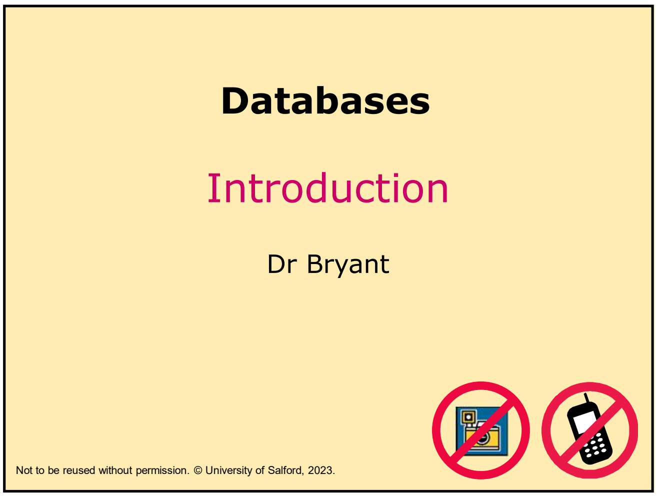
Database Systems Lecture Slides

CRN 32741, UMC G400 10045

Dr Bryant

Semester 1 of 2023/2024

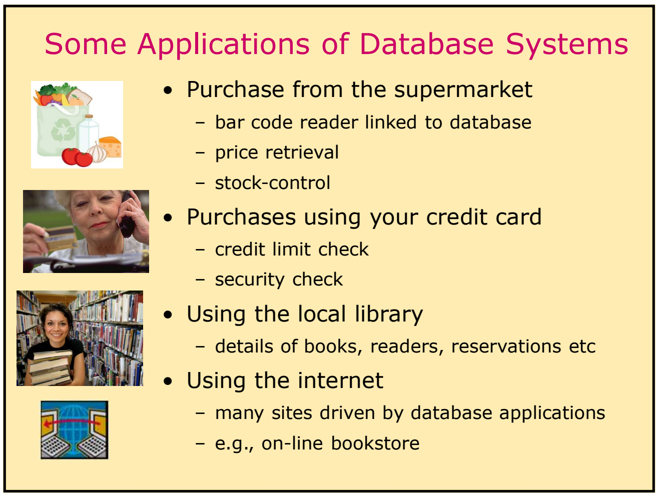


Databases

Introduction

Dr Bryant

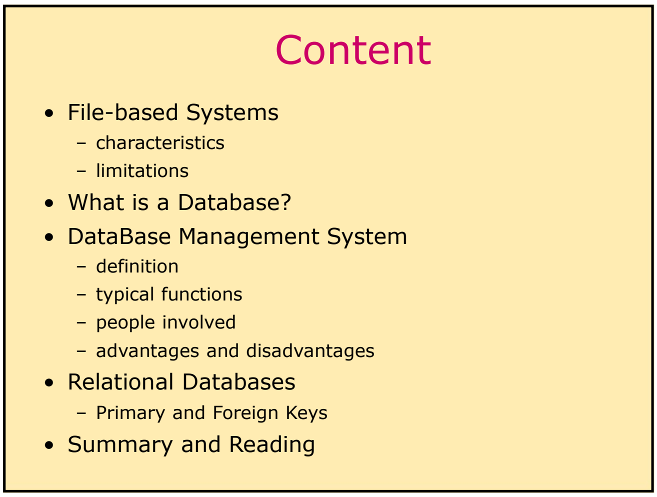
1



Some Applications of Database Systems

- Purchase from the supermarket
  - bar code reader linked to database
  - price retrieval
  - stock-control
- Purchases using your credit card
  - credit limit check
  - security check
- Using the local library
  - details of books, readers, reservations etc
- Using the internet
  - many sites driven by database applications
  - e.g., on-line bookstore

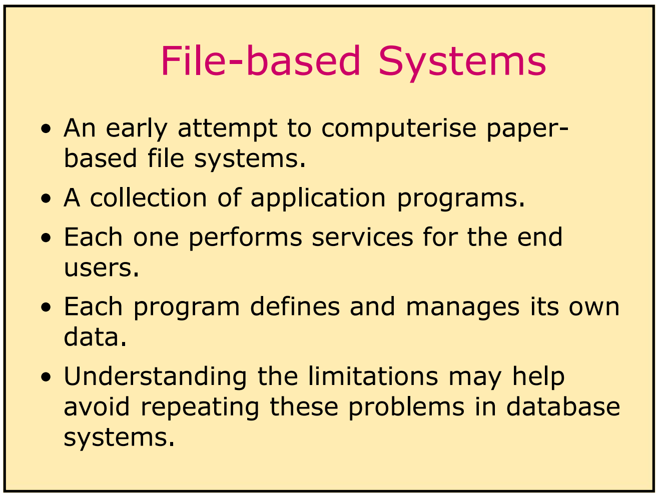
2



Content

- File-based Systems
  - characteristics
  - limitations
- What is a Database?
- DataBase Management System
  - definition
  - typical functions
  - people involved
  - advantages and disadvantages
- Relational Databases
  - Primary and Foreign Keys
- Summary and Reading

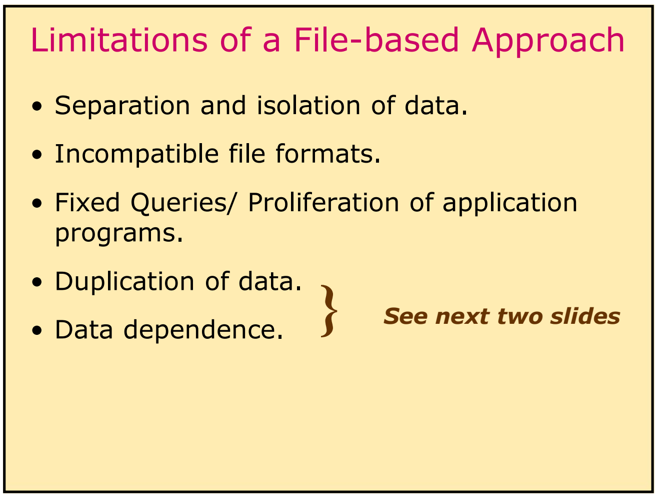
3



File-based Systems

- An early attempt to computerise paper-based file systems.
- A collection of application programs.
- Each one performs services for the end users.
- Each program defines and manages its own data.
- Understanding the limitations may help avoid repeating these problems in database systems.

4

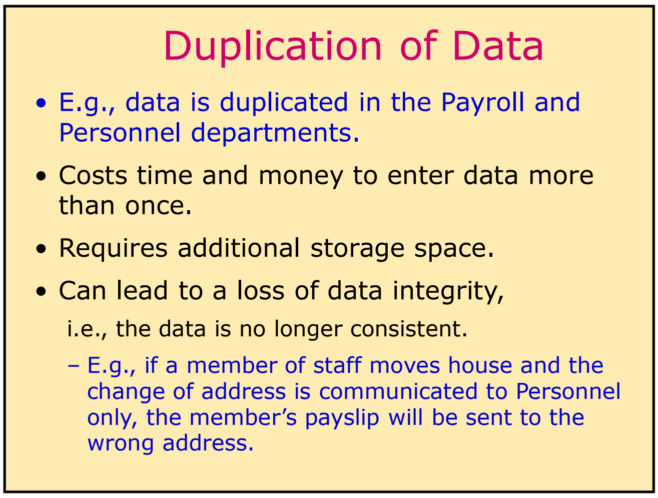


Limitations of a File-based Approach

- Separation and isolation of data.
- Incompatible file formats.
- Fixed Queries/ Proliferation of application programs.
- Duplication of data.
- Data dependence.

} ***See next two slides***

5



Duplication of Data

- E.g., data is duplicated in the Payroll and Personnel departments.
- Costs time and money to enter data more than once.
- Requires additional storage space.
- Can lead to a loss of data integrity,

  i.e., the data is no longer consistent.
  - E.g., if a member of staff moves house and the change of address is communicated to Personnel only, the member's payslip will be sent to the wrong address.

6

Data Dependence

- The physical structure and storage of the data files and records are defined in the application code.
- Makes it difficult to change structure.
- Any change to the structure could result in many programs having to be modified and retested.

7

Definitions of a Database

1. A shared collection of logically related data (and a description of this data), designed to meet the information needs of an organisation.
2. A self-describing collection of integrated records.

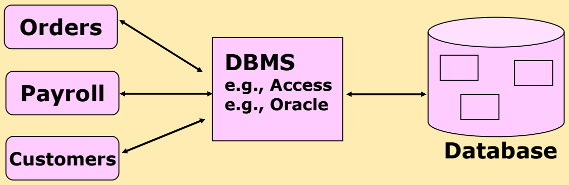
The database holds not only the organisation's operational data but also a description of this data.

We will consider what is meant by "logically related" in a subsequent lecture.

8

A Simple View of a Database System

- Database Systems are characterised by:
  - A collection of data stored on files (a.k.a. the **Database**).
  - A piece of software called a **DataBase Management System (DBMS).**
  - A variety of users who use **User Programs.**



Access to the Database is controlled by the DBMS.

9

Content

- File-based Systems
  - characteristics
  - limitations
- What is a Database?
- **DataBase Management System**
  - definition
  - typical functions
  - people involved
  - advantages and disadvantages
- Relational Databases
  - Primary and Foreign Keys
- Summary and Reading

10

DataBase Management System (DBMS)

A software system that enables users to define, create, and maintain the database and which provides controlled access to this database.

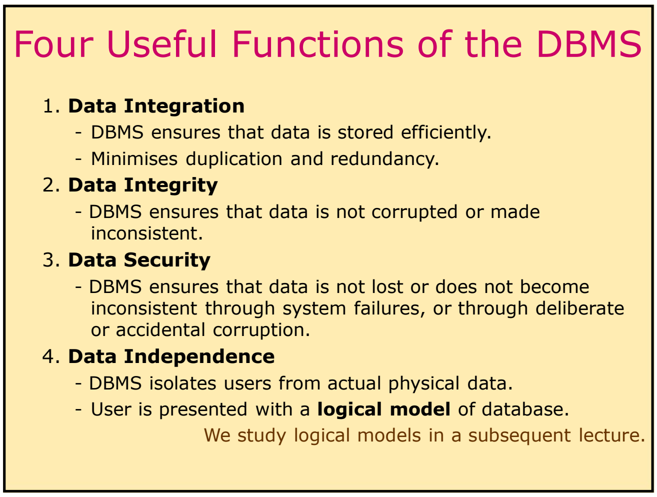
11

The DBMS *(continued)*

- As users of a database we use the DBMS to access data in the following ways:
  1. Add new data
  2. Delete data
  3. Update data
  4. Retrieve data
- The above can be achieved through **User Programs.**

12

Dr C.H. Bryant, School of SEE, Salford, UK.

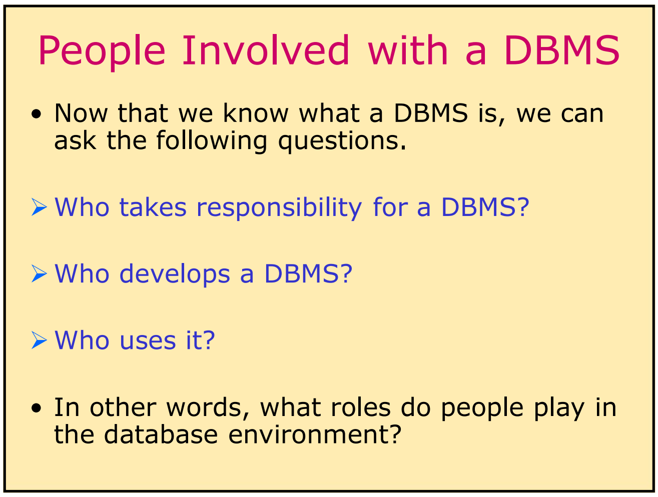


Four Useful Functions of the DBMS

1. **Data Integration**
   - DBMS ensures that data is stored efficiently.
   - Minimises duplication and redundancy.
2. **Data Integrity**
   - DBMS ensures that data is not corrupted or made inconsistent.
3. **Data Security**
   - DBMS ensures that data is not lost or does not become inconsistent through system failures, or through deliberate or accidental corruption.
4. **Data Independence**
   - DBMS isolates users from actual physical data.
   - User is presented with a **logical model** of database.

We study logical models in a subsequent lecture.

13



People Involved with a DBMS

- Now that we know what a DBMS is, we can ask the following questions.

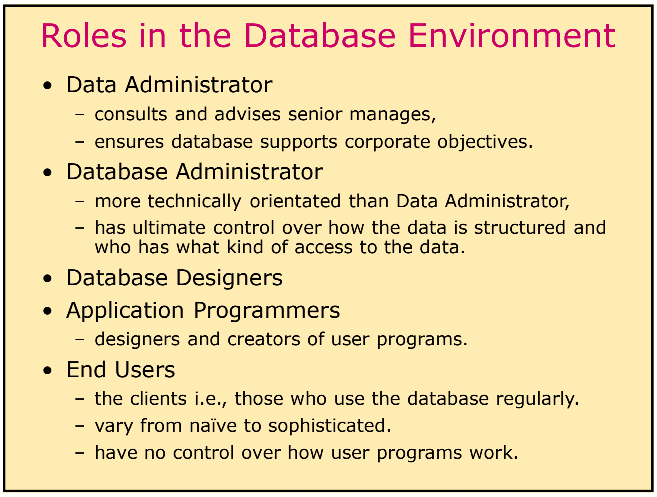
➢ Who takes responsibility for a DBMS?

➢ Who develops a DBMS?

➢ Who uses it?

- In other words, what roles do people play in the database environment?

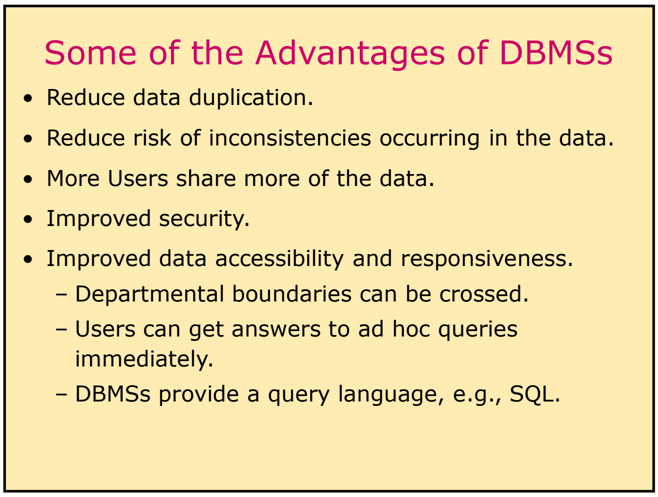
14



Roles in the Database Environment

- Data Administrator
  - consults and advises senior manages,
  - ensures database supports corporate objectives.
- Database Administrator
  - more technically orientated than Data Administrator,
  - has ultimate control over how the data is structured and who has what kind of access to the data.
- Database Designers
- Application Programmers
  - designers and creators of user programs.
- End Users
  - the clients i.e., those who use the database regularly.
  - vary from naïve to sophisticated.
  - have no control over how user programs work.

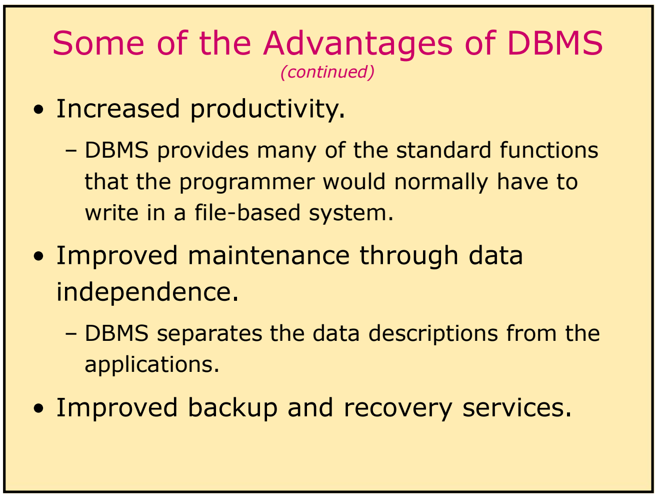
15



Some of the Advantages of DBMSs

- Reduce data duplication.
- Reduce risk of inconsistencies occurring in the data.
- More Users share more of the data.
- Improved security.
- Improved data accessibility and responsiveness.
  - Departmental boundaries can be crossed.
  - Users can get answers to ad hoc queries immediately.
  - DBMSs provide a query language, e.g., SQL.

16

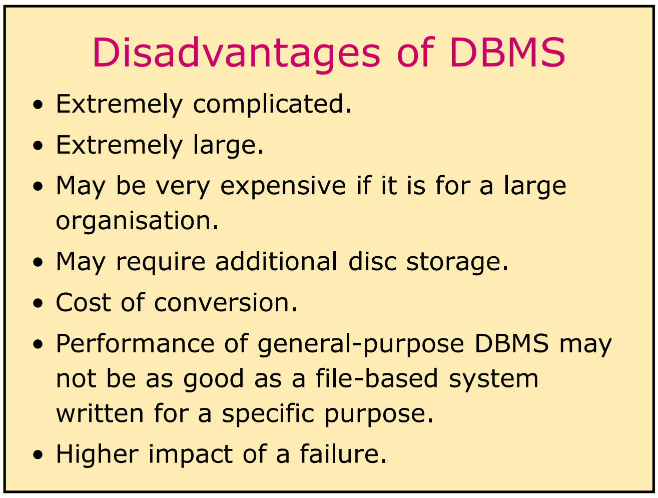


Some of the Advantages of DBMS

(continued)

- Increased productivity.
  - DBMS provides many of the standard functions that the programmer would normally have to write in a file-based system.
- Improved maintenance through data independence.
  - DBMS separates the data descriptions from the applications.
- Improved backup and recovery services.

17



Disadvantages of DBMS

- Extremely complicated.
- Extremely large.
- May be very expensive if it is for a large organisation.
- May require additional disc storage.
- Cost of conversion.
- Performance of general-purpose DBMS may not be as good as a file-based system written for a specific purpose.
- Higher impact of a failure.

18

Summary so Far

- Understanding the limitations of a file-based approach may help avoid repeating these problems in database systems.
- A database is a shared collection of logically related data (and a description of this data), designed to meet the information needs of an organisation.
- A DBMS is a software system that enables users to define, create, and maintain the database and which provides controlled access to this database.
- Users of a database use the DBMS to add, delete, update and retrieve data.

19

Content

- File-based Systems
  - characteristics
  - limitations
- What is a Database?
- Data Base Management System
  - definition
  - typical functions
  - people involved
  - advantages and disadvantages
- **Relational Databases**
  - Primary and Foreign Keys
- Summary and Reading

20

The Relational DBMS

- The dominant DBMS in use today.
- Estimated sales of approximately $15-$20 billion per year.
- Long history by computing standards.
  - Based on a seminal paper by E.F.Codd published in 1970.
- Simple logical structure.
- Sound theoretical foundation.

21

Relational Databases

In Relational Databases, data is stored in files called relations (a.k.a. tables).

- A relation consists of rows and columns.
- A relation carries data on one kind of entity, e.g., students.

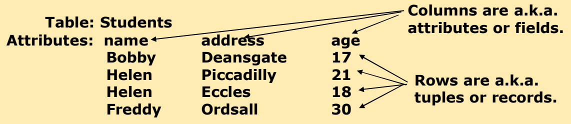


Table: Students

Attributes:	name	address	age
	Bobby	Deansgate	17
	Helen	Piccadilly	21
	Helen	Eccles	18
	Freddy	Ordsall	30

Columns are a.k.a. attributes or fields.

Rows are a.k.a. tuples or records.

Convention for describing relations is:

entity-name (<list of attributes>)

E.g., Students(name, address, age).

22

Duplicate Values

Table: Students

Attributes:	name	address	age
	Bobby	Deansgate	17
	Helen	Piccadilly	21
	Helen	Eccles	18
	Freddy	Ordsall	30

The problem with the above table is that *Helen* cannot be uniquely identified.

We need a column in the table which has a set of values containing no duplicates, so that every row is uniquely identified.

- No such column exists in the above table.
- The above table is not strictly a relation.
- We need to add a PRIMARY KEY attribute.

23

Primary Key

Hence

Table: Students

Attributes:	name	address	age
	Bobby	Deansgate	17
	Helen	Piccadilly	21
	Helen	Eccles	18
	Freddy	Ordsall	30

becomes

Table: Students

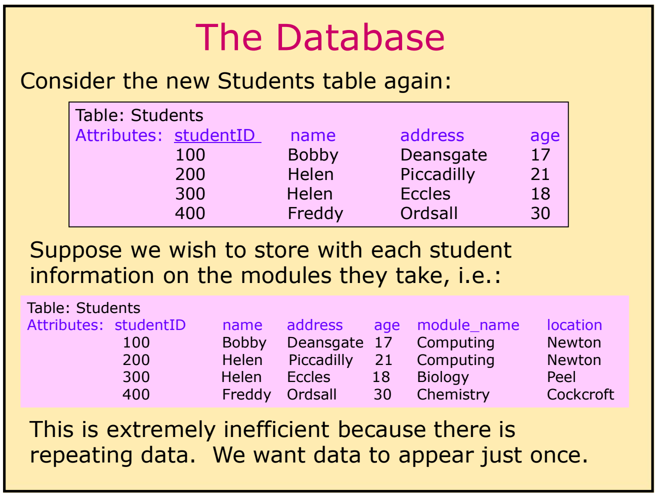
Attributes:	studentID	name	address	age
	100	Bobby	Deansgate	17
	200	Helen	Piccadilly	21
	300	Helen	Eccles	18
	400	Freddy	Ordsall	30

Students table is now described as:

Students(studentID, name, address, age)

i.e., underline the primary key and make it 1st attribute in list.

24



The Database

Consider the new Students table again:

Table: Students

Attributes:	studentID	name	address	age
	100	Bobby	Deansgate	17
	200	Helen	Piccadilly	21
	300	Helen	Eccles	18
	400	Freddy	Ordsall	30

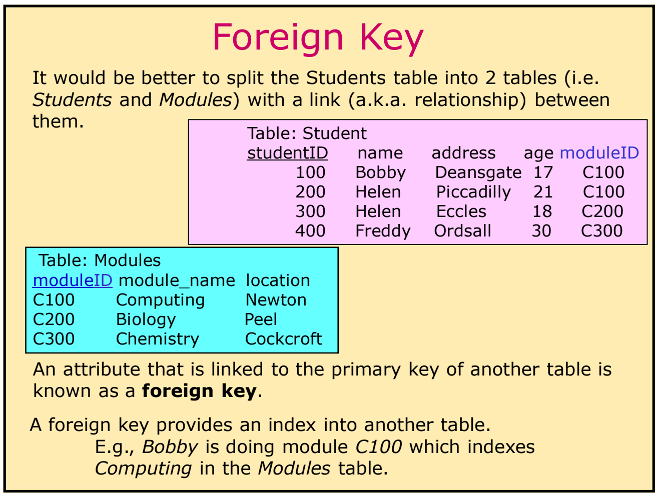
Suppose we wish to store with each student information on the modules they take, i.e.:

Table: Students

Attributes:	studentID	name	address	age	module_name	location
	100	Bobby	Deansgate	17	Computing	Newton
	200	Helen	Piccadilly	21	Computing	Newton
	300	Helen	Eccles	18	Biology	Peel
	400	Freddy	Ordsall	30	Chemistry	Cockcroft

This is extremely inefficient because there is repeating data. We want data to appear just once.

25



Foreign Key

It would be better to split the Students table into 2 tables (i.e. *Students* and *Modules*) with a link (a.k.a. relationship) between them.

Table: Student

studentID	name	address	age	moduleID
100	Bobby	Deansgate	17	C100
200	Helen	Piccadilly	21	C100
300	Helen	Eccles	18	C200
400	Freddy	Ordsall	30	C300

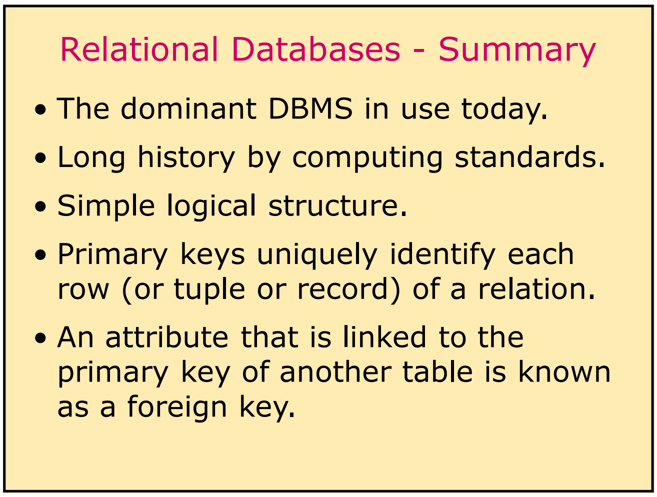
Table: Modules

moduleID	module_name	location
C100	Computing	Newton
C200	Biology	Peel
C300	Chemistry	Cockcroft

An attribute that is linked to the primary key of another table is known as a **foreign key**.

A foreign key provides an index into another table.
E.g., *Bobby* is doing module *C100* which indexes *Computing* in the *Modules* table.

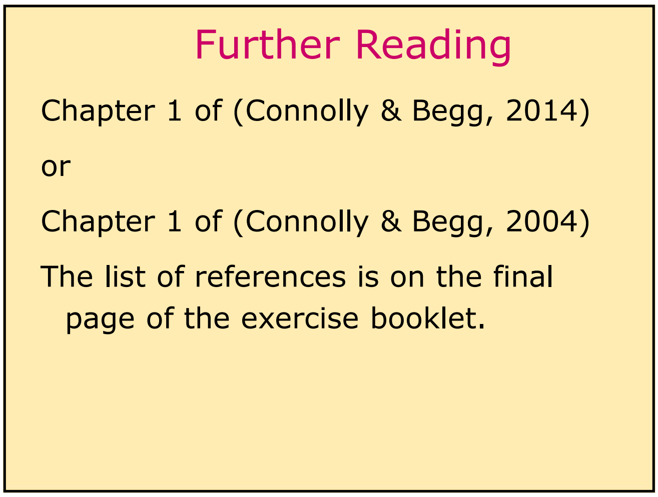
26



Relational Databases - Summary

- The dominant DBMS in use today.
- Long history by computing standards.
- Simple logical structure.
- Primary keys uniquely identify each row (or tuple or record) of a relation.
- An attribute that is linked to the primary key of another table is known as a foreign key.

27



Further Reading

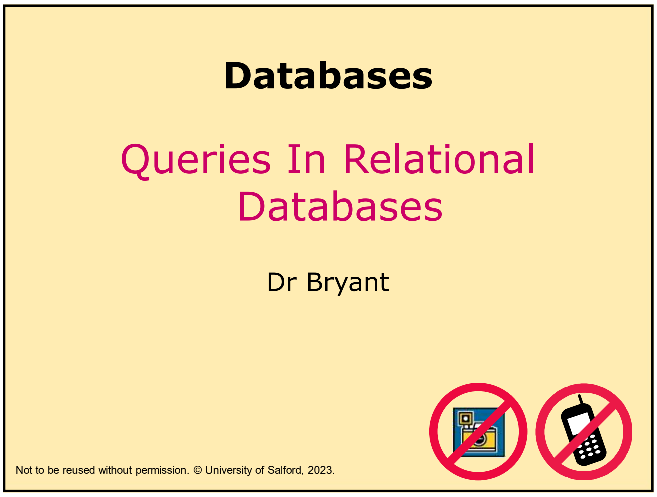
Chapter 1 of (Connolly & Begg, 2014)

or

Chapter 1 of (Connolly & Begg, 2004)

The list of references is on the final page of the exercise booklet.

28

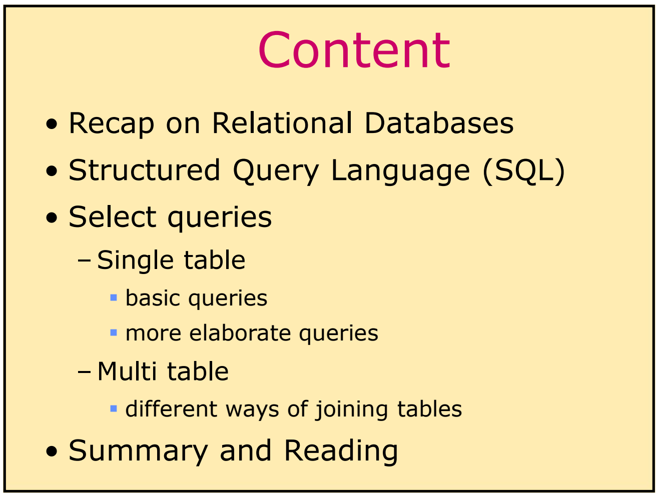


Databases

Queries In Relational Databases

Dr Bryant

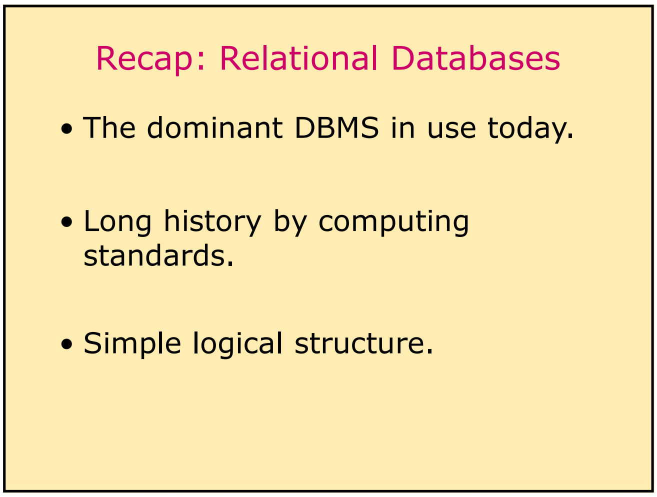
1



Content

- Recap on Relational Databases
- Structured Query Language (SQL)
- Select queries
  - Single table
    - basic queries
    - more elaborate queries
  - Multi table
    - different ways of joining tables
- Summary and Reading

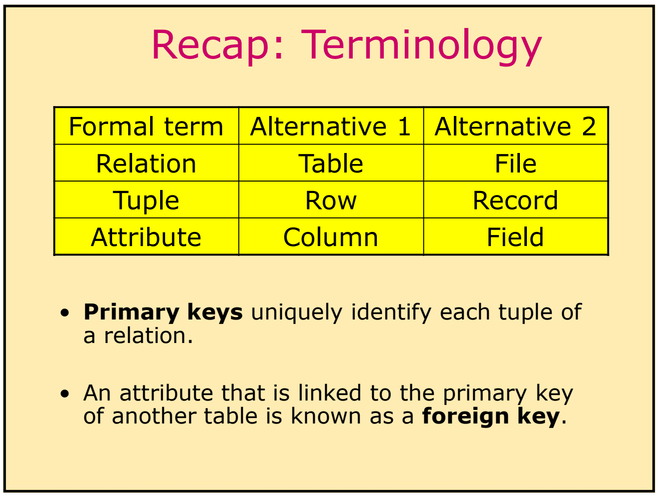
2



Recap: Relational Databases

- The dominant DBMS in use today.
- Long history by computing standards.
- Simple logical structure.

