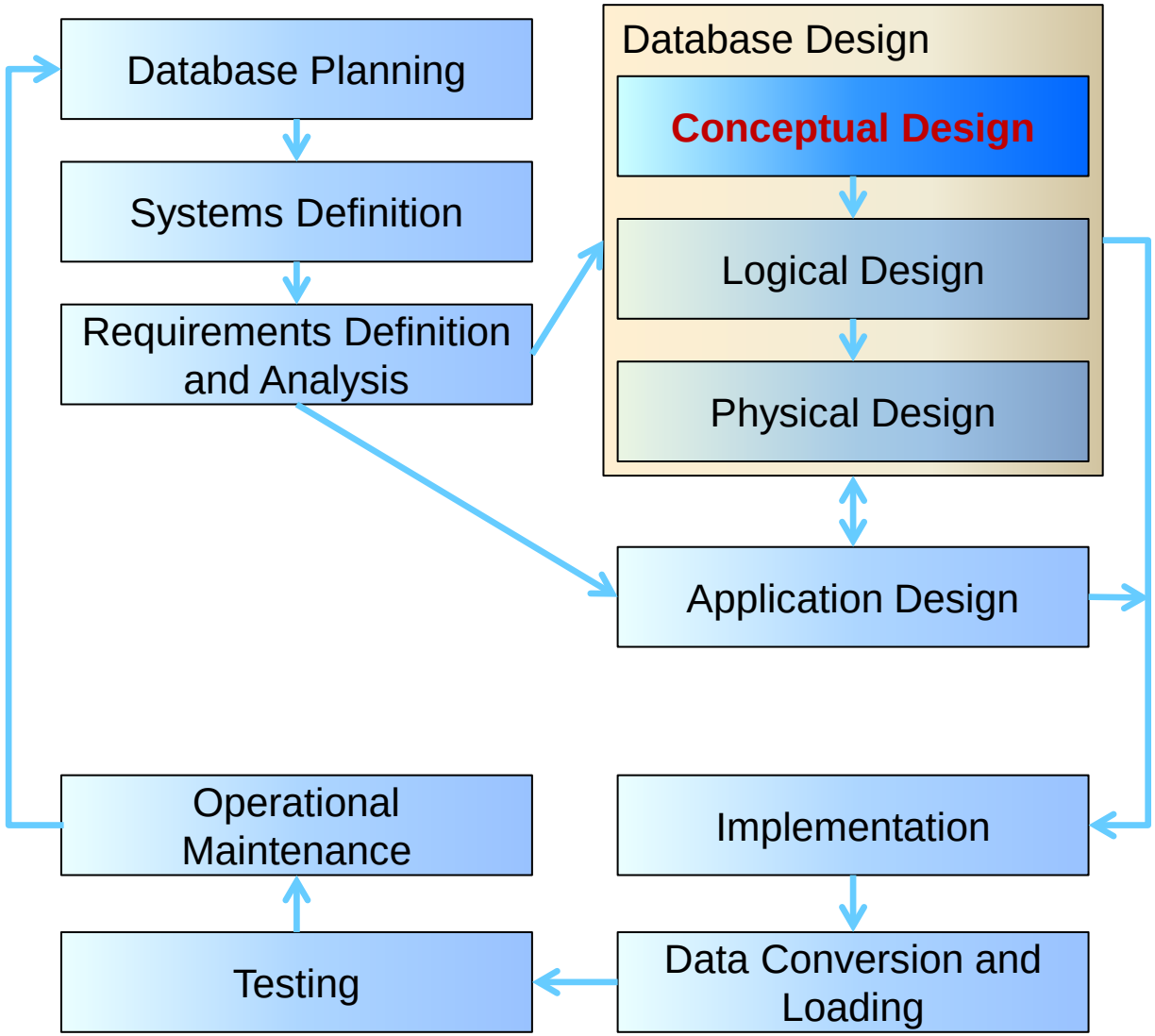
Database Development Lifecycle



- Collection and analysis of requirements for the new system

© The University of Melbourne 2018

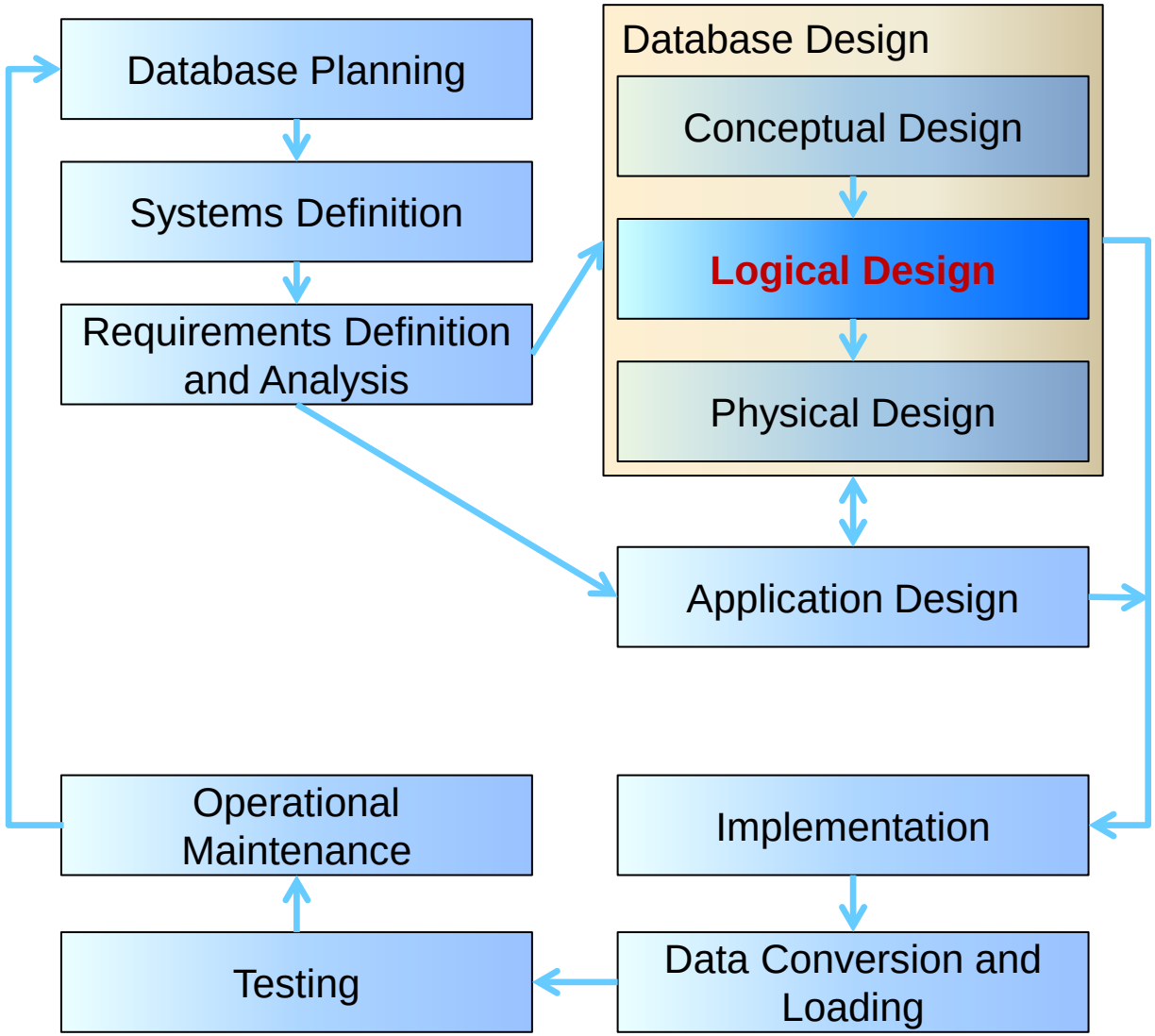
Database Development Lifecycle



- High-level, first-pass model of entities and their connections
- Typically omits attributes
- Could potentially be implemented in a non-relational database
- Thus can include many-to-many relationships, repeating groups, composite attributes

© The University of Melbourne 2018

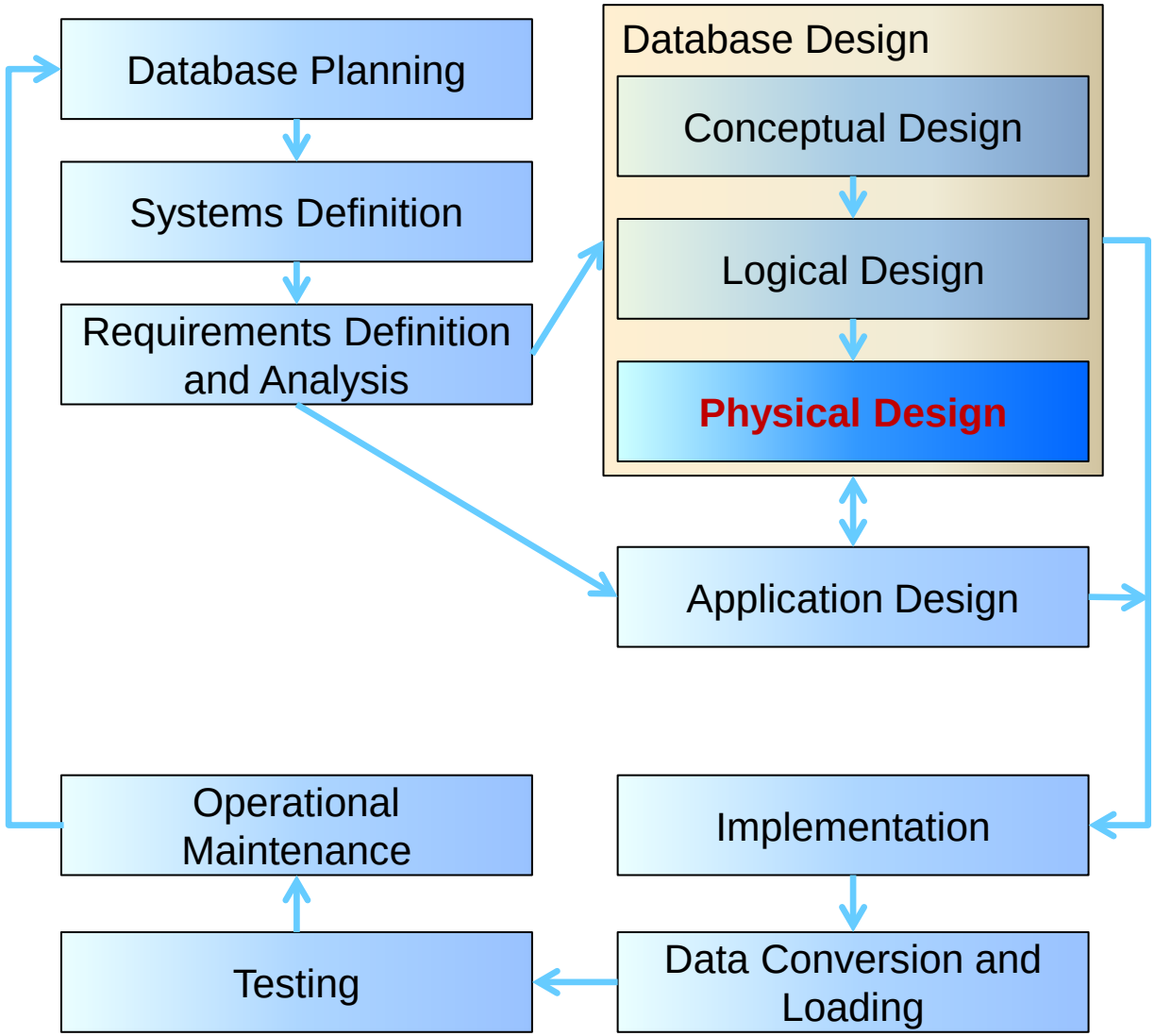
Database Development Lifecycle



- Builds on the conceptual design
- Designing now for a relational database
- Includes columns and keys
- Independent of a specific vendor and other physical considerations

© The University of Melbourne 2018

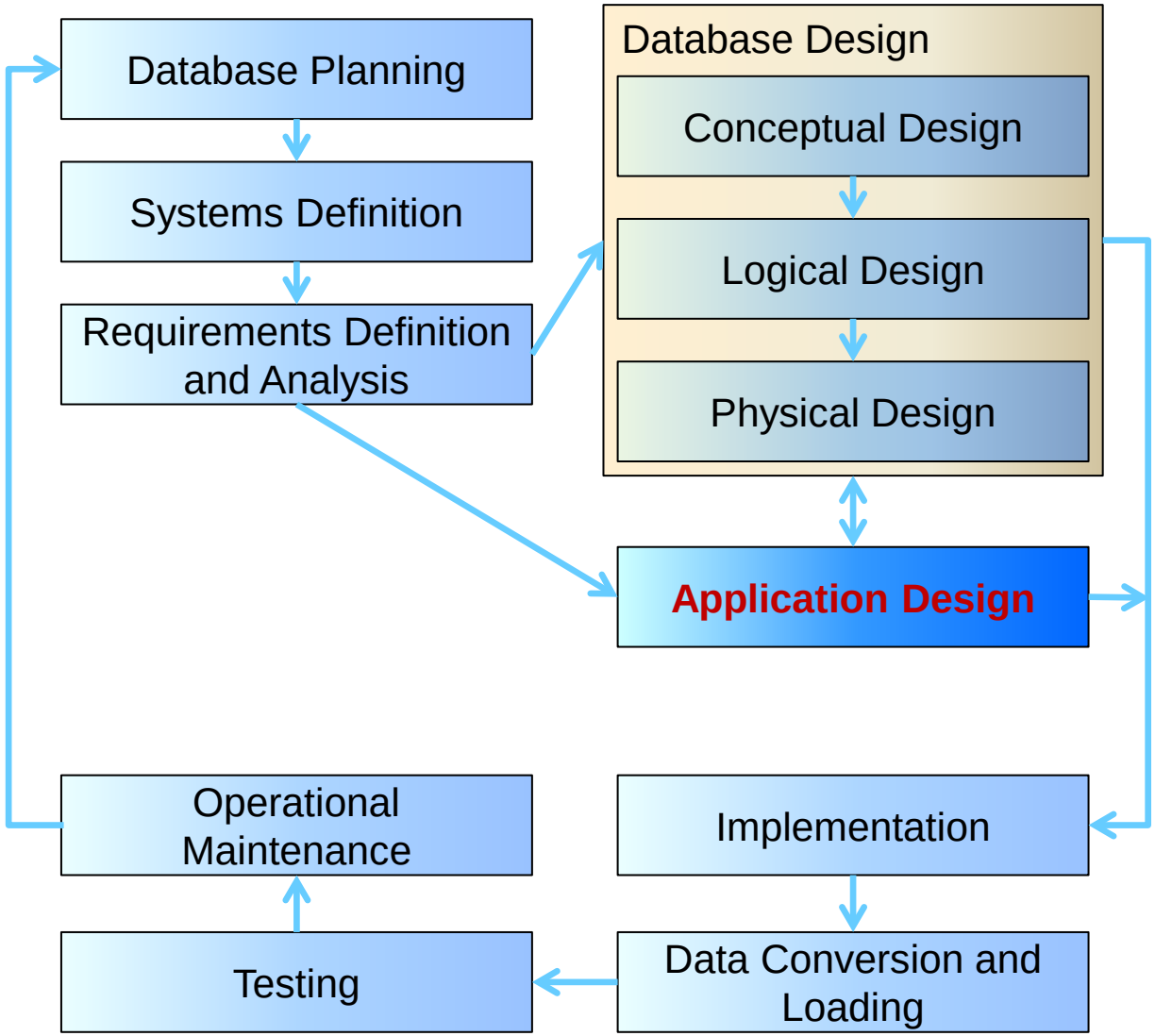
Database Development Lifecycle



- Implements the logical design for a specific DBMS.
- Describes:
  - Base tables
  - Data types
  - Indexes
  - Integrity constraints
  - File organisation
  - Security measures
- We will cover some aspects of physical design

© The University of Melbourne 2018

Database Development Lifecycle

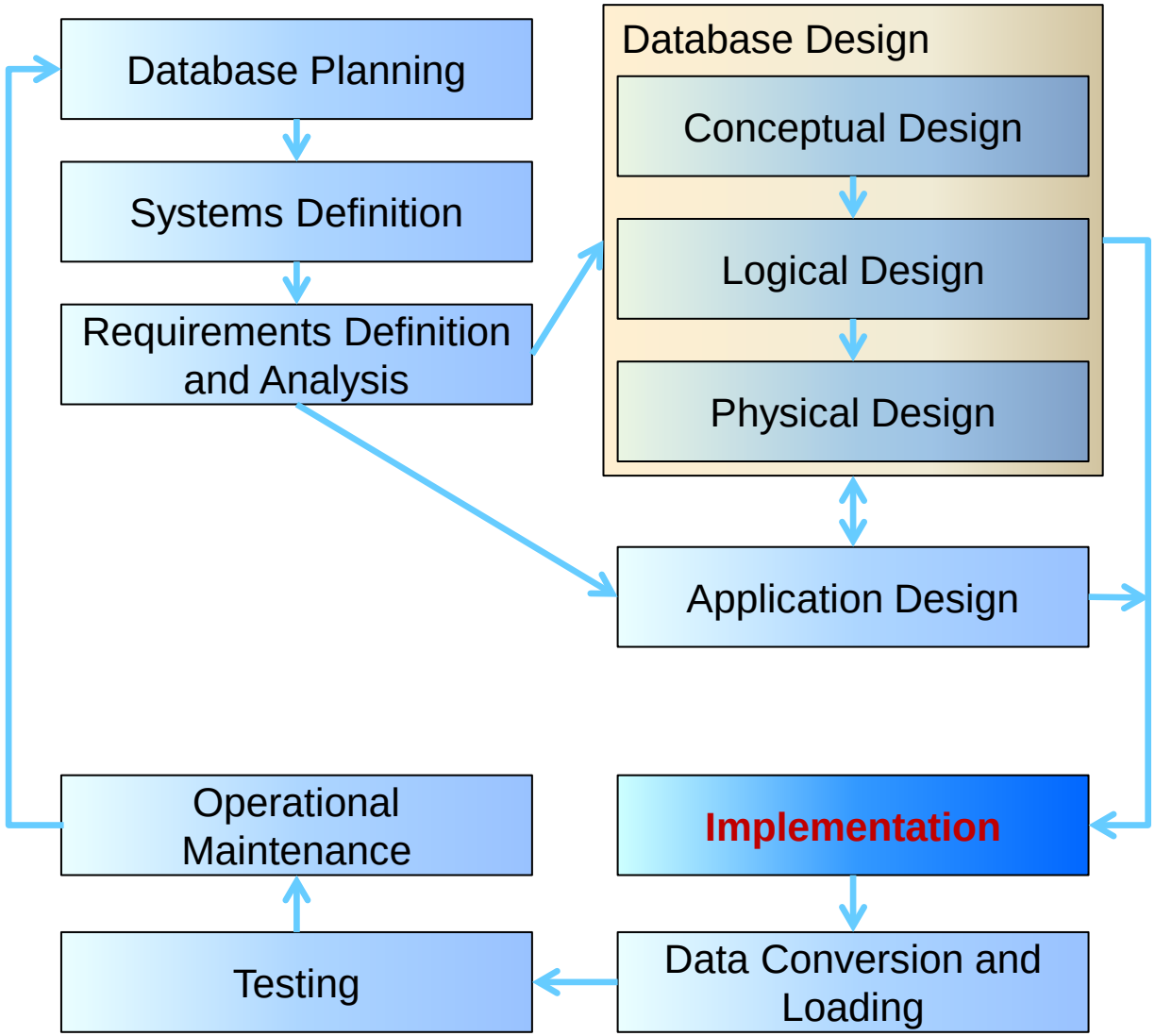


- Done in conjunction with database design
- Design of the interface and application programs that use and process the database

- Mostly outside scope of the course, but discussed in week 7

© The University of Melbourne 2018

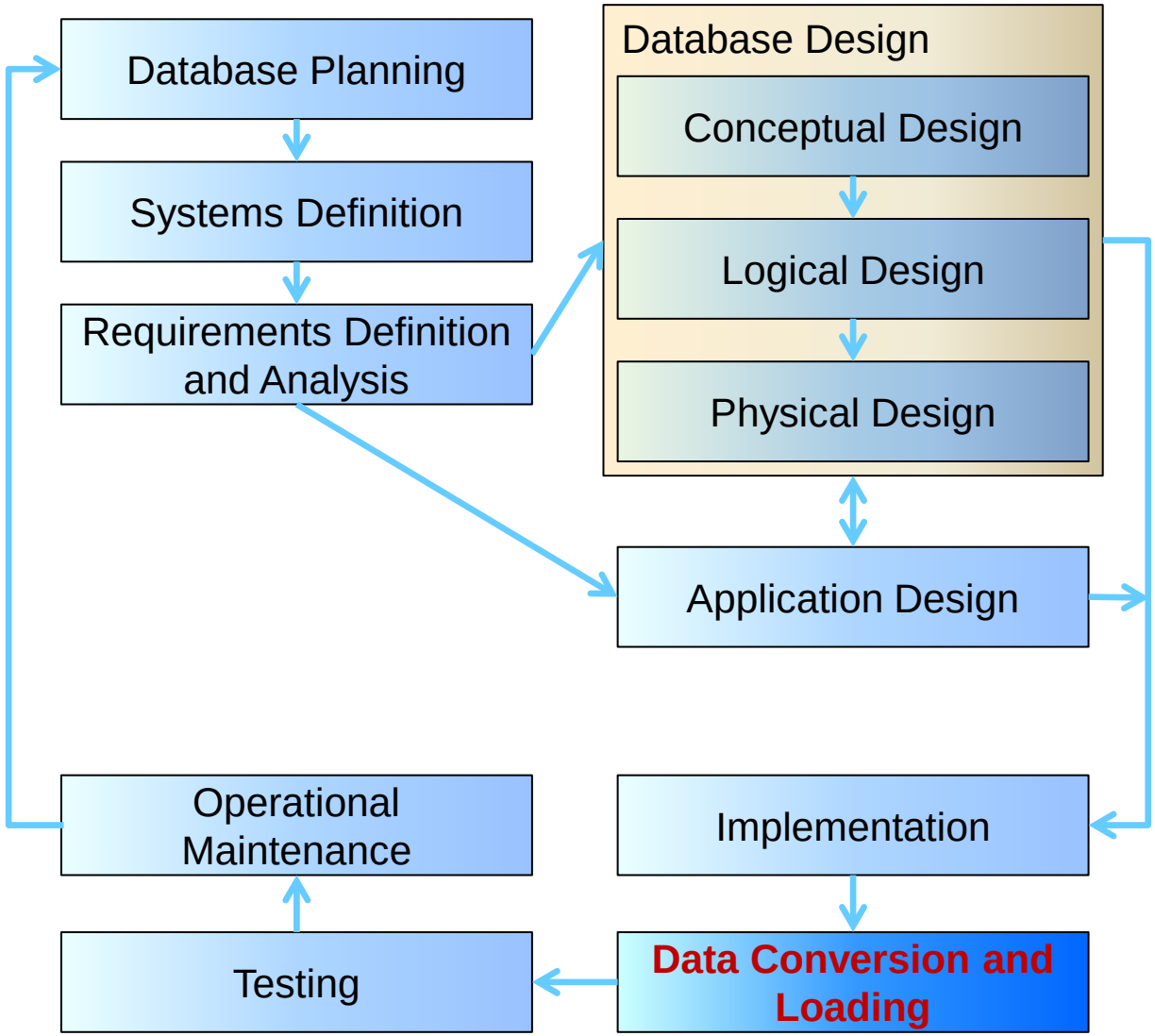
Database Development Lifecycle



- Implementation of the design as a working database

© The University of Melbourne 2018

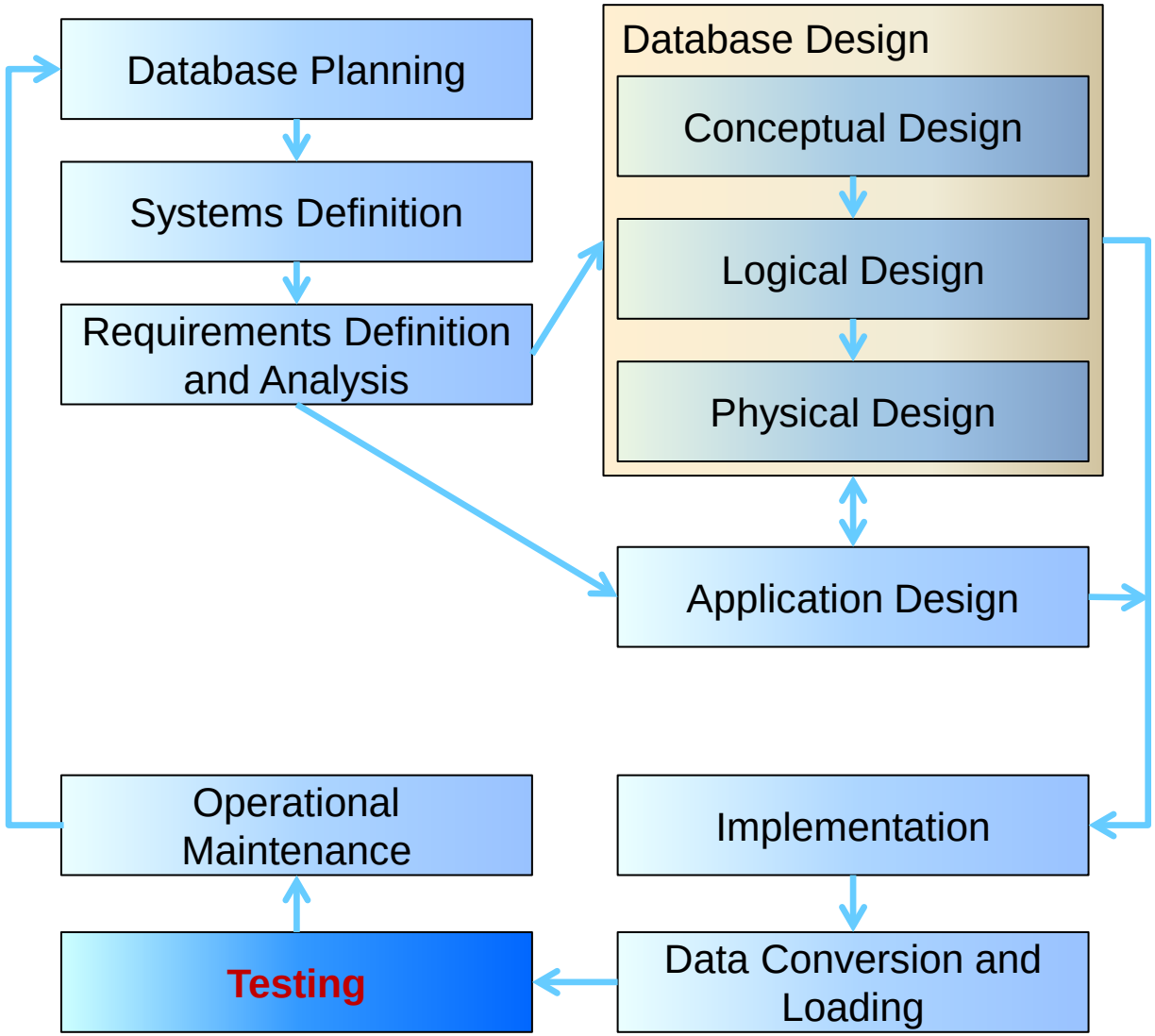
Database Development Lifecycle



- Transfer existing data into the database
- Conversion from old systems
- Non trivial task

© The University of Melbourne 2018

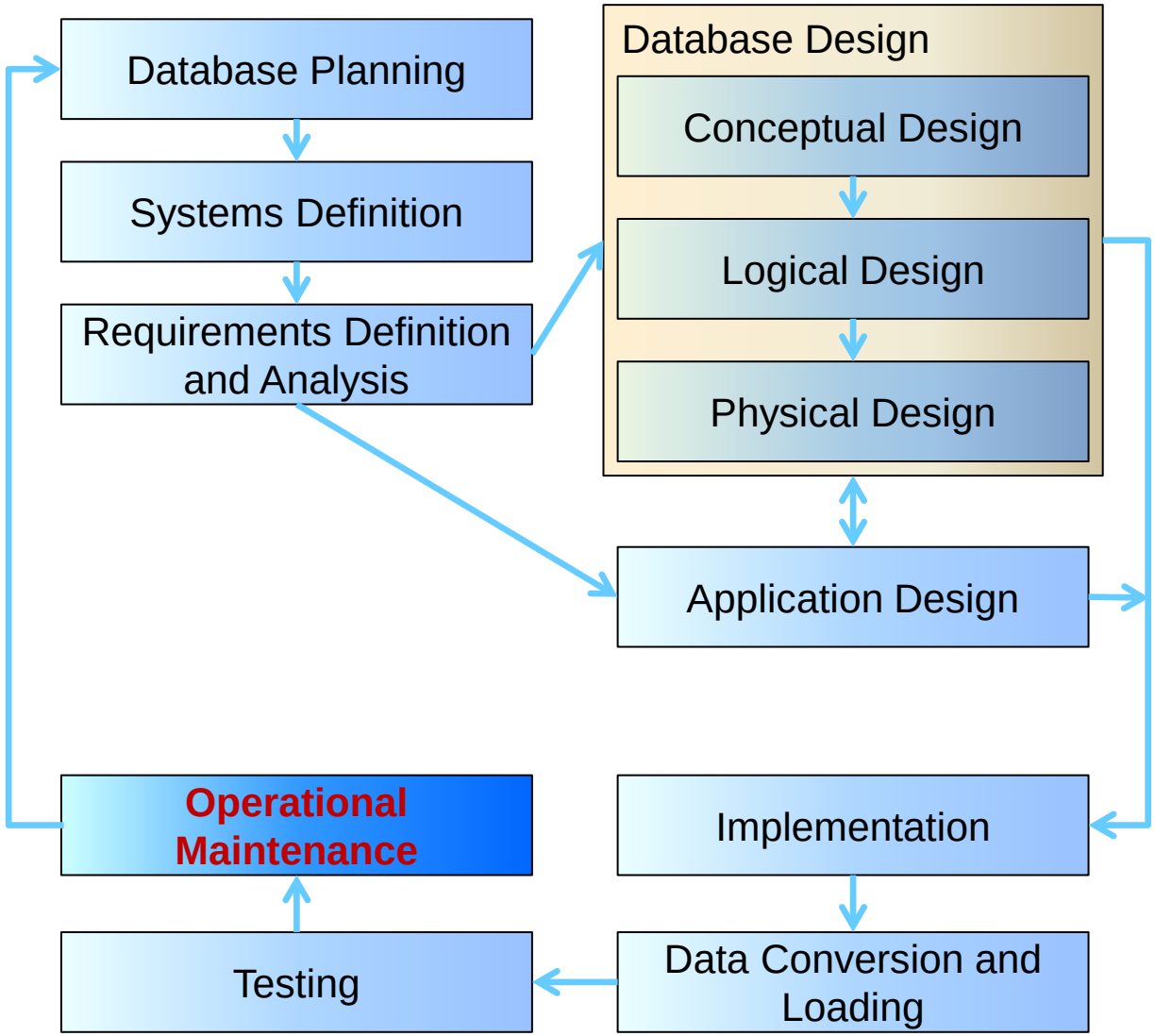
Database Development Lifecycle



- Running the database to find errors in the design / setup
- Other issues also
  - Performance
  - Robustness
  - Recoverability
  - Adaptability
- Mostly outside scope of the course

© The University of Melbourne 2018

Database Development Lifecycle

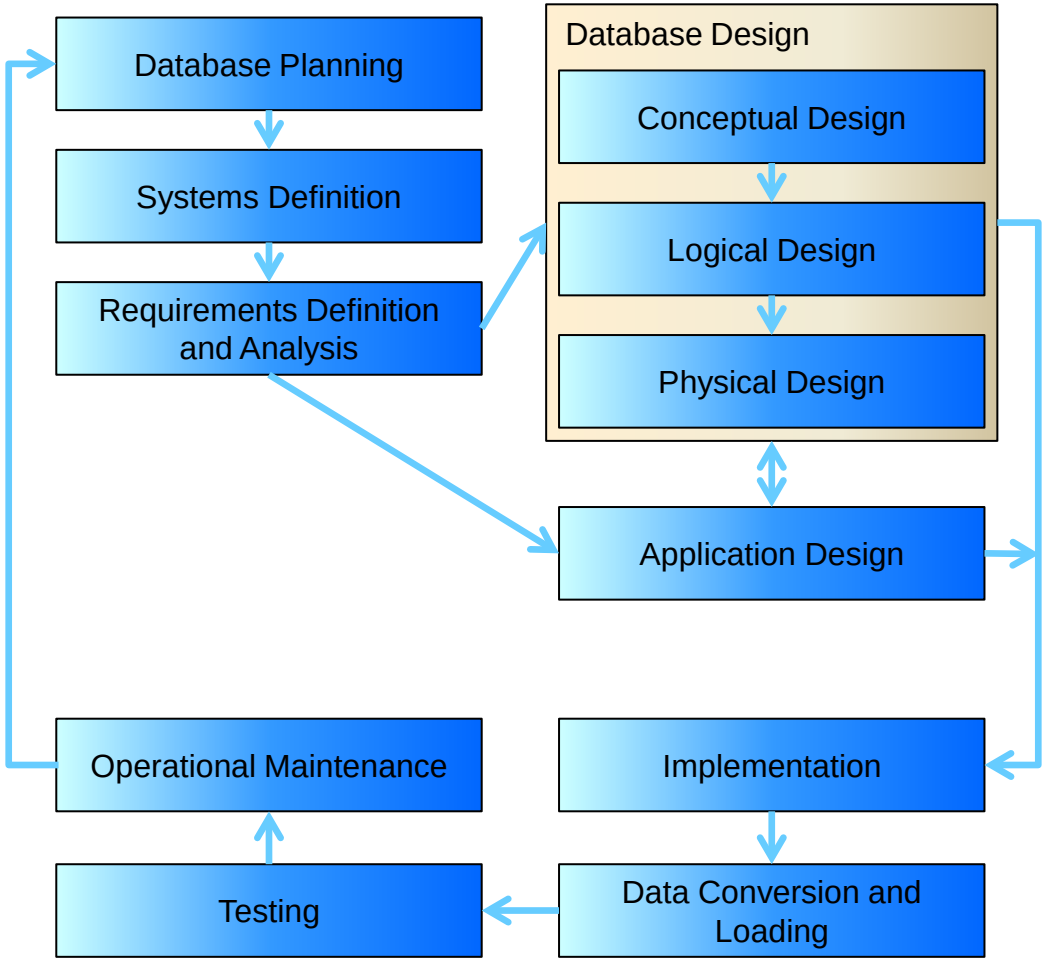


- The process of monitoring and maintaining the database following its commissioning
- Monitoring and improving performance
- Handling changes to requirements

- We will touch on some of these topics later in the semester, especially in week 9

© The University of Melbourne 2018

Summary of database lifecycle



Now we'll work through one example …

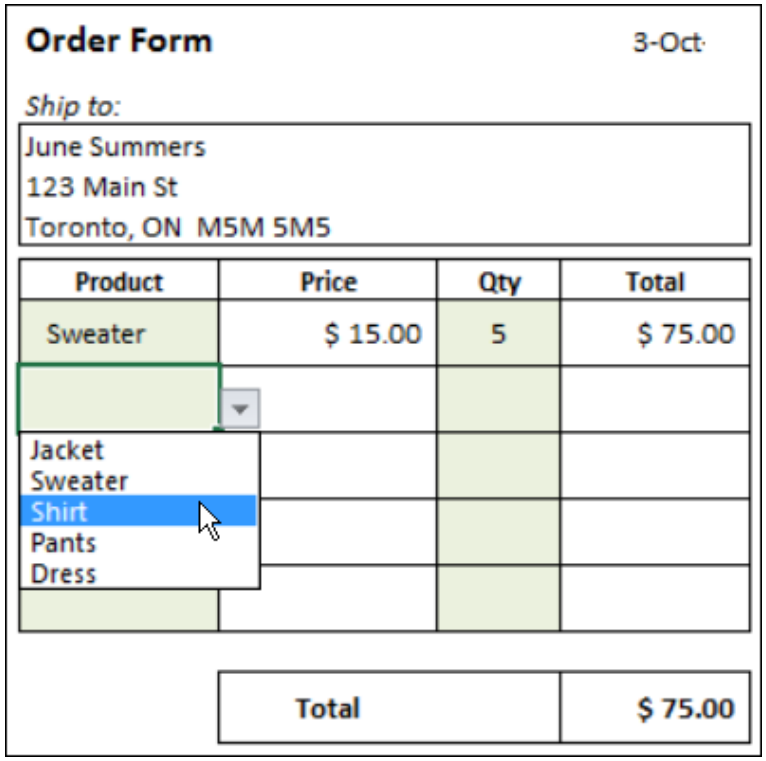
© The University of Melbourne 2018

Case Study: design the db

Data Modelling

Case for this lecture: Orders system

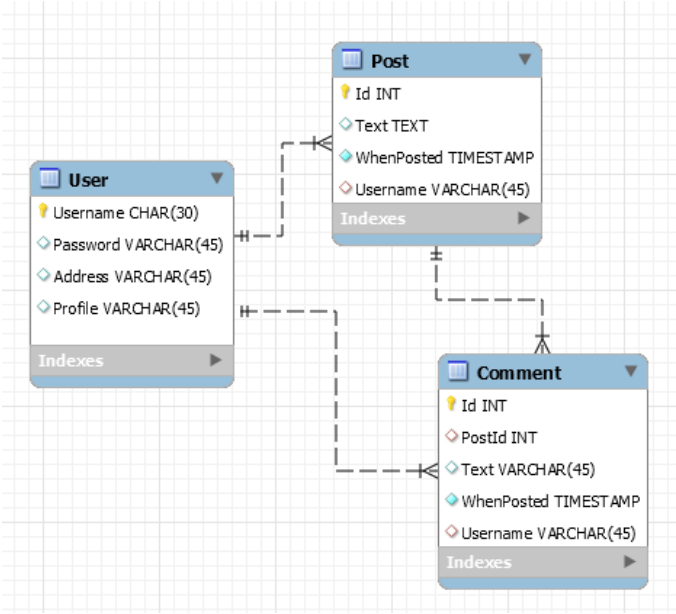
- Our company sells many products. About each product we record its id, name, and price.
- We have many customers. About each customer we record their customer id, name, and address.
- Customers place orders for products. Each order is placed by one customer on a particular date. Over time a customer may place several orders, though some may register but not place any orders.
- Each order must contain at least one order-item, but may contain several. Each order-item records a quantity ordered of one product.



© The University of Melbourne 2018

Recall from week 1: Data Modelling

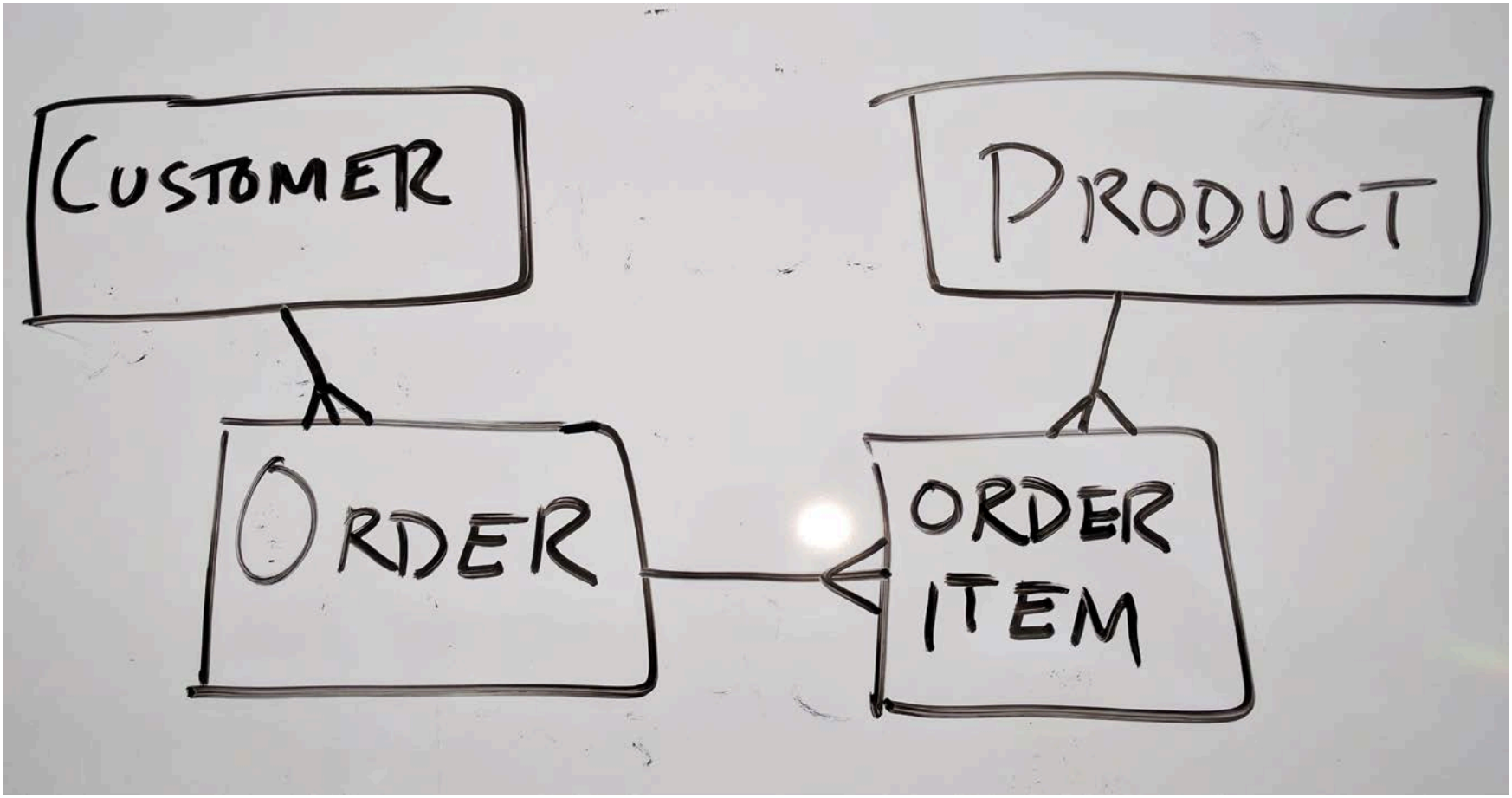
1. What are the entities that need to be tracked?
2. What information will be recorded about each entity?
3. What are the relationships between entities?
4. What are the cardinalities of relationships?



© The University of Melbourne 2018

Conceptual Data Model

(we will create this together during the lecture)

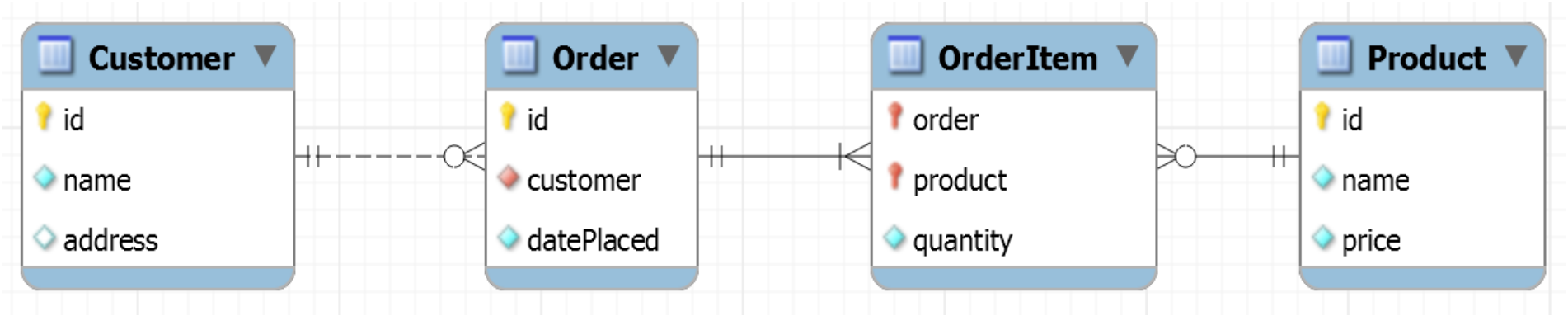


© The University of Melbourne 2018

Logical Data Model

(we will create this together during the lecture)

New question:
What is the *primary key* of each table?

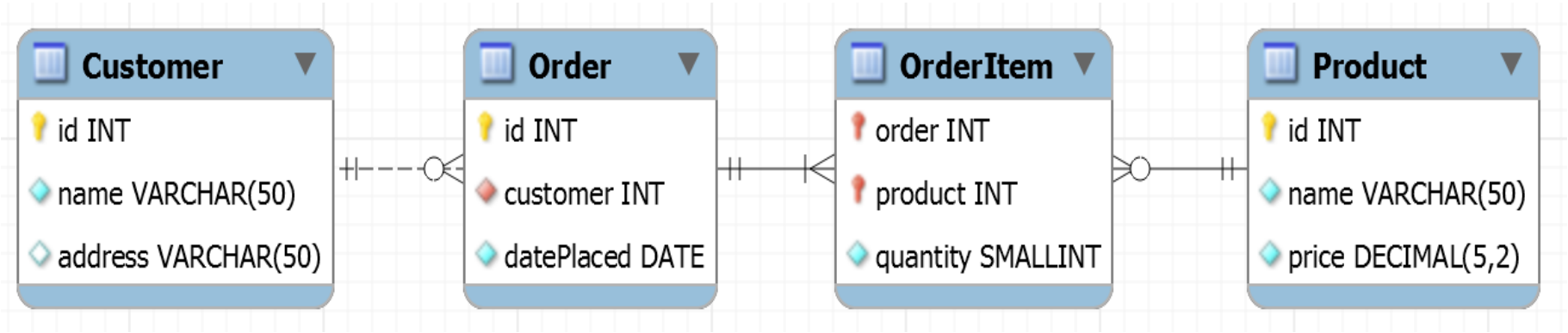


© The University of Melbourne 2018

Physical Model

(we will create this together during the lecture)

New question:
What is the *data type* of each column?



© The University of Melbourne 2018

Case Study: implement the db

SQL

Create tables

Can be done in several ways:

1. manual DDL commands
2. SQL "setup script" (right)
3. forward engineer from MySQL Workbench and similar tools

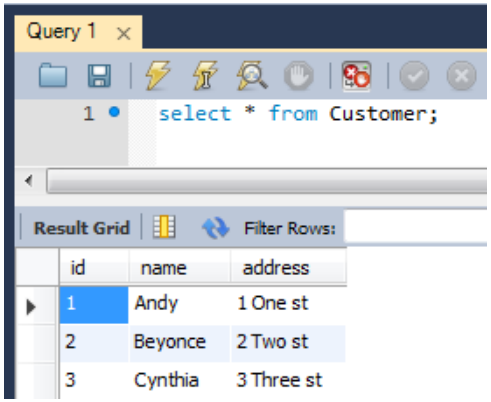
```sql
CREATE TABLE IF NOT EXISTS `Customer` (
  `id` INT NOT NULL,
  `name` VARCHAR(50) NOT NULL,
  `address` VARCHAR(50) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `Order` (
  `id` INT NOT NULL,
  `customer` INT NOT NULL,
  `datePlaced` DATE NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_Order_Customer_idx` (`customer` ASC),
  CONSTRAINT `fk_Order_Customer`
    FOREIGN KEY (`customer`)
    REFERENCES `Customer` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `Product` (
  `id` INT NOT NULL,
  `name` VARCHAR(50) NOT NULL,
  `price` DECIMAL(5,2) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `OrderItem` (
  `order` INT NOT NULL,
  `product` INT NOT NULL,
  `quantity` SMALLINT NOT NULL,
  PRIMARY KEY (`order`, `product`),
  INDEX `fk_OrderItem_Product1_idx` (`product` ASC),
  CONSTRAINT `fk_OrderItem_Order1`
    FOREIGN KEY (`order`)
    REFERENCES `Order` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_OrderItem_Product1`
    FOREIGN KEY (`product`)
    REFERENCES `Product` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```


© The University of Melbourne 2018

Insert data

Can be done in several ways:

1. manual Insert commands
2. SQL script file (right)
3. Insert in Workbench model
4. via application software




```sql
/* INSERT DATA */

insert into Customer values (1,'Andy','1 One st');
insert into Customer values (2,'Beyonce','2 Two st');
insert into Customer values (3,'Cynthia','3 Three st');

insert into Product values (1,'apple',1.11);
insert into Product values (2,'bread',2.22);
insert into Product values (3,'cabbage',3.33);
insert into Product values (4,'dates',4.44);

insert into `Order` values (1,1,'2017-6-1');
insert into `Order` values (2,2,'2017-6-2');
insert into `Order` values (3,3,'2017-6-3');
insert into `Order` values (4,1,'2017-6-4');
insert into `Order` values (5,2,'2017-6-5');

insert into OrderItem values (1,1,1);
insert into OrderItem values (1,2,2);
insert into OrderItem values (1,3,3);
insert into OrderItem values (2,2,4);
insert into OrderItem values (2,4,5);
insert into OrderItem values (3,1,6);
insert into OrderItem values (3,2,7);
insert into OrderItem values (3,3,8);
insert into OrderItem values (3,4,9);
insert into OrderItem values (4,4,10);
insert into OrderItem values (5,3,9);
```


© The University of Melbourne 2018

Query the data

IntroToMySql*

Don't Limit

```sql
SELECT name, price FROM product
WHERE price > 3;

SELECT * FROM customer
ORDER BY name DESC;

SELECT AVG(price) FROM PRODUCT;

SELECT * FROM orderitem JOIN product
ON orderitem.product = product.id;

SELECT product, COUNT(*), SUM(price)
FROM orderitem JOIN product
ON orderitem.product = product.id
GROUP BY product;
```


© The University of Melbourne 2018

Change the data

demoQueries*

Don't Limit

```sql
UPDATE product
SET Price = 3.50
WHERE name = 'cabbage'; /* change price of cabbage */

INSERT INTO `order`
values (6, 1, '2017-7-13');  /* add a new order */

INSERT INTO orderitem
values (6, 2, 20), (6, 3, 30); /* add items to our new order */

DELETE from product
WHERE name = 'dates';   /* delete one product */
```


© The University of Melbourne 2018

Next 3 weeks

- More detailed understanding of database design
  - Conceptual design
  - Logical design
  - Physical design
- More detailed understanding of SQL
  - Operations on a single table
  - Joining multiple tables

© The University of Melbourne 2018

Dr Greg Wadley

THE UNIVERSITY OF MELBOURNE

INFO90002 Database Systems & Information Modelling

Week 03

Data Modelling (1)

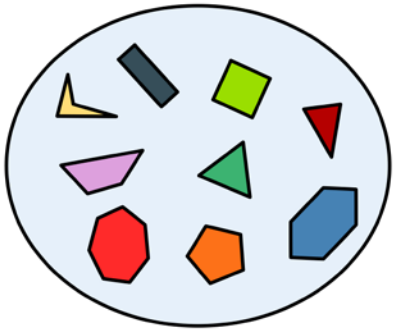
Today's session

- Data modelling
  - ER modelling conventions
  - Identifying entities and business rules
  - one-to-many (1-M) relationships
- SQL
  - Overview and history
  - Create tables
  - Insert data into tables
  - Read data from tables

© University of Melbourne 2018

Databases and ER models

- A database can be thought of as a representation of
  - a collection of entity sets, and
  - relationships between the entities
- An entity is an object or abstract concept or event which can be distinguished from other entities
  - example: product, order, sale, person, movie, tweet
- Entities have attributes that describe the entity and distinguish it from other entities in the same entity set
  - example attributes: EmployeeName, Address



- (reminder – what are “sets”?
  - union, intersection, Cartesian product)

© University of Melbourne 2018

Valid and invalid entities

- An Entity
  - Will have many instances in the database
  - Has several attributes
  - Is necessary for the system to work
- Examples
  - Person: EMPLOYEE, STUDENT, PATIENT
  - Place: STORE, WAREHOUSE, STATE, CITY
  - Object: PRODUCT, MACHINE, BUILDING, VEHICLE
  - Event: SALE, REGISTRATION, BROADCAST
  - Abstract: ACCOUNT, UNI SUBJECT, ROLE
- Entities do not usually include:
  - An output of the system (i.e. a report)
  - The system itself
  - The company that owns the system

© University of Melbourne 2018

Naming Conventions

- Entities
  - singular nouns
    - Employee, Customer, Sale (without or without capital letter)
- Attributes
  - usually a noun
    - itemColour, quantitySold, id
- Relationships
  - verbs or verb phrases
    - has, wants, manages, performs work for
- Use names meaningful to the domain
  - try not to abbreviate names
    - except num or nbr for number, ID for identifier
  - conventions on UPPER and lower case

© University of Melbourne 2018

Modelling a Single Entity

- By searching for nouns in the case we can identify entities

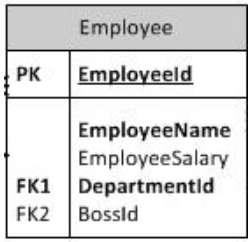
  (for example – Customer)

- What things would we need to record about the Customer
  - these become the customer's Attributes

- How can we identify individual Customers?
  - by name?
  - by address?
- Now we can draw it as an entity in the ER diagram

© University of Melbourne 2018

Mapping ER diagram to database tables



EmployeeID	EmployeeName	EmployeeSalary	DepartmentID	BossID
1	Alice	75000.00	1	0
2	Ned	45000.00	11	1
3	Andrew	25000.00	11	2
4	Clare	22000.00	11	2
5	Todd	38000.00	2	1
6	Nancy	22000.00	2	5
7	Brier	43000.00	9	1
8	Sarah	56000.00	9	7
9	Sophie	35000.00	10	1
10	Sanjay	15000.00	6	3

- Entity set
  - Often corresponds to a table in the database
- Entity instance
  - Often corresponds to a row in a table
- Attribute
  - Often corresponds to a column in a table
- Relationship set (link between entity sets)
  - Often corresponds to a Foreign Key in a table
- Relationship instance (link between entity instances)
  - Foreign Key value = Primary Key value

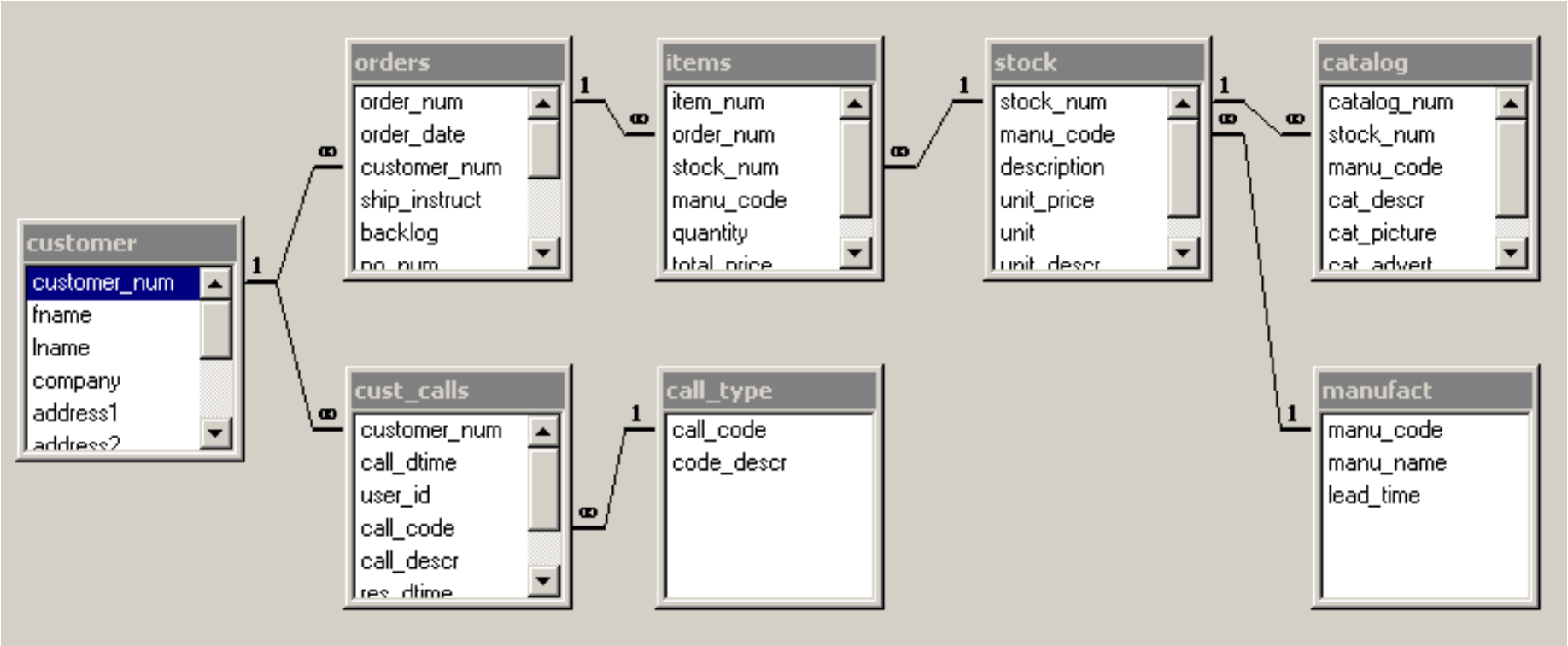
© University of Melbourne 2018

How to draw ER diagrams?

- You can use paper or the software of your choice
- *Visio* is a good (commercial) tool, available in labs
- https://www.lucidchart.com/ and other web apps
- Assignment 1 model should be made in *MySQL Workbench*
- In the exam you'll need to model *on paper*
- Diagrams in lecture slides are made in Workbench and Visio
- My suggestion is:
  - Use pen-and-paper or whiteboard for early Conceptual modelling
  - Use paper, Visio or Workbench for subsequent Logical modelling
  - Use Workbench for the final Physical model

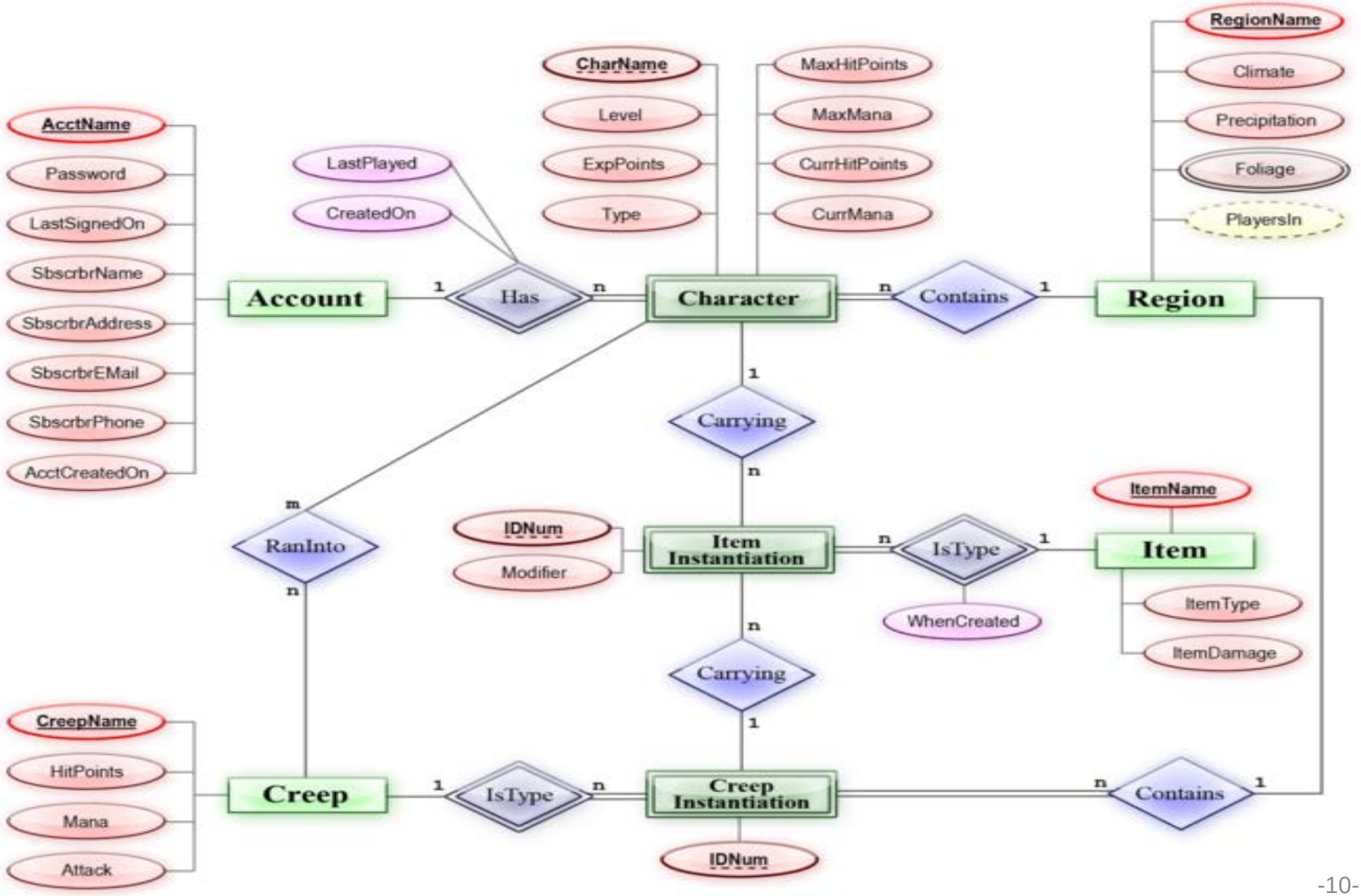
© University of Melbourne 2018

Variation in ER diagram standards

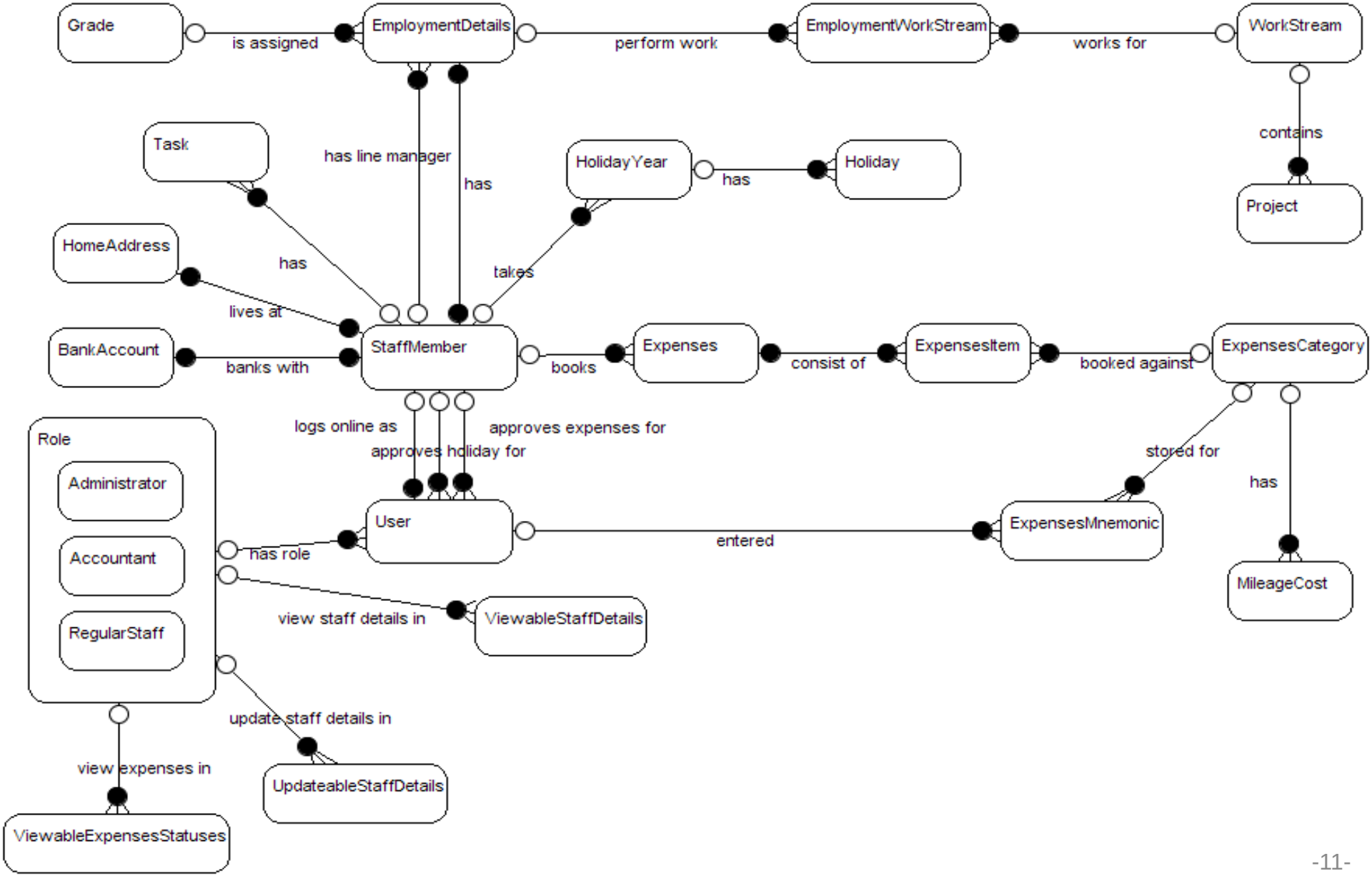


© University of Melbourne 2018

Variation in ER diagram standards



Variation in ER diagram standards

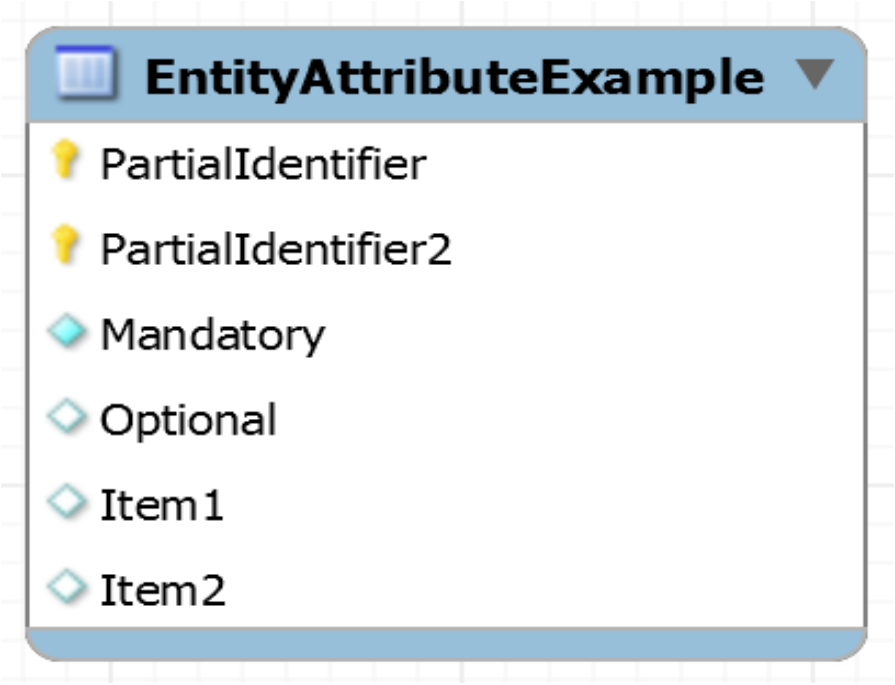


Representing entities and attributes

- Entity

Entity1	
PK	Identifier
	Ent1Attribute1 Ent1Attribute2

- Attributes



- Key (or Identifier)
  - Fully identifies an instance
- Partial Key
  - Partially identifies an instance
- Attributes
  - Mandatory
  - Optional
  - Derived
    - [YearsEmployed]
  - Multivalued
    - {Skill}
  - Composite
    - Name (First, Middle, Last)

© University of Melbourne 2018

Conceptual design for single entity

(drawn using Visio software)

Customer1	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType** **CustAddress(Line1, Line 2, Suburb, Postcode, Country)**

- underline = primary key
- bold = not null
- () = composite attribute

© University of Melbourne 2018

Convert to *logical* design

Customer1	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType** **CustAddress(Line1, Line 2, Suburb, Postcode, Country)**

Customer1	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType** **CustAddLine1** **CustAddLine2** **CustSuburb** **CustPostcode** **CustCountry**

- Composite attributes become individual attributes
- Multi-valued attributes become a new table
- Resolve many-many relationships via a new table
- Add foreign keys at crows foot end of relationships

© University of Melbourne 2018

Convert to *physical* design

- Determine data types for each attribute
- Key, nullable

Customer1		
PK	CustomerID	SMALLINT
	CustFirstName	VARCHAR(100)
	CustMiddleName	VARCHAR(100)
	CustLastName	**VARCHAR(100)**
	BusinessName	VARCHAR(100)
	CustType	**CHAR(1)**
	CustAddLine1	**VARCHAR(100)**
	CustAddLine2	**VARCHAR(100)**
	CustSuburb	**VARCHAR(60)**
	CustPostcode	**CHAR(6)**
	CustCountry	**VARCHAR(60)**

Physical Name	Data Type	Req'd	PK	Notes
CustomerID	SMALLINT	☑	☑	CustomerID identifies Customer1
CustFirstName	VARCHAR(100)	☐	☐	CustFirstName is of Customer1
CustMiddleName	VARCHAR(100)	☐	☐	CustMiddleName is of Customer1
CustLastName	VARCHAR(100)	☑	☐	CustLastName is of Customer1
BusinessName	VARCHAR(100)	☐	☐	BusinessName is of Customer1
CustType	CHAR(1)	☑	☐	NOTE: This will be implemented as an ENUM type in MySQL with t
CustAddLine1	VARCHAR(100)	☑	☐	CustAddLine1 is of Customer1
CustAddLine2	VARCHAR(100)	☑	☐	CustAddLine2 is of Customer1
CustSuburb	VARCHAR(60)	☑	☐	CustSuburb is of Customer1
CustPostcode	CHAR(6)	☑	☐	CustPostcode is of Customer1
CustCountry	VARCHAR(60)	☑	☐	CustCountry is of Customer1

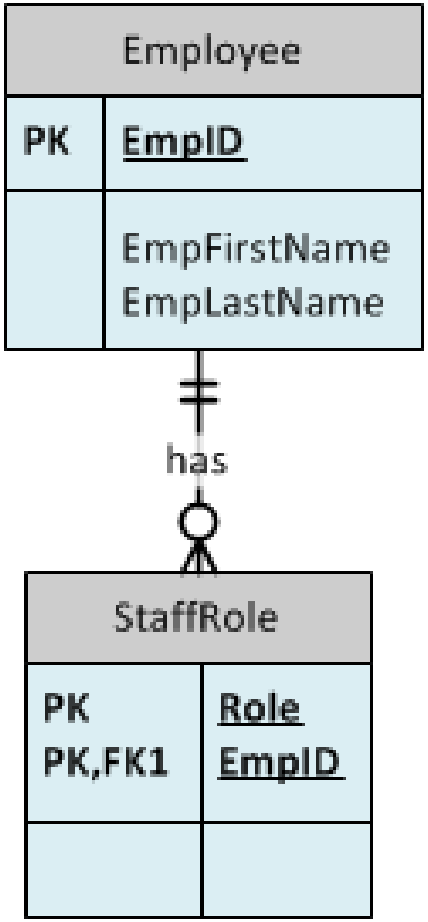
© University of Melbourne 2018

Dealing with multi-valued attributes

Conceptual Design

Employee	
PK	EmpID
	EmpFirstName EmpLastName {Role}

Logical Design



StaffRole is an example of a *weak entity*

© University of Melbourne 2018

Business Rules and Relationships

- Business rules are assertions that constrain entities
- Can impact structure and behaviour of the database
- Business rules can be assertions about attributes
  - "Quantity bought must be between 1 and 200." (assertion)
    - quantity bought (attribute)
- Or business rules can be assertions about entities
  - "A customer sets up at least one account." (assertion)
    - customer (entity)
    - account (entity)
- The latter kind of business rules are represented in our data models as *relationships between entities*.

© University of Melbourne 2018

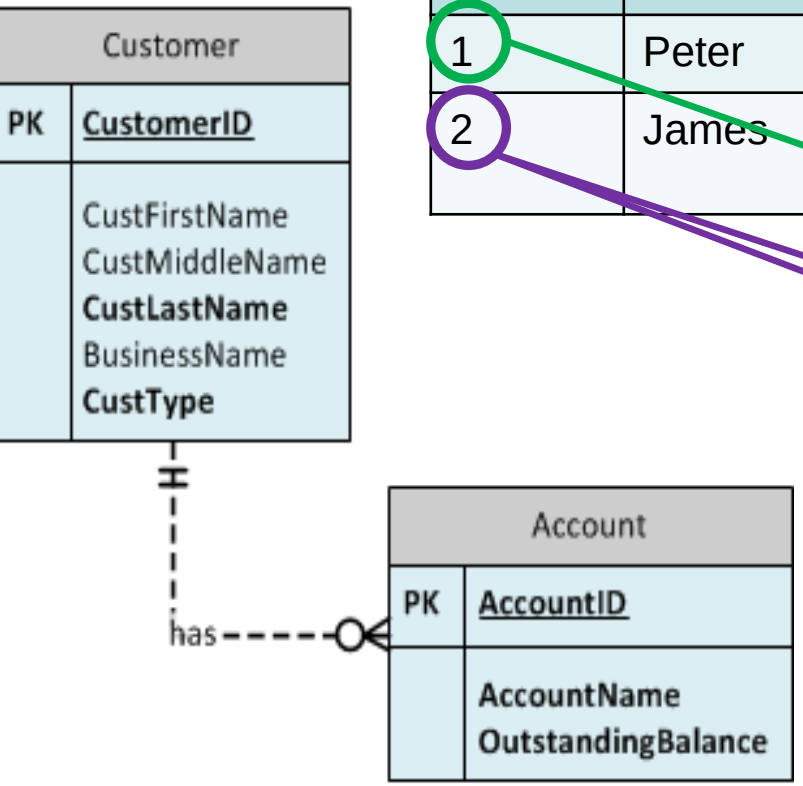
Keys

- Keys or Identifiers are used to identify individual entity instances
  - Primary Key
    - (set of) columns, the values in which uniquely identify each instance
    - no column can be removed from the key without losing uniqueness
  - Candidate Key
    - the set of possible primary keys (choose one to be the PK)
  - Surrogate Key
    - system-assigned serial number (used if natural PK is unavailable or unsuitable)
  - Composite Key
    - a key which is made up of more than one attribute
      - e.g. for the entity “airline flight” we might use the composite key
        - » FlightNumber + FlightDate
  - Foreign Key
    - the key used to link to a primary key in another table
      - helps us to join tables in a Select statement
- Primary Keys are
  - unique
  - never null
  - do not change their value

© University of Melbourne 2018

Two entities with 1-M relationship

- Example: "A customer can set up several accounts."
  - The tables get linked through a foreign key



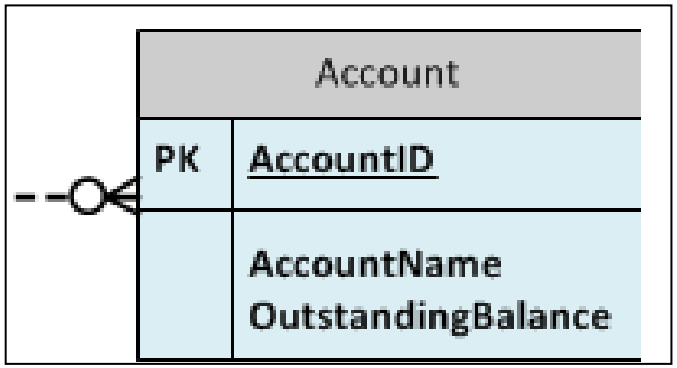
CustID	Customer FirstName	CustMiddle Name	CustLast Name	Business Name	CustType
1	Peter		Smith		Personal
2	James		Jones	JJ Enterprises	Company

AccountID	AccountName	Outstanding Balance	CustID
01	Peter Smith	245.25	1
05	JJ Ent.	552.39	2
06	JJ Ent. Mgr	10.25	2

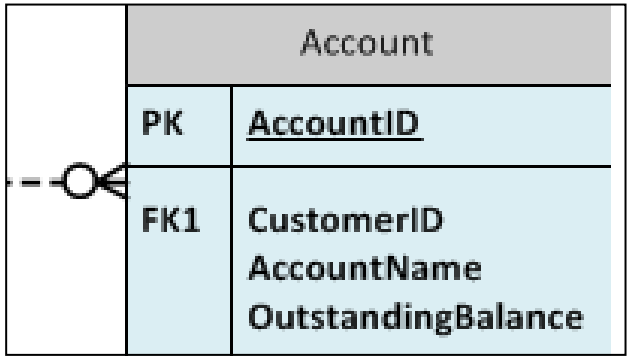
© University of Melbourne 2018

Conceptual to logical design – with FK

- Conceptual Design



- Logical Design

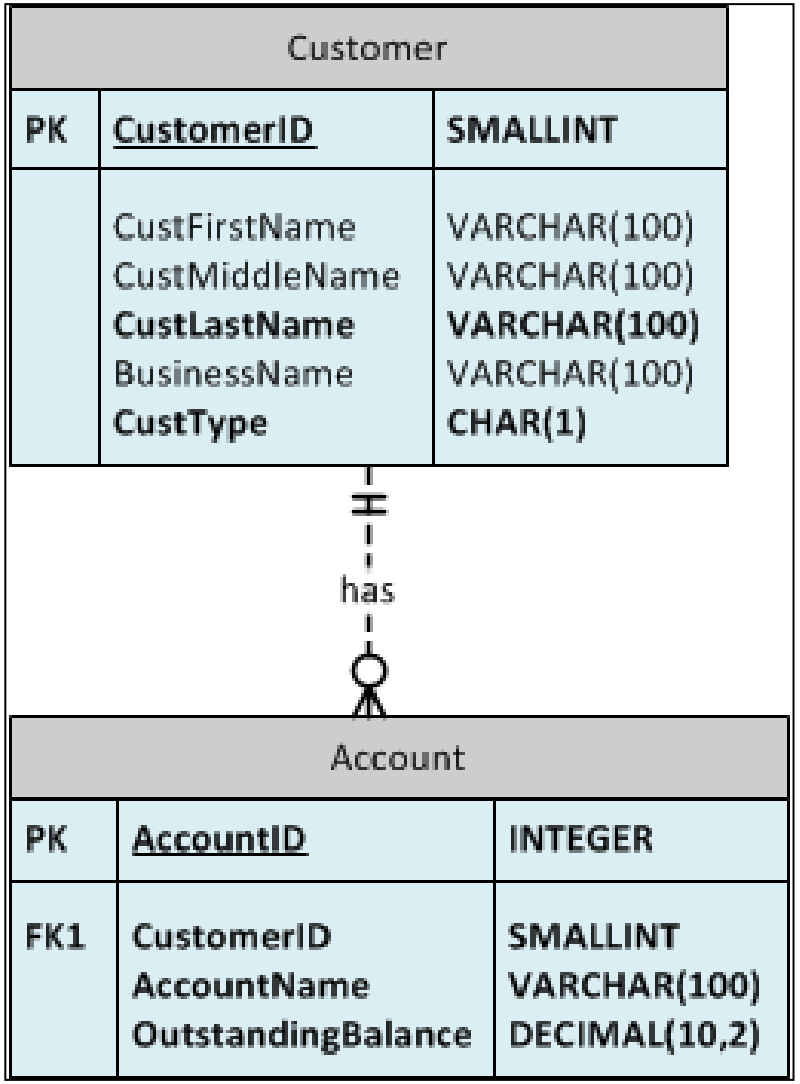


– Add foreign keys at crows feet end of relationships
  - FK1 – CustomerID
    – This is the link to the customer table
    – Every CustomerID in Account must be present in Customer
      » Referential integrity

© University of Melbourne 2018

Physical design - with FK

- Attribute data types



FK must have the same data type as the PK it refers to.

© University of Melbourne 2018

Now we can create the tables

```
CREATE TABLE Customer (
  CustomerID        smallint                    auto_increment,
  CustFirstName     varchar(100),
  CustMiddleName    varchar(100),
  CustLastName      varchar(100)                NOT NULL,
  BusinessName      varchar(200),
  CustType          enum('Personal','Company')  NOT NULL,
  PRIMARY KEY  (CustomerID)
) ENGINE=InnoDB;
```

```
CREATE TABLE Account (
  AccountID             smallint        auto_increment,
  AccountName           varchar(100)    NOT NULL,
  OutstandingBalance    DECIMAL(10,2)   NOT NULL,
  CustomerID            smallint        NOT NULL,
  PRIMARY KEY  (AccountID),
  FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Referential Actions how foreign keys guarantee referential integrity.

© University of Melbourne 2018

Insert - with FK

Current Database

CustID	CustomerFirstName	CustMiddle Name	CustLastName	BusinessName	CustType
1	Peter		Smith		Personal
2	James		Jones	JJ Enterprises	Company

AccountID	AccountName	OutstandingBalance	CustID

Insert a row…

```
INSERT INTO ACCOUNT VALUES (DEFAULT, 'My New Account', 0, 5);
```

What happens?

INSERT INTO ACCOUNT VALUES (DEFAULT, ...	Error Code: 1452. Cannot add or update a child row: a f

Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`db_seanbm/account`, CONSTRAINT `account_ibfk_1` FOREIGN KEY (`CustomerID`) REFERENCES `customer` (`CustomerID`))

© University of Melbourne 2018

Insert - with FK

- Run the Inserts…

```
INSERT INTO ACCOUNT VALUES (DEFAULT, 'Peter Smith', 245.25, 1);
INSERT INTO ACCOUNT VALUES (DEFAULT, 'JJ ENt.', 552.39, 2);
INSERT INTO ACCOUNT VALUES (DEFAULT, 'JJ ENt. Mgr', 10.25, 2);
```

CustID	CustomerFirstName	CustMiddle Name	CustLastName	BusinessName	CustType
1	Peter		Smith		Personal
2	James		Jones	JJ Enterprises	Company
3	Akin		Smithies	Bay Wart	Company

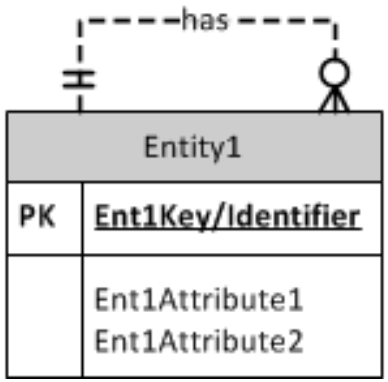
AccountID	AccountName	OutstandingBalance	CustID
01	Peter Smith	245.25	1
02	JJ Ent.	552.39	2
03	JJ Ent. Mgr	10.25	2

© University of Melbourne 2018

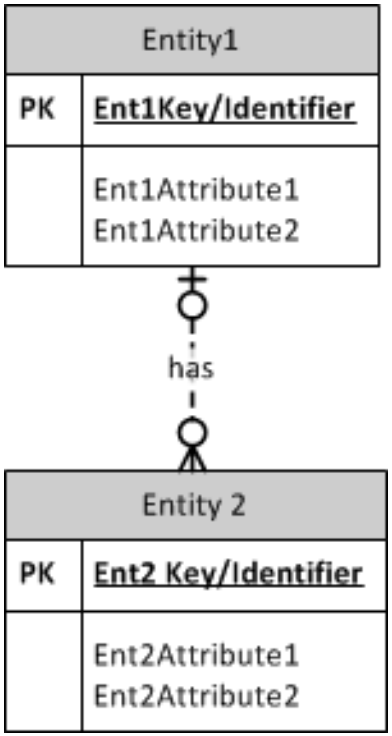
Relationship Degree

How many entities take part in the relationship?

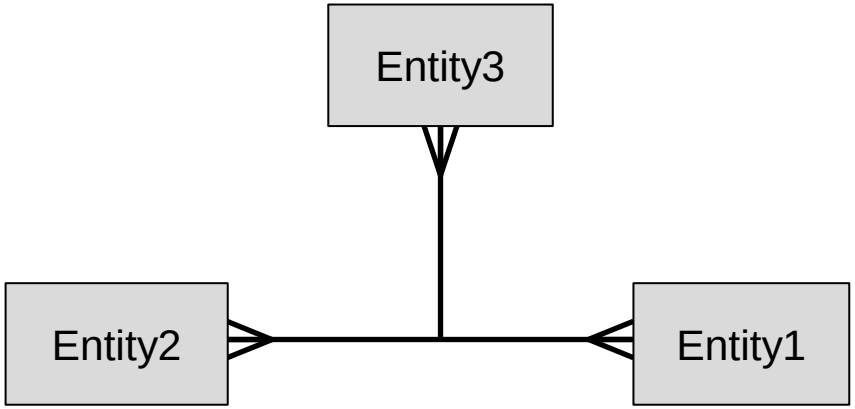
Unary (1)



Binary (2)



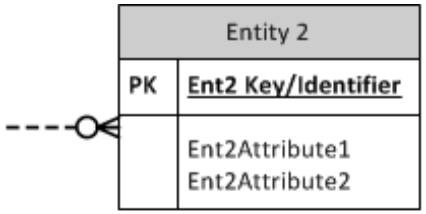
Ternary (3)



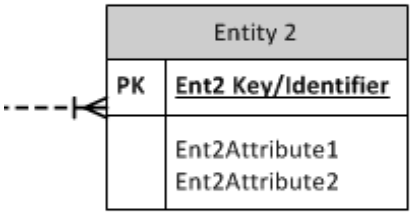
© University of Melbourne 2018

Relationship Cardinality

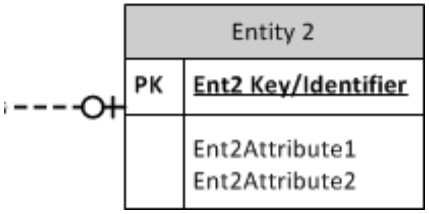
- One to One
  - Each entity in one set is related to 0 or 1 in the other.
- One to Many
  - Each entity in one set is related to many in the other.
- Many to Many
  - Each entity in either set can be related to many in the other set
  - These require an extra step to implement in a relational database.



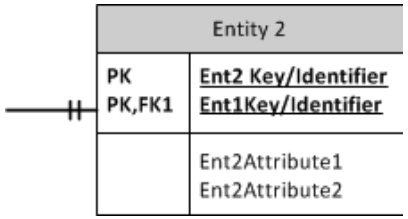
Optional Many



Mandatory Many



Optional One

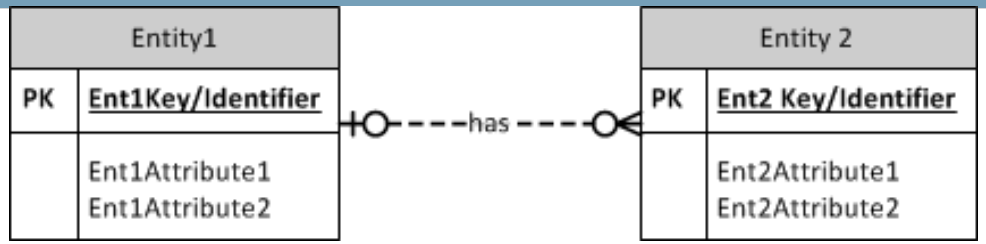


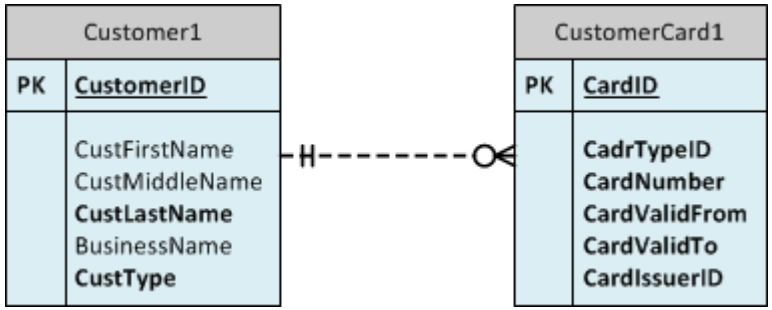
Mandatory One

© University of Melbourne 2018

Strong and Weak entities

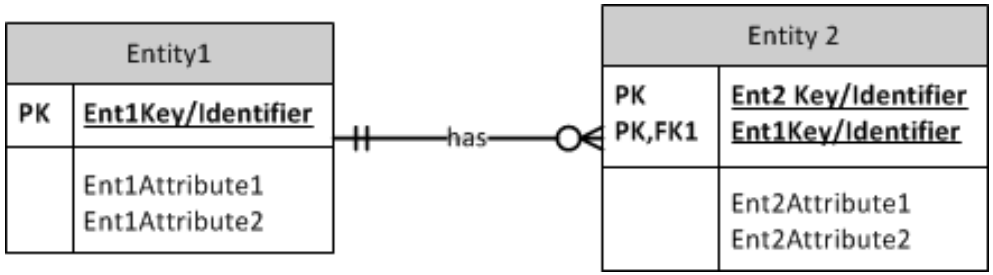
- Strong Entity
  - entity 2's PK is independent of the PKs of other entities

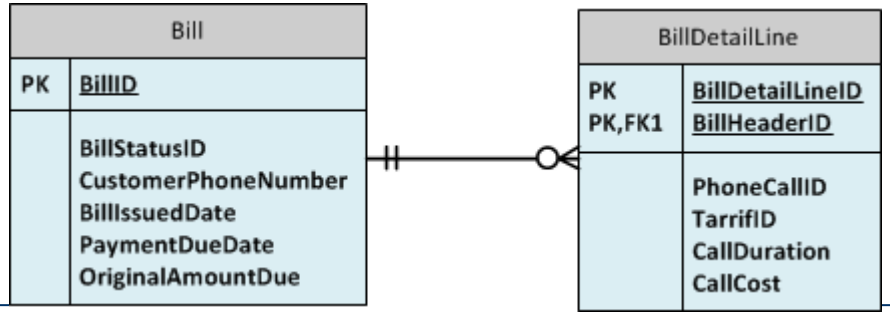




Weak Entity

- entity 2's PK depends on (includes) the PK of entity 1





© University of Melbourne 2018

Dr Greg Wadley

THE UNIVERSITY OF MELBOURNE

INFO90002 Database Systems & Information Modelling

Week 03

SQL (1)

What is SQL

- SQL – or SEQUEL – is a language used to create, access and maintain relational databases
- Based on relational algebra and relational calculus
- SQL (DML) supports CRUD (Create, Read, Update, Delete)
  - Insert, Select, Update, Delete commands
- You can see the latest SQL 2011 standard at
  - http://www.jtc1sc32.org/doc/N2151-2200/32N2153T-text_for_ballot-FDIS_9075-1.pdf
- SQL is a widely used standard and there are resources online:
  - http://en.wikipedia.org/wiki/SQL
  - http://en.wikipedia.org/wiki/Category:SQL_keywords
  - https://dev.mysql.com/doc/refman/5.7/en/sql-syntax.html
  - https://www.w3schools.com/sql/

© University of Melbourne 2018

Brief history

1974	IBM develops SEQUEL (renamed to SQL) based on Codd research
1979…	Oracle, IBM etc release RDBMS with SQL language
1986	1st SQL Standard (ANSI)
1989	2nd SQL Standard (ANSI) – includes referential integrity
1992	3rd SQL Standard (ISO) – most widely conformed to by vendors
1997…	dynamic websites enabled by SQL
1999	SQL-1999 – 4th SQL Standard (ISO) – Object support, recursion, procedures and flow control
2003	SQL-2003 – 5th SQL Standard (ISO) – XML support, auto number
2006	SQL-2006 – 6th SQL Standard (ISO) – Defines SQL use with XML
2008	SQL-2008 – 7th SQL Standard (ISO) – FETCH command added
2008	HTML 5 with SQLite built in
2011	SQL-2011 – 8th SQL Standard (ISO) – temporal databases

© University of Melbourne 2018

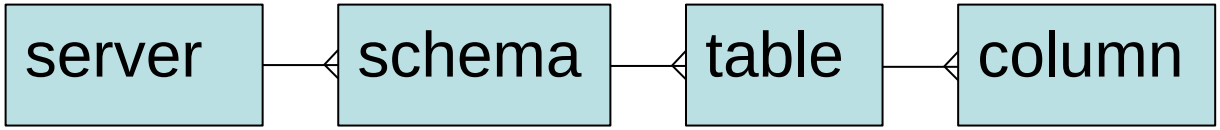
How we use SQL

- during Implementation of the database
  - Implement tables from physical design using DDL Create Table
- during Production
  - use Select commands to read the data from the tables
  - use DML Insert, Delete, Update commands to update data
  - use DML Alter, Drop commands to update the database structure

© University of Melbourne 2018

MySQL

- We are using the MySQL implementation of SQL
  - If you are using other DBMS (such as ORACLE or SQLServer) you will need to check their implementation of SQL.
    - differences can range from valid keywords to data types
- The university's MySQL server = version 5.7.9
- You can get the latest version of MySQL (5.7) from
    - http://dev.mysql.com/downloads/
    - Community edition = FOSS
  - Get syntax help for MySQL SQL statements at
    - http://dev.mysql.com/doc/refman/5.7/en/sql-syntax.html
- Explore your server with these commands:
  - show schemas; (alternatively, 'show databases')
  - show tables;
  - describe table;



© University of Melbourne 2018

The SQL Language

- Consists of:
  - Data Definition Language (DDL)
    - to define and set up the database
    - CREATE, ALTER, DROP
      - also TRUNCATE, RENAME
  - Data Manipulation Language (DML)
    - to manipulate and read data in tables
    - SELECT, INSERT, DELETE, UPDATE
      - MySQL also provides others…. eg REPLACE
  - Data Control Language (DCL)
    - to control access to the database
      - GRANT, REVOKE
  - Other Commands
    - administer the database
    - transaction control

© University of Melbourne 2018

The SELECT Statement in detail

SELECT [ALL | DISTINCT] *select_expr* [, *select_expr* ...]

List the columns (and expressions) that are returned from the query

[FROM *table_references*

Indicate the table(s) or view(s) from where the data is obtained

[WHERE *where_condition*]

Indicate the conditions on whether a particular row will be in the result

[GROUP BY {*col_name* | *expr* } [ASC | DESC], ...]

Indicate categorisation of results

[HAVING *where_condition*]

Indicate the conditions under which a particular category (group) is included in the result

[ORDER BY {*col_name* | *expr* | *position*} [ASC | DESC], ...]

Sort the result based on the criteria

[LIMIT {[*offset*,] *row_count* | *row_count* OFFSET *offset*}]

Limit which rows are returned by their return order (ie 5 rows, 5 rows from row 2)

]

© University of Melbourne 2018

Select entire contents of table

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
1 •  SELECT * FROM Customer;
2
```

Edit: Export: Autosize:

CustomerID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType
1	Peter	NULL	Smith	NULL	Personal
2	James	NULL	Jones	JJ Enterprises	Company
3	Akin	NULL	Smithies	Bay Wart	Company
4	Julie	Anne	Smythe	Konks	Company
5	Jen	NULL	Smart	BRU	Company
6	Lim	NULL	Lam	NULL	Personal
7	Kim	NULL	Unila	Saps	Company
8	James	Jay	Jones	JJ's	Company
9	Keith	NULL	Samson	NULL	Personal
NULL	NULL	NULL	NULL	NULL	NULL

© University of Melbourne 2018

Select specific columns

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
SELECT CustLastName, CustFirstName
    FROM Customer;
```

Export: Autosize:

CustLastName	CustFirstName
Smith	Peter
Jones	James
Smithies	Akin
Smythe	Julie
Smart	Jen
Lam	Lim
Unila	Kim
Jones	James
Samson	Keith

© University of Melbourne 2018

WHERE clause: select specific rows

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
SELECT CustLastName FROM Customer
    WHERE CustLastName = "Smith";
```

Export | Autosize

CustLastName
Smith

© University of Melbourne 2018

WHERE clause with LIKE

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
SELECT CustLastName FROM Customer
    WHERE CustLastName LIKE "Sm%";
```

Export Autosize

CustLastName
Smith
Smithies
Smythe
Smart

© University of Melbourne 2018

Select with ORDER BY

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
SELECT CustLastName, CustType
    FROM Customer
    ORDER BY CustLastName;
```

Export

CustLastName	CustType
Jones	Company
Jones	Company
Lam	Personal
Samson	Personal
Smart	Company
Smith	Personal
Smithies	Company
Smythe	Company
Unila	Company

```
SELECT CustLastName, CustType
    FROM Customer
    ORDER BY CustLastName DESC;
```

Export

CustLastName	CustType
Unila	Company
Smythe	Company
Smithies	Company
Smith	Personal
Smart	Company
Samson	Personal
Lam	Personal
Jones	Company
Jones	Company

© University of Melbourne 2018

Select with LIMIT

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
SELECT CustLastName, CustType
    FROM Customer
    ORDER BY CustLastName
    LIMIT 5;
```

Export

CustLastName	CustType
Jones	Company
Jones	Company
Lam	Personal
Samson	Personal
Smart	Company

```
SELECT CustLastName, CustType
    FROM Customer
    ORDER BY CustLastName
    LIMIT 5
    OFFSET 3;
```

Export

CustLastName	CustType
Samson	Personal
Smart	Company
Smith	Personal
Smithies	Company
Smythe	Company

CustLastName	CustType
Jones	Company
Jones	Company
Lam	Personal
Samson	Personal
Smart	Company
Smith	Personal
Smithies	Company
Smythe	Company
Unila	Company

(what happens if we Limit without Ordering?)

© University of Melbourne 2018

Select with GROUP BY

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
SELECT CustType, Count(CustomerID)
    FROM Customer
    GROUP BY CustType;
```

Export: Autosize:

CustType	Count(CustomerID)
Personal	3
Company	6

```
SELECT CustType, Count(CustomerID) AS Count
    FROM Customer
    GROUP BY CustType;
```

Export: Autosize:

CustType	Count
Personal	3
Company	6

© University of Melbourne 2018

Select with WHERE and GROUP BY

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
SELECT CustType, Count(CustomerID)
    FROM Customer
    WHERE CustLastName LIKE "Sm%"
    GROUP BY CustType;
```

Export Autosize

CustType	Count(CustomerID)
Personal	1
Company	3

© University of Melbourne 2018

Select with GROUP BY and HAVING

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
select custtype, count(CustomerId) from customer
where custlastname like 'Sm%'
group by custtype
having count(CustomerId) = 3;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

custtype	count(CustomerId)
Company	3

"Having" works on groups
the way "Where" works on individual rows

© University of Melbourne 2018

Inner join, Natural join

- Data about an entity is spread across 2 tables – so join them
- Inner/Equi join - Join rows where FK value = PK value

```
SELECT * FROM Customer INNER JOIN Account
    ON Customer.CustomerID = Account.CustomerID;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance	CustomerID
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25	1
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39	2
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25	2

- Natural Join gives the same result as Inner Join
  - requires PK and FK columns to have the same name

```
SELECT * FROM Customer NATURAL JOIN Account;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25

© University of Melbourne 2018

Outer join

- Outer join
  - Can be left or right (see difference below)
  - Includes records from left/right table that don’t have a matching row

```
SELECT * FROM Customer LEFT OUTER JOIN Account
    ON Customer.CustomerID = Account.CustomerID;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleNa	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance	CustomerID
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25	1
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39	2
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25	2
3	Akin	NULL	Smithies	Bay Wart	Company	NULL	NULL	NULL	NULL

```
SELECT * FROM Customer RIGHT OUTER JOIN Account
    ON Customer.CustomerID = Account.CustomerID;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance	CustomerID
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25	1
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39	2
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25	2

© University of Melbourne 2018

Cartesian join

- What if there is no join condition?

```
SELECT * FROM Customer, Account;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance	CustomerID
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25	1
2	James	NULL	Jones	JJ Enterprises	Company	1	Peter Smith	245.25	1
3	Akin	NULL	Smithies	Bay Wart	Company	1	Peter Smith	245.25	1
1	Peter	NULL	Smith	NULL	Personal	2	JJ ENt.	552.39	2
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39	2
3	Akin	NULL	Smithies	Bay Wart	Company	2	JJ ENt.	552.39	2
1	Peter	NULL	Smith	NULL	Personal	3	JJ ENt. Mgr	10.25	2
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25	2
3	Akin	NULL	Smithies	Bay Wart	Company	3	JJ ENt. Mgr	10.25	2

NOT CORRECT: lack of join conditions -> Cartesian product

(every row in Customer combined with every record in Account)

© University of Melbourne 2018

Dr Greg Wadley

INFO 90002 Database Systems & Information Modelling

Week 04

Data Modelling 2

Today's Session…

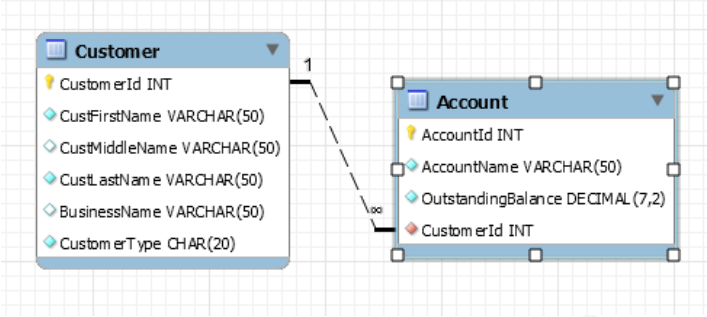
- More relationship types
  - Many to Many
    - Associative entity
  - One to One
  - Recursive / Unary relationships
    - One-to-one, One-to-many, Many-to-Many
  - Multiple One to Many Relationships
    - between the same pair of entities
  - Ternary relationships
    - 3 tables are involved

© University of Melbourne 2018

Recap: one-to-many relationships

- Data are spread across 2 tables
- Inner join = Join rows where FK value = PK value

CustID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType
1	Peter		Smith		Personal
2	James		Jones	JJ Enterprises	Company



AccountID	AccountName	OutstandingBalance	CustID
01	Peter Smith	245.25	1
05	JJ Ent.	552.39	2
06	JJ Ent. Mgr	10.25	2

```
SELECT *
FROM Customer INNER JOIN Account
ON Customer.Customerid = Account.Customerid;
```

CustomerId	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustomerType	AccountId	AccountName	OutstandingBalance	CustomerId
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25	1
2	James	NULL	Jones	JJ Enterprises	Company	5	JJ Ent.	552.39	2
2	James	NULL	Jones	JJ Enterprises	Company	6	JJ Ent. Mgr	10.25	2

© University of Melbourne 2018

Data Integrity Constraints

- Domain Integrity
  - Valid values and domain
    - selection of data type constrains possible data values
  - Default value
    - takes this value if no explicit value is given on Insert
  - Null value control
    - allows or prohibits empty fields
  - Check constraint
    - limits range of allowable values (not available in MySQL)
- Entity Integrity Constraints
  - Primary key cannot be null
  - No component of a composite key can be null
  - Primary key must be unique

© University of Melbourne 2018

Referential Integrity

- Each non-null FK value must match a PK value
  - Rules for update and delete (SQL CREATE statement)
    - RESTRICT or NO ACTION
      - Don't allow deletes or updates of the parent table if related rows exist in the child table
    - CASCADE
      - Automatically delete/update the child table if related rows are deleted/updated in the parent table
    - SET NULL
      - Set the foreign key to NULL in the child table if deleting/updating the key in parent table

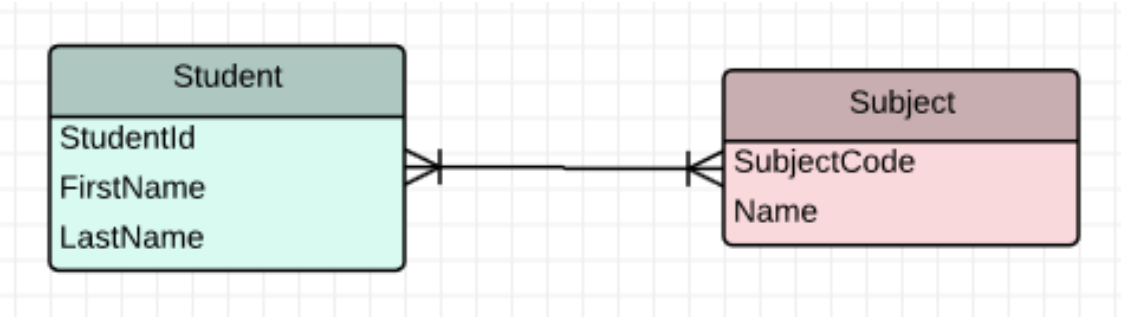
```
CONSTRAINT `fk_StudentSubject_Student`
  FOREIGN KEY (`StudentId`)
  REFERENCES `Student` (`StudentId`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
CONSTRAINT `fk_StudentSubject_Subject1`
  FOREIGN KEY (`SubjectCode`)
  REFERENCES `Subject` (`SubjectCode`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION)
```


© University of Melbourne 2018

Many-to-Many Relationships

Many to Many relationships

- Example: we need to design a Student Records database
- Each student will take more than one subject,
  and each subject will be taken by more than one student
- Where do we record who took what subject and their result?



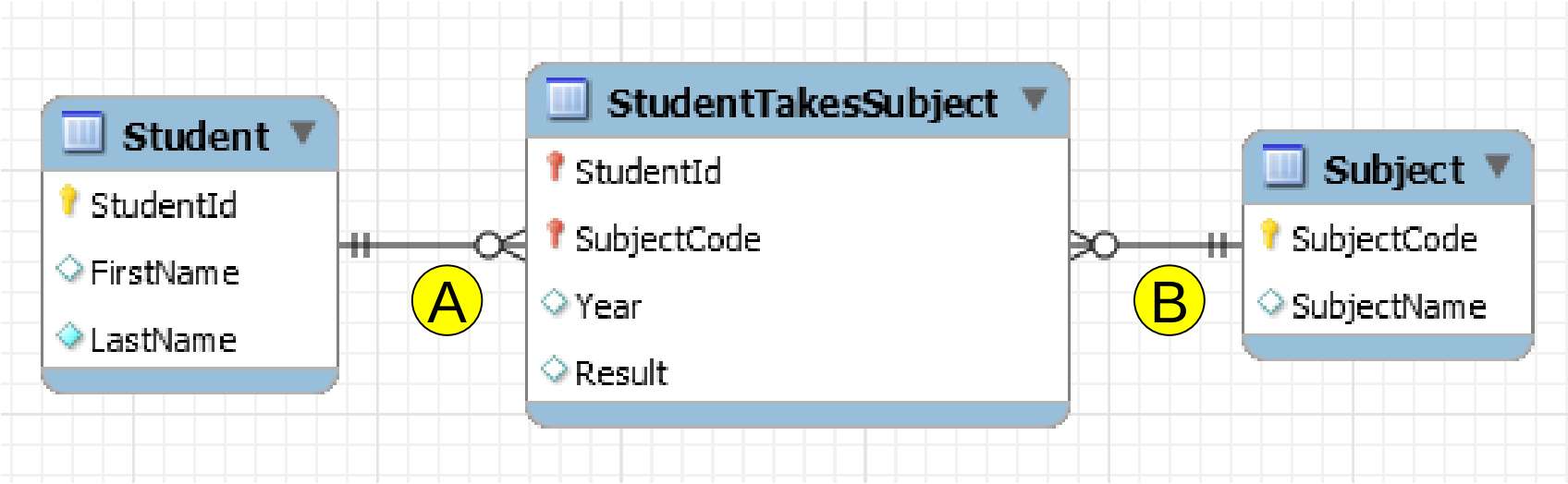
StudentId	FirstName	LastName
11111	John	Lennon
22222	Paul	McCartney
33333	George	Harrison

SubjCode	Name
INFO90002	Database
ISYS90026	Fundamentals
ISYS90081	Organisational

© University of Melbourne 2018

Problems modelling Many-Many…

- Relational database doesn't directly support M-M…
  - so we create an Associative Entity between the other 2 entities (when converting Conceptual to Logical model)
  - each of these 2 relationships is like any 1-M relationship

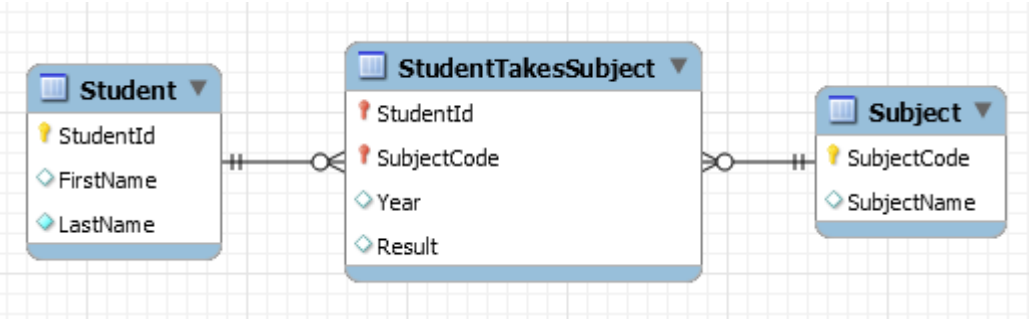


- We can add attributes to the associative entity to record when the student took the subject and the result they got.
- Associate Entities are also called 'Join Tables' and many other names, see https://en.wikipedia.org/wiki/Junction_table

© University of Melbourne 2018

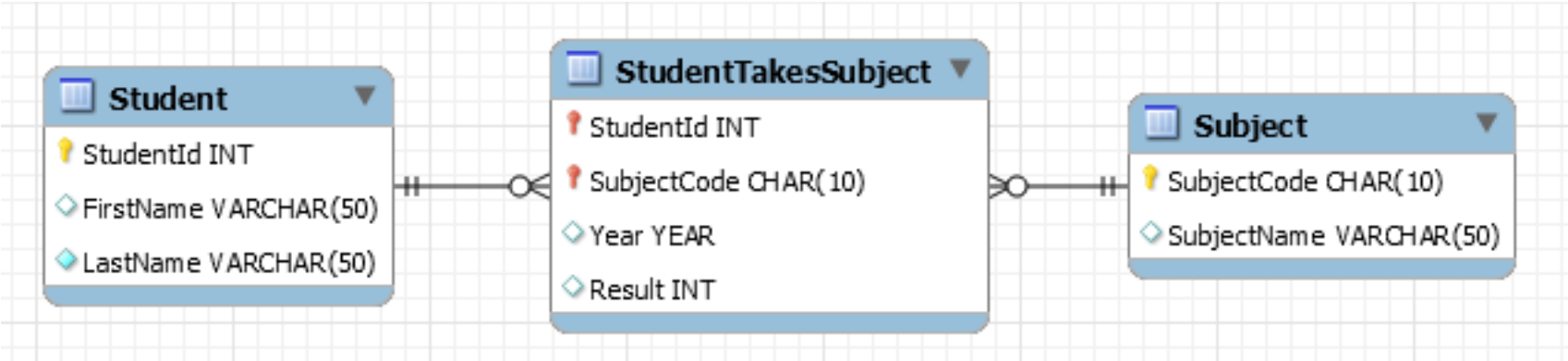
Associative Entities

- When to create
  - when going from Conceptual to Logical phase of design
  - to implement a Many-to-Many relationship
  - to implement a Ternary relationship
- The associative entity
  - has an independent meaning
  - has a unique identifier, usually a combination of FKs
  - may have attributes other than the FKs
  - may participate in other relationships



© University of Melbourne 2018

Many-Many Physical Model



- Choose data types
- Example decisions:
  - are results integers or floating point?
  - are StudentIds number or strings?
  - how long are people's names?

© University of Melbourne 2018

Many-Many CREATE statements

- Order of creation is important!
  - so is order of deletion…
- Create tables *without* foreign keys first
  - *drop* tables without foreign keys *last*

```sql
-- -----------------------------------------------------
-- Table `Student`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Student` (
  `StudentId` INT NOT NULL,
  `FirstName` VARCHAR(50) NULL,
  `LastName` VARCHAR(50) NULL,
  PRIMARY KEY (`StudentId`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Subject`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `Subject` (
  `SubjectCode` CHAR(10) NOT NULL,
  `SubjectName` VARCHAR(50) NULL,
  PRIMARY KEY (`SubjectCode`))
ENGINE = InnoDB;
```


© University of Melbourne 2018

Many-Many CREATE statements

- Order of creation is important!
  - so is order of deletion…
- Create tables *with* foreign keys last
  - *drop* tables with foreign keys *first*

```
-- -----------------------------------------------------
-- Table `StudentTakesSubject`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `StudentTakesSubject` (
  `StudentId` INT NOT NULL,
  `SubjectCode` CHAR(10) NOT NULL,
  `Year` YEAR NULL,
  `Result` INT NULL,
  PRIMARY KEY (`StudentId`, `SubjectCode`),
  INDEX `fk_StudentSubject_Subject1_idx` (`SubjectCode` ASC),
  CONSTRAINT `fk_StudentSubject_Student`
    FOREIGN KEY (`StudentId`)
    REFERENCES `Student` (`StudentId`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_StudentSubject_Subject1`
    FOREIGN KEY (`SubjectCode`)
    REFERENCES `Subject` (`SubjectCode`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```


© University of Melbourne 2018

Adding data to a M-M relationship

- Insert into the join table *last*

```sql
-- ----------------------------------------------------
-- Data for table `Student`
-- ----------------------------------------------------
INSERT INTO `Student` VALUES (11111, 'John', 'Lennon');
INSERT INTO `Student` VALUES (22222, 'Paul', 'McCartney');
INSERT INTO `Student` VALUES (33333, 'George', 'Harrison');
INSERT INTO `Student` VALUES (44444, 'Ringo', 'Starr');
-- ----------------------------------------------------
-- Data for table `Subject`
-- ----------------------------------------------------
INSERT INTO `Subject` VALUES ('INFO90002', 'Database Systems and Information Modelling'
INSERT INTO `Subject` VALUES ('ISYS90026', 'Fundamentals of Information Systems');
INSERT INTO `Subject` VALUES ('ISYS90081', 'Organisational Processes');
INSERT INTO `Subject` VALUES ('ISYS90048', 'Managing ICT Infrastructure');
INSERT INTO `Subject` VALUES ('ISYS90045', 'Professional ICT Consulting');
-- ----------------------------------------------------
-- Data for table `StudentTakesSubject`
-- ----------------------------------------------------
INSERT INTO `StudentTakesSubject` VALUES (11111, 'INFO90002', 2014, 85);
INSERT INTO `StudentTakesSubject` VALUES (11111, 'ISYS90026', 2015, 77);
INSERT INTO `StudentTakesSubject` VALUES (22222, 'INFO90002', 2014, 90);
```


© University of Melbourne 2018

How to read complete student results

- Three table join

```
SELECT *
FROM Student NATURAL JOIN StudentTakesSubject
NATURAL JOIN Subject;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

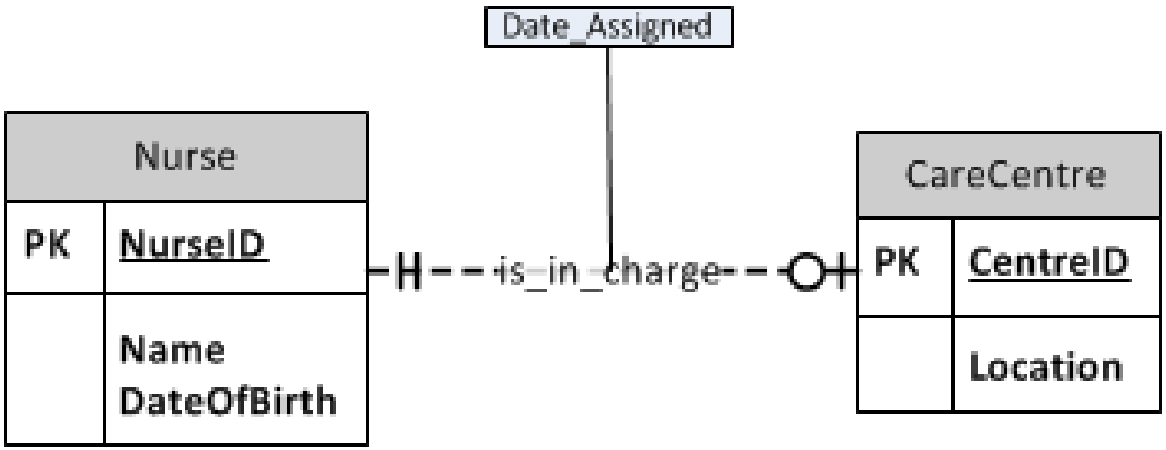
SubjectCode	StudentId	FirstName	LastName	Year	Result	SubjectName
INFO90002	11111	John	Lennon	2014	85	Database Systems and...
ISYS90026	11111	John	Lennon	2015	77	Fundamentals of Info...
INFO90002	22222	Paul	McCartney	2014	90	Database Systems and...

© University of Melbourne 2018

One-to-One Relationships

Binary One-One Relationship

- Given this example… How do we implement it…



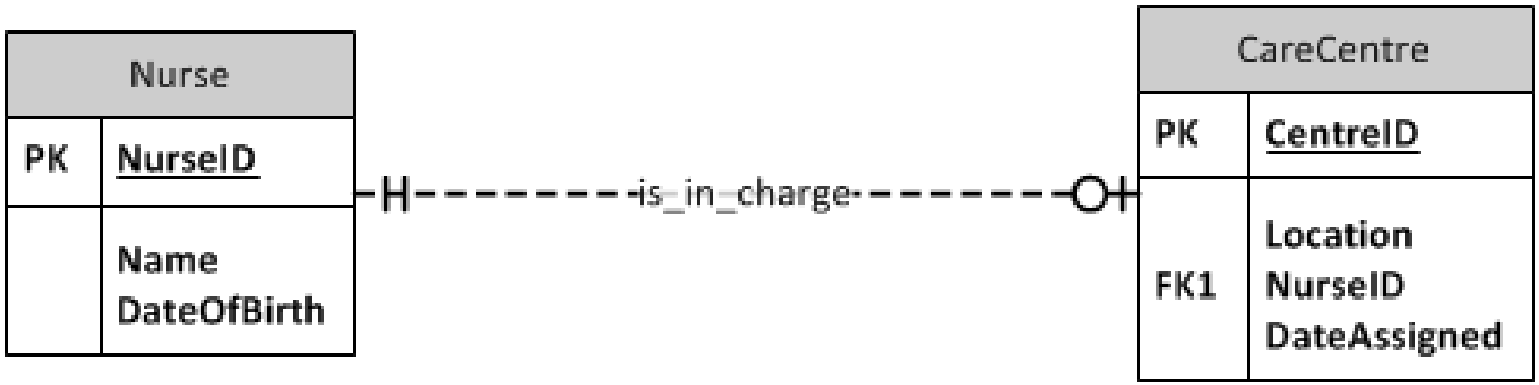
- Note: Date_assigned is an attribute of the relationship

- Need to decide whether to put the foreign key inside Nurse or CareCentre (in which case you would have the Date_Assigned in the same location)
  - Where would the least NULL values be?
  - The rule is the OPTIONAL side of the relationship gets the foreign key

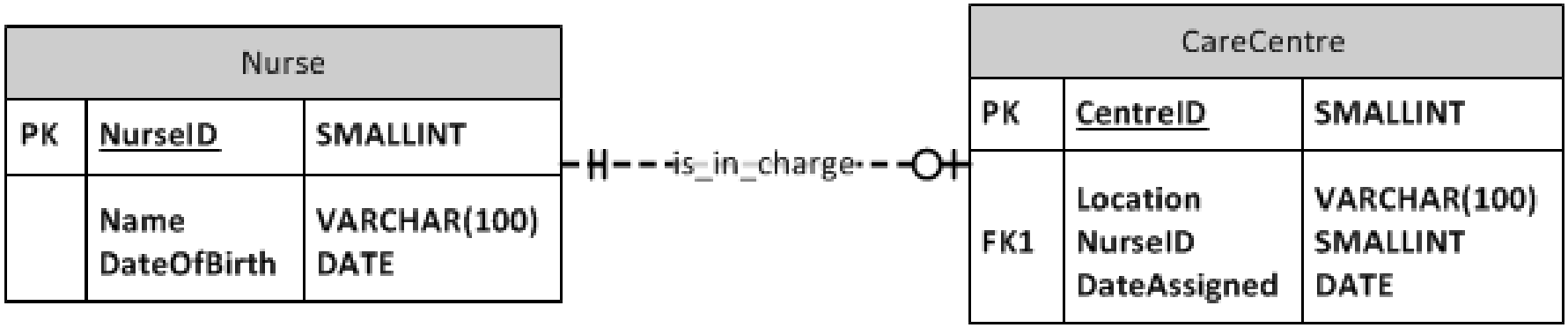
© University of Melbourne 2018

Binary One-One Relationship – Logical and Physical

- Logical
  - Nurse = (NurseID, Name, DateOfBirth)
  - CareCentre = (CentreID, Loction, NurseID, DateAssigned)



- Physical



© University of Melbourne 2018

1-1 Implementation in SQL

```sql
CREATE TABLE Nurse (
  NurseID                 smallint,
  Name                    varchar(100)    NOT NULL,
  DateOfBirth             varchar(100)    NOT NULL,
  PRIMARY KEY  (NurseID)
) ENGINE=InnoDB;


CREATE TABLE CareCentre (
  CentreID                  smallint,
  Location                  varchar(150)  NOT NULL,
  NurseID                   smallint      NOT NULL,
  DateAssigned              DATE          NOT NULL,
  PRIMARY KEY  (CentreID),
  FOREIGN KEY (NurseID) REFERENCES Nurse(NurseID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
) ENGINE=InnoDB;
```

- have to insert into Nurse 1st, then into CareCentre
- query it by joining the Nurse and CareCentre tables

© University of Melbourne 2018

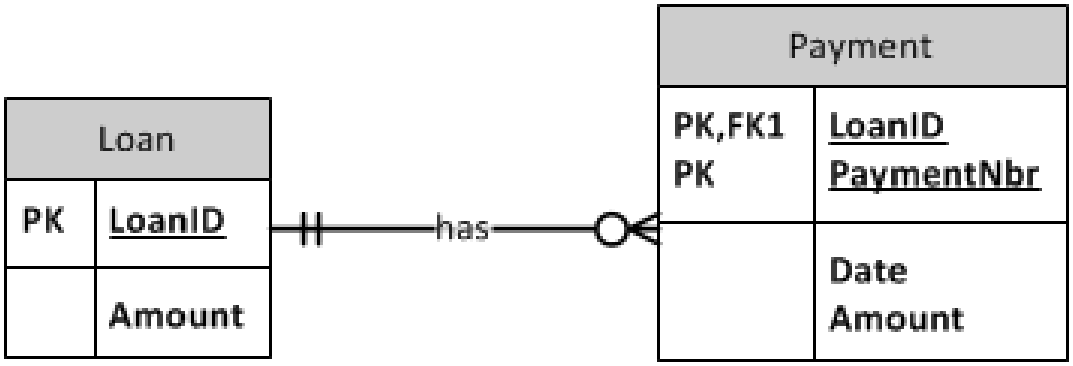
Summary of Binary Relationships

- One-to-Many
  - primary key on ONE side becomes foreign key on MANY side
- Many-to-Many
  - create an Associative Entity (a new table) with a compound primary key consisting of 2 FKs that refer to the other 2 tables
  - you then have two One-to-Many joins
- One-to-One
  - decide in which table to put the foreign key
  - foreign key on the optional side refers to primary key on the mandatory side

© University of Melbourne 2018

1-M special case – "Identifying Relationship"

- How to deal with an Identifying relationship
  - i.e. a relationship between weak child and strong parent tables
  - Foreign Key defines the relationship at the crows foot end.
  - and FK becomes part of the Primary Key

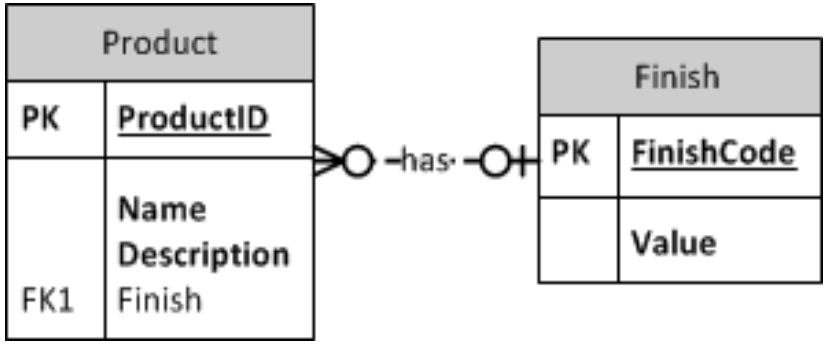


  - Logical Design
    - Loan = (LoanID, Amount)
    - Payment = (LoanID, PaymentID, Date, Amount)
  - Physical Design = normal one-to-many relationship

© University of Melbourne 2018

1-M special case – "Lookup table"

- Consider the following logical design



ProductID	Nam	…	Finish
1	Chai		A
2	Desk		C
3	Tabl		B
4	Book		A

Code	Value
A	Birch
B	Maple
C	Oak

- Physical design decision
  - Implement as 2 tables or one? trade-off = speed vs data integrity

ProductID	Name	…	Finish
1	Chair		Birch
2	Desk		Oak
3	Table		Maple
4	Bookcase		Birch

© University of Melbourne 2018

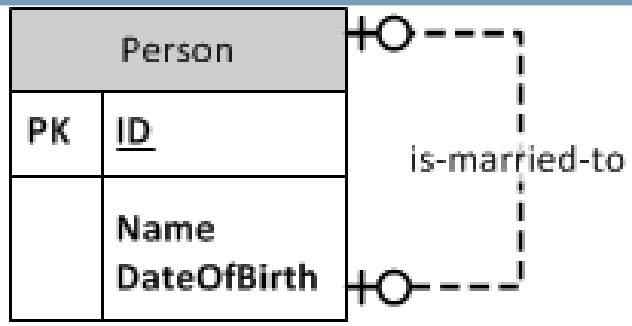
Unary Relationships

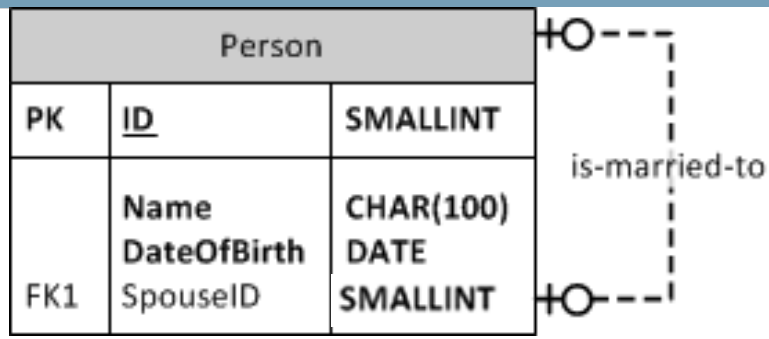
Unary Relationships

- "Unary" means there is only one table
  - the table is in a relationship with itself
  - (a row in the table can be linked to other rows in the same table)
- Operate in the same way exactly as binary relationships
  - One-to-One
    - put a Foreign key in the entity
  - One-to-Many
    - put a Foreign key in the entity
  - Many-to-Many
    - create an extra table - Associative Entity
    - put two Foreign keys in the Associative Entity
      - the two FKs need different names
      - the FKs become the combined PK of the Associative Entity

© University of Melbourne 2018

Unary – One-to-One





Logical Design

(ID, Name, DateOfBirth, SpouseID)

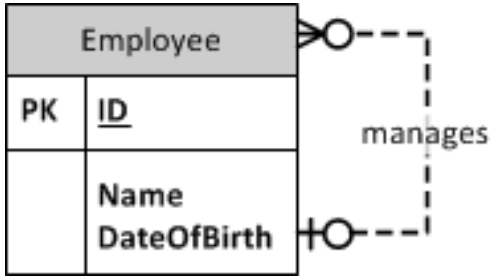
Physical Design

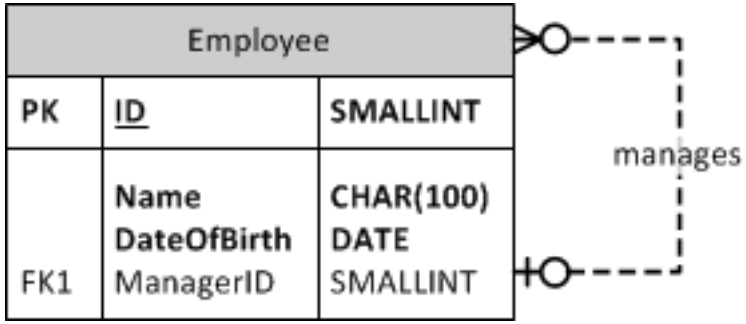
```
CREATE TABLE Person (
   ID                  smallint,
   Name                varchar(150)  NOT NULL,
   DateOfBirth         DATE          NOT NULL,
   SpouseID            smallint      ~~NOT NULL~~,
   PRIMARY KEY  (ID),
   FOREIGN KEY (SpouseID) REFERENCES Person(ID)
         ON DELETE RESTRICT
         ON UPDATE CASCADE
) ENGINE=InnoDB;
```

ID	Name	DOB	SpouseID
1	Ann	1969-06-12	3
2	Fred	1971-05-09	
3	Chon	1982-02-10	1
4	Nancy	1991-01-01	

© University of Melbourne 2018

Unary – One-to-Many





Logical Design

(ID, Name, DateOfBirth, ManagerID)

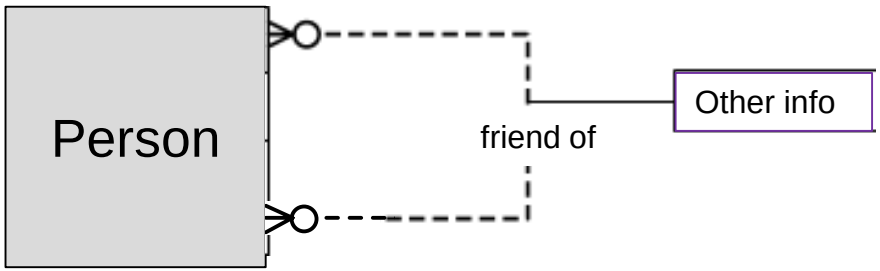
Physical Design