3

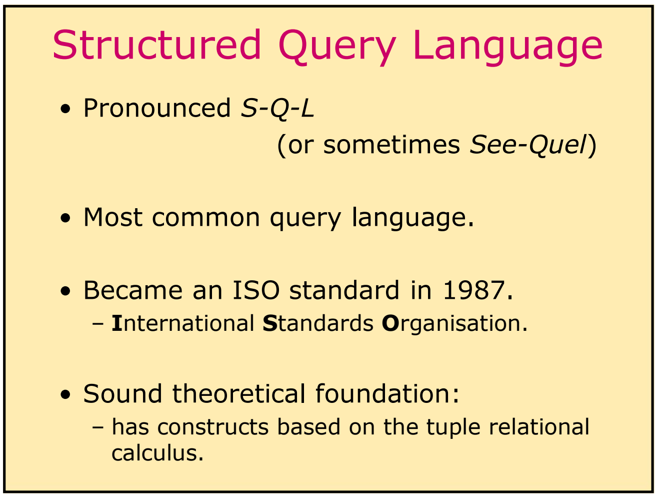


Recap: Terminology

Formal term	Alternative 1	Alternative 2
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

- **Primary keys** uniquely identify each tuple of a relation.
- An attribute that is linked to the primary key of another table is known as a **foreign key**.

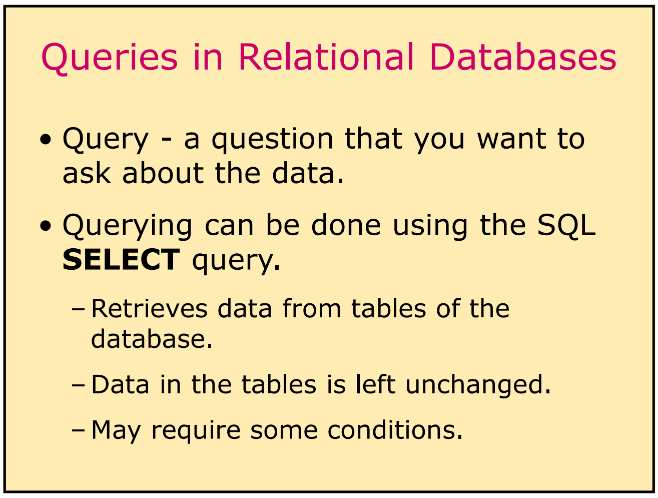
4



Structured Query Language

- Pronounced *S-Q-L* (or sometimes *See-Quel*)
- Most common query language.
- Became an ISO standard in 1987.
  - **I**nternational **S**tandards **O**rganisation.
- Sound theoretical foundation:
  - has constructs based on the tuple relational calculus.

5



Queries in Relational Databases

- Query - a question that you want to ask about the data.
- Querying can be done using the SQL **SELECT** query.
  - Retrieves data from tables of the database.
  - Data in the tables is left unchanged.
  - May require some conditions.

6

SQL SELECT statement

- Queries may be created in a DBMS using the SQL SELECT statement.
- The basic standard syntax is:

```
SELECT <list of attributes>
FROM <table name>
[ WHERE <condition(s)> ];
```

Square brackets indicate WHERE clause is optional.

SELECT specifies which columns are to appear in the result.

FROM specifies the table(s) to be used.

WHERE filters the rows subject to some condition.

7

Content

- Recap on Relational Databases
- Structured Query Language (SQL)
- Select queries
  - **Single table**
    - basic queries
    - more elaborate queries
  - Multi table
    - different ways of joining tables
- Summary and Reading

8

Subject of Examples

- The next 10 examples I will give during this lecture concern data on:
  - drivers;
  - number of penalty points on their licence.

9

Single-Table Select Queries

Consider the following table on which we will perform single-table select queries:

Drivers

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

The results of a query are data laid out in a tabular manner and are not stored permanently.

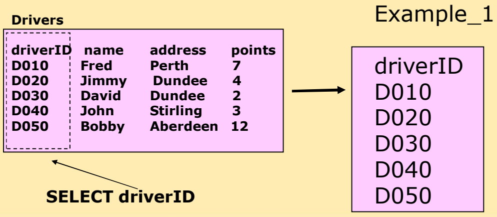
10

SELECT Example 1

EXAMPLE: List all the ids (driverID) from the Drivers table.

```
SELECT driverID
FROM Drivers;
```

NOTE: There is no use of the WHERE clause.



Drivers

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

Example_1

driverID
D010
D020
D030
D040
D050

SELECT driverID

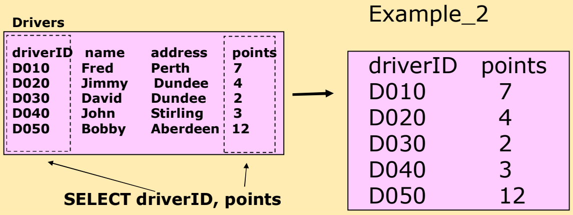
11

SELECT Example 2

EXAMPLE: List all the ids (driverID) and points from the Drivers table.

```
SELECT driverID, points
FROM Drivers;
```

NOTE: We use commas to separate columns (attributes).



Drivers

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

Example_2

driverID	points
D010	7
D020	4
D030	2
D040	3
D050	12

SELECT driverID, points

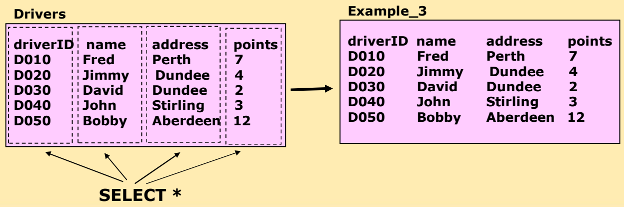
12

SELECT Example 3

EXAMPLE: List all the attributes (i.e., all the columns) from the Drivers table.

```
SELECT *
FROM Drivers;
```

NOTE: We use * (asterisk) for all columns (attributes).



Drivers

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

Example_3

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

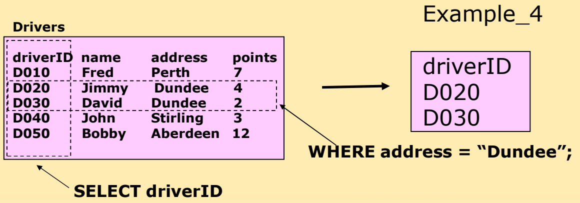
SELECT *

13

SELECT Example 4

EXAMPLE: List all the ids (driverID) of the drivers in the Drivers table who are from Dundee.

```
SELECT driverID
FROM Drivers
WHERE address = “Dundee”;
```



Drivers

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

Example_4

driverID
D020
D030

WHERE address = “Dundee”;

SELECT driverID

14

Content

- Recap on Relational Databases
- Structured Query Language (SQL)
- Select queries
  - Single table
    - basic queries
    - **more elaborate queries**
  - Multi table
    - different ways of joining tables
- Summary and Reading

15

Extending SELECT

- Consider the following questions.

List all the names and addresses of the drivers whose address begin with the letters “St”.

List the names of drivers from Dundee or Aberdeen.

Calculate the number of drivers.

List the names of drivers whose licence points are between 7 and 15.

List the names of drivers in alphabetical order.

- Can we write SELECT statements to answer these questions?

To do so, we need to go beyond the basic syntax.

16

The LIKE clause

- Sometimes we may want to pattern match within certain columns (attributes) of a table.
- E.g., list the names and addresses of drivers whose addresses begin with string “St”.

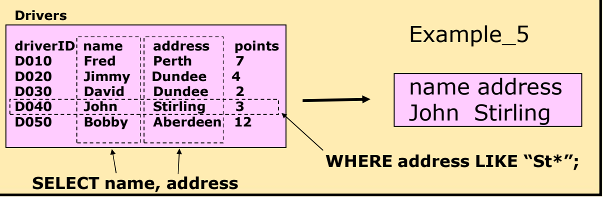
Standard	Access	Meaning	Example
_ underscore	?	matches a single character	LIKE “Sm?th”
%	*	matches any number of characters	LIKE “St*”

17

SELECT Example 5

EXAMPLE: List all the names and addresses of the drivers whose address begin with the letters “St” from the Drivers table.

```
SELECT name, address
FROM Drivers
WHERE address LIKE “St*”;
```



Drivers

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

Example_5

name	address
John	Stirling

WHERE address LIKE “St*”;

SELECT name, address

18

The IN Operator

The IN operator is used to check if an attribute(s) has a value from a set of values.

Syntax is: **[NOT] IN (<value-list>)**

EXAMPLE 6: Given the following table:

Drivers

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

a) List the names of drivers from Dundee or Aberdeen.

```
SELECT name
FROM Drivers
WHERE address IN ('Dundee', 'Aberdeen');
```

Example_6a

name
Jimmy
David
Bobby

b) List the names of drivers *not* from Dundee or Aberdeen.

```
SELECT name
FROM Drivers
WHERE address NOT IN ('Dundee', 'Aberdeen');
```

Example_6b

name
Fred
John

Aggregate Functions

Aggregate functions perform calculations on the values in **the column of a table.**

COUNT - counts the number of values in a column.

SUM - calculates the sum (total) of all values in a column.

AVG - calculates the average of all values in a column.

MAX - gets the maximum value in a column.

MIN - gets the minimum value in a column.

SELECT Example 7

Calculate

- the number of drivers,
- the total number of points,
- the average number of points, and
- the range of points from the Drivers table.

```
SELECT COUNT(driverID), SUM(points), AVG(points),
MAX(points) - MIN(points)
FROM Drivers;
```

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

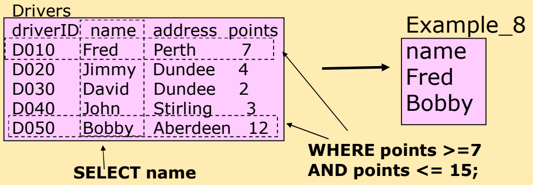
Example_7

COUNT(driverID)	SUM(points)	AVG(points)	MAX(points)-MIN(points)
5	28	5.6	10

SELECT Example 8

EXAMPLE: List the names of drivers whose licence points are between 7 and 15 from the Drivers table.

```
SELECT name
FROM Drivers
WHERE points >= 7 AND points <= 15;
```



Drivers

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

Example_8

name
Fred
Bobby

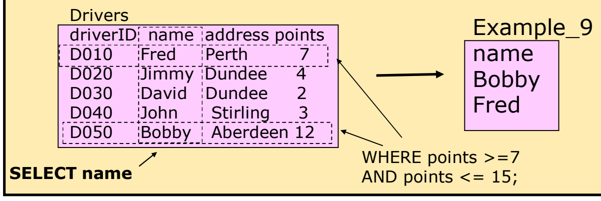
SELECT name

WHERE points >=7 AND points <= 15;

SELECT Example 9

EXAMPLE: List in alphabetical order the names of drivers whose licence points are between 7 and 15 from the Drivers table.

```
SELECT name
FROM Drivers
WHERE points >= 7 AND points <= 15
ORDER BY name;
```



Drivers

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

Example_9

name
Bobby
Fred

SELECT name

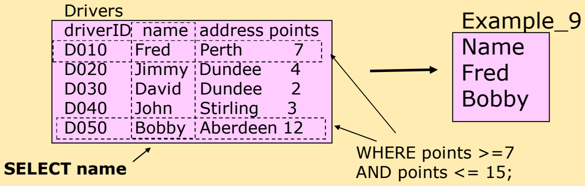
WHERE points >=7 AND points <= 15;

SELECT Example 10

EXAMPLE: List in reverse alphabetical order the names of drivers whose licence points are between 7 and 15 from the Drivers table.

```
SELECT name
FROM Drivers
WHERE points >= 7 AND points <= 15
ORDER BY name DESC;
```

NOTE: DESC means descending



Drivers

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

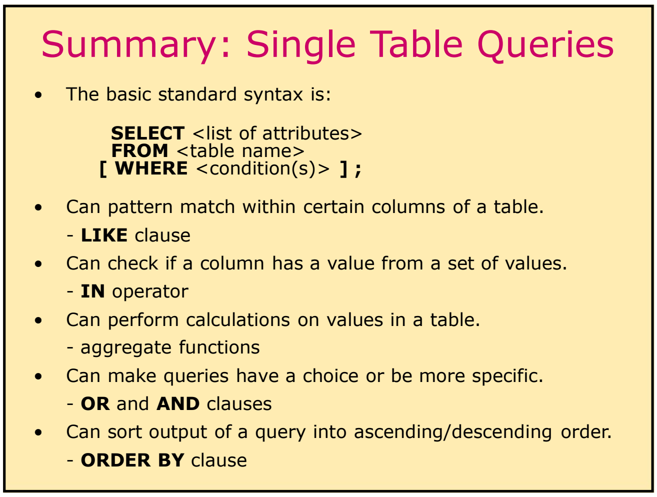
Example_9

Name
Fred
Bobby

SELECT name

WHERE points >=7 AND points <= 15;

Dr C.H. Bryant, School of SEE, Salford, UK.



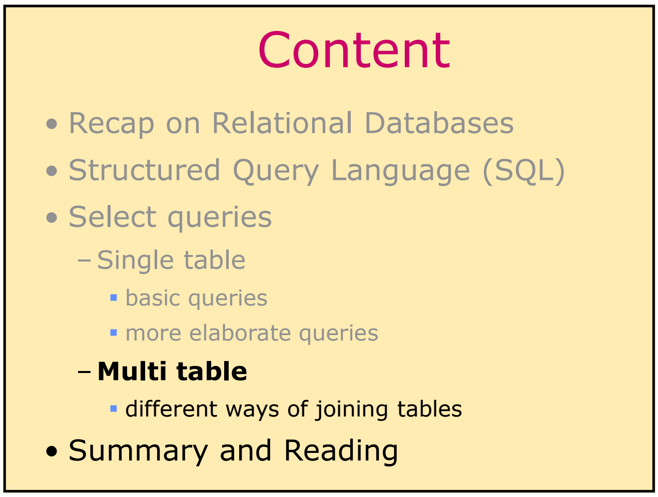
Summary: Single Table Queries

- The basic standard syntax is:

```
SELECT <list of attributes>
FROM <table name>
[ WHERE <condition(s)> ] ;
```

- Can pattern match within certain columns of a table.
  - **LIKE** clause
- Can check if a column has a value from a set of values.
  - **IN** operator
- Can perform calculations on values in a table.
  - aggregate functions
- Can make queries have a choice or be more specific.
  - **OR** and **AND** clauses
- Can sort output of a query into ascending/descending order.
  - **ORDER BY** clause

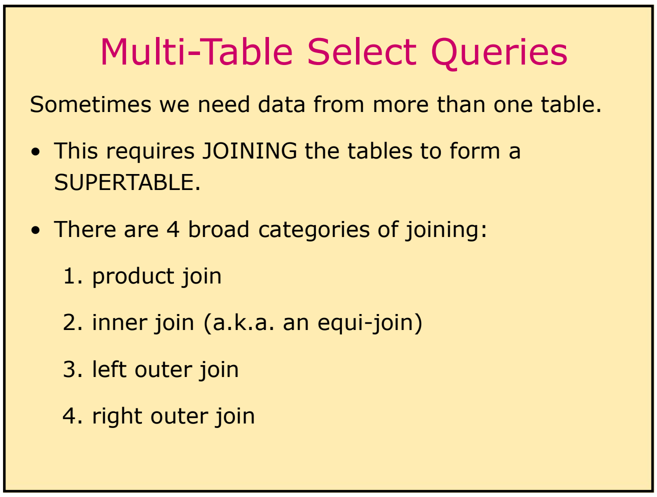
25



Content

- Recap on Relational Databases
- Structured Query Language (SQL)
- Select queries
  - Single table
    - basic queries
    - more elaborate queries
  - **Multi table**
    - different ways of joining tables
- Summary and Reading

26

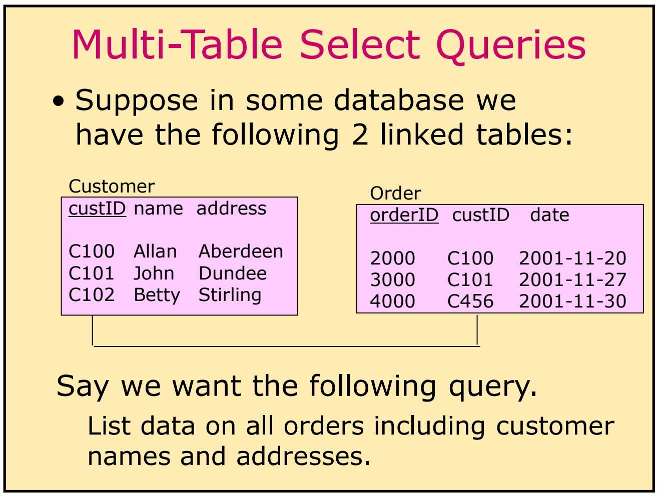


Multi-Table Select Queries

Sometimes we need data from more than one table.

- This requires JOINING the tables to form a SUPERTABLE.
- There are 4 broad categories of joining:
  1. product join
  2. inner join (a.k.a. an equi-join)
  3. left outer join
  4. right outer join

27



Multi-Table Select Queries

- Suppose in some database we have the following 2 linked tables:

Customer

custID	name	address
C100	Allan	Aberdeen
C101	John	Dundee
C102	Betty	Stirling

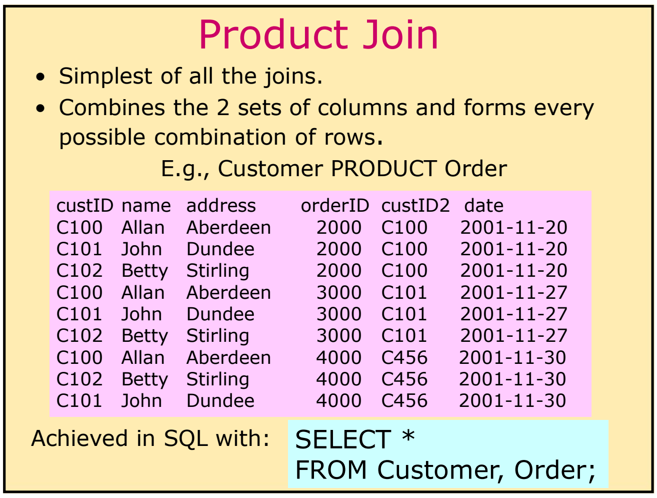
Order

orderID	custID	date
2000	C100	2001-11-20
3000	C101	2001-11-27
4000	C456	2001-11-30

Say we want the following query.

List data on all orders including customer names and addresses.

28



Product Join

- Simplest of all the joins.
- Combines the 2 sets of columns and forms every possible combination of rows.

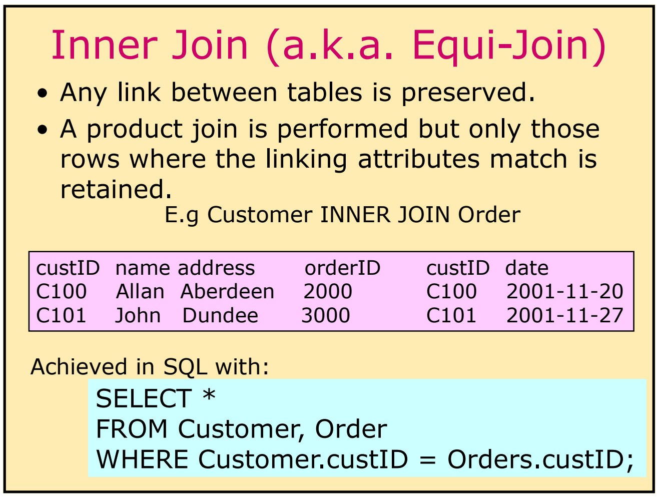
E.g., Customer PRODUCT Order

custID	name	address	orderID	custID2	date
C100	Allan	Aberdeen	2000	C100	2001-11-20
C101	John	Dundee	2000	C100	2001-11-20
C102	Betty	Stirling	2000	C100	2001-11-20
C100	Allan	Aberdeen	3000	C101	2001-11-27
C101	John	Dundee	3000	C101	2001-11-27
C102	Betty	Stirling	3000	C101	2001-11-27
C100	Allan	Aberdeen	4000	C456	2001-11-30
C102	Betty	Stirling	4000	C456	2001-11-30
C101	John	Dundee	4000	C456	2001-11-30

Achieved in SQL with:

```
SELECT *
FROM Customer, Order;
```

29



Inner Join (a.k.a. Equi-Join)

- Any link between tables is preserved.
- A product join is performed but only those rows where the linking attributes match is retained.

E.g Customer INNER JOIN Order

custID	name	address	orderID	custID	date
C100	Allan	Aberdeen	2000	C100	2001-11-20
C101	John	Dundee	3000	C101	2001-11-27

Achieved in SQL with:

```
SELECT *
FROM Customer, Order
WHERE Customer.custID = Orders.custID;
```

30

Inner Join *(continued)*

- More useful kind of join for data retrieval.
- Contains all the orders and full details of all the associated customers, bar one,
- i.e., orderID = 4000 which has an unmatched customer.
  - Break in database integrity.

31

Left Outer Join

This is like an Inner Join but any rows in the 1st table that do not have a match in the 2nd table are still included in the output, but matched to NULL.

E.g., Customer LEFT OUTER JOIN Order

custID	name	address	orderID	date
C100	Allan	Aberdeen	2000	2001-20-11
C101	John	Dundee	3000	2001-11-27
C102	Betty	Stirling	NULL	NULL

```
SELECT *
FROM Customer, Order
WHERE Customer.custID = Orders.custID(+);
```

i.e., may be NULL

32

Right Outer Join

This is like an Inner Join but any rows in the 2nd table that do not have a match in the 1st table are still included in the output, but matched to NULL.

E.g., Customer RIGHT OUTER JOIN Order

name	address	orderID	custID	date
Allan	Aberdeen	2000	C100	2001-11-20
John	Dundee	3000	C101	2001-11-27
NULL	NULL	4000	C456	2001-11-30

```
SELECT *
FROM Customer, Order
WHERE Customer.custID(+) = Orders.custID;
```

33

Rest of query works on supertable

- Performing a multi-table queries requires joining the tables to form a supertable.
  - It is from this supertable that data is extracted.
- The type of join will determine the contents of the supertable and hence what data can be extracted.

```
SELECT name, orderID
FROM Customer, Order
WHERE Customer.custID(+) = Order.custID
AND date > 25/11/01;
```

name	address	orderID	custID	date
Allan	Aberdeen	2000	C100	2001-11-20
John	Dundee	3000	C101	2001-11-27
NULL	NULL	4000	C456	2001-11-30

→

name	orderID
John	3000
NULL	4000

34

Summary: Multi Table Queries

- Sometimes we need data from more than one table.
- Requires joining the tables to form a supertable.
- 4 broad categories of joining:
  1. product join
  2. inner join (a.k.a. an equi-join)
  3. left outer join
  4. right outer join

35

Further Reading

Sections 6.3.1 and 6.3.2. of (Connolly & Begg, 2014)

or

pages 41-53 of (Connolly & Begg, 2004)

or

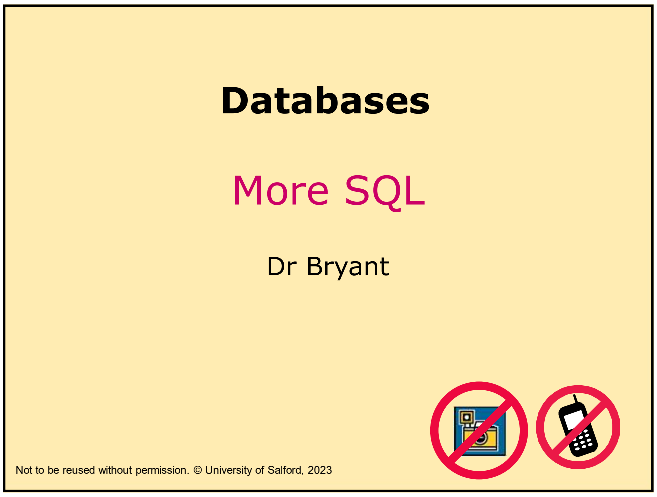
Chapter 2 and Section 5.1 of (Donahoo & Speegle, 2005)

or

Section 3.3.1 and 3.3.2 of (Silberschatz et al., 2019).

The references are on the last page of the exercise booklet.

36

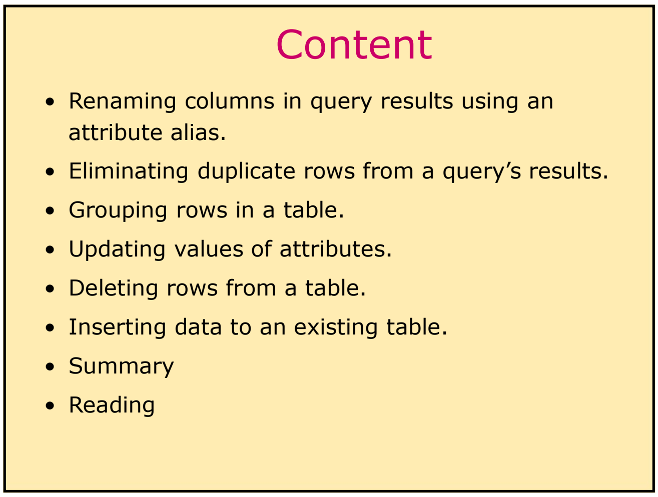


Databases

More SQL

Dr Bryant

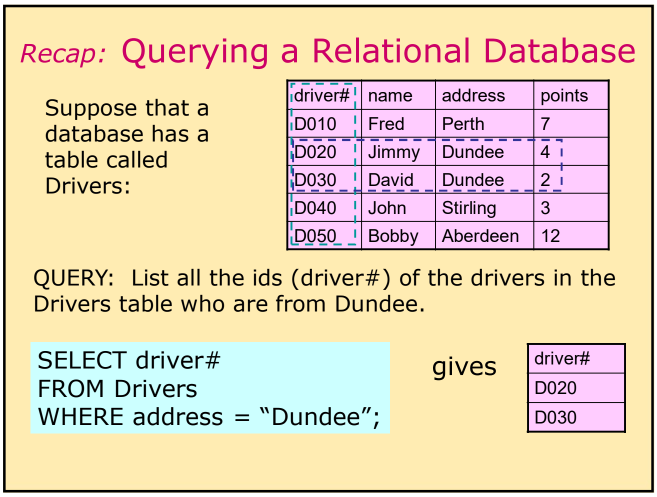
1



Content

- Renaming columns in query results using an attribute alias.
- Eliminating duplicate rows from a query's results.
- Grouping rows in a table.
- Updating values of attributes.
- Deleting rows from a table.
- Inserting data to an existing table.
- Summary
- Reading

2



Recap: Querying a Relational Database

Suppose that a database has a table called Drivers:

driver#	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

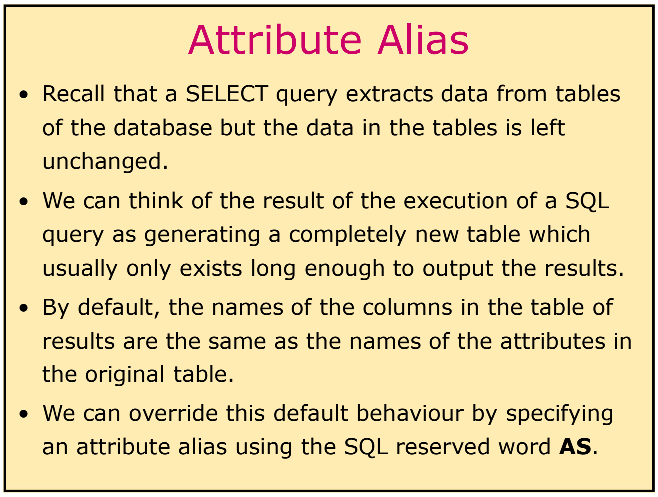
QUERY: List all the ids (driver#) of the drivers in the Drivers table who are from Dundee.

```
SELECT driver#
FROM Drivers
WHERE address = "Dundee";
```

gives

driver#
D020
D030

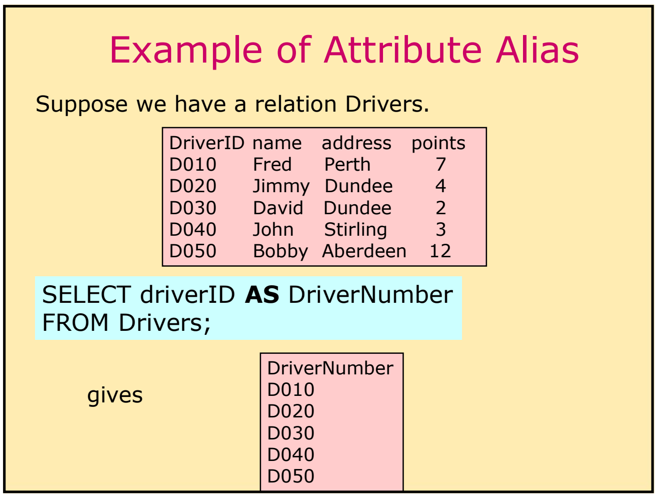
3



Attribute Alias

- Recall that a SELECT query extracts data from tables of the database but the data in the tables is left unchanged.
- We can think of the result of the execution of a SQL query as generating a completely new table which usually only exists long enough to output the results.
- By default, the names of the columns in the table of results are the same as the names of the attributes in the original table.
- We can override this default behaviour by specifying an attribute alias using the SQL reserved word **AS**.

4



Example of Attribute Alias

Suppose we have a relation Drivers.

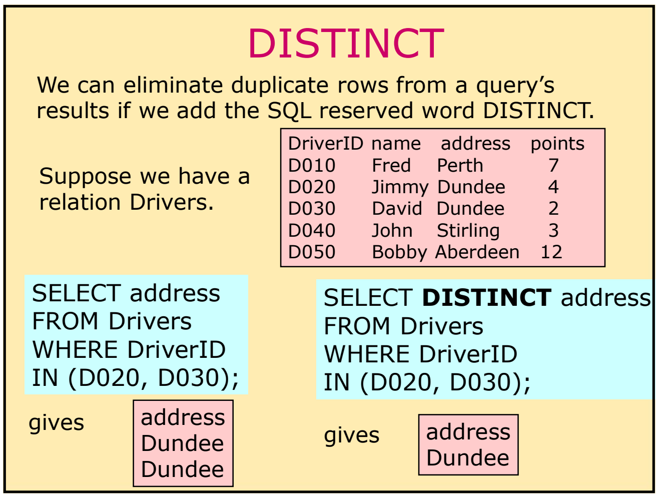
DriverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

```
SELECT driverID AS DriverNumber
FROM Drivers;
```

gives

DriverNumber
D010
D020
D030
D040
D050

5



DISTINCT

We can eliminate duplicate rows from a query's results if we add the SQL reserved word DISTINCT.

Suppose we have a relation Drivers.

DriverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

```
SELECT address
FROM Drivers
WHERE DriverID
IN (D020, D030);
```

gives

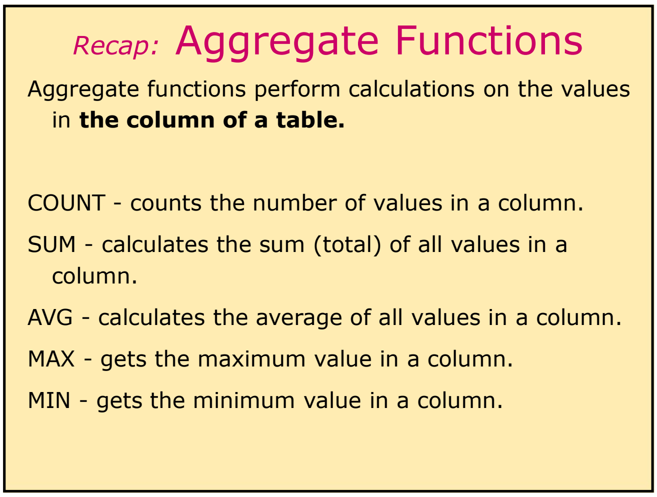
address
Dundee
Dundee

```
SELECT DISTINCT address
FROM Drivers
WHERE DriverID
IN (D020, D030);
```

gives

address
Dundee

6



Recap: Aggregate Functions

Aggregate functions perform calculations on the values in **the column of a table.**

COUNT - counts the number of values in a column.

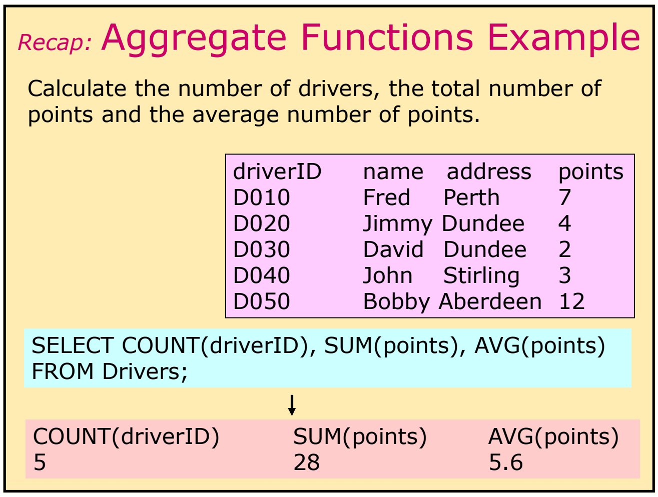
SUM - calculates the sum (total) of all values in a column.

AVG - calculates the average of all values in a column.

MAX - gets the maximum value in a column.

MIN - gets the minimum value in a column.

7



Recap: Aggregate Functions Example

Calculate the number of drivers, the total number of points and the average number of points.

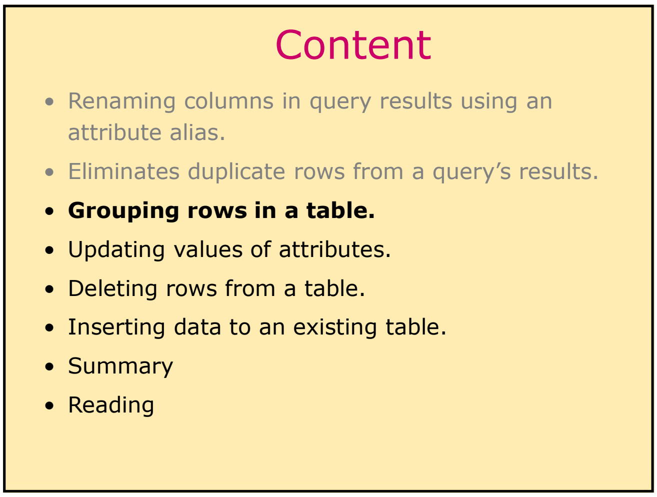
driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

```
SELECT COUNT(driverID), SUM(points), AVG(points)
FROM Drivers;
```

↓

COUNT(driverID)	SUM(points)	AVG(points)
5	28	5.6

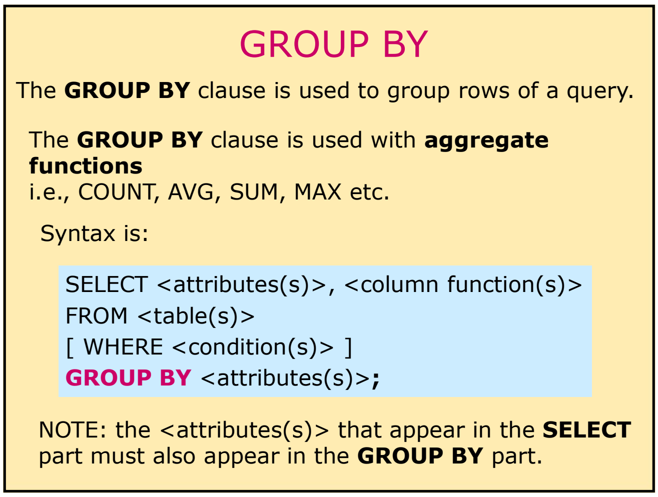
8



Content

- Renaming columns in query results using an attribute alias.
- Eliminates duplicate rows from a query's results.
- **Grouping rows in a table.**
- Updating values of attributes.
- Deleting rows from a table.
- Inserting data to an existing table.
- Summary
- Reading

9



GROUP BY

The **GROUP BY** clause is used to group rows of a query.

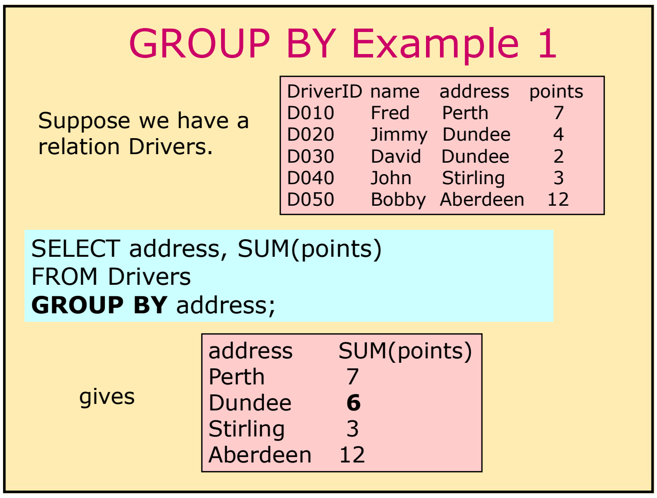
The **GROUP BY** clause is used with **aggregate functions**
i.e., COUNT, AVG, SUM, MAX etc.

Syntax is:

```
SELECT <attributes(s)>, <column function(s)>
FROM <table(s)>
[ WHERE <condition(s)> ]
GROUP BY <attributes(s)>;
```

NOTE: the <attributes(s)> that appear in the **SELECT** part must also appear in the **GROUP BY** part.

10



GROUP BY Example 1

Suppose we have a relation Drivers.

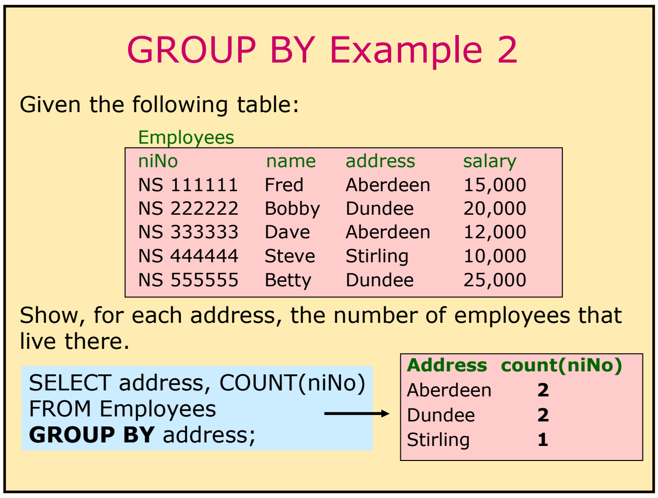
DriverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

```
SELECT address, SUM(points)
FROM Drivers
GROUP BY address;
```

gives

address	SUM(points)
Perth	7
Dundee	**6**
Stirling	3
Aberdeen	12

11



GROUP BY Example 2

Given the following table:

Employees

niNo	name	address	salary
NS 111111	Fred	Aberdeen	15,000
NS 222222	Bobby	Dundee	20,000
NS 333333	Dave	Aberdeen	12,000
NS 444444	Steve	Stirling	10,000
NS 555555	Betty	Dundee	25,000

Show, for each address, the number of employees that live there.

```
SELECT address, COUNT(niNo)
FROM Employees
GROUP BY address;
```

→

Address	**count(niNo)**
Aberdeen	**2**
Dundee	**2**
Stirling	**1**

12

GROUP BY Example 2 *(continued)*

In order to do the previous query, an intermediate table is created that is grouped by (sorted on) address.

Employees

niNo	name	address	salary	
NS 111111	Fred	Aberdeen	15,000	grouping 1
NS 333333	Dave	Aberdeen	12,000	
NS 222222	Bobby	Dundee	20,000	grouping 2
NS 555555	Betty	Dundee	25,000	
NS 444444	Steve	Stirling	10,000	grouping 3

count(niNo) is applied to each of the 3 groupings.

13

HAVING Clause

The **HAVING** clause is used with the **GROUP BY** clause to filter groups of rows.

Syntax is:

```
SELECT <attributes(s)>, <column function(s)>
FROM <table(s)>
[ WHERE <condition(s)> ]
GROUP BY <attributes(s)>
HAVING  <condition(s)>;
```

Groups for which the HAVING condition does not evaluate to true are not included in the output.

14

HAVING Example

Given the following table:

Employees

niNo	name	address	salary
NS 111111	Fred	Aberdeen	15,000
NS 222222	Bobby	Dundee	20,000
NS 333333	Dave	Aberdeen	12,000
NS 444444	Steve	Stirling	10,000
NS 555555	Betty	Dundee	25,000

Show the addresses whose employees' average salary is greater than £21,000. Also show the average salary.

```
SELECT address, AVG(salary)
FROM Employees
GROUP BY address
HAVING AVG(salary) > 21000;
```

→

address	avg(salary)
Dundee	22500

NOTE: compare with intermediate table from 2 slides ago

15

Comparison of HAVING and WHERE

- They serve different purposes.
- The WHERE clause removes rows **before** grouping.
- The HAVING clause filters groups.
- Aggregate functions cannot be used in the WHERE clause.

16

Content

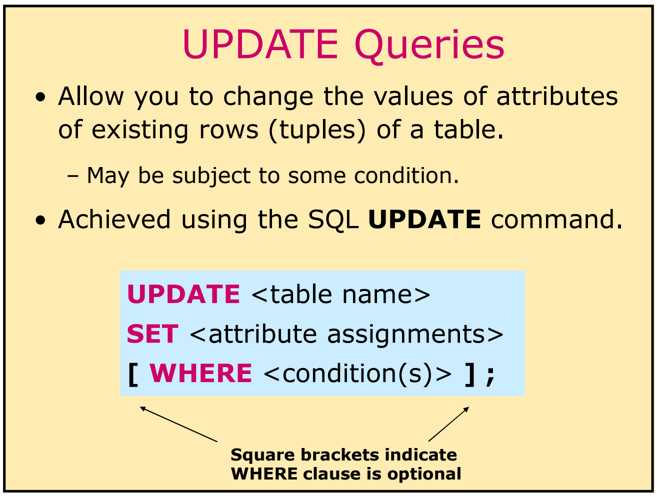
- Renaming columns in query results using an attribute alias.
- Eliminates duplicate rows from a query's results.
- Grouping rows in a table.
- **Updating values of attributes.**
- Deleting rows from a table.
- Inserting data to an existing table.
- Summary
- Reading

17

Changing Data in Tables

- The **SELECT** query extracts data from tables,
  - the tables remain unchanged.
- We can have queries which change the data in the tables:
  - **UPDATE** queries,
  - **DELETE** queries,
  - **INSERT** queries.

18



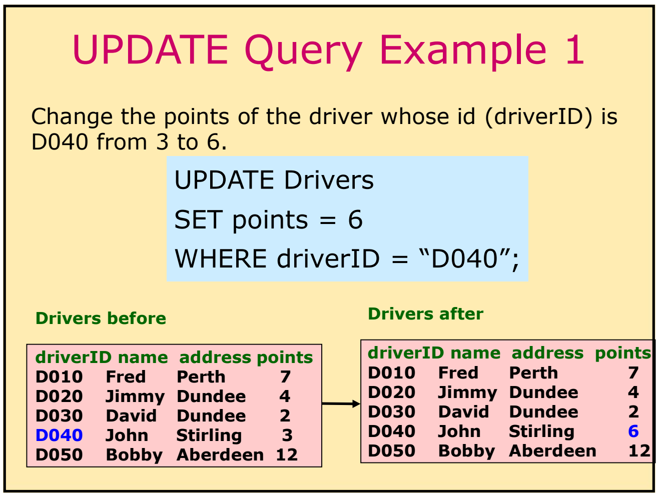
UPDATE Queries

- Allow you to change the values of attributes of existing rows (tuples) of a table.
  - May be subject to some condition.
- Achieved using the SQL **UPDATE** command.

```
UPDATE <table name>
SET <attribute assignments>
[ WHERE <condition(s)> ] ;
```

Square brackets indicate WHERE clause is optional

19



UPDATE Query Example 1

Change the points of the driver whose id (driverID) is D040 from 3 to 6.

```
UPDATE Drivers
SET points = 6
WHERE driverID = “D040”;
```

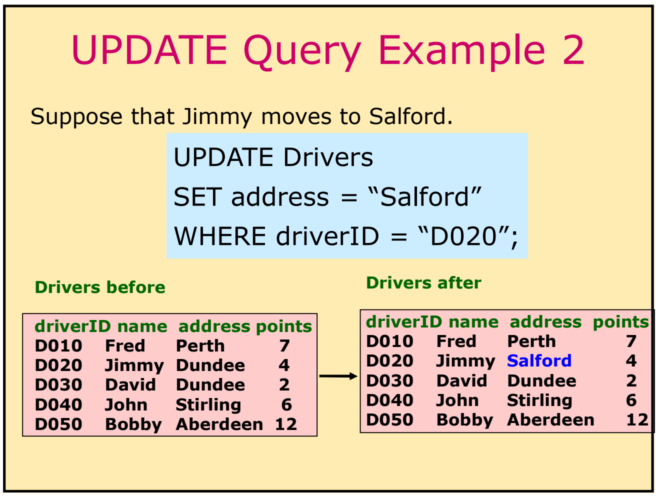
Drivers before

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

Drivers after

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	6
D050	Bobby	Aberdeen	12

20



UPDATE Query Example 2

Suppose that Jimmy moves to Salford.

```
UPDATE Drivers
SET address = “Salford”
WHERE driverID = “D020”;
```

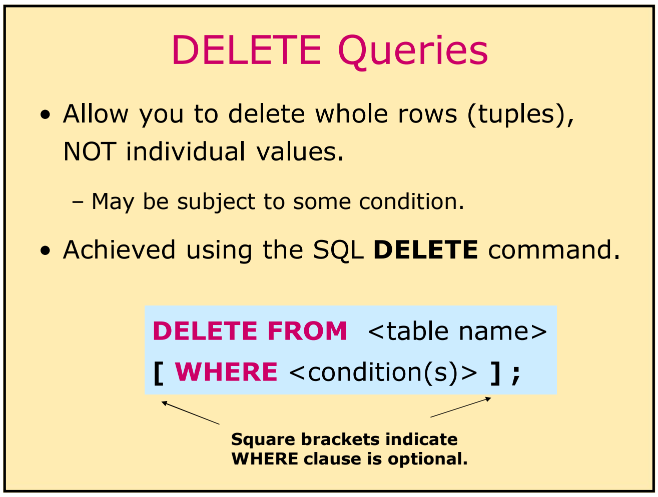
Drivers before

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	6
D050	Bobby	Aberdeen	12

Drivers after

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Salford	4
D030	David	Dundee	2
D040	John	Stirling	6
D050	Bobby	Aberdeen	12

21



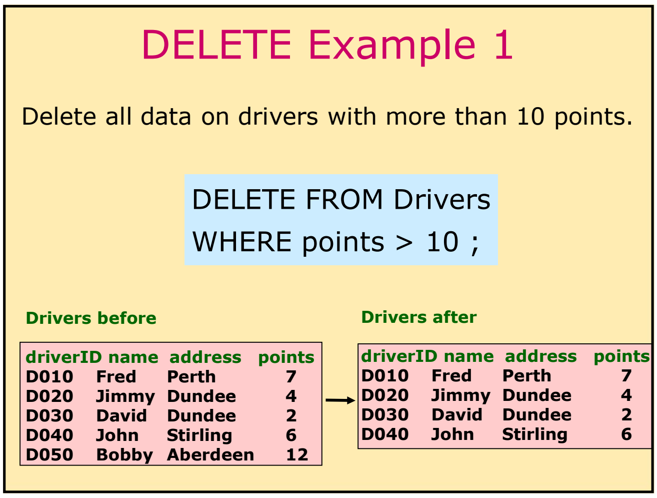
DELETE Queries

- Allow you to delete whole rows (tuples), NOT individual values.
  - May be subject to some condition.
- Achieved using the SQL **DELETE** command.

```
DELETE FROM  <table name>
[ WHERE <condition(s)> ] ;
```

Square brackets indicate WHERE clause is optional.

22



DELETE Example 1

Delete all data on drivers with more than 10 points.

```
DELETE FROM Drivers
WHERE points > 10 ;
```

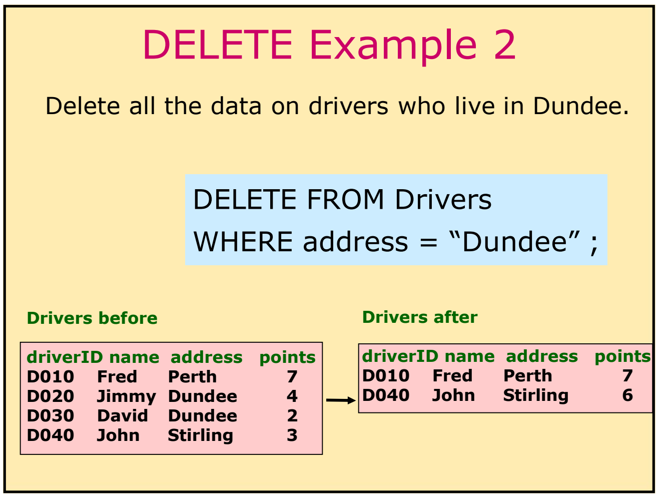
Drivers before

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	6
D050	Bobby	Aberdeen	12

Drivers after

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	6

23



DELETE Example 2

Delete all the data on drivers who live in Dundee.

```
DELETE FROM Drivers
WHERE address = “Dundee” ;
```

Drivers before

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3

Drivers after

driverID	name	address	points
D010	Fred	Perth	7
D040	John	Stirling	6

24

INSERT Queries

An **INSERT** query will add data to an existing table without deleting it or any of its records.

INSERT INTO <table name> [**(**<attribute list>**)**]

<SELECT statement> | **VALUES (**<value list>**);**

<attribute list> is optional (used for partial information)

The | symbol indicates that you must either use the SELECT statement or the VALUES clause but not both at the same time.

25

INSERT - Example 1

Add a new driver into the Driver table whose id (driverID) is *D060*, name is *Betty*, address is *Inverness* and points is *6.*

INSERT INTO Drivers
VALUES (“D060”, “Betty”, “Inverness”, 6);

Drivers

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	6

→

Drivers

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	6
D060	Betty	Inverness	6

26

INSERT - Example 2

Add a new driver into the Driver table whose id (driverID) is *D070*, name is *Jeannie*, and no address and points information is given.

INSERT INTO Drivers **(driverID, name)**
VALUES **(“D070”, “Jeannie”);**

Drivers

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	6
D060	Betty	Inverness	6

→

Drivers

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	6
D060	Betty	Inverness	6
D070	Jeannie		

27

Summary

- Rename a column in a query’s results using **AS**.
- **DISTINCT** eliminates duplicate rows from a query’s results.
- Query on groups of rows of a table using the **GROUP BY** and **HAVING** clauses.
- Change the data in the tables using:
  - **UPDATE**
  - **DELETE** (whole rows)
  - **INSERT** (whole or part of rows)

28

Further Reading

Sections 6.3.4 and 6.3.10 of (Connolly & Begg, 2014)

or

Sections 3.2.5 and 3.2.8 of (Connolly & Begg, 2004)

or

Chapter 4 of (Donahoo & Speegle, 2005)

or

Sections 3.7 and 3.9 of (Silberschatz et al., 2019)

The list of references is on the final page of the exercise booklet.

29

Databases

Data Description Language: Creating, Altering and Destroying Tables Using SQL

Dr Bryant

1

Introduction

As well as querying and changing data, there are other things we can do in SQL, such as:

- Create relations (tables) using the **CREATE TABLE** statement.
- Modify relations using the **ALTER TABLE** statement.
- Destroy relations using the **DROP TABLE** statement.

2

Content

- SQL Data Types
- A special value - NULL
- Creating a table
  - default values
  - constraints
- Adding columns to tables
- Destroying tables
- Summary and Reading

3

SQL Data Types for Character Strings

The most common SQL data types are:

- CHARACTER(L) or CHAR(L)
  - A fixed-length character string containing exactly L characters.
  - If the string contains fewer characters, then the remaining characters contain padding characters.
  - The padding characters are usually spaces.
- CHARACTER VARYING(L) or VARCHAR(L)
  - A variable-length character string that may hold up to L characters.
  - Only the specified number of characters are stored, so there is never any padding.

4

SQL Data Types for Numeric Data

The most common SQL numeric data types are:

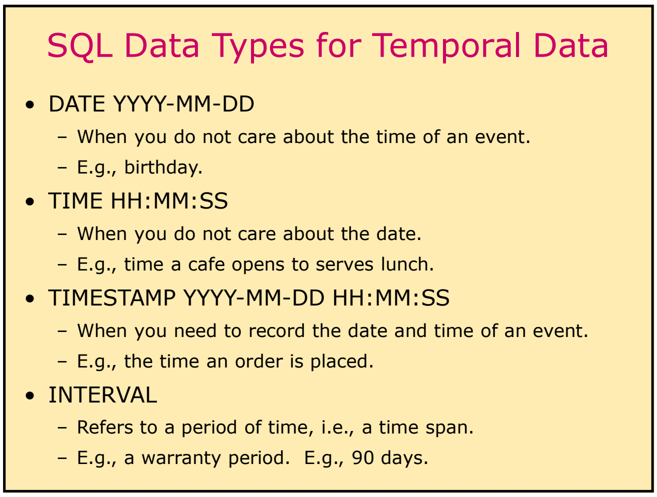
- INTEGER or INT
  - A signed whole number.
  - The range of possible values is DBMS dependent.
- NUMERIC(P,S)
  - A signed, fixed-point number.
  - P (precision) specifies the total number of digits in the number.
  - S (scale) specifies the number of digits to the right of the decimal place.
  - E.g., NUMERIC(5,2) specifies a type ranging from -999.99 to 999.99.

5

SQL Data Type Boolean

- A type which can only have one of three values: true, false and unknown.
- The name Boolean commemorates George Boole (1815-1864) who first placed the study of logic on a sound mathematical basis.
- We use logic to reason about truth.
- Sometimes we just want to record (in a database) whether something is true or not.

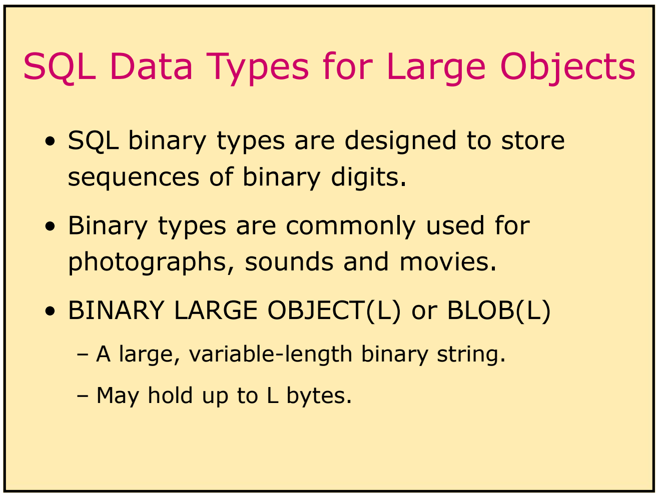
6



SQL Data Types for Temporal Data

- DATE YYYY-MM-DD
  - When you do not care about the time of an event.
  - E.g., birthday.
- TIME HH:MM:SS
  - When you do not care about the date.
  - E.g., time a cafe opens to serves lunch.
- TIMESTAMP YYYY-MM-DD HH:MM:SS
  - When you need to record the date and time of an event.
  - E.g., the time an order is placed.
- INTERVAL
  - Refers to a period of time, i.e., a time span.
  - E.g., a warranty period. E.g., 90 days.

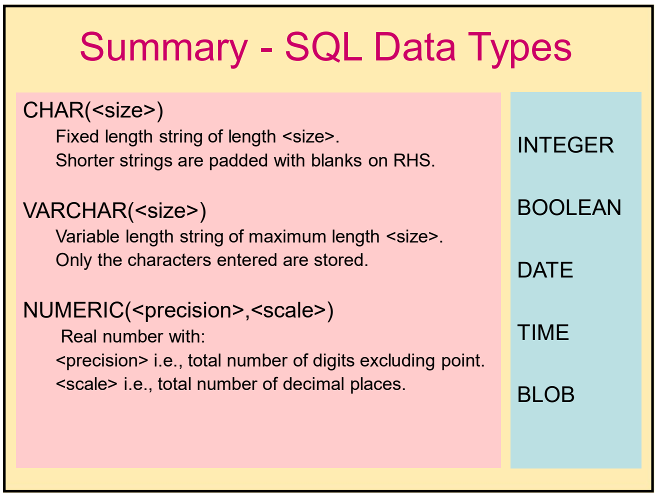
7



SQL Data Types for Large Objects

- SQL binary types are designed to store sequences of binary digits.
- Binary types are commonly used for photographs, sounds and movies.
- BINARY LARGE OBJECT(L) or BLOB(L)
  - A large, variable-length binary string.
  - May hold up to L bytes.

8



Summary - SQL Data Types

CHAR(<size>)
Fixed length string of length <size>.
Shorter strings are padded with blanks on RHS.

VARCHAR(<size>)
Variable length string of maximum length <size>.
Only the characters entered are stored.

NUMERIC(<precision>,<scale>)
Real number with:
<precision> i.e., total number of digits excluding point.
<scale> i.e., total number of decimal places.

INTEGER

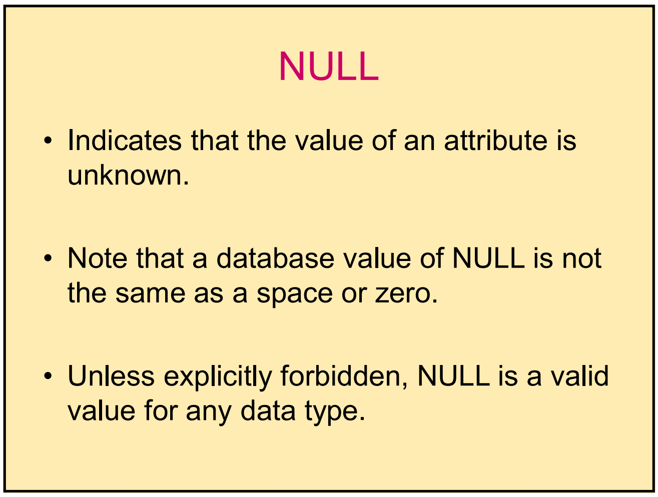
BOOLEAN

DATE

TIME

BLOB

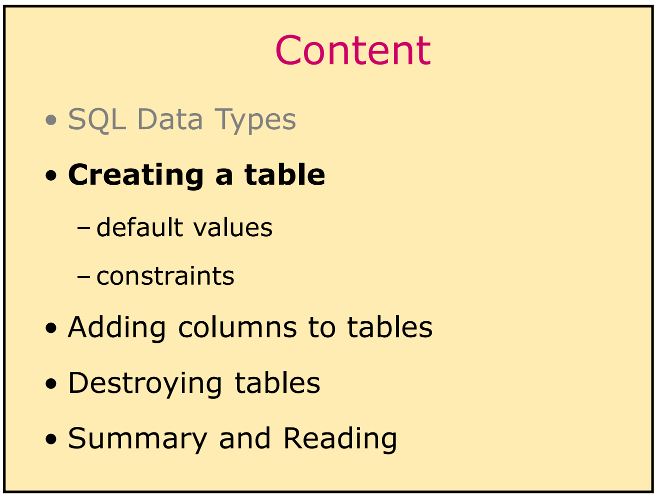
9



NULL

- Indicates that the value of an attribute is unknown.
- Note that a database value of NULL is not the same as a space or zero.
- Unless explicitly forbidden, NULL is a valid value for any data type.