```
CREATE TABLE Employee (
   ID                  smallint,
   Name                varchar(150)  NOT NULL,
   DateOfBirth         DATE          NOT NULL,
   ManagerID           smallint      ~~NOT NULL~~,
   PRIMARY KEY  (ID),
   FOREIGN KEY (ManagerID) REFERENCES Employee(ID)
         ON DELETE RESTRICT
         ON UPDATE CASCADE
) ENGINE=InnoDB;
```

ID	Name	DOB	MngrID
1	Ann	1969-06-12	
2	Fred	1971-05-09	1
3	Chon	1982-02-10	1
4	Nancy	1991-01-01	1

© University of Melbourne 2018

Unary – Many-to-Many

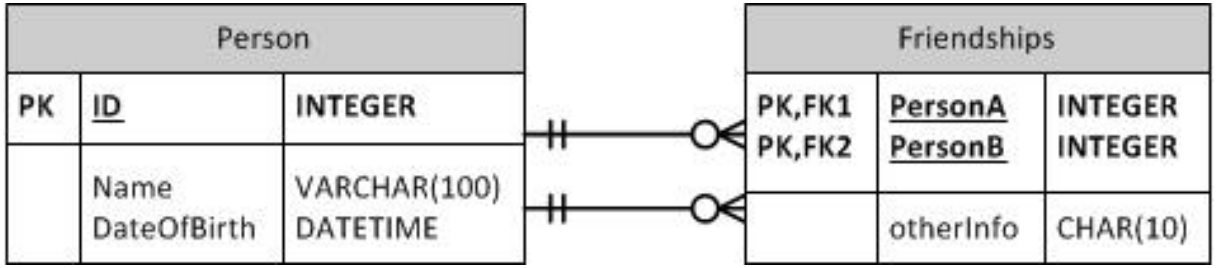


- Logical Design
  - Set up Associative Entity as for any M-M relationship
    - Person = (ID, Name, DateOfBirth)
    - Friendship = (PersonA, PersonB, otherInfo)

© University of Melbourne 2018

Unary – Many-to-Many

- Physical Design



- Implementation

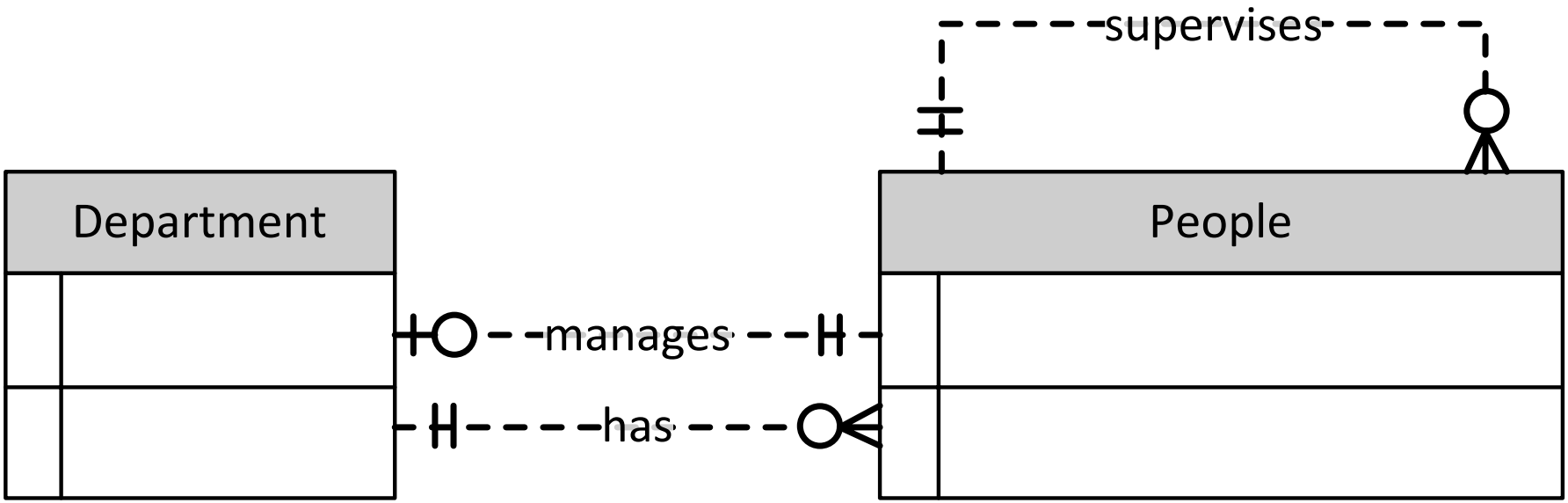
```
-- ----------------------------------------------------------
-- Table `mydb`.`Person`
-- ----------------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Person` (
  `ID` INT NOT NULL,
  `Name` VARCHAR(50) NULL,
  `DateOfBirth` DATE NULL,
  PRIMARY KEY (`ID`))
ENGINE = InnoDB;
```

```
-- ----------------------------------------------------------
-- Table `mydb`.`Friendship`
-- ----------------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Friendship` (
  `PersonA` INT NOT NULL,
  `PersonB` INT NOT NULL,
  `otherInfo` CHAR(10) NULL,
  PRIMARY KEY (`PersonA`, `PersonB`),
  INDEX `fk_Friendship_Person1_idx` (`PersonB` ASC),
  CONSTRAINT `fk_Friendship_Person`
    FOREIGN KEY (`PersonA`)
    REFERENCES `mydb`.`Person` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Friendship_Person1`
    FOREIGN KEY (`PersonB`)
    REFERENCES `mydb`.`Person` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```


© University of Melbourne 2018

Multiple Relationships

- Entities can be related in several ways simultaneously



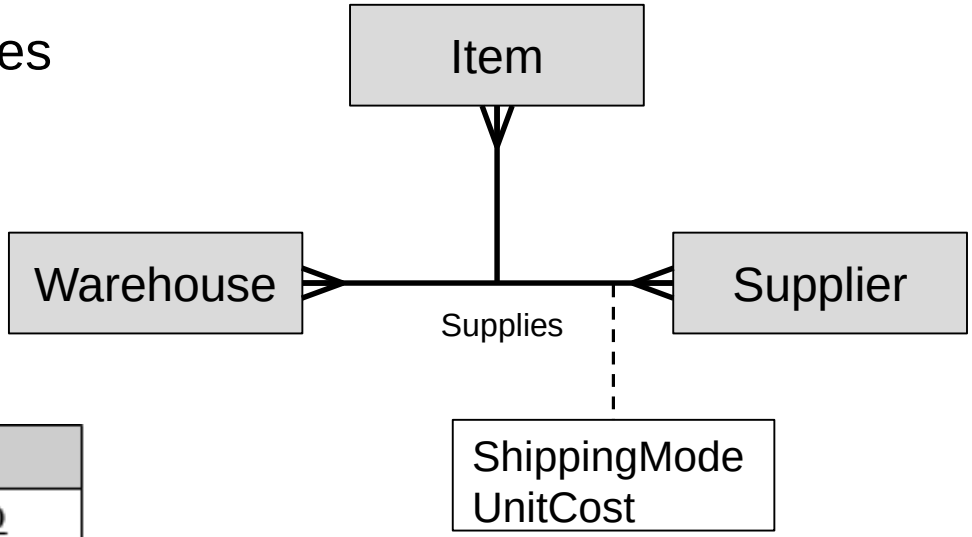
- Treat this the same was as any other One-to-Many, One-to-One relationship

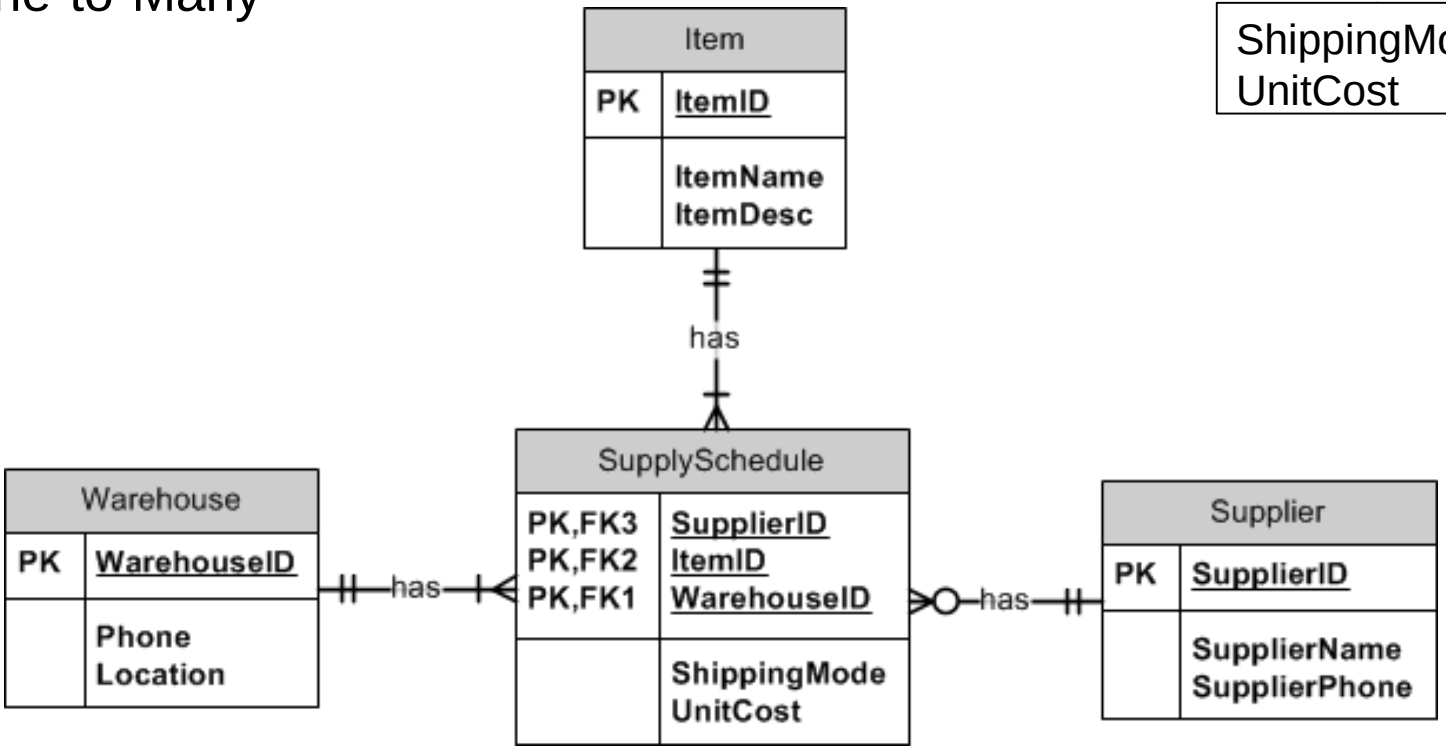
© University of Melbourne 2018

Ternary Relationships

Ternary relationships

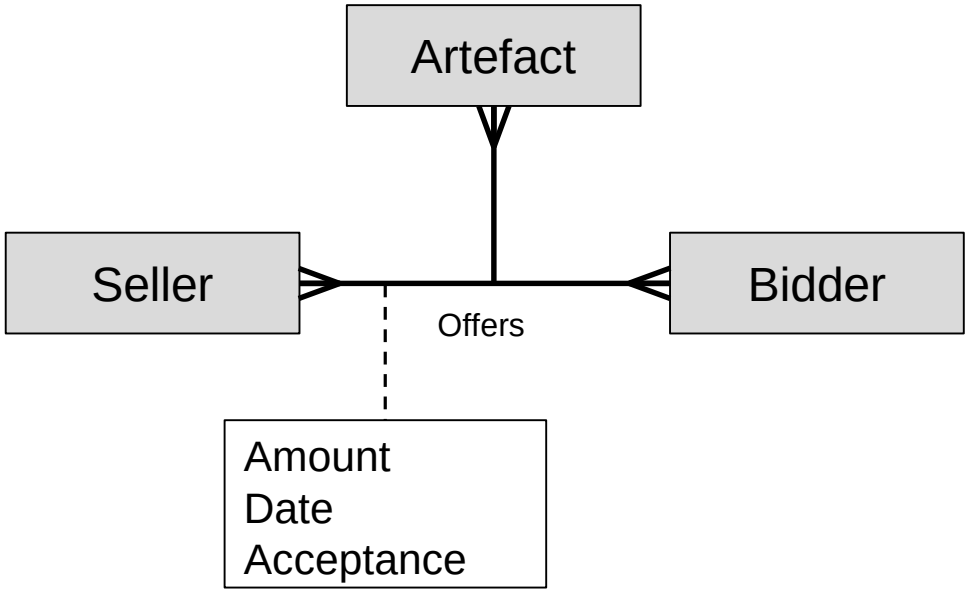
- Relationships between **three** entities
- Generate an Associative Entity
- Set up three One-to-Many relationships
- These are like any other One-to-Many





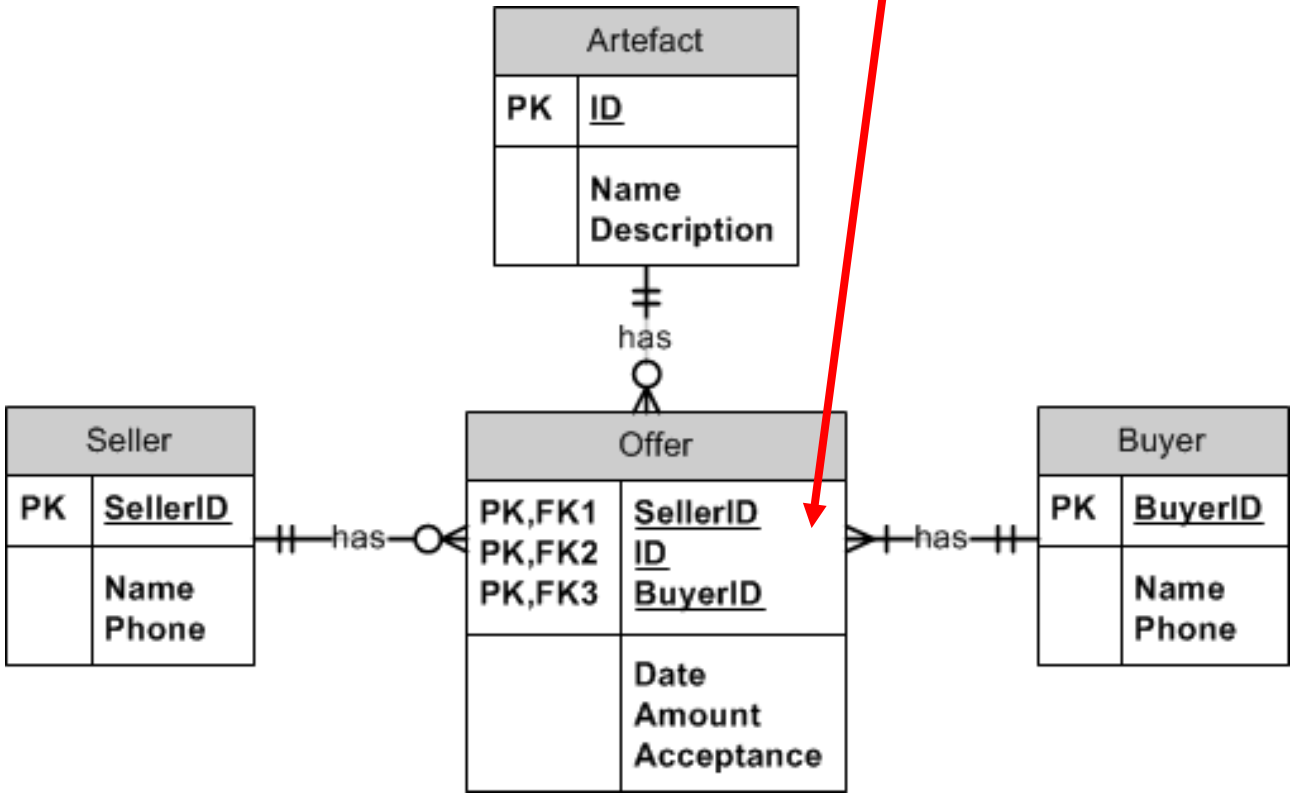
© University of Melbourne 2018

Ternary example – auction



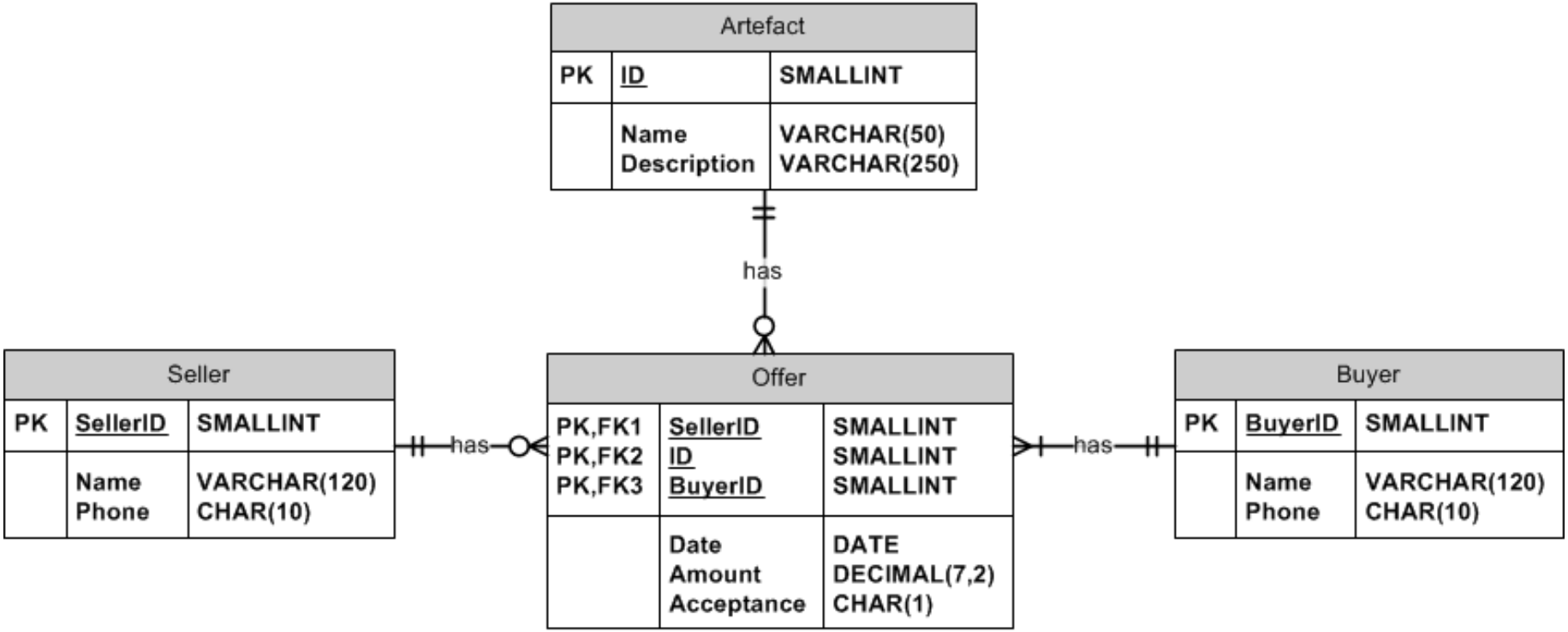
BUSINESS RULES:

- an Artefact can be sold several times by several different Sellers (over time)
- each Buyer can only make 1 offer to buy per Seller/Artefact combination



© University of Melbourne 2018

Auction Bids - Physical



© University of Melbourne 2018

Auction Bids – Table Creation

```
CREATE TABLE Seller (
  SellerID          smallint,
  Name              varchar(120) NOT NULL,
  Phone             char(10)     NOT NULL,
  PRIMARY KEY  (SellerID)
) ENGINE=InnoDB;
```

```
CREATE TABLE Buyer (
  BuyerID          smallint,
  Name              varchar(120) NOT NULL,
  Phone             char(10)     NOT NULL,
  PRIMARY KEY  (BuyerID)
) ENGINE=InnoDB;
```

```
CREATE TABLE Artefact (
  ID            smallint,
  Name          varchar(50)     NOT NULL,
  Description   varchar(250)    NOT NULL,
  PRIMARY KEY  (ID)
) ENGINE=InnoDB;
```


© University of Melbourne 2018

Auction Bids – Table Creation

```
CREATE TABLE Offer (
  SellerID          smallint     NOT NULL,
  ArtefactID        smallint     NOT NULL,
  BuyerID           smallint     NOT NULL,
  Date              DATE         NOT NULL,
  Amount            DECIMAL(12,2) NOT NULL,
  Acceptance        CHAR(1) NOT NULL DEFAULT "N",
  PRIMARY KEY  (SellerID, ArtefactID, BuyerID),
  FOREIGN KEY (ArtefactID) REFERENCES Artefact(ID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
  FOREIGN KEY (SellerID) REFERENCES Seller(SellerID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
  FOREIGN KEY (BuyerID) REFERENCES Buyer(BuyerID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
) ENGINE=InnoDB;
```


© University of Melbourne 2018

Auction Bids – Data Creation

```
INSERT INTO Seller VALUES (1, "Abby", "0233232232");
INSERT INTO Seller VALUES (2, "Ben", "0311111111");

INSERT INTO Buyer VALUES (1, "Maggie", "0333333333");
INSERT INTO Buyer VALUES (2, "Nicole", "0444444444");

INSERT INTO Artefact VALUES (1, "Vase", "Old Vase");
INSERT INTO Artefact VALUES (2, "Knife", "Old Knife");

INSERT INTO Offer VALUES (1, 1, 1, "2012-06-20", 81223.23, DEFAULT);
INSERT INTO Offer VALUES (1, 1, 2, "2012-06-20", 82223.23, DEFAULT);
INSERT INTO Offer VALUES (2, 2, 1, "2012-06-20", 19.95, DEFAULT);
INSERT INTO Offer VALUES (2, 2, 2, "2012-06-20", 23.00, DEFAULT);
```

- list all Offers. Show Artefact, Seller, Buyer and Offer details
- this is a FOUR table join

```
SELECT * FROM Artefact
       INNER JOIN Offer ON Artefact.ID = Offer.ArtefactID
       INNER JOIN Seller ON Seller.SellerID = Offer.SellerID
       INNER JOIN Buyer ON Buyer.BuyerID = Offer.BuyerID;
```


© University of Melbourne 2018

Ternary Query Output

ID	Name	Description	SellerID	ArtefactID	BuyerID	Date	Amount	Accep	SellerID	Name	Phone	BuyerID	Name	Phone
1	Vase	Old Vase	1	1	1	2012-06-20	81223.23	N	1	Abby	0233232232	1	Maggie	0333333333
1	Vase	Old Vase	1	1	2	2012-06-20	82223.23	N	1	Abby	0233232232	2	Nicole	0444444444
2	Knife	Old Knife	2	2	1	2012-06-20	19.95	N	2	Ben	0311111111	1	Maggie	0333333333
2	Knife	Old Knife	2	2	2	2012-06-20	23.00	N	2	Ben	0311111111	2	Nicole	0444444444

- Note the value of Accepted
  - “N” – the default value from our create statement
- Note that some columns have ambiguous names
  - SellerID
  - BuyerID
  - Name
  - Phone

© University of Melbourne 2018

Better output by using aliases

```
SELECT A.ID, A.Name AS Artefact, A.Description AS ArtDesc, Date AS OfferDate,
       Amount AS OfferAmount, Acceptance AS OfferAccepted, S.SellerID,
       S.Name AS Seller, S.Phone AS SellerPhone, B.BuyerID, B.Name AS Buyer,
       B.Phone AS BuyerPhone
       FROM Artefact A
            INNER JOIN Offer O ON A.ID = O.ArtefactID
            INNER JOIN Seller S ON S.SellerID = O.SellerID
            INNER JOIN Buyer B ON B.BuyerID = O.BuyerID;
```

ID	Artefact	ArtDesc	OfferDate	OfferAmount	OfferAccepted	SellerID	Seller	SellerPhone	BuyerID	Buyer	BuyerPhone
1	Vase	Old Vase	2012-06-20	81223.23	N	1	Abby	0233232232	1	Maggie	0333333333
1	Vase	Old Vase	2012-06-20	82223.23	N	1	Abby	0233232232	2	Nicole	0444444444
2	Knife	Old Knife	2012-06-20	19.95	N	2	Ben	0311111111	1	Maggie	0333333333
2	Knife	Old Knife	2012-06-20	23.00	N	2	Ben	0311111111	2	Nicole	0444444444

- aliases for table names: “A” “O” “S” “B”
- aliases for column names: Artefact, ArtDesc etc

© University of Melbourne 2018

Dr Greg Wadley

INFO90002
Database Systems & Information Modelling

Week 05

Normalisation

This lecture

- Normalisation
  - 1st Normal Form (1NF)
  - 2nd Normal Form (2NF)
  - 3rd Normal Form (3NF)
  - Boyce Codd Normal Form (BCNF)

© University of Melbourne 2018

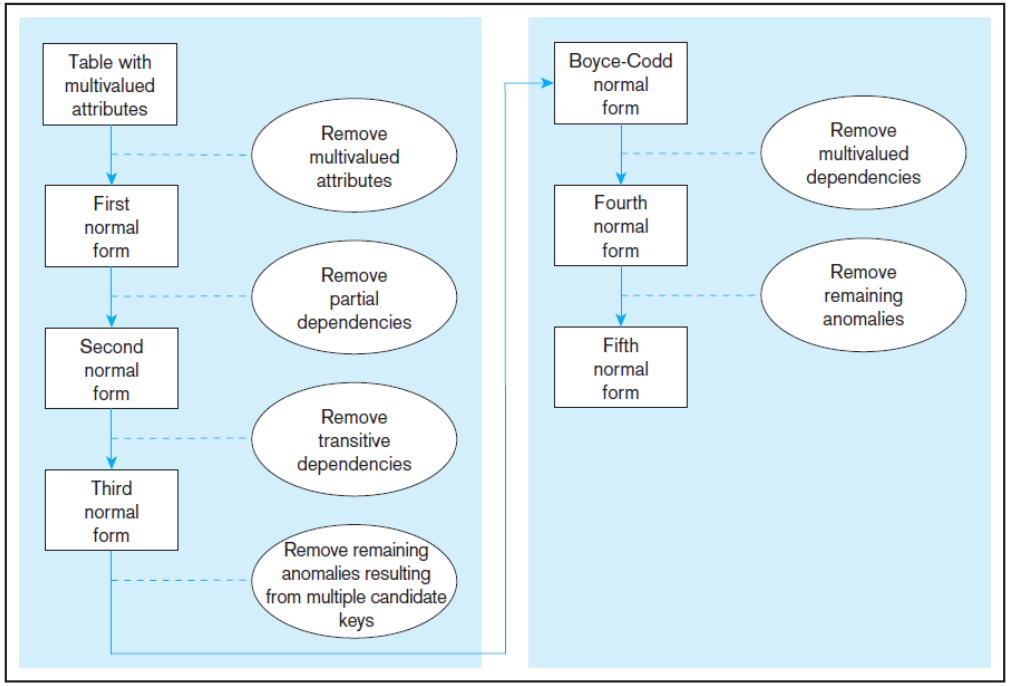
Normalisation is...

- A theoretical foundation for the relational model
- Application of a series of rules that improve db design
- Last phase of logical design
- Normalisation rules are designed to
  - Prevent data redundancy
  - Prevent update anomalies
- Normalisation is biased towards optimizing *Updates*
  - i.e. it assumes non-key columns are updated frequently
  - thus it can slow down *Selects* because of the need to join tables
  - thus designers sometimes “de-normalise” some tables to improve performance
- Every attribute must be functionally dependent on
  “The key, the whole key, and nothing but the key.”

© University of Melbourne 2018

Normalisation: a process

- Normal Form
  - the state of a relation resulting from applying rules about the functional dependency of some attributes upon others
- Design progresses in stages from 1NF via 3NF, possibly as far as 6NF
  - in this course we only cover 1NF to 3.5NF (Boyce-Codd NF)
  - this is often sufficient in practice



flowchart from Hoffer et al. *Modern Database Management*

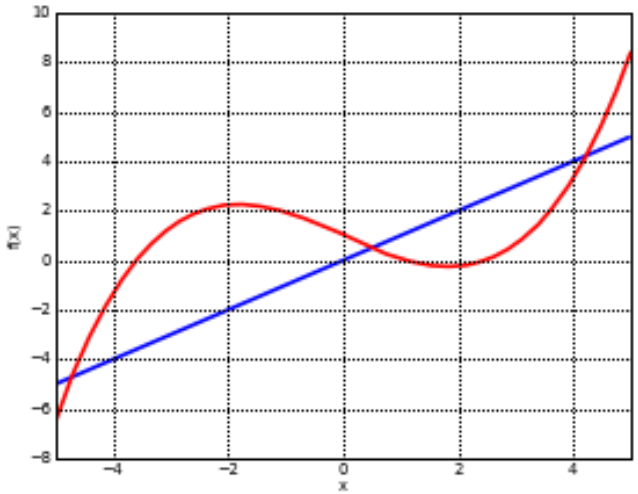
© University of Melbourne 2018

Definitions

- Functional dependency
  - a set of attributes X *determines* another set of attributes Y iff each value of X is associated with only one value of Y
  - written $X \rightarrow Y$ (similar to $y = f(x)$)
- Determinants
  - the attribute(s) on the *left hand side* of the arrow
- Key and Non-Key attributes
  - each attribute is either part of the primary key or it is not
- Partial functional dependency
  - a functional dependency of one or more non-key attributes upon *part (but not all)* of the primary key
- Transitive dependency
  - a functional dependency between 2 (or more) non-key attributes

© University of Melbourne 2018

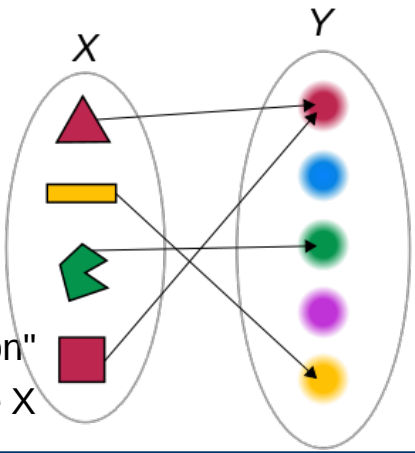
Functions and functional dependence



y = f(x)
for each x there is no more than one y

CustId	Firstname	Lastname
1	Fred	Brown
2	Fred	Smith
3	Julia	Brown
4	Tony	Abbott
5	Julia	Gillard

CustId determines Firstname and Lastname
neither Firstname nor Lastname determine anything



the "colour-of-the-shape function"
X determines Y, but Y does not determine X

© University of Melbourne 2018

Functional dependency

- A relationship between the attributes of one relation
  - one or more attributes determine the value of another attribute
- A primary key must functionally determine all non-key attributes of an entity
  - stock exchange company code → company name and details
    - If we know the company code, we know the value of company name
    - (http://www.asx.com.au/prices/understanding-asx-ticker-codes.htm)
  - studentId, subjectCode → dateCompleted, resultObtained
    - for any combination of studentId and subjectCode, there is only one dateCompleted and one resultObtained
- Determinant
  - only primary keys should be determinants

© University of Melbourne 2018

Hierarchy of Normal Forms

- 1NF
  - Any *multivalued* attributes and repeating groups have been removed. There is a SINGLE value (possibly null) at the intersection of each row and column of the table.
- 2NF
  - Any *partial* functional dependencies have been removed (ie all non-key attributes are identified by the WHOLE key)
- 3NF
  - Any *transitive* dependencies have been removed (ie non-key attributes are identified by ONLY the PRIMARY key)
- BCNF
  - Any remaining anomalies that result from functional dependencies have been removed (ie because there was more than one candidate primary key for the same non-keys)

© University of Melbourne 2018

Normalisation example

- To begin:
  - Write in relational notation
    List all the attributes that must be recorded, as one big relation
  - Don't include anything that you can derive (e.g. total sales)
  - Use parentheses "()" to indicate repeating groups
  - Underline the primary key

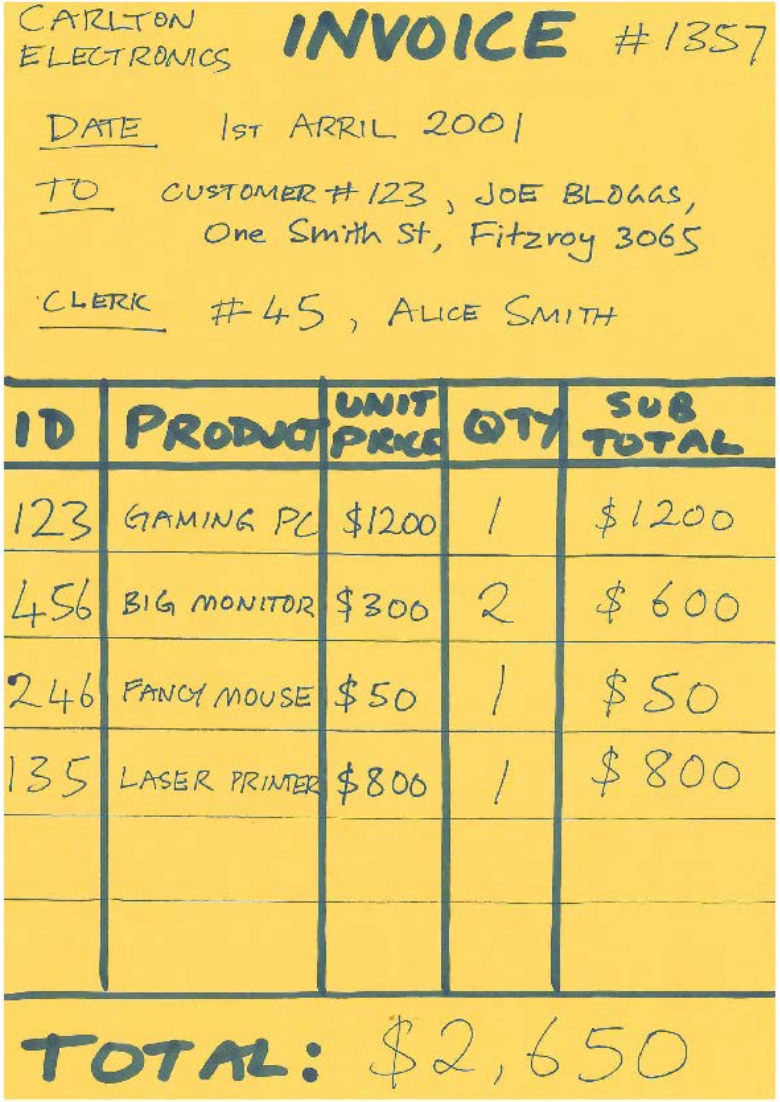
  - INVOICE (<u>InvoiceNo</u>, Date, CustomerNo, CustomerName, CustomerAddress, ClerkNo, ClerkName, (ProductNo, ProductDesc, UnitPrice, Qty))

© University of Melbourne 2018

First normal form (1NF)

- Separate repeating groups into new tables
  - if an attribute is multi-valued,
  - or a group of attributes is repeated several times for one entity,
  - create a new table containing the repeating data

example: paper invoices typically include a repeating group which represents each product sold in that sale


CARLTON ELECTRONICS **INVOICE** #1357

DATE 1ST ARRIL 2001

TO CUSTOMER #123, JOE BLOGGS, One Smith St, Fitzroy 3065

CLERK #45, ALICE SMITH

ID	PRODUCT	UNIT PRICE	QTY	SUB TOTAL
123	GAMING PC	$1200	1	$1200
456	BIG MONITOR	$300	2	$600
246	FANCY MOUSE	$50	1	$50
135	LASER PRINTER	$800	1	$800

TOTAL: $2,650

© University of Melbourne 2018

Our example (1NF)

- INVOICE (InvoiceNo, Date, CustomerNo, CustomerName, CustomerAddress, ClerkNo, ClerkName, (ProductNo, ProductDesc, UnitPrice, Qty))

- Repeating group is (ProductNo, ProductDesc, UnitPrice, Qty)
- The new table is as follows:
  - LINEITEM (*InvoiceNo*, ProductNo, ProductDesc, UnitPrice, Qty)
    (foreign keys get italics or dotted underline)
- The repeating fields will be removed from the original relation, leaving the following.
  - INVOICE (InvoiceNo, Date, CustomerNo, CustomerName, CustomerAddress, ClerkNo, ClerkName)
- These two relations together comprise a database in 1NF

© University of Melbourne 2018

Second normal form (2NF)

- Relevant only for relations whose primary key is a COMPOSITE key
- Remove partial functional dependencies.
  - an attribute is dependent on only *part of* the primary key
  - i.e. dependent on *one* of the columns in a composite primary key
- Create a separate table, containing the functionally dependent data, and the part of the key on which it depends.

© University of Melbourne 2018

Our example (2NF)

- INVOICE (InvoiceNo, Date, CustomerNo, CustomerName, CustomerAddress, ClerkNo, ClerkName)
- LINEITEM (*InvoiceNo*, ProductNo, ProductDesc, UnitPrice, Qty)

- INVOICE doesn't have a composite key, so it is already in 2NF
- LINEITEM is not in 2NF, because ProductDesc and UnitPrice are determined by ProductNo, which is *part of* the primary key. We need to create a new table from these fields

- The new table will contain the following fields:
  - PRODUCT (ProductNo, ProductDesc, UnitPrice)

© University of Melbourne 2018

Our example (2NF)

- All of these fields except the primary key will be removed from the original table. The primary key will be left in the original table to allow linking of data:
  - LINEITEM (*InvoiceNo*, *ProductNo*, Qty)

- Our database is now in second normal form:
  - INVOICE (InvoiceNo, Date, CustomerNo, CustomerName, CustomerAddress, ClerkNo, ClerkName)
  - LINEITEM (*InvoiceNo*, *ProductNo*, Qty)
  - PRODUCT (ProductNo, ProductDesc, UnitPrice)

© University of Melbourne 2018

Third normal form (3NF)

- Remove *transitive dependencies*
  - a functional dependency where one non-key attribute is functionally dependent on another non-key attribute
  - thus its value is only indirectly determined by the primary key.

- Create a separate table containing the determinant attribute and the fields that are functionally dependent on it.
  - keep a copy of the key in the original table.

© University of Melbourne 2018

Our example 3NF

- INVOICE (<u>InvoiceNo</u>, Date, CustomerNo, CustomerName, CustomerAddress, ClerkNo, ClerkName)
- LINEITEM (*<u>InvoiceNo</u>*, *<u>ProductNo</u>*, Qty)
- PRODUCT (<u>ProductNo</u>, ProductDesc, UnitPrice)

- LINEITEM and PRODUCT are in 3NF as there are not any transitive dependencies
  - All the attributes only depend on the key
- INVOICE is not in 3NF
  - Transitive dependencies exist
    - CustomerName and CustomerAddress depend on CustomerNo
    - ClerkName depends on ClerkNo

© University of Melbourne 2018

Our example 3NF

- The new tables will be:
  - CUSTOMER (<u>CustomerNo</u>, CustomerName, CustomerAddress)
  - CLERK (<u>ClerkNo</u>, ClerkName)
- All of these fields except the primary keys will be removed from the original table. The PKs will be left in the original table (as FKs) to allow linking of data, as follows:
  - INVOICE (<u>InvoiceNo</u>, Date, *CustomerNo*, *ClerkNo*)
- Together with the unchanged tables below, these tables make up the database in third normal form.
  - LINEITEM (<u>*InvoiceNo*</u>, <u>*ProductNo*</u>, Qty)
  - PRODUCT (<u>ProductNo</u>, ProductDesc, UnitPrice)

© University of Melbourne 2018

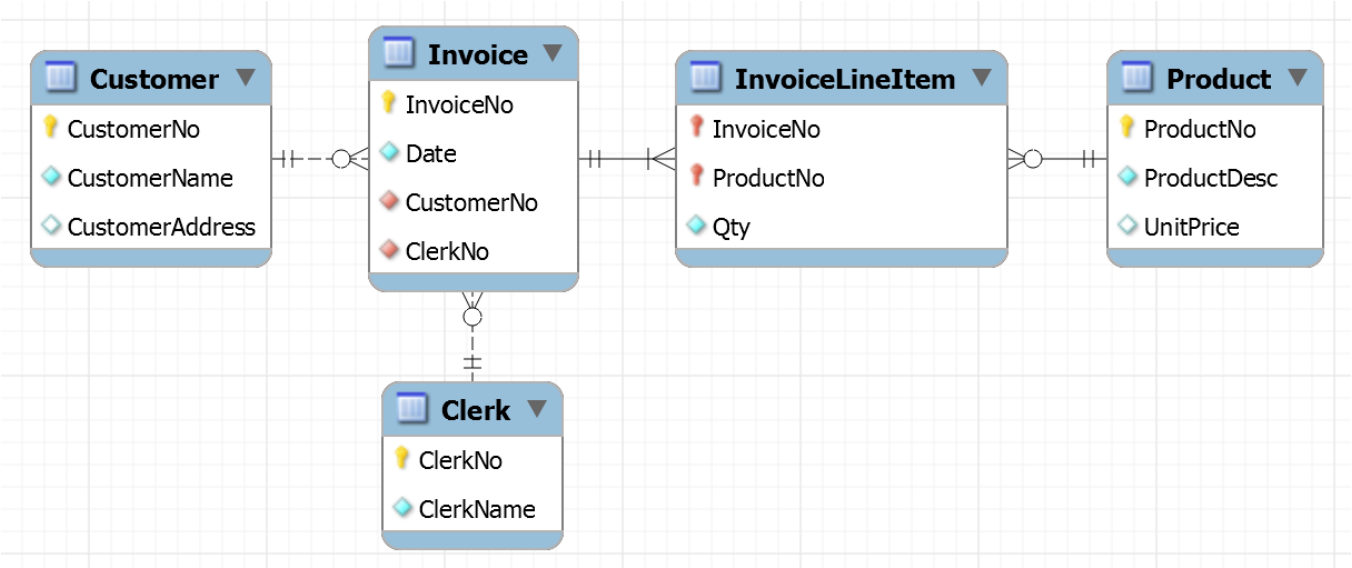
What if we did *not* normalize ?

- Repetition of data
  - Individual facts are stored many times.
- Update anomalies
  - To change a fact, we must change it many times.
- Deletion anomalies
  - Information about entity A is stored inside entity B.
    If we delete all B rows, we lose our record of A.
- Insertion anomalies
  - To record a new A, we must insert a B.

© University of Melbourne 2018

Our example schema in 3NF

- CUSTOMER (CustomerNo, CustomerName, CustomerAddress)
- CLERK (ClerkNo, ClerkName)
- PRODUCT (ProductNo, ProductDesc, UnitPrice)
- INVOICE (InvoiceNo, Date, *CustomerNo*, *ClerkNo*)
- LINEITEM (*InvoiceNo*, *ProductNo*, Qty)



© University of Melbourne 2018

Beyond 3NF

- It is often sufficient to stop normalizing at 3NF
- BCNF, 4NF, 5NF and even 6NF have been defined
  - we'll touch on BCNF
  - it is suggested that you read about the others, see
  - http://en.wikipedia.org/wiki/Database_normalization#Normal_forms
  - but we will not assess beyond 3NF

© University of Melbourne 2018

Boyce-Codd normal form (BCNF)

- Can arise when …
  - not every determinant in the relation is a candidate key

- A simple test is to check whether …
  - "Every non-key attribute must provide a fact about the key, the whole key, and nothing but the key."

© University of Melbourne 2018

BCNF example

- Consider the following relation
  - Student (StudentID, Major, Advisor, MajorGPA)
- It is in 3NF … but …
- Are there remaining, unidentified functional dependencies?

StudentID	Major	Advisor	MajorGPA
123	Physics	Hawking	4.0
123	Music	Mahler	3.3
456	Literature	Michener	3.2
745	Music	Bach	3.7
983	Physics	Hawking	3.5

© University of Melbourne 2018

Problems with this relation

- Update anomaly
  - Hawking is replaced by Einstein
- Insertion anomaly
  - Babbage advises comp. sci., but has no students
- Deletion anomaly
  - e.g. delete the only student with a particular advisor (456)

StudentID	Major	Advisor	MajorGPA
123	Physics	Hawking	4.0
123	Music	Mahler	3.3
456	Literature	Michener	3.2
745	Music	Bach	3.7
983	Physics	Hawking	3.5

© University of Melbourne 2018

Normalise to BCNF

- Modify the relation so that
  - the determinant in the relation that is not part of the key becomes a component of the primary key of the revised relation
  - the attribute that is functionally dependant on that determinant becomes a non key attribute
- The relation becomes
  - Student (<u>StudentID, Advisor</u>, Major, MajorGPA)
- Can this be normalized further?
  - Its not in 2NF (partial functional dependency) – so normalise it
  - Student (<u>StudentID, *Advisor*</u>, MajorGPA)
  - Advisor (<u>Advisor</u>, Major)

© University of Melbourne 2018

Now in BCNF

- Student (StudentID, *Advisor*, MajorGPA)
- Advisor (Advisor, Major)

STUDENT

SID	Advisor	MajGPA
123	Hawking	4.0
123	Mahler	3.3
456	Michener	3.2
789	Bach	3.7
678	Hawking	3.5

ADVISOR

Advisor	Major
Hawking	Physics
Mahler	Music
Michener	Literature
Bach	Music

© University of Melbourne 2018

Normalisation Exercise

Now it is your turn!

Normalization example

The Melbourne Multi-Campus School

STUDENT RECORD

Student Id: ___123456_____

Name: ___Joe Bloggs_____

Date of Birth: __1st April 1990_____

Campus: ____Fitzroy_____

Campus Address: ____123 Smith St, Fitzroy 3065_____

Subject Code	Subject Name	YearTaken	Result
MAT	Maths	2015	
PHY	Physics	2015	
ART	Art	2014	60
BIO	Biology	2014	90

Graduation Date: _____

- The *Melbourne Multi-Campus School* has been using a paper-based information system.
- Student data is recorded on these sheets of paper.
- Now they are going to computerize.
- How shall we store the data in a relational database?

© University of Melbourne 2018

Dr Greg Wadley

INFO90002
Database Systems & Information Modelling

Week 5

Physical Database Design

This lecture

- Physical database design
  - Estimating usage
  - Data types
  - Indexing
  - De-normalisation
- Aims of physical design
  - Translate the logical description of data into the technical specifications for storing and retrieving data on disk
  - Create a design for storing data that will provide good performance and insure database integrity, recoverability and security

© University of Melbourne 2018

Physical Design Process

Inputs

- Normalised data model
- Attribute definitions
- Volume estimates
- Response time expectations
- Data security needs
- Backup/recovery needs
- Integrity expectations
- DBMS used

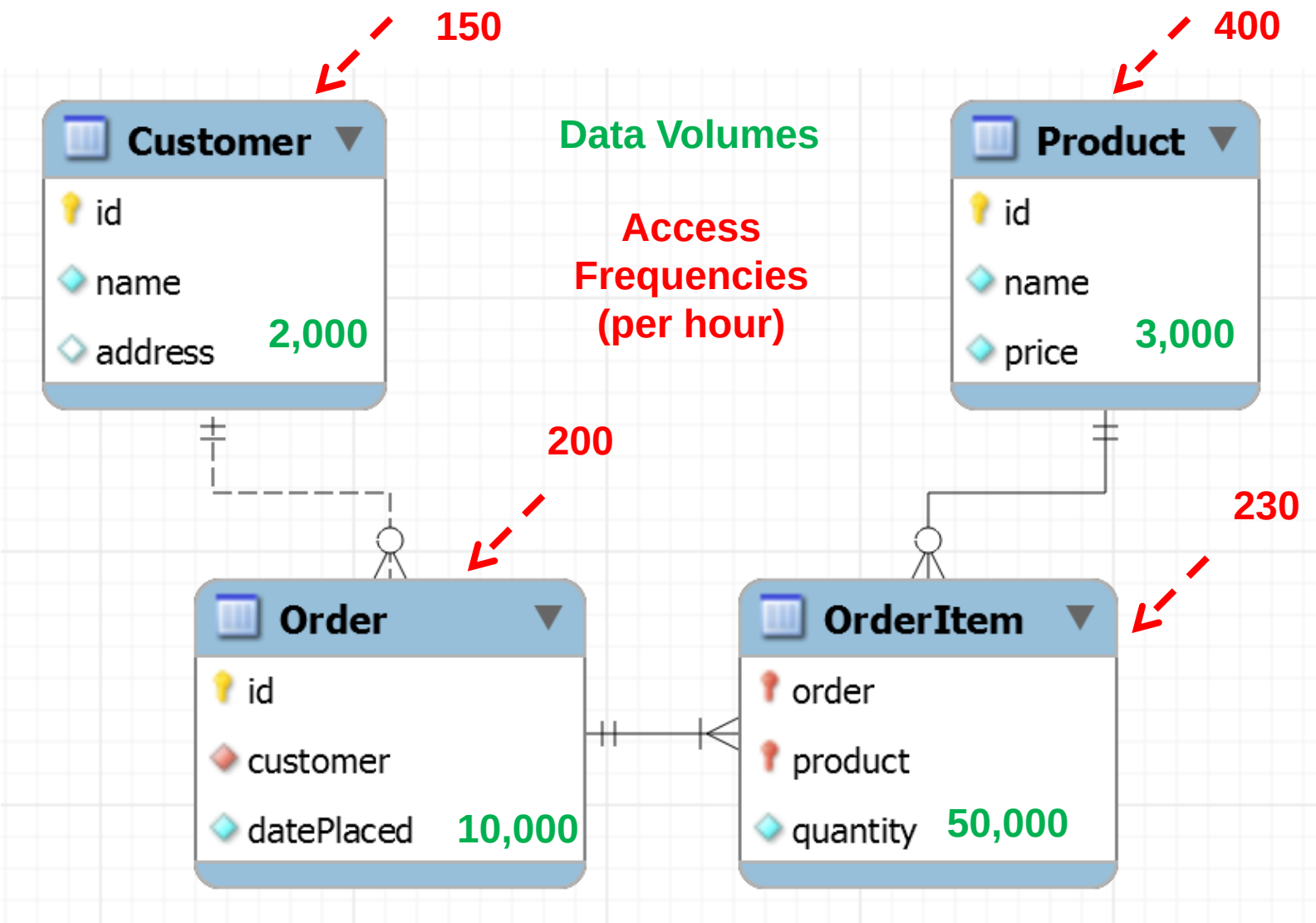
leads to

Decisions

- Attribute data types
- Physical record descriptions (doesn't always match logical design)
- File organisations
- Indexes
- Query optimisation

© University of Melbourne 2018

Estimating Usage



© University of Melbourne 2018

Choosing data types

- Column: smallest unit of data in database
- Data types help DBMS to store and use information efficiently
- You should choose data types that:
  - enforce data integrity (quality)
  - can represent all possible values
  - support all required data manipulations
  - minimize storage space
  - maximize performance (e.g. fixed or variable length)
- The major data types are:
  - text or character
  - number
  - dates and time

© University of Melbourne 2018

Character types (MySQL)

- **CHAR(M):** A fixed-length string that is always right-padded with spaces to the specified length when stored on disc. The range of M is 1 to 255.
- **CHAR:** Synonym for CHAR(1).
- **VARCHAR(M):** A variable-length string. Only the characters inserted are stored – no padding. The range of M is 1 to 65535 characters.
- **BLOB, TEXT:** A binary or text object with a maximum length of 65535 (2^16) bytes (blob) or characters (text). Not stored inline with row data.
- **LONGBLOB, LONGTEXT:** A BLOB or TEXT column with a maximum length of 4,294,967,295 (2^32 - 1) characters.
- **ENUM** ('value1','value2',…) up to 65,535 members.

© University of Melbourne 2018

Number types (MySQL)

- Integers
  - **TINYINT**: Signed (-128 to 127), Unsigned (0 to 255)
  - **BIT**, **BOOL:** synonyms for TINYINT
  - **SMALLINT**:
    Signed (-32,768 to 32,767), Unsigned (0 to 65,535 – 64k)
  - **MEDIUMINT**:
    Signed (-8388608 to 8388607), Unsigned (0 to 16777215 –16M)
  - **INT / INTEGER**:
    Signed (-2,147,483,648 to 2,147,483,647),
    Unsigned (0 to 4,294,967,295 – 4G or 2^32)
  - **BIGINT**:
    Signed (-9223372036854775808 to 9223372036854775807),
    Unsigned (0 to 18,446,744,073,709,551,615 - 2^64)
  - **Don't** use the "(M)" number for integers

© University of Melbourne 2018

Number types (MySQL)

- Real numbers (fractions)
  - **FLOAT**: single-precision floating point, allowable values: -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38.
  - **DOUBLE / REAL**: double-precision, allowable values: -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308.
  - optional M = number of digits stored, D = number of decimals.
  - Float and Double are often used for scientific data.
  - **DECIMAL**[(M[,D])]: fixed-point type. Good for money values.
  - M = precision (number of digits stored), D = number of decimals

© University of Melbourne 2018

Time types

- **DATE** 1000-01-01 to 9999-12-31
- **TIME** -838:59:59 to 838:59:59
  (time of day or elapsed time)
- **DATETIME** 1000-01-01 00:00:00 to
  9999-12-31 23:59:59
- **TIMESTAMP** 1970-01-01 00:00:00 - ~ 2037
  Stored in UTC, converted to local
- **YEAR** 1901 to 2155

How is DATETIME different to TIMESTAMP?

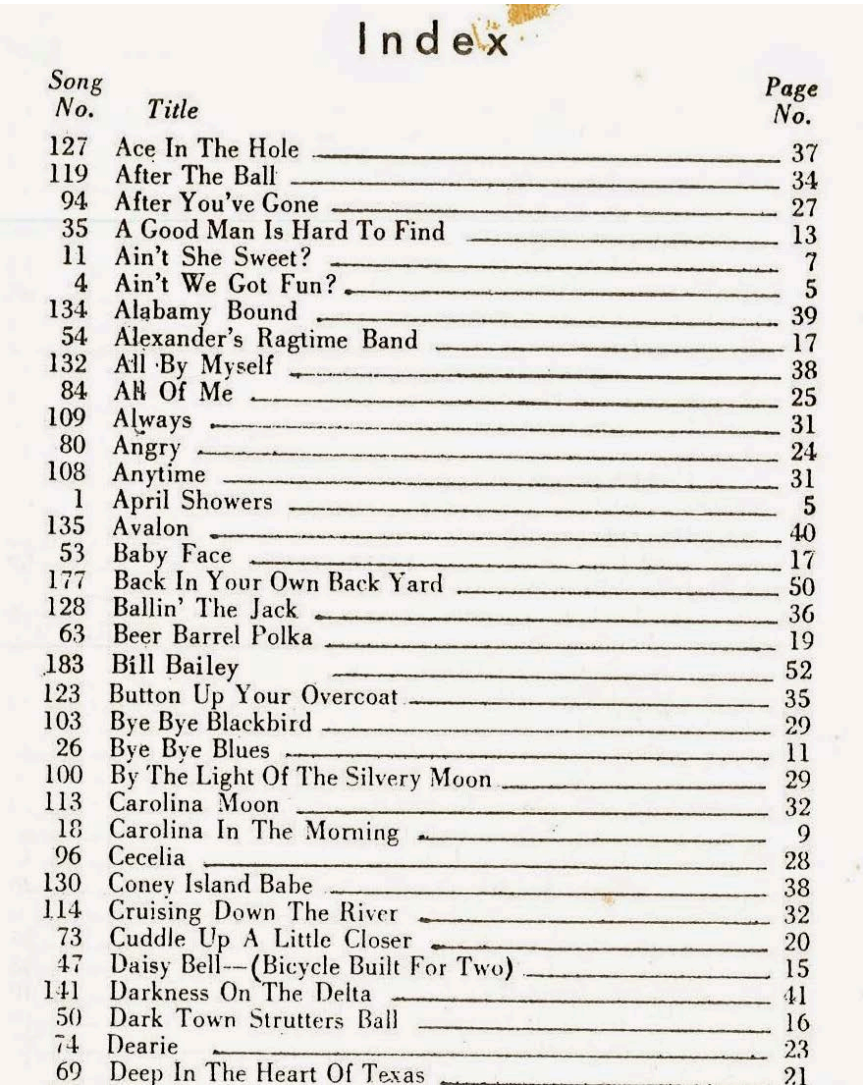
– TIMESTAMP is for automatically adding “now” to a row
– DATETIME is for referring to a time in the real world
– since version 5.6.4, both can store fractions of a second

© University of Melbourne 2018

Indexing columns

- Index – a separate file that contains pointers to table rows, to allow fast retrieval

- Similar to an index in a book



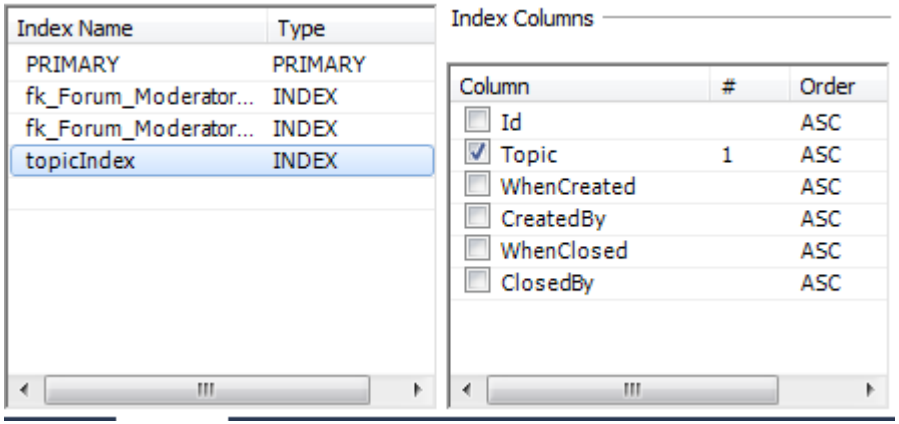
Index

Song No.	Title	Page No.
127	Ace In The Hole	37
119	After The Ball	34
94	After You've Gone	27
35	A Good Man Is Hard To Find	13
11	Ain't She Sweet?	7
4	Ain't We Got Fun?	5
134	Alabamy Bound	39
54	Alexander's Ragtime Band	17
132	All By Myself	38
84	All Of Me	25
109	Always	31
80	Angry	24
108	Anytime	31
1	April Showers	5
135	Avalon	40
53	Baby Face	17
177	Back In Your Own Back Yard	50
128	Ballin' The Jack	36
63	Beer Barrel Polka	19
183	Bill Bailey	52
123	Button Up Your Overcoat	35
103	Bye Bye Blackbird	29
26	Bye Bye Blues	11
100	By The Light Of The Silvery Moon	29
113	Carolina Moon	32
18	Carolina In The Morning	9
96	Cecelia	28
130	Coney Island Babe	38
114	Cruising Down The River	32
73	Cuddle Up A Little Closer	20
47	Daisy Bell--(Bicycle Built For Two)	15
141	Darkness On The Delta	41
50	Dark Town Strutters Ball	16
74	Dearie	23
69	Deep In The Heart Of Texas	21

© University of Melbourne 2018

Indexing columns

- You choose which columns to index
- PKs and FKs are automatically indexed
- Nominate columns to index when creating tables




```
CREATE TABLE IF NOT EXISTS `UniChat`.`Forum` (
  `Id` INT NOT NULL,
  `Topic` VARCHAR(100) NULL,
  `WhenCreated` TIMESTAMP NULL,
  `CreatedBy` INT NULL,
  `WhenClosed` TIMESTAMP NULL,
  `ClosedBy` INT NULL,
  PRIMARY KEY (`Id`),
  INDEX `fk_Forum_Moderator1_idx` (`CreatedBy` ASC),
  INDEX `fk_Forum_Moderator2_idx` (`ClosedBy` ASC),
  INDEX `topicIndex` (`Topic` ASC),
  CONSTRAINT `fk_Forum_Moderator1`
    FOREIGN KEY (`CreatedBy`)
    REFERENCES `UniChat`.`Lecturer` (`Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Forum_Moderator2`
    FOREIGN KEY (`ClosedBy`)
    REFERENCES `UniChat`.`Lecturer` (`Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```


© University of Melbourne 2018

When to use Indexes

- use on larger tables
- on columns which are frequently in WHERE clauses
- or in ORDER BY and GROUP BY clauses
- if column has >100 distinct values but not if <30 values

Id	Title	Classification	ReleaseYear	Genre	CostWholesale	PricePerView
1	Titanic	PG	1999	1	5.02	0.67
2	The Lord of the Ring...	PG	2004	2	1.31	0.58
3	Pirates of the Carib...	PG	2006	2	1.49	1.40
4	Harry Potter and the...	PG	2002	3	3.50	0.75
5	Pirates of the Carib...	PG	2007	2	3.06	1.04
6	Harry Potter and the...	PG	2007	4	4.78	1.47
7	Star Wars: Episode I...	PG	2001	5	4.71	0.72
8	The Lord of the Ring...	PG	2003	3	9.23	0.67
9	Jurassic Park	PG	2001	6	2.02	0.72
10	Harry Potter and the...	PG	2006	4	2.41	1.09
11	Spider-Man 3	PG	2007	2	6.00	0.79

lm 2 ×

© University of Melbourne 2018

When *NOT* to use Indexes

- Limit the use of indexes for *volatile* databases
  - Why?
  - "Volatile" = data are frequently changed
  - When table data are changed
    (e.g. by inserts, deletes, updates)
    indexes need to be updated

© University of Melbourne 2018

De-normalisation

- Normalisation
  - removes data redundancy
  - solves Insert, Update and Delete anomalies
  - makes it easier to maintain information in a consistent state
- However
  - it leads to more tables in the database
  - typically these need to be joined during Selects, which is expensive to do
  - sometimes we decide to ‘de-normalize’

© University of Melbourne 2018

Denormalization example: Look-up table

PRODUCT File

Product_No	Description	Finish	...
B100	Chair	C	
B120	Desk	A	
M128	Table	C	
T100	Bookcase	B	
...	...	...	

FINISH Look-up Table

Code	Value
A	Birch
B	Maple
C	Oak

Lookup table saves space and improves data consistency, but costs an additional read to obtain actual value

© University of Melbourne 2018

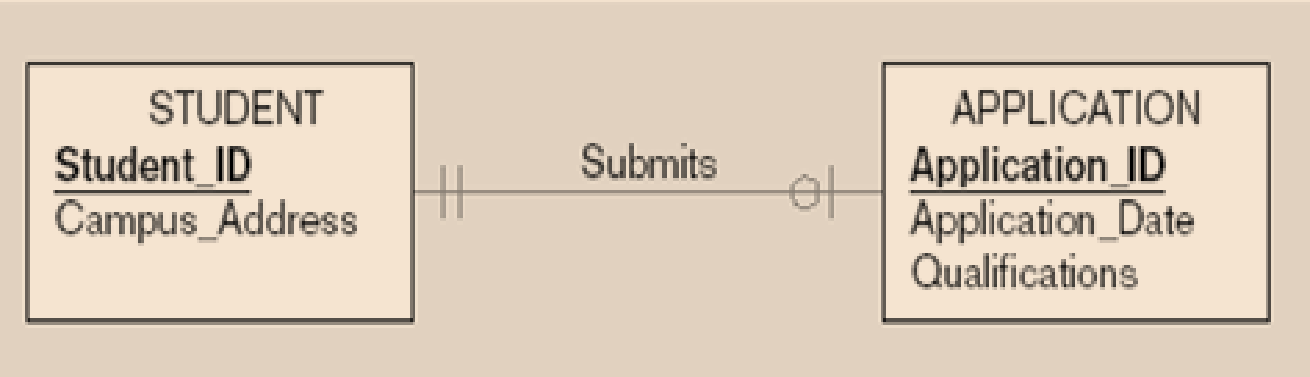
Denormalisation opportunities

- You might want to de-normalise if
  - database speeds are unacceptable
  - there are going to be very few INSERTs, UPDATEs, or DELETEs
  - there are going to be lots of SELECTs that involve the joining of tables

- Examples
  - one-to-one relationship
  - many-to-many relationship with attributes
  - reference data / lookup table
    (1:N relationship where data on 1-side not used in any other relationship)

© University of Melbourne 2018

One-to-One



Normalized relations:

STUDENT

Student_ID	Campus_Address

APPLICATION

Application_ID	Application_Date	Qualifications	Student_ID

Denormalized relation:

STUDENT

Student_ID	Campus_Address	Application_Date	Qualifications

and Application_Date and Qualifications may be null

© University of Melbourne 2018

Many-to-Many with non-key attributes

Normalized relations:

VENDOR

Vendor_ID	Address	Contact_Name

ITEM

Item_ID	Description

PRICE QUOTE

Vendor_ID	Item_ID	Price

Extra Table Access

Denormalized relations:

VENDOR

Vendor_ID	Address	Contact_Name

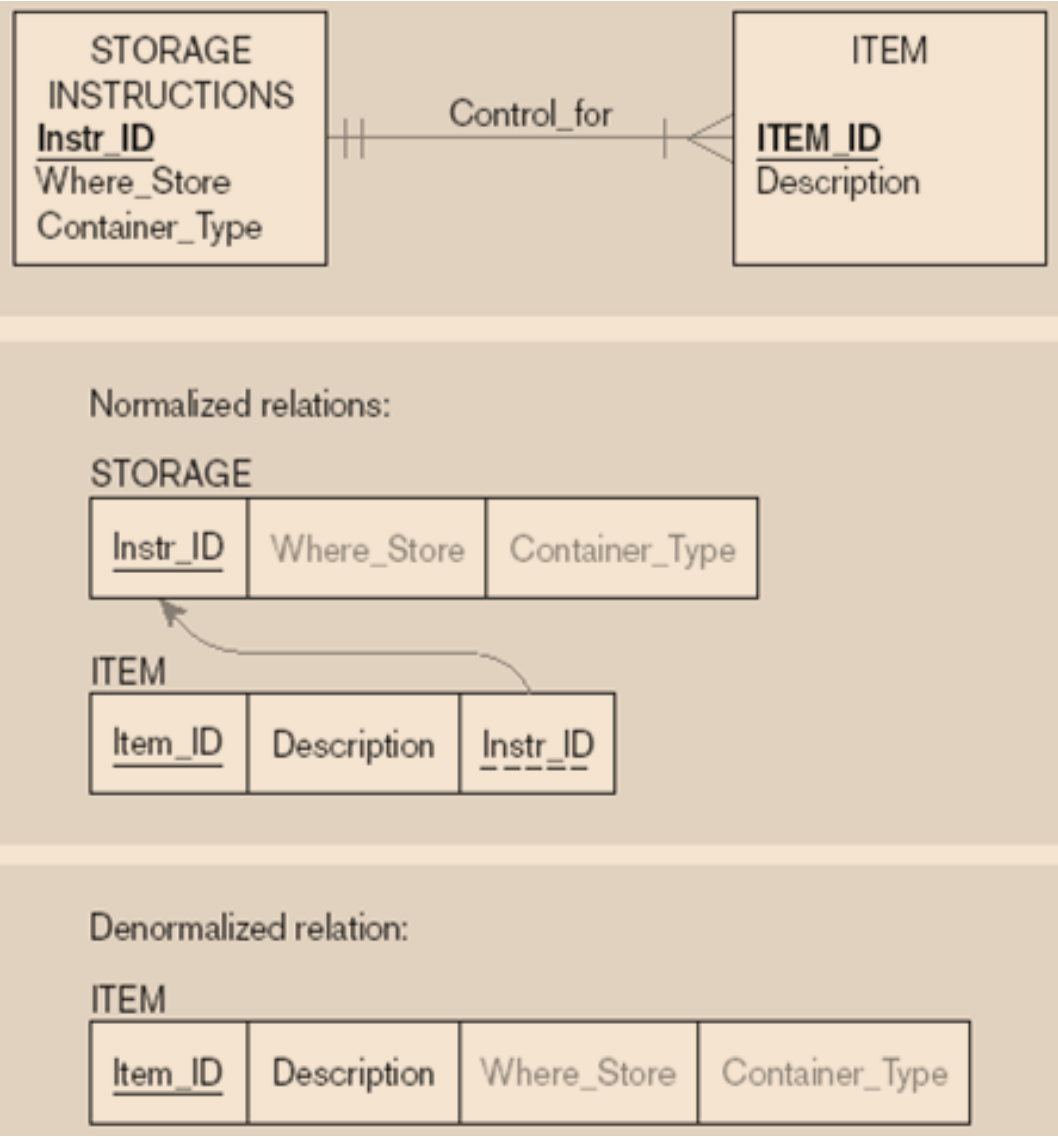
ITEM QUOTE

Vendor_ID	Item_ID	Description	Price

Same Item – Many Descriptions

© University of Melbourne 2018

Reference Data



extra table access required

data duplication

© University of Melbourne 2018

Next Week

- Next week we finish learning about data modelling and SQL.
- After that we begin the second half of the subject.
- In week 7 we explore how databases are embedded within larger applications, in particular, web apps

© University of Melbourne 2018

Dr Greg Wadley

THE UNIVERSITY OF MELBOURNE

INFO90002 Database Systems & Information Modelling

Week 06

SQL 2

Today's Session…

- Nested/Sub queries
- DML
  - Comparison & Logic Operators, Set Operations, Multiple record INSERTs, INSERT from a table, UPDATE, DELETE, REPLACE
- DDL
  - ALTER and DROP, TRUNCATE, RENAME
- DCL
  - GRANT and REVOKE
- Views

© University of Melbourne 2018

SQL – Subqueries / nested queries

- Select allows you to nest *sub-queries* inside the main or "outer" query
- A nested query is simply another Select query you write to produce a table of data
  - remember that all select queries return a "table"
- A common use of sub-queries is to perform tests
  - set membership, set comparisons
- Often there is an equivalent Join query
- Put the subquery inside round brackets

```
SELECT DISTINCT saleId FROM Sale
WHERE departmentid IN
    (SELECT departmentId FROM Department
    WHERE floor = 2);
```

(3, 7)

© University of Melbourne 2018

Comparison Operators for SubQueries

- IN / NOT IN
  - is the value a member of the set returned by the Subquery?
- ALL
  - true if all values returned meet the condition
- WHERE [NOT] EXISTS
  - true if the subquery yields any [/ no] results

© University of Melbourne 2018

Set Comparison examples

- auction example: Buyer, Seller, Artefact, Offer tables

ID	Name	Description
1	Vase	Old Vase
2	Knife	Old Knife
3	Pot	Old Pot

SellerID	Name	Phone
1	Abby	0233232232
2	Ben	0311111111
3	Carl	0333333333

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444
3	Oleg	0555555555

SellerID	ArtefactID	BuyerID	Date	Amount	Acceptance
1	1	1	2012-06-20	81223.23	N
1	1	2	2012-06-20	82223.23	N
2	2	1	2012-06-20	19.95	N
2	2	2	2012-06-20	23.00	N

© University of Melbourne 2018

Set queries

- which Artefacts *don't have* offers made on them?