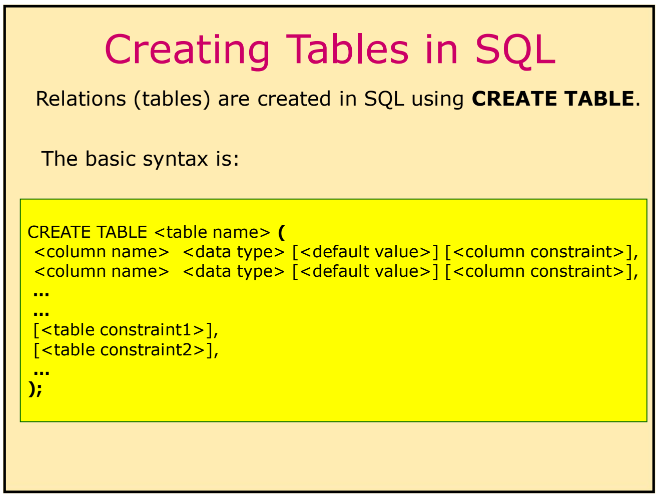
10



Content

- SQL Data Types
- **Creating a table**
  - default values
  - constraints
- Adding columns to tables
- Destroying tables
- Summary and Reading

11



Creating Tables in SQL

Relations (tables) are created in SQL using **CREATE TABLE**.

The basic syntax is:

```
CREATE TABLE <table name> (
 <column name>  <data type> [<default value>] [<column constraint>],
 <column name>  <data type> [<default value>] [<column constraint>],
 …
 …
 [<table constraint1>],
 [<table constraint2>],
 …
);
```

12

Example of Creating Tables

Create tables for the following relations:

Students(studentID, title, name, city)

Courses(courseID, courseName, startDate, endDate)

```
CREATE TABLE students (
   studentID   INTEGER,
   title       VARCHAR(4),
   name        VARCHAR(20),
   city        VARCHAR(20)
   );
```

```
CREATE TABLE courses (
   courseID     CHAR(4),
   courseName   VARCHAR(20),
   startDate    DATE,
   endDate      DATE
   );
```

Would give the following relations:

Table name: students
Attributes: studentID, title, name, city

Table name: courses
Attributes: courseID, courseName, startDate, endDate

13

Default Values

- When a new row is created using INSERT, any columns without a specified value are assigned the default value.
- Unless otherwise specified, the default value is NULL.
- We can specify a default value for a column by adding DEFAULT <value expression> to the create table statement.

```
CREATE TABLE students (
   studentID   INTEGER,
   title       VARCHAR(4),
   name        VARCHAR(20),
   city        VARCHAR(20) DEFAULT "Salford"
   );
```

14

Constraints

- A DBMS can do much more than just store and access data.
- It can also enforce constraints on what data are allowed in the database.
- The DBMS enforces constraints by not allowing any data which violates the constraints to be added to the database.
- Any INSERT, UPDATE or DELETE that would result in a constraint violation is rejected without changing the database.

15

Column Constraints

- There are many types of constraints.
- Today we will focus mainly on just one type.
- A column constraint applies to one particular column of a relation (table).
- Recall that the basic syntax of **CREATE TABLE** is:

```
CREATE TABLE <table name> (
 <column name>  <data type> [<default value>] [<column constraint>],
 <column name>  <data type> [<default value>] [<column constraint>],
 …
 …
 [<table constraint1>],
 [<table constraint2>],
 …
);
```

16

NOT NULL

- Prohibits NULL values for a particular column.
- E.g., suppose that a student must have a name.

```
CREATE TABLE students (
   studentID      INTEGER,
   title          VARCHAR(4),
   name           VARCHAR(20) NOT NULL,
   city           VARCHAR(20) DEFAULT "Salford"
   );
```

17

UNIQUE

- Forces distinct column values.
- E.g., every employee has a unique National Insurance (NI) number.

```
CREATE TABLE employee (
   employeeID     INTEGER,
   niNumber       CHAR(11) UNIQUE,
   name           VARCHAR(20),
   age            INTEGER
   );
```

- UNIQUE is only applied to non-NULL values.

18

Declaring a Primary Key

```
CREATE TABLE employee (
   employeeID     INTEGER PRIMARY KEY,
   niNumber       CHAR(11) UNIQUE,
   name           VARCHAR(20),
   age            INTEGER
   );
```

- Recall that a primary key uniquely identifies a tuple (row) in the table.
- No values of the primary key may be NULL, so we do not need the NOT NULL constraint for employeeID.

19

Comparison

PRIMARY KEY	UNIQUE
There is, at most, one primary key for each table.	No limit on the number of columns that are declared to be unique.
No values of the primary key may be NULL.	Allows NULL values.

20

Declaring a Foreign Key

Suppose that we want all the values of course in the students table either to reference a courseID from the courses table or to be NULL.

```
CREATE TABLE students (
   studentID   INTEGER PRIMARY KEY,
   title       VARCHAR(4),
   name        VARCHAR(20),
   city        VARCHAR(20),
   course      CHAR(4) REFERENCES courses(courseID)
   );
```

```
CREATE TABLE courses (
   courseID      CHAR(4) PRIMARY KEY,
   courseName    VARCHAR(20),
   startDate     DATE,
   endDate       DATE
   );
```

21

Naming Constraints

CONSTRAINT <constraint name> <constraint>

Why bother?

- When you attempt an INSERT, UPDATE or DELETE that violates a constraint, SQL rejects the operation and issues an error message. Many DBMSs include the name of the violated constraint in the error message.
- We can delete constraints by name.

22

Naming Constraints

Consider again the example of students and courses.

This slide shows the constrains that we have already studied, but with names added.

```
CREATE TABLE students (
  studID INTEGER CONSTRAINT student_pk PRIMARY KEY,
  title VARCHAR(4),
  name VARCHAR(20) CONSTRAINT name_not_null NOT NULL,
  city VARCHAR(20) CONSTRAINT city_default DEFAULT “Salford”,
  course CHAR(4) CONSTRAINT student_fk REFERENCES courses(couresID)
);
```

```
CREATE TABLE courses (
  coursesID CHAR(4)  CONSTRAINT course_pk PRIMARY KEY,
  courseName VARCHAR(20),
  startDate DATE,
  endDate DATE
);
```

23

Table Constraints

- So far we have focused on column constraints.
- However, we need to study table constraints *briefly* now.
- Otherwise you will not be able to implement composite primary keys.
- We study composite keys later in this module.

24

Creating Table Constraints in SQL

Recall that the basic syntax of **CREATE TABLE** is

```
CREATE TABLE <table name> (
 <column name>  <data type> [<default value>] [<column constraint>],
 <column name>  <data type> [<default value>] [<column constraint>],
 …
 …
 [<table constraint1>],
 [<table constraint2>],
 …
);
```

25

Table Constraints and Composite Primary Keys

- An example of a composite primary key is:
  supply(<u>partID, supplierID</u>)
- How can you create a composite primary key in SQL?

```
CREATE TABLE supply (
  partID INTEGER,
  supplierID INTEGER,
  PRIMARY KEY (partID , supplierID)
);
```

You cannot create a composite primary key by simply adding the PRIMARY KEY constraint to more than one column in the **table** because SQL will think you are trying to create multiple primary keys, which is not allowed.

26

Exercise

Write SQL statements that create tables for the following relations.

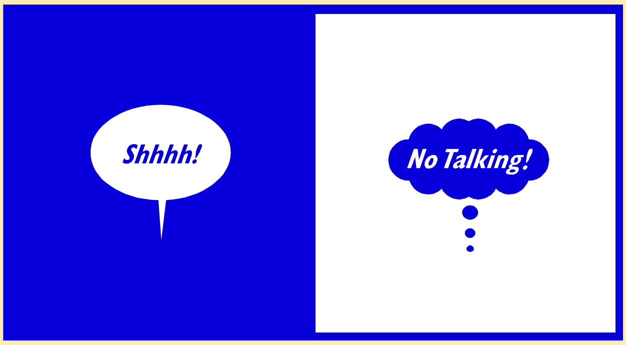
- team(<u>teamID</u>, name)
- member(<u>memberID</u>, niNumber, address, **teamID**)

Your SQL statements should impose the following constraints.

- Every team must have a name.
- The attribute teamID in the member relation is a foreign key to teamID in the team relation.
- The default value of a member's address is Salford.
- National Insurance numbers are unique.

27

Solution to Exercise



28

Content

- SQL Data Types
- Creating a table from scratch
  - default values
  - constraints
- **Adding columns to tables**
- Destroying tables
- Summary and Reading

29

Altering Tables in SQL

Tables can be modified in SQL using ALTER TABLE.

The syntax is:

```
ALTER TABLE <table name>
ADD <column name> <data type>;
```

EXAMPLE: Modify students table to have a course column.

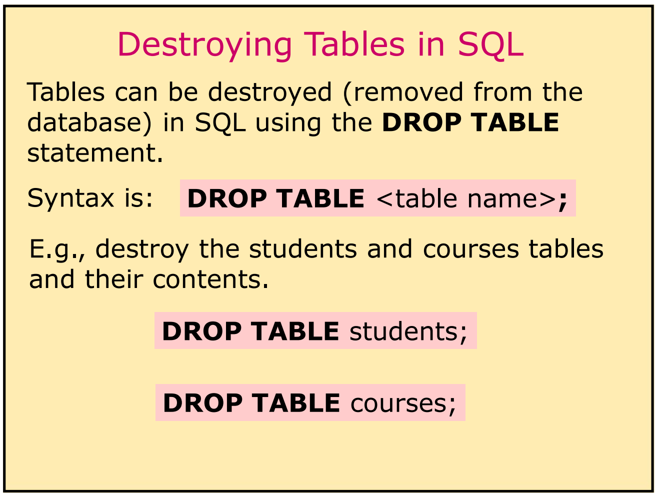
```
ALTER TABLE students
ADD course CHAR(4);
```

Table: students
Attributes: studentID, title, name, city

↓

Table: students
Attributes: studentID, title, name, city, **course**

30



Destroying Tables in SQL

Tables can be destroyed (removed from the database) in SQL using the **DROP TABLE** statement.

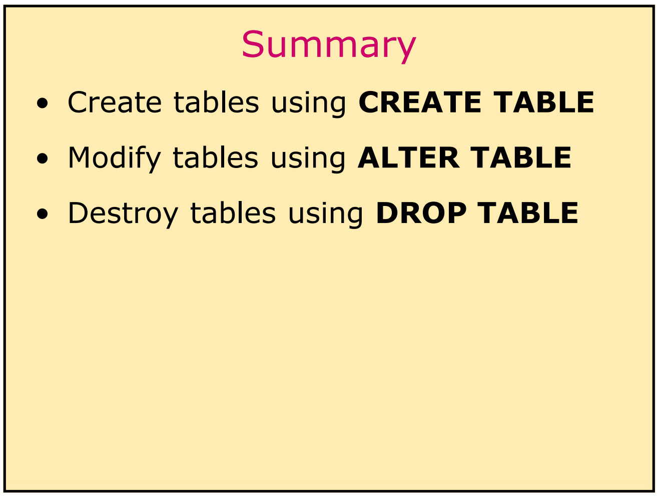
Syntax is: **DROP TABLE** <table name>**;**

E.g., destroy the students and courses tables and their contents.

DROP TABLE students;

DROP TABLE courses;

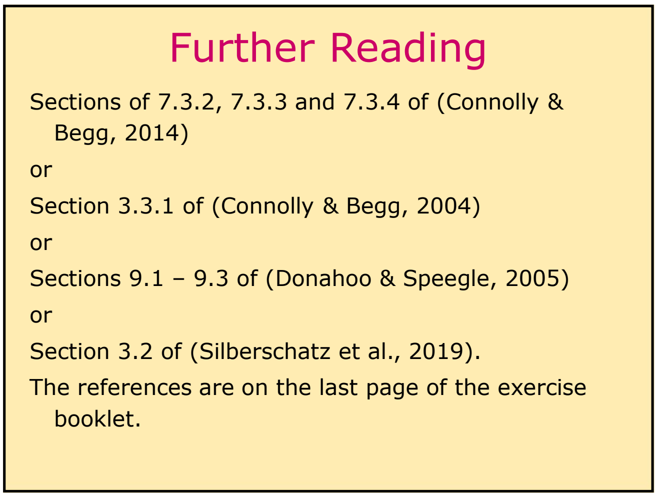
31



Summary

- Create tables using **CREATE TABLE**
- Modify tables using **ALTER TABLE**
- Destroy tables using **DROP TABLE**

32



Further Reading

Sections of 7.3.2, 7.3.3 and 7.3.4 of (Connolly & Begg, 2014)

or

Section 3.3.1 of (Connolly & Begg, 2004)

or

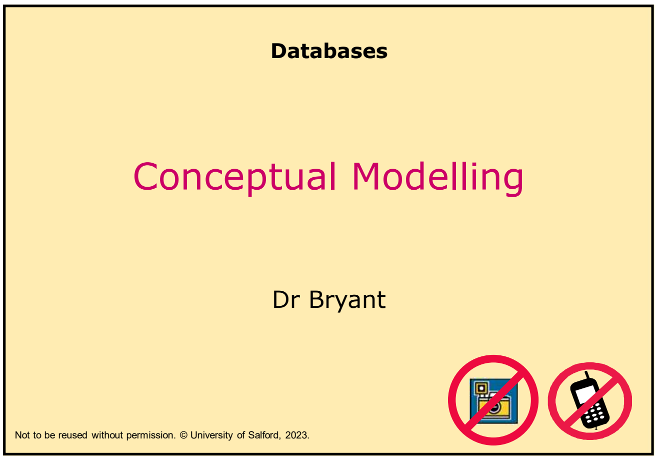
Sections 9.1 – 9.3 of (Donahoo & Speegle, 2005)

or

Section 3.2 of (Silberschatz et al., 2019).

The references are on the last page of the exercise booklet.

33

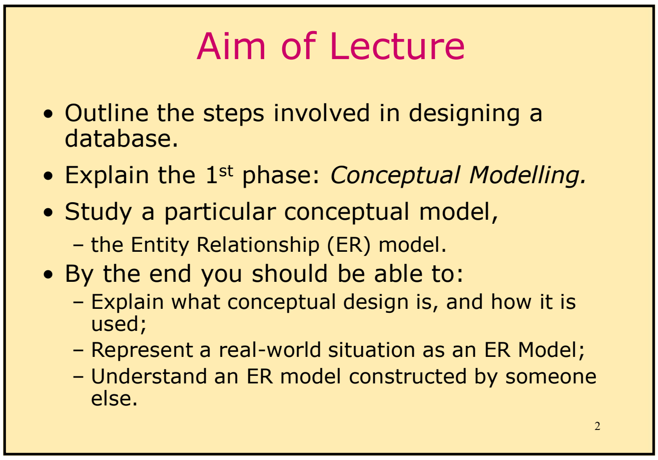


Databases

Conceptual Modelling

Dr Bryant

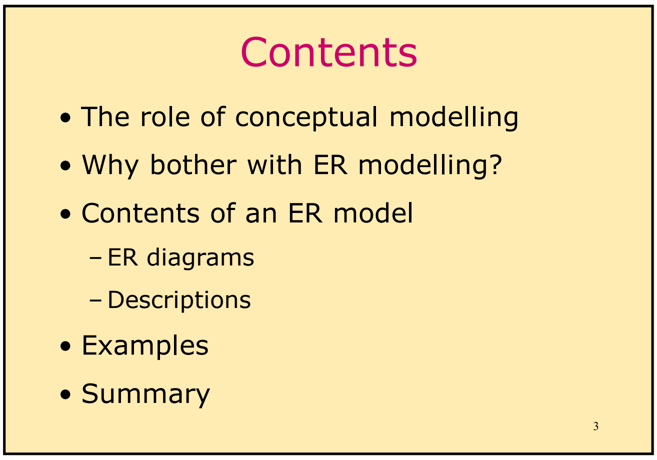
1



Aim of Lecture

- Outline the steps involved in designing a database.
- Explain the 1st phase: *Conceptual Modelling.*
- Study a particular conceptual model,
  - the Entity Relationship (ER) model.
- By the end you should be able to:
  - Explain what conceptual design is, and how it is used;
  - Represent a real-world situation as an ER Model;
  - Understand an ER model constructed by someone else.

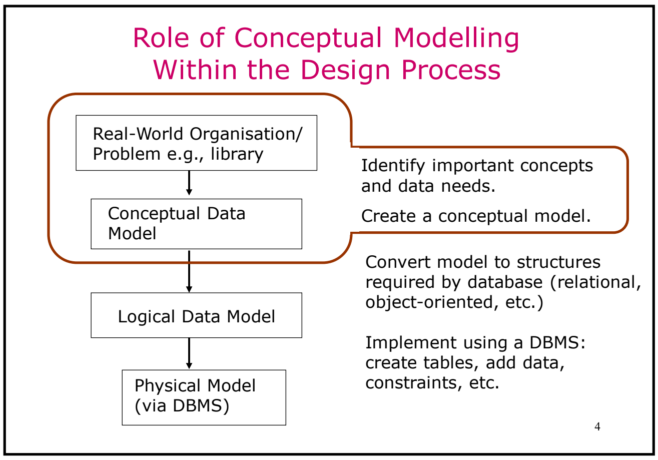
2



Contents

- The role of conceptual modelling
- Why bother with ER modelling?
- Contents of an ER model
  - ER diagrams
  - Descriptions
- Examples
- Summary

3



Role of Conceptual Modelling Within the Design Process

Real-World Organisation/ Problem e.g., library
Conceptual Data Model
Logical Data Model
Physical Model (via DBMS)

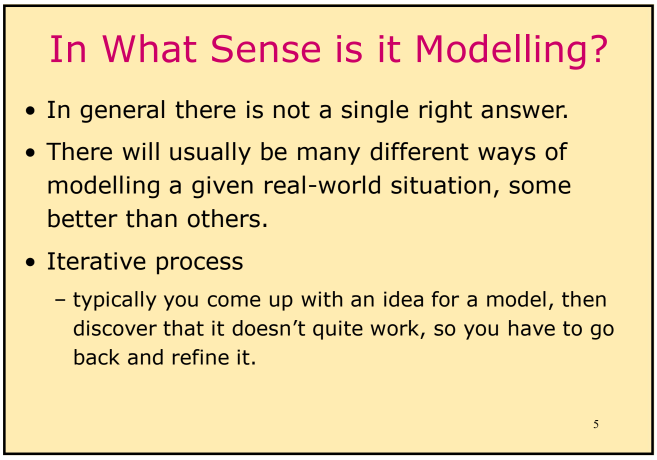
Identify important concepts and data needs.

Create a conceptual model.

Convert model to structures required by database (relational, object-oriented, etc.)

Implement using a DBMS: create tables, add data, constraints, etc.

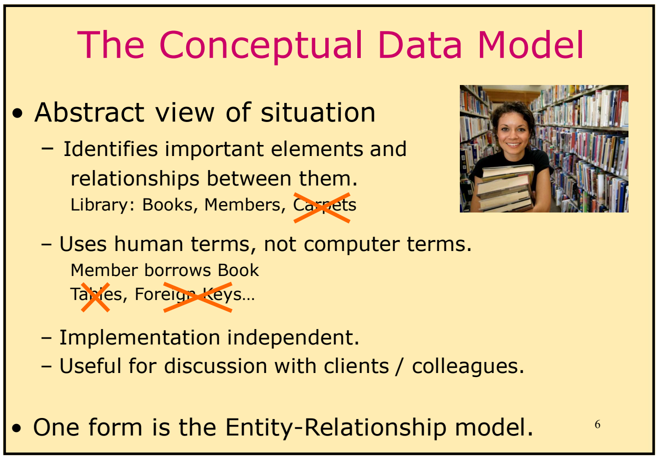
4



In What Sense is it Modelling?

- In general there is not a single right answer.
- There will usually be many different ways of modelling a given real-world situation, some better than others.
- Iterative process
  - typically you come up with an idea for a model, then discover that it doesn't quite work, so you have to go back and refine it.

5

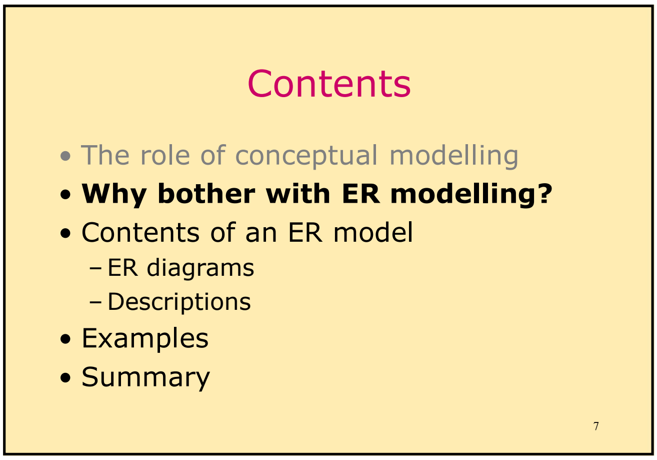


The Conceptual Data Model

- Abstract view of situation
  - Identifies important elements and relationships between them.
    Library: Books, Members, ~~Carpets~~
  - Uses human terms, not computer terms.
    Member borrows Book
    ~~Tables, Foreign Keys~~…
  - Implementation independent.
  - Useful for discussion with clients / colleagues.
- One form is the Entity-Relationship model.

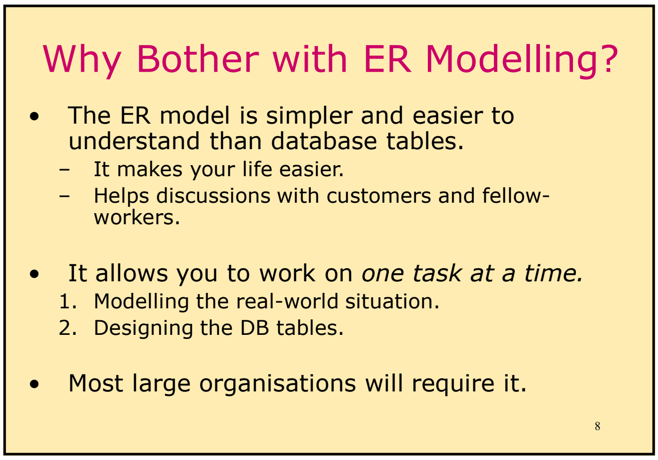
6

Dr C.H. Bryant, School of SEE, Salford, UK.



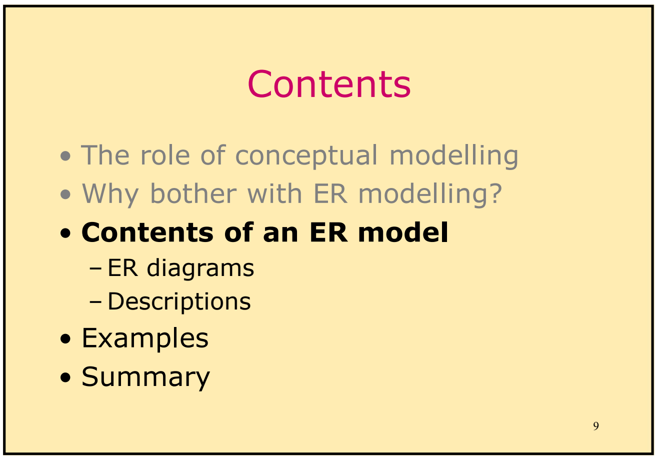
Contents

- The role of conceptual modelling
- **Why bother with ER modelling?**
- Contents of an ER model
  - ER diagrams
  - Descriptions
- Examples
- Summary



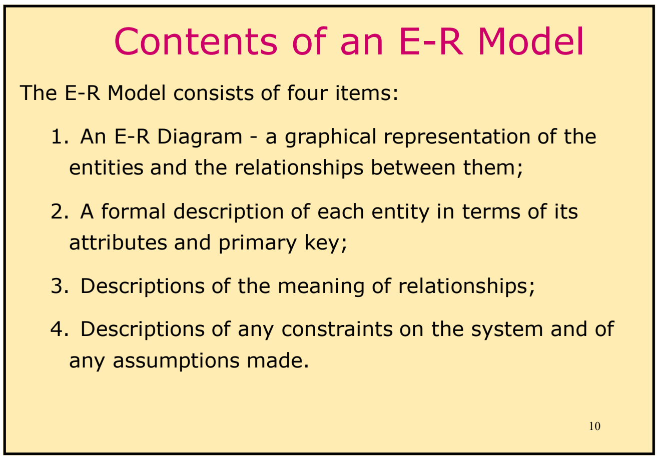
Why Bother with ER Modelling?

- The ER model is simpler and easier to understand than database tables.
  - It makes your life easier.
  - Helps discussions with customers and fellow-workers.
- It allows you to work on *one task at a time.*
  1. Modelling the real-world situation.
  2. Designing the DB tables.
- Most large organisations will require it.



Contents

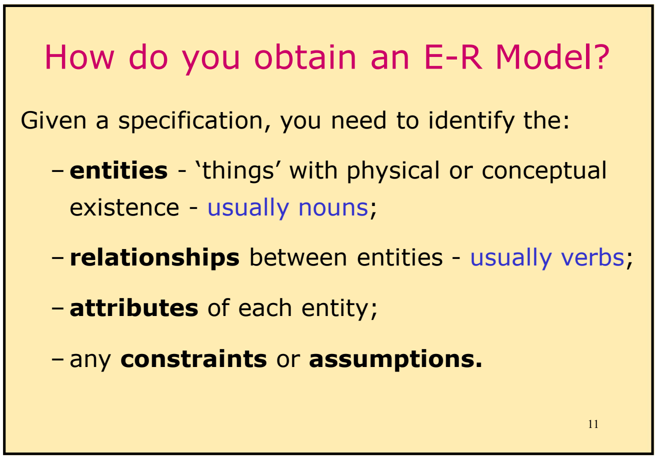
- The role of conceptual modelling
- Why bother with ER modelling?
- **Contents of an ER model**
  - ER diagrams
  - Descriptions
- Examples
- Summary



Contents of an E-R Model

The E-R Model consists of four items:

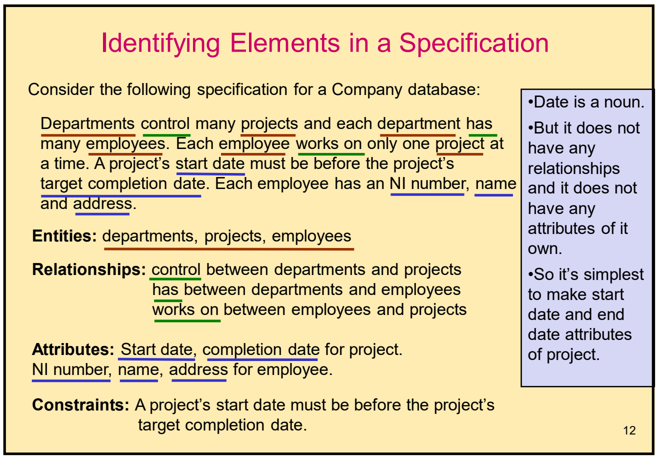
1. An E-R Diagram - a graphical representation of the entities and the relationships between them;
2. A formal description of each entity in terms of its attributes and primary key;
3. Descriptions of the meaning of relationships;
4. Descriptions of any constraints on the system and of any assumptions made.



How do you obtain an E-R Model?

Given a specification, you need to identify the:

- **entities** - 'things' with physical or conceptual existence - usually nouns;
- **relationships** between entities - usually verbs;
- **attributes** of each entity;
- any **constraints** or **assumptions.**



Identifying Elements in a Specification

Consider the following specification for a Company database:

Departments control many projects and each department has many employees. Each employee works on only one project at a time. A project's start date must be before the project's target completion date. Each employee has an NI number, name and address.

Entities: departments, projects, employees

Relationships: control between departments and projects
has between departments and employees
works on between employees and projects

Attributes: Start date, completion date for project.
NI number, name, address for employee.

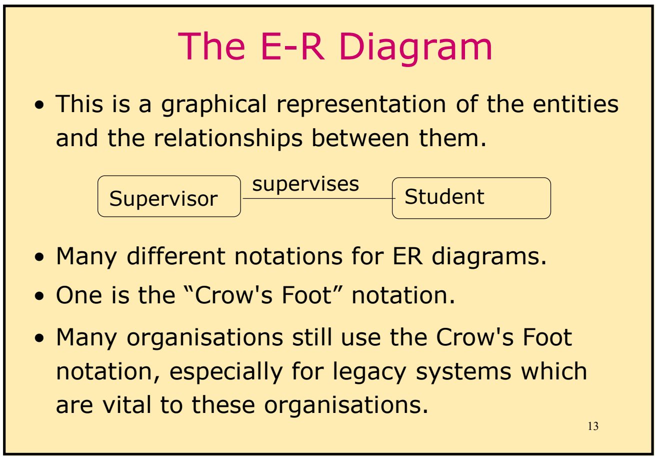
Constraints: A project's start date must be before the project's target completion date.

•Date is a noun.

•But it does not have any relationships and it does not have any attributes of it own.

•So it's simplest to make start date and end date attributes of project.

Dr C.H. Bryant, School of SEE, Salford, UK.



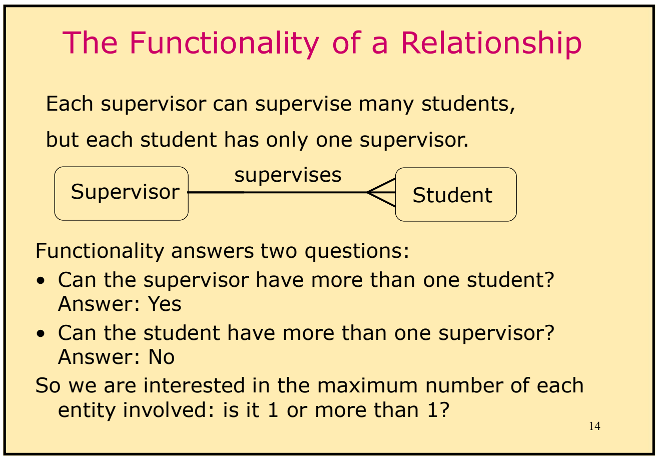
The E-R Diagram

- This is a graphical representation of the entities and the relationships between them.

Supervisor — supervises — Student

- Many different notations for ER diagrams.
- One is the "Crow's Foot" notation.
- Many organisations still use the Crow's Foot notation, especially for legacy systems which are vital to these organisations.

13



The Functionality of a Relationship

Each supervisor can supervise many students,
but each student has only one supervisor.

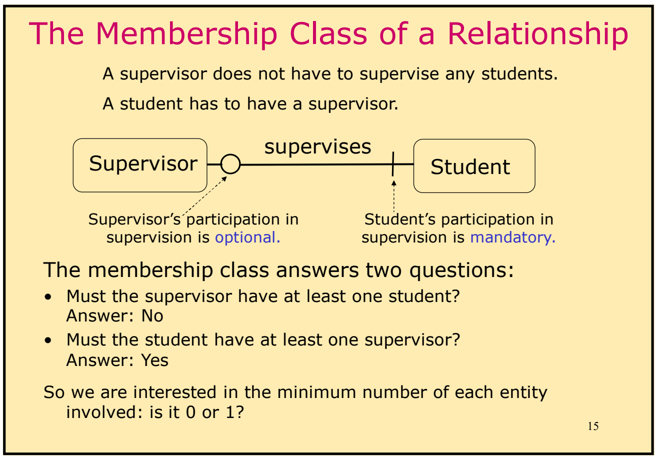
Supervisor — supervises — Student

Functionality answers two questions:

- Can the supervisor have more than one student? Answer: Yes
- Can the student have more than one supervisor? Answer: No

So we are interested in the maximum number of each entity involved: is it 1 or more than 1?

14



The Membership Class of a Relationship

A supervisor does not have to supervise any students.

A student has to have a supervisor.

Supervisor — supervises — Student

Supervisor's participation in supervision is optional.

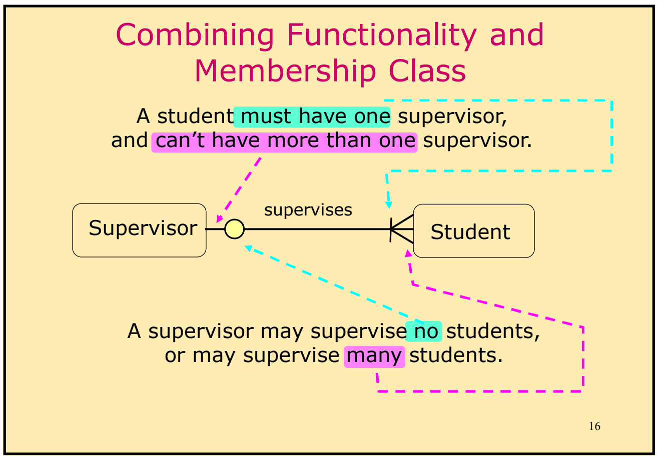
Student's participation in supervision is mandatory.

The membership class answers two questions:

- Must the supervisor have at least one student? Answer: No
- Must the student have at least one supervisor? Answer: Yes

So we are interested in the minimum number of each entity involved: is it 0 or 1?

15



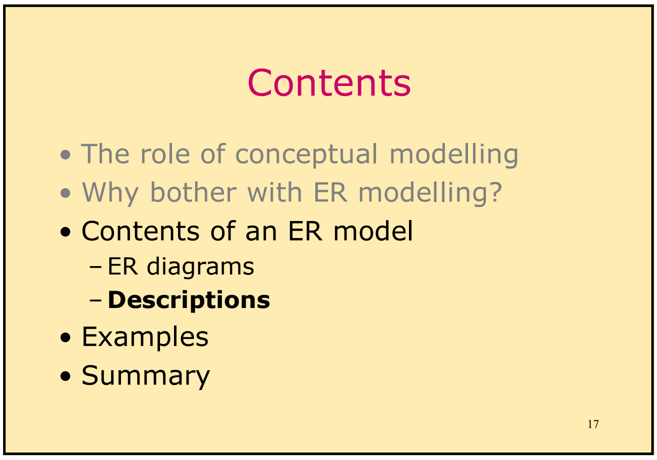
Combining Functionality and Membership Class

A student must have one supervisor,
and can't have more than one supervisor.

Supervisor — supervises — Student

A supervisor may supervise no students,
or may supervise many students.

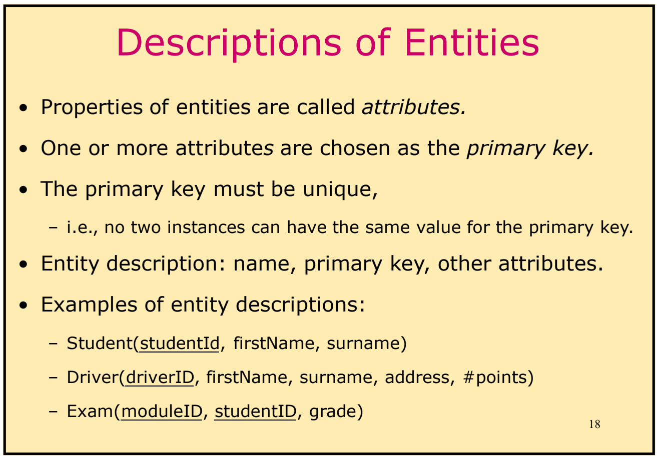
16



Contents

- The role of conceptual modelling
- Why bother with ER modelling?
- Contents of an ER model
  - ER diagrams
  - **Descriptions**
- Examples
- Summary

17

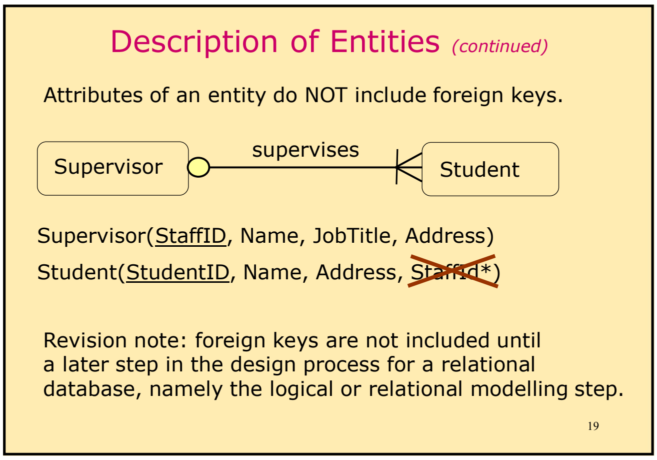


Descriptions of Entities

- Properties of entities are called *attributes.*
- One or more attributes are chosen as the *primary key.*
- The primary key must be unique,
  - i.e., no two instances can have the same value for the primary key.
- Entity description: name, primary key, other attributes.
- Examples of entity descriptions:
  - Student(studentId, firstName, surname)
  - Driver(driverID, firstName, surname, address, #points)
  - Exam(moduleID, studentID, grade)

18

Conceptual Modelling



Description of Entities *(continued)*

Attributes of an entity do NOT include foreign keys.

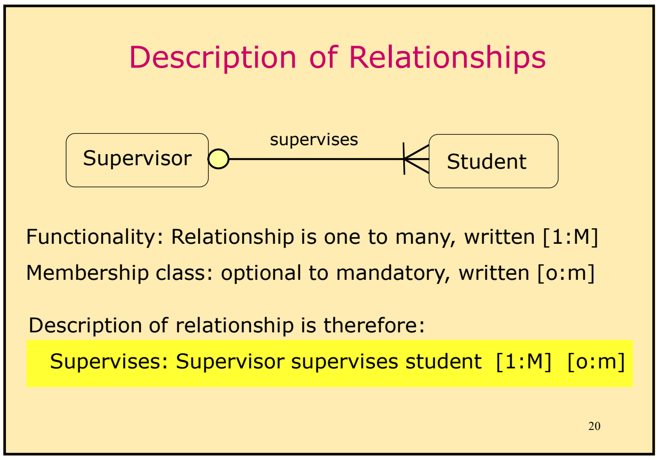
Supervisor — supervises — Student

Supervisor(<u>StaffID</u>, Name, JobTitle, Address)

Student(<u>StudentID</u>, Name, Address, ~~StaffId*~~)

Revision note: foreign keys are not included until a later step in the design process for a relational database, namely the logical or relational modelling step.

19



Description of Relationships

Supervisor — supervises — Student

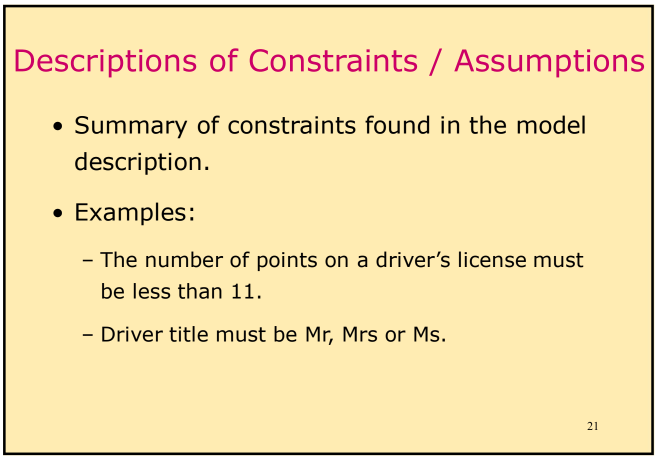
Functionality: Relationship is one to many, written [1:M]

Membership class: optional to mandatory, written [o:m]

Description of relationship is therefore:

Supervises: Supervisor supervises student [1:M] [o:m]

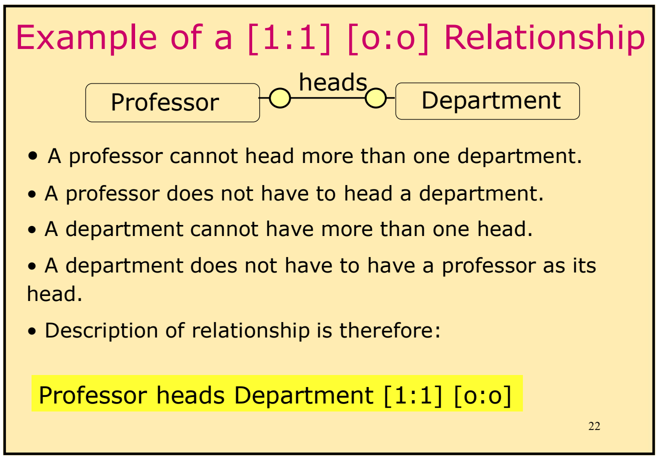
20



Descriptions of Constraints / Assumptions

- Summary of constraints found in the model description.
- Examples:
  - The number of points on a driver's license must be less than 11.
  - Driver title must be Mr, Mrs or Ms.

21



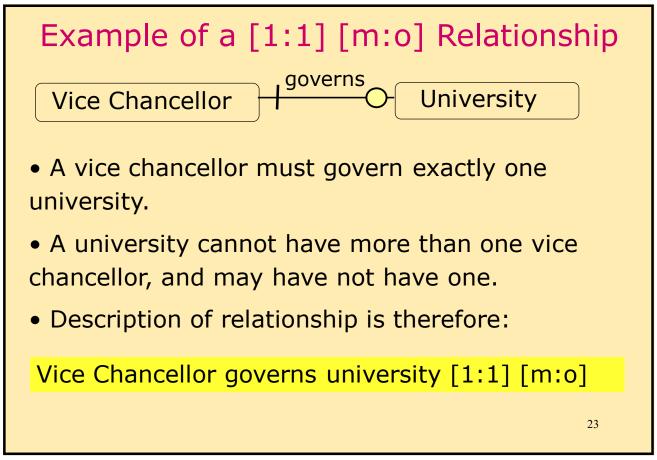
Example of a [1:1] [o:o] Relationship

Professor — heads — Department

- A professor cannot head more than one department.
- A professor does not have to head a department.
- A department cannot have more than one head.
- A department does not have to have a professor as its head.
- Description of relationship is therefore:

Professor heads Department [1:1] [o:o]

22



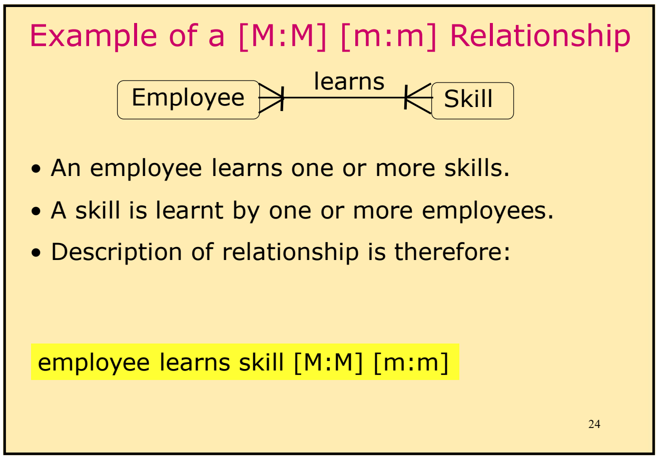
Example of a [1:1] [m:o] Relationship

Vice Chancellor — governs — University

- A vice chancellor must govern exactly one university.
- A university cannot have more than one vice chancellor, and may have not have one.
- Description of relationship is therefore:

Vice Chancellor governs university [1:1] [m:o]

23



Example of a [M:M] [m:m] Relationship

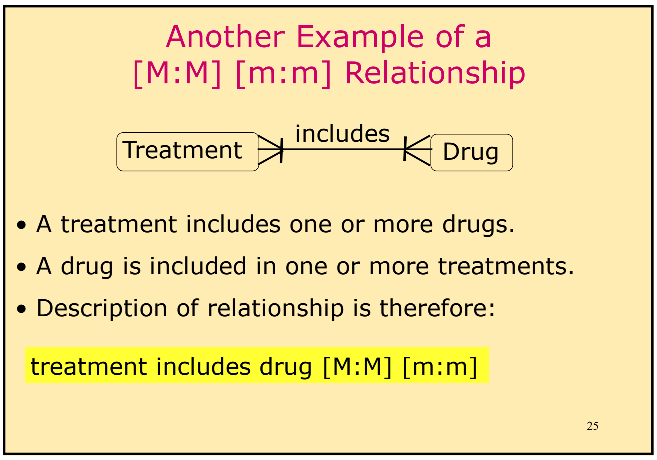
Employee — learns — Skill

- An employee learns one or more skills.
- A skill is learnt by one or more employees.
- Description of relationship is therefore:

employee learns skill [M:M] [m:m]

24

Dr C.H. Bryant, School of SEE, Salford, UK.



Another Example of a [M:M] [m:m] Relationship

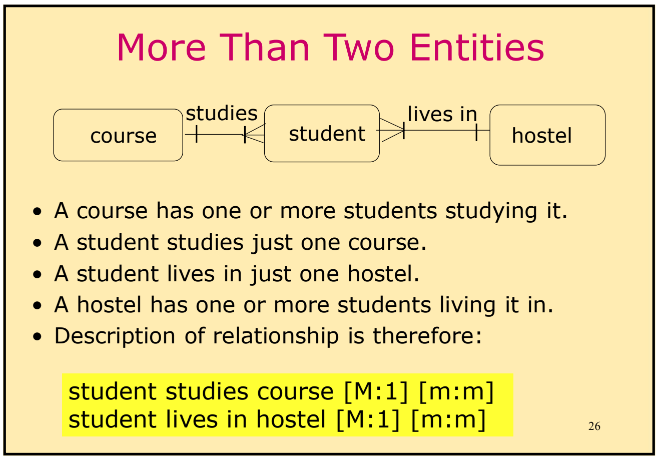
Treatment
includes
Drug

- A treatment includes one or more drugs.
- A drug is included in one or more treatments.
- Description of relationship is therefore:

treatment includes drug [M:M] [m:m]

25

25



More Than Two Entities

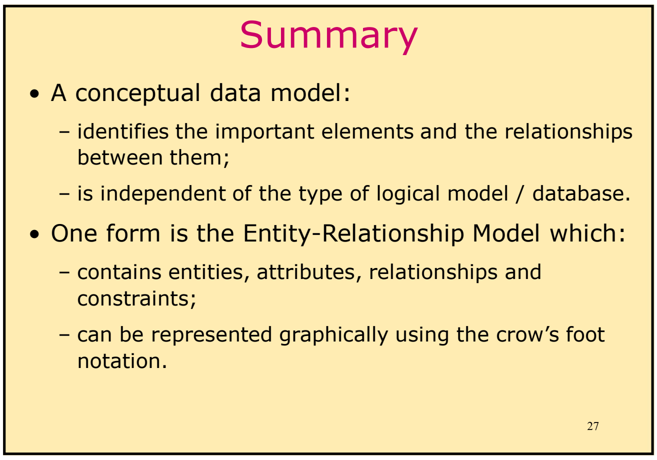
course
studies
student
lives in
hostel

- A course has one or more students studying it.
- A student studies just one course.
- A student lives in just one hostel.
- A hostel has one or more students living it in.
- Description of relationship is therefore:

student studies course [M:1] [m:m]
student lives in hostel [M:1] [m:m]

26

26

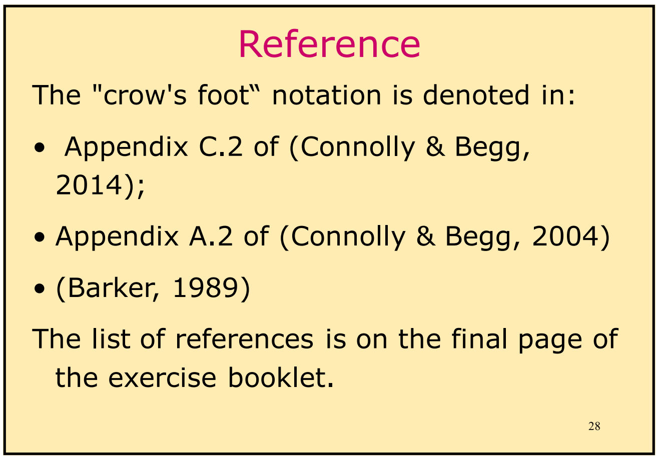


Summary

- A conceptual data model:
  - identifies the important elements and the relationships between them;
  - is independent of the type of logical model / database.
- One form is the Entity-Relationship Model which:
  - contains entities, attributes, relationships and constraints;
  - can be represented graphically using the crow's foot notation.

27

27



Reference

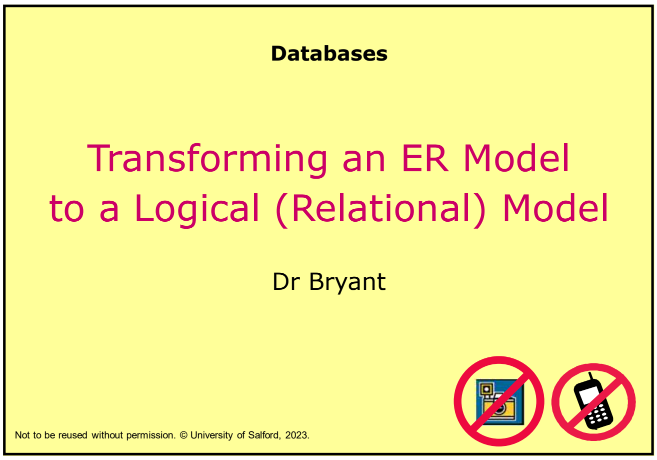
The "crow's foot‟ notation is denoted in:

- Appendix C.2 of (Connolly & Begg, 2014);
- Appendix A.2 of (Connolly & Begg, 2004)
- (Barker, 1989)

The list of references is on the final page of the exercise booklet.

28

28

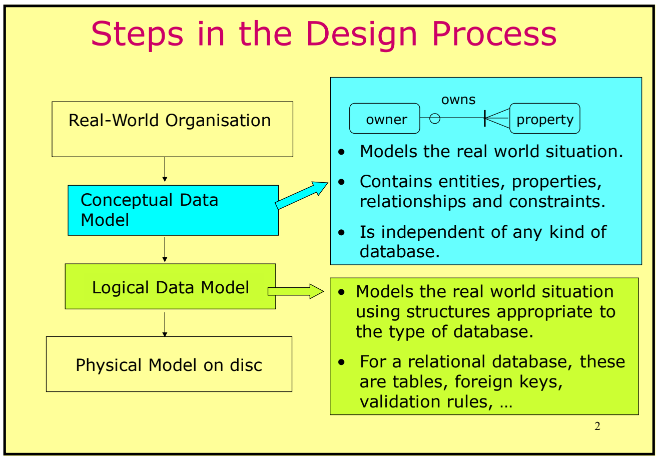


Databases

Transforming an ER Model to a Logical (Relational) Model

Dr Bryant

1

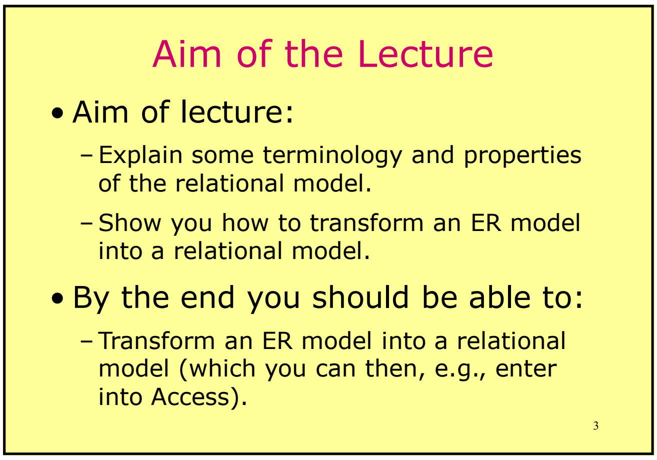


Steps in the Design Process

- Models the real world situation.
- Contains entities, properties, relationships and constraints.
- Is independent of any kind of database.

- Models the real world situation using structures appropriate to the type of database.
- For a relational database, these are tables, foreign keys, validation rules, …

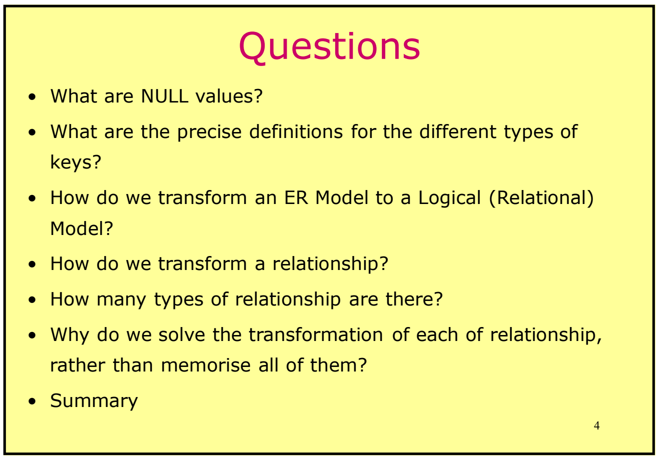
2



Aim of the Lecture

- Aim of lecture:
  - Explain some terminology and properties of the relational model.
  - Show you how to transform an ER model into a relational model.
- By the end you should be able to:
  - Transform an ER model into a relational model (which you can then, e.g., enter into Access).

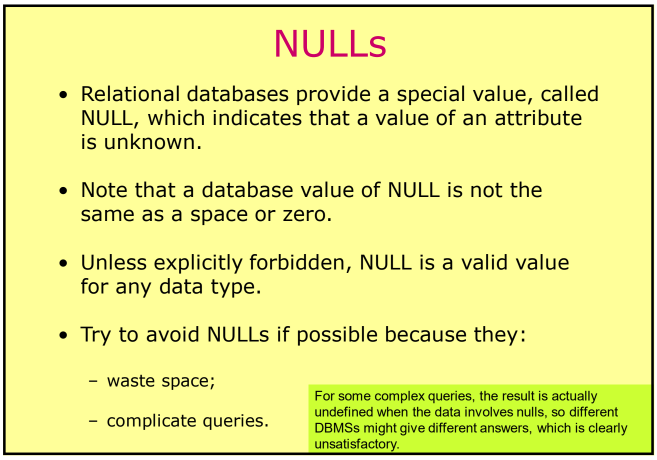
3



Questions

- What are NULL values?
- What are the precise definitions for the different types of keys?
- How do we transform an ER Model to a Logical (Relational) Model?
- How do we transform a relationship?
- How many types of relationship are there?
- Why do we solve the transformation of each of relationship, rather than memorise all of them?
- Summary

4

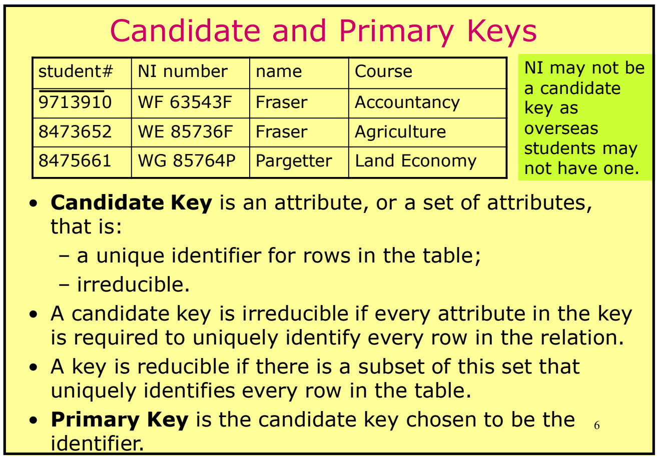


NULLs

- Relational databases provide a special value, called NULL, which indicates that a value of an attribute is unknown.
- Note that a database value of NULL is not the same as a space or zero.
- Unless explicitly forbidden, NULL is a valid value for any data type.
- Try to avoid NULLs if possible because they:
  - waste space;
  - complicate queries.

For some complex queries, the result is actually undefined when the data involves nulls, so different DBMSs might give different answers, which is clearly unsatisfactory.

5



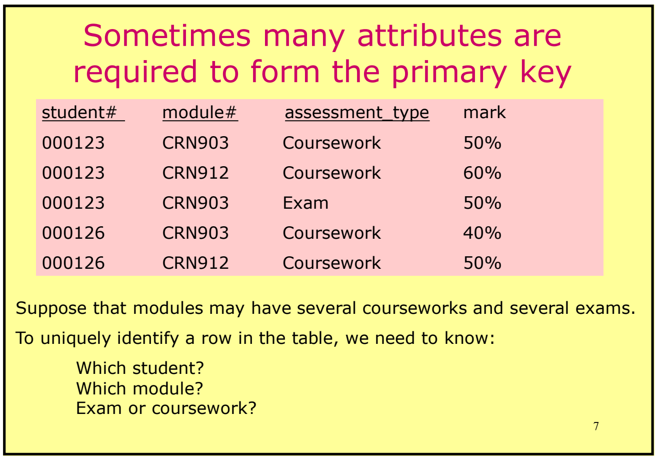
Candidate and Primary Keys

student#	NI number	name	Course
9713910	WF 63543F	Fraser	Accountancy
8473652	WE 85736F	Fraser	Agriculture
8475661	WG 85764P	Pargetter	Land Economy

NI may not be a candidate key as overseas students may not have one.

- **Candidate Key** is an attribute, or a set of attributes, that is:
  - a unique identifier for rows in the table;
  - irreducible.
- A candidate key is irreducible if every attribute in the key is required to uniquely identify every row in the relation.
- A key is reducible if there is a subset of this set that uniquely identifies every row in the table.
- **Primary Key** is the candidate key chosen to be the identifier.

6



Sometimes many attributes are required to form the primary key

student#	module#	assessment_type	mark
000123	CRN903	Coursework	50%
000123	CRN912	Coursework	60%
000123	CRN903	Exam	50%
000126	CRN903	Coursework	40%
000126	CRN912	Coursework	50%

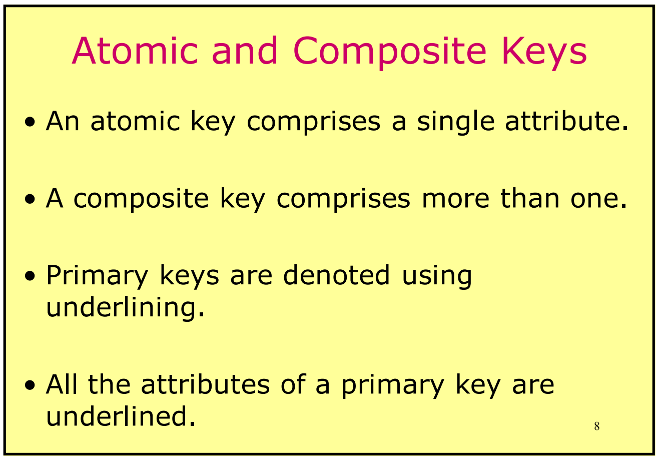
Suppose that modules may have several courseworks and several exams.

To uniquely identify a row in the table, we need to know:

Which student?
Which module?
Exam or coursework?

7

7

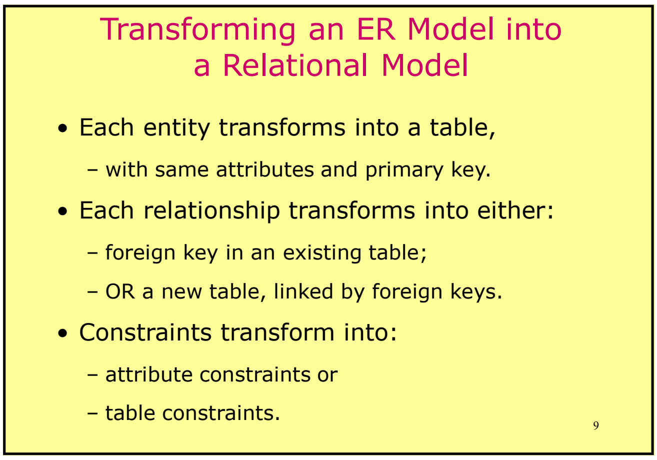


Atomic and Composite Keys

- An atomic key comprises a single attribute.
- A composite key comprises more than one.
- Primary keys are denoted using underlining.
- All the attributes of a primary key are underlined.

8

8

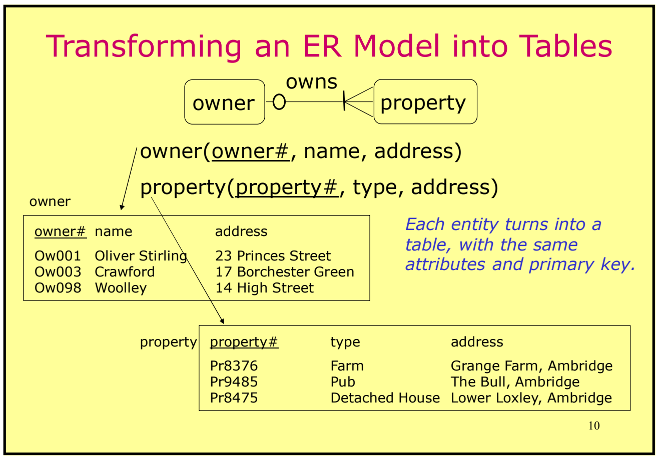


Transforming an ER Model into a Relational Model

- Each entity transforms into a table,
  - with same attributes and primary key.
- Each relationship transforms into either:
  - foreign key in an existing table;
  - OR a new table, linked by foreign keys.
- Constraints transform into:
  - attribute constraints or
  - table constraints.