```
SELECT * FROM Artefact
    WHERE ID NOT IN
        (SELECT ArtefactID FROM Offer);
```

ID	Name	Description
3	Pot	Old Pot

- which Buyers *haven't* made a bid for Artefact 3?

```
SELECT * FROM Buyer
    WHERE BuyerID NOT IN
        (SELECT BuyerID FROM Offer
            WHERE ArtefactID = 3);
```

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444
3	Oleg	0555555555

- which Buyers *haven't* made a bid for the "Pot" Artefact?

```
SELECT * FROM Buyer
    WHERE BuyerID NOT IN
        (SELECT BuyerID FROM Offer
            WHERE ArtefactID IN
                (SELECT ID FROM Artefact
                    WHERE Name = "Pot"));
```

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444
3	Oleg	0555555555

© University of Melbourne 2018

Multiple subquery

- which Buyers have made a bid for the “Knife” Artefact?

```
SELECT * FROM Buyer
    WHERE BuyerID IN
        (SELECT BuyerID FROM Offer
            WHERE ArtefactID IN
                (SELECT ID FROM Artefact
                    WHERE Name = "Knife"));
```

BuyerID	Name	Phone
1	Maggie	0333333333
2	Nicole	0444444444

There is often an equivalent Join that will achieve the same result. The above is equivalent to:

```
SELECT Buyer.*
FROM Buyer NATURAL JOIN Offer NATURAL JOIN Artefact
WHERE Artefact.name = 'Knife' ;
```


© University of Melbourne 2018

Aggregate functions

These functions operate on a set of values (e.g. in a column of a table) and return a single value

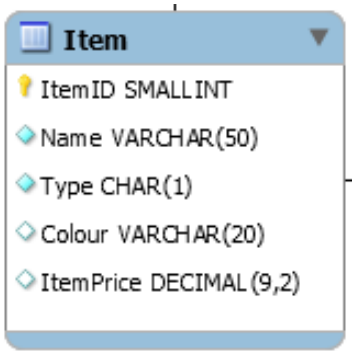
- AVG()
  - Average value
- MIN()
  - Minimum value
- MAX()
  - Maximum value
- COUNT()
  - Number of values
- SUM()
  - Sum of values

- and there are others …
  - http://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html
- These ignore null values, and return null if all values are null.
- But COUNT(*) counts the rows not the values,
  and thus even if the value is NULL it is still counted.

© University of Melbourne 2018

Finding above average

- Consider the Item table in our labs database
- Which items have a price that is higher than the average?




```
SELECT * FROM Item
WHERE itemPrice >
    (SELECT AVG(itemPrice) FROM Item);
```

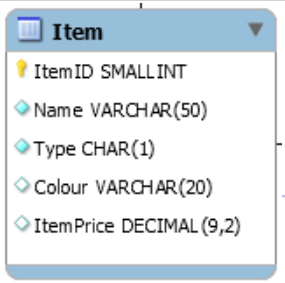
234.766400

itemID	Name	Type	Colour	itemPrice
1	Boots Riding	C	Brown	235.00
2	Horse saddle	R	Brown	1895.00
12	Gortex Rain Coat	C	Green	249.75
19	Tent - 2 person	F	Khaki	399.95
20	Tent - 8 person	F	Khaki	785.96
21	Tent - 4 person	F	Blue	638.95
24	Boots - Womens Goretex	C	Grey	289.95
25	Boots - Mens Hiking	C	Grey	299.95

© University of Melbourne 2018

Finding a maximum

- Which item has the highest cost?




```
SELECT * FROM Item
WHERE itemPrice =
    (SELECT MAX(itemPrice) FROM Item);
```

1,895

itemID	Name	Type	Colour	itemPrice
2	Horse saddle	R	Brown	1895.00
NULL	NULL	NULL	NULL	NULL

```
SELECT * FROM Item
ORDER BY itemprice DESC
LIMIT 1;
```

- Will these two methods always give the same answer?

© University of Melbourne 2018

Finding a maximum

- another method

```
SELECT * FROM Item
WHERE itemPrice >= ALL
    (SELECT itemPrice FROM Item);
```

- and another: a “correlated subquery”

```
SELECT * FROM Item A
WHERE itemPrice > ALL
    (SELECT itemPrice FROM Item B
    WHERE A.itemId != B.itemId);
```


© University of Melbourne 2018

Case and calculation

- SQL keywords are *not* case-sensitive.
  - the traditional convention is to CAPITALISE them for clarity
- Table names *are* case sensitive in Unix, but not Windows
  (and possibly *not* case-sensitive if you use the InnoDb storage engine)
  - Account <> account <> ACCOUNT (in Unix)
- Column names are *not* case-sensitive
  - ACCOUNTID == AccountID == AcCoUnTID
- Case-sensitivity of DATA ('strings in quotes') depends on character set used.
  (The default 'latin1' set is *not* case-sensitive.)
- SQL handles expressions including maths:
  - SELECT 1*2+3/4-5;
  - SELECT now();

© University of Melbourne 2018

Comparison and Logic Operators

- Comparison

Operator	Description
=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<> OR !=	Not equal to

- Logic
  - SQL supports AND, NOT, OR logical operators
    - SELECT * FROM Furniture
      WHERE ((Type= ‘Chair’ AND Colour = ‘Black’)
      OR NOT (Type = ‘Lamp’ AND Colour = ‘White’));

© University of Melbourne 2018

Set Operations

- We can combine results from two or more queries that return *the same number of columns* - although it usually only makes sense if they are the *same columns*.
- UNION
  - Show all rows returned from the queries, without duplicates
- INTERSECT
  - Show only rows that are common in the queries
- EXCEPT
  - Show only rows that are different in the queries
- [UNION/INTERSECT/EXCEPT] ALL
  - If you want duplicate rows shown in the results you need to use the ALL keyword, e.g. UNION ALL.

- In MySQL only UNION and UNION ALL are supported

© University of Melbourne 2018

UNION example

```
SELECT * FROM Department
WHERE floor = 1
UNION
SELECT * FROM Department
WHERE floor = 3;
```

DepartmentID	DepartmentName	DepartmentFloor	DepartmentPhone	ManagerID
6	Navigation	1	41	3
8	Books	1	81	4
4	Equipment	3	57	3
NULL	NULL	NULL	NULL	NULL

(what if the subsets overlap?)

© University of Melbourne 2018

Formatting the result

- FORMAT()
  - changes format of output of Select
  - e.g. FORMAT (N, D)
    - N: A number which may be an integer, a decimal or a float.
    - D: How many decimals the output contains
    - FORMAT(123456.1234, 2) gives ‘123,456.12’
- CAST()
  - changes data type of output
  - e.g. CAST (Expression AS Type)
    - CAST(“1234.55” AS UNSIGNED) Gives 1235
    - CAST("1234.55" AS DECIMAL(7,1)) Gives 1234.6
    - Valid types include
      - BINARY[(N)], CHAR[(N)], DATE, DATETIME, DECIMAL[(M[,D])], SIGNED, TIME, UNSIGNED

© University of Melbourne 2018

Output without Format

```
SELECT Department.DepartmentID, SUM(EmployeeSalary*Bonus) AS TotSalary
FROM Department INNER JOIN Employee ON Department.DepartmentID = Employee.DepartmentID
GROUP BY Department.DepartmentID;
```

DepartmentID	TotSalary
1	67499.9982118607
2	60000
3	32639.9993896484
4	27039.9990081787
5	15000
6	15000
7	16500.0003576279
8	15149.9998569489
9	99000
10	35000
11	101200.002193451

messy

© University of Melbourne 2018

Formatting output (Format)

```
SELECT Department.DepartmentID, FORMAT(SUM(EmployeeSalary*Bonus),2) AS TotSalary
FROM Department INNER JOIN Employee ON Department.DepartmentID = Employee.DepartmentID
GROUP BY Department.DepartmentID;
```

DepartmentID	TotSalary
1	67,500.00
2	60,000.00
3	32,640.00
4	27,040.00
5	15,000.00
6	15,000.00
7	16,500.00
8	15,150.00
9	99,000.00
10	35,000.00
11	101,200.00

but Format() converts numbers to strings …

what happens now if we sort by TotSalary?

© University of Melbourne 2018

Formatting output (Format)

```
SELECT Department.DepartmentID, FORMAT(SUM(EmployeeSalary*Bonus),2) AS TotSalary
FROM Department INNER JOIN Employee ON Department.DepartmentID = Employee.DepartmentID
GROUP BY Department.DepartmentID
ORDER BY TotSalary DESC;
```

DepartmentID	TotSalary
9	99,000.00
1	67,500.00
2	60,000.00
10	35,000.00
3	32,640.00
4	27,040.00
7	16,500.00
8	15,150.00
6	15,000.00
5	15,000.00
11	101,200.00

wrong

© University of Melbourne 2018

Formatting output (Cast)

```
SELECT Department.DepartmentID, CAST(SUM(EmployeeSalary*Bonus) AS DECIMAL(9,2)) AS TotSalary
FROM Department INNER JOIN Employee ON Department.DepartmentID = Employee.DepartmentID
GROUP BY Department.DepartmentID
ORDER BY TotSalary DESC;
```

DepartmentID	TotSalary
11	101200.00
9	99000.00
1	67500.00
2	60000.00
10	35000.00
3	32640.00
4	27040.00
7	16500.00
8	15150.00
6	15000.00
5	15000.00

These are numbers, so ordering works again

© University of Melbourne 2018

Other useful functions

- IFNULL()
  - Can convert a null to a zero (can be useful in calculations)
    - SELECT 1 + IFNULL(wagevalue, 0)
    - gives 1+0 for null fields, and 1+wagevalue for non null fields
    - failure to do this results in a NULL answer for values where wagevalue is NULL

(example on next two slides)

© University of Melbourne 2018

Don’t want NULLs in output?

```
SELECT e.ID, e.Name, e.Address, DateHired, DateLeft,
       EmployeeType, ContractNumber, BillingRate,
       AnnualSalary, StockOption, HourlyRate
  FROM Employee e
  LEFT OUTER JOIN Hourly h ON e.ID = h.ID
  LEFT OUTER JOIN Salaried s ON e.ID = s.ID
  LEFT OUTER JOIN Consultant c ON e.ID = c.ID;
```

ID	Name	Address	DateHired	DateLeft	EmployeeType	ContractNumber	BillingRate	AnnualSalary	StockOption	HourlyRate
1	Sean	Sean's Address	2012-02-02	NULL	S	NULL	NULL	92000.00	N	NULL
2	Linda	Linda's Address	2011-06-12	NULL	S	NULL	NULL	92300.00	Y	NULL
3	Alice	Alice's Address	2012-12-02	NULL	H	NULL	NULL	NULL	NULL	23.43
4	Alan	Alan's Address	2010-01-22	NULL	H	NULL	NULL	NULL	NULL	29.43
5	Peter	Peter's Address	2010-09-07	NULL	C	19223	210.00	NULL	NULL	NULL
6	Rich	Rich's Address	2012-05-19	NULL	C	19220	420.00	NULL	NULL	NULL

© University of Melbourne 2018

Nulls replaced with zeros

```
SELECT e.ID, e.Name, e.Address, DateHired,
    EmployeeType, IFNULL(ContractNumber,0) ContractNbr,
    IFNULL(BillingRate,0) BillRate, IFNULL(AnnualSalary,0) Salary,
    IFNULL(StockOption,"") StockOpt, IFNULL(HourlyRate,0) HrlyRate
    FROM Employee e
    LEFT OUTER JOIN Hourly h ON e.ID = h.ID
    LEFT OUTER JOIN Salaried s ON e.ID = s.ID
    LEFT OUTER JOIN Consultant c ON e.ID = c.ID;
```

ID	Name	Address	DateHired	EmployeeType	ContractNbr	BillRate	Salary	StockOpt	HrlyRate
1	Sean	Sean's Address	2012-02-02	S	0	0.00	92000.00	N	0.00
2	Linda	Linda's Address	2011-06-12	S	0	0.00	92300.00	Y	0.00
3	Alice	Alice's Address	2012-12-02	H	0	0.00	0.00		23.43
4	Alan	Alan's Address	2010-01-22	H	0	0.00	0.00		29.43
5	Peter	Peter's Address	2010-09-07	C	19223	210.00	0.00		0.00
6	Rich	Rich's Address	2012-05-19	C	19220	420.00	0.00		0.00

© University of Melbourne 2018

Other useful functions

- LOWER() / UPPER()
  - Change string to lower / upper case
    - e.g. SELECT LOWER(‘That’) gives ‘that’
    - SELECT UPPER(‘That’) gives ‘THAT’
- LEFT() / RIGHT()
  - Returns the leftmost / rightmost N characters from a string
    - e.g. SELECT LEFT(‘This is a test’, 6) gives “This i”
    - e.g. SELECT RIGHT(‘This is a test’, 6) gives “a test”
- Date and time functions
  - http://dev.mysql.com/doc/refman/5.5/en/date-and-time-functions.html
    - including DATEDIFF(), TIMEDIFF(), NOW() or TIMESTAMP(), CURDATE(), CURTIME()

© University of Melbourne 2018

More on INSERT

- Inserting records from another table
  - Note: table must already exist

```
INSERT INTO NewEmployee
    SELECT * FROM Employee;
```

- Insert multiple rows

```
INSERT INTO Employee VALUES
    (DEFAULT, "A", "A's Addr", "2012-02-02", NULL, "S"),
    (DEFAULT, "B", "B's Addr", "2012-02-02", NULL, "S"),
    (DEFAULT, "C", "C's Addr", "2012-02-02", NULL, "S");
```

```
INSERT INTO Employee
    (Name, Address, DateHired, EmployeeType)
    VALUES
        ("D", "D's Addr", "2012-02-02", "C"),
        ("E", "E's Addr", "2012-02-02", "C"),
        ("F", "F's Addr", "2012-02-02", "C");
```


© University of Melbourne 2018

More on UPDATE

- Be careful to specify a WHERE clause
  - unless you want it to operate on EVERY row in the table

```
UPDATE Hourly
    SET HourlyRate = HourlyRate * 1.10;
```

- Increase salaries greater than $100k by 10% and all other salaries by 5%

```
UPDATE Salaried
    SET AnnualSalary = AnnualSalary * 1.05
    WHERE AnnualSalary <= 100000;
UPDATE Salaried
    SET AnnualSalary = AnnualSalary * 1.10
    WHERE AnnualSalary > 100000;
```

- Any problems with this?

© University of Melbourne 2018

Flow Control using CASE

- A better solution is to use the CASE expression

```
UPDATE Salaried
    SET AnnualSalary =
        CASE
            WHEN AnnualSalary <= 100000
            THEN AnnualSalary * 1.05
            ELSE AnnualSalary * 1.10
        END;
```

- now we process each row independently, one at a time

© University of Melbourne 2018

Flow Control using CASE

- CASE can also be used in SELECT statements
- e.g “Calculate our annual bonuses. Give each employee a 10% bonus, except those who work work in Clothes or Books, who get 20%.”

```sql
SELECT employeeId, lastName, departmentId, salary,
CASE
    WHEN departmentId in
        (SELECT departmentId FROM Department WHERE name in ('clothes', 'books'))
    THEN salary * 0.2
    ELSE salary * 0.1
END as bonus
FROM employee
ORDER BY departmentid;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employeeId	lastName	departmentId	salary	bonus
1	Munro	1	125000.00	12500.000
11	Skeeter	2	45000.00	9000.000
13	Smith	3	46000.00	9200.000
12	Montez	3	46000.00	9200.000
15	Mason	4	45000.00	4500.000
14	Innit	4	41000.00	4100.000

Result 11

© University of Melbourne 2018

Yes/No Questions using CASE

- You can use CASE to answer yes/no or true/false questions.
- e.g “Are there more than ten customers?”

```
/* Are there more than ten customers? */
SELECT
CASE
    WHEN COUNT(*) > 10
    THEN 'yes'
    ELSE 'no'
END as answer
FROM Customer;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

answer
no

```
/* General true/false question */
SELECT
CASE
    WHEN 1 = 2
    THEN 'true'
    ELSE 'false'
END as answer
;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

answer
false

© University of Melbourne 2018

DELETE, REPLACE

- REPLACE
  - REPLACE works the same as INSERT
    - EXCEPT that if an old row in a table has a key value the same as the new row, then it is overwritten…
- DELETE
  - be careful to use a WHERE clause … What does this do?

```
DELETE FROM Employee;
```

    - Usually you should do use a filter:

```
DELETE FROM Employee
    WHERE Name = "Grace";
```

  - If you delete a row that has rows in other tables dependent on it, either:
    - the dependent rows are deleted too, or
    - the dependent rows get ‘null’ or a default, or
    - your attempt to delete is blocked
    - you decide what action to take when you set up the tables
      - ON DELETE CASCADE or ON DELETE RESTRICT…

© University of Melbourne 2018

Views

- a View is a select statement that persists, and can be treated as though it were a table by other SQL statements
- Used to:
  - hide the complexity of queries from users
  - hide structure of data from users
  - hide data from users
    - different users use different views
      - e.g. allow someone to access employee table, but not salaries column
    - one way of improving database security
- To create a view…
  - **CREATE VIEW** nameofview **AS** validSelectStatement
  - its definition (but not its output) is stored in the database
  - can be used as though it is a table

© University of Melbourne 2018

CREATE VIEW example

```sql
CREATE VIEW DepartmentSales AS
SELECT departmentId, name, COUNT(*) as numSales
FROM Department NATURAL JOIN Sale
GROUP BY departmentId;
```

```sql
SELECT * FROM DepartmentSales;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

departmentId	name	numSales
2	Books	6
3	Clothes	8
4	Equipment	6
5	Furniture	4
6	Navigation	13
7	Recreation	6

```sql
SELECT * FROM DepartmentSales
WHERE numSales > 5;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

departmentId	name	numSales
2	Books	6
3	Clothes	8
4	Equipment	6
6	Navigation	13
7	Recreation	6

© University of Melbourne 2018

When can we Update or Insert a view?

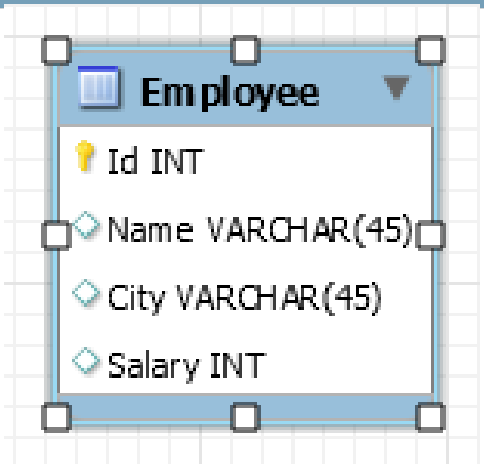
- Conditions that must be satisfied:
  - the select clause only contains attribute names
    - not expressions, aggregates or distinct
  - any attributes not listed in the select clause can be set to null
  - the query does not have a group by or having clause

- MySQL conditions for updatable views are quite stringent
  - see http://dev.mysql.com/doc/refman/5.0/en/view-updatability.html

© University of Melbourne 2018

Updating a View: example

Underlying base table ->



Id	Name	City	Salary
1	John Lennon	Sydney	100000
2	Paul McCartney	Melbourne	80000
3	George Harrison	Melbourne	90000
4	Ringo Starr	Brisbane	110000
NULL	NULL	NULL	NULL

```
CREATE VIEW MelbRestricted AS
(SELECT id, name, city from Employee
WHERE city = 'Melbourne');
```

Id	Name	City
2	Paul McCartney	Melbourne
3	George Harrison	Melbourne

```
SELECT * FROM MelbRestricted;
```

```
INSERT INTO MelbRestricted VALUES
(null, 'Yoko Ono', 'Melbourne');
```

Id	Name	City	Salary
1	John Lennon	Sydney	100000
2	Paul McCartney	Melbourne	80000
3	George Harrison	Melbourne	90000
4	Ringo Starr	Brisbane	110000
6	Yoko Ono	Melbourne	NULL
NULL	NULL	NULL	NULL

© University of Melbourne 2018

More DDL commands

- (beyond CREATE)
- ALTER
  - Allows us to add or remove columns from a table
    - ALTER TABLE TableName ADD AttributeName AttributeType
    - ALTER TABLE TableName DROP AttributeName
      - not supported by all vendors (MySQL supports it)
- RENAME
  - Allows the renaming of tables
    - RENAME TABLE CurrentTableName TO NewTableName

© University of Melbourne 2018

More DDL commands

- TRUNCATE
  - like “DELETE FROM table” but it does more
  - differences are vendor-specific, see http://stackoverflow.com/questions/139630/whats-the-difference-between-truncate-and-delete-in-sql and https://dev.mysql.com/doc/refman/5.0/en/truncate-table.html
  - in MySQL, resets auto_increment PKs
  - cannot ROLL BACK a TRUNCATE command
    - have to get data back from backup…
- DROP
  - potentially DANGEROUS
    - Removes the table definition and the data in the table
      - There is NO UNDO COMMAND! (have to restore from backup)
    - DROP TABLE TableName

© University of Melbourne 2018

Data Control Language / Other Commands

- DCL
  - Users and permissions
    - **CREATE USER, DROP USER**
    - **GRANT, REVOKE**
    - **SET PASSWORD**
- Other commands offered
  - Database administration
    - **BACKUP TABLE, RESTORE TABLE**
    - **ANALYZE TABLE**
  - Miscellaneous
    - **DESCRIBE tablename**
    - **USE db_name**
  - MySql calls these ‘Database Administration Statements’

© University of Melbourne 2018

SQL Language in summary

- Data Definition Language (DDL)
  - To define and set up the database
  - CREATE, ALTER, DROP
    - Also TRUNCATE, RENAME
- Data Manipulation Language (DML)
  - To maintain and use the database
  - SELECT, INSERT, DELETE, UPDATE
    - MySQL also provides others…. eg REPLACE
- Data Control Language (DCL)
  - To control access to the database
    - GRANT, REVOKE
- Other Commands
  - Administer the database
  - Transaction Control

© University of Melbourne 2018

Dr Greg Wadley

THE UNIVERSITY OF MELBOURNE

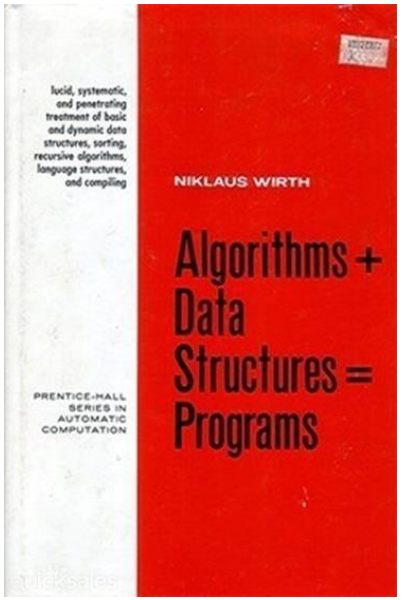
INFO90002 Database Systems & Information Modelling

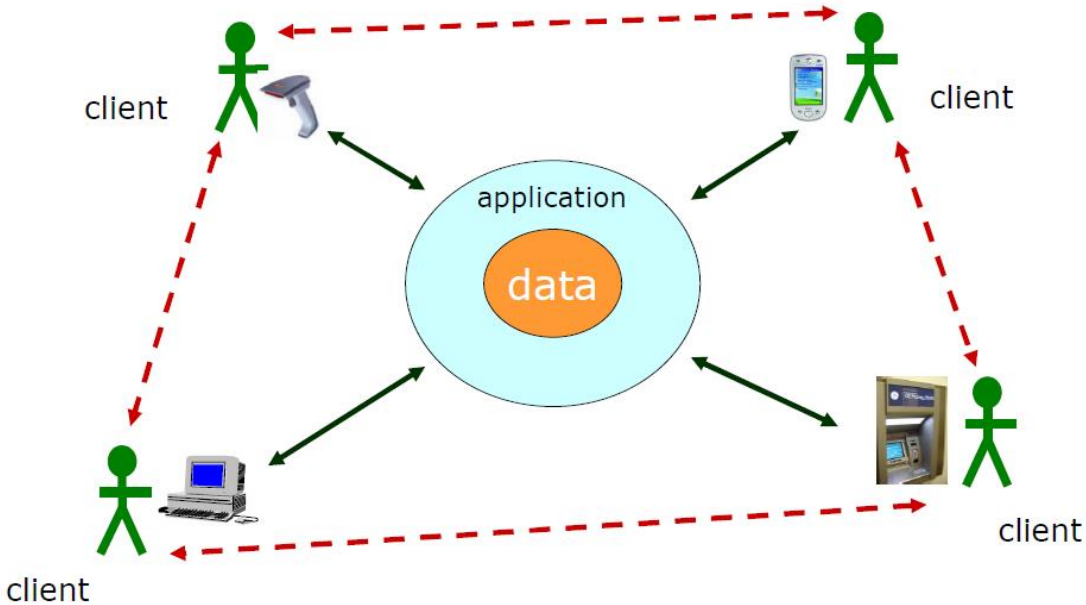
Week 07

Databases in Applications

Today's Session…

- How end-users access the database
- Business logic
- Stored procedures and triggers
- Embedding databases inside applications
- Application architectures

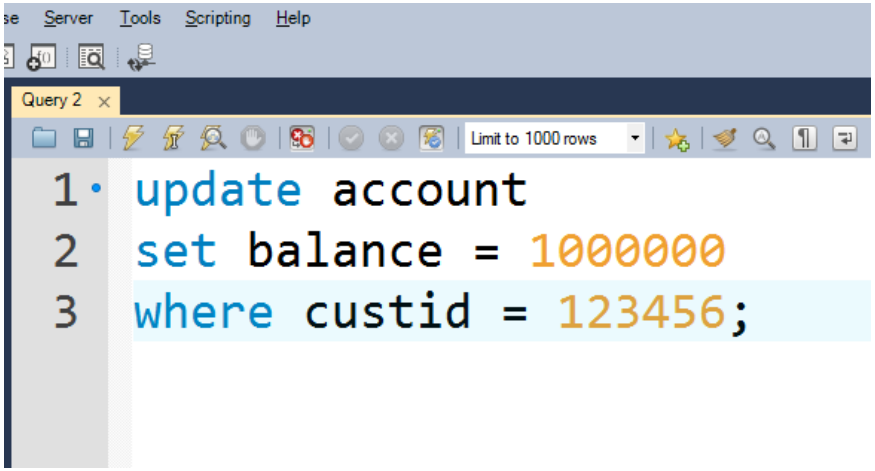


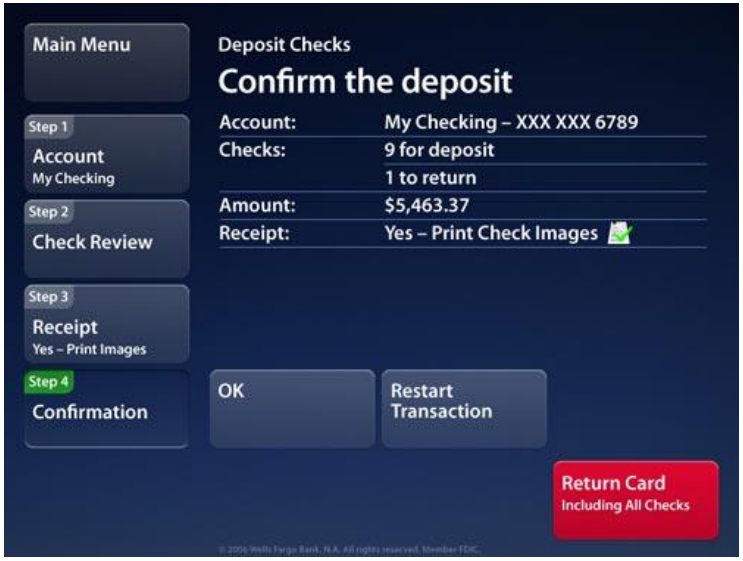


© University of Melbourne 2018

Limitations of SQL

- SQL is declarative, intuitive, versatile, but …
  - cannot express all possible queries in SQL
  - need to enforce business rules beyond domain/ref integrity
  - need procedural constructs such as loops and decisions
  - would you give end-users a query browser? Why not?
  - need a user interface that is both friendly and constraining





© University of Melbourne 2018

How to handle business logic?

- Examples of business logic:
  - Check name and password. If good, login, if bad, error message
  - Insert one row in *Order* table, then several in *OrderItem* table
  - Check amount < balance. If so, subtract amount from one row in bank account table, then add amount to another row
  - For all rows in Customer table, send out monthly statements
- Procedural programming languages can do:
  - Sequence (several steps performed in order)
  - Iteration (loops)
  - Control flow (conditionals, decisions)
  - User interface (accept input and present output for users)
- SQL is specialized for low-level data access

© University of Melbourne 2018

Example business logic

- Customer places an order
  - Accept inputs from user (e.g. via web form)
  - Insert row into Order table
  - Repeat for each product ordered:
    - Check Product table shows sufficient quantity in stock. If so:
      - Insert one row into OrderItem table
      - Change Product table in-stock, Customer table amount-owing
  - If no errors encountered, end successfully
- Customer moves money from savings to credit card account
  - Accept inputs from user (via ATM, internet banking or mobile app)
  - Select balance from savings account
  - Is there enough money to withdraw? If so:
    - Update savings account balance = balance – withdrawal
    - Update credit card balance = balance + withdrawal
  - If no errors encountered, end successfully

© University of Melbourne 2018

SQL + Procedural Language

- Need to combine data manipulation with the ability to handle sequence, iteration, decision. Different approaches:
  - "Embedded SQL"
    - "host language" = C, Fortran, Cobol, Java etc.
    - SQL statements are embedded in code and replaced with library calls during compilation
  - "Dynamic SQL"
    - host language sends SQL to DBMS via middleware e.g. ODBC
    - data is passed back to program as record-set
    - host language can handle business and presentation logic
    - example in next lecture "Web Applications"
  - Stored Procedures, Triggers
    - procedural code is stored and executed in the DBMS
    - enforce business logic within the database
    - in SQL-92 standard, but implemented differently in different DBMS

© University of Melbourne 2018

Stored Procedures and Triggers

- Advantages
  - Compiled SQL statements
  - Faster code execution
  - Reduced network traffic
  - Improved security and data integrity
  - Business logic under control of DBA
  - Thinner clients
- Disadvantages
  - Code is not under the control of the application programmer
  - Proprietary language
    - e.g. MySQL SP's can't be used in Oracle or SQL Server

© University of Melbourne 2018

Example stored procedure

1. accept person details as inputs
2. check whether the person is already in the database
3. if yes, return error
4. if no, add to database

(source: Hoffer chapter 8)

```
CREATE OR REPLACE PROCEDURE p_registerstudent
(
 p_first_name   IN  VARCHAR2
 p_last_name    IN  VARCHAR2
 p_email        IN  VARCHAR2
 p_username     IN  VARCHAR2
 p_password     IN  VARCHAR2
 p_error        OUT  VARCHAR2
)
IS
l_user_exists NUMBER := 0;
l_error      VARCHAR2(2000);

BEGIN

BEGIN
    SELECT COUNT(*)
    INTO   l_user_exists
    FROM   users
    WHERE  username = p_username;

  EXCEPTION
  WHEN OTHERS THEN
    l_error := 'Error: Could not verify username';
  END;

IF l_user_exists = 1 THEN
  l_error := 'Error: Username already exists !';
ELSE

  BEGIN
    INSERT INTO users VALUES(p_first_name,p_last_name,p_email,p_username,p_password,SYSDATE);

  EXCEPTION
    WHEN OTHERS THEN
      l_error := 'Error: Could not insert user';
  END;
END IF;

p_error = l_error;
END p_registerstudent;
```

Procedure p_registerstudent accepts first and last name, email, username, and password as inputs and returns the error message(if any).

This query checks whether the username entered already exists in the database.

If the username already exists, an error message is created for the user.

If the username does not exist in the database, the data entered are inserted into the database.

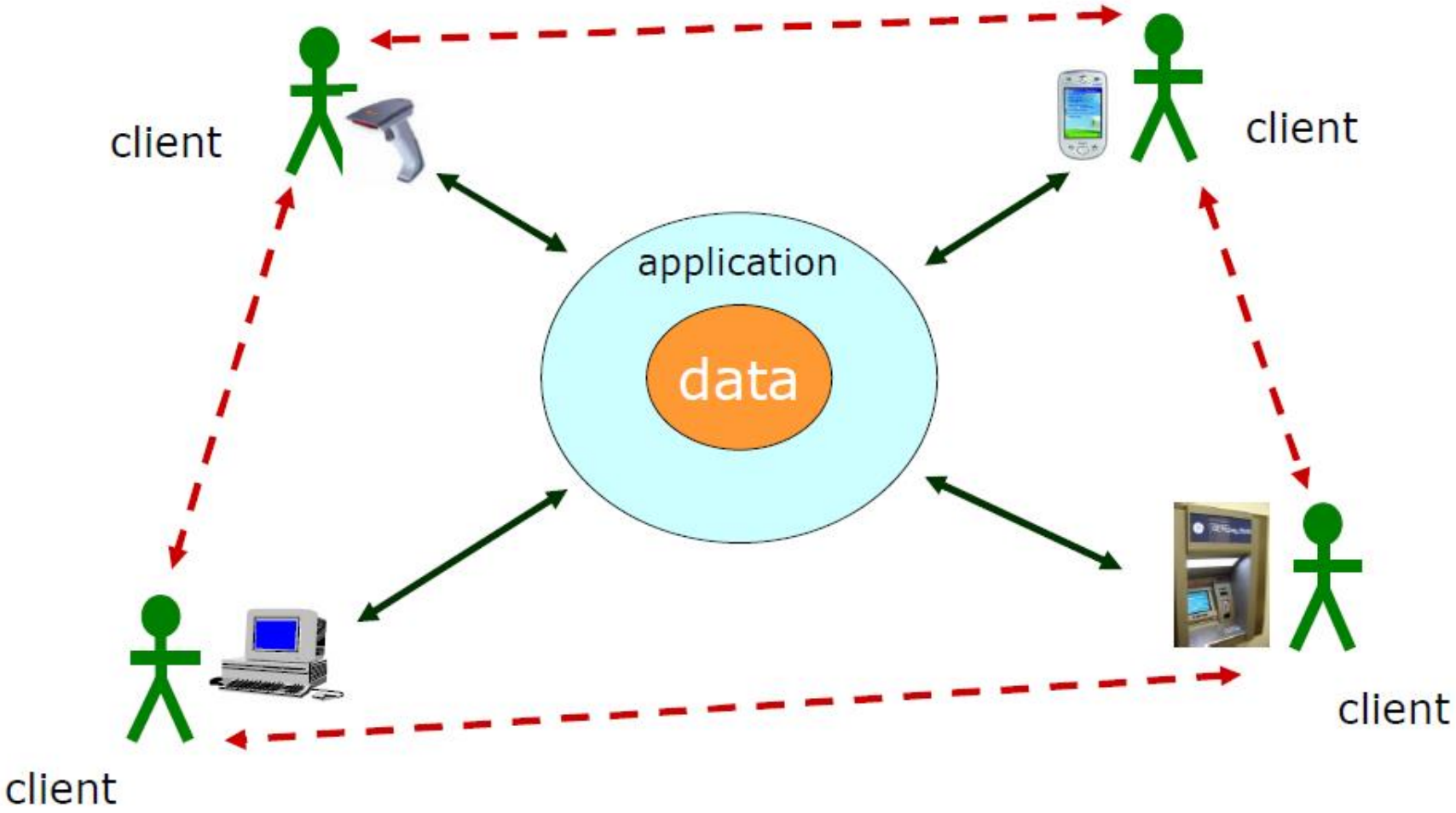
© University of Melbourne 2018

Application Architectures

system architecture = “fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution“

¯ ISO/IEC/IEEE 42010:2011
Systems and software engineering — Architecture description

System architecture



© University of Melbourne 2018

System architecture

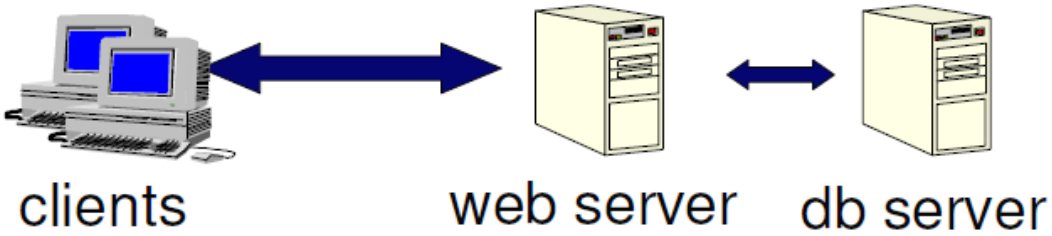
- An information system must provide
  - Presentation logic
    - input (keyboard, touchscreen, voice, sensor etc.)
    - output (large screen, printer, phone, ATM etc.)
  - Business logic
    - input and command handling
    - enforcement of business rules
  - Storage logic
    - persistent storage of data
    - enforcement of data integrity

© University of Melbourne 2018

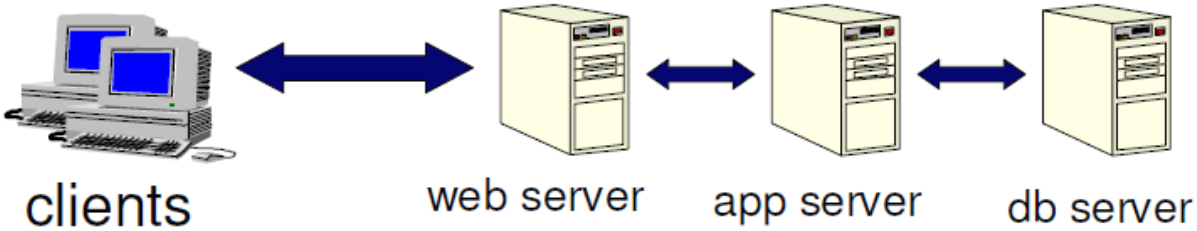
Multi-tiered architectures

- 2 tiers

- 3 tiers



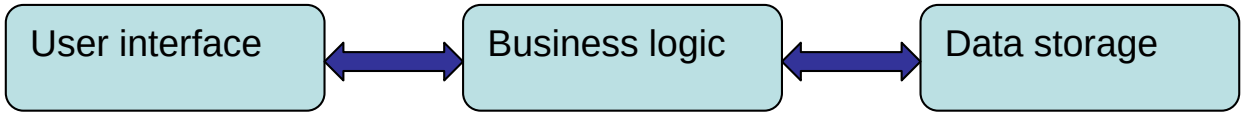
- 4 tiers



© University of Melbourne 2018

Evolution of application architectures

- Mainframe / dumb terminal
  - One large computer handles all logic
  - Problems: doesn't scale with number of users
- Client-Server architecture
  - 2-tier: e.g. file server, database, web
  - 3-tier: separation of Presentation, Processing and Storage logic
- Web architecture
  - a particular form of 3 or 4 tier architecture



© University of Melbourne 2018

Mainframe ("1Tier")

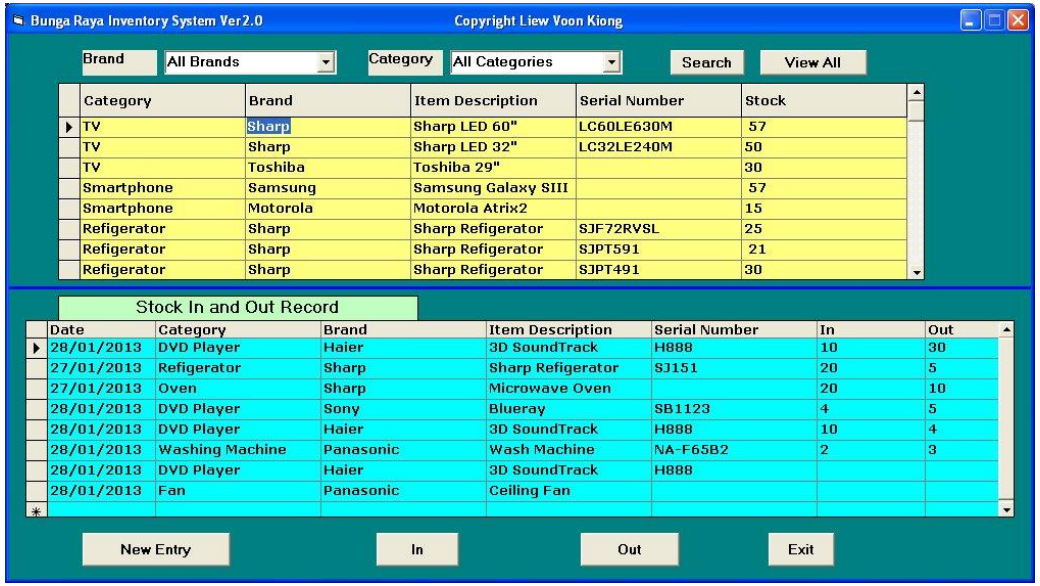
- Mainframes and mini-computers
- Dumb terminals (no processing at client end)
- Entire application ran on the same computer
  - Database
  - Business logic
  - User interface

- Enabling technologies included:
  - Embedded SQL
  - Report generators

© University of Melbourne 2018

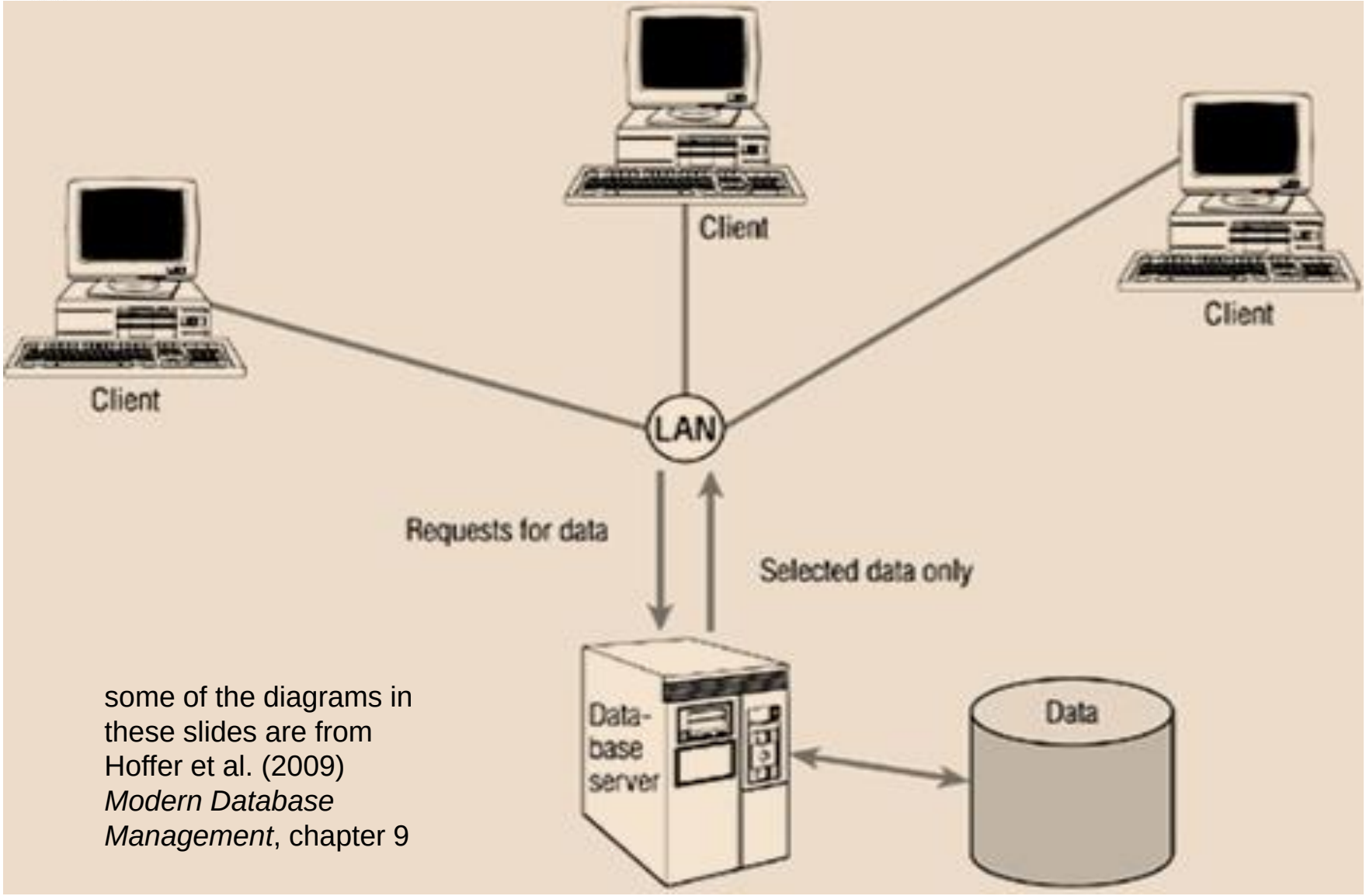
Client Server - 2 Tier

- Server is a relational DBMS
  - data storage and access is done at the DBMS
- SQL queries sent to DB server, which returns raw data
- Presentation, business logic is handled in client application
- Platforms like Visual Basic (1990s into 2000s)



© University of Melbourne 2018

2 Tier Example

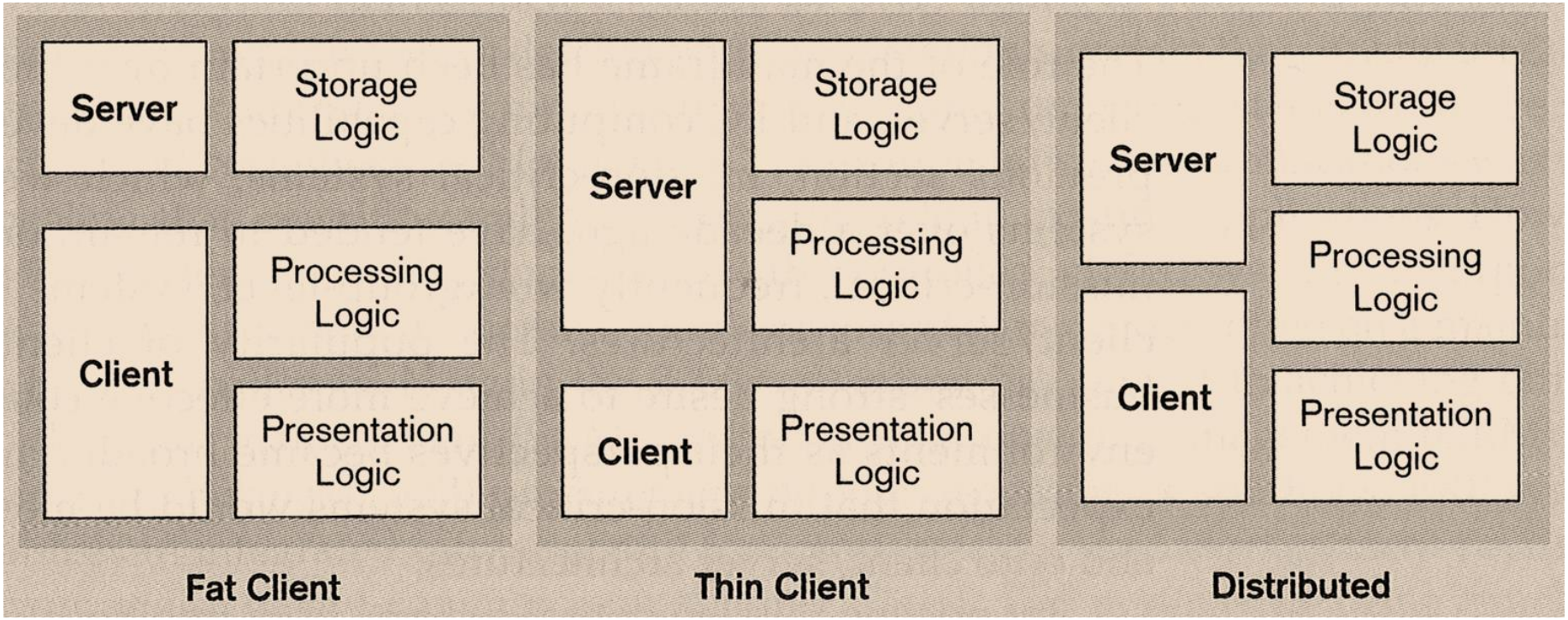


some of the diagrams in these slides are from Hoffer et al. (2009) *Modern Database Management*, chapter 9

© University of Melbourne 2018

Distribution of Processing Logic

- 2-tier distributions
  - Processing logic could be at client, server, or both



© University of Melbourne 2018

2-Tier advantages and disadvantages

- Advantages
  - Clients and server share processing load
  - Good data integrity since data is all processed centrally
  - Stored procedures allow some business rules to be implemented on the database server
- Disadvantages
  - Presentation, data model, business logic are intertwined at client
  - If DB schema changes, all clients break
  - Updates need to be deployed to all clients
  - DB connection for every client, thus difficult to scale
  - Difficult to implement beyond the organization (to customers)

© University of Melbourne 2018

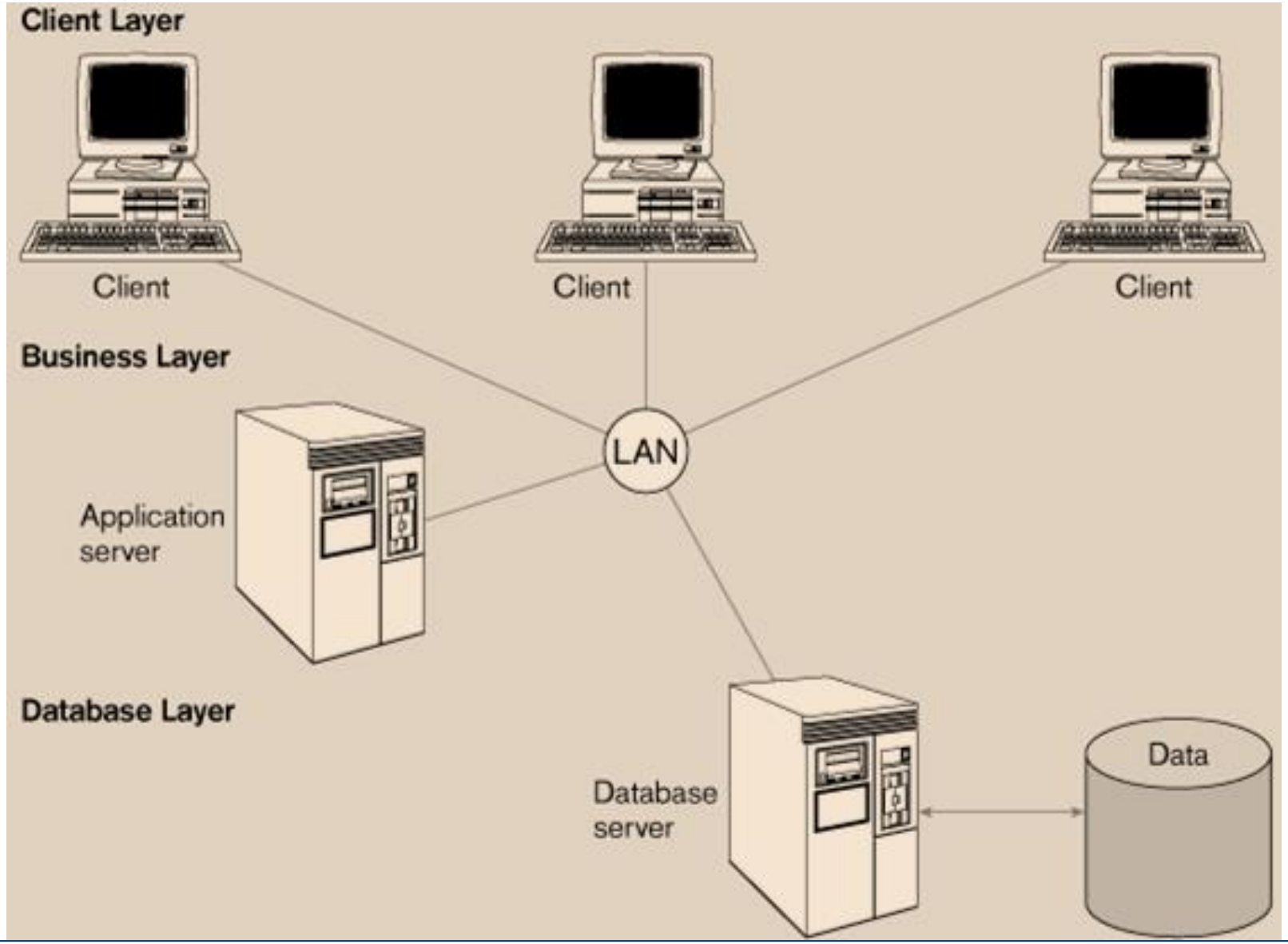
3-Tier architecture

- Client program <-> Application server <-> Database server

- Presentation logic
  - Client handles interface
    - Thinner clients
      - Limited or no data storage (possibly no hard disk)
- Business logic
  - Application Server deals with business logic
- Storage logic
  - Database server deals with data persistence and access

© University of Melbourne 2018

A Three-tier architecture - Example



© University of Melbourne 2018

3-Tier advantages and disadvantages

- Advantages
  - Scalability
  - Technological flexibility (can change business logic easily)
  - Can swap out any single component fairly easily
  - Long-term cost reduction
  - Improved security – customer machine does presentation only
- Disadvantages
  - High short-term costs
  - Tools and training
  - Complex to design
  - Variable standards

© University of Melbourne 2018

3-Tier (web based – see next lecture)

- Browser handles presentation logic
- Browser talks to web server via simple, standard protocol
- Business logic and data storage handled on server(s)
- Pros
  - Everyone has a browser
  - No need for install and maintain client software
  - HTML and HTTP are simple standards, widely supported
  - Opens up the possibility of global access to database
- Cons
  - Even more complexity in the middle-tier
  - Simple standards = hard to make complex application
  - Global access = potential security nightmare (next page)

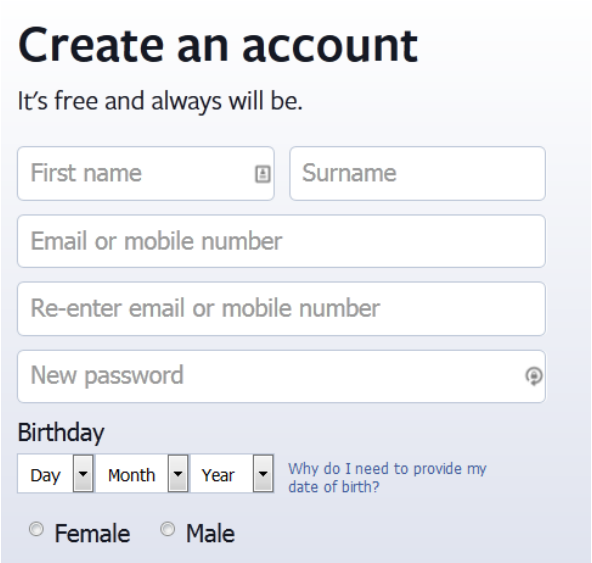
© University of Melbourne 2018

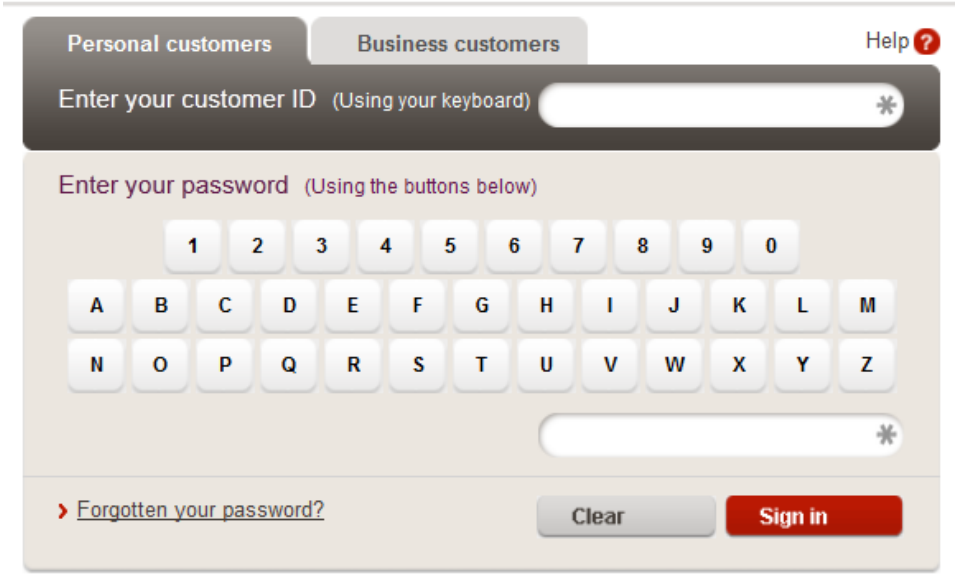
Security in multi-tier applications

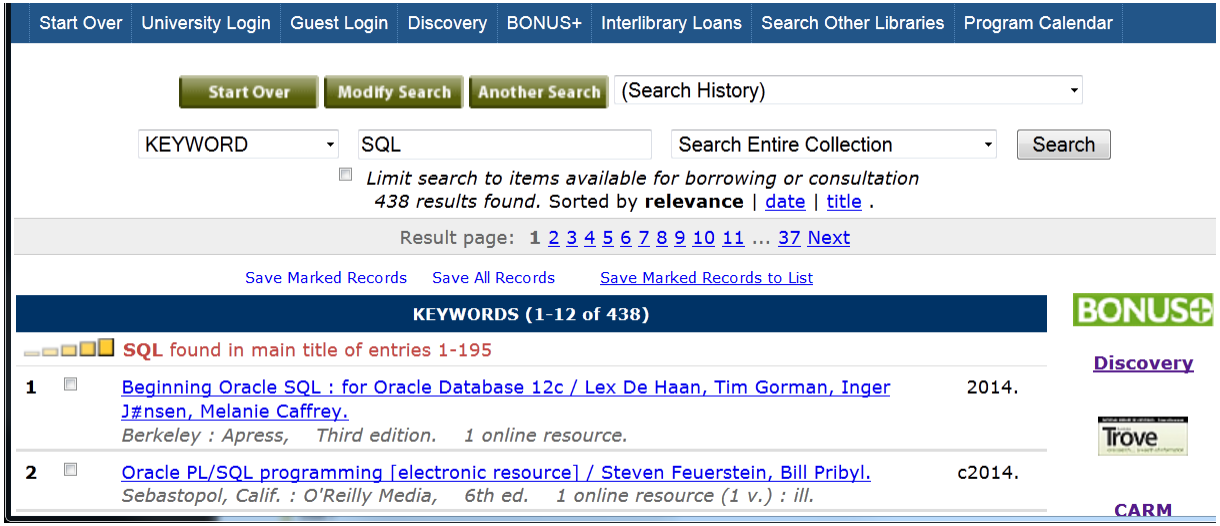
- Network environment creates complex security issues
- Security can be enforced at different tiers:
  - application password security
    - for allowing access to the application software
  - database-level password security
    - for determining access privileges to tables
  - secure client/server communication
    - via encryption

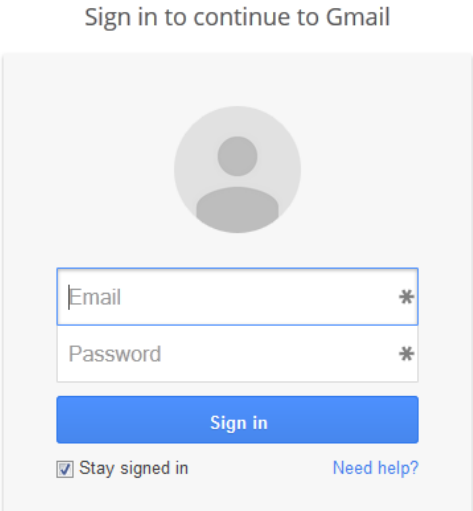
© University of Melbourne 2018

Example web applications









© University of Melbourne 2018

Dr Greg Wadley

THE UNIVERSITY OF MELBOURNE

INFO90002 Database Systems & Information Modelling

Week 07

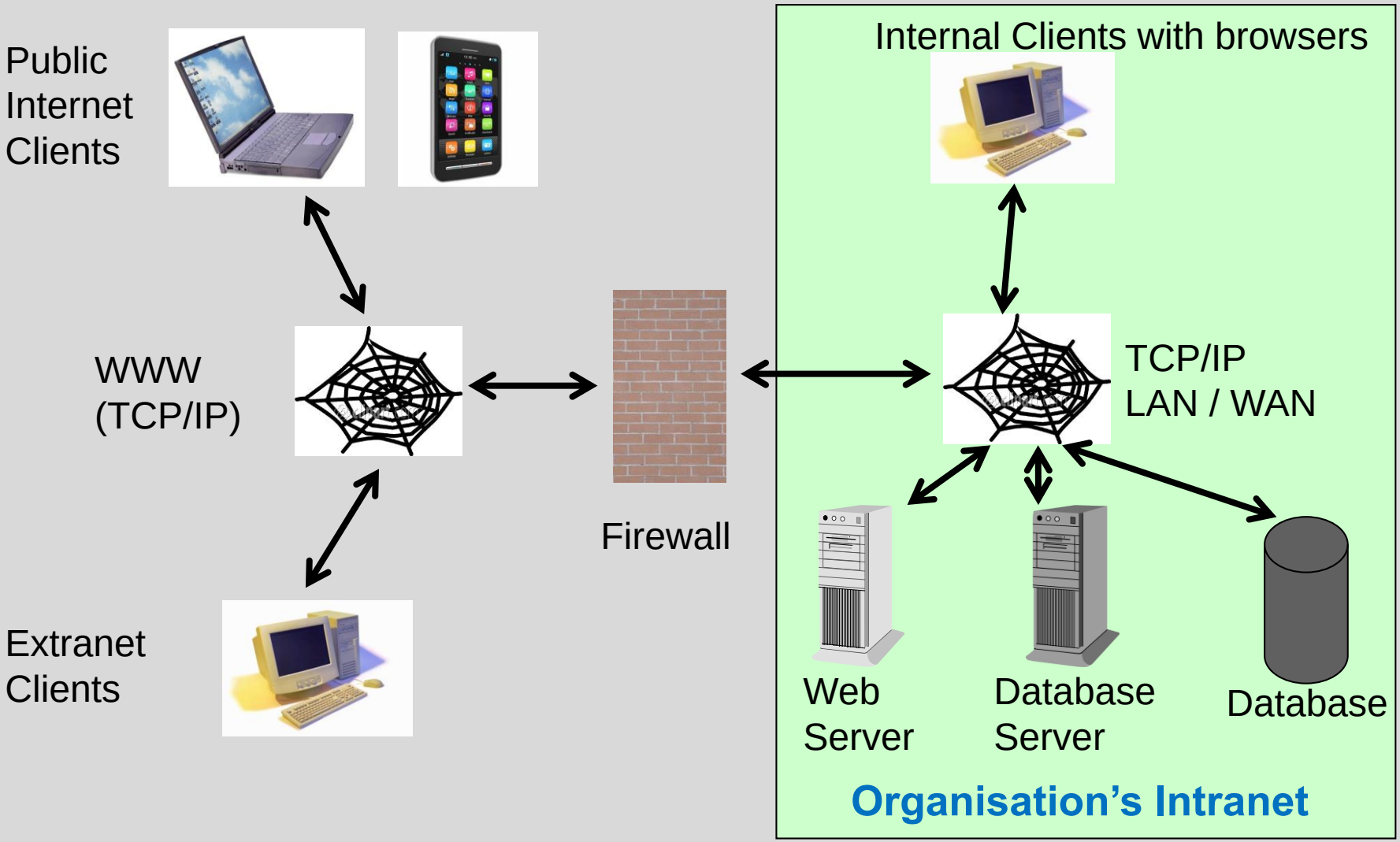
Web Apps

Today's Session…

- Why web apps?
- How web apps work
- Making an HTML document
- Connecting to the DB
- Demo web app
- Web services

© University of Melbourne 2018

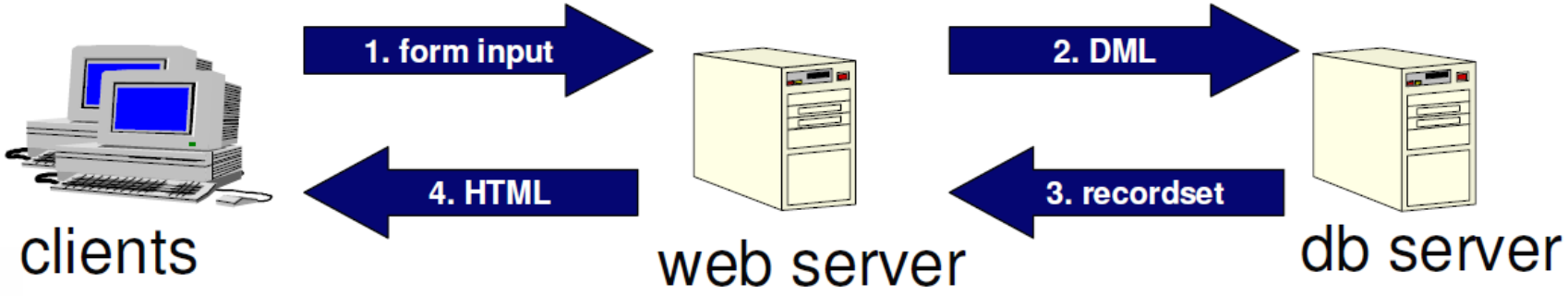
Architecture of a web app



© University of Melbourne 2018

Why create web applications?