9

9



Transforming an ER Model into Tables

owner(owner#, name, address)

property(property#, type, address)

owner

owner#	name	address
Ow001	Oliver Stirling	23 Princes Street
Ow003	Crawford	17 Borchester Green
Ow098	Woolley	14 High Street

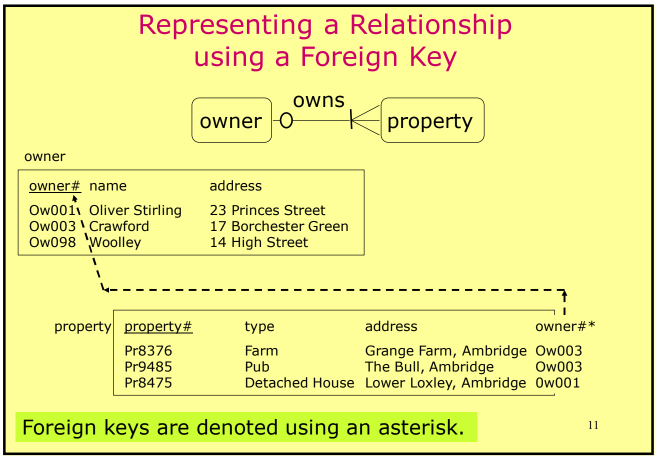
Each entity turns into a table, with the same attributes and primary key.

property

property#	type	address
Pr8376	Farm	Grange Farm, Ambridge
Pr9485	Pub	The Bull, Ambridge
Pr8475	Detached House	Lower Loxley, Ambridge

10

10



Representing a Relationship using a Foreign Key

owner

owner#	name	address
Ow001	Oliver Stirling	23 Princes Street
Ow003	Crawford	17 Borchester Green
Ow098	Woolley	14 High Street

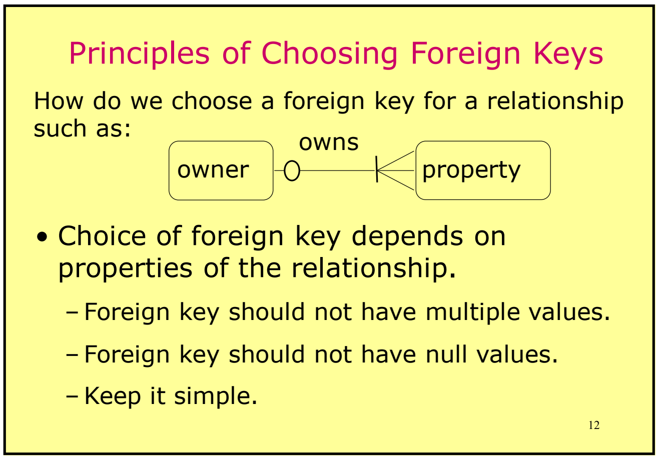
property

property#	type	address	owner#*
Pr8376	Farm	Grange Farm, Ambridge	Ow003
Pr9485	Pub	The Bull, Ambridge	Ow003
Pr8475	Detached House	Lower Loxley, Ambridge	Ow001

Foreign keys are denoted using an asterisk.

11

11



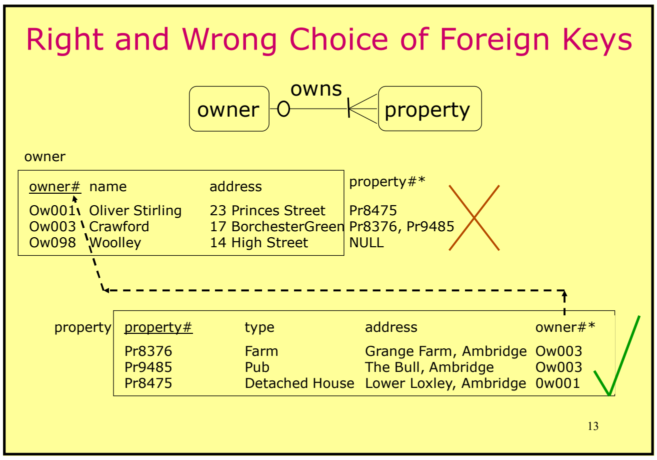
Principles of Choosing Foreign Keys

How do we choose a foreign key for a relationship such as:

- Choice of foreign key depends on properties of the relationship.
  - Foreign key should not have multiple values.
  - Foreign key should not have null values.
  - Keep it simple.

12

12



Right and Wrong Choice of Foreign Keys

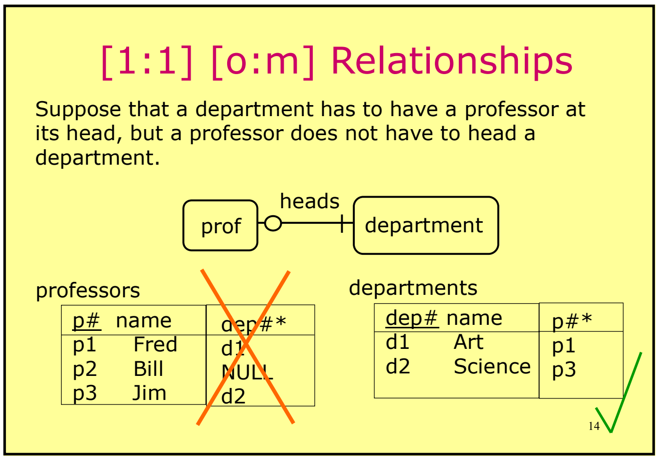
owner

owner#	name	address	property#*
Ow001	Oliver Stirling	23 Princes Street	Pr8475
Ow003	Crawford	17 BorchesterGreen	Pr8376, Pr9485
Ow098	Woolley	14 High Street	NULL

property

property#	type	address	owner#*
Pr8376	Farm	Grange Farm, Ambridge	Ow003
Pr9485	Pub	The Bull, Ambridge	Ow003
Pr8475	Detached House	Lower Loxley, Ambridge	0w001

13



[1:1] [o:m] Relationships

Suppose that a department has to have a professor at its head, but a professor does not have to head a department.

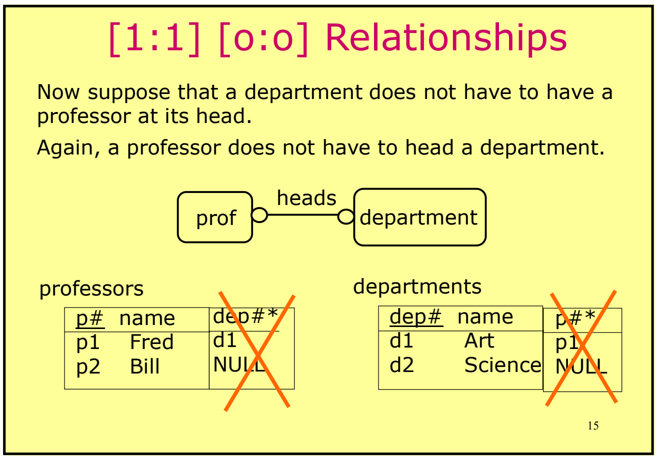
professors

p#	name	dep#*
p1	Fred	d1
p2	Bill	NULL
p3	Jim	d2

departments

dep#	name	p#*
d1	Art	p1
d2	Science	p3

14



[1:1] [o:o] Relationships

Now suppose that a department does not have to have a professor at its head.

Again, a professor does not have to head a department.

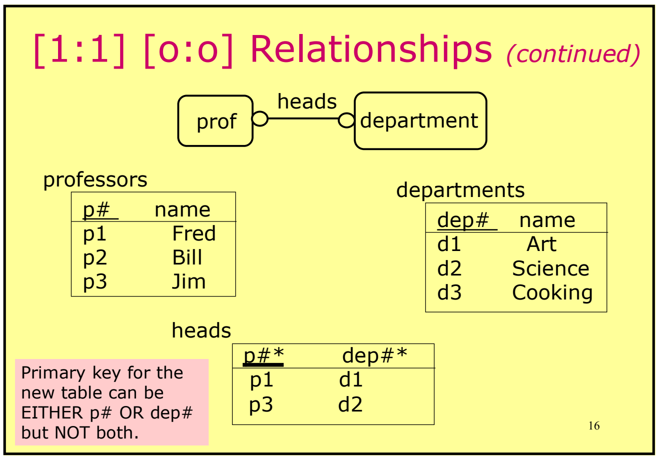
professors

p#	name	dep#*
p1	Fred	d1
p2	Bill	NULL

departments

dep#	name	p#*
d1	Art	p1
d2	Science	NULL

15



[1:1] [o:o] Relationships *(continued)*

professors

p#	name
p1	Fred
p2	Bill
p3	Jim

departments

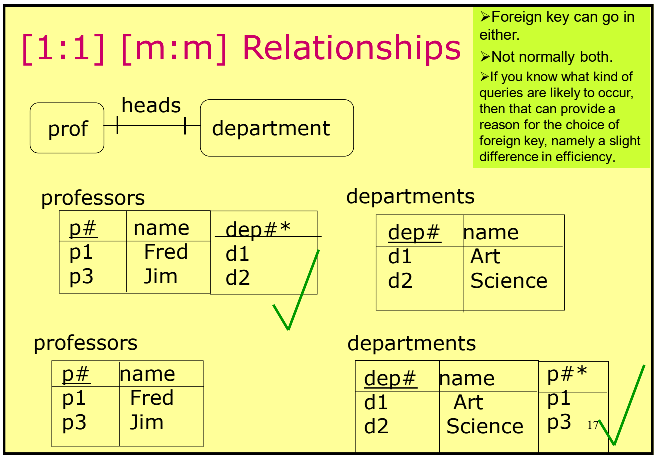
dep#	name
d1	Art
d2	Science
d3	Cooking

heads

p#*	dep#*
p1	d1
p3	d2

Primary key for the new table can be EITHER p# OR dep# but NOT both.

16



[1:1] [m:m] Relationships

- Foreign key can go in either.
- Not normally both.
- If you know what kind of queries are likely to occur, then that can provide a reason for the choice of foreign key, namely a slight difference in efficiency.

professors

p#	name	dep#*
p1	Fred	d1
p3	Jim	d2

departments

dep#	name
d1	Art
d2	Science

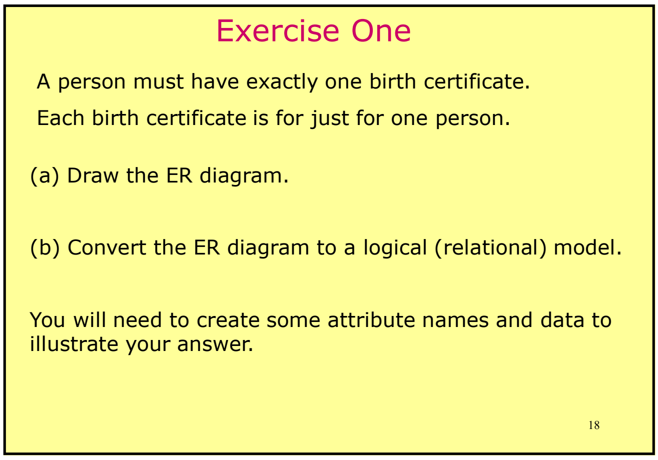
professors

p#	name
p1	Fred
p3	Jim

departments

dep#	name	p#*
d1	Art	p1
d2	Science	p3

17



Exercise One

A person must have exactly one birth certificate.

Each birth certificate is for just for one person.

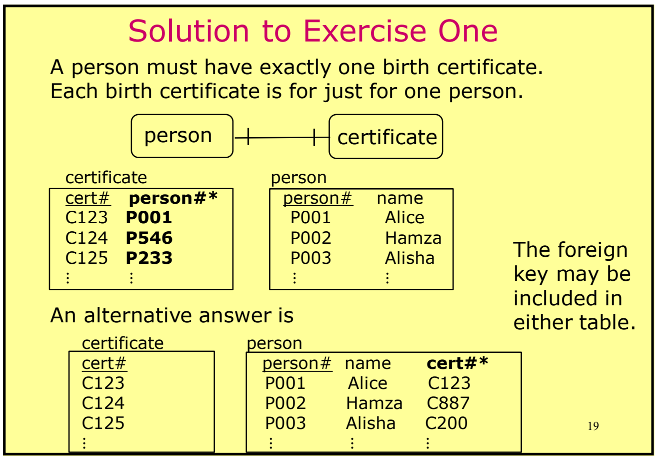
(a) Draw the ER diagram.

(b) Convert the ER diagram to a logical (relational) model.

You will need to create some attribute names and data to illustrate your answer.

18

Dr C.H. Bryant, School of SEE, Salford, UK.



Solution to Exercise One

A person must have exactly one birth certificate.
Each birth certificate is for just for one person.

person
certificate

certificate

cert#	person#*
C123	**P001**
C124	**P546**
C125	**P233**
⋮	⋮

person

person#	name
P001	Alice
P002	Hamza
P003	Alisha
⋮	⋮

The foreign key may be included in either table.

An alternative answer is

certificate

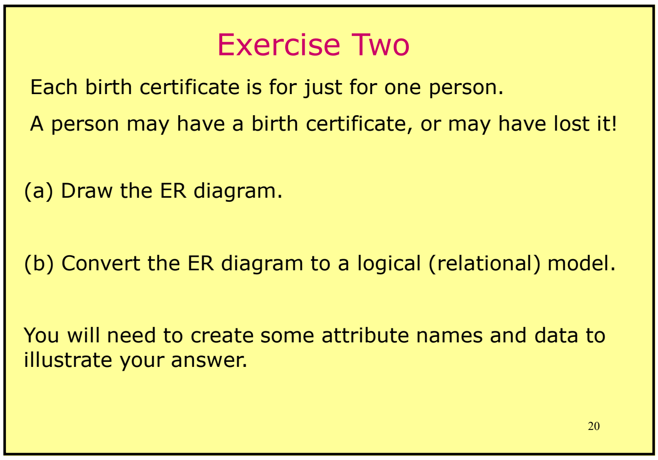
cert#
C123
C124
C125
⋮

person

person#	name	cert#*
P001	Alice	C123
P002	Hamza	C887
P003	Alisha	C200
⋮	⋮	⋮

19

19



Exercise Two

Each birth certificate is for just for one person.

A person may have a birth certificate, or may have lost it!

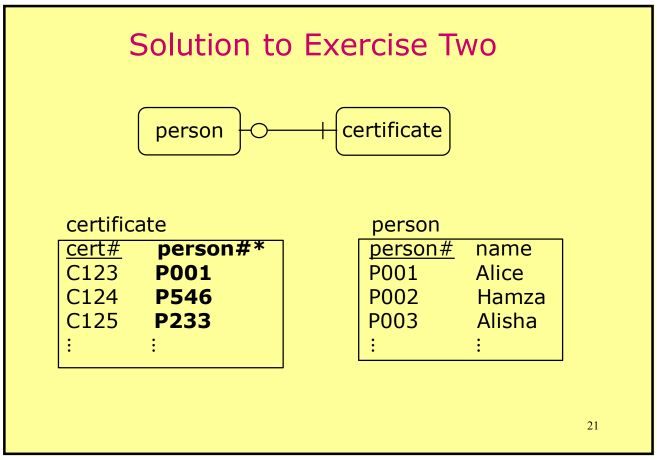
(a) Draw the ER diagram.

(b) Convert the ER diagram to a logical (relational) model.

You will need to create some attribute names and data to illustrate your answer.

20

20



Solution to Exercise Two

person
certificate

certificate

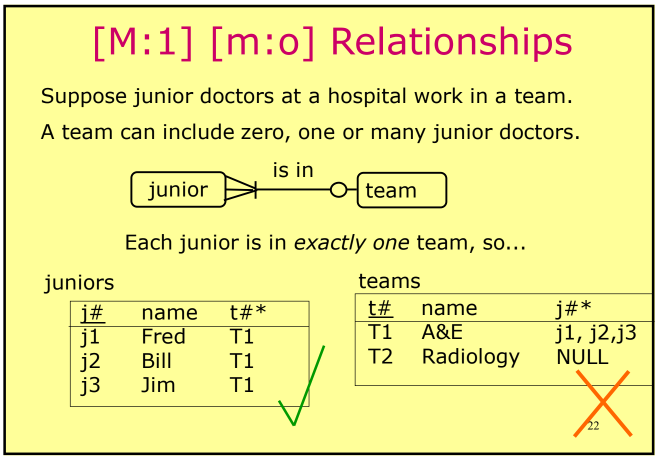
cert#	person#*
C123	**P001**
C124	**P546**
C125	**P233**
⋮	⋮

person

person#	name
P001	Alice
P002	Hamza
P003	Alisha
⋮	⋮

21

21



[M:1] [m:o] Relationships

Suppose junior doctors at a hospital work in a team.

A team can include zero, one or many junior doctors.

junior
is in
team

Each junior is in *exactly one* team, so...

juniors

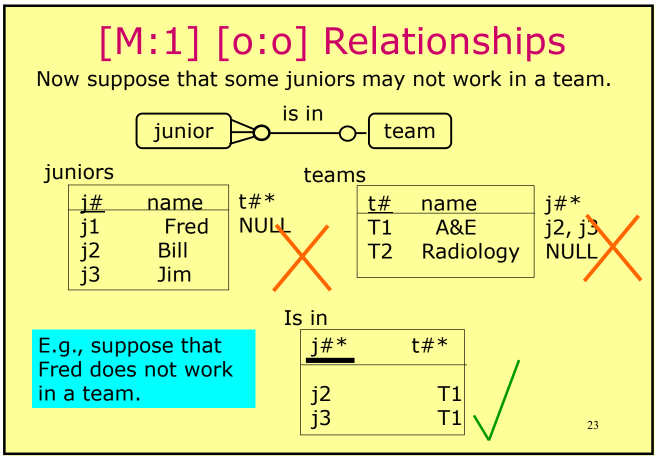
j#	name	t#*
j1	Fred	T1
j2	Bill	T1
j3	Jim	T1

teams

t#	name	j#*
T1	A&E	j1, j2,j3
T2	Radiology	NULL

22

22



[M:1] [o:o] Relationships

Now suppose that some juniors may not work in a team.

junior
is in
team

juniors

j#	name	t#*
j1	Fred	NULL
j2	Bill	
j3	Jim	

teams

t#	name	j#*
T1	A&E	j2, j3
T2	Radiology	NULL

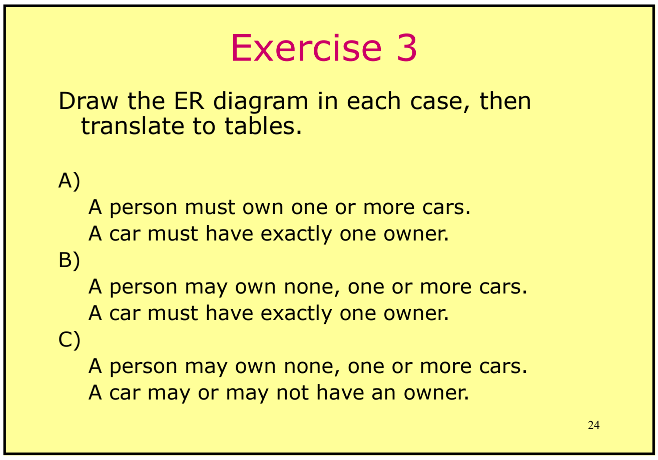
E.g., suppose that Fred does not work in a team.

Is in

j#*	t#*
j2	T1
j3	T1

23

23



Exercise 3

Draw the ER diagram in each case, then translate to tables.

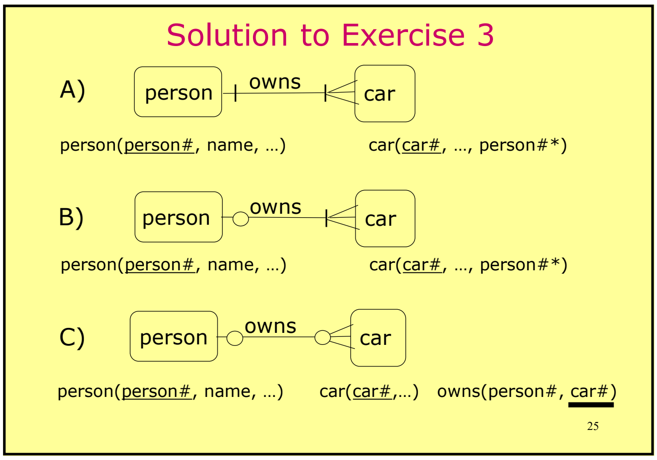
A)
A person must own one or more cars.
A car must have exactly one owner.

B)
A person may own none, one or more cars.
A car must have exactly one owner.

C)
A person may own none, one or more cars.
A car may or may not have an owner.

24

24



Solution to Exercise 3

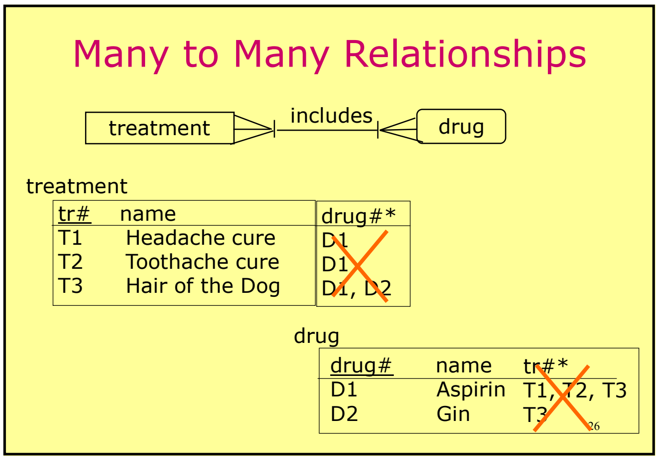
A) person(person#, name, …) car(car#, …, person#*)

B) person(person#, name, …) car(car#, …, person#*)

C) person(person#, name, …) car(car#,…) owns(person#, car#)

25

25



Many to Many Relationships

treatment

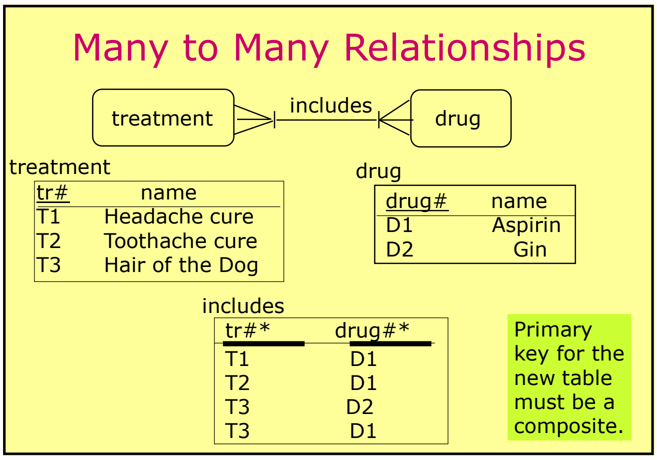
tr#	name	drug#*
T1	Headache cure	~~D1~~
T2	Toothache cure	~~D1~~
T3	Hair of the Dog	~~D1, D2~~

drug

drug#	name	tr#*
D1	Aspirin	~~T1, T2, T3~~
D2	Gin	~~T3~~

26

26



Many to Many Relationships

treatment

tr#	name
T1	Headache cure
T2	Toothache cure
T3	Hair of the Dog

drug

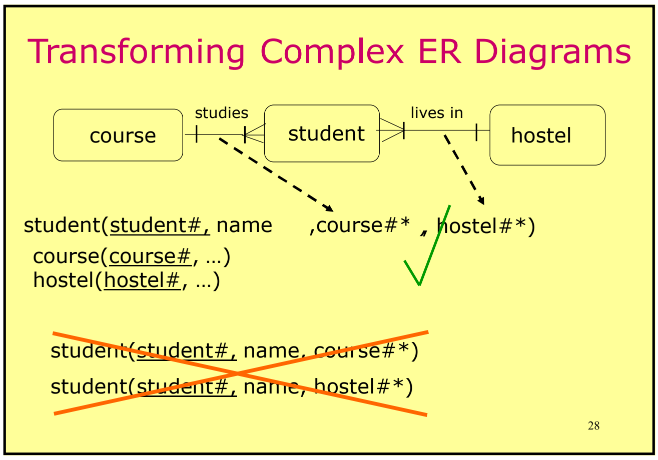
drug#	name
D1	Aspirin
D2	Gin

includes

tr#*	drug#*
T1	D1
T2	D1
T3	D2
T3	D1

Primary key for the new table must be a composite.

27



Transforming Complex ER Diagrams

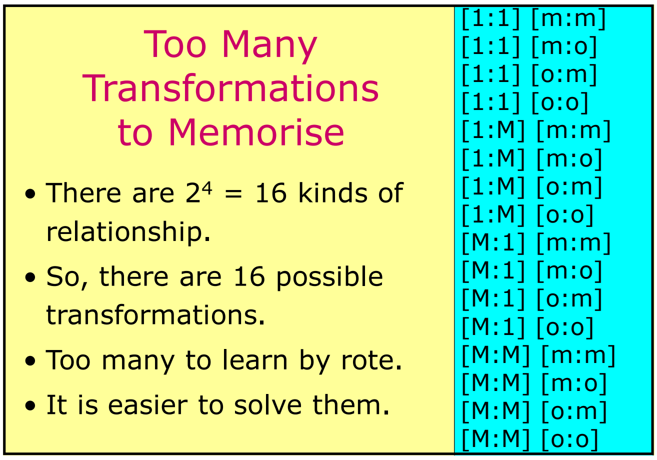
student(student#, name ,course#* , hostel#*) ✓

course(course#, …)
hostel(hostel#, …)

~~student(student#, name, course#*)~~
~~student(student#, name, hostel#*)~~

28

28

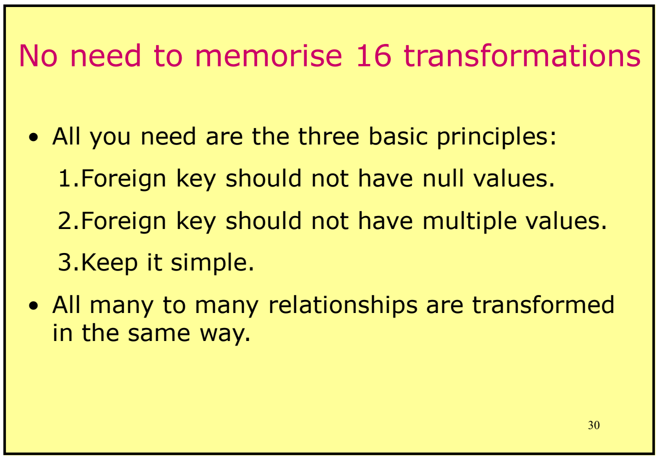


Too Many Transformations to Memorise

- There are 2^4 = 16 kinds of relationship.
- So, there are 16 possible transformations.
- Too many to learn by rote.
- It is easier to solve them.

[1:1] [m:m]
[1:1] [m:o]
[1:1] [o:m]
[1:1] [o:o]
[1:M] [m:m]
[1:M] [m:o]
[1:M] [o:m]
[1:M] [o:o]
[M:1] [m:m]
[M:1] [m:o]
[M:1] [o:m]
[M:1] [o:o]
[M:M] [m:m]
[M:M] [m:o]
[M:M] [o:m]
[M:M] [o:o]

29

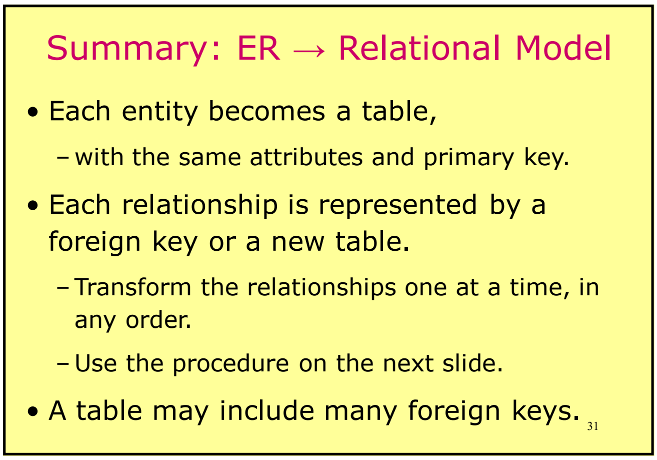


No need to memorise 16 transformations

- All you need are the three basic principles:
  1. Foreign key should not have null values.
  2. Foreign key should not have multiple values.
  3. Keep it simple.
- All many to many relationships are transformed in the same way.

30

30

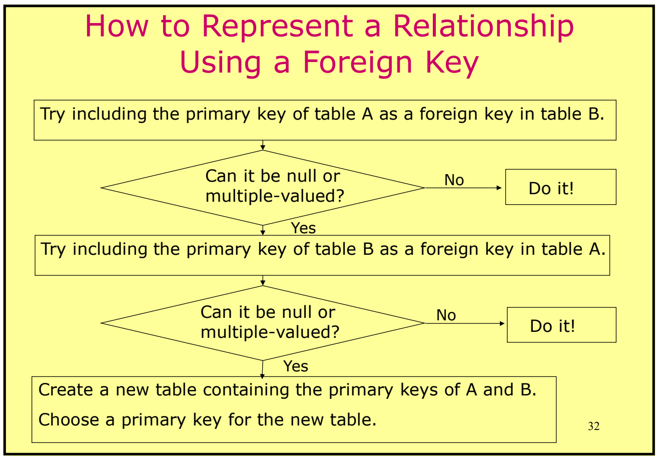


Summary: ER → Relational Model

- Each entity becomes a table,
  - with the same attributes and primary key.
- Each relationship is represented by a foreign key or a new table.
  - Transform the relationships one at a time, in any order.
  - Use the procedure on the next slide.
- A table may include many foreign keys.

31

31



How to Represent a Relationship Using a Foreign Key

Try including the primary key of table A as a foreign key in table B.

Can it be null or multiple-valued?

No

Do it!

Yes

Try including the primary key of table B as a foreign key in table A.

Can it be null or multiple-valued?

No

Do it!

Yes

Create a new table containing the primary keys of A and B.

Choose a primary key for the new table.

32

32

Databases

Normalisation

Dr Bryant

Contents

- Why normalisation is useful
- Functional Dependency
- First normal form (1NF)
  - Repeating Groups
  - Information redundancy
  - Types of anomaly
- Full and Partial Functional Dependency
- Second normal form (2NF)
- Transitive Dependency
- Third normal form (3NF)
- Relationship between 1NF, 2NF and 3NF
- Summary and Reading

Designing a Database

- We have been studying how to design a database.
- We began with conceptual modelling.
- We focused on one particular form - the Entity-Relationship Model.
- In the previous lecture, we saw how we can transform an ER Model to a Logical (Relational) Model.