- Web browsers are ubiquitous
- No need to install client software for external customers
- Simple communication protocols
- Platform and Operating System independent
- Reduction in development time and cost
- Has enabled eGov, eBusiness, eCommerce, B2B, B2C



© University of Melbourne 2018

Web infrastructure

- Browser
  - Software that retrieves and displays HTML documents
- Web Server
  - Software that responds to requests from browsers by transmitting HTML and other documents to browsers
- Web pages (HTML documents)
  - Static web pages
    - content established at development time
  - Dynamic web pages
    - content dynamically generated using data from database
- World Wide Web (WWW)
  - The total set of interlinked hypertext documents residing on Web servers worldwide

© University of Melbourne 2018

Web-related languages

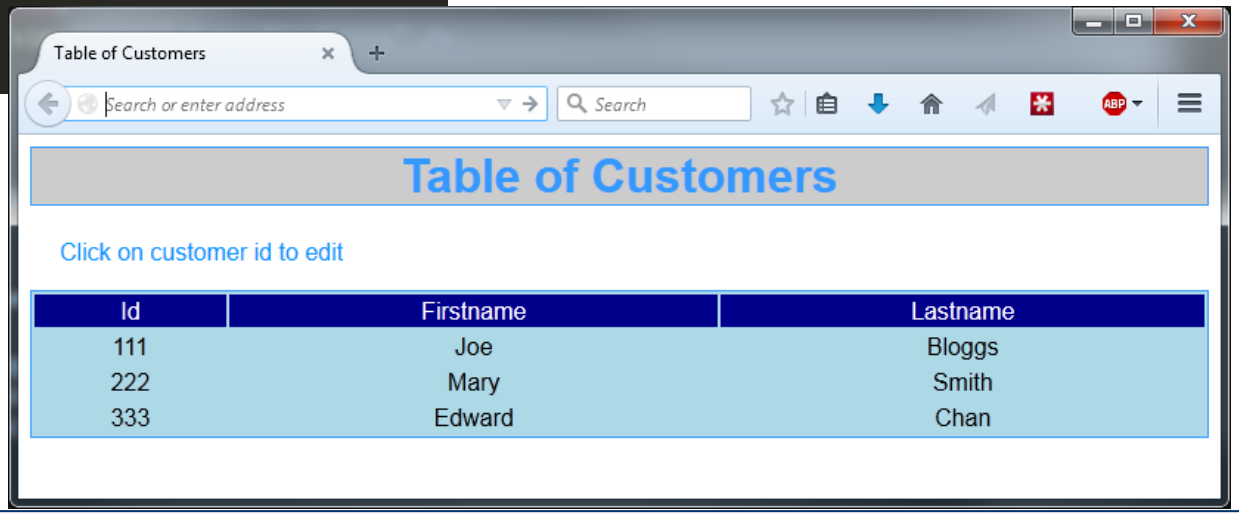
- Hypertext Markup Language (HTML)
  - Markup language used to define a web page
- Cascading Style Sheets (CSS)
  - Control appearance of an HTML document
- JavaScript (JS)
  - Scripting language that enable interactivity in HTML documents
- Extensible Markup Language (XML)
  - Markup language used to transport data between web services

© University of Melbourne 2018

Web page = HTML document

- a structured file of elements defined by HTML tags
- interpreted by web browser for display

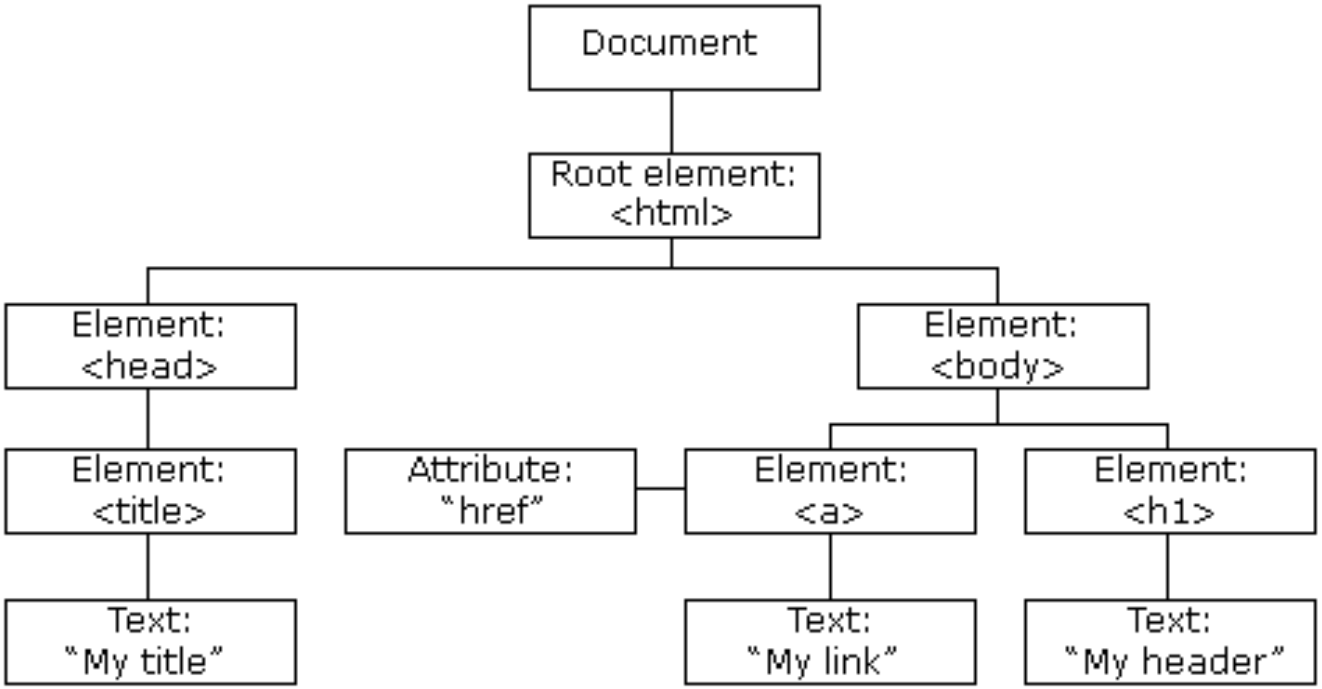
```html
<head>
    <title>Table of Customers</title>
    <link rel="stylesheet" href="simple.css" type="text/css" />
</head>

<body>
    <h1>Table of Customers</h1>
    <p>Click on customer id to edit</p>
    <table>
        <thead>
            <tr><td>Id<td>Firstname<td>Lastname</tr>
        </thead>
        <tr><td>111<td>Joe<td>Bloggs</tr>
        <tr><td>222<td>Mary<td>Smith</tr>
        <tr><td>333<td>Edward<td>Chan</tr>
    </table>
</body>
```



© University of Melbourne 2018

Structure of an HTML document

- elements are structured as a tree (one web page = one tree)
- divided into a HEAD and a BODY
- the BODY is what you see displayed in the browser
- BODY is divided into elements such as headings, paragraphs, tables, lists …



picture source: W3 Schools

© University of Melbourne 2018

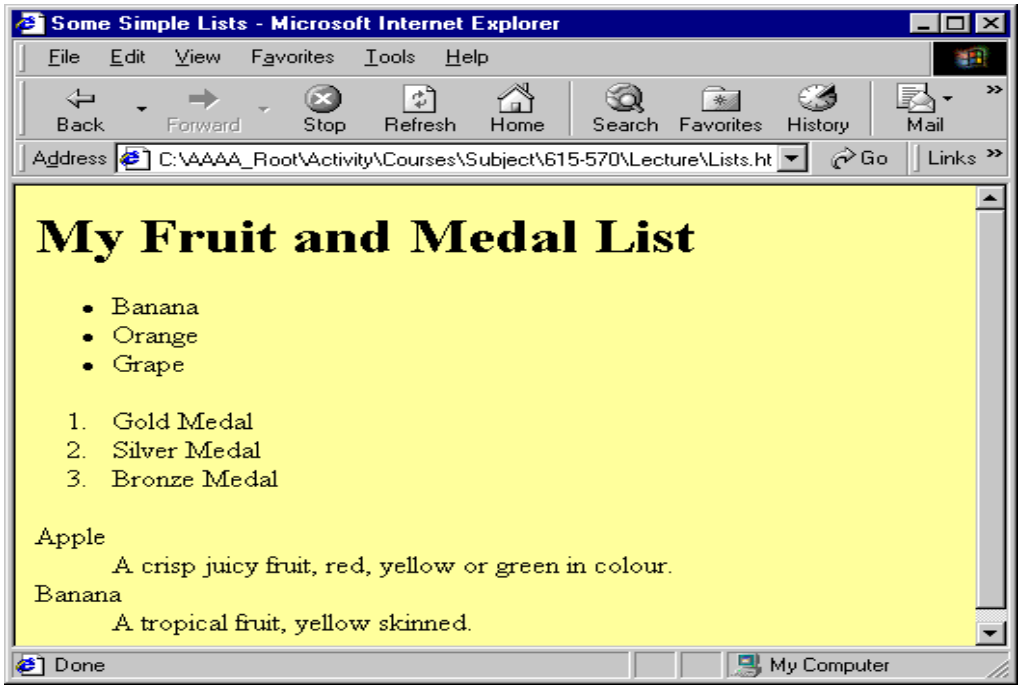
Important HTML elements

<HEAD> … </HEAD>	• document header.
<BODY> … </BODY>	• document body
<H1> … </H1>	• Heading type 1
<H6> … </H6>	• … to Heading type 6.
<P> … </P>	• paragraph.
<TABLE>	• table
<TR>	• table row
<TD>	• table data
<UL>	• list
<LI>	• list item

© University of Melbourne 2018

HTML list

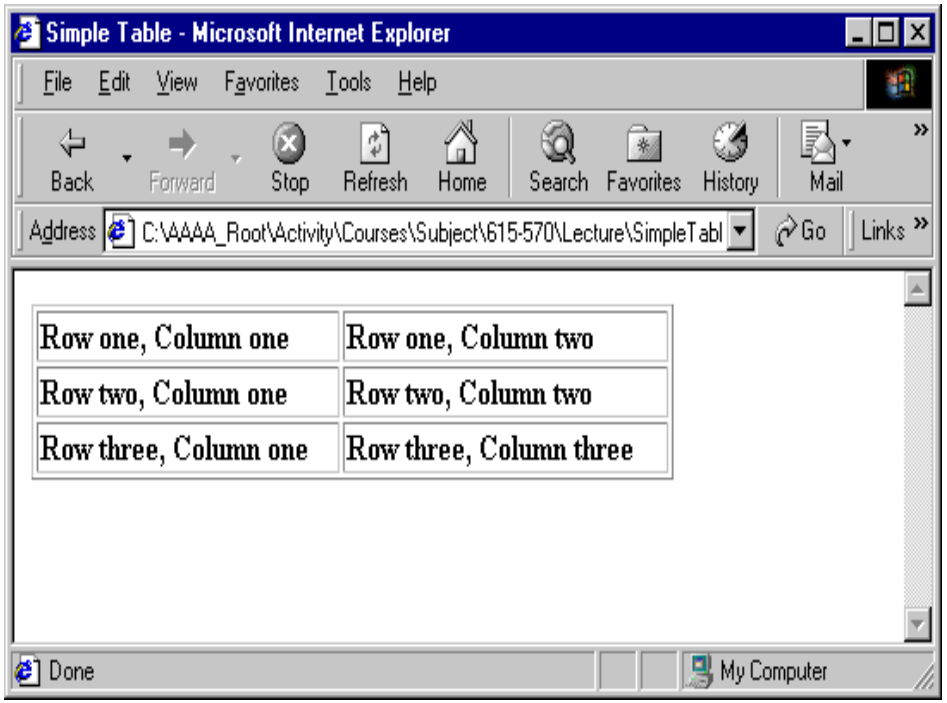
```
<HTML> <HEAD> <title>Some Simple Lists</title> </HEAD>
<BODY bgcolor="#FFFF99">
<H1>My Fruit and Medal List </H1>
<UL>
  <LI>Banana</LI>
  <LI>Orange</LI>
  <LI>Grape</LI>
</UL>
<OL>
  <LI>Gold Medal</LI>
  <LI>Silver Medal</LI>
  <LI>Bronze Medal</LI>
</OL>
<DL>
  <DT>Apple
    <DD>A crisp juicy fruit, red, yellow or green in colour.
  <DT>Banana
    <DD>A tropical fruit, yellow skinned.
</DL>
</BODY> <HTML>
```



© University of Melbourne 2018

HTML table

```
<HTML>  <HEAD> <TITLE>Simple Table</TITLE>
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</HEAD>
<BODY>
<TABLE>
  <TR>
    <TD>Row one, Column one</TD>
    <TD>Row one, Column two</TD>
  </TR>
  <TR>
    <TD>Row two, Column one</TD>
    <TD>Row two, Column two</TD>
  </TR>
  <TR>
    <TD>Row three, Column one</TD>
    <TD>Row three, Column three</TD>
  </TR>
</TABLE>  </BODY> </HTML>
```

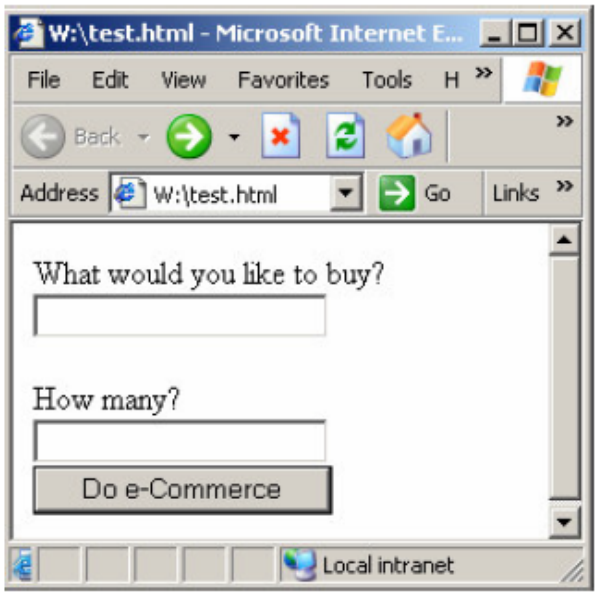


© University of Melbourne 2018

HTML form

- Forms allow users to input data to a web page
- The web server process the user's input using the file named in the 'action' attribute.

```
<form action = "buy.pl" method="post">

<p> What would you like to buy? <br>
<input type="text" name="product">

<p> How many? <br>
<input type="text" name="quantity"> <br>

<input type="submit" value="Do e-Commerce">
</form>
```



Example HTML form

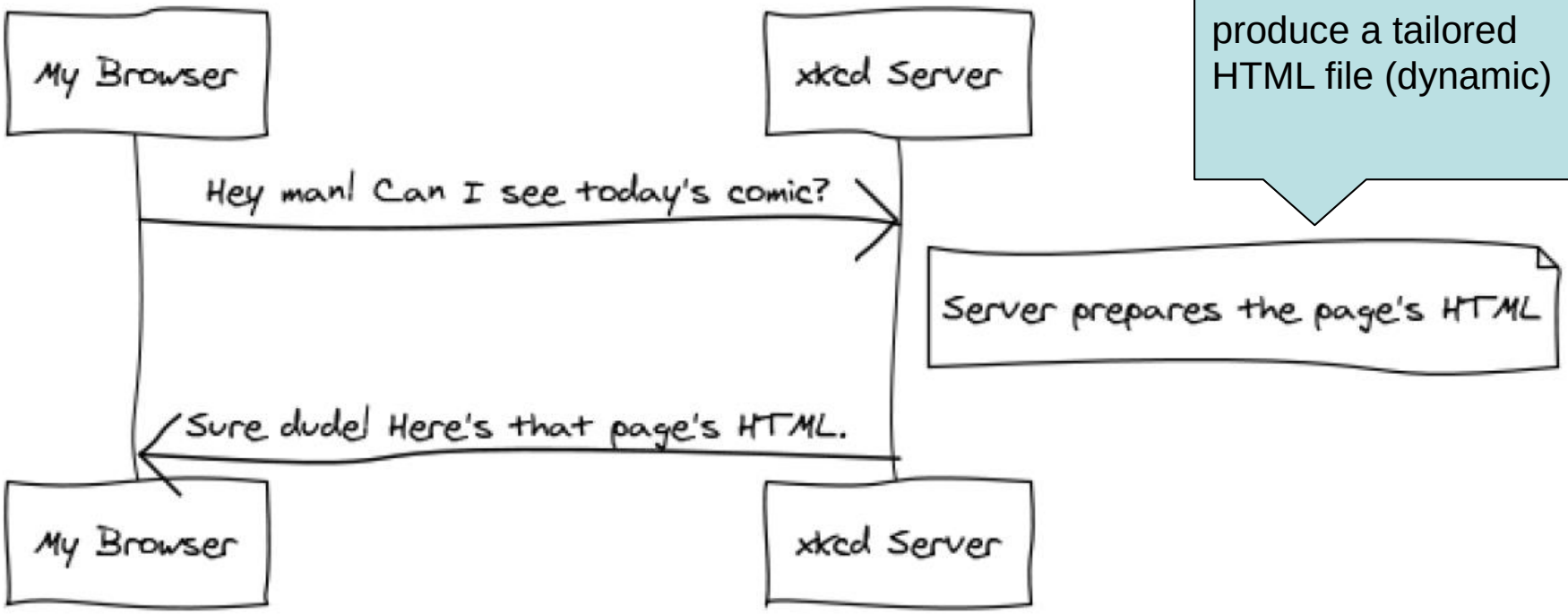
browser displays form,
sends input data to a script called 'buy.pl'

© University of Melbourne 2018

HTTP: how HTML documents move

- User wants to see a web page
- Types URL into browser
- Browser fetches page from server and displays it

could simply load a pre-prepared HTML file (static), or query the database to produce a tailored HTML file (dynamic)



picture source: Symfony Book

© University of Melbourne 2018

Static vs Dynamic web pages

- STATIC web page
  - the URL identifies a file on the server's file system
  - server fetches the file and sends it to the browser
  - the file contains HTML
  - browser interprets the HTML for display on screen
- DYNAMIC web page
  - URL identifies a program to be run
  - web app runs the program
  - program typically retrieves data from database
  - elements such as TABLE, LIST are populated with data
    - web app uses LOOPS to fill the contents of TABLEs and LISTs.
    - e.g. SELECT * FROM Product; (returns a set of product entities)
    - FOR p IN ProductList, print a row in HTML table

© University of Melbourne 2018

Simple web app using PHP and SQL

- program logs into db
- selects all rows from database table
- displays them inside an HTML table

```php
<?php

print '<h1> This page selects from a table </h1>';

print '<p> connecting to database ... </p>';
// connect to server, select database
$link = mysql_connect('localhost', 'root', '')
    or die('Could not connect: ' . mysql_error());
print '<p> connected successfully </p>';
mysql_select_db('webappdemo') or die('could not select database');

// perform SQL query
$query = 'SELECT * FROM mytable';
$result = mysql_query($query) or die('Query failed: ' . mysql_error());

print '<h2> table starts now </h2>';

// print results in an HTML table
print "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
    print "\t<tr>\n";
    foreach ($line as $col_value) {
        print "\t\t<td>$col_value</td>\n";
    }
    print "\t</tr>\n";
}
print "</table>\n";

```


© University of Melbourne 2018

Simple web app: *select*

```php
<?php

print '<h1> This page selects from a table </h1

print '<p> connecting to database ... </p>';
// connect to server, select database
$link = mysql_connect('localhost', 'root', '')
    or die('Could not connect: ' . mysql_error(
print '<p> connected successfully </p>';
mysql_select_db('webappdemo') or die('could not

// perform SQL query
$query = 'SELECT * FROM mytable';
$result = mysql_query($query) or die('Query fai

print '<h2> table starts now </h2>';

// print results in an HTML table
print "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL
    print "\t<tr>\n";
    foreach ($line as $col_value) {
        print "\t\t<td>$col_value</td>\n";
    }
    print "\t</tr>\n";
}
print "</table>\n";
```

localhost/demo/select.php

This page selects from a table

connecting to database ...

connected successfully

table starts now

1 first row

2 second row

3 third row - working nicely

form starts now

3	
hird row - working nicely	

Submit Query

© University of Melbourne 2018

Simple web app: *insert*

```php
print '<h2> form starts now </h2>';

// display a form for entering data
print '<form action="insert.php" method="post">';
print '<input type="text" name="number" value="type a number" />
print '<input type="text" name="string" value="type a string" />
print '<input type="submit" value="send to database" />';
print '</form>';
?>
```

form starts now

3

third row - working nicel

Submit Query

```php
<?php

print '<p> connecting to database ... </p>';
// connect to server, select database
$link = mysql_connect('localhost', 'root', '')
    or die('Could not connect: ' . mysql_error());
print '<p> connected successfully </p>';
mysql_select_db('webappdemo') or die('could not select database');

// form the INSERT statement from the user's input
$sql="insert into mytable values ('$_POST[number]','$_POST[string]')";

// run the INSERT statement
if (!mysql_query($sql,$link))
  die('Error: ' . mysql_error());

// print friendly message
print "<p> 1 record added:  </p>";
print "<ul>";
print "<li>the number was: " . $_POST['number'];
print "<li>the string was: " . $_POST['string'];
print "</ul>";

// close connection to database
mysql_close($link);

?>
```

localhost/demo/insert.php

connecting to database ...

connected successfully

1 record added:

- the number was: 3
- the string was: third row - working nicely

© University of Melbourne 2018

Problems with old-style web apps

```
$sql = "insert into EVENT values (null, null, 'L', '" . $_SESSION["thisClient"] . "', 'logged in')";
mysql_query($sql);
```

- Placing “raw” SQL inside PHP/HTML files
  - Mixes presentation, business, database logic
  - Hard to maintain when things change
  - Want separation of concerns e.g. MVC
- Lots of reinvention of wheels
  - each dev writes their own solution to common features
  - e.g. login security, presentation templates, database access
- Increasing variety of clients e.g. phones and tablets
  - Manually program for different platforms
- => web application frameworks
  - examples: Ruby on Rails, .Net, Symfony, AngularJS, Django

© University of Melbourne 2018

Web Services

- The WWW allows humans to access remote databases
- Web Services allow *computers* to access remote databases
- 2 major approaches: SOAP and REST
  - Simple Object Access Protocol
  - Representational State Transfer
- structured data usually returned in XML or JSON format
- REST nouns are resources, addressed via URIs
- REST verbs correspond to DML statements
- GET (select), POST (insert), PUT (update), DELETE (delete)
- Try this example web service
  https://www.googleapis.com/books/v1/volumes?q=quilting

© University of Melbourne 2018

XML and JSON data formats

- used by web services for data exchange

- XML
  eXensible Markup Language

- JSON
  JavaScript Object Notation

(example sourced from W3 schools)

The following JSON example defines an employees object, with an array of 3 employee records:

JSON Example

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

The following XML example also defines an employees object with 3 employee records:

XML Example

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```


© University of Melbourne 2018

David Eccles

INFO90002 Database Systems & Information Modelling

Week 08

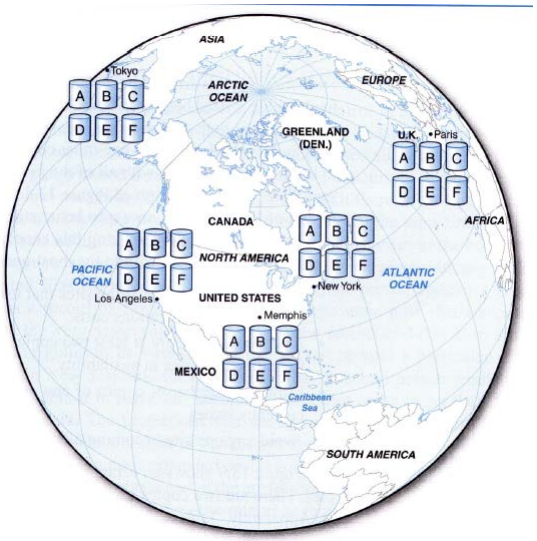
Distributed Databases

Today's lecture

- What is a distributed database?
- Why are they used, and how they work
- Pros and cons of different approaches
  - material in this lecture is drawn from Hoffer et al. (2013) *Modern Database Management* 11th edition, chapter 12, available online at http://wps.prenhall.com/bp_hoffer_mdm_11/230/58943/15089539.cw/index.html
  - pictures on this page are from Gillenson (2005) *Fundamentals of Database Management Systems*

centralized database

distributed database



replicated database

© University of Melbourne 2018

Definitions

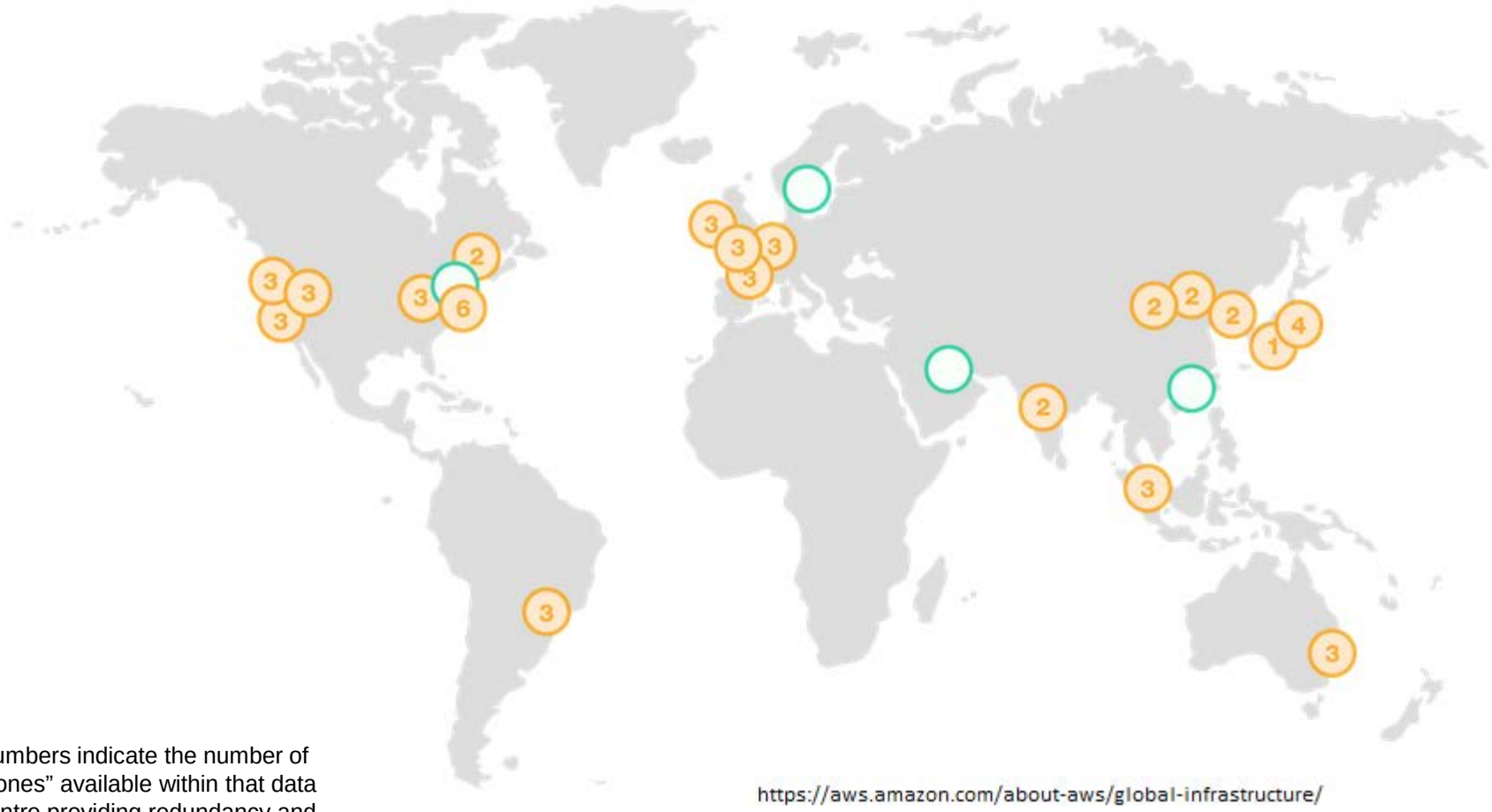
- Distributed Database
  - a single logical database physically spread across multiple computers in multiple locations that are connected by a data communications link
  - *appears to users as though it is one database*
- Decentralized Database
  - a collection of independent databases which are not networked together as one logical database
  - appears to users as though many databases

- We are concerned with *distributed* databases

© University of Melbourne 2018

Example – Amazon AWS

Global Infrastructure



Numbers indicate the number of "zones" available within that data centre providing redundancy and availability

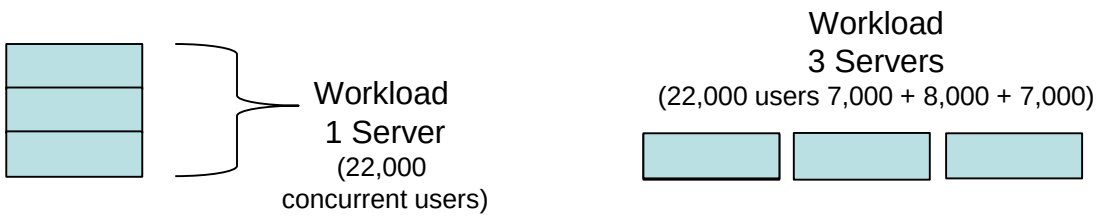
https://aws.amazon.com/about-aws/global-infrastructure/

© University of Melbourne 2018

Distributed DBMS Advantages

- Good fit for geographically distributed organizations / users
  - Utilise the internet
- Data located near site with greatest demand
  - ESPN Weekend Sports Scores

- Faster data access (to local data)
- Faster data processing
  - workload is shared between each physical server



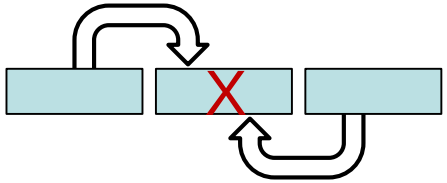
© University of Melbourne 2018

Distributed DBMS Advantages

- Allows modular growth
  - add new servers as load increases

- Increased reliability and availability
  - less danger of a single-point of failure (SPOF)

- Supports database recovery
  - data is replicated across multiple sites



© University of Melbourne 2018

Disadvantages

- Complexity of management and control
  - db or/and application must stitch together data across sites
    - Who and where is the current version of the record (row & column)?
    - Who is waiting to update that information and where are they?
    - How does the logic display this to the web & application server?
- Data integrity
  - additional exposure to improper updating
    - If two users in two locations update the record at the exact same time who decides which statement should “win”?
    - Solution: Transaction Manager or Master-slave design
- Security
  - many server sites -> higher chance of breach
    - Multiple access sites require protection including network and storage infrastructure from both cyber & physical attacks

© University of Melbourne 2018

Disadvantages

- Lack of standards
  - different Relational DDBMS vendors use different protocols
- Increased training & maintenance costs
  - more complex IT infrastructure
    - Increased Disk storage ($)
    - Fast intra and inter network infrastructure ($$)
    - Clustering software ($)
    - Network Speed ($$)
- Increased storage requirements
  - Replication model

© University of Melbourne 2018

Objectives and Trade-offs

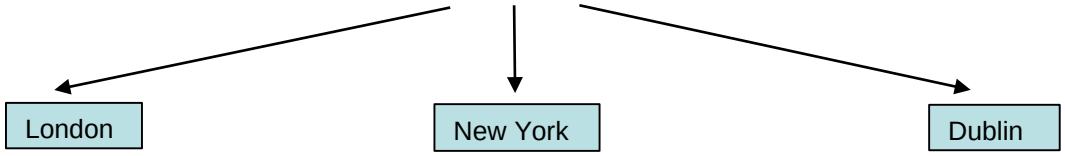
- Location transparency
  - a user does not need to know where particular data are stored
- Local autonomy
  - a node can continue to function for local users if connectivity to the network is lost
- Trade-offs
  - Availability vs Consistency
  - Synchronous vs Asynchronous updates

© University of Melbourne 2018

Location Transparency

- A user (or program) accessing data do not need to know the location of the data in the network of DBMS's
- Requests to retrieve or update data from any site are automatically forwarded by the system to the site or sites related to the processing request
- All data in the network appears as a single logical database stored at one site to the users
- A single query can join data from tables in multiple sites

```
SELECT hometeam, homescore, awayteam, awayscore
FROM results inner join codes
ON results.codeid = codes.codeid
WHERE sportscode in ('NFL', 'Hurling', 'EPL');
```



© University of Melbourne 2018

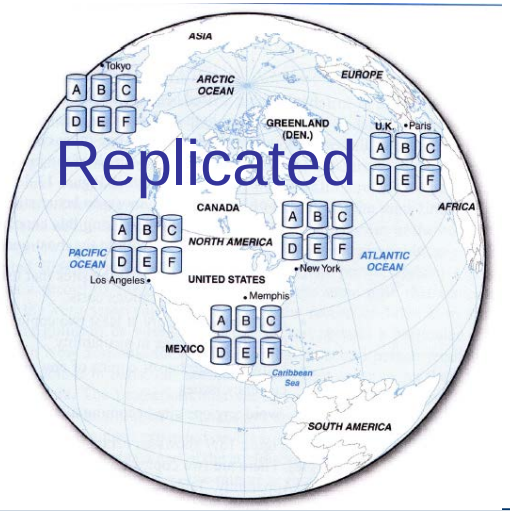
Local Autonomy

- Users can administer their local database
  - control local data (e.g Hurling results)
  - administer security
  - log transactions
  - recover when local failures occur
  - provide full access to local data
- Being able to operate locally when connections to other databases fail

© University of Melbourne 2018

Distribution options

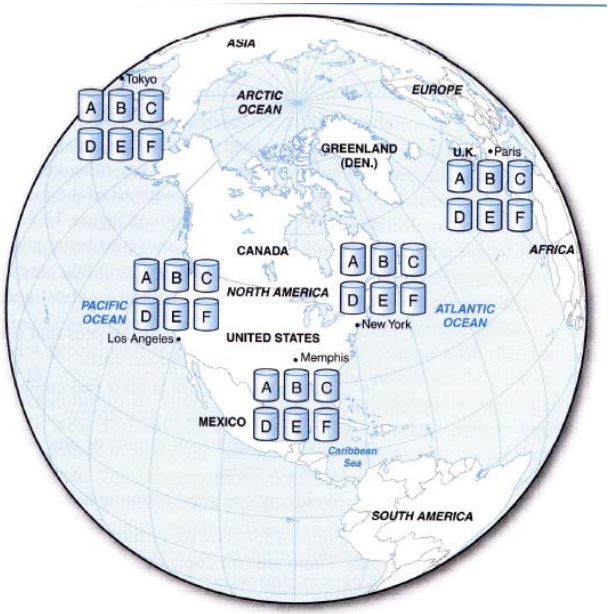
- Data replication
  - Data copied across sites
- Horizontal partitioning
  - Table rows distributed across sites
- Vertical partitioning
  - Table columns distributed across sites
- Combinations of the above



© University of Melbourne 2018

Replication - advantages

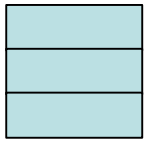
- High reliability due to redundant copies of data
- Fast access to data at the location where it is most accessed
- May avoid complicated distributed integrity routines
  - replicated data is refreshed at scheduled intervals
- Decoupled nodes don't affect data availability
  - transactions proceed even if some nodes are down
- Reduced network traffic at prime time
  - if updates can be delayed
- This is currently popular as a way of achieving high availability for global systems.
  - Most SQL & NoSQL databases offer replication



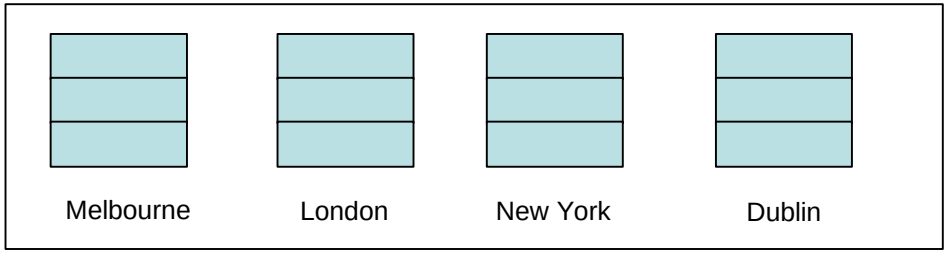
© University of Melbourne 2018

Replication - disadvantages

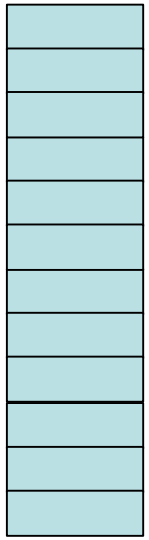
- Need more storage space
  - Each server stores a copy of the row
- Data Integrity:
  - high tolerance for out-of-date data may be required
  - updates may cause performance problems for busy nodes
  - retrieve incorrect data if updates have not arrived



Centralised Database
One database in one server
(1 copy of data)



Distributed Database
One database in 4 physical servers
(4 copies of data)

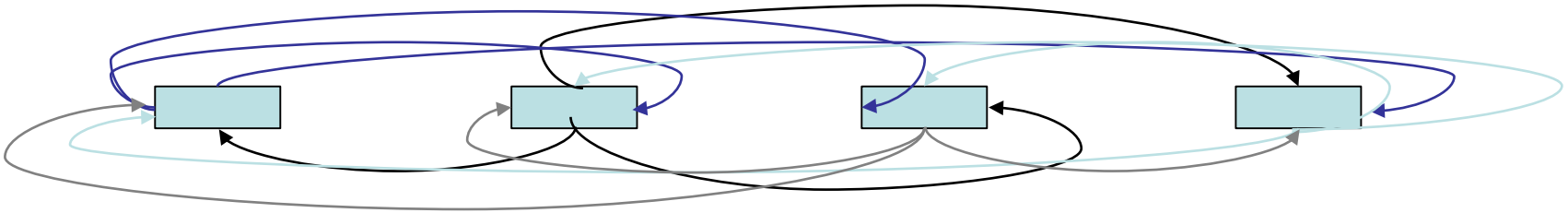


Data Size

© University of Melbourne 2018

Replication - disadvantages

- Takes time for update operations
  - high tolerance for out-of-date data may be required
  - updates may cause performance problems for busy nodes



- Network communication capabilities
  - updates can place heavy demand on telecommunications/networks

© University of Melbourne 2018

Synchronous updates

- Data is continuously kept up to date
  - users anywhere can access data and get the same answer.
- If any copy of a data item is updated anywhere on the network, the same update is immediately applied to all other copies or it is aborted.
- Ensures data integrity and minimizes the complexity of knowing where the most recent copy of data is located.
- Can result in slow response time and high network usage
  - the DDBMS spends time checking that an update is accurately and completely propagated across the network.
  - The committed updated record must be identical in all servers

© University of Melbourne 2018

Asynchronous updates

- Some delay in propagating data updates to remote databases
  - some degree of at least temporary inconsistency is tolerated
  - may be ok it is temporary and well managed
- Acceptable response time
  - updates happen locally and data replicas are synchronized in batches and predetermined intervals
- May be more complex to plan and design
  - need to ensure the right level of data integrity and consistency
- Suits some information systems more than others
  - compare commerce/finance systems with social media

© University of Melbourne 2018

Horizontal partitioning

- Different rows of a table at different sites
- Advantages
  - data stored close to where it is used
    - efficiency
  - local access optimization
    - better performance
  - only relevant data is stored locally
    - security
  - unions across partitions
    - ease of query
- Disadvantages
  - accessing data across partitions
    - inconsistent access speed
  - no data replication
    - backup vulnerability (SPOF)

Team table

ID	Team	City	Code	Region	League
1	Arsenal	London	Football	Europe	EPL
2	Jets	NYC	Grid Iron	Americas	NFL
3	Carlton FC	Melbourne	Aussie Rules	APAC	AFL
4	Racing92	Paris	Rugby	Europe	Top14
5	Yankees	NYC	Baseball	Americas	MLB
6	Swifts	Sydney	Netball	APAC	ANZ

© University of Melbourne 2018

Example horizontal partitioning

ID	Team	City	Code	Region	League
1	Arsenal	London	Football	Europe	EPL
2	Jets	NYC	Grid Iron	Americas	NFL
3	Carlton FC	Melbourne	Aussie Rules	APAC	AFL
4	Racing92	Paris	Rugby	Europe	Top14
5	Yankees	NYC	Baseball	Americas	MLB
6	Swifts	Sydney	Netball	APAC	ANZ

Horizontal Partitioning based on Region

London

Team

1,Arsenal,London, Football, Europe, EPL

4,Racing92, Paris, Rugby, Europe, Top14

Melbourne

Team

3,CarltonFC,Melbourne, Aussie Rules, APAC, AFL

6,Swifts, Sydney, Netball, APAC, ANZ

New York

Team

2, Jets, NYC, Grid Iron, Americas, NFL

5, Yankees, NYC, Baseball, Americas, MLB

© University of Melbourne 2018

Vertical partitioning

- Different columns of a table at different sites
- Advantages and disadvantages are the same as for horizontal partitioning
  - except
    - combining data across partitions is more difficult because it requires joins (instead of unions)

Player table

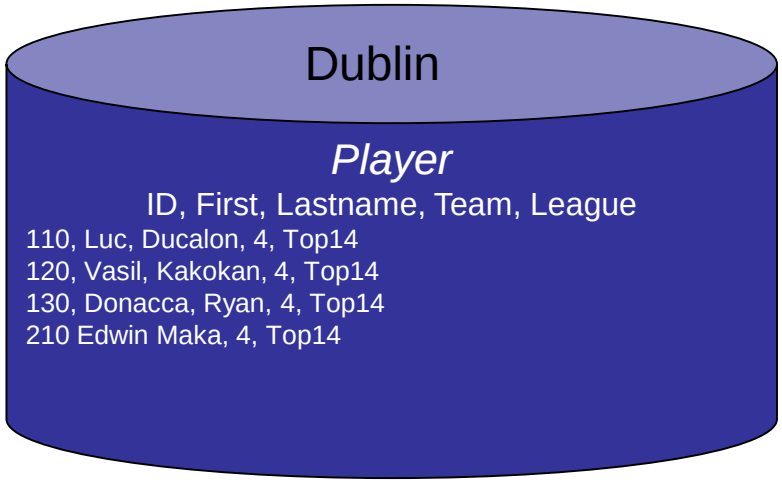
ID	Firstname	Lastname	Team	League	Photo	Biography
110	Luc	Ducalon	4	Top14		Ipso locum
120	Vasil	Kakokan	4	Top14		Ipso locum est
130	Donacca	Ryan	4	Top14	<null>	
210	Edwin	Maka	4	Top14		

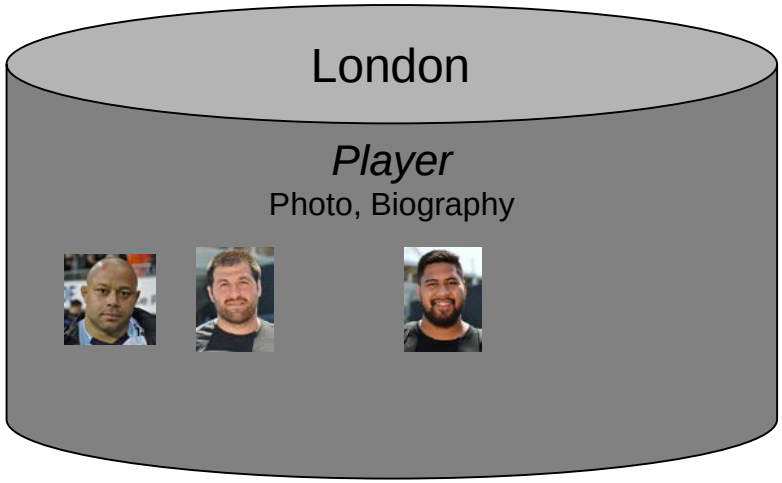
© University of Melbourne 2018

Example vertical partitioning

ID	Firstname	Lastname	Team	League	Photo	Biography
110	Luc	Ducalon	4	Top14		Ipso locum
120	Vasil	Kakokan	4	Top14		Ipso locum est
130	Donacca	Ryan	4	Top14	<null>	
210	Edwin	Maka	4	Top14		

Vertical Partitioning based on column requirements





© University of Melbourne 2018

Comparing 5 configurations

- Centralized database, distributed access
  - DB is at one location, and accessed from everywhere
- Replication with periodic snapshot update
  - many locations, each data copy updated periodically
- Replication with near real-time synchronization of updates
  - many locations, each data copy updated in near real time
- Partitioned, integrated, one logical database
  - data partitioned across at many sites, within a logical database, and a single DBMS
- Partitioned, independent, nonintegrated segments
  - data partitioned across many sites.
  - independent, non-integrated segments
  - multiple DBMS, multiple computers

© University of Melbourne 2018

Comparing Configurations

	Reliability	Expandability	Communication Overhead	Management	Data Consistency
Centralised	POOR Depends on central server.	POOR Single Server is limited by memory & storage maximums.	VERY HIGH Traffic heads to one centralised location.	EXCELLENT One very large site is easier to manage.	EXCELLENT All users always see the same data.
Replicated with Snapshots	GOOD Redundancy and tolerated delays in data synch.	VERY GOOD Cheap to scale up with new servers.	LOW to MEDIUM Intermittent bursts of network traffic (but not constant flooding of network).	VERY GOOD Each copy is alike.	MEDIUM Update delays are tolerable with snapshot catch ups for data consistency.
Synchronised Replication	EXCELLENT Redundancy and minimal delays.	VERY GOOD Low cost and only linear growth in synchronisation.	MEDIUM Constant messages to maintain synchronisation.	MEDIUM Data collusions need to be resolved and need good design and management.	VERY GOOD Close to precise consistency.
Integrated Partitions	GOOD Effective use of partitioning and redundancy.	VERY GOOD New nodes only get the data they need and no need to change DB design.	LOW to MEDIUM Most queries are local, but global queries to create temporary comms load.	DIFFICULT Distributed table updates require tight precise coordination.	VERY POOR Requires considerable effort and inconsistencies are not tolerated.
Decentralised Independent Partitions	GOOD Depends on local DB availability.	GOOD New sites are independent of all other sites.	LOW Little or no traffic needs to be communicated across the network.	VERY GOOD Easy – as each site is independent of the other sites and minimal need to share data.	LOW No guarantee of consistency – therefore high chance of consistency.

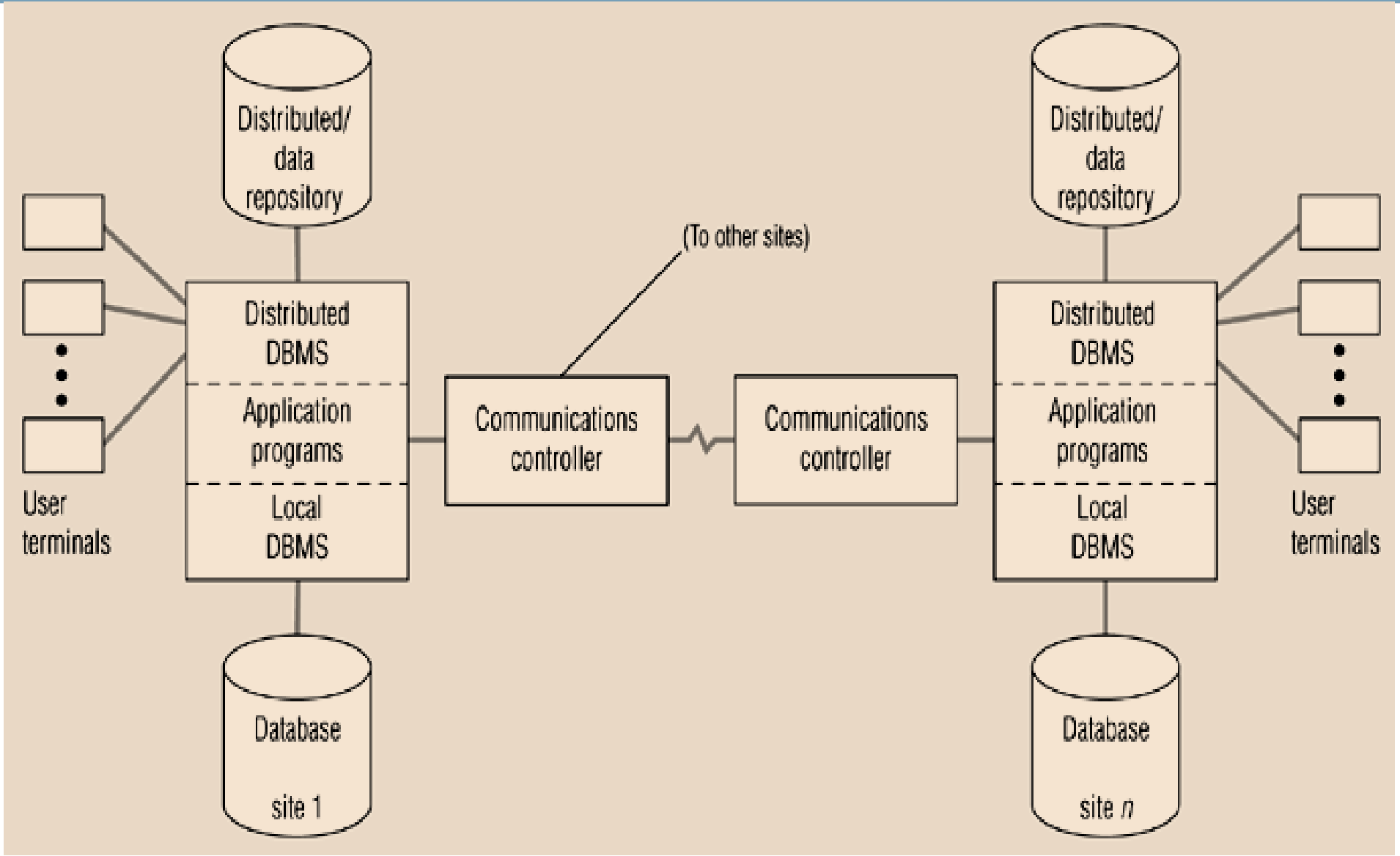
© University of Melbourne 2018

Functions of a distributed DBMS

- Locate data with a distributed data dictionary
- Determine location from which to retrieve data and process query components
- DBMS translation between nodes with different local DBMSs (using middleware)
- Data consistency (via multiphase commit protocols)
- Global primary key control
- Scalability
- Security, concurrency, query optimization, failure recovery

© University of Melbourne 2018

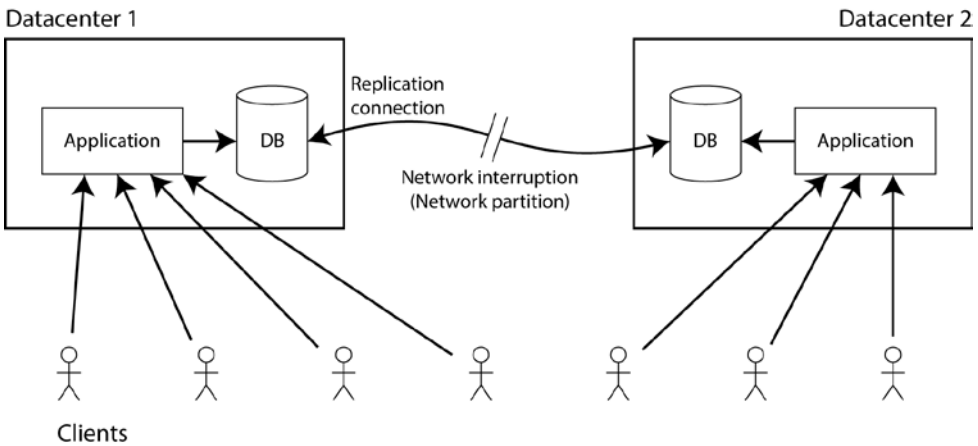
Distributed DBMS architecture



© University of Melbourne 2018

Network partitions

- Imagine you have a synchronously-updating, replicated database –
- Now imagine that the link between 2 nodes is interrupted



- What are your choices?
  - shut down the system (to avoid inconsistency)
  - keep it available to users (and accept inconsistency)
- More about this in the NoSQL lecture …

(diagram from https://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html)

© University of Melbourne 2018

Recent trends

- *Distributed* storage and processing is currently one of the most important research topics in database
- example use scenario: Internet startup
  - faces sudden increase in number and distribution of users
- see industry panel discussion
  - at *Google I/O 2013* (software developer conference)
  - https://youtu.be/zxwsOueJU4Q

© University of Melbourne 2018

Date's 12 Commandments for Distributed Databases

- Local Site Independence
  - Each local site can act as an independent, autonomous, centralised DBMS. Each site is responsible for security, concurrency, control, backup and recovery.
- Central Site Independence
  - No site in the network relies on a central site or any other site. All sites have the same capabilities.
- Failure Independence
  - The system is not affected by node failures. The system is in continuous operation even in the case of a node failure or an expansion of the network.
- Location Transparency
  - The user does not need to know the location of the data to retrieve those data.

© University of Melbourne 2018

Date's 12 Commandments for Distributed Databases

- Fragmentation Transparency
  - Data fragmentation is transparent to the user, who sees only one logical database. The user does not need to know the name of the database fragments to retrieve them.
- Replication Transparency
  - The user sees only one logical database. The DDBMS transparently selects the database fragment(s) to access. To the user, the DDBMS manages all fragments transparently.
- Distributed Query Processing
  - A distributed query may be executed at several different sites. Query optimization is performed transparently by the DDBMS.
- Distributed Transaction Processing
  - A transaction may update data at several different sites, and the transaction is executed transparently.

© University of Melbourne 2018

Date's 12 Commandments for Distributed Databases

- Hardware Independence
  - The system must run on any hardware platform.
- Operating System Independence
  - The system must run on any operating system platform.
- Network Independence
  - The system must run on any network platform.
- Database Independence
  - The system must support any vendor's database product

© University of Melbourne 2018

Week 08 Distributed Databases

Week 08 Transactions & Concurrency

David Eccles

INFO90002 Database Systems and Information Modelling

Today's Session…

- Why we need user-defined transactions
- Properties of transactions
- How to use transactions
- Concurrent access to data
- Locking and deadlocking
- Database recovery

  - Material for this lecture comes from
    - Hoffer et al. (2013) Modern Database Management 11th edition, chapters 7 and 11
    - Coronel et al. (2013) Database Systems: Design, Implementation and Management, Chapter 10.

Recall from last week …

- Example "business transactions":
  - Insert one row in *Order* table, then several in *OrderItem* table
  - Check amount < balance. If so, subtract amount from one row in bank account table, then add amount to another row
  - For all rows in Customer table, send out monthly statements
- Each requires several distinct database operations …

- e.g. move money from savings account to credit card account
  - Accept inputs from user (via ATM, internet banking or mobile app)
  - Select balance from savings account
  - Is there enough money to withdraw? If so:
    - Update savings account balance = balance – withdrawal
    - Update credit card balance = balance + withdrawal
  - If no errors encountered, end successfully

What is a (database) Transaction?

- A logical unit of work that must either be entirely completed or aborted (indivisible, atomic)
- DML & DDL statements are already atomic
- RDBMS also allows for *user-defined* transactions
- These are a sequence of DML statements, such as
  - a series of UPDATE statements to change values
  - a series of INSERT statements to add rows to tables
  - DELETE statements to remove rows
- Transactions will be treated as atomic
- A successful transaction changes the database from one consistent state to another
  - All data integrity constraints are satisfied

Why do we need Transactions?

- Transactions solve TWO problems:

1. users need the ability to define a unit of work

2. concurrent access to data by >1 user or program

(also acts as an “undo” for
manual database manipulation)

Problem 1: Unit of work

- Single DML or DDL command (implicit transaction)
  - e.g. Update 700 records,
    but database crashes after 200 records processed
  - Restart server: you will find no changes to any records
  - Changes are “all or none”
- Multiple statements (user-defined transaction)
  - START TRANSACTION; (or, ‘BEGIN’)
    - SQL statement;
    - SQL statement;
    - SQL statement;
    - …
  - COMMIT; (commits the whole transaction)
    - Or ROLLBACK (to undo everything)
- SQL keywords: BEGIN, COMMIT, ROLLBACK

Business transactions as units of work

- Each transaction consists of several SQL statements, embedded within a larger application program
- Transaction needs to be treated as an indivisible unit of work
- “Indivisible” means that either the whole job gets done, or none gets done: if an error occurs, we don’t leave the database with the job half done, in an inconsistent state

In the case of an error:

- Any SQL statements already completed must be reversed
- Show an error message to the user
- When ready, the user can try the transaction again
- This is briefly annoying – but inconsistent data is disastrous

Demo: Transaction as unit of work

- Demonstrate user-defined transactions using MySQL server. I encourage you to try this.

```
/* bank account transfer example */

/* check what is now in accounts*/
select * from Account;
select sum(balance) from Account;

start transaction;

/* transfer chosen amount from one account to another */
set @amount = 100;
update Account set balance = balance - @amount where id = 1;
update Account set balance = balance + @amount where id = 2;

commit;

/* check what is now in accounts*/
select * from Account;
select sum(balance) from Account;
```

sult Grid | Filter Rows: | Edit: | Export/Import:

Id	AccountType	Balance
1	savings	5000.00
2	credit	1000.00
NULL	NULL	NULL

Transaction Properties (ACID)

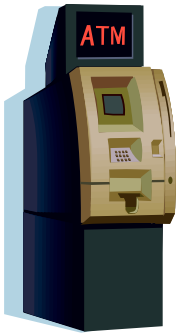
- Atomicity
  - A transaction is treated as a single, indivisible, logical unit of work. All operations in a transaction must be completed; if not, then the transaction is aborted
- Consistency
  - Constraints that hold before a transaction must also hold after it
  - (multiple users accessing the same data see the same value)
- Isolation
  - Changes made during execution of a transaction cannot be seen by other transactions until this one is completed
- Durability
  - When a transaction is complete, the changes made to the database are permanent, even if the system fails

Problem 2: Concurrent access

- What happens if we have multiple users accessing the database at the same time…
- Concurrent execution of DML against a shared database
- Note that the sharing of data among multiple users is where much of the benefit of databases derives – users communicate and collaborate via shared data
- But what could go wrong?
  - lost updates
  - uncommitted data
  - inconsistent retrievals

The Lost Update problem

Alice



Read account balance (balance = $1000)

Withdraw $100 (balance = $900)

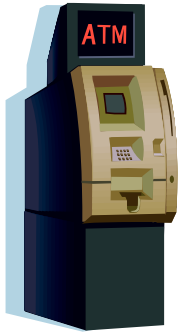
Write balance balance = $900

Time

t1a ***t2a*** ***t3a***

t1b ***t2b*** ***t3b***

Bob



Read account balance (balance = $1000)

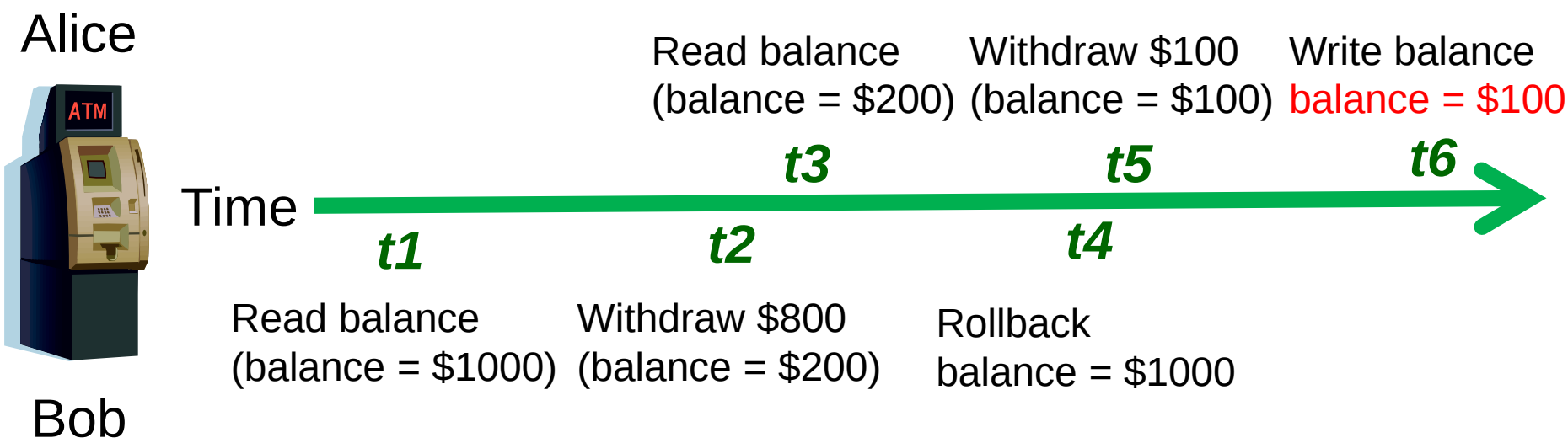
Withdraw $800 (balance = $200)

Write balance balance = $200

Balance should be $100

The Uncommitted Data problem

- Uncommitted data occurs when two transactions execute concurrently and the first is rolled back after the second has already accessed the uncommitted data



Balance should be $900

The Inconsistent Retrieval problem

- Occurs when one transaction calculates some aggregate functions over a set of data, while other transactions are updating the data
  - Some data may be read after they are changed and some before they are changed, yielding inconsistent results

Alice	Bob
SELECT SUM(Salary) FROM Employee;	UPDATE Employee SET Salary = Salary * 1.01 WHERE EmpID = 33;
	UPDATE Employee SET Salary = Salary * 1.01 WHERE EmpID = 44;
(finishes calculating sum)	COMMIT;

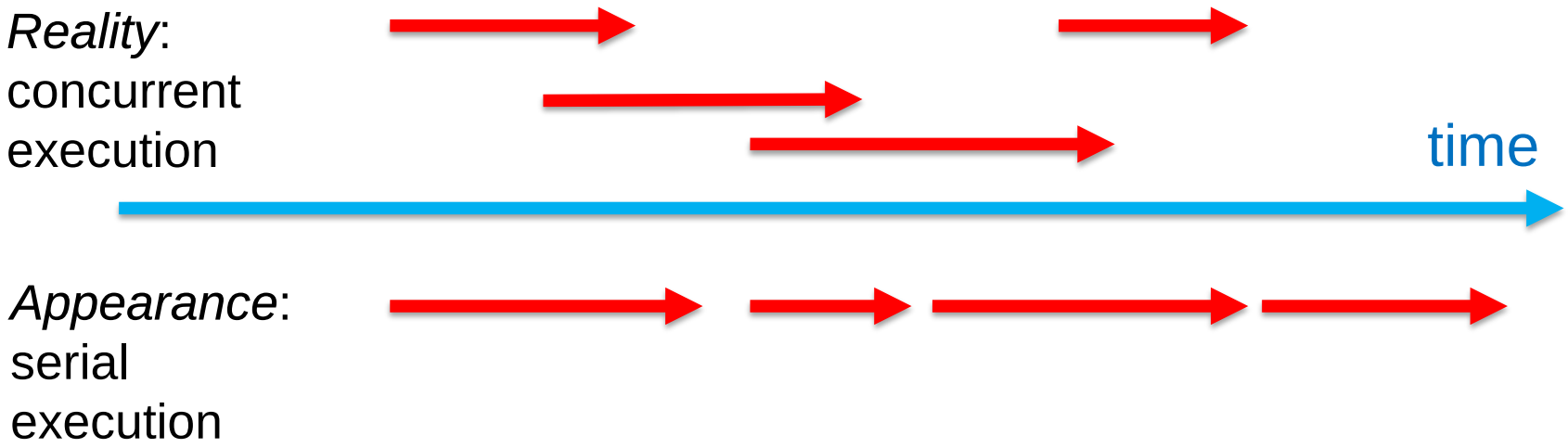
Example: Inconsistent Retrieval

Time	Trans-action	Action	Value	T1 SUM	Comment
1	T1	Read Salary for EmpID 11	10,000	10,000	
2	T1	Read Salary for EmpID 22	20,000	30,000	
3	T2	Read Salary for EmpID 33	30,000		
4	T2	Salary = Salary * 1.01			
5	T2	Write Salary for EmpID 33	30,300		
6	T1	Read Salary for EmpID 33	30,300	60,300	*after* update
7	T1	Read Salary for EmpID 44	40,000	100,300	*before* update
8	T2	Read Salary for EmpID 44	40,000		
9	T2	Salary = Salary * 1.01			
10	T2	Write Salary for EmpID 44	40,400		
11	T2	COMMIT			
12	T1	Read Salary for EmpID 55	50,000	150,300	
13	T1	Read Salary for EmpID 66	60,000	210,300	

we want either
before $210,000 or
after $210,700

Serializability

- Transactions ideally are "serializable"
  - Multiple, concurrent transactions *appear as if* they were executed one after another
  - Ensures that the concurrent execution of several transactions yields consistent results



but true serial execution (i.e. no concurrency) is very expensive!

Concurrency control methods

- To achieve efficient execution of transactions, the DBMS creates a schedule of read and write operations for concurrent transactions
- Interleaves the execution of operations, based on concurrency control algorithms such as locking and time stamping
- Several methods of concurrency control
  - *Locking* is the main method used
  - Alternate methods
    - Time Stamping
    - Optimistic Methods

Concurrency Control with Locking

- Lock
  - Guarantees exclusive use of a data item to a current transaction
    - T1 acquires a lock prior to data access; the lock is released when the transaction is complete
    - T2 does not have access to data item currently being used by T1
    - T2 has to wait until T1 releases the lock
  - Required to prevent another transaction from reading inconsistent data
- Lock manager
  - Responsible for assigning and policing the locks used by the transactions
- Question: at what granularity should we apply locks?

Lock Granularity: options

- Database-level lock
  - Entire database is locked
  - Good for batch processing but unsuitable for multi-user DBMSs
  - T1 and T2 can not access the same database concurrently even if they use different tables
  - (SQLite, Access)
- Table-level lock
  - Entire table is locked - as above but not quite as bad
  - T1 and T2 can access the same database concurrently as long as they use different tables
  - Can cause bottlenecks, even if transactions want to access different parts of the table and would not interfere with each other
  - Not suitable for highly multi-user DBMSs

Lock Granularity: options

- Page-level lock
  - An entire disk page is locked (a table can span several pages and each page can contain several rows of one or more tables)
  - Not commonly used now
- Row-level lock
  - Allows concurrent transactions to access different rows of the same table, even if the rows are located on the same page
  - Improves data availability but with high overhead (each row has a lock that must be read and written to)
  - Currently the most popular approach (MySQL, Oracle)
- Field-level lock
  - Allows concurrent transactions to access the same row, as long as they access different attributes within that row
  - Most flexible lock but requires an extremely high level of overhead
  - Not commonly used

Types of Locks

- Binary Locks (1 or 0)
  - has only two states: locked (1) or unlocked (0)
  - eliminates “Lost Update” problem
    - the lock is not released until the statement is completed
  - considered too restrictive to yield optimal concurrency, as it locks even for two READs (when no update is being done)
- The alternative is to allow both Exclusive and Shared locks
  - often called Read and Write locks

Exclusive and Shared Locks

- Exclusive lock (X)
  - access is reserved for the transaction that locked the object
  - must be used when transaction intends to WRITE
  - granted if and only if no other locks are held on the data item
  - in MySQL: “select … for update”

- Shared lock (S)
  - other transactions are also granted Read access
  - issued when a transaction wants to READ data, and no Exclusive lock is held on that data item
    - multiple transactions can each have a shared lock on the same data item if they are all just reading it
  - in MySQL: “select … lock in share mode”

Demo: Concurrent execution

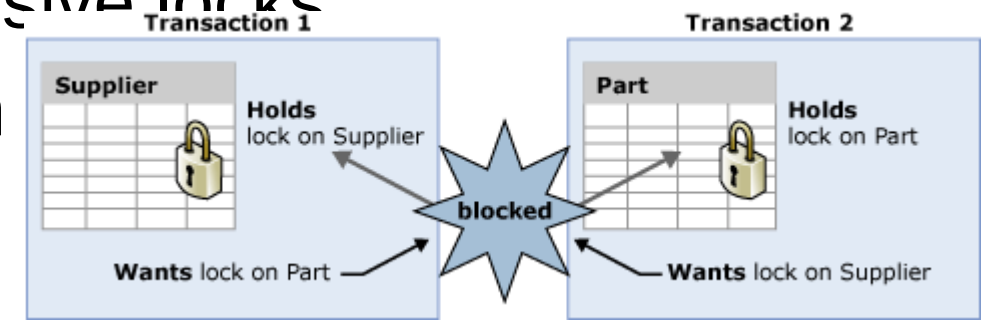
- Demonstrates locking and concurrency with transactions.
- I encourage you to try this. You need to start two command-prompt windows, and connect both to the same MySQL server. This simulates two applications concurrently accessing the same data.

```
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>mysql -u root test
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 20
Server version: 5.6.21 MySQL Community Server (GPL)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All

Oracle is a registered trademark of Oracle Corporation and/o
affiliates. Other names may be trademarks of their respectiv
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the curren

mysql> select * from Account;
```

```
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>
H:\>mysql -u root test
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 21
Server version: 5.6.21 MySQL Community Server (GPL)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All

Oracle is a registered trademark of Oracle Corporation and/
affiliates. Other names may be trademarks of their respecti
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the curre

mysql> select * from Account;
```

Deadlock

- Condition that occurs when two transactions wait for each other to unlock data
  - T1 locks data item X, then wants Y
  - T2 locks data item Y, then wants X
  - each waits to get a data item which the other transaction is already holding
  - could wait forever if not dealt with
- Only happens with exclusive locks
- Deadlocks are dealt with
  - prevention
  - detection
  - (we won't go into the details of how in this course)



Alternative concurrency control methods

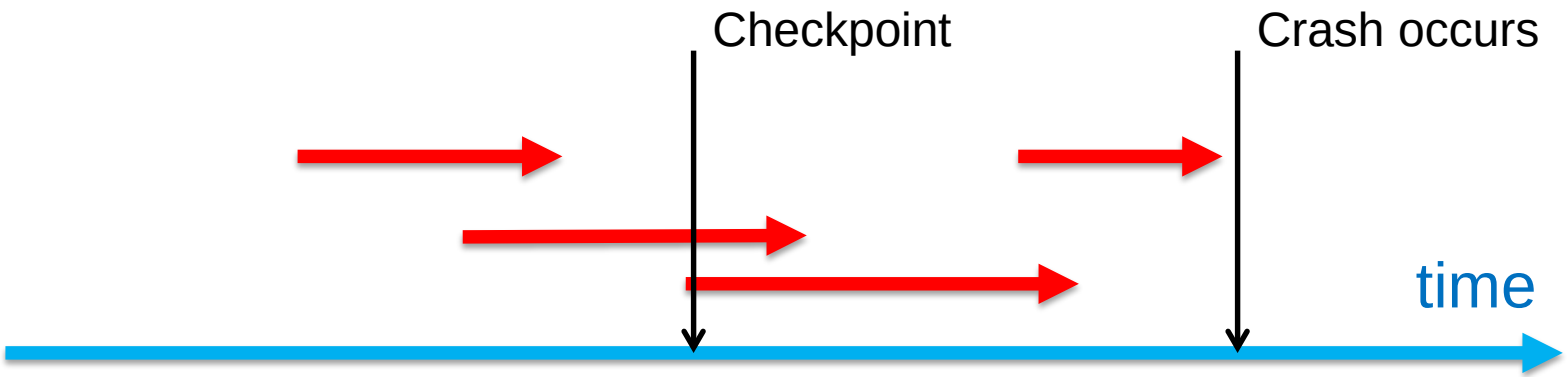
- Timestamp
  - Assigns a global unique timestamp to each transaction
  - Each data item accessed by the transaction gets the timestamp
  - Thus for every data item, the DBMS knows which transaction performed the last read or write on it
  - When a transaction wants to read or write, the DBMS compares its timestamp with the timestamps already attached to the item and decides whether to allow access
- Optimistic
  - Based on the assumption that the majority of database operations *do not* conflict
  - Transaction is executed without restrictions or checking
  - Then when it is ready to commit, the DBMS checks whether it any of the data it read has been altered – if so, rollback

Logging transactions

- Want to restore database to a previous consistent state
- If transaction cannot be completed, it must be aborted and any changes rolled back
- To enable this, DBMS tracks all updates to data
- This *transaction log* contains:
  - a record for the beginning of the transaction
  - for each SQL statement
    - operation being performed (update, delete, insert)
    - objects affected by the transaction
    - “before” and “after” values for updated fields
    - pointers to previous and next transaction log entries
  - the ending (COMMIT) of the transaction

Transaction log

- Also provides the ability to restore a corrupted database
- If a system failure occurs, the DBMS will examine the log for all uncommitted or incomplete transactions and it will restore the database to a previous state



© David Eccles, University of Melbourne 2018

Example transaction log

TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	****Start Transaction				
352	101	341	363	UPDATE	PRODUCT	54778-2T	PROD_QOH	45	43
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	615.73	675.62
365	101	363	Null	COMMIT	**** End of Transaction				
397	106	Null	405	START	****Start Transaction				
405	106	397	415	INSERT	INVOICE	1009			1009,10016, ...
415	106	405	419	INSERT	LINE	1009,1			1009,1, 89-WRE-Q,1, ...
419	106	415	427	UPDATE	PRODUCT	89-WRE-Q	PROD_QOH	12	11
423				CHECKPOINT					
427	106	419	431	UPDATE	CUSTOMER	10016	CUST_BALANCE	0.00	277.55
431	106	427	457	INSERT	ACCT_TRANSACTION	10007			1007,18-JAN-2004, ...
457	106	431	Null	COMMIT	**** End of Transaction				
521	155	Null	525	START	****Start Transaction				
525	155	521	528	UPDATE	PRODUCT	2232/QWE	PROD_QOH	6	26
528	155	525	Null	COMMIT	**** End of Transaction				
* * * * * C *R*A* S* H * * * *									

Dr Greg Wadley &
David Eccles

THE UNIVERSITY OF MELBOURNE

INFO90002 Database Systems & Information Modelling

Week 09

Database Administration

News

- Three hour lecture today – no more labs
- Hour 1 Database Administration
- Hour 2 Database Security, Ethics and Backu
- Hour 3 Assignment 1 Feedback

© University of Melbourne 2017

Today's Session…

HOUR 1:

- The 'Database Administrator' role
- Architecture – Understanding the DBMS
  - concepts
- Performance improvement
  - concepts
  - common approaches e.g. indexes

HOUR 2

- Security
  - requirements
  - web apps and SQL injection
- Ethics
- Backup & Recovery

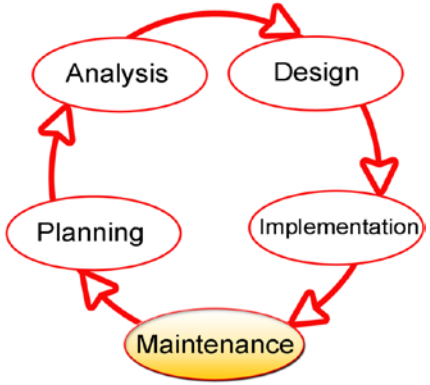
HOUR 3

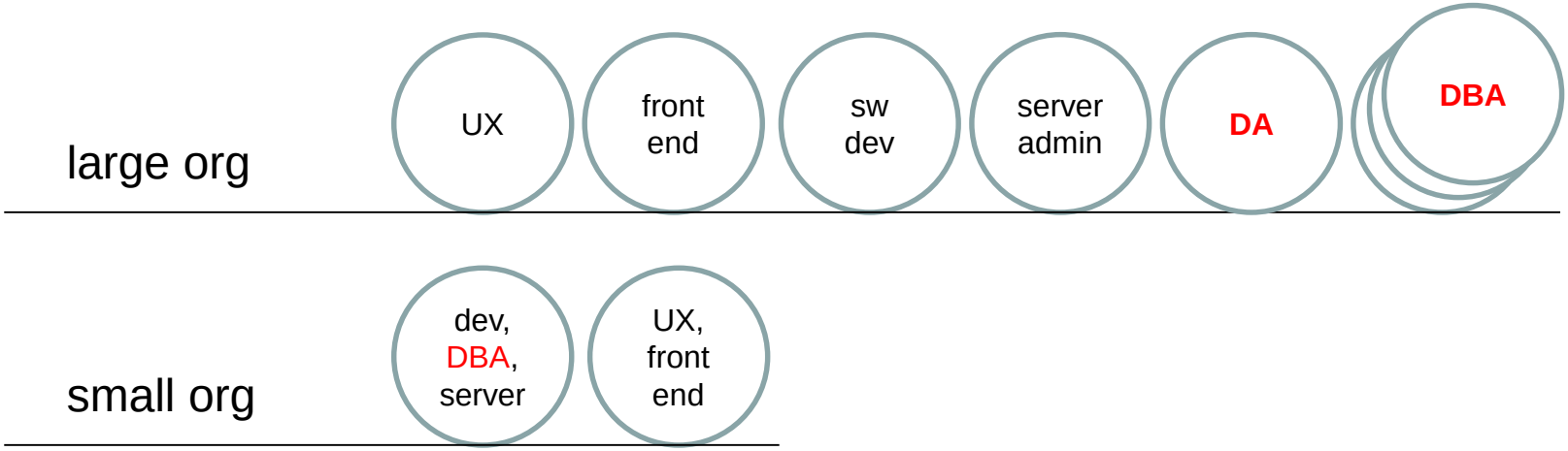
- Assignment 1 Feedback

© University of Melbourne 2017

The DBA role

- primarily concerned with “maintenance” / “ops” phase
- but should be consulted during all phases of development
- “Database Administrator” or “DBA”
- often framed as a “job” or a “person”
- Large companies – many DBA’s
- Small company developer is the DBA
- DBA role can be made redundant by Cloud-based DBMS or “database as a service” DAAS (often IAAS or PAAS)

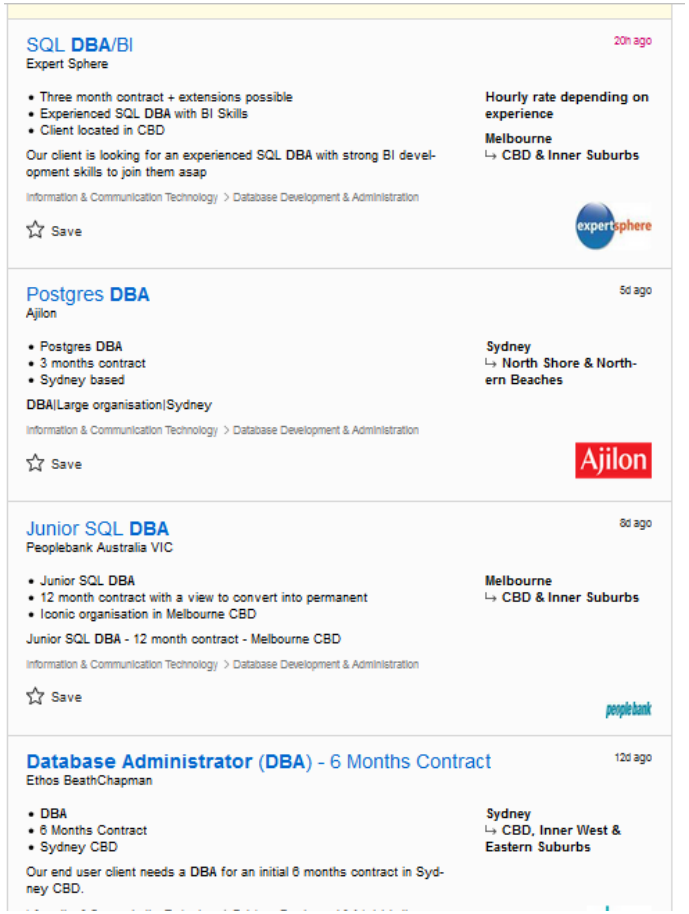




© University of Melbourne 2017

Data and Database Administration

- Data Administrator (management role)
  - data policies, procedures and standards
  - planning
  - data conflict resolution
  - managing info repository
  - internal marketing
  - similar to “Chief Data Officer”
- Database Administrator (technical role)
  - analyze and design DB
  - select DBMS / tools / vendor
  - install and upgrade DBMS
  - tune DBMS performance
  - manage security, privacy, integrity
  - backup and recovery



(Hoffer et al., chapter 11)

© University of Melbourne 2017

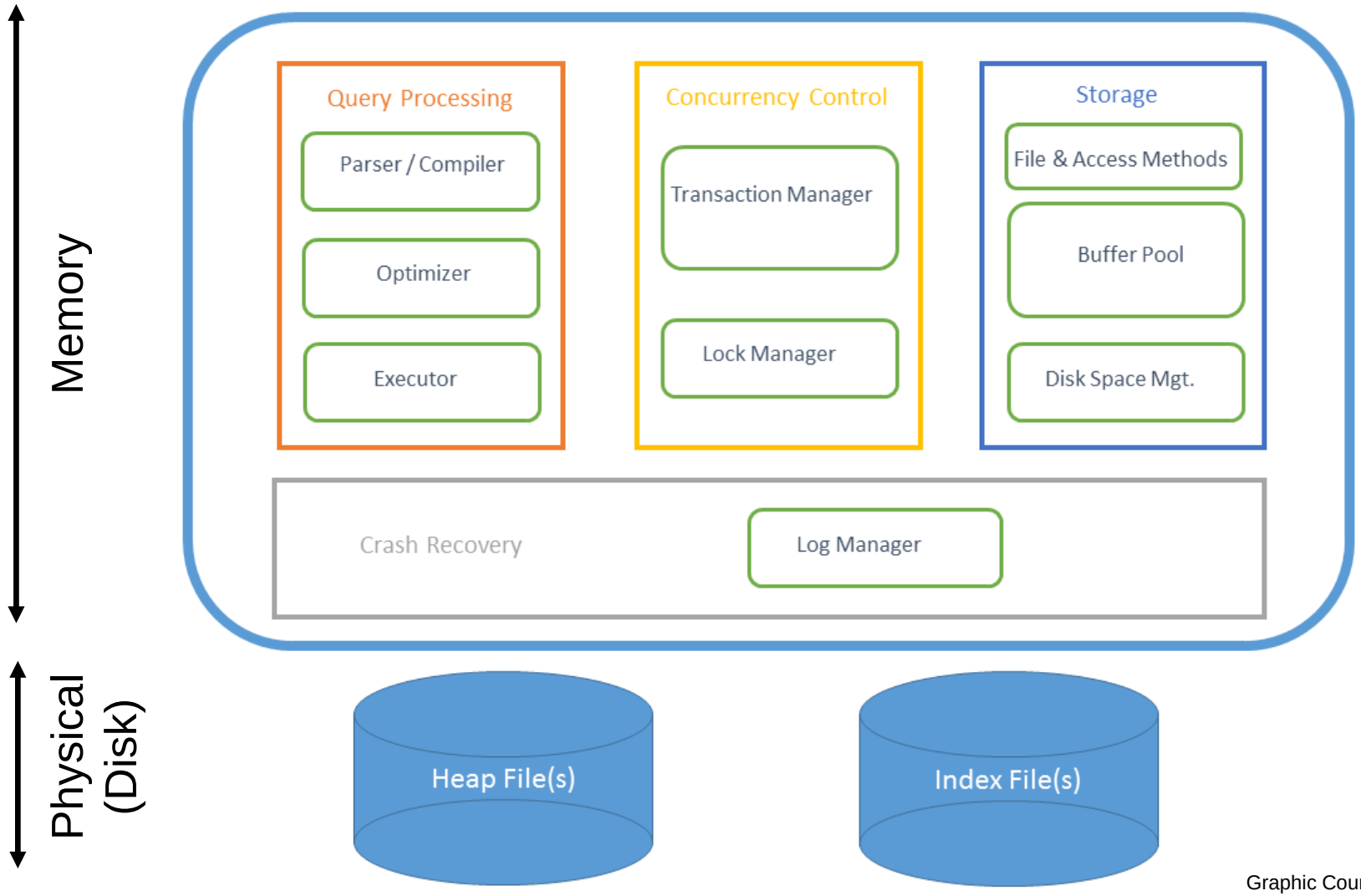
Architecture of a Database Management System (DBMS)

Database Systems Architecture

- A Database Management System (DBMS)
- Exists as one entity in two places
  - In Memory
  - Physically on disk
- Both places
  - Manage Data (the reason we have the DBMS)
  - Manage Performance (how it performs as it is used & grows)
  - Manage Concurrency (manages high volumes of users)
  - Manage Recoverability (assist in recovery and availability)
- One place is persistent the other transient
  - Disk representation is always present
  - Memory – transient – only exists when DBMS is running

© University of Melbourne 2017

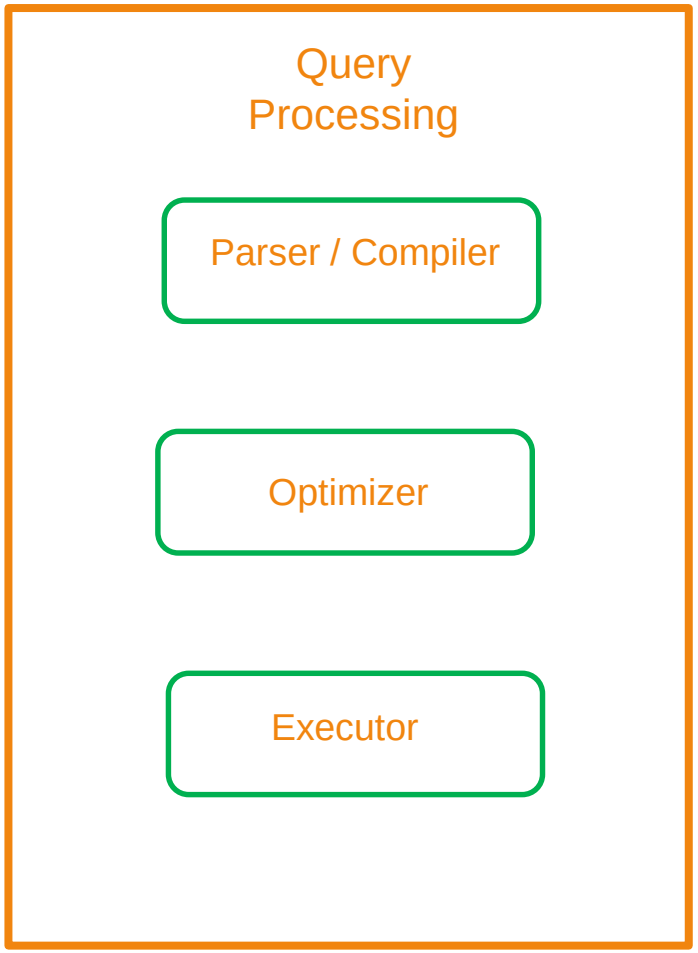
DBMS Overview



Graphic Courtesy of
Dr Renata Borovica-Gajic

© University of Melbourne 2017

Query Processing



Parsing

- Syntax is correct & can “compile”
- DBMS User Permissions
- Resources (Data, Code, be able to Record Changes/Retrieve results)

Optimizing

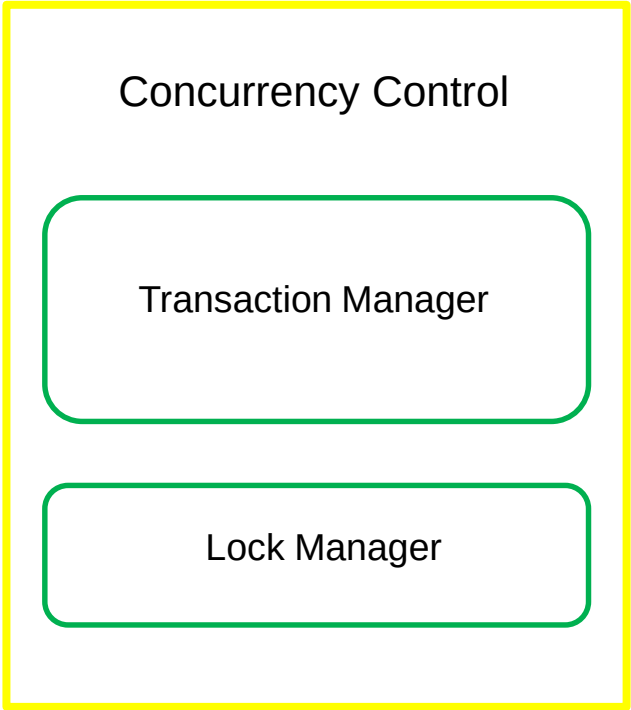
- Execution Plan & Execution Cost
- Evaluate indexes, table scans, hashing
- Eliminate worst, consider best options
- Lowest cost theoretically “best”

Execution

- Meet the ACID test,
- Atomic: All rows succeed or all fail
- Ensure resources are available
  - Data, Log changes, Memory, Cursor to do the work for the USER

© University of Melbourne 2017

Concurrency Control



- Manages the work of the DBMS.
- Transaction Manager handles all aspects of the SQL transaction - which DBMS user wants WHAT resource
- Lock Manager is a list of what resources are locked and by which user at what level
- not only tables, indexes
  - buffers, cursor, memory addresses of resources

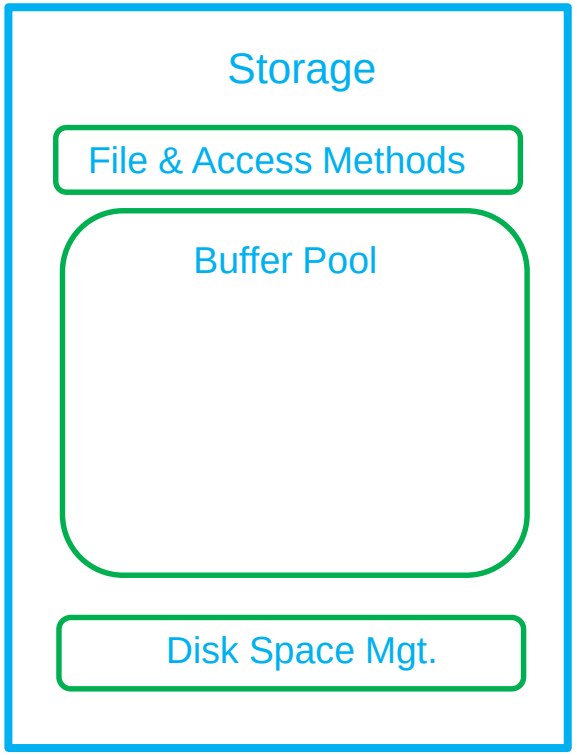
© University of Melbourne 2017

Concurrency Control

- Essential to manage large scalable DBMS
- Enables 100,000s of concurrent users
- Like a Traffic Policemen controlling the flow of traffic
  - Who can do what (allowed to do what they need to do)
  - Who has to wait (queue)
  - Who can travel through the intersection concurrently

© University of Melbourne 2017

Storage

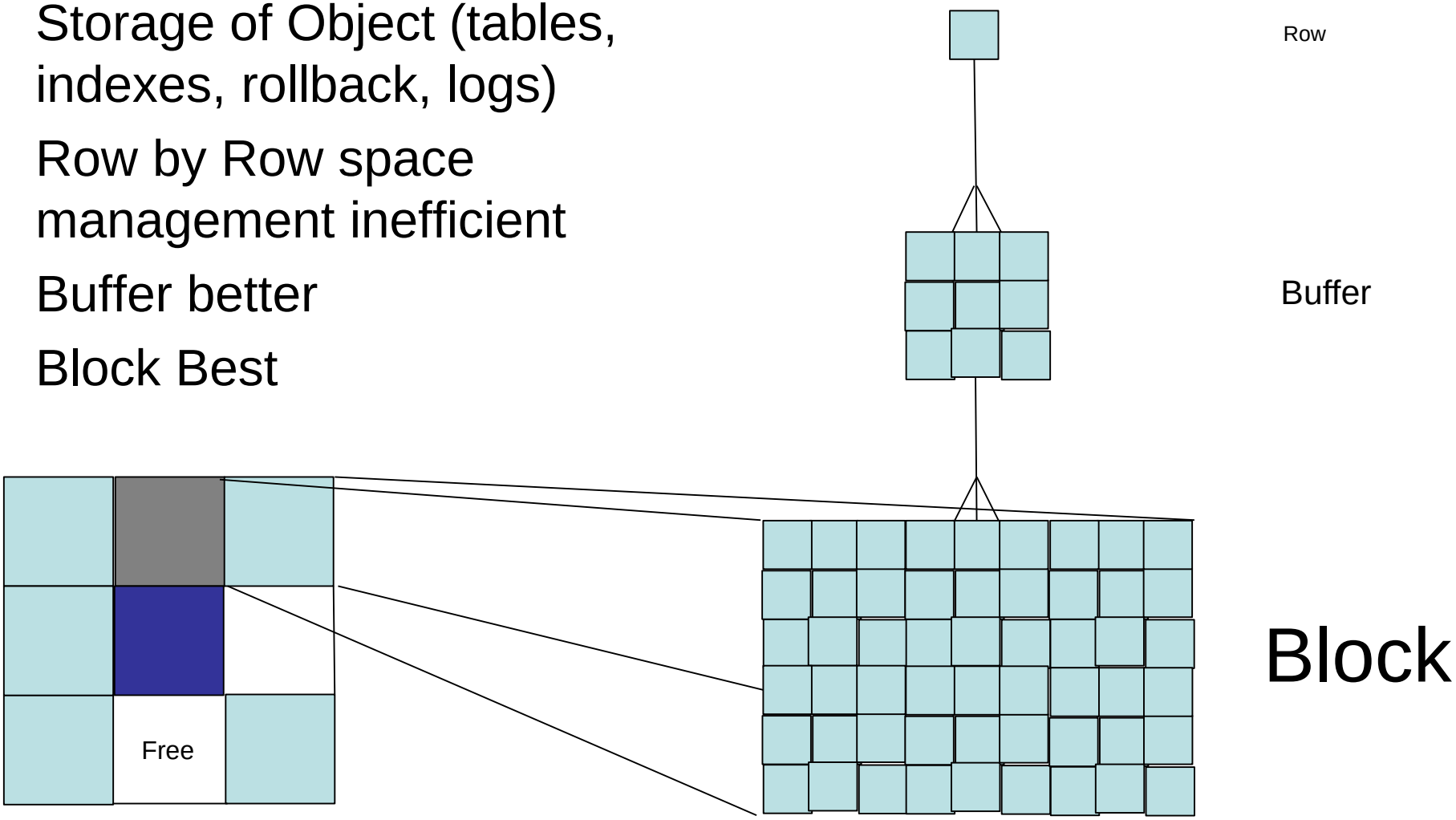


- File & Access Methods
  - Disk to Memory to Disk
  - Read a buffer or a block of buffers
- Buffer Pool
  - Data in memory
    - Row data
    - Index data
  - Organised
- Disk Space Management
  - How to organise growth of data on disk efficiently by writing efficiently.

© University of Melbourne 2017

Storage – Disk Space Management

- Hierarchical Structure
- Storage of Object (tables, indexes, rollback, logs)
- Row by Row space management inefficient
- Buffer better
- Block Best



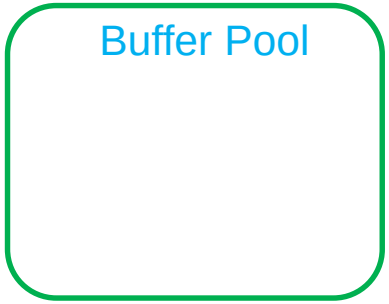
© University of Melbourne 2017

Buffer Pool

- Many Object Types (Tables, Indexes, Undo)
- Each buffer contains rows, btree leaf etc.
- Each buffer can have one of four status types:
  - Current
    - In use current committed version of data (row)
  - Active
    - Most recent change (may not be committed)

COMMIT (Current: DepartmentID=9)

  - Stale
    - An old version of the data
  - Aged
    - Old and about to be removed from buffer pool



DepartartmentID=8

DepartartmentID=9

DepartmentID=8

DepartmentID=8

© University of Melbourne 2017

Log Manager

- Recovery
- Log Manager records ALL changes
  - Statement
  - Transaction
    - Statement
    - Rollback values
    - Before and After values
    - Timestamp begin
      - txn, savepoint & commit
  - Database
    - Data Dictionary Changes

© University of Melbourne 2017

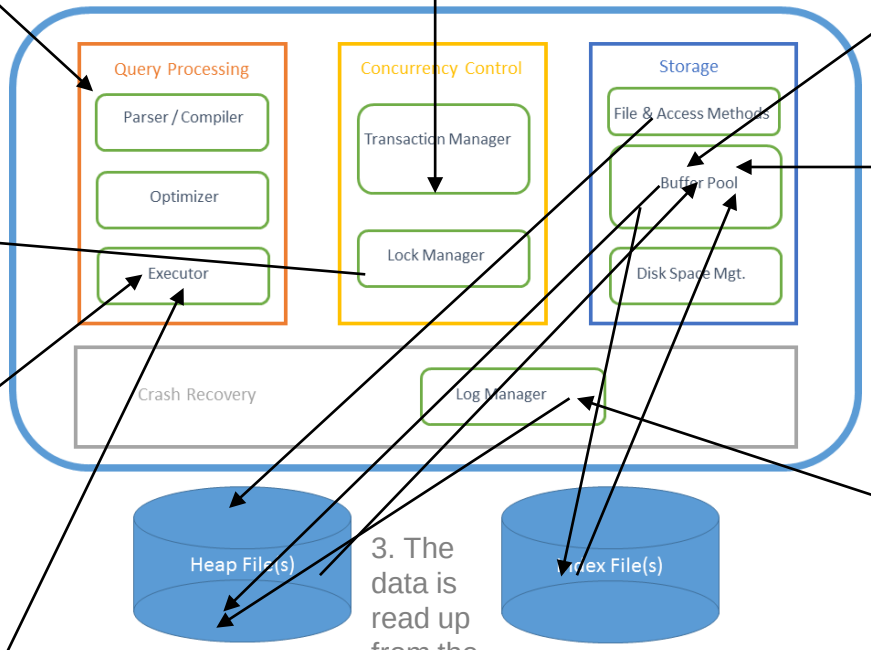
DBMS

```
UPDATE employee
SET departmentid = 9
WHERE firstname = 'Nancy'
AND departmentid = 8;
```

1. The code is parsed & optimized

4. The Concurrency manager ensures all the tables and index data is in the buffer pool and ready to be used in the execution of the SQL.

6a In the buffer pool we record the before value the after value and the change to the value



2. We check if the table & index data (rows) are in the buffer pool. If it is we skip to 4. If not we go to 3.

5a When the rows & indexes are locked we execute the query

5b. The SQL is executed

3. The data is read up from the table & index on disk

6b We record a timestamp and the before value, the after value and the change to the value for all transactions in the Log Manager

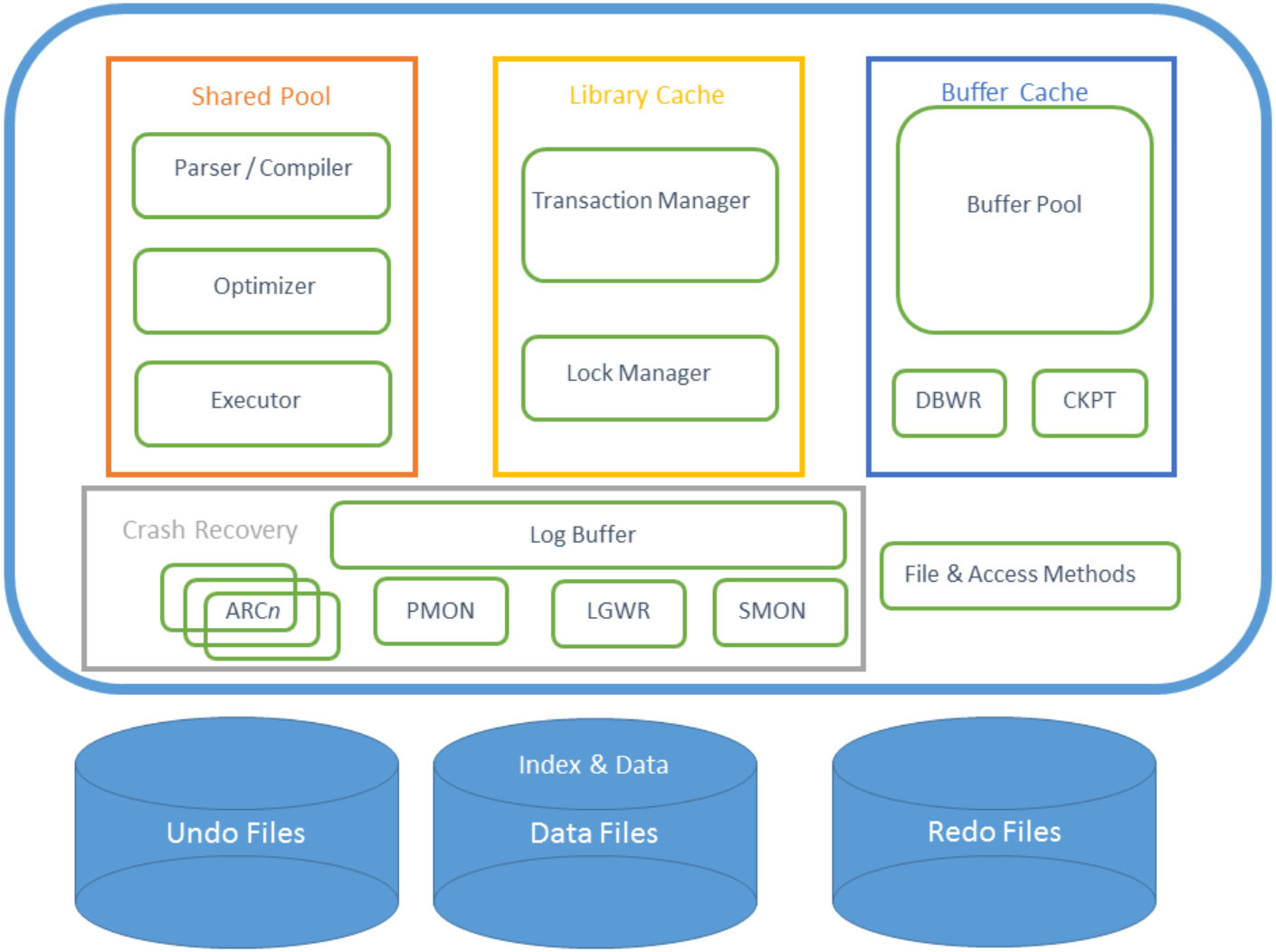
7. The user commits the change to make it permanent

8. The changed data and transaction log are written to disk

Graphic Courtesy of
Dr Renata Borovica-Gajic

© University of Melbourne 2017

How Oracle* DBMS looks



* This slide is not examinable

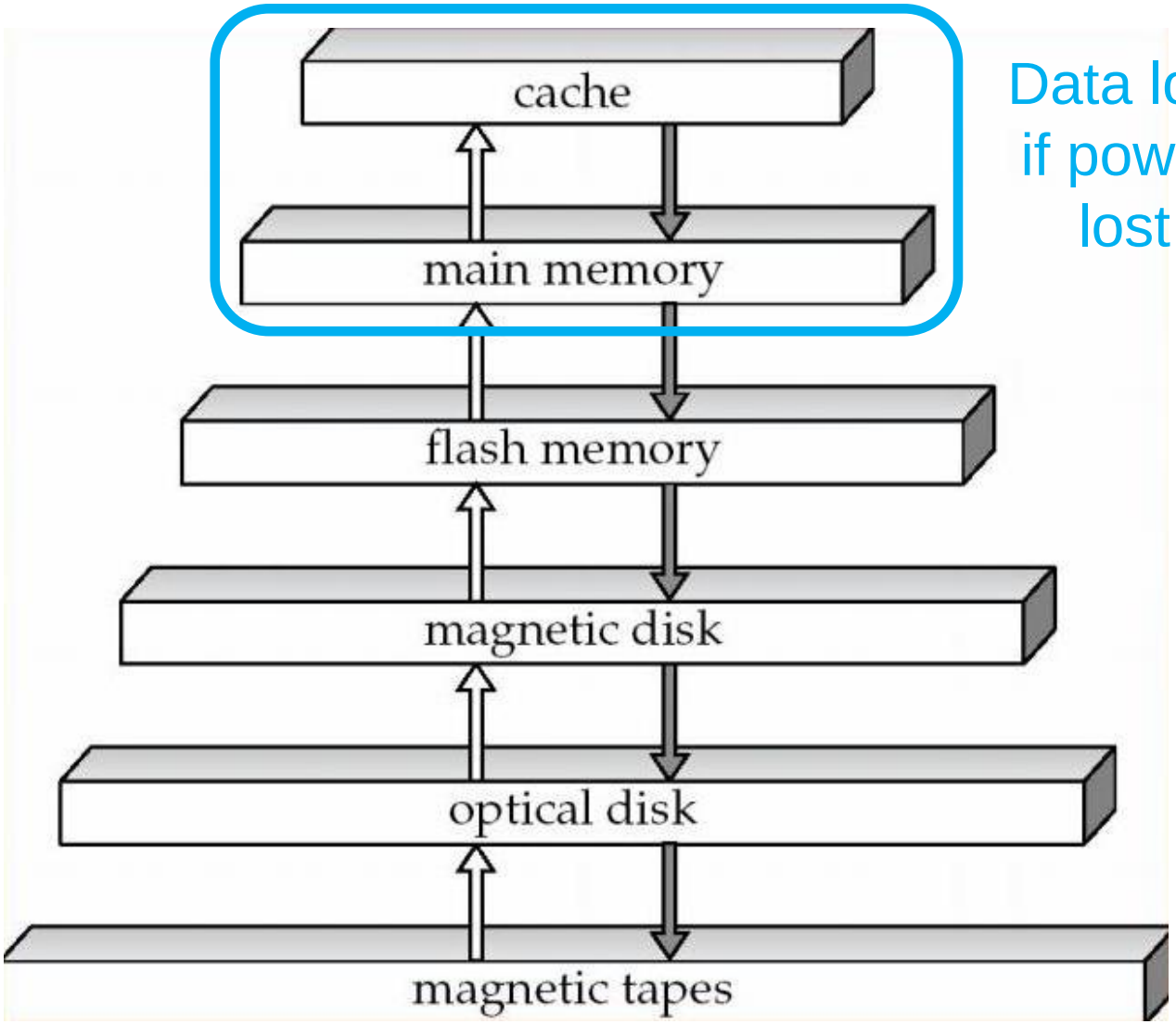
© University of Melbourne 2017

Database Performance

Storage media hierarchy

faster, more expensive, smaller capacity

slower, cheaper, *older*, bigger capacity



Data lost if power lost

© University of Melbourne 2017

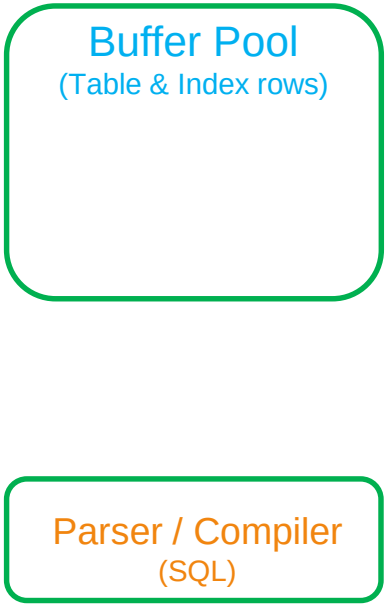
What affects database performance?

- caching data in memory, e.g. data buffers
- placement of data files across disc drives
- use of fast storage such as SSD
- database replication and server clustering
- use of indexes to speed up searches and joins
- good choice of data types (especially PKs)
- good program logic
- good query execution plans
- good code (no deadlocks)

© University of Melbourne 2017

Caching Data in Memory

- Data and Code found in memory
- Avoids a read
- Reads are expensive
- Goal in to minimize reads (& writes)
  - Writes are necessary (recovery logs, changed data)
- "in memory databases"
  - all code all data loaded into memory on db start & stays until shutdown



© University of Melbourne 2017

Data file location & Fast Disks (SSD)

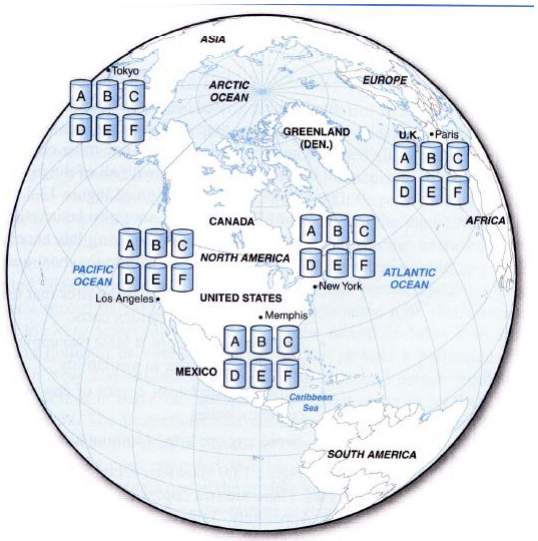
- Spread the files across the physical server
- We can't avoid writes
  - Spread files across many disks
    - Avoid contention
      - (many users competing for same resource)
  - Recovery Logs (always writing)
    - faster disk
- SSD (Solid State Drives)
  - No moving parts – nothing to break down
  - Faster I/O (Input & Output compared to other disk types)

© University of Melbourne 2017

Distribution & Replication

- Distributed data
  - Spreads the load
  - Data kept only where it is needed
  - Less work per physical server – faster response times

- Replicated Data
  - Spreads Load
  - Less work per physical server – faster response times



© University of Melbourne 2017

When to create indexes

- for each table, choose the columns you will index:
- column is *queried frequently* (used in Where clauses)
- columns that are used for *joins (PK to FK)*
- primary *keys* (automatic in most DBMS)
- foreign *keys* (automatic in MySQL)
- unique columns (automatic in most DBMS)
- large tables only - small tables do not require indexes
  - if you frequently retrieve less than about 15% of the rows
- wide range of values (good for regular indexes).
- small range of values (good for bitmap/hash indexes).

covered already in week 6

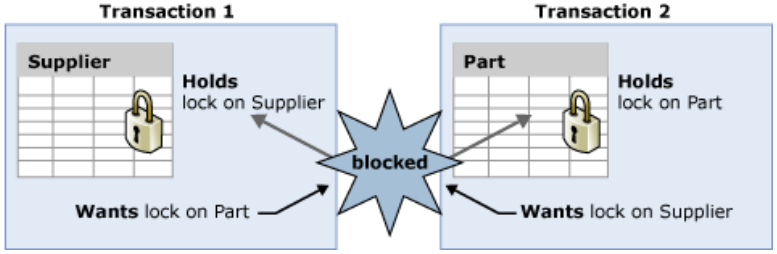
source: Oracle® Database Application Developer's Guide

© University of Melbourne 2017

Good Choices

- Good Data Types
  - INTEGERS for PK FK & PFK
- Good Program Logic & Code
  - Transaction design

    BEGIN TRANSACTION
    SELECT
    UPDATE
    UPDATE
    UPDATE
    COMMIT
  - Avoid Long Complex Transactions that never commit or savepoint
  - Avoid coding deadlocks!
  - Appropriate Locking strategy
    - Row b4 Buffer b4 Table b4 Database
    - Consider Lock Timeouts (if not automatic)



Deadlock Scenario

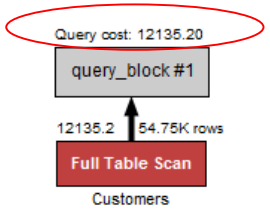
© University of Melbourne 2017

Good Execution Plans

- The best execution plan has the lowest “cost”
- Known as Cost Based Optimization (CBO)

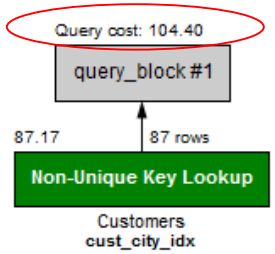
```
SELECT cust_first_name, cust_last_name, cust_marital_status,
cust_city, cust_income_level
FROM Customers
WHERE cust_last_name = 'Parkburg'
AND cust_first_name = 'Peter'
AND cust_city = 'Trafford';
```

	cust_first_name	cust_last_name	cust_marital_status	cust_city	cust_income_level
	Peter	Parkburg	single	Trafford	H: 150,000 - 169,999



Index on where condition (cust_city) improves cost:

```
CREATE INDEX cust_city_idx
ON customers(cust_city);

SELECT cust_first_name, cust_last_name, cust_marital_status,
cust_city, cust_income_level
FROM Customers
WHERE cust_last_name = 'Parkburg'
AND cust_first_name = 'Peter'
AND cust_city = 'Trafford';
```



© University of Melbourne 2017

Break Time (10 minutes)

9 Database Security

David Eccles & Dr Greg Wadley

INFO90002

Database Systems &

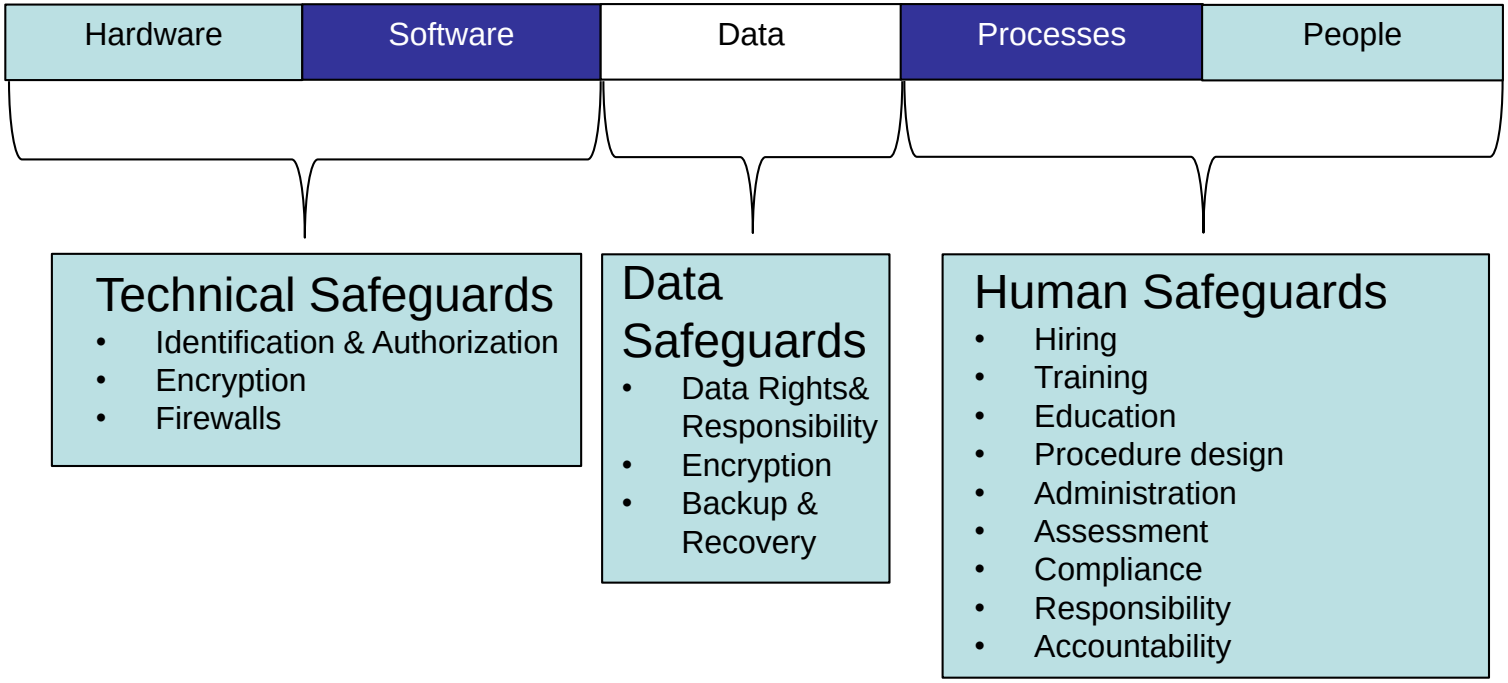
Information Modelling

Database Security

- Legal and Policy issues
- Ethics and Privacy issues
- Access and Control issues

Database Security

- Five Component Framework (5CF) Approach:



Rights & Responsibilities

- Professional Codes of Conduct
- Ethical & Moral considerations
- Privacy
- Legislative requirements
- Company Policy requirements

Professional Codes of Conduct

AUSTRALIAN COMPUTER SOCIETY

ACS CODE OF ETHICS

The ACS Code of Ethics is part of the ACS Constitution. As an ACS member you must uphold and advance the honour, dignity and effectiveness of being a professional. This entails, in addition to being a good citizen and acting within the law, your adherence to the following Society values:

1 The Primacy of the Public Interest
You will place the interests of the public above those of personal, business or sectional interests.

2 The Enhancement of Quality of Life
You will strive to enhance the quality of life of those affected by your work.

3 Honesty
You will be honest in your representation of skills, knowledge, services and products.

4 Competence
You will work competently and diligently for your stakeholders.

5 Professional Development
You will enhance your own professional development, and that of your colleagues and staff.

6 Professionalism
You will enhance the integrity of the Society and the respect of its members for each other.

This Code of Ethics applies to all ACS members regardless of their role or specific area of expertise in the ICT industry.

- Promises by professions to regulate themselves in the general interest of society
- Where we don't have clear laws we need ethical behaviours to ensure social harmony

Ethical & Moral considerations

- Do we always have the right to access information?
- You can apply ethical principles:
  - Immanuel Kant's Categorical Imperative
    - "If an action is not right for everyone to take then it is not right for anyone to take"
  - Risk aversion principle:

    "Take the action that produces the least harm or incurs the least cost"

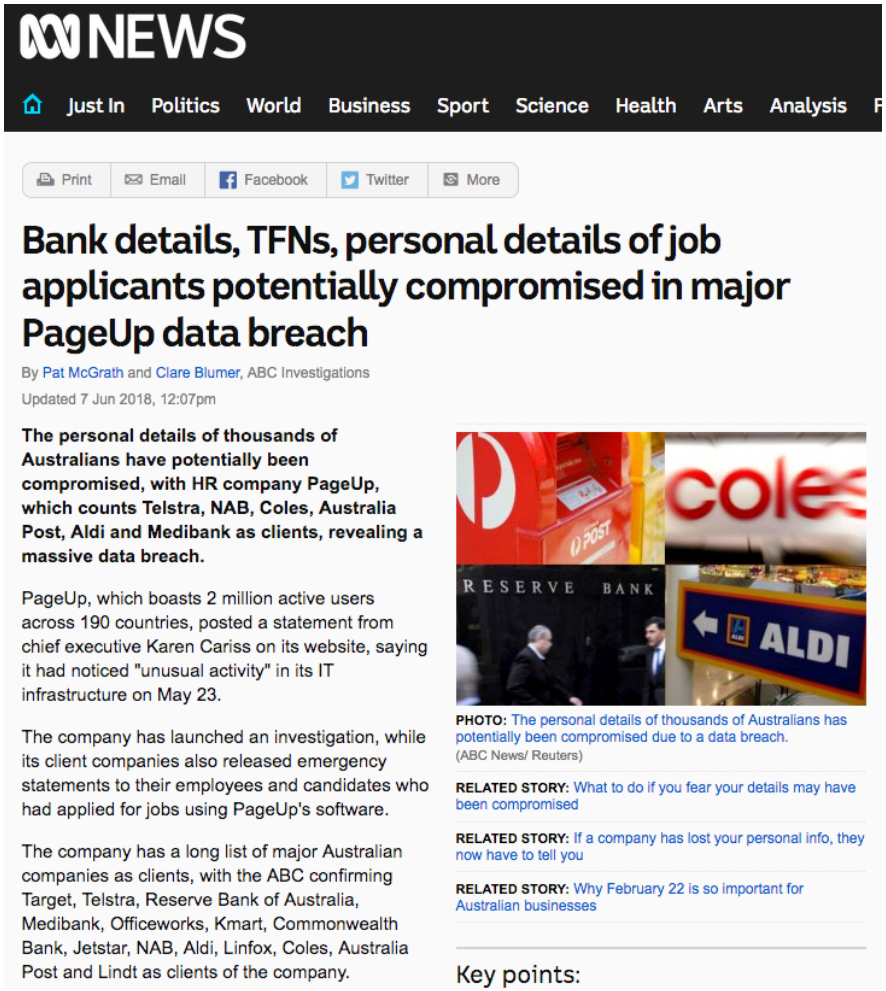
Laws & Policies

- Privacy and Data protection act 2014 (Vic Gov)
- Privacy Act 1988 (Commonwealth of Australia)
- Privacy Amendment (Private Sector) Act 2000
- Health Records Act 2001 (Vic Gov)
- Privacy Act of 1988 has been reformed to include larger fines for “repeated and serious breaches of privacy regulations”.
- University of Melbourne Privacy Policy:

  Privacy Policy http://policy.unimelb.edu.au/MPF1104

Data Responsibilities

- Page Up Data leak
- To apply for jobs
  - Proof of Australian Citizenship (Passport)
  - Proof have a current drivers license (Victorian Drivers License)
  - Date of Birth
- Job applicants have had sensitive private information leaked



ABC NEWS

Just In | Politics | World | Business | Sport | Science | Health | Arts | Analysis

Print | Email | Facebook | Twitter | More

Bank details, TFNs, personal details of job applicants potentially compromised in major PageUp data breach

By Pat McGrath and Clare Blumer, ABC Investigations

Updated 7 Jun 2018, 12:07pm

The personal details of thousands of Australians have potentially been compromised, with HR company PageUp, which counts Telstra, NAB, Coles, Australia Post, Aldi and Medibank as clients, revealing a massive data breach.

PageUp, which boasts 2 million active users across 190 countries, posted a statement from chief executive Karen Cariss on its website, saying it had noticed "unusual activity" in its IT infrastructure on May 23.

The company has launched an investigation, while its client companies also released emergency statements to their employees and candidates who had applied for jobs using PageUp's software.

The company has a long list of major Australian companies as clients, with the ABC confirming Target, Telstra, Reserve Bank of Australia, Medibank, Officeworks, Kmart, Commonwealth Bank, Jetstar, NAB, Aldi, Linfox, Coles, Australia Post and Lindt as clients of the company.

PHOTO: The personal details of thousands of Australians has potentially been compromised due to a data breach. (ABC News/ Reuters)

RELATED STORY: What to do if you fear your details may have been compromised

RELATED STORY: If a company has lost your personal info, they now have to tell you

RELATED STORY: Why February 22 is so important for Australian businesses

Key points:

Human Safeguards

- Hiring – Hire the right people
- Training – Train them
- Education – Educate them (SQL Injection; ransomware, trojans, worms)
- Procedures – How to prevent, what to do when (*not if*) it happens
- Administration – Not Set & Forget – active monitoring and enforcement
- Compliance – Privacy, Data Laws and Company policy
- Accountability – Someone is *responsible* and will be *accountable* if something goes wrong

Technical Safeguards – Access Control

- Access Control
- Access Control Policy (People & Procedures)
  - A high level set of rules to grant, revoke and or deny access to the database
- Access Control Model (Procedure)
  - The model is the formalised policy of the rules
- Access Control Mechanism (Data & Technical)
  - The mechanism is the means to enforce the policy

Technical Safeguards - Access Control System

- Policy: Only HR & Payroll managers should be able see the salary of employees. Only HR staff should be able to see an employee's date of birth
- Model:
  - Everybody who is a HR Manager or Payroll Manager will be able to see the employee table, all other job roles such as HR Staff, or Payroll Clerk will have to access the employee table via a view which omits the salary information. Only HR staff should be able to see an employees date of birth
- Mechanism: Role Based AC

```
CREATE VIEW V_EMPLOYEE
AS
SELECT employeeid, firstname,
Lastname, departmentid, bossid,dateofbirth
FROM employee;

GRANT select on V_EMPLOYEE
to HRstaff;
```

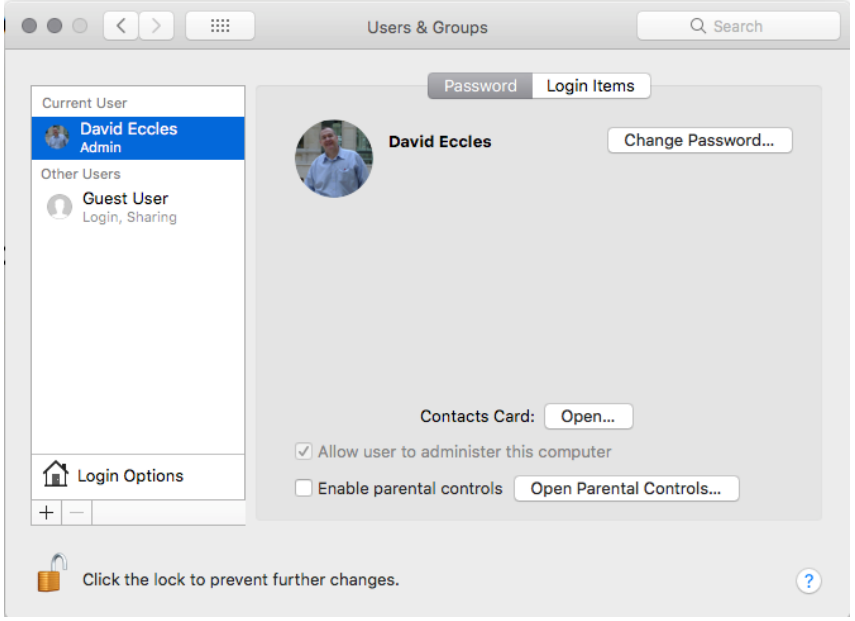
Staff	Role	EmpID
Helen Jones	HR Manager	56
Van Nguyen	Payroll Manager	23
Cathy Bates	HR Staff	101
Wolfgang Tuck	Payroll Clerk	27

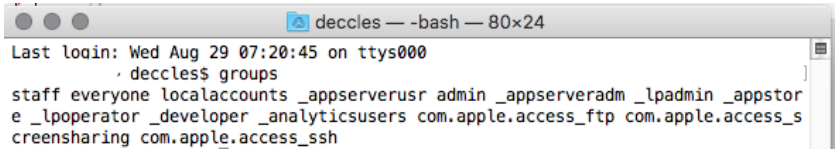
Access Control Types

- DAC Discretionary Access Control
  - Based on the identity of the user requesting access
  - Explicitly states which user (subject) can perform which action (action) on which resource (object)
  - Authorization is triple:
    - Subject (User)
    - Object (Table)
    - Action (DML, DDL, DQL)
- Types of DAC
  - Authorization Table
  - Access Control List
  - Capability

Access Control: Other types

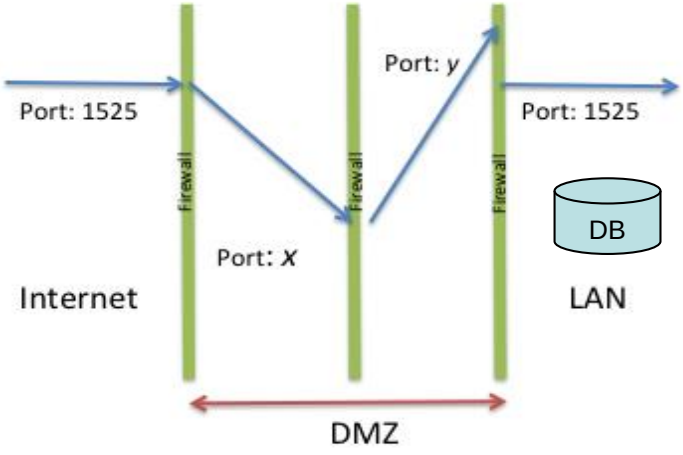
- Mandatory Access Control
  - Single Sign On
  - Active Directory
- Role Based Access Control
  - Unix / Linux / Mac Groups





Technical Safeguard - Firewalls

- Protective layer between your LAN and the WAN / Internet
- All network traffic is quarantined and authenticated
- Often several layers and types of firewall
- DMZ – Demilitarized Zone



Data Safeguard - Encryption

- Encryption turns “clear text” into “*k4#h2nsk7”
- involves very big prime number calculations used to scramble clear text
  - Then “Salting” adding a byte or two to the encrypted string
  - Public Key and a Private Key
    - Public key is broadcast
    - Private key is required to unencrypt
      - (Your PIN on your ATM card is a Private Key)
- Encryption *does not* hide data - it masks data
  - Delays the data being revealed
      - (may be hours/weeks/years/centuries/millennia)

Data Safeguards – What is a Backup?