Identifying and Fixing Deign Faults

- In this lecture, you will learn how to:

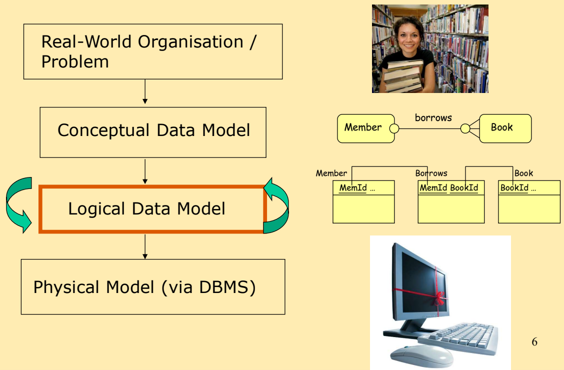
1. identify faults in the table design and
2. how to restructure your tables to remove the faults.

- In other words, you are going to learn how ensure that your tables are "**normalised**".

Identifying and Fixing Deign Faults

- Usually a good ER model will lead to a well-designed database. You can confirm this by checking it is "normalised".
- The situation you are modelling may change. As you change your database, you need to ensure it remains "normalised".
- You may inherit a messy database from someone else, and be asked to tidy it. When you tidy it, you need to ensure it ends up "normalised".

Where does Normalisation Fit in?

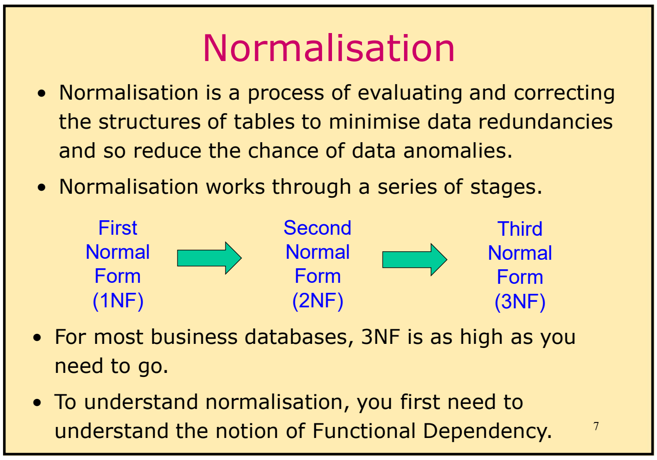


Real-World Organisation / Problem

Conceptual Data Model

Logical Data Model

Physical Model (via DBMS)



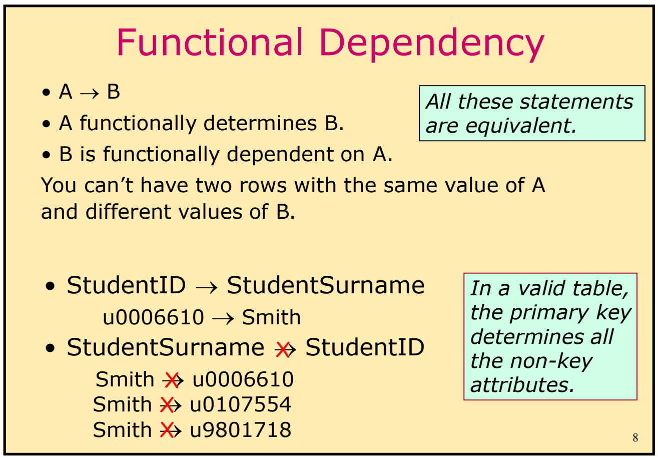
Normalisation

- Normalisation is a process of evaluating and correcting the structures of tables to minimise data redundancies and so reduce the chance of data anomalies.
- Normalisation works through a series of stages.

First Normal Form (1NF) → Second Normal Form (2NF) → Third Normal Form (3NF)

- For most business databases, 3NF is as high as you need to go.
- To understand normalisation, you first need to understand the notion of Functional Dependency.

7



Functional Dependency

- A → B
- A functionally determines B.
- B is functionally dependent on A.

All these statements are equivalent.

You can’t have two rows with the same value of A and different values of B.

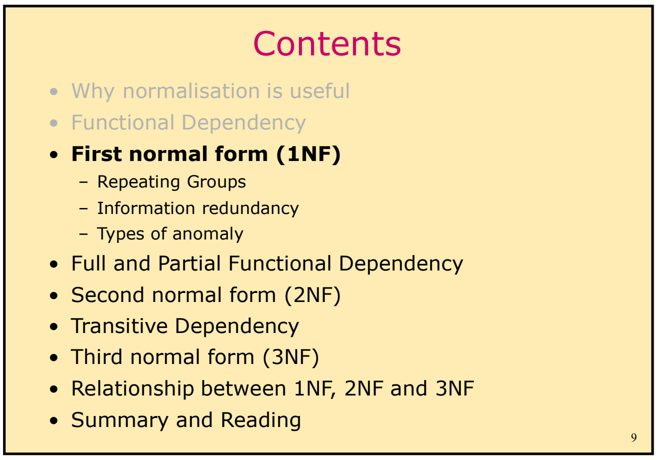
- StudentID → StudentSurname

  u0006610 → Smith
- StudentSurname $\not\to$ StudentID

  Smith $\not\to$ u0006610
  Smith $\not\to$ u0107554
  Smith $\not\to$ u9801718

In a valid table, the primary key determines all the non-key attributes.

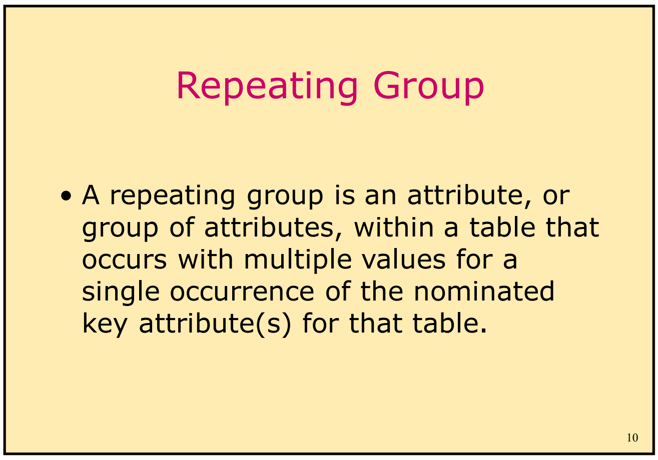
8



Contents

- Why normalisation is useful
- Functional Dependency
- **First normal form (1NF)**
  - Repeating Groups
  - Information redundancy
  - Types of anomaly
- Full and Partial Functional Dependency
- Second normal form (2NF)
- Transitive Dependency
- Third normal form (3NF)
- Relationship between 1NF, 2NF and 3NF
- Summary and Reading

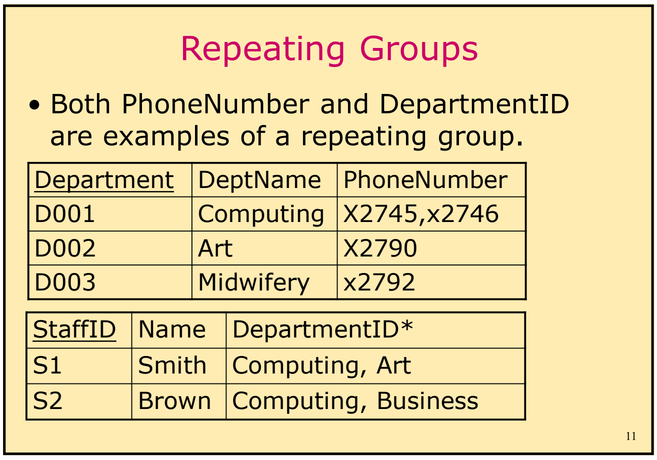
9



Repeating Group

- A repeating group is an attribute, or group of attributes, within a table that occurs with multiple values for a single occurrence of the nominated key attribute(s) for that table.

10



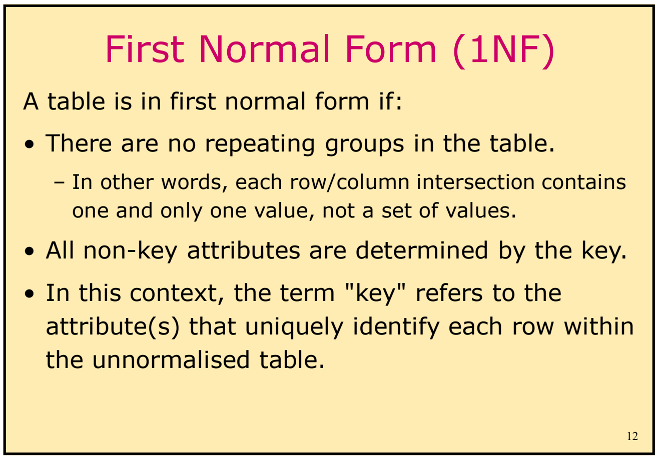
Repeating Groups

- Both PhoneNumber and DepartmentID are examples of a repeating group.

Department	DeptName	PhoneNumber
D001	Computing	X2745,x2746
D002	Art	X2790
D003	Midwifery	x2792

StaffID	Name	DepartmentID*
S1	Smith	Computing, Art
S2	Brown	Computing, Business

11

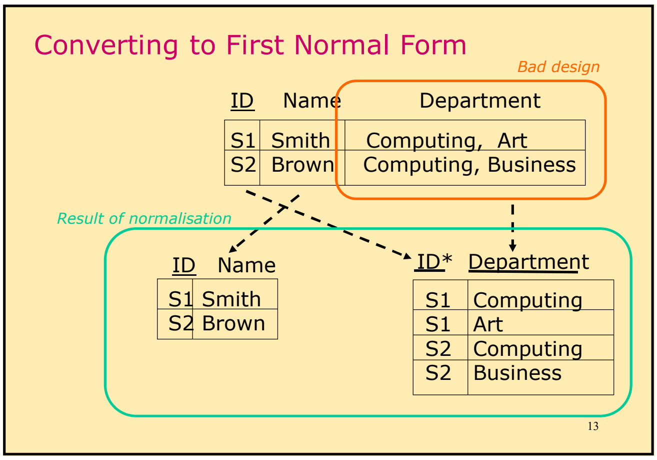


First Normal Form (1NF)

A table is in first normal form if:

- There are no repeating groups in the table.
  - In other words, each row/column intersection contains one and only one value, not a set of values.
- All non-key attributes are determined by the key.
- In this context, the term "key" refers to the attribute(s) that uniquely identify each row within the unnormalised table.

12



Converting to First Normal Form

Bad design

ID	Name	Department
S1	Smith	Computing, Art
S2	Brown	Computing, Business

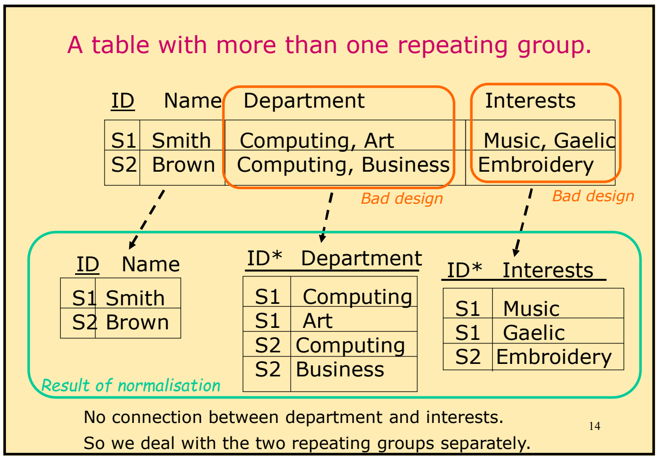
Result of normalisation

ID	Name
S1	Smith
S2	Brown

ID*	Department
S1	Computing
S1	Art
S2	Computing
S2	Business

13

13



A table with more than one repeating group.

ID	Name	Department	Interests
S1	Smith	Computing, Art	Music, Gaelic
S2	Brown	Computing, Business	Embroidery

Bad design

Bad design

ID	Name
S1	Smith
S2	Brown

ID*	Department
S1	Computing
S1	Art
S2	Computing
S2	Business

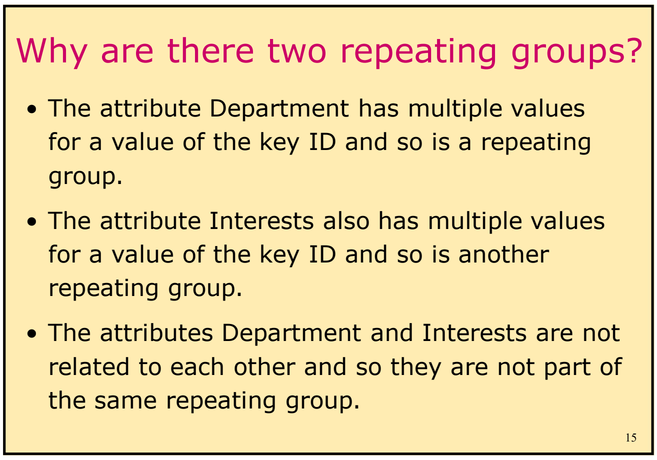
ID*	Interests
S1	Music
S1	Gaelic
S2	Embroidery

Result of normalisation

No connection between department and interests.
So we deal with the two repeating groups separately.

14

14

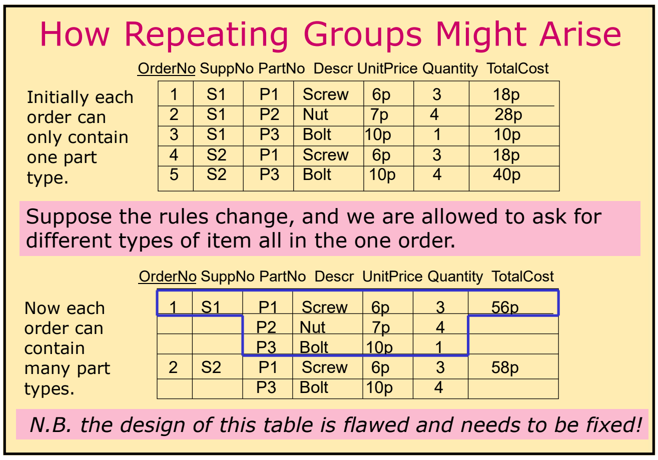


Why are there two repeating groups?

- The attribute Department has multiple values for a value of the key ID and so is a repeating group.
- The attribute Interests also has multiple values for a value of the key ID and so is another repeating group.
- The attributes Department and Interests are not related to each other and so they are not part of the same repeating group.

15

15



How Repeating Groups Might Arise

Initially each order can only contain one part type.

OrderNo	SuppNo	PartNo	Descr	UnitPrice	Quantity	TotalCost
1	S1	P1	Screw	6p	3	18p
2	S1	P2	Nut	7p	4	28p
3	S1	P3	Bolt	10p	1	10p
4	S2	P1	Screw	6p	3	18p
5	S2	P3	Bolt	10p	4	40p

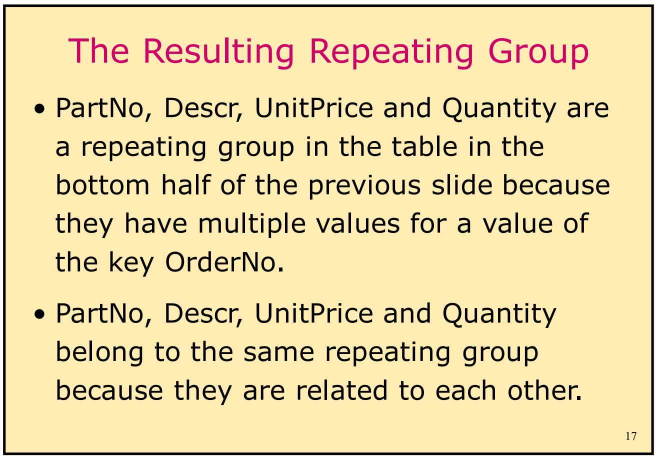
Suppose the rules change, and we are allowed to ask for different types of item all in the one order.

Now each order can contain many part types.

OrderNo	SuppNo	PartNo	Descr	UnitPrice	Quantity	TotalCost
1	S1	P1	Screw	6p	3	56p
		P2	Nut	7p	4	
		P3	Bolt	10p	1	
2	S2	P1	Screw	6p	3	58p
		P3	Bolt	10p	4	

N.B. the design of this table is flawed and needs to be fixed!

16

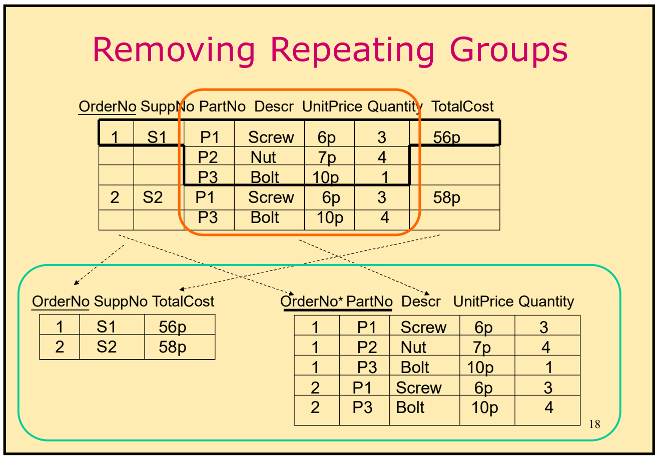


The Resulting Repeating Group

- PartNo, Descr, UnitPrice and Quantity are a repeating group in the table in the bottom half of the previous slide because they have multiple values for a value of the key OrderNo.
- PartNo, Descr, UnitPrice and Quantity belong to the same repeating group because they are related to each other.

17

17



Removing Repeating Groups

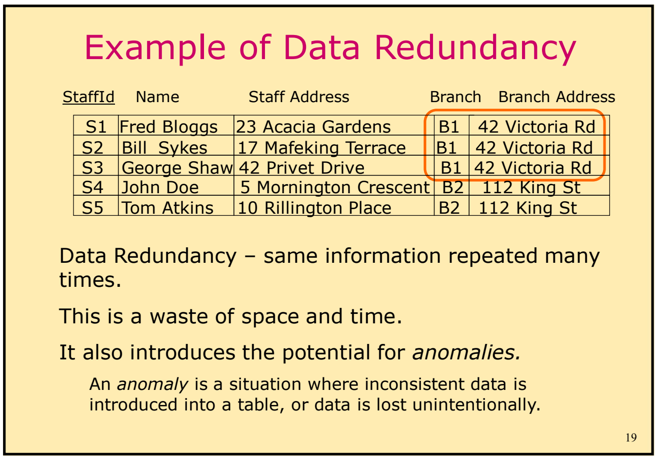
OrderNo	SuppNo	PartNo	Descr	UnitPrice	Quantity	TotalCost
1	S1	P1	Screw	6p	3	56p
		P2	Nut	7p	4	
		P3	Bolt	10p	1	
2	S2	P1	Screw	6p	3	58p
		P3	Bolt	10p	4	

OrderNo	SuppNo	TotalCost
1	S1	56p
2	S2	58p

OrderNo*	PartNo	Descr	UnitPrice	Quantity
1	P1	Screw	6p	3
1	P2	Nut	7p	4
1	P3	Bolt	10p	1
2	P1	Screw	6p	3
2	P3	Bolt	10p	4

18

18



Example of Data Redundancy

StaffId	Name	Staff Address	Branch	Branch Address
S1	Fred Bloggs	23 Acacia Gardens	B1	42 Victoria Rd
S2	Bill Sykes	17 Mafeking Terrace	B1	42 Victoria Rd
S3	George Shaw	42 Privet Drive	B1	42 Victoria Rd
S4	John Doe	5 Mornington Crescent	B2	112 King St
S5	Tom Atkins	10 Rillington Place	B2	112 King St

Data Redundancy – same information repeated many times.

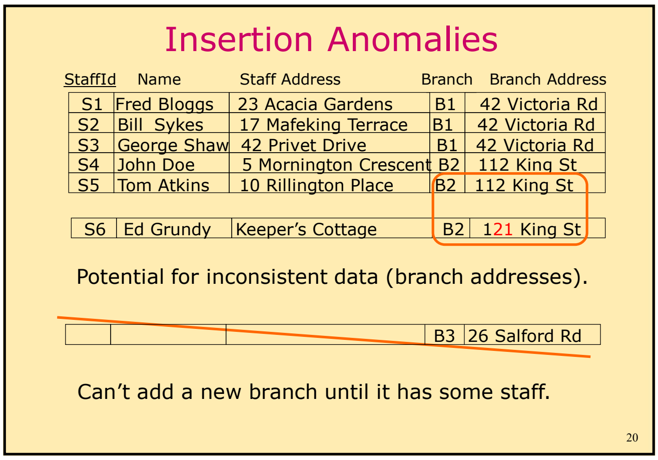
This is a waste of space and time.

It also introduces the potential for *anomalies.*

An *anomaly* is a situation where inconsistent data is introduced into a table, or data is lost unintentionally.

19

19



Insertion Anomalies

StaffId	Name	Staff Address	Branch	Branch Address
S1	Fred Bloggs	23 Acacia Gardens	B1	42 Victoria Rd
S2	Bill Sykes	17 Mafeking Terrace	B1	42 Victoria Rd
S3	George Shaw	42 Privet Drive	B1	42 Victoria Rd
S4	John Doe	5 Mornington Crescent	B2	112 King St
S5	Tom Atkins	10 Rillington Place	B2	112 King St

S6	Ed Grundy	Keeper's Cottage	B2	121 King St

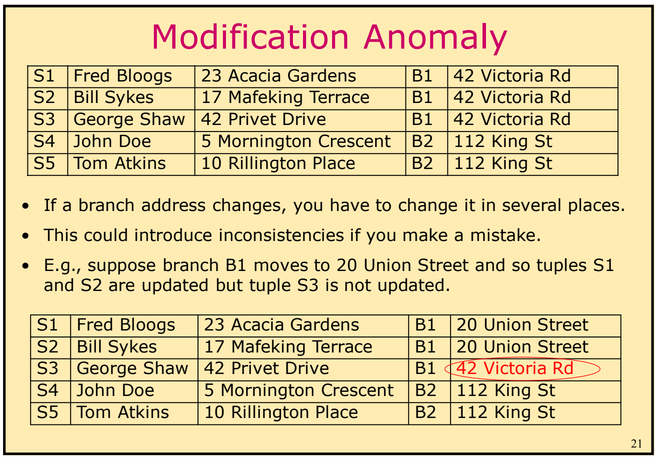
Potential for inconsistent data (branch addresses).

~~ ~~	~~ ~~	~~ ~~	~~B3~~	~~26 Salford Rd~~

Can't add a new branch until it has some staff.

20

20



Modification Anomaly

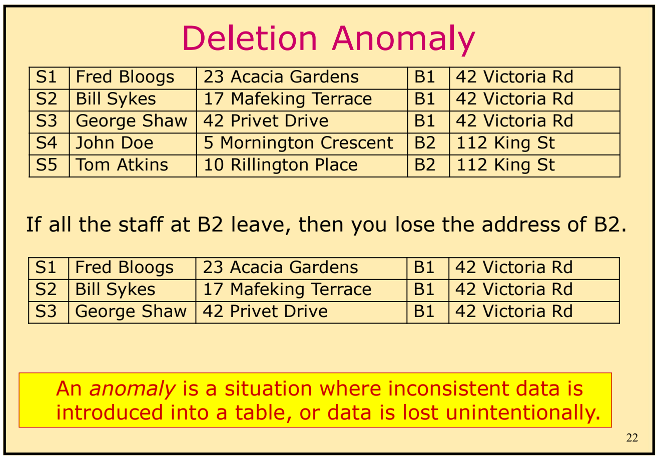
S1	Fred Bloggs	23 Acacia Gardens	B1	42 Victoria Rd
S2	Bill Sykes	17 Mafeking Terrace	B1	42 Victoria Rd
S3	George Shaw	42 Privet Drive	B1	42 Victoria Rd
S4	John Doe	5 Mornington Crescent	B2	112 King St
S5	Tom Atkins	10 Rillington Place	B2	112 King St

- If a branch address changes, you have to change it in several places.
- This could introduce inconsistencies if you make a mistake.
- E.g., suppose branch B1 moves to 20 Union Street and so tuples S1 and S2 are updated but tuple S3 is not updated.

S1	Fred Bloggs	23 Acacia Gardens	B1	20 Union Street
S2	Bill Sykes	17 Mafeking Terrace	B1	20 Union Street
S3	George Shaw	42 Privet Drive	B1	42 Victoria Rd
S4	John Doe	5 Mornington Crescent	B2	112 King St
S5	Tom Atkins	10 Rillington Place	B2	112 King St

21

21



Deletion Anomaly

S1	Fred Bloggs	23 Acacia Gardens	B1	42 Victoria Rd
S2	Bill Sykes	17 Mafeking Terrace	B1	42 Victoria Rd
S3	George Shaw	42 Privet Drive	B1	42 Victoria Rd
S4	John Doe	5 Mornington Crescent	B2	112 King St
S5	Tom Atkins	10 Rillington Place	B2	112 King St

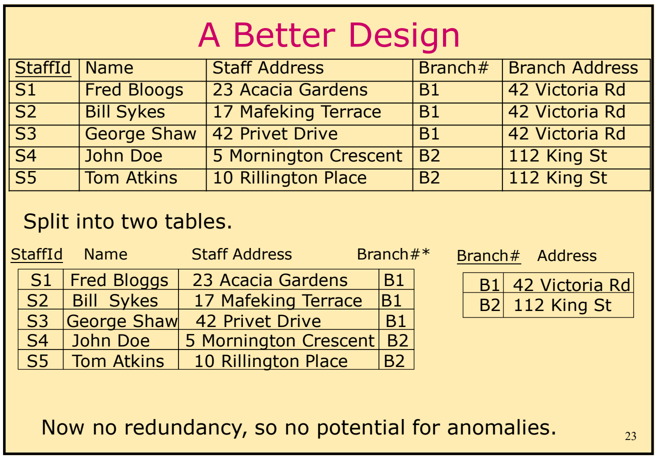
If all the staff at B2 leave, then you lose the address of B2.

S1	Fred Bloggs	23 Acacia Gardens	B1	42 Victoria Rd
S2	Bill Sykes	17 Mafeking Terrace	B1	42 Victoria Rd
S3	George Shaw	42 Privet Drive	B1	42 Victoria Rd

An *anomaly* is a situation where inconsistent data is introduced into a table, or data is lost unintentionally.

22

22



A Better Design

StaffId	Name	Staff Address	Branch#	Branch Address
S1	Fred Bloggs	23 Acacia Gardens	B1	42 Victoria Rd
S2	Bill Sykes	17 Mafeking Terrace	B1	42 Victoria Rd
S3	George Shaw	42 Privet Drive	B1	42 Victoria Rd
S4	John Doe	5 Mornington Crescent	B2	112 King St
S5	Tom Atkins	10 Rillington Place	B2	112 King St

Split into two tables.

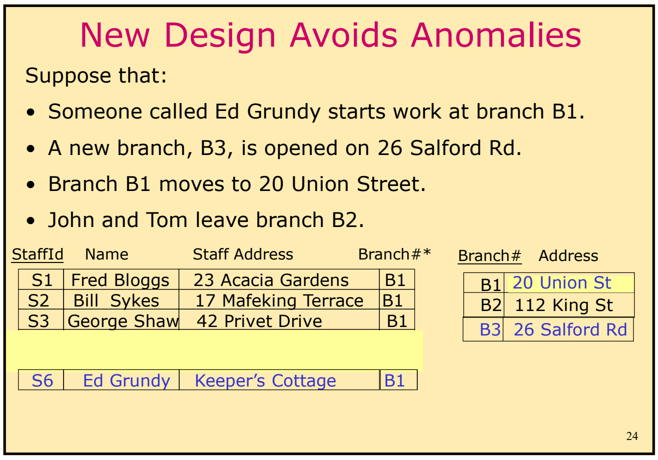
StaffId	Name	Staff Address	Branch#*
S1	Fred Bloggs	23 Acacia Gardens	B1
S2	Bill Sykes	17 Mafeking Terrace	B1
S3	George Shaw	42 Privet Drive	B1
S4	John Doe	5 Mornington Crescent	B2
S5	Tom Atkins	10 Rillington Place	B2

Branch#	Address
B1	42 Victoria Rd
B2	112 King St

Now no redundancy, so no potential for anomalies.

23

23



New Design Avoids Anomalies

Suppose that:

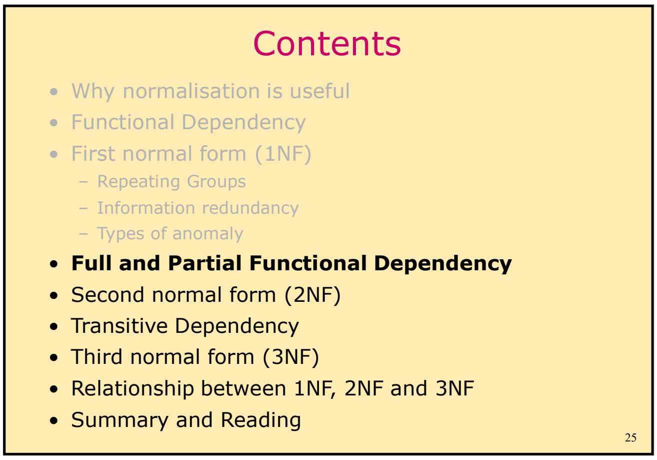
- Someone called Ed Grundy starts work at branch B1.
- A new branch, B3, is opened on 26 Salford Rd.
- Branch B1 moves to 20 Union Street.
- John and Tom leave branch B2.