- A backup is a copy of your data
  - however there are several types of backup
- If data becomes corrupted or deleted or held to ransom it can be restored from the backup copy
- A backup and recovery strategy is needed
  - To plan how data is backed up
  - To plan how it will be recovered

Backups protect data from …

- human error
  - e.g. accidental drop or delete
  - example: http://www.theaustralian.com.au/australian-it/human-error-triggered-nab-software-corruption/story-e6frgakx-1225962953523

- hardware or software malfunction
  - bug in application
  - hard drive (failure or corruption)
  - CPU
  - memory

Must also protect against

- malicious activity
  - security compromise
    - server, database, application
- natural or man made disasters
  - consider the *scale* of the damage
- government regulation
  - historical archiving rules
  - Metadata collection (AUS)
  - GDP (EU) HIPPA (US)
  - Privacy Rules

Security

Texas cops lose evidence going back eight years in ransomware attack

We have to get very, very tough on cyber and cyber warfare… and backups?

By Alexander J Martin 27 Jan 2017 at 16:57 36 SHARE ▼

♫ I hacked the sheriff, but I did not hack his deputy ♫

Updated Cockrell Hill, Texas has a population of just over 4,000 souls and a police force that managed to lose eight years of evidence when a departmental server was compromised by ransomware.

In a public statement, the department said the malware had been introduced to the department's systems through email. Specifically, it arrived "from a cloned email address imitating a department issued email address" and after taking root, requested 4 Bitcoin in ransom, worth about $3,600 today, or "nearly $4,000" as the department put it.

Categories of Failure

Failures can be divided into the following categories:

- Statement failure
  - Syntactically incorrect

```
SELECT employeeid, firstname, lastname, celery
FROM employee;
```

- User Process failure
  - The process doing the work fails (errors, dies)

MySQL Workbench has stopped working

- Network failure
  - Network failure between the user and the database

Unable to connect to info20003db.eng.unimelb.edu.au

- User error
  - User accidentally drops the rows, table, database

```
-- What does this do?
DROP table employee;
Rollback;
```

- Memory failure
  - Memory fails, becomes corrupt
- Media Failure
  - Disk failure, corruption, deletion

Types of Backups

- Physical vs. Logical
- Online vs. Offline
- Full vs. Incremental
- Onsite vs. Offsite

Backups – Physical vs. Logical

Physical

- raw copies of files and directories
- suitable for large databases that need fast recovery
- database is preferably offline ("cold" backup) when backup occurs
  - MySQL Enterprise automatically handles *file* locking, so database is not wholly off line
- backup = exact copies of the database directories and files
- backup should include logs
- backup is only portable to machines with a similar configuration
- to restore
  - shut down DBMS
  - copy backup over current structure on disk

Logical

- backup completed through SQL queries
- slower than physical
  - SQL Selects rather than OS copy
- output is larger than physical
- doesn't include log or config files
- machine independent
- server is available during the backup
- in MySQL can use the backup using
  - Mysqldump
  - SELECT … INTO OUTFILE
- to restore
  - Use mysqlimport, or LOAD DATA INFILE within the mysql client

Backups Offline vs. Online

Online (LIVE) or HOT

- backups occur when the database is “live”
- clients don’t realise a backup is in progress
- need to have appropriate locking to ensure integrity of data
- No downtime or outage

Physical & Logical backups

Offline (Shutdown) COLD

- backups occur when the database is stopped
- simpler to perform
- offline backup is preferable, but not available in all situations
  e.g. applications without downtime

Logical backups only

Backups Full vs. Incremental

Full backup

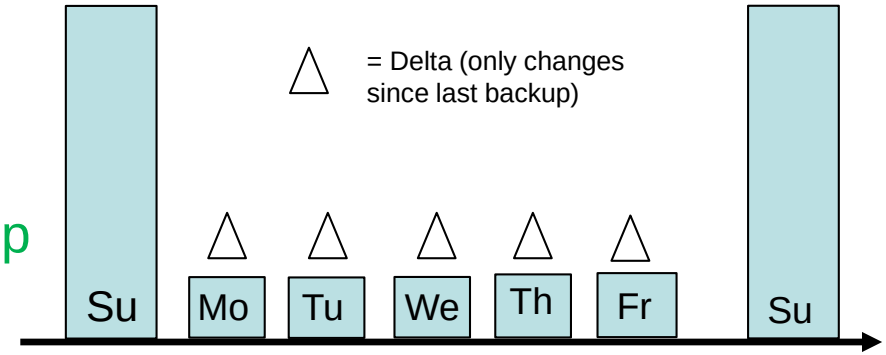
- a full backup is where the complete database is backed up
  - Physical (online or offline)
  - Logical (online)
- it includes everything you need to get the database operational in the event of a failure

Incremental Backup

- only the changes since last backup are backed up
- for most databases this means only backup log files
- to restore:
  - stop the database, copy backed up log files to disk
  - start the database and tell it to redo the log files

Backups Create a Backup Policy

- Backup strategy is usually a combination of full and incremental backups
  - for example:
    - weekly full backup
    - weekday incremental backup



- Conduct backups when database load is low
- If you replicate the database, use the mirror database for backups to negate any performance concerns with the main database
- **TEST** your backup before you **NEED** your backup!

Offsite Backup

- enables disaster recovery & business continuity
  - remote site (e.g. ASIC require 100 km away)
- backup tapes transported to underground vault
  - NAB Knox City vault (15 feet silicon wall)
- remote mirror database maintained via replication
  - Telstra Data Centres (Melbourne & Sydney)
- backup to Cloud

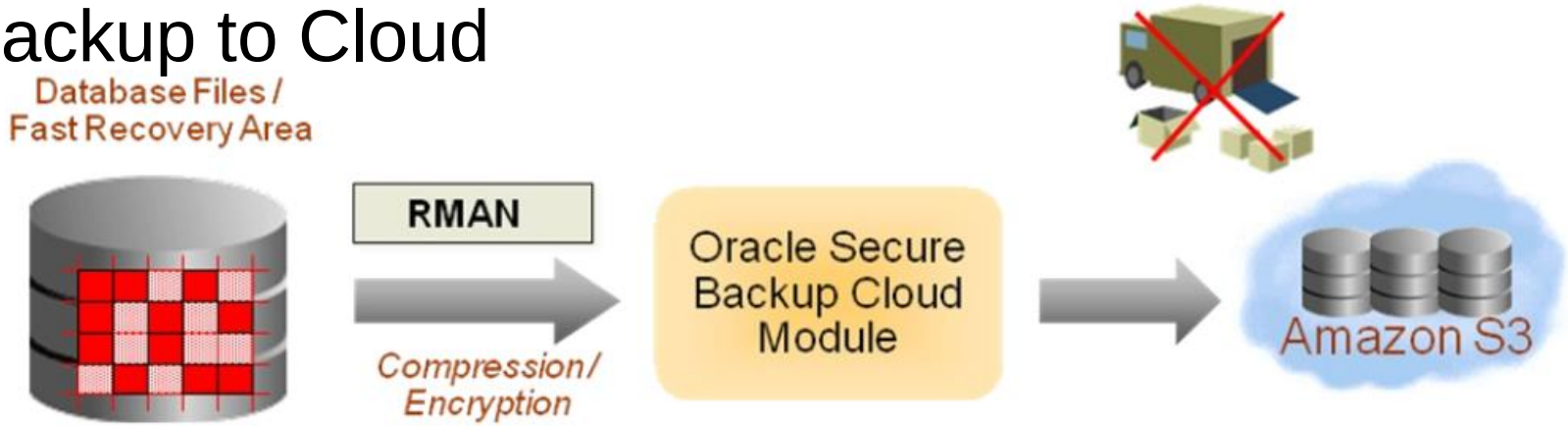
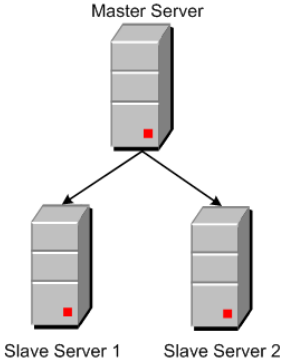


Figure 1. Oracle Database backup in the Cloud

Other ways to reduce risk of data loss

- Server replication

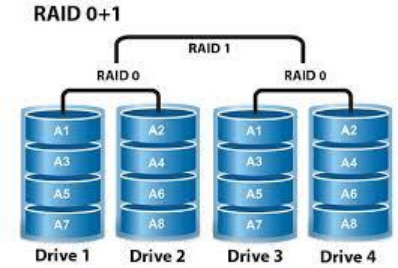


Problem	Protection?
accidental drop or delete	data loss!
server failure	protected
security compromise	limited protection

- Server cluster

Problem	Protection?
accidental drop or delete	data loss!
server failure	protected
security compromise	limited protection

- RAID



Problem	Protection?
accidental drop or delete	data loss!
server failure	data may be lost!
security compromise	all data compromised!

Database Hardening - checklist

- DB Physical Hardening
  - (harder to get to the server room)
- Firewalls for DB Servers
- Database Software; App / Web Server & App Code
  - (regularly patched, constant checking for vulnerabilities)
- Client Workstations / Browsers
  - least privilege rule
- Admin SU (Super User) accounts, permissions & passwords
- User roles, permissions, passwords & reporting
- Change Management
- Auditing
- Backup & Recovery

OWASP top ten web app vulnerabilities

https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

OWASP Top Ten web app vulnerabilities

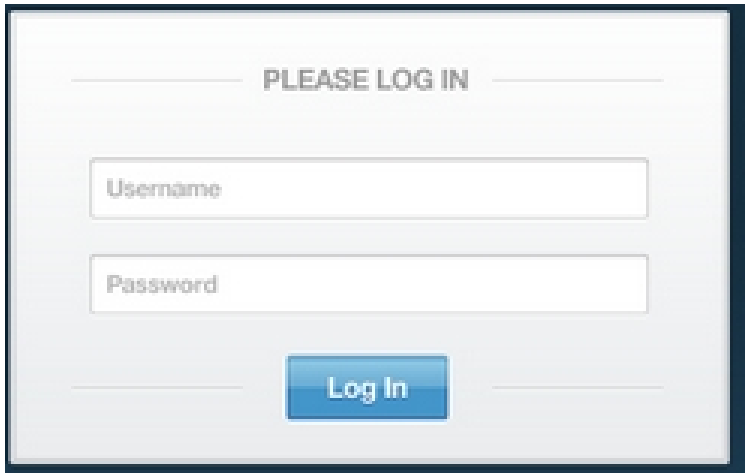
A1:2017-Injection	Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2:2017-Broken Authentication	Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
A3:2017-Sensitive Data Exposure	Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.
A4:2017-XML External Entities (XXE)	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

Protecting against SQL injection

- SQL Injection attacks
  - a technique used to exploit web applications that use *user input within database queries*
  - malicious code is entered into a data entry field in such a way that it becomes part of SQL commands that are run against the database
  - How to prevent:
    - sanitize user inputs
    - pass inputs as parameters to a stored procedure, rather than directly building the SQL string in the code

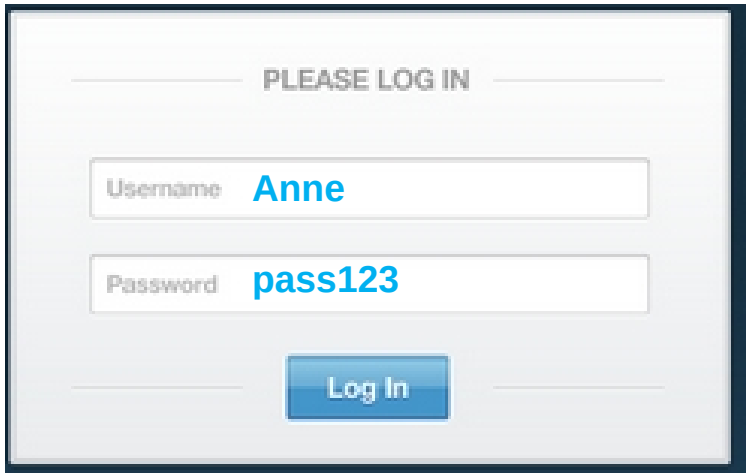
Login

- user inputs are used to form an SQL statement



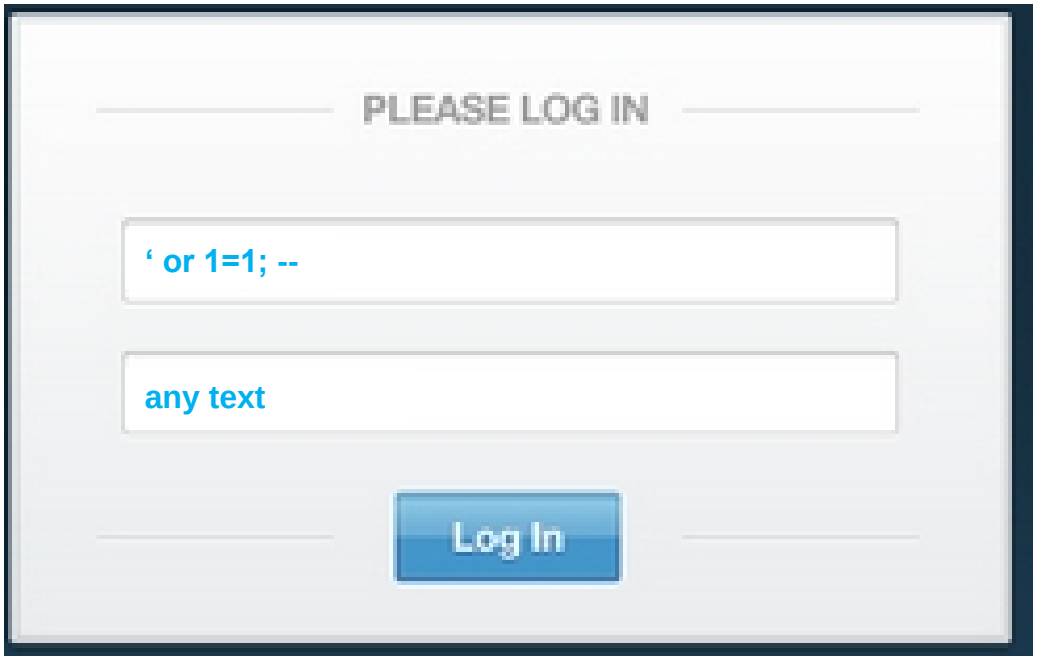

```
SELECT *
FROM User
WHERE username = ' @name '
and password = ' @pw ';
```

Programmer wants:




```
SELECT *
FROM User
WHERE username = 'Anne'
and password = 'pass123';
```

SQL injection: malicious input



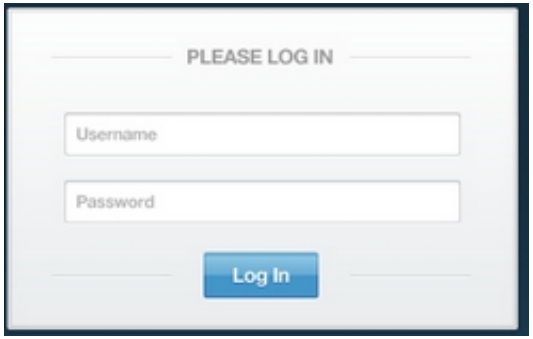
Text entered in @name string now

- closes the string
- adds a condition that is always true
- ends the SQL statement
- begins a comment with '- - ' to neutralize the rest of the SQL

```
SELECT *
FROM User
WHERE username = '' or 1=1; -- '
and password = 'any text';
```

SQL injection: prevention

- Primary defences:
  - Prepared Statements (parameterized queries)
  - Stored Procedures
    - (both mean SQL is no longer 'dynamic')
  - i.e. “escape” all user input
    - turns SQL special characters like ‘ ; -- into ordinary characters
- Additional defences:
  - Principle of Least Privilege
    - don't give application accounts DBA privileges
  - White List input validation
    - check input is from a list of acceptable values



Source: OWASP SQL Injection Prevention Cheat Sheet

THE UNIVERSITY OF MELBOURNE

9 Database Security, Ethics and Privacy

David Eccles

INFO90002 Database Systems & Information Modelling

Dr Greg Wadley

THE UNIVERSITY OF MELBOURNE

INFO90002 Database Systems & Information Modelling

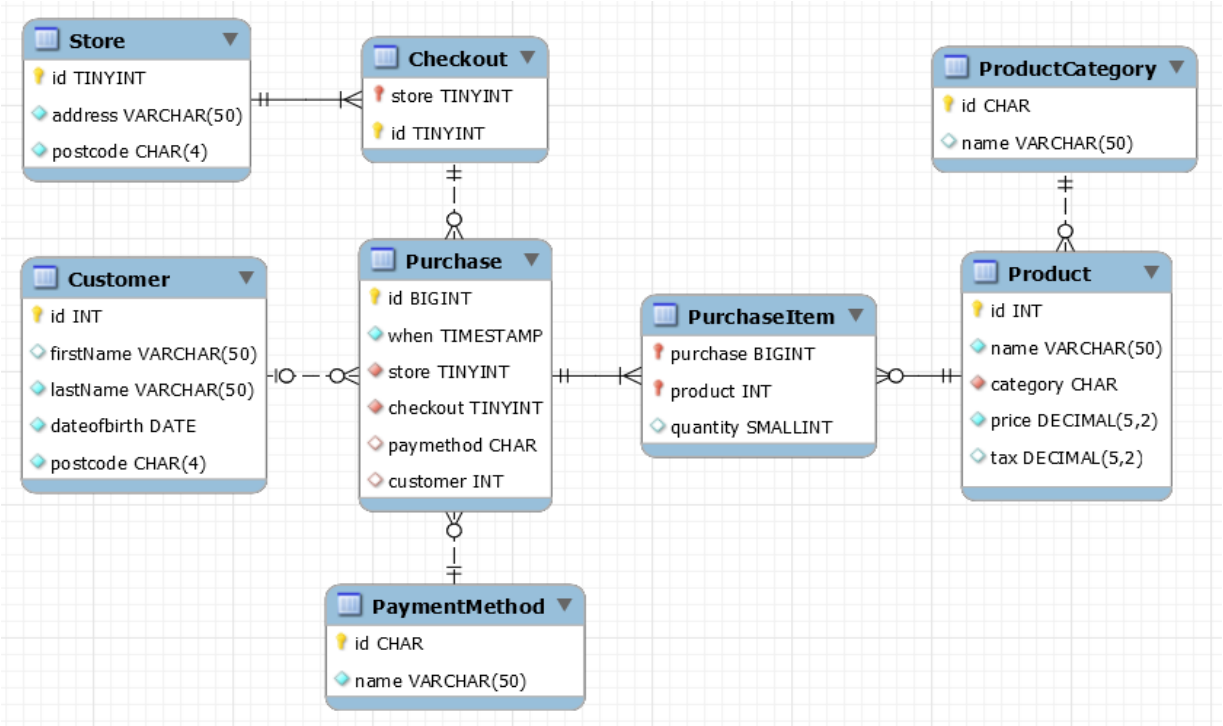
Week 10

NoSQL databases

Today's Session…

- Relational model
- Why NoSQL?
- example 1: MongoDB
- example 2: Firebase
- example 3: Neo4J
- Types of NoSQL
- CAP theorem
- ACID vs BASE
- NoSQL users

material in this lecture is drawn from http://martinfowler.com/books/nosql.html, including talk at GOTO conference 2012 and Thoughtworks article at https://www.thoughtworks.com/insights/blog/nosql-databases-overview



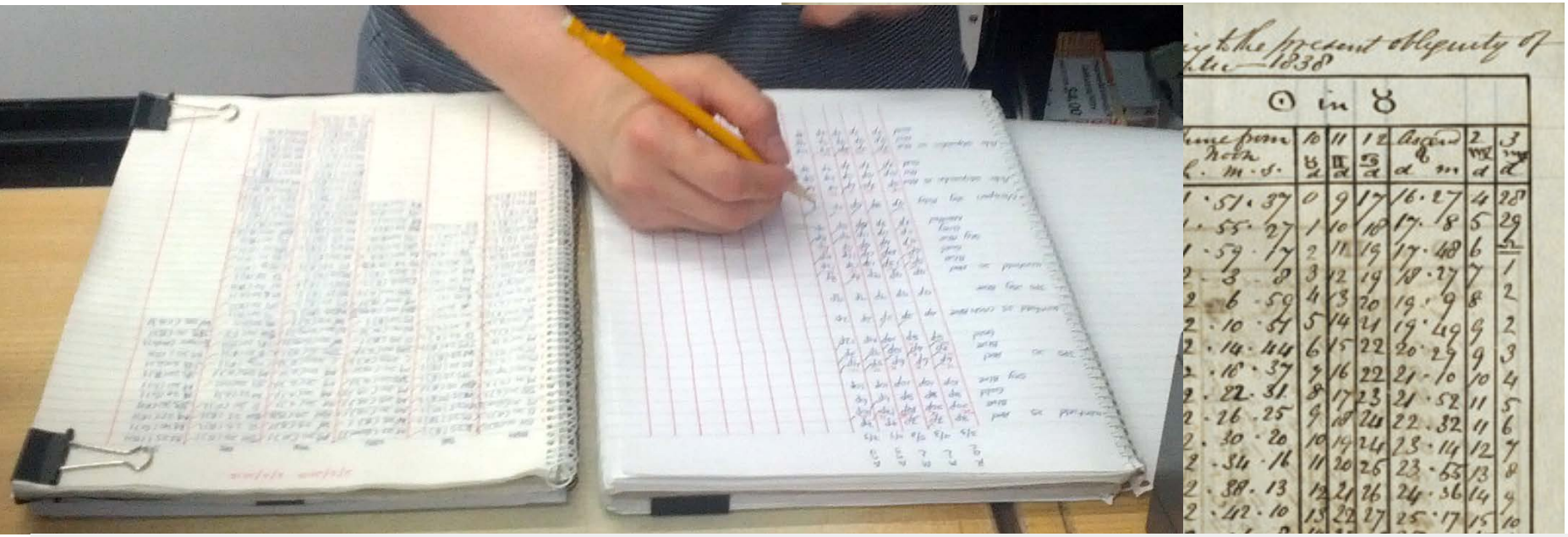
© University of Melbourne 2018

The dominance of the relational model

- Advantages of relational db
  - flexible, suits any data model
  - can integrate multiple applications via shared data store
  - standard within and between organizations
  - standard interface language SQL
  - ad-hoc queries, across and within "data aggregates"
  - fast, reliable, concurrent, consistent
- Problems of relational db
  1. object-relational (OR) “impedance mismatch”
  2. not good with big data
  3. not good with distributed (partitioned) databases
- adoption of NoSQL driven by “cons” of Relational
- but ‘polyglot persistence‘ = Relational will not go away

© University of Melbourne 2018

Problem 1: Much data is tabular …

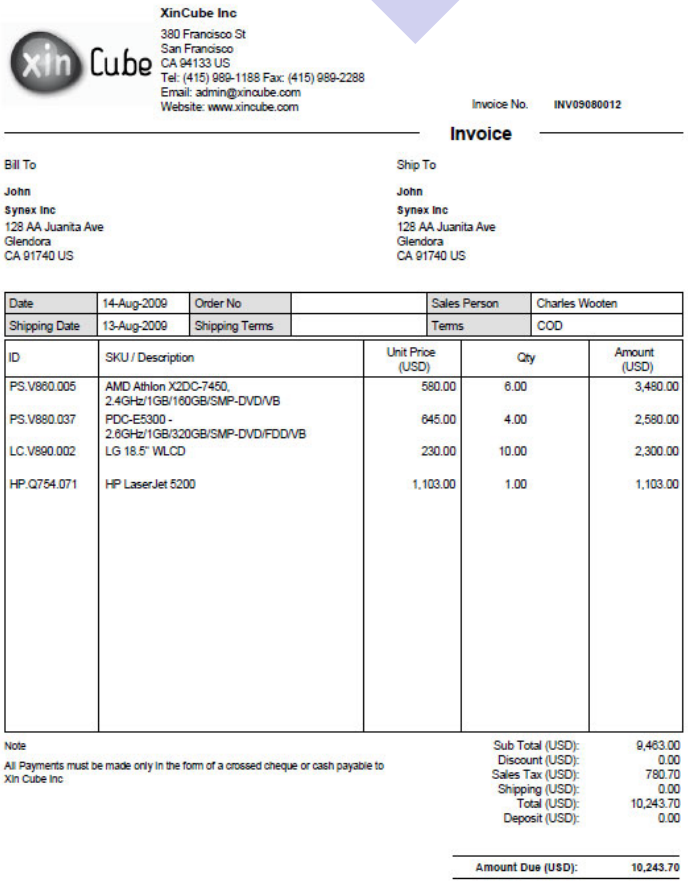


EMP_RECORD…	EMP_ID	EMP_REGION	EMP_DEPT	EMP_HIRE_D…	E…	E…	EMP_…	EMP_SALARY	EMP_NAME	EMP_SKIL
68	3715	4	153	09061987	9	6	1987	14000000	IRENE HIRSH	041085
62	39412	1	650	03119590	3	11	9590	167000000	ANN FAHEY	031099
56	1939	2	265	09281988	9	28	1988	21300000	EMILY WILM…	021077
50	3502	2	165	07041985	7	4	1985	19500000	CATHEZINE …	011015
44	4435	2	117	05141989	5	14	1989	17000000	AGNES KING	00
68	1673	3	138	07021985	7	2	1985	16800000	MARTIN XU	041033
62	4181	3	161	02031988	2	3	1988	15900000	JOHN DURN	030045
56	1443	1	265	12028900	12	2	8900	6000000	PAT DUNN	021055
50	3607	3	127	08072000	8	7	2000	18300000	ANDREA HIN…	011014
44	1775	3	288	02051989	2	5	1989	2700000	PETER JONES	00
68	1209	2	165	05121986	5	12	1986	17300000	DIDRA WILK…	041065

© University of Melbourne 2018

… but some is not inherently tabular

sometimes one aggregate is stored across many tables



XinCube Inc
380 Francisco St
San Francisco
CA 94133 US
Tel: (415) 989-1188 Fax: (415) 989-2288
Email: admin@xincube.com
Website: www.xincube.com

Invoice No. INV09080012

Invoice

Bill To
John
Synex Inc
128 AA Juanita Ave
Glendora
CA 91740 US

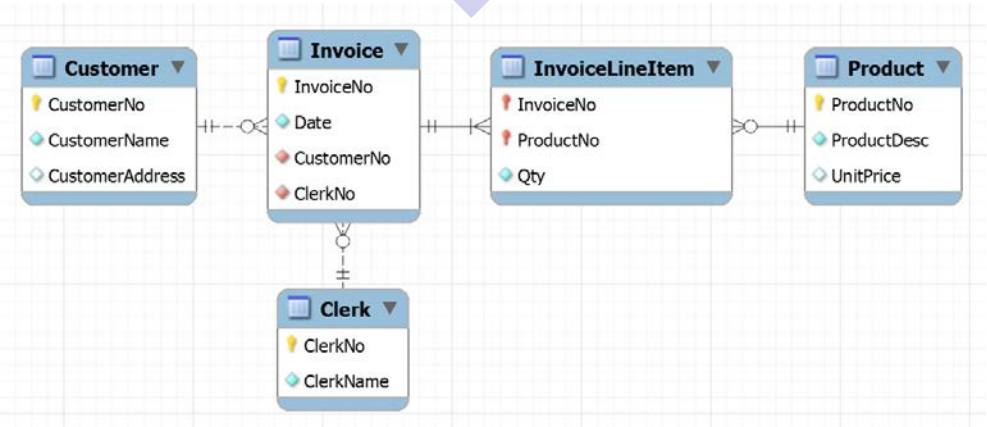
Ship To
John
Synex Inc
128 AA Juanita Ave
Glendora
CA 91740 US

Date	14-Aug-2009	Order No		Sales Person	Charles Wooten
Shipping Date	13-Aug-2009	Shipping Terms		Terms	COD

ID	SKU / Description	Unit Price (USD)	Qty	Amount (USD)
PS.V860.005	AMD Athlon X2DC-7450, 2.4GHz/1GB/160GB/SMP-DVD/VB	580.00	6.00	3,480.00
PS.V880.037	PDC-E5300 - 2.6GHz/1GB/320GB/SMP-DVD/FDD/VB	645.00	4.00	2,580.00
LC.V890.002	LG 18.5" WLCD	230.00	10.00	2,300.00
HP.Q754.071	HP LaserJet 5200	1,103.00	1.00	1,103.00

Note
All Payments must be made only in the form of a crossed cheque or cash payable to Xin Cube Inc

Sub Total (USD):	9,463.00
Discount (USD):	0.00
Sales Tax (USD):	780.70
Shipping (USD):	0.00
Total (USD):	10,243.70
Deposit (USD):	0.00
Amount Due (USD):	**10,243.70**



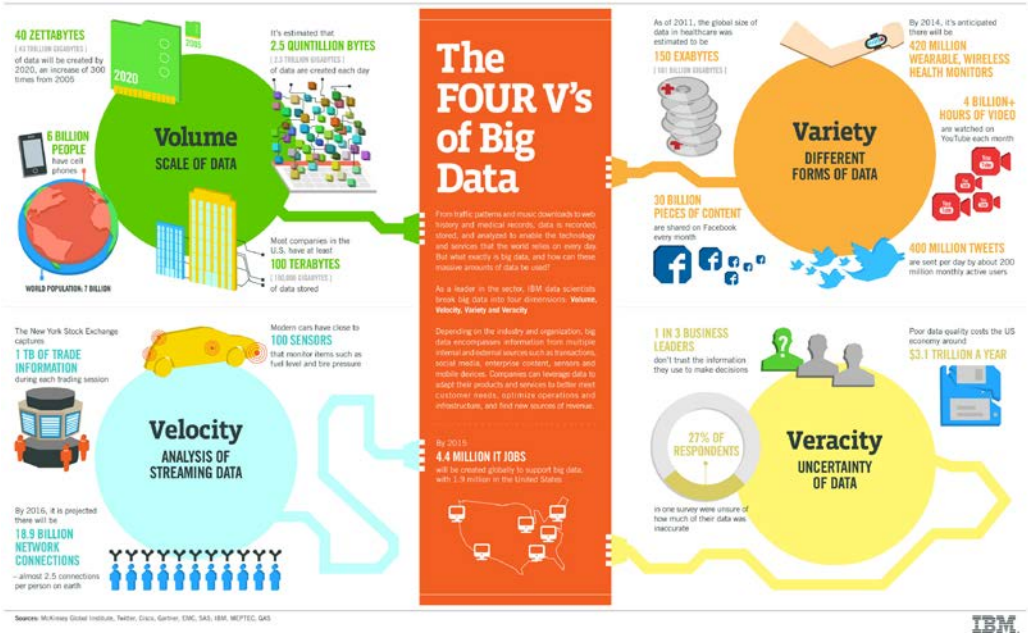
There is a lot of work to dissemble and reassemble the aggregate.

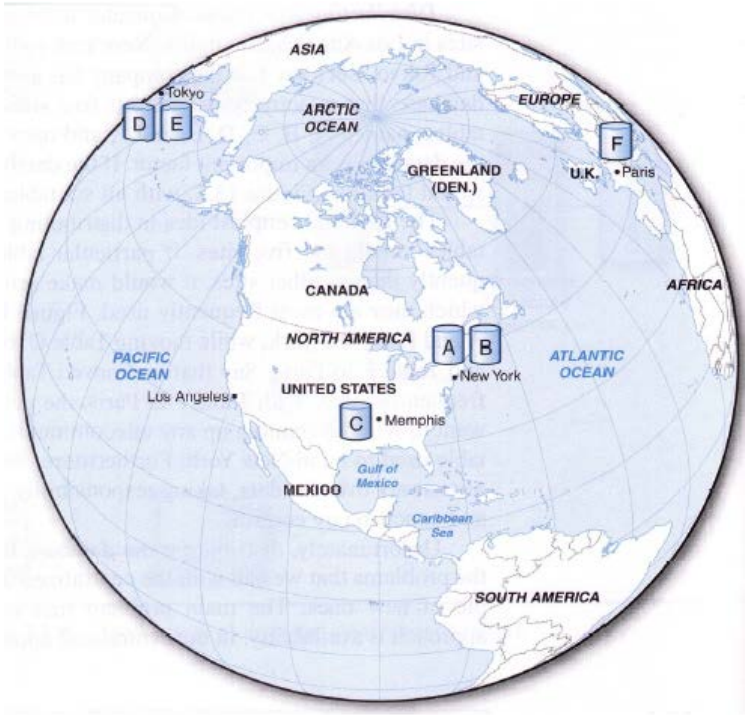
but it enables queries *across* aggregates such as:

```
SELECT productno, sum(qty)
FROM InvoiceLineItem
GROUP BY productno;
```


© University of Melbourne 2018

Problems 2, 3: Big Data, Distributed





Big Data = data sets with volume, velocity and variety that traditional relational databases struggle to cope with.

Distributed, especially partitioned, databases are not a good fit for some relational features e.g. foreign keys and transactions.

© University of Melbourne 2018

What is a NoSQL database?

- Features
  - doesn't use relational model (tables)
  - doesn't use SQL language
  - designed to run on distributed servers
  - most are open-source
  - built for the modern web
  - schema-less (though "implicit schema" in the application)
  - 'eventually consistent'
- Goals
  - to improve programmer productivity (OR mismatch)
  - to handle larger data volumes and throughput (big data)

from *NoSQL Databases: An Overview*
by Pramod Sadalage, Thoughtworks (2014)

© University of Melbourne 2018

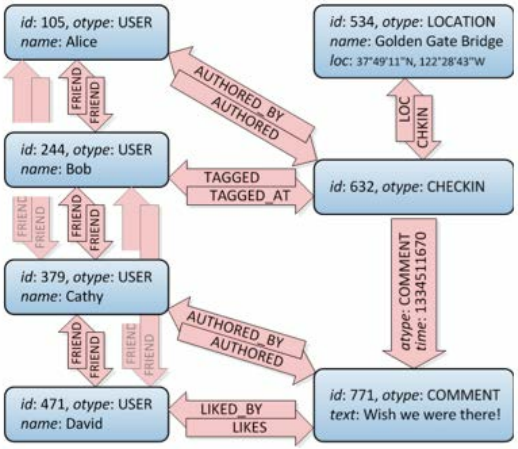
Three Demos

- We'll see 3 NoSQL databases in action:
  - MongoDB (https://www.mongodb.com/)
  - Firebase (https://firebase.google.com/)
  - Neo4J (https://neo4j.com/)
-

Before we start, we need to know a bit about:

  - JSON
  - Graphs

```
{
    id: 111111,
    name: "Alan",
    born: 1990,
    address: "1 Smith st",
    subjects: [
        { subject: "Database", result: "H1" },
        { subject: "Programming", result: "H2A" }
    ]
}
```



© University of Melbourne 2018

JSON documents

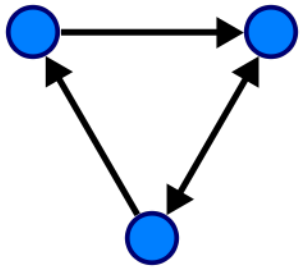
- JavaScript Object Notation
- represents a (JavaScript) object and its properties
- an object consists of a set of attribute-value pairs, including arrays of objects
- has a ‘tree’ structure
- originally used for transmitting data between computers
- now the storage format for Document databases

```
[       // array of students
{
    id: 111111,
    name: "Alan",
    born: 1990,
    address: "1 Smith st",
    subjects: [
        { subject: "Database", result: "H1" },
        { subject: "Programming", result: "H2A" }
    ]
}
,
{
    id: 222222,
    name: "Betty",
    born: 1992,
    address: "2 Two st",
    awards: "Best Student",
    subjects: [
        { subject: "Maths", result: "H1" },
        { subject: "Science", result: "H1" },
        { subject: "History", result: "H1"}
    ]
}
,
{
    id: 333333,
    name: "Chris",
    born: 1990,
    address: "3 Three st",
    subjects: [
        { subject: "Database", result: "H1" }
    ]
}
]
```

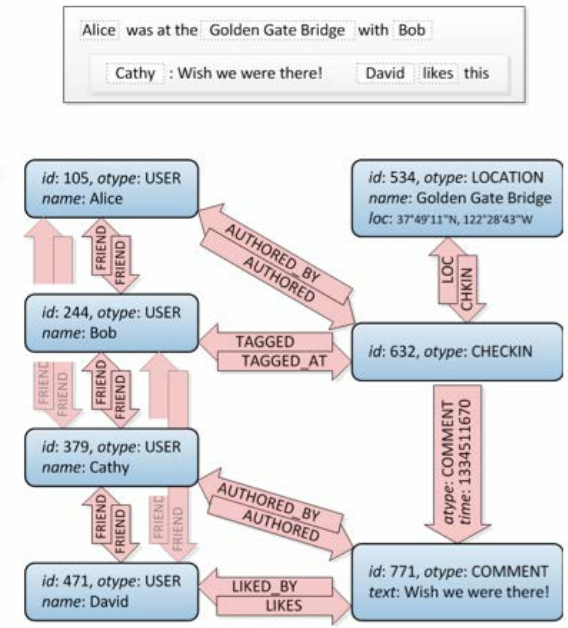

© University of Melbourne 2018

Graphs

- A data structure consisting of nodes/vertices and arcs/edges
- Nodes represent entities
- Arcs represent relationships
- May be directed or undirected
- In a graph database:
  - nodes and arcs can have properties and types
  - the emphasis is on relationships



directed graph (source: Wikipedia)



social graph (source: Facebook)

© University of Melbourne 2018

demo: the *MongoDB* document store

Log into the mongod server using the "mongo“ shell.

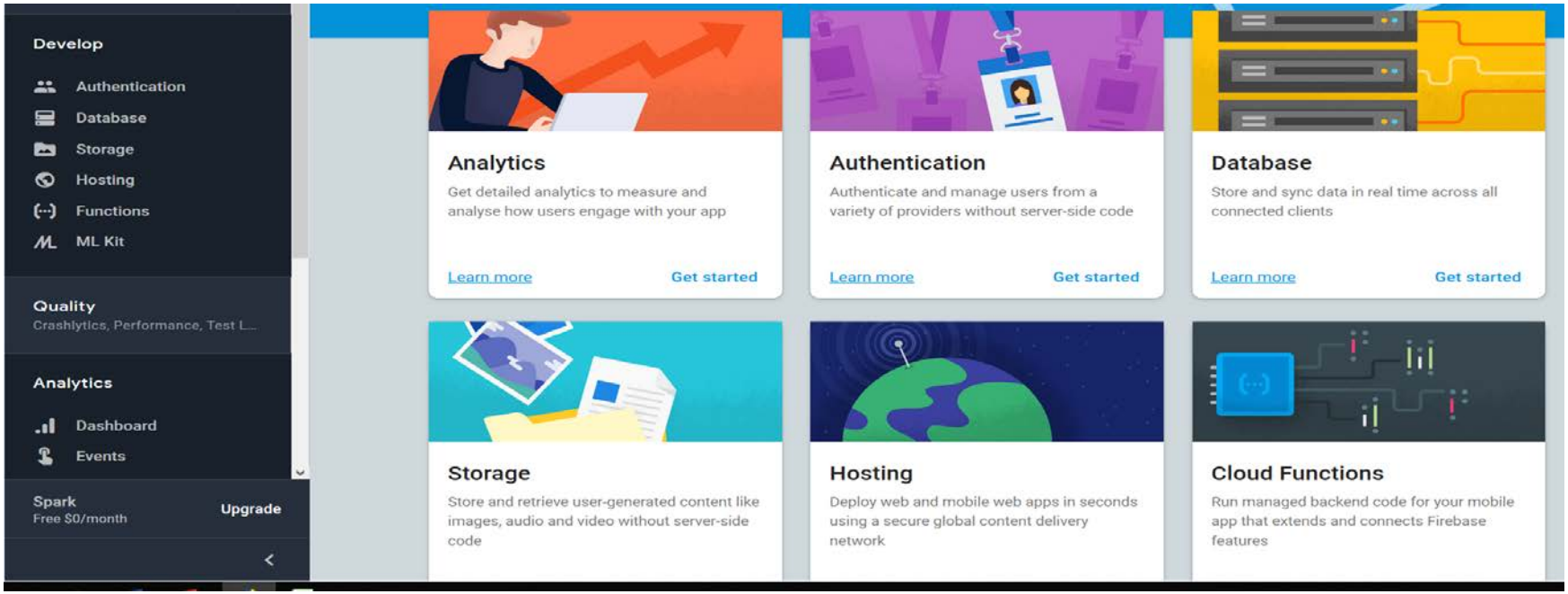
```
show dbs            // show a list of all databases
use test            // use the database called 'test'
show collections    // show all collections in the database 'test'

db.students.insert( {name: "Jack", born: 1982} )        // add a doc to collection
db.students.insert( {name: "Jill", born: 1980} )        // add a doc to collection
db.students.find()                                      // list all docs in students
db.students.find( {name: "Jill"} )      // list all docs where name field = 'Jill'

db.students.update( {name: "Jack"}, {$set: {born: 1980}} ) // change Jack's year
db.students.update( {id:222222}, {$addToSet:
            {subjects: {subject: "English", result: "H1"}}} )

db.students.find( {id:222222}, {_id:false, subjects:true} ).forEach(printjson)
db.students.remove( {born: 1980} )              // delete docs where year = 1980
```


© University of Melbourne 2018

demo: *Google Firebase*

"The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. When you build cross-platform apps with our iOS, Android, and JavaScript SDKs, all of your clients share one Realtime Database instance and automatically receive updates with the newest data."

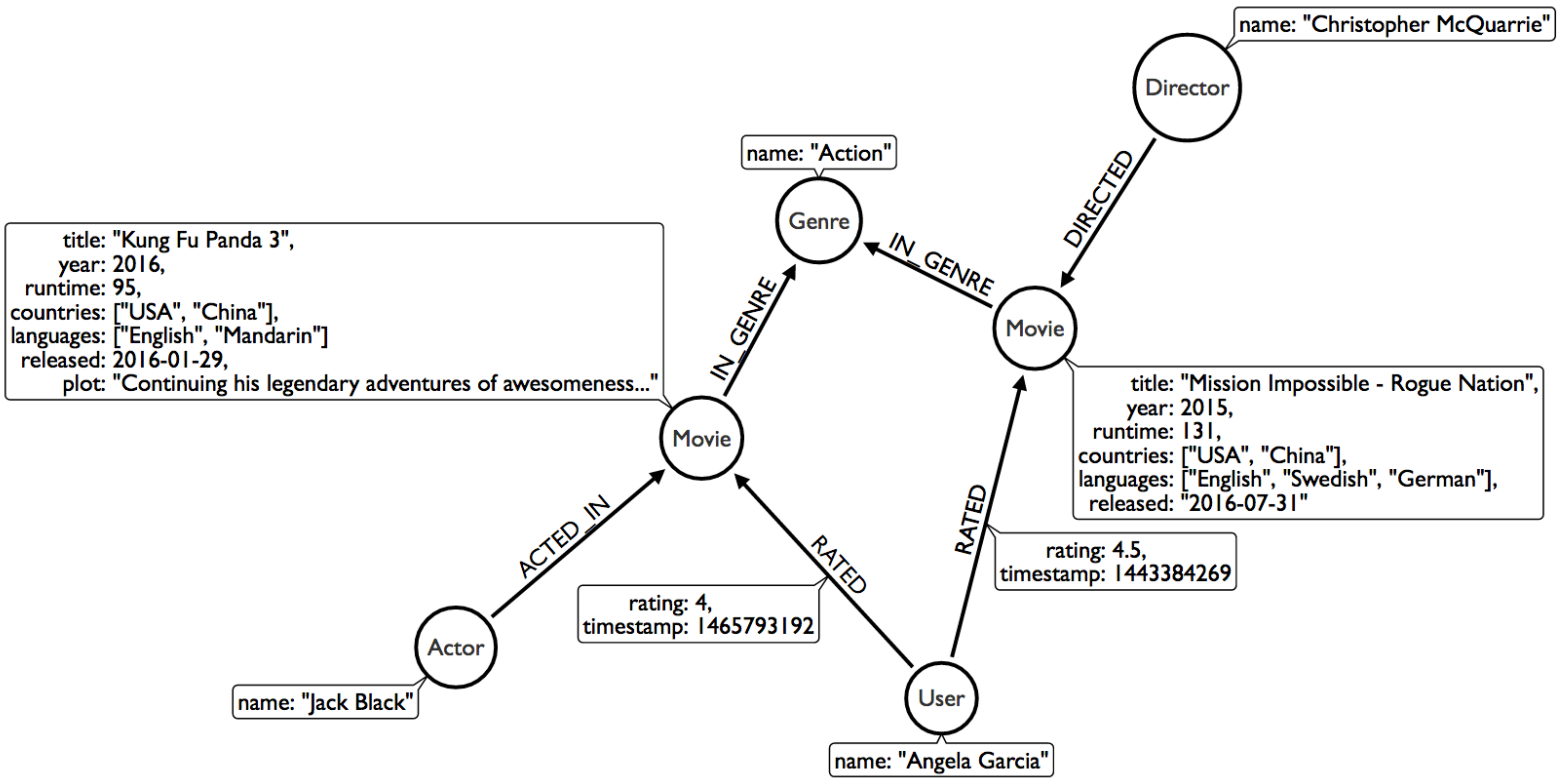

```
messageJSON
  04:48:19
    msg: "\"amm\""
    user: "\"Greg\""
  04:49:32
    msg: "\"b\""
    user: "\"Mary\""
  18:15:55
    msg: "\"hello there\""
    user: "\"Alice\""
```

© University of Melbourne 2018

demo: the *Neo4J* graph database

Nodes
Movie, Actor, Director, User, Genre are the labels used in this example.
Relationships
ACTED_IN, IN_GENRE, DIRECTED, RATED are the relationships used in this example.
Properties
title, name, year, rating are some of the properties used in this example.



© University of Melbourne 2018

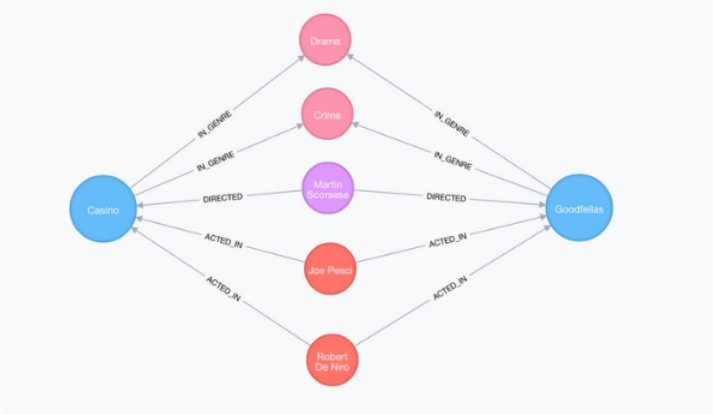
Neo4J queries

queries are written in the *Cypher* language

find	MATCH (m:Movie)<-[:RATED]-(u:User)	Search for an existing graph pattern
filter	WHERE m.title CONTAINS "Matrix"	Filter matching paths to only those matching a predicate
aggregate	WITH m.title AS movie, COUNT(*) AS reviews	Count number of paths matched for each movie
return	RETURN movie, reviews	Specify columns to be returned by the statement
order	ORDER BY reviews DESC	Order by number of reviews, in descending order
limit	LIMIT 5;	Only return first five records

Content-Based Filtering

Recommend items that are similar to those that a user is viewing, rated highly or purchased previously.

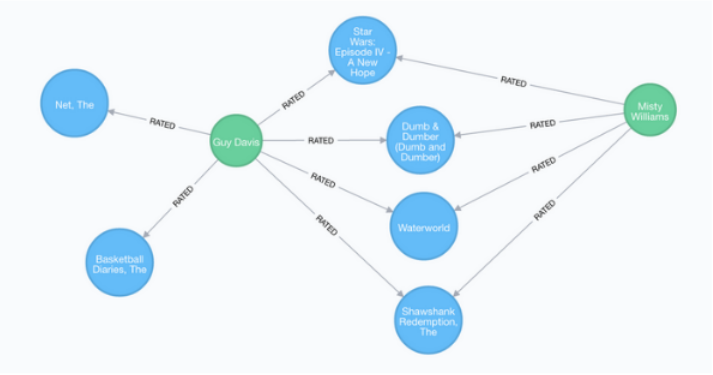


"Products similar to the product you're looking at now"

```
MATCH p=(m:Movie {title: "Net, The"})-[:ACTED_IN|:IN_GENRE|:DIRECTED*2]-()
RETURN p LIMIT 25
```

Collaborative Filtering

Use the preferences, ratings and actions of other users in the network to find items to recommend.

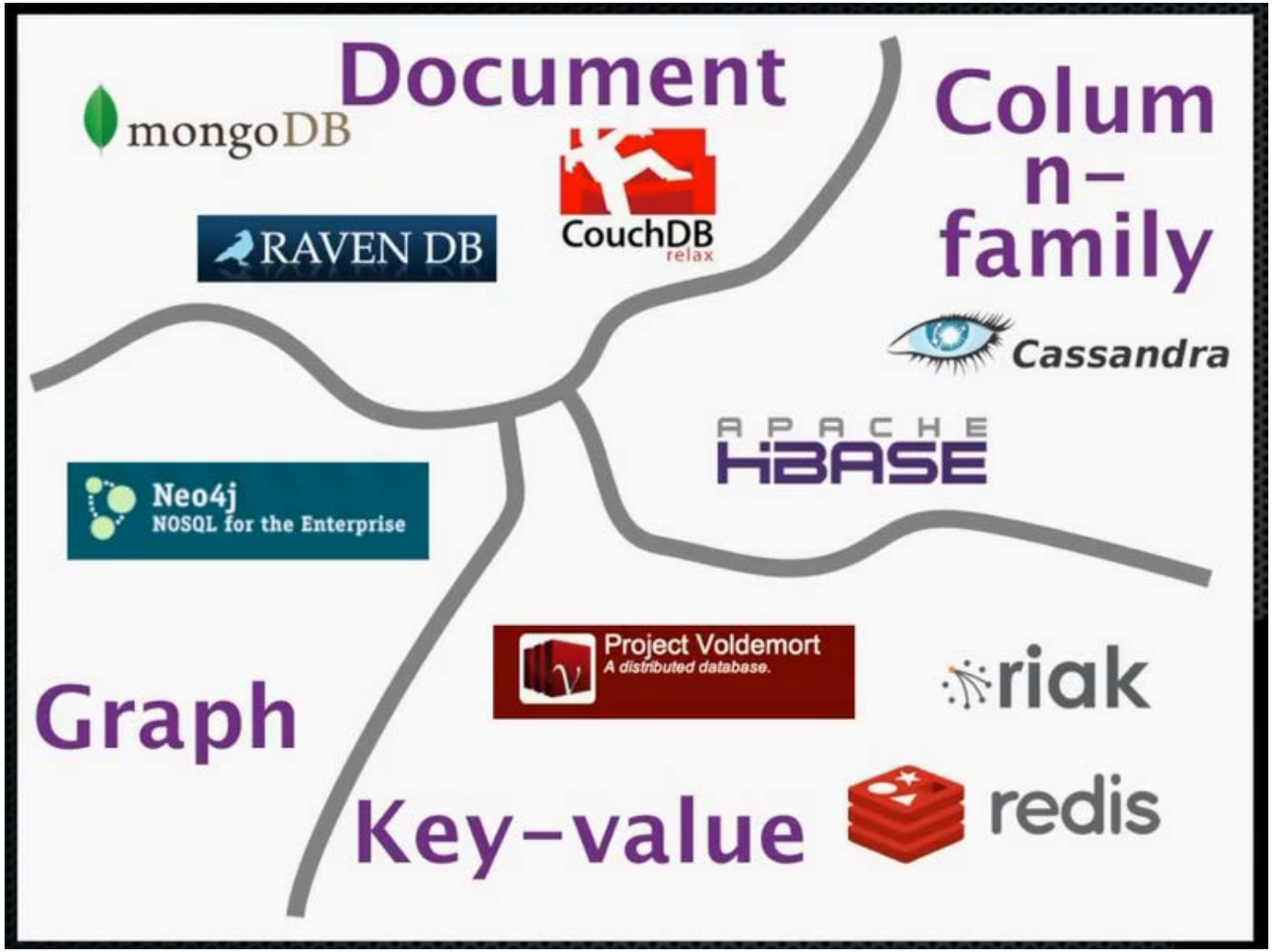


"Users who bought this thing, also bought that other thing."

```
MATCH (m:Movie {title: "Crimson Tide"})<-[:RATED]-(u:User)-[:RATED]->(rec:Movie)
RETURN rec.title AS recommendation, COUNT(*) AS usersWhoAlsoWatched
ORDER BY usersWhoAlsoWatched DESC LIMIT 25
```


© University of Melbourne 2018

Types of NoSQL database



(diagram from Martin Fowler)

© University of Melbourne 2018

Types of NoSQL: key-value store

Key = primary key

Value = anything
(number, array, image, JSON)
The application is in charge of interpreting what it means.

examples: Riak, Redis, Memcached, Berkeley DB, Project Voldemort, Couchbase

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

© University of Melbourne 2018

Types of NoSQL: document database

Like a key-value db, except that the "value" (document) is "examinable" by the db, so its contents can be queried and updated

document = object represented as JSON file

examples: MongoDB, CouchDB, Terrastore, OrientDB, RavenDB,

<Key=CustomerID>

```
{
   "customerid": "fc986e48ca6"
   "customer":
   {
   "firstname": "Pramod",
   "lastname": "Sadalage",
   "company": "ThoughtWorks",
   "likes": [ "Biking","Photography" ]
   }
   "billingaddress":
   { "state": "AK",
      "city": "DILLINGHAM",
      "type": "R"
    }
}
```

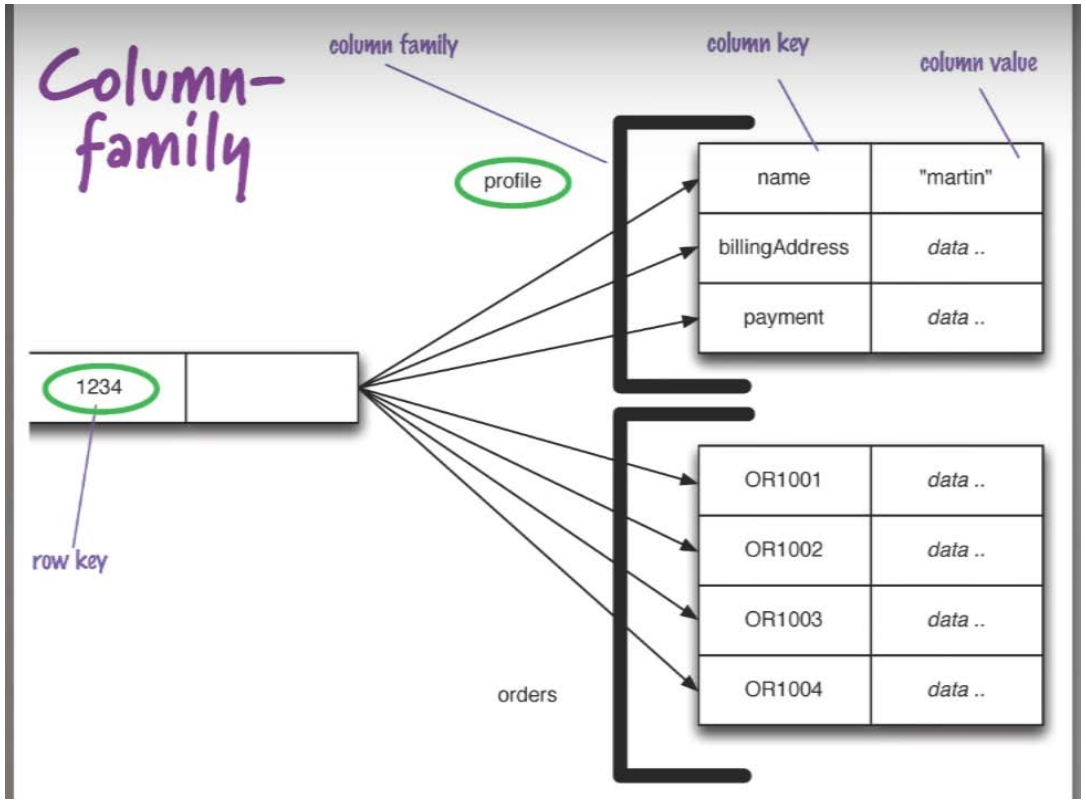

© University of Melbourne 2018

Types of NoSQL: column family

"Column family" is like a relational table.
It contains many "rows".
But each row can store a *different set of columns*.

Columns rather than rows are stored together on disk. Makes analysis by column faster – not for OLTP.

examples: Cassandra, BigTable, HBase, DynamoDB



© University of Melbourne 2018

Aggregate-oriented databases

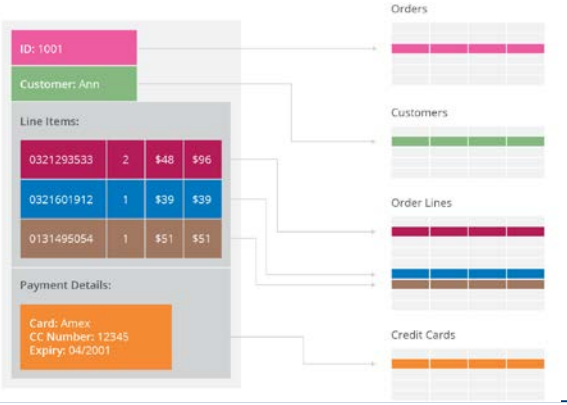
Key-value, *document store* and *column-family* are "aggregate-oriented" databases (in Fowler's terminology)

Pros

- entire aggregate of data is stored together
- less need for transactions
- efficient storage on clusters / distributed databases

Cons

- hard to analyse across subfields of aggregates
  e.g. sum over products instead of orders



Product	revenue	prior revenue
321293533	3083	7043
321601912	5032	4782
131495054	2198	3187
...	...	...

© University of Melbourne 2018

Types of NoSQL: graph database

A ‘graph' is a node-and-arc network
Social graphs (e.g. friendship graphs) are common examples
Graphs are difficult to program in relational DB
A *graph DB* stores entities and their relationships
Graph queries deduce knowledge from the graph

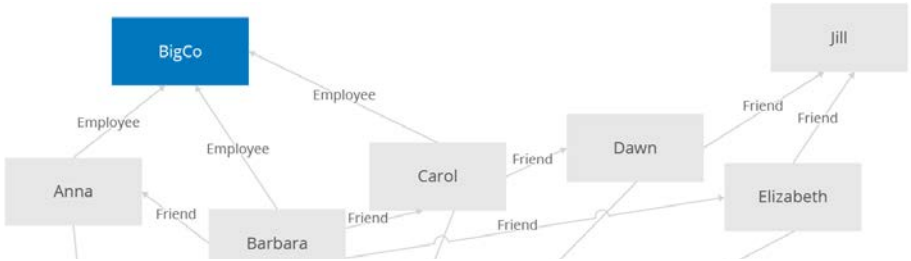


Table 2-1. Finding extended friends in a relational database versus efficient finding in Neo4j

Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

examples:

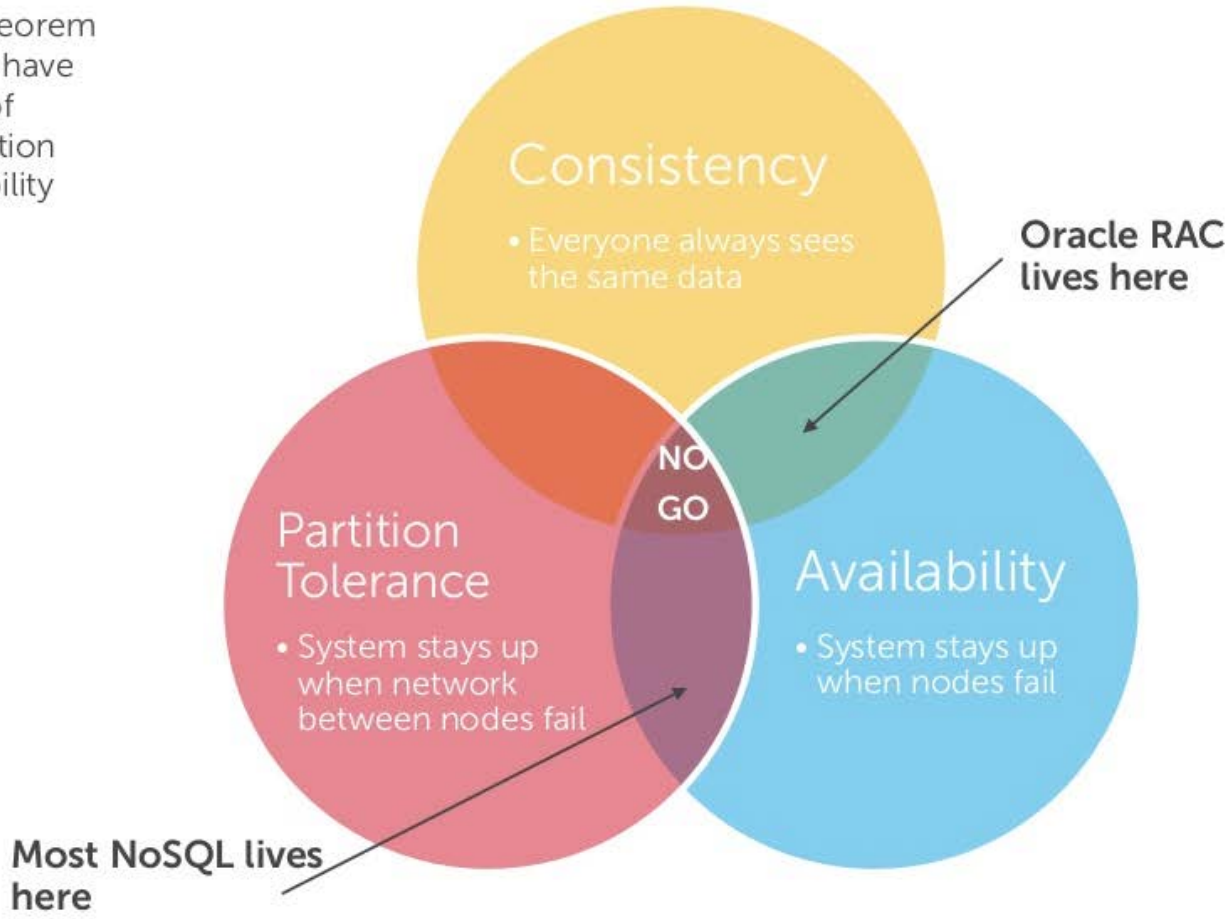
- Neo4J
- Infinite Graph
- OrientDB
- FlockDB
- TAO

© University of Melbourne 2018

Distributed data: the CAP theorem

CAP Theorem says something has to give

- CAP (Brewer's) Theorem says you can only have two out of three of Consistency, Partition Tolerance, Availability

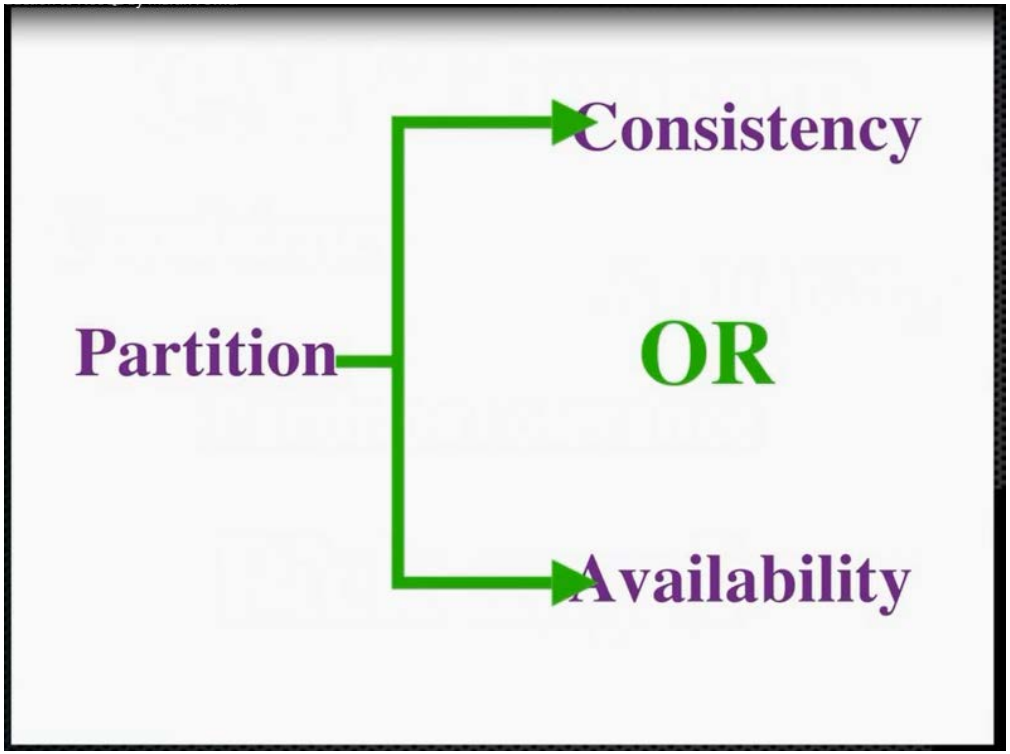


© University of Melbourne 2018

CAP theorem: alternative presentation

Fowler's version of CAP theorem …

if you have a distributed database,
when a partition occurs,
you must then choose consistency OR availability.



© University of Melbourne 2018

ACID vs BASE

ACID (Atomic, Consistent, Isolated, Durable)
vs
BASE (Basically Available, Soft state, Eventual consistency)

Basically Available: This constraint states that the system does guarantee the *availability* of the data; there will be a response to any request. But data may be in an inconsistent or changing state.

Soft state: The state of the system could change over time - even during times without input there may be changes going on due to 'eventual consistency'.

Eventual consistency: The system will eventually become consistent once it stops receiving input. The data will propagate to everywhere it needs to, sooner or later, but the system will continue to receive input and is not checking the consistency of every transaction before it moves onto the next one.

© University of Melbourne 2018

Some notable NoSQL users

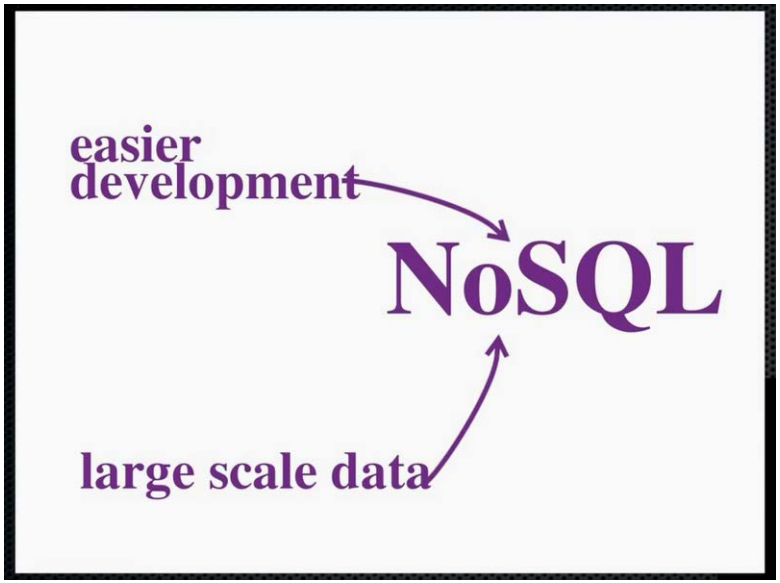
- Google – BigTable
  - search, gmail, maps, youtube
- Facebook – Cassandra, Tao, Giraph
  - messaging, social graph
- Amazon – SimpleDB, DynamoDB
  - large scale e-commerce and analytics, cloud db
- Instagram - Cassandra
  - social media newsfeed
- LinkedIn – CouchDB, MongoDB
  - monitoring and analysis of operational data
- The Guardian - MongoDB
  - newspaper articles, user identity
- FourSquare - MongoDB
  - venues and user checkins

© University of Melbourne 2018

Do *you* need to know NoSQL?

Q. Do only big web companies like Google, Amazon and Facebook need NoSQL?

A. In fact, any organization is likely to have to start dealing with large amounts of data (due to web, mobile, sensors etc), while some are adopting NoSQL to avoid object-relational mismatch (making programming easier).



but Relational DBMS will probably continue to be used in many applications

© University of Melbourne 2018

Dr Greg Wadley

THE UNIVERSITY OF MELBOURNE

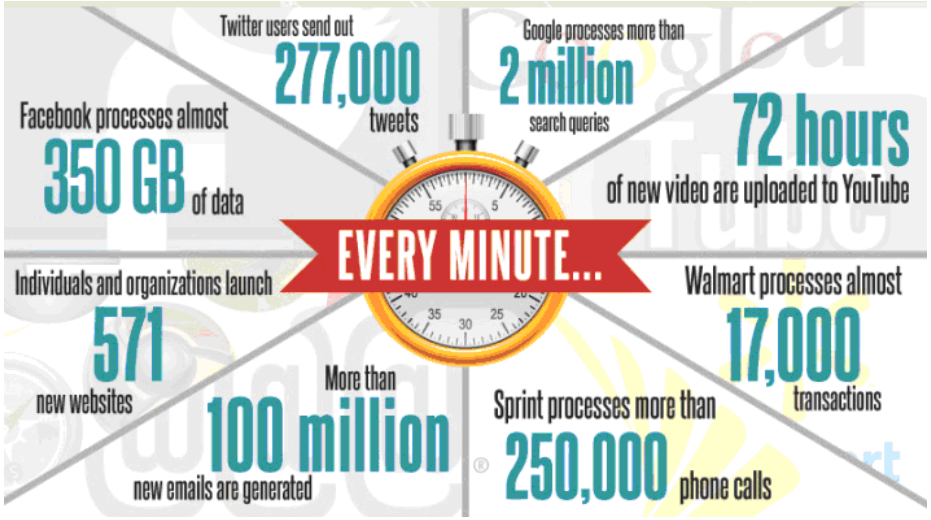
INFO90002 Database Systems & Information Modelling

Week 12

Database Trends

Today's Session…

- Relational status-quo
  - Major vendors
  - Current offerings
  - Market share
- Challenges to the relational status quo
  - Big Data
  - Cloud storage
  - Object-oriented applications
- Responses
  - NoSQL
  - NewSQL
  - Hadoop
  - In-memory databases



© University of Melbourne 2018

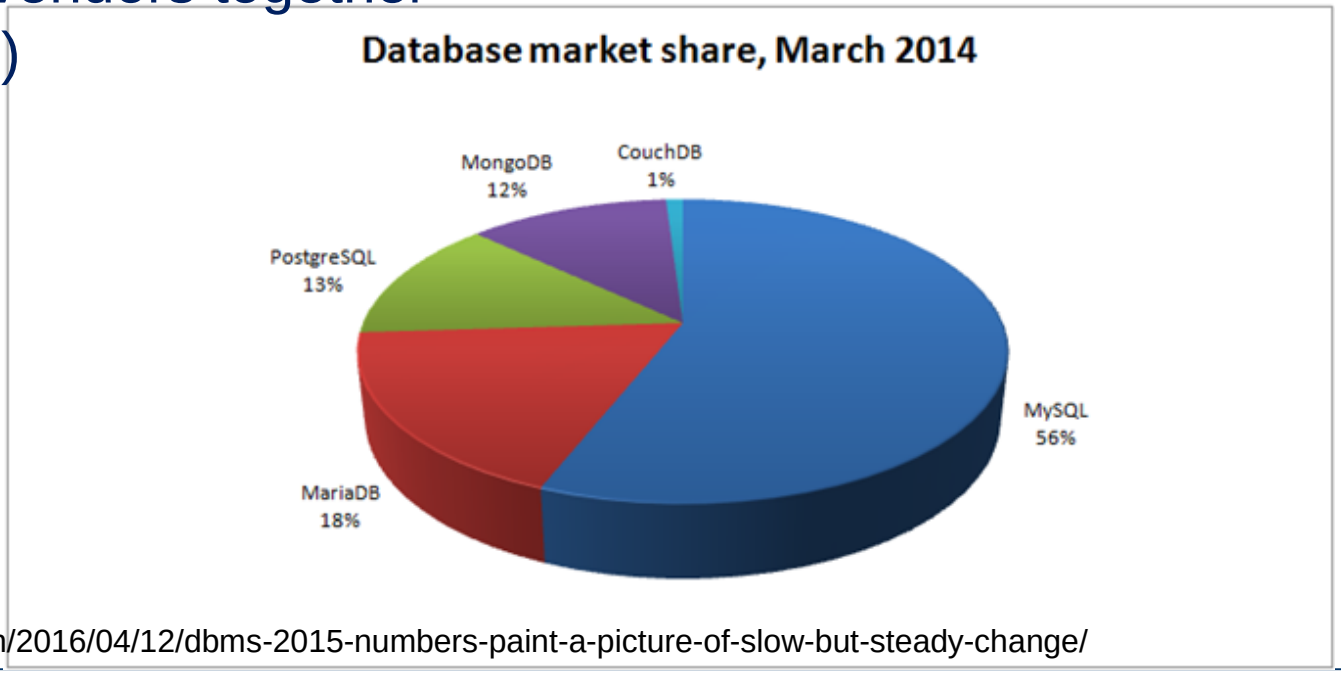
The Relational status-quo

- major vendors
- current offerings
- market share

Leading DBMS vendors

- Top *commercial* DBMS (Gartner, 2015)
  - 41.6% Oracle
  - 19.4% Microsoft Sql Server
  - 16.5% IBM DB2
  - (top 3 own over ¾ of market)
  - (top 5 NoSQL vendors together are at 8th place)

Open source DBMS



http://blogs.gartner.com/merv-adrian/2016/04/12/dbms-2015-numbers-paint-a-picture-of-slow-but-steady-change/

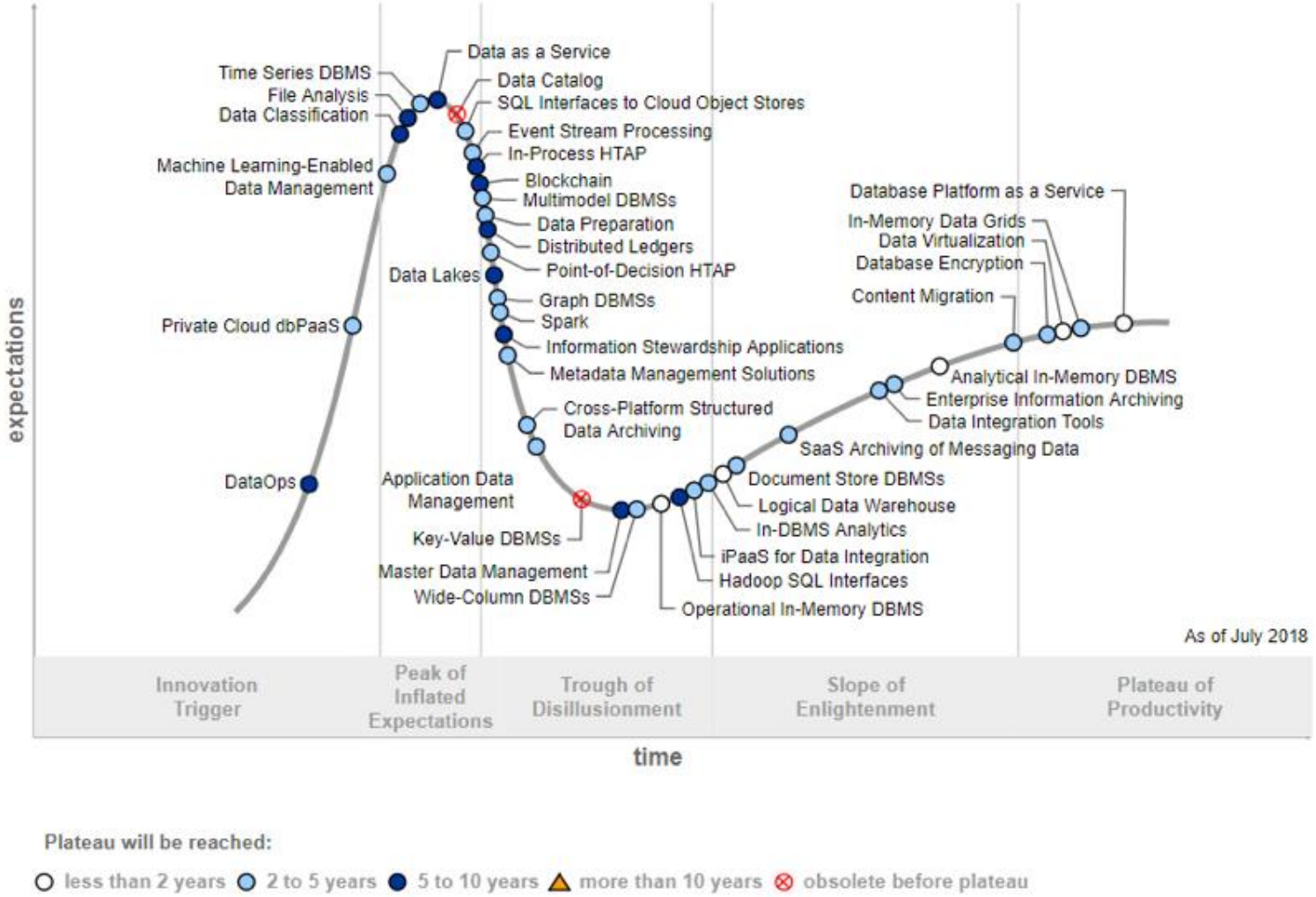
© University of Melbourne 2018

DBMS popularity

- **Number of mentions of the system on websites**, measured as number of results in search engines queries. At the moment, we use Google and Bing for this measurement. In order to count only relevant results, we are searching for <system name> together with the term database, e.g. "Oracle" and "database".
- **General interest in the system.** For this measurement, we use the frequency of searches in Google Trends.
- **Frequency of technical discussions about the system.** We use the number of related questions and the number of interested users on the well-known IT-related Q&A sites Stack Overflow and DBA Stack Exchange.
- **Number of job offers, in which the system is mentioned.** We use the number of offers on the leading job search engines Indeed and Simply Hired.
- **Number of profiles in professional networks, in which the system is mentioned.** We use the internationally most popular professional network LinkedIn.
- **Relevance in social networks.** We count the number of Twitter tweets, in which the system is mentioned.

© University of Melbourne 2018

Gartner's "Hype Cycle"



https://www.gartner.com/en/newsroom/press-releases/2018-09-11-gartner-hype-cycle-for-data-management-positions-three-technologies-in-the-innovation-trigger-phase-in-2018

© University of Melbourne 2018

Gartner's "Magic Quadrant"

"In this Magic Quadrant, Gartner evaluates the strengths and weaknesses of 11 providers that it considers most significant in the marketplace, and provides readers with a graph (the Magic Quadrant) plotting the vendors based on their ability to execute and their completeness of vision. The graph is divided into four quadrants: niche players, challengers, visionaries, and leaders. … Gartner drastically changed revenue requirements for inclusion, and as a result, axed 10 providers. … Microsoft has put a little distance between itself and Oracle for the top spot in the space. ... SAP took a minor step back from the pack on the vertical axis, allowing Amazon Web Services to sneak through and supplant itself as the only real competition Microsoft and Oracle have left."

(Solutions Review, 2017)

Source: Gartner (November 2017)

https://solutionsreview.com/data-management/whats-changed-2017-gartner-magic-quadrant-for-operational-database-management-systems/

© University of Melbourne 2018

Current major relational offerings

MySQL

The world's most popular open source database

Contact MySQL | Login | Register

MYSQL.COM DOWNLOADS DOCUMENTATION DEVELOPER ZONE

MySQL Server MySQL Enterprise Workbench InnoDB Cluster MySQL NDB Cluster Connectors More

Search this Manual

MySQL 8.0 Reference Manual

version 8.0

Run SQL Server on your favourite platform

It's all about choice. Now, you can get SQL Server 2017's industry-leading performance and security on Windows, Linux, and Docker containers.

Try now > Watch now ▷

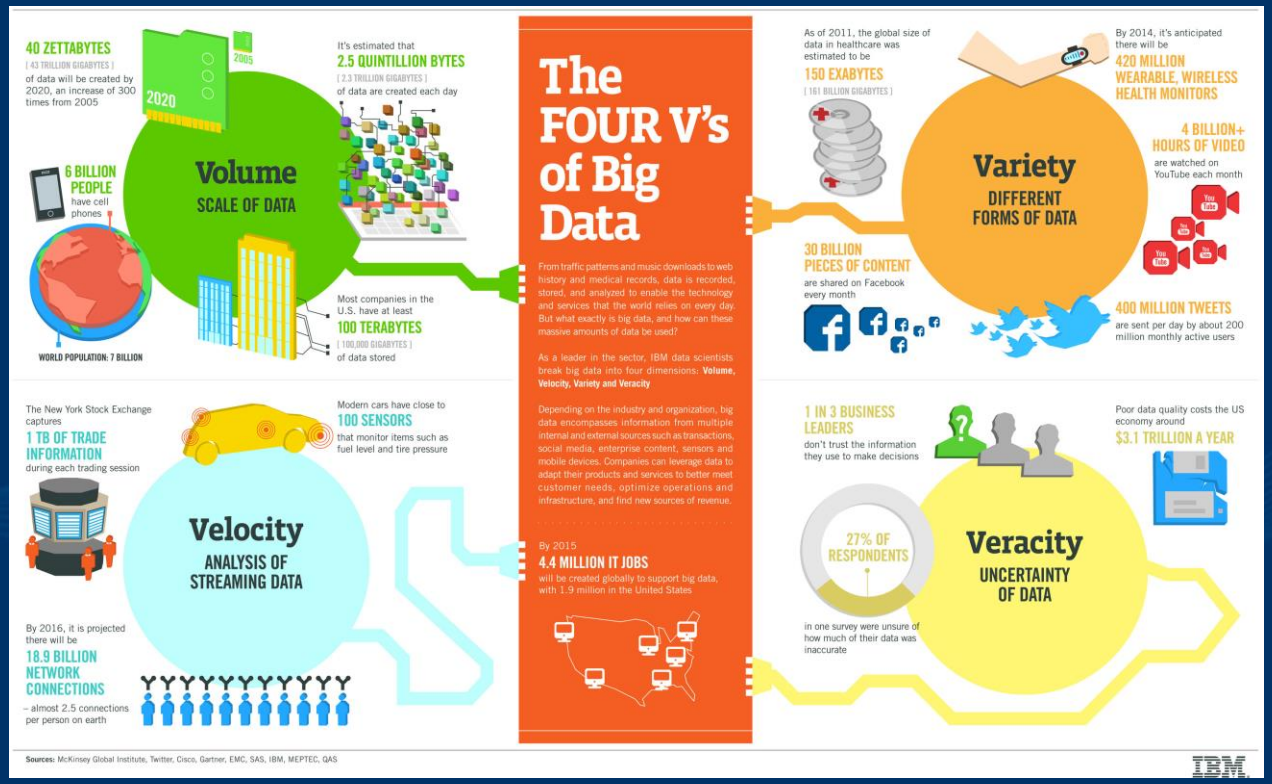
Oracle Database 18c

Oracle Database 18c, the latest generation of the world's most popular database is now available on Oracle Exadata, Oracle Database Appliance and Oracle Database Cloud. It's the first annual release in Oracle's new database software release model, and a core component of Oracle's Autonomous Database Cloud.

Get Started →

© University of Melbourne 2018

Challenge: Big Data



Extremely large data sets

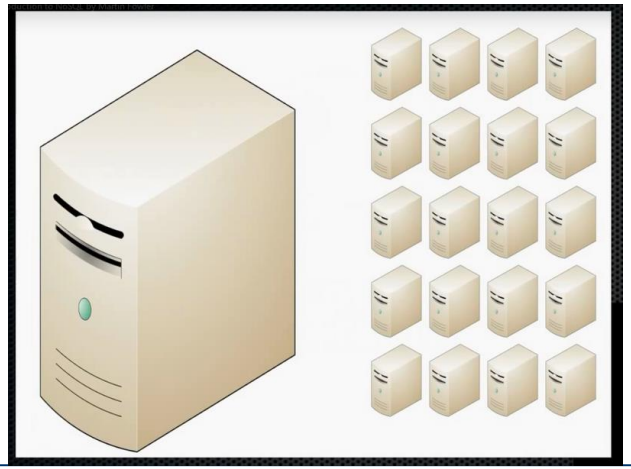
- Difficult to process high volume and velocity of data using traditional database management and processing tools
- Challenges include capture, storage, search, sharing, analysis, and visualization.
- Relational databases emerged when data was generated within organizations, mostly manually by employees. This scenario still occurs, and relational may be best here.
- But data is now generated by: social media, global website clickstreams, machinery, sensors, scientific instruments.
- "Internet of Things" … 30 billion wireless devices by 2030 (http://www.dbta.com/Editorial/Trends-and-Applications/Powering-the-Internet-of-Things-with-Real-Time-Hadoop-103469.aspx)
- Exabytes … 1,000,000 terabytes 1,000,000,000 gigabytes
- Volume and velocity of data is too high for SQL databases

© University of Melbourne 2018

Solution: distributed database

- Large internet companies like Google and Amazon found they could not store huge data sets in relational databases
- How to scale up capacity?
- Vertical scaling (bigger servers) gives diminishing returns
- Therefore horizontal scaling:

  spread data across many small servers
- But since relational is not so good with distributed data, these companies devised new databases that are designed from the ground up to run across multiple servers
- Google: *Bigtable*
- Amazon: *Dynamo*
- inspired the NoSQL movement



© University of Melbourne 2018

Big Data – problems

- In its infancy
  - tools for using and analysing big data, as well as standards, are still being developed
- Need the right talent and technology and structure of workflows to optimize the use of big data
- Requires expensive professionals ("data scientists")

  "There will be a shortage of talent necessary for organizations to take advantage of big data. By 2018, the United States alone could face a shortage of 140,000 to 190,000 people with deep analytical skills as well as 1.5 million managers and analysts with the know-how to use the analysis of big data to make effective decisions" (McKinsey Global Institute)
- https://mbs.edu/education-development/degreeprograms/masterofbusinessanalytics
- http://science-courses.unimelb.edu.au/study/degrees/master-of-data-science/

© University of Melbourne 2018

Challenge: Cloud storage

Cloud Datastore

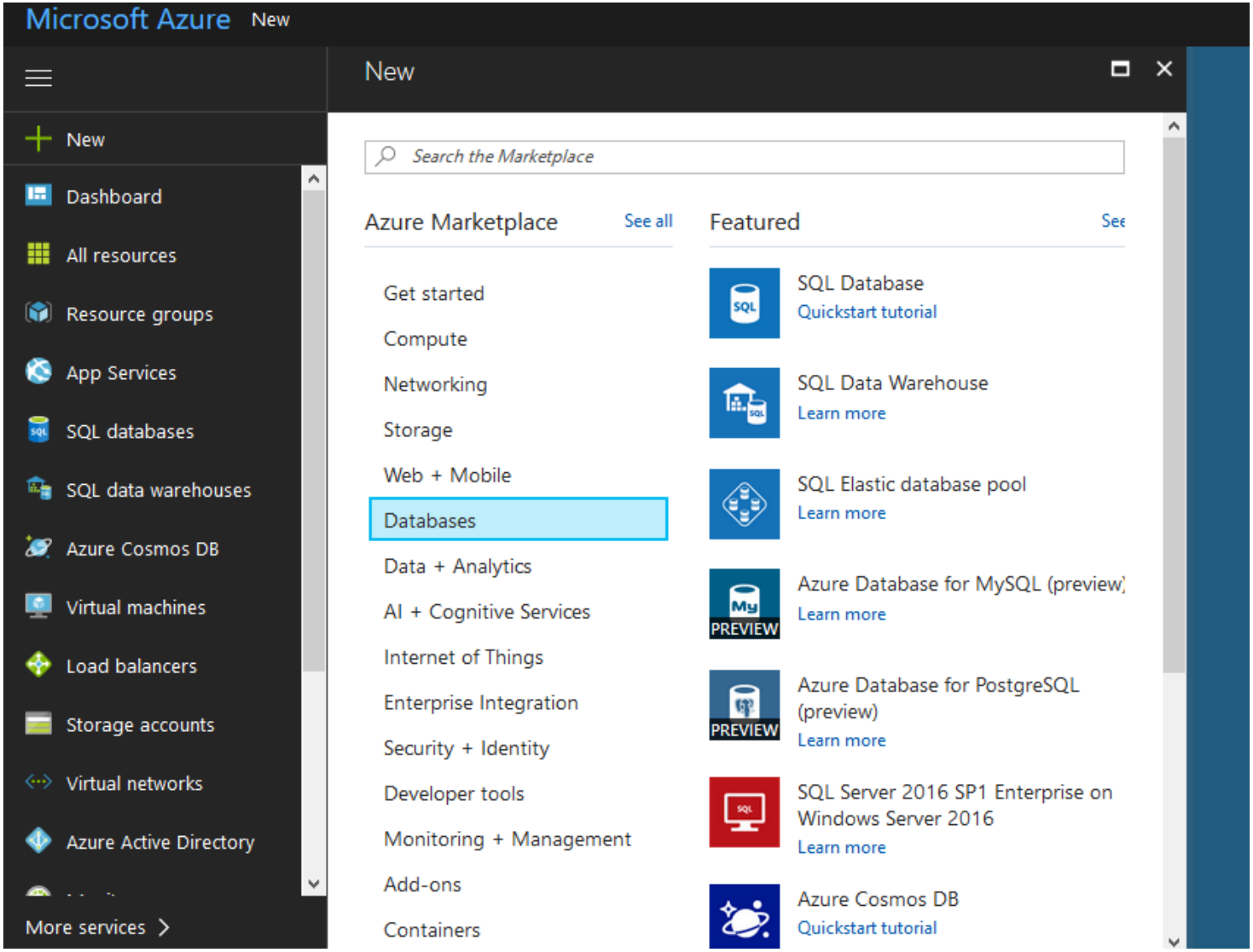
Microsoft Azure

Database as a service (DBaaS)

- Similar to other cloud services
  - DBMS and data are in the cloud
  - your application connects as required
  - pay per usage: quantity data stored, input/output
  - DBMS is administered by cloud provider
    - reduces need for in-house DBA
    - may be managed via web console
- Can be presented as either
  - database as a service (DBaaS)
    - relational database
    - non-relational (NoSQL) database
  - Virtual machine with a database installed
- Advantages
  - Simplifies setting up, and especially scaling up, your database

© University of Melbourne 2018

Example: Microsoft Azure



© University of Melbourne 2018

Potential cloud db issues

- Data security
  - provider may not have fully integrated security structure
  - may need to resort to encryption
- Legal frameworks
  - need to (continue to) conform to laws governing use of data
    - Can the data be moved out of a political jurisdiction?
    - How must the data be secured?
- Large movements of data between your site and cloud
  - during initial setup
  - during ongoing integration with local data

sourced from InfoWorld (2012)
http://www.infoworld.com/d/cloud-computing/the-unpleasant-truths-about-database-service-208450

© University of Melbourne 2018

Example cloud database vendors

- Information from www.mitre.org/work/tech_papers/2012/11_4727/cloud_database_service_dbaas.pdf
- SQL
  - Amazon Relational Database Service
  - Microsoft SQL Azure
  - Oracle Cloud
  - Rackspace
- NoSQL
  - Google Datastore
  - Amazon SimpleDB
  - Microsoft Azure DocumentDB

© University of Melbourne 2018

Comparison of cloud db

	Amazon RDS (MySQL)	Microsoft SQL Azure	Google Datastore	Amazon SimpleDB
Type	**RDBMS**	**RDBMS**	**NoSQL**	**NoSQL**
Maximum amount of data that can be stored	1 terabyte per database[2]	50 gigabytes per database[3]	Not published for entire database, but 1 MB limit on a subset of data (called an entity). Limit to the number of indexes.	10 gigabytes per database domain (roughly equivalent to an RDBMS table)[4]
Ease of software portability with similar, locally hosted capability	High. MySQL instantiation in cloud is very similar to the local instantiated version.	High. Most SQL Server features are available in SQL Azure.	Medium/Low. Requires Java Data Objects or Datastore-specific interface and use of App Engine.	Medium. Requires SimpleDB-specific interface.
Transaction capabilities	Yes	Yes	Yes	Yes
Configurability and ability to tune database	High. MySQL instantiation in cloud.	Medium. Can create indexes and stored procedures, but no control over memory allocation or similar resources.	Low	Low
Database accessible as "stand-alone" offering	Yes	Yes	No. Requires Google App Engine application layer.	Yes
FISMA Certified	No	No	No	No
Can designate where data is stored (e.g., region or data center)	Yes	Yes	No	Yes
Replication	Yes	Yes	Yes	Yes

© University of Melbourne 2018

Pricing & Performance

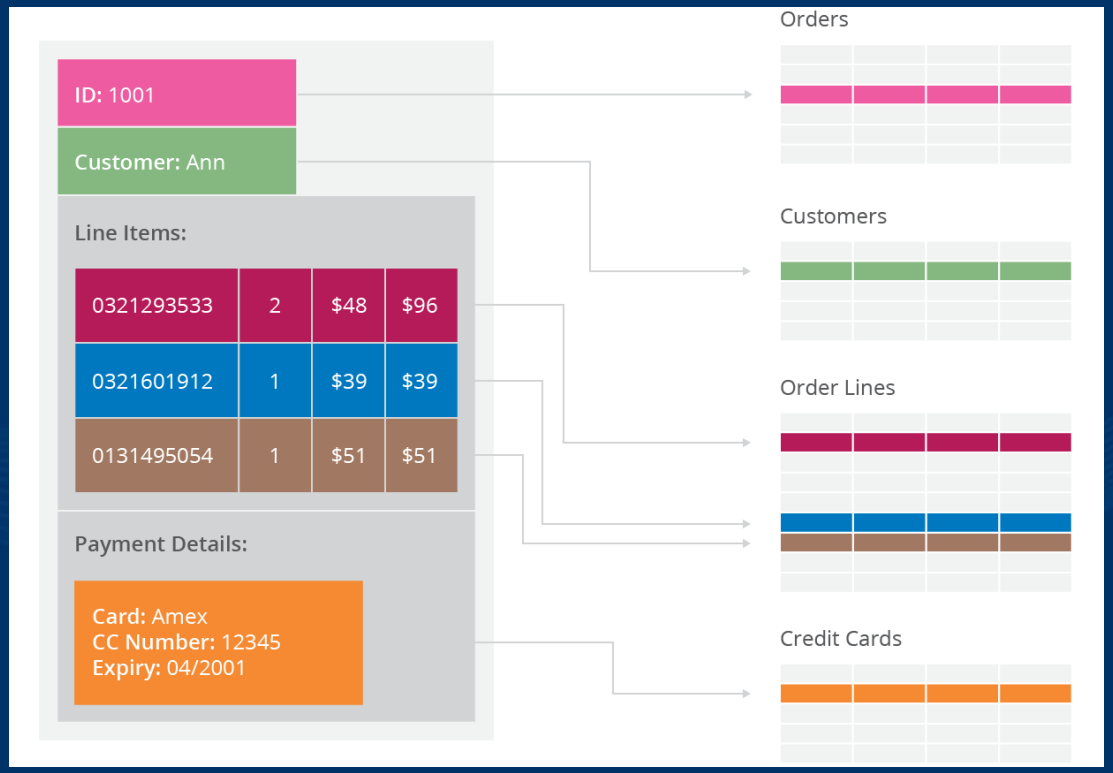
	Amazon RDS (MySQL)	Microsoft SQL Azure	Google Datastore	Amazon SimpleDB
Example Pricing for Processing (Refer to Sections 2-5 for details)*	Ranges from $0.11 per RDS hour for smallest instance to $2.60 per hour for largest instance	Ranges from $9.99 per database with up to 1 GB of storage to $499.95 per database with up to 50 GB of storage per month	$0.10 per App Engine CPU hour (required for accessing Datastore)	$0.14 per SimpleDB unit hour
Example "On-demand" Pricing for Data Transfers (Refer to Sections 2-5 for details)*	Inbound $0.10 per GB and outbound ranges from $0.15 per GB to $0.08 per GB, depending on volume	Inbound $0.10 per GB Outbound $0.15 per GB	Inbound $0.10 per GB Outbound $0.12 per GB	Inbound $0.10 per GB and outbound from $0.15 to $0.08 per GB, depending on volume.
Example Monthly Pricing for data storage (Refer to Sections 2-5 for details)*	$0.10 per GB plus $0.10 per 1 million I/O requests	Included in processing pricing	$0.15 per GB	$0.25 per GB

Single Client	Average writes per second	Average reads per second
Amazon MySQL RDS*	2,567	2,551
Microsoft SQL Azure	406	410
Google Datastore**	288	200
Amazon SimpleDB	208	63

5 Clients	Average writes per second	Average reads per second
Amazon MySQL RDS*	7576	7905
Microsoft SQL Azure	1737	1893
Google Datastore**	N/A	N/A
Amazon SimpleDB	689	281

© University of Melbourne 2018

Challenge: Object-orientated application software



Relational vs OO mismatch

- Relational databases
  - Individual entities are represented by a simple row structure
- Object-oriented programming
  - OO allows more complex data structures
  - complex objects must be normalized to fit relational tables
- How can an OO program persist data in a database?
  - relational database (simple, powerful, but not OO)
  - object-relational databases (complex data types, SQL 1999)
  - object-oriented databases (did not become popular)
  - object-relational mapping software (currently popular)
  - NoSQL (becoming popular)

© University of Melbourne 2018

Object-oriented DBMS

- Direct storage of object-oriented data
- Offers benefits of OO and Relational
  - directly persists objects
  - offers performance and security of relational DB
- Products include:
  - Db4o, InterSystems, McObject, Objectivity, Progress, Versant, ObjectDB, Gemstone, VelocityDB …
- But did not become popular in the marketplace
  - often tied to particular OO programming languages
  - lack of compatibility between different OODMBSs
  - lack of standard ad hoc query language
  - RDBMS are used to integrate applications
  - (but, re-emerging in the form of NoSQL ‘document databases’)

© University of Melbourne 2018

Responses

"The database market is back in play after a 30-year old freeze in which Oracle dominated the high end, and Microsoft the mid market. Then along came open source, the cloud, NoSQL, in memory and everything changed….The idea that everything is relational? Those days are gone."

Redmonk analyst James Governor

NoSQL

- Features
  - doesn't use relational model or SQL language
  - designed to run on distributed servers
  - most are open-source
  - built for the modern web
  - schema-less (though there may be an "implicit schema")
  - 'eventually consistent'
  - tend not to offer relational features like ACID, locks, joins
- Goals
  - to improve programmer productivity (OR mismatch)
  - to handle larger data volumes and throughput (big data)

from *NoSQL Databases: An Overview*
by Pramod Sadalage, Thoughtworks (2014)

© University of Melbourne 2018

NewSQL

- Features
  - like NoSQL, but offers ACID transactions
  - suited where consistency is important, e.g. financial data
  - offers traditional relational features like SQL, transactions
  - but scales horizontally like NoSQL systems
  - distributed, automatic sharding

- Example products
  - H-Store
  - VoltDB
  - Spanner

	CLOUD SPANNER	TRADITIONAL RELATIONAL	TRADITIONAL NON-RELATIONAL
Schema	✓ **Yes**	✓ Yes	✗ No
SQL	✓ **Yes**	✓ Yes	✗ No
Consistency	✓ **Strong**	✓ Strong	✗ Eventual
Availability	✓ **High**	✗ Failover	✓ High
Scalability	✓ **Horizontal**	✗ Vertical	✓ Horizontal
Replication	✓ **Automatic**	⟳ Configurable	⟳ Configurable

© University of Melbourne 2018

In-memory databases

- data are stored in memory and processed there
- disk is for long-term persistence
- primary motive is ***speed*** – of storage, transactions, analytics
- access speed of main memory is much faster than disk
- SAP HANA is the best-known example
- can use up to 100 terabytes memory
- supports OLTP and OLAP
- appeared in 2010
- (there are other in-memory databases)

© University of Melbourne 2018

Hadoop

- for data-processing, not operational database
- Hadoop = framework of software utilities including:
  - HDFS: (Hadoop Distributed File System) for distributed processing across a cluster of many computers
  - Hbase: database for data storage
  - MapReduce: for processing data across the cluster

- based originally on Google technology
- about 10 years old
- used by Yahoo, Facebook, available on cloud

© University of Melbourne 2018

Dr Greg Wadley

INFO90002 Database Systems & Information Modelling

Week 12

Pre-exam wrap-up

"The fundamental move from the cinematic form to the database form is a move from narrative and linearity to relationality. The database has no inherent narrative structure; there is no natural sequentiality to objects in a database. What structure the database provides is one of relations – between objects and their properties, from object to object, and so on."

- Paul Dourish (2017) *The Stuff of Bits.* p106

The semester, in summary

Learning about databases is essential because databases are:

- ubiquitous
  - there is a database behind most of today's exciting technologies
- standard
  - 30+ years of standardization mean that the relational database skills you learned will be applicable in any IT career
- starting to change
  - but recent challenges to the relational standard are creating significant opportunities for new players

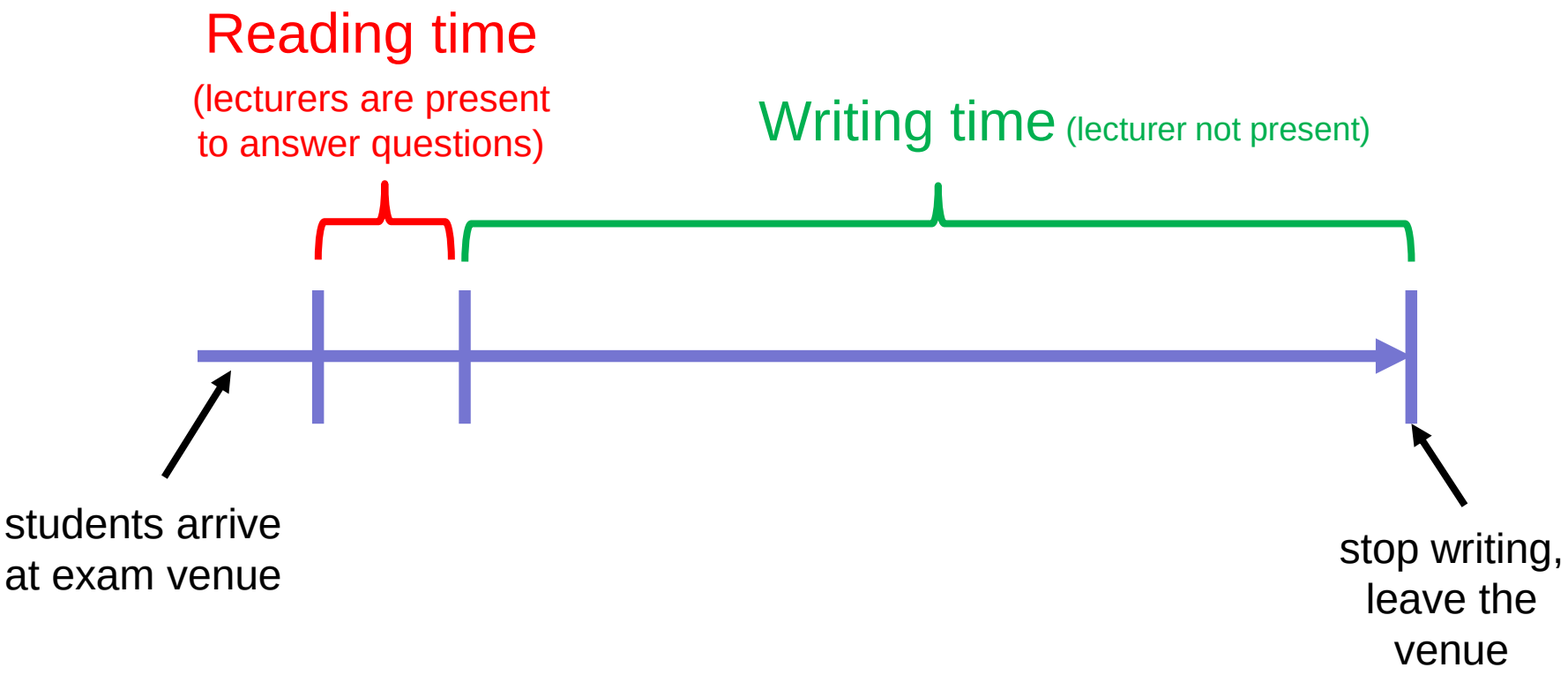
© University of Melbourne 2018

Exam Guide

- About the exam
  - covers a range of material from week 1 to week 12
  - about half of the marks are for Data Modelling and SQL
  - the rest are for other topics
  - closed book = don't bring any text materials with you
  - bring only a pen (or two) and your student card
  - 15 minutes of reading time + 3 hours writing
  - see https://students.unimelb.edu.au/admin/ (see Exam section)
- Hurdle requirement:
  - to pass the subject students must obtain at least:
    - 50% of the marks from non-exam based assessment
    - 50% of the marks from the exam
  - https://handbook.unimelb.edu.au/2018/subjects/info90002/assessment

© University of Melbourne 2018

Exam time-line



© University of Melbourne 2018

Exam page 1

THE UNIVERSITY OF MELBOURNE

School of Computing and Information Systems
End of Semester 2 2018 Examination

INFO90002 DATABASE SYSTEMS & INFORMATION MODELLING

Reading time: 15 minutes

Writing time: 180 minutes

This paper has 5 pages, including this page.

Authorised Materials:	No materials are authorised.

Instructions to Invigilators:

The examination paper IS TO REMAIN in the examination room.

Students are to be provided with standard answer books.

Instructions to Students:

Ensure your student number is written on all script books.

The total mark for this paper is 70 marks, representing 70% of your final assessment.

This exam paper has 10 questions, some with multiple parts. Attempt all questions.

Answer all questions on the *lined* pages of your answer book. The *unlined* pages of answer books are for draft working and notes and will not be marked.

If you fill your first answer book, you may ask for an extra one.

Start the answer to each numbered question on a new page in the answer book.

Write legibly in blue or black pen.

Textual answers can be in point form.

All electronic devices (including mobile phones, watches and calculators) must be switched off and remain under your desk throughout the examination.

No items, including electronic devices, may be taken to the bathroom.

© University of Melbourne 2018

Material examined

DM and SQL

further topics

Week	Starts	Lecture 1	Lecture 2	Lab / Lecture 3	Hoffer	Extra reading	Homework	Assessment
1	23 Jul	Intro to subject	Intro to databases	Intro to MySQL	ch 1	Wikipedia, Hoffer video, HIstory of databases	noun-verb analysis [answer]	
2	30 Jul	Designing a database	Implementing a database	lab: data modelling, SQL, MySQL guide & install SQL script	2	SE Radio 'Relational Databases'	Real Estate model [answer]	assignment 1 released
3	6 Aug	Data modelling 1	SQL 1	lab: data modelling, SQL	2, 6	Simsion ch 1, Hoffer video	Uni Courses model	
4	13 Aug	Data modelling 2		lab: data modelling, SQL	3, 7	Simsion chapter 3, Hoffer video	how would you model an *invoice*?	
5	20 Aug	Normalization	Physical design	lab: SQL skills 1 & Solutions	4	Simsion chapter 4, Hoffer video, Kent (1983) Normalization		assignment 2 released
6	27 Aug	SQL 2		lab: SQL skills 2	5	podcast on 'SQL' MySQL data types		Asst 1 due
7	3 Sep	Databases in applications	Web apps	lab: SQL skills 3	8, 14	O'Reilly video: Intro to Web		
8	10 Sep	Transactions and concurrency (lecture by Dave Eccles) scott.sql & king.sql	Distributed databases (lecture by Dave Eccles)	lab: SQLReferential Integrity & scott.sql	11	podcast on 'CAP Theorem' Panel discussion on distributed		Asst 2 due
9	17 Sep	Database Administration (lecture by Dave Eccles)	Security and Ethics (lecture by Dave Eccles)	discuss Asst 1		MySQL database administration Oracle database administration		
	24 Sep	mid semester break - no classes						
10	1 Oct	NoSQL databases	... continued	discuss Asst 2		Martin Fowler NoSQL overview		
11	8 Oct	guest lecture Andrew Cecchin, Associate in Risk and Forensic Consulting, PwC	guest lecture Tim Bishop, CEO, Man With A Van	revision 1: you choose the topics	9			
12	15 Oct	Industry Trends	Wrapup, discuss exam	revision 2: you choose the topics		how Facebook stores data Information Week article		

© University of Melbourne 2018

Type of questions

- Half (47%) = Data modelling and SQL
- Half (53%) = Short-answer questions about other material

  – must write in black or blue pen
    - DON'T use a red pen
    - pencil is only for drafting, not the answer to be marked
    - only the lined pages are marked
      (use the unlined pages for rough drafts)
    - start each new numbered question on a new page
    - examiners must be able to read your answers clearly

  – Since there are total 70 marks,
    and you have 3 hours writing time,
    we suggest you budget about 180/70 ~= 2.5 minutes per mark.

© University of Melbourne 2018

Short-answer questions: How much should I write?

- The exam does not prescribe a set quantity of text to write per question.
- We suggest instead that you budget your time according to the marks that each question is worth.
- Question words like "*list*", "*describe*" and "*explain*", in combination with the marks allotted to a question, give an indication of the amount of detail you are expected to provide.
- This is illustrated in the sample on the right. (The sample is simply a rough guide to structure and quantity.)

Q. List three relational database vendors (1 mark)

- Oracle
- Microsoft
- IBM

Q. Describe three relational database vendors (3 marks)

- Oracle Corporation, based in California, sell a range of software including their well-known database management system.
- Microsoft, based in Washington state USA, is the largest software manufacturer in the world, and offer a RDBMS called 'Sql Server'.
- IBM or "International Business Machine" began in 1911 and sell the relational DBMS "DB2".

Q. Describe the relational database vendor "Oracle", explaining its success. (6 marks)

Oracle is the second-largest software company in the world, second only to Microsoft. The company was founded in 1978 by Larry Ellison, who is still Oracle's CTO. "Oracle Database" has the largest market share of any relational DBMS.

A number of factors contribute to Oracle's success, including:

- it offered the first commercial RDBMS
- It has been a frequent innovator of new RDBMS features
- it offered RDBMS that ran on emerging Unix servers in the 1980s
- it offers a range of products + services including application software and training courses
- It has adapted to changes in the I.T. industry such as cloud + No SQL.

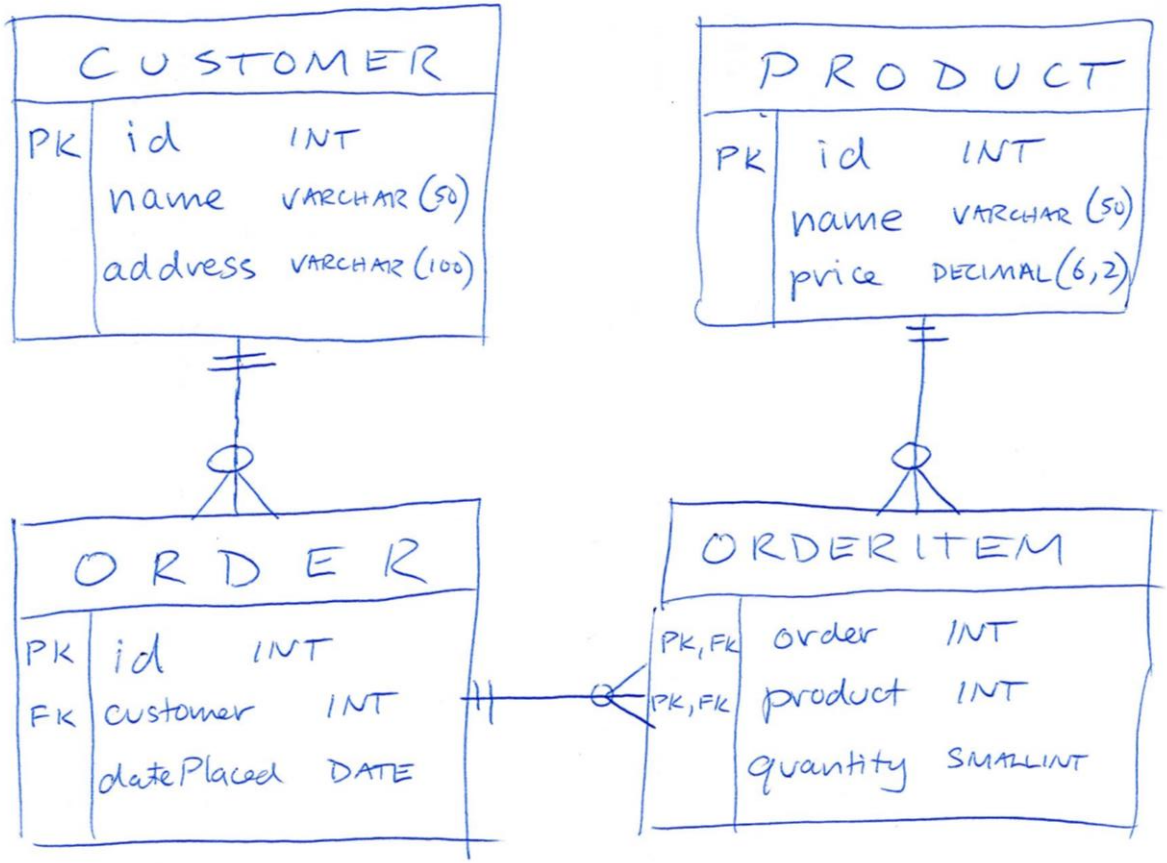
In 2010, Oracle aquired Sun Microsystems and thereby became the owner of Java, MySQL, and a Unix server OS.

Oracle's corporate campus at Redwood shores features buildings designed to look like hard disk drives.

© University of Melbourne 2018

Drawing ER diagrams by hand: what style should I use?

- You can use the following style to hand-draw an ER diagram. This contains most of the information you'd find in a Visio or Workbench diagram, except:
  - no need to worry about nullable/mandatory columns
  - no need to worry about dotted lines for non-identifying relationships



© University of Melbourne 2018

Summary of the course

- the following slides present a summary of the course
- they include *example* exam questions, to help you study
- these are probably NOT the actual exam questions!

© University of Melbourne 2018

Summary

- Weeks 1-2
  - Brief history of data storage and processing
    - paper, punch cards, computer files, relational databases
  - Overview of relational databases
    - tables, SQL language
  - Advantages of relational db over file processing
    - Data independence, Minimise redundancy, Improved consistency, Improved sharing, Reduced program maintenance, Better application development productivity, Improved data quality, Standards, Ad-hoc data queries … and some disadvantages
  - Database development lifecycle
    - Planning, Definition, Requirements Analysis, Design, Implementation, Data loading, Testing, Maintenance
    - Design = conceptual, logical, physical stages

© University of Melbourne 2018

Weeks 1-2 – Sample Questions

- Describe the phases of the database development lifecycle
- What are the advantages of relational databases?

© University of Melbourne 2018

Summary

- Weeks 3 – 6
  - Database Design
    - Conceptual, logical and physical
    - ER diagrams
    - How to work through conceptual, logical and physical design
  - SQL programming
    - DML statements
    - DDL statements
    - DCL statements

© University of Melbourne 2018

Weeks 3-6 Sample Questions

- Given "this description of a business case" produce a Conceptual / Logical / Physical data model
  - use homework examples for study: answers provided
  - and examples on next few slides: answers not provided
  - textbooks are full of worked examples

- Given "this data model", write SQL queries to:
  - insert
  - update
  - delete
  - select data

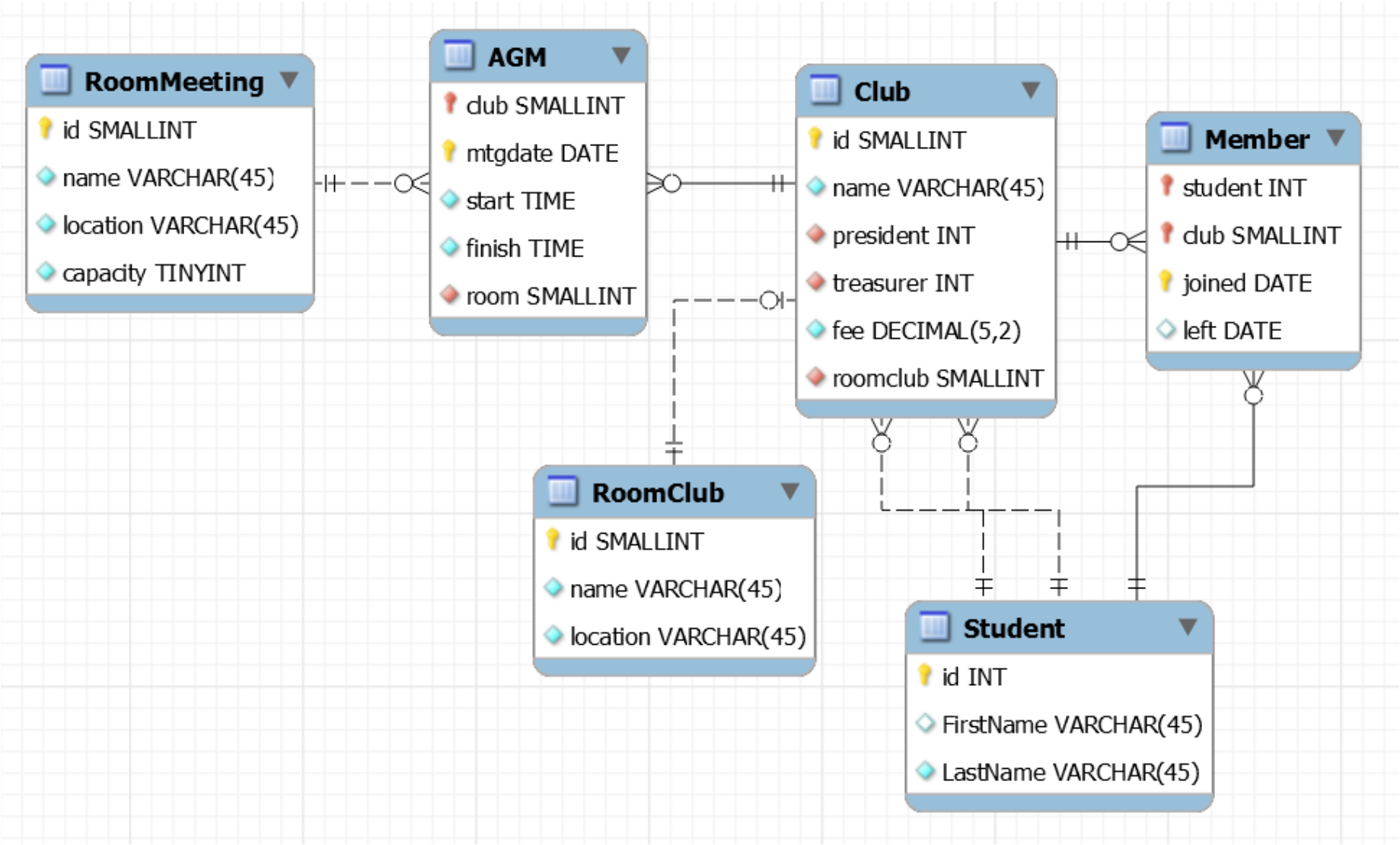
© University of Melbourne 2018

Extra data-modelling examples

- The Student Union wants a database to help manage up to 1000 clubs. Each club can have several hundred members, and any student can be a member of several clubs. Each club must have a president and a treasurer (a student can be the president or treasurer in more than one club). Each club can set its own membership fee.
- Each club is assigned a clubroom where it can hold meetings. Each clubroom is assigned to one club at most. The student union wants to keep track of which clubroom belongs to which club, but is not interested in keeping any information about the meetings held in the clubrooms.
- The clubrooms are too small for clubs to use for their annual general meetings (AGM), and so the Student Union also provides a number of shared meeting rooms specifically for this purpose. Each club informs the Student Union of the date and time it wishes to hold its AGM, and the Union then chooses a meeting room that is free at that date and time, and is large enough to hold that club's AGM.
- Information about club's finances, assets, etc. will not be maintained in this database.

© University of Melbourne 2018

Suggested model for student clubs



© University of Melbourne 2018

Extra data-modelling examples

- A major university's Department of Computing is offering degrees in 10 locations within Australia. Each year about 10,000 students graduate. The Department keeps track of each graduate's student number, birth name, country of birth, current country of citizenship, current name, current address, and the name of each subject the student successfully completed.
- In order to maintain strong ties to its graduates the Department holds many functions (dinners, receptions, conferences and seminars) around Australia. Events have a title, date, location and type. The Department keeps track of which students have attended each event. For each attendance of a graduate at an event, a comment can be recorded if new information about the graduate is discovered.
- The Department also keeps track of students via phone, mail and email. A member of staff in the department is assigned as a mentor for each graduand. Other information stored includes the location where the graduand attended the IS course. Because of the nature of the course a graduand may have attended the course at several locations.

© University of Melbourne 2018

Extra data-modelling examples

- A shipping magnate owns many container ships.
- Containers are collected at one port and delivered to another port.
- Customers pay a negotiated fee for the delivery of each individual container.
- Each ship has a sailing schedule that lists the ports the ship will visit over the next six months.
- The schedule shows expected arrival and departure dates at each port.
- The daily charge for use of each port is also included.
- Design a database to support this business.

© University of Melbourne 2018

Extra data-modelling examples

- Sally operates a cinema chain. She has given you the following requirements.
- “My cinema chain has many customers. Each cinema has multiple theatres.
- “Movies are shown throughout the day starting at 11AM and finishing at 1AM. Each movie is given a two hour time slot. We never show a movie in more than one theatre at a time, but we do shift movies between theatres as the required number of seats changes.
- “I am interested in knowing how many people, classified into adults and children, attend each showing of a movie. I vary ticket prices by movie and time slot. For instance, *Mad Max 7* at 10AM costs $10 for everyone, but at 11PM it is $15.”

© University of Melbourne 2018

Summary

- Week 5 (also informs data modelling)
  - Normalisation
    - functional dependency, determinants
    - 1st, 2nd, 3rd, BC normal forms
    - what happens if the database isn't normalised
    - an example of normalising a set of relations
  - Physical Implementation
    - inputs to and decisions made in the physical design stage
    - choosing correct data types - why this is important
    - de-normalisation
    - when to create indexes

© University of Melbourne 2018

Week 5 Sample Questions (normalization)

- What is normalisation and why is it used?
- Describe 1st, 2nd and 3rd normal form
- Given the following relation, carry out normalisation to 3rd normal form. …
- Are the following relations in 3rd (or 2nd or 1st) normal form? …

© University of Melbourne 2018

Week 5 Sample Questions (physical design)

- What are the inputs to the (or what are the decisions made in the) Physical design process?
- What is de-normalisation, how is it used, give an example?
- When should we use indexing?
- What data types should we use for the following columns? …

© University of Melbourne 2018

Summary

- Week 7 – Databases in Applications
  - Reasons we put applications between users and databases
    - Presentation / UI
    - Business logic
  - Embedding SQL in application software
  - 1, 2, 3-tiered architectures
  - Web applications

© University of Melbourne 2018

Week 7 Sample Questions

- Why should the presentation, business logic and data be separated in internet-based database applications?
- What are the advantages of 2/3/4 tier application architectures?
- How has the web impacted database design and administration?

© University of Melbourne 2018

Summary

- Week 8 - Transactions and Concurrency
  - Definition of a transaction
  - Transaction properties
    - ACID & Serializability
  - Transaction logging
  - Concurrency
    - problems:
      - lost updates, uncommitted data, inconsistent retrieval
    - how to handle these problems
    - locking: types and granularity

© University of Melbourne 2018

Week 8 Sample Questions (transactions)

- What does “atomic” mean?
- Explain the three concurrency problems
- What is the purpose of locking,
  and what types of locks can be used?
- How does the optimistic method solve the concurrency problem?

© University of Melbourne 2018

Summary

- Week 8 - Distributed databases
  - Defined the term distributed database
  - Advantages and disadvantages
  - Looks at the objectives and trade-offs in distributed database
    - ease of access / location transparency, local autonomy
    - asynchronous / synchronous updates
  - Data replication
    - types
    - advantages and disadvantages
    - description and comparison of options

© University of Melbourne 2018

Week 8 Sample Questions (distributed databases)

- What is a distributed database?
- Discuss the advantages and disadvantages of different distributed database options
- What is partitioning, how does each type work?
- What things does a DDBMS provide in addition to DBMS functionality?

© University of Melbourne 2018

Summary

- Week 9 - Database Administration, Security, Ethics
  - The DBA and Data Administrator roles
  - Architecture of a RDBMS
    - components and their interactions
  - Performance
    - what affects database performance
    - common approaches to monitoring and tuning performance
  - Security
    - threats
    - access control
    - web app security, SQL injection
  - Backup and Recovery
    - types of failure, types of backups
    - other methods to reduce likelihood of data loss

© University of Melbourne 2018

Week 9 Sample Questions

- Describe and differentiate the DBA and DA roles
- Describe common mechanisms used to improve the performance of database servers
- Security measures for databases include access control and encryption – describe and explain these
- What are the different types of backup?

© University of Melbourne 2018

Summary

- Week 10 and 12 – NoSQL and Industry Trends
  The relational status-quo
  - major vendors
  - current offerings
- Challenges to the relational status quo
  - big data
  - cloud storage
  - object-oriented applications
- Responses
  - NoSQL
  - NewSQL
  - Hadoop
  - In-memory databases

© University of Melbourne 2018

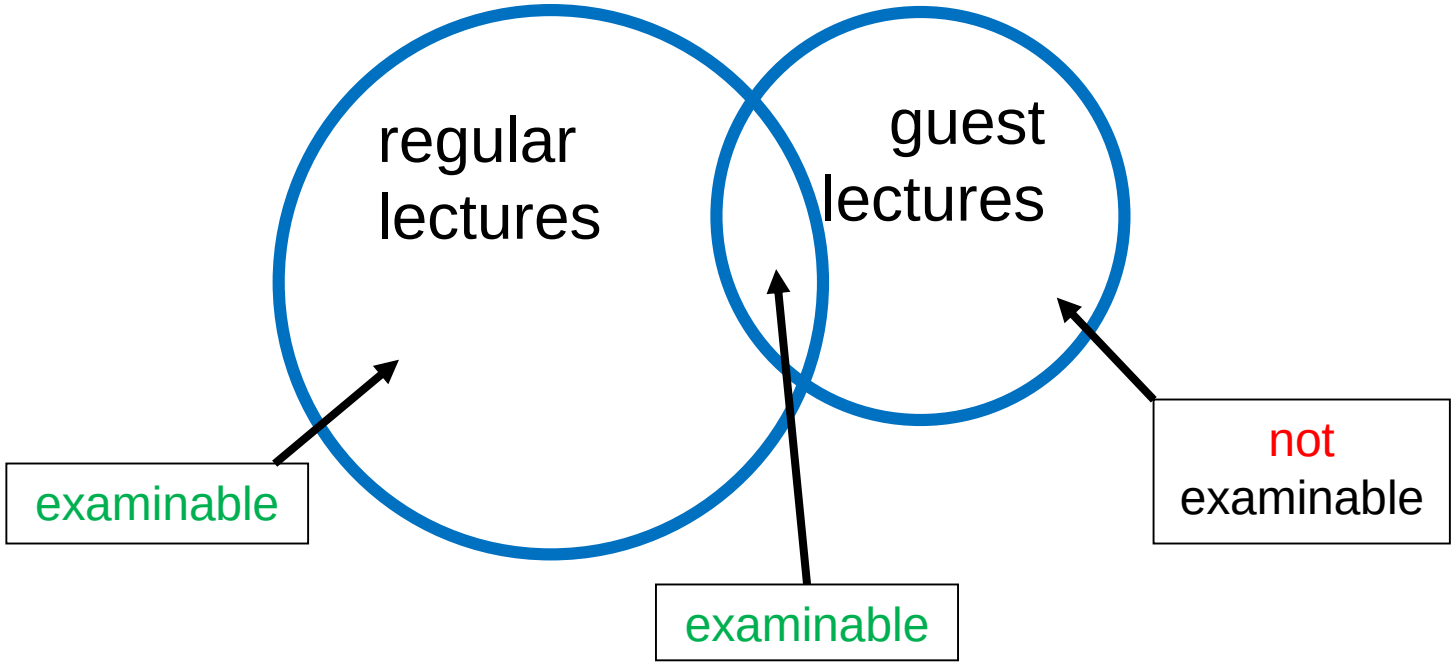
Week 10/12 Sample Questions

- Who are the leading commercial and open-source vendors?
- What are the pros and cons of storing data in the cloud?
- What is driving the adoption of cloud storage?
- What is "eventual consistency"?

© University of Melbourne 2018

Guest lectures

- Guest lectures in week 11 are not *directly* examined



© University of Melbourne 2018

THE UNIVERSITY OF MELBOURNE

Student Experience Survey

Subject Experience Survey (SES)

- **The SES is available now for you to fill in**
- **https://ses.unimelb.edu.au/#students**
- **Your feedback helps us to continuously improve our subjects**

I am particularly interested in your views on:

- assignments – more/less?
- guest lectures – more/less?
- modelling – more/less?
- SQL – more/less?
- NoSQL – more/less?
- labs – SQL only, or other topics?
- what would you add, remove or change about the course?

© University of Melbourne 2018

Thank you!

- Thank you for taking INFO90002
- We hope you find it useful for your career
- Good luck in the exam!