StaffId	Name	Staff Address	Branch#*
S1	Fred Bloggs	23 Acacia Gardens	B1
S2	Bill Sykes	17 Mafeking Terrace	B1
S3	George Shaw	42 Privet Drive	B1
S6	Ed Grundy	Keeper's Cottage	B1

Branch#	Address
B1	20 Union St
B2	112 King St
B3	26 Salford Rd

24

24

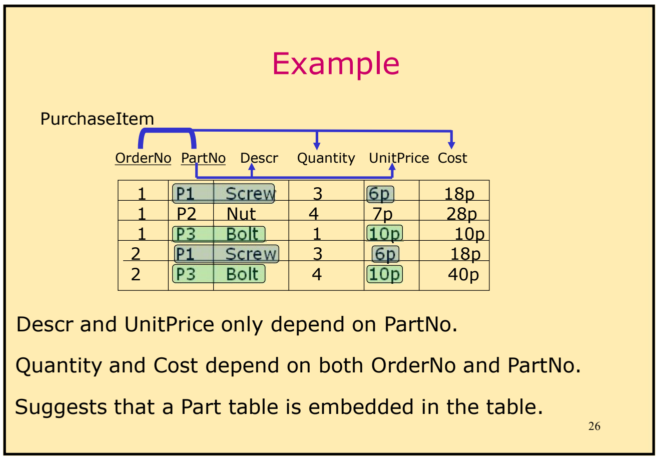


Contents

- Why normalisation is useful
- Functional Dependency
- First normal form (1NF)
  - Repeating Groups
  - Information redundancy
  - Types of anomaly
- **Full and Partial Functional Dependency**
- Second normal form (2NF)
- Transitive Dependency
- Third normal form (3NF)
- Relationship between 1NF, 2NF and 3NF
- Summary and Reading

25

25



Example

PurchaseItem

OrderNo
PartNo
Descr
Quantity
UnitPrice
Cost

1	P1	Screw	3	6p	18p
1	P2	Nut	4	7p	28p
1	P3	Bolt	1	10p	10p
2	P1	Screw	3	6p	18p
2	P3	Bolt	4	10p	40p

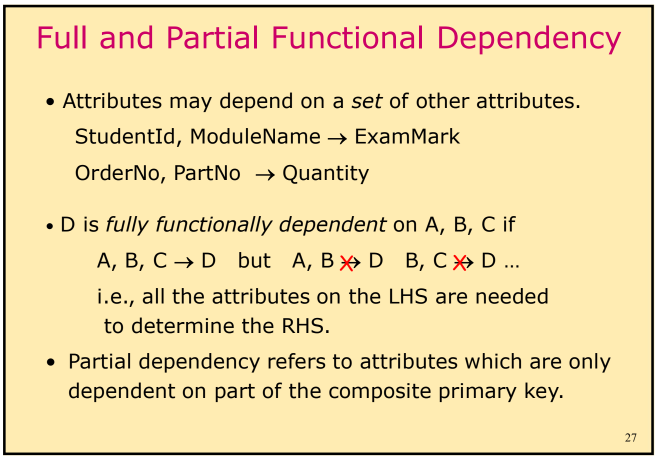
Descr and UnitPrice only depend on PartNo.

Quantity and Cost depend on both OrderNo and PartNo.

Suggests that a Part table is embedded in the table.

26

26



Full and Partial Functional Dependency

- Attributes may depend on a *set* of other attributes.

  StudentId, ModuleName → ExamMark

  OrderNo, PartNo → Quantity

- D is *fully functionally dependent* on A, B, C if

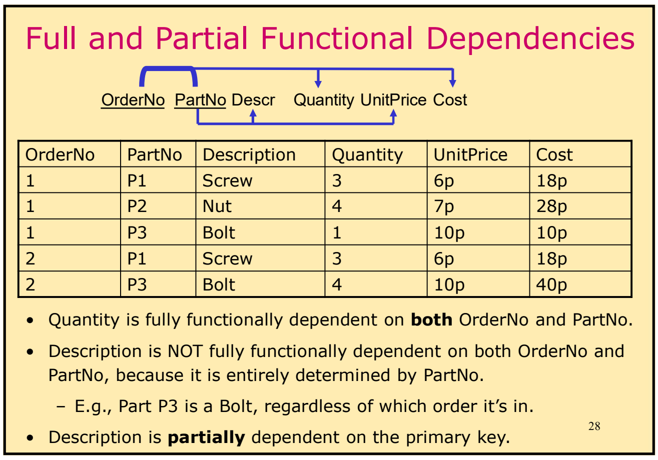
  A, B, C → D but A, B ↛ D B, C ↛ D ...

  i.e., all the attributes on the LHS are needed to determine the RHS.

- Partial dependency refers to attributes which are only dependent on part of the composite primary key.

27

27



Full and Partial Functional Dependencies

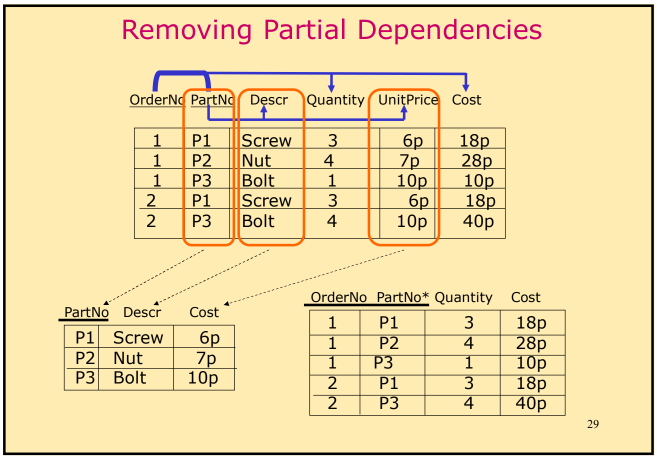
OrderNo
PartNo
Descr
Quantity
UnitPrice
Cost

OrderNo	PartNo	Description	Quantity	UnitPrice	Cost
1	P1	Screw	3	6p	18p
1	P2	Nut	4	7p	28p
1	P3	Bolt	1	10p	10p
2	P1	Screw	3	6p	18p
2	P3	Bolt	4	10p	40p

- Quantity is fully functionally dependent on **both** OrderNo and PartNo.
- Description is NOT fully functionally dependent on both OrderNo and PartNo, because it is entirely determined by PartNo.
  - E.g., Part P3 is a Bolt, regardless of which order it's in.
- Description is **partially** dependent on the primary key.

28

28



Removing Partial Dependencies

OrderNo
PartNo
Descr
Quantity
UnitPrice
Cost

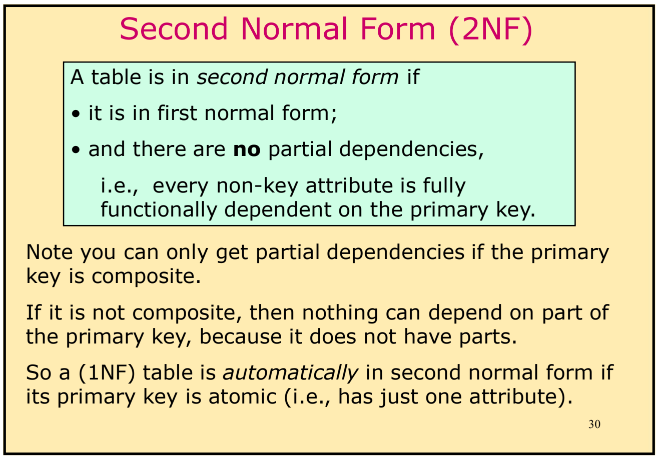
1	P1	Screw	3	6p	18p
1	P2	Nut	4	7p	28p
1	P3	Bolt	1	10p	10p
2	P1	Screw	3	6p	18p
2	P3	Bolt	4	10p	40p

PartNo	Descr	Cost
P1	Screw	6p
P2	Nut	7p
P3	Bolt	10p

OrderNo	PartNo*	Quantity	Cost
1	P1	3	18p
1	P2	4	28p
1	P3	1	10p
2	P1	3	18p
2	P3	4	40p

29

29



Second Normal Form (2NF)

A table is in *second normal form* if

- it is in first normal form;
- and there are **no** partial dependencies,

  i.e., every non-key attribute is fully functionally dependent on the primary key.

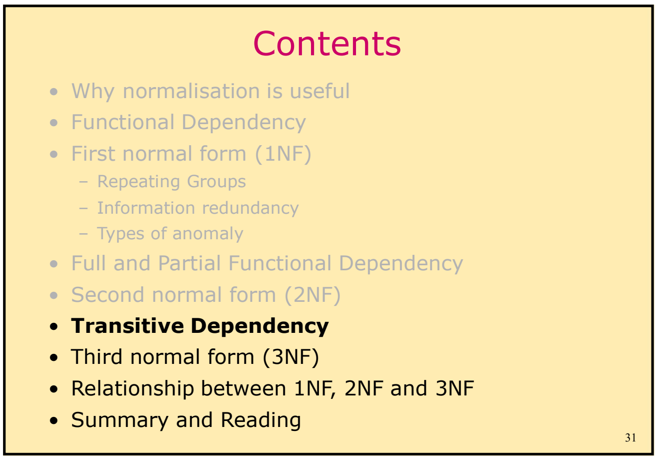
Note you can only get partial dependencies if the primary key is composite.

If it is not composite, then nothing can depend on part of the primary key, because it does not have parts.

So a (1NF) table is *automatically* in second normal form if its primary key is atomic (i.e., has just one attribute).

30

30

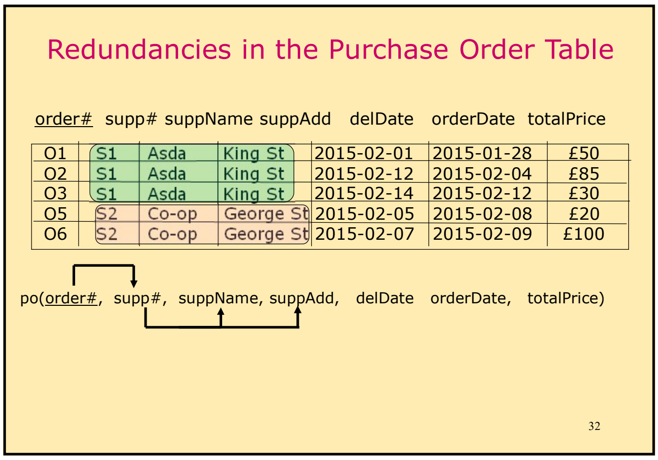


Contents

- Why normalisation is useful
- Functional Dependency
- First normal form (1NF)
  - Repeating Groups
  - Information redundancy
  - Types of anomaly
- Full and Partial Functional Dependency
- Second normal form (2NF)
- **Transitive Dependency**
- Third normal form (3NF)
- Relationship between 1NF, 2NF and 3NF
- Summary and Reading

31

31



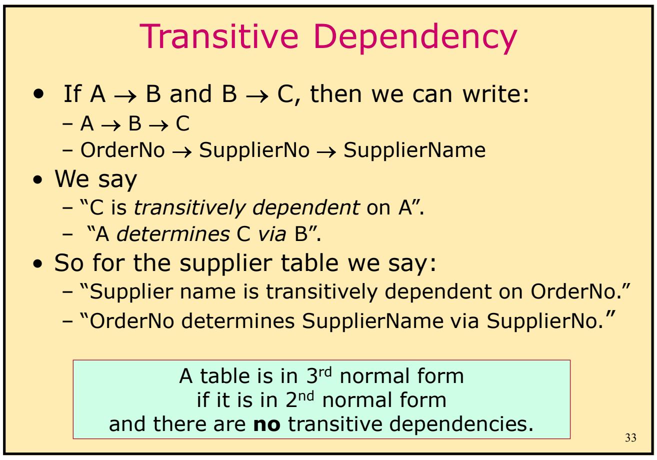
Redundancies in the Purchase Order Table

<u>order#</u>	supp#	suppName	suppAdd	delDate	orderDate	totalPrice
O1	S1	Asda	King St	2015-02-01	2015-01-28	£50
O2	S1	Asda	King St	2015-02-12	2015-02-04	£85
O3	S1	Asda	King St	2015-02-14	2015-02-12	£30
O5	S2	Co-op	George St	2015-02-05	2015-02-08	£20
O6	S2	Co-op	George St	2015-02-07	2015-02-09	£100

po(order#, supp#, suppName, suppAdd, delDate orderDate, totalPrice)

32

32



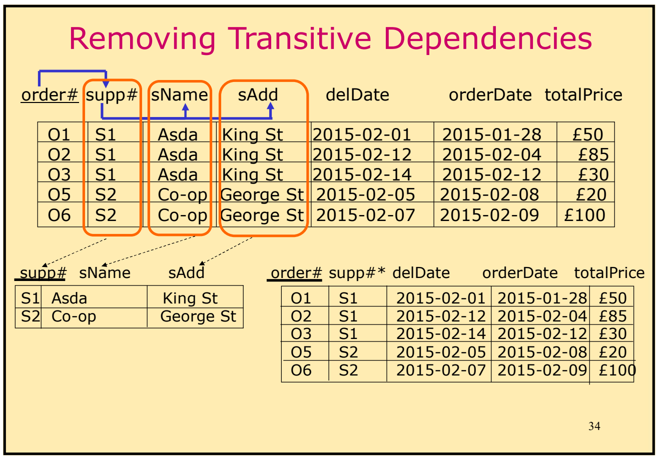
Transitive Dependency

- If A → B and B → C, then we can write:
  - A → B → C
  - OrderNo → SupplierNo → SupplierName
- We say
  - "C is *transitively dependent* on A".
  - "A *determines* C *via* B".
- So for the supplier table we say:
  - "Supplier name is transitively dependent on OrderNo."
  - "OrderNo determines SupplierName via SupplierNo."

A table is in 3rd normal form
if it is in 2nd normal form
and there are **no** transitive dependencies.

33

33



Removing Transitive Dependencies

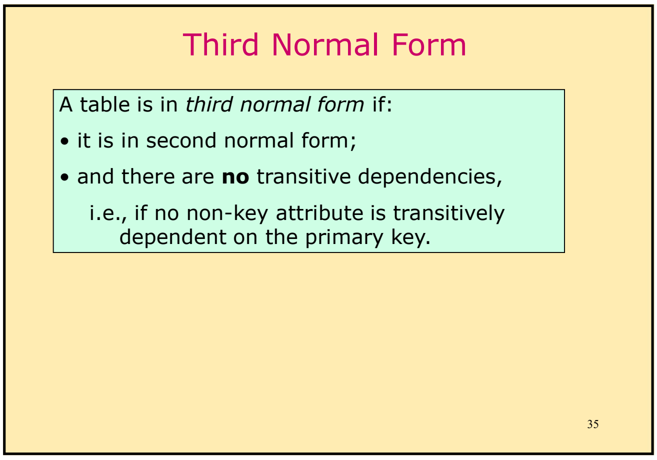
<u>order#</u>	supp#	sName	sAdd	delDate	orderDate	totalPrice
O1	S1	Asda	King St	2015-02-01	2015-01-28	£50
O2	S1	Asda	King St	2015-02-12	2015-02-04	£85
O3	S1	Asda	King St	2015-02-14	2015-02-12	£30
O5	S2	Co-op	George St	2015-02-05	2015-02-08	£20
O6	S2	Co-op	George St	2015-02-07	2015-02-09	£100

<u>supp#</u>	sName	sAdd
S1	Asda	King St
S2	Co-op	George St

<u>order#</u>	supp#*	delDate	orderDate	totalPrice
O1	S1	2015-02-01	2015-01-28	£50
O2	S1	2015-02-12	2015-02-04	£85
O3	S1	2015-02-14	2015-02-12	£30
O5	S2	2015-02-05	2015-02-08	£20
O6	S2	2015-02-07	2015-02-09	£100

34

34



Third Normal Form

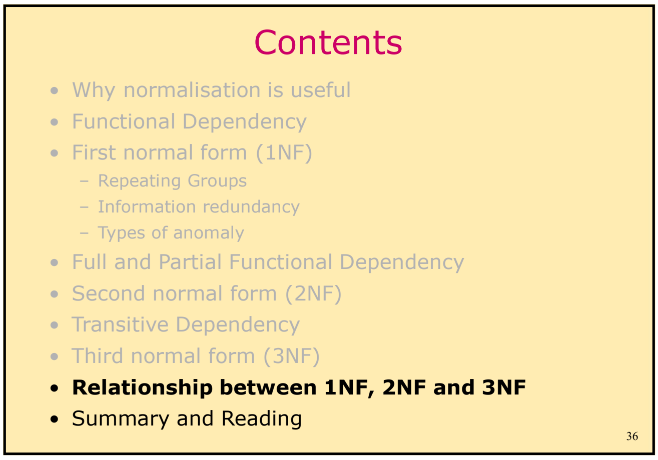
A table is in *third normal form* if:

- it is in second normal form;
- and there are **no** transitive dependencies,

i.e., if no non-key attribute is transitively dependent on the primary key.

35

35

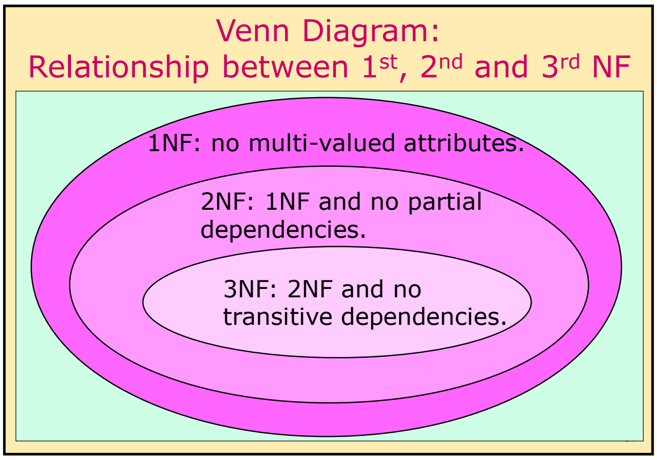


Contents

- Why normalisation is useful
- Functional Dependency
- First normal form (1NF)
  - Repeating Groups
  - Information redundancy
  - Types of anomaly
- Full and Partial Functional Dependency
- Second normal form (2NF)
- Transitive Dependency
- Third normal form (3NF)
- **Relationship between 1NF, 2NF and 3NF**
- Summary and Reading

36

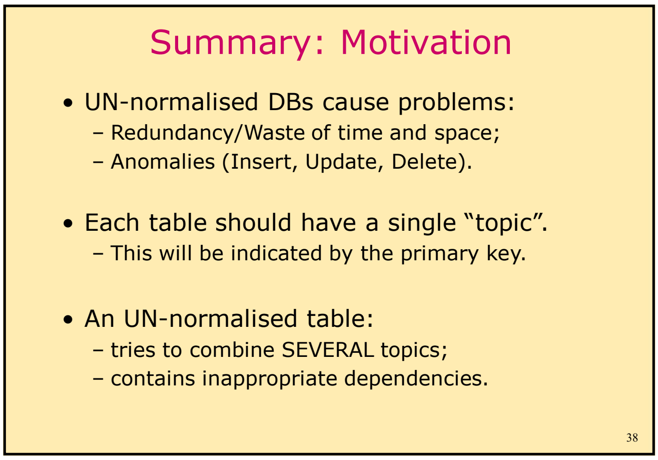
36



Venn Diagram: Relationship between 1st, 2nd and 3rd NF

1NF: no multi-valued attributes.
2NF: 1NF and no partial dependencies.
3NF: 2NF and no transitive dependencies.

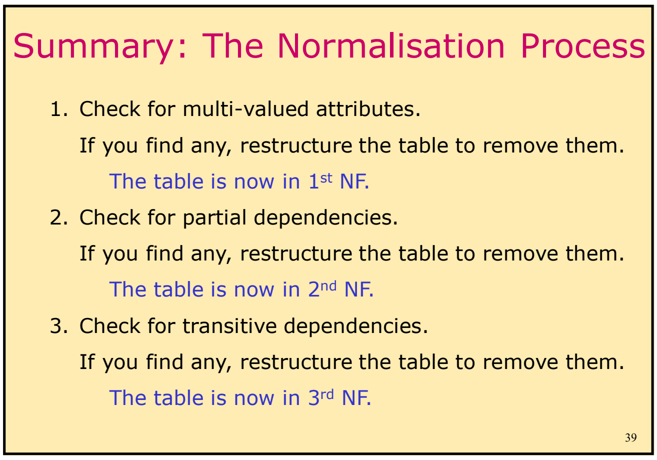
37



Summary: Motivation

- UN-normalised DBs cause problems:
  - Redundancy/Waste of time and space;
  - Anomalies (Insert, Update, Delete).
- Each table should have a single “topic”.
  - This will be indicated by the primary key.
- An UN-normalised table:
  - tries to combine SEVERAL topics;
  - contains inappropriate dependencies.

38



Summary: The Normalisation Process

1. Check for multi-valued attributes.

   If you find any, restructure the table to remove them.

   The table is now in 1st NF.
2. Check for partial dependencies.

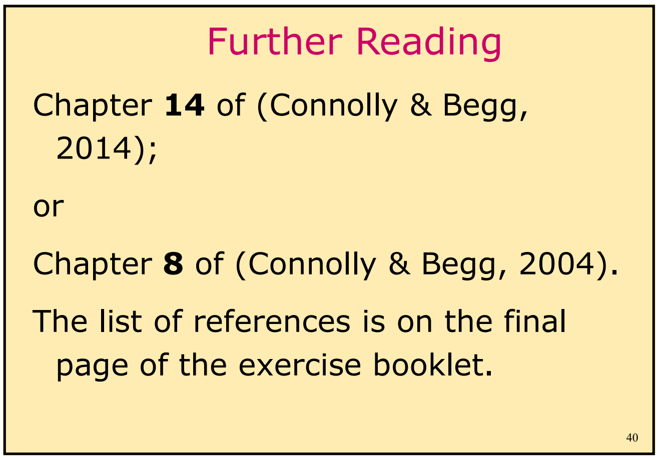
   If you find any, restructure the table to remove them.

   The table is now in 2nd NF.
3. Check for transitive dependencies.

   If you find any, restructure the table to remove them.

   The table is now in 3rd NF.

39



Further Reading

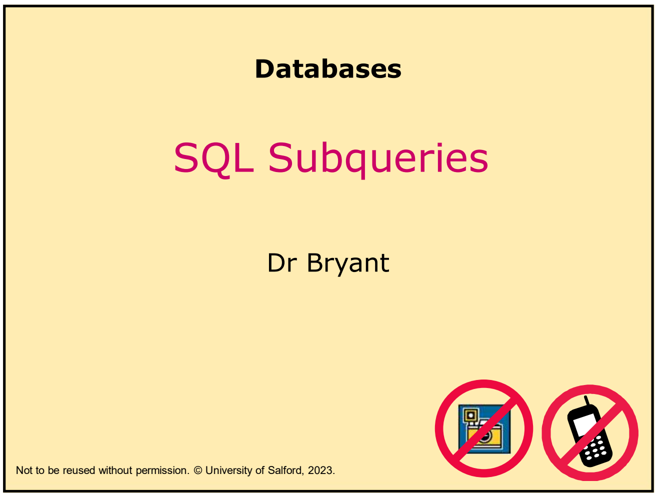
Chapter **14** of (Connolly & Begg, 2014);

or

Chapter **8** of (Connolly & Begg, 2004).

The list of references is on the final page of the exercise booklet.

40

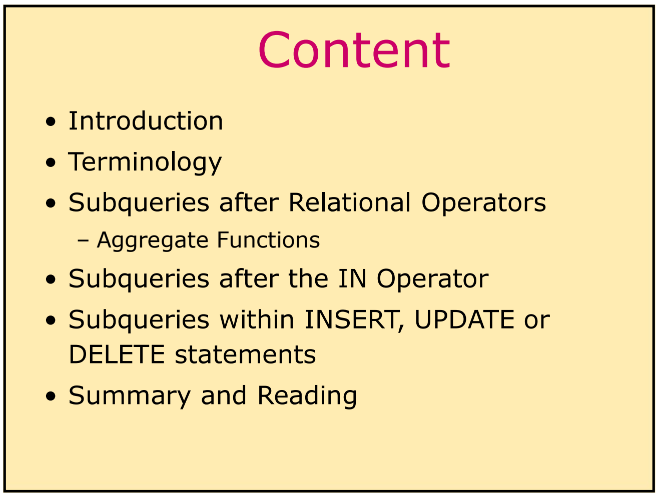


Databases

SQL Subqueries

Dr Bryant

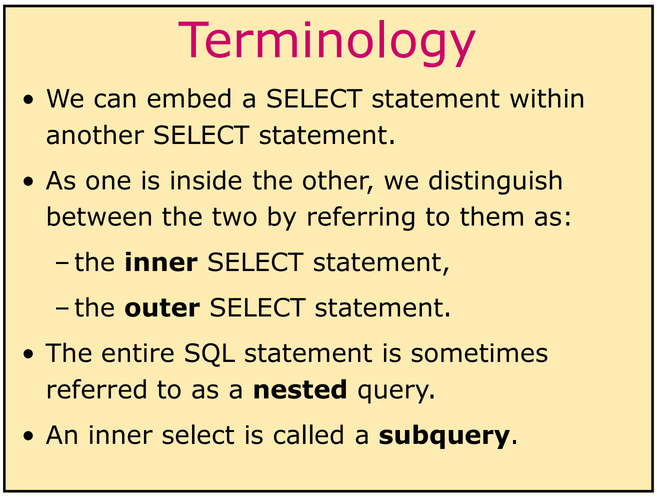
1



Content

- Introduction
- Terminology
- Subqueries after Relational Operators
  - Aggregate Functions
- Subqueries after the IN Operator
- Subqueries within INSERT, UPDATE or DELETE statements
- Summary and Reading

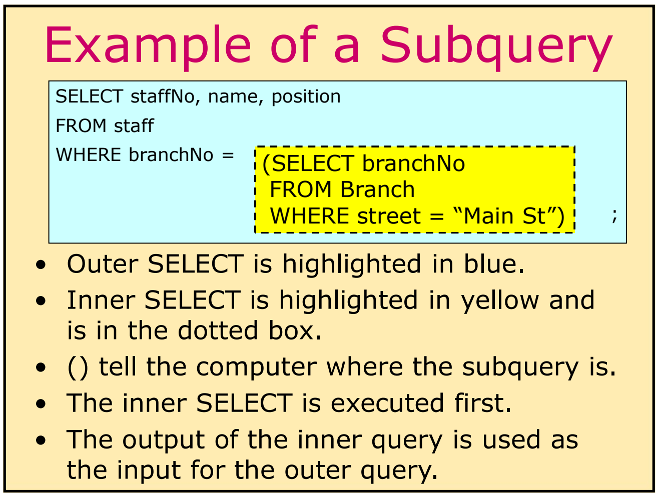
2



Terminology

- We can embed a SELECT statement within another SELECT statement.
- As one is inside the other, we distinguish between the two by referring to them as:
  - the **inner** SELECT statement,
  - the **outer** SELECT statement.
- The entire SQL statement is sometimes referred to as a **nested** query.
- An inner select is called a **subquery**.

3

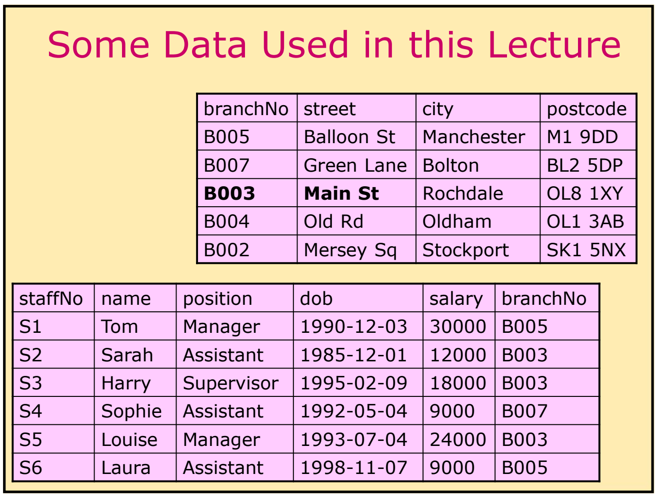


Example of a Subquery

```
SELECT staffNo, name, position
FROM staff
WHERE branchNo =  (SELECT branchNo
                   FROM Branch
                   WHERE street = "Main St")  ;
```

- Outer SELECT is highlighted in blue.
- Inner SELECT is highlighted in yellow and is in the dotted box.
- () tell the computer where the subquery is.
- The inner SELECT is executed first.
- The output of the inner query is used as the input for the outer query.

4

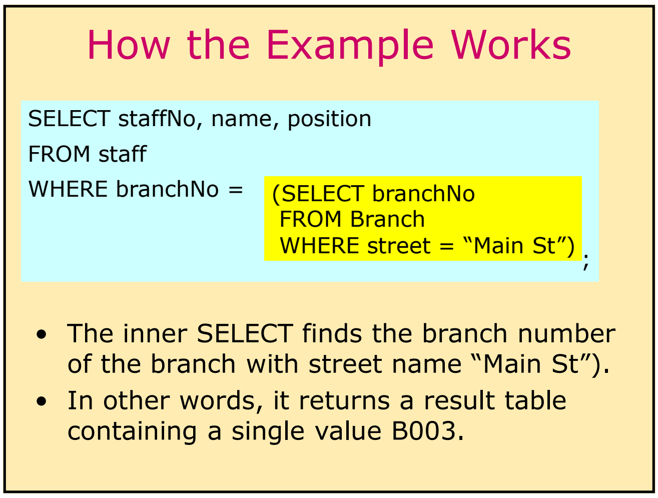


Some Data Used in this Lecture

branchNo	street	city	postcode
B005	Balloon St	Manchester	M1 9DD
B007	Green Lane	Bolton	BL2 5DP
B003	**Main St**	Rochdale	OL8 1XY
B004	Old Rd	Oldham	OL1 3AB
B002	Mersey Sq	Stockport	SK1 5NX

staffNo	name	position	dob	salary	branchNo
S1	Tom	Manager	1990-12-03	30000	B005
S2	Sarah	Assistant	1985-12-01	12000	B003
S3	Harry	Supervisor	1995-02-09	18000	B003
S4	Sophie	Assistant	1992-05-04	9000	B007
S5	Louise	Manager	1993-07-04	24000	B003
S6	Laura	Assistant	1998-11-07	9000	B005

5



How the Example Works

```
SELECT staffNo, name, position
FROM staff
WHERE branchNo =  (SELECT branchNo
                   FROM Branch
                   WHERE street = "Main St");
```

- The inner SELECT finds the branch number of the branch with street name "Main St").
- In other words, it returns a result table containing a single value B003.

6

Result of the Subquery

staffNo	name	position	dob	salary	branchNo
S1	Tom	Manager	1990-12-03	30000	B005
S2	Sarah	Assistant	1985-12-01	12000	**B003**
S3	Harry	Supervisor	1995-02-09	18000	**B003**
S4	Sophie	Assistant	1992-05-04	9000	B007
S5	Louise	Manager	1993-07-04	24000	**B003**
S6	Laura	Assistant	1998-11-07	9000	B005

```
SELECT staffNo, name, position
FROM staff
WHERE branchNo =    (SELECT branchNo
                     FROM Branch
                     WHERE street = "Main St")          ;
```

The outer query returns

staffNo	name	position
S2	Sarah	Assistant
S3	Harry	Supervisor
S5	Louise	Manager

7

Subqueries after Relational Operators

- A subquery can be used immediately following a relational operator in a WHERE clause or a HAVING clause.

  =   <   >   <=   >=   <>

- The subquery must appear on the right-hand side of the comparison.
- The subquery must return one value; otherwise the DBMS will raise an error.
- The value must be of a comparable data type.

8

Data Used in Examples

Drivers and the number of penalty points on their licence.

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

9

Recap: Aggregate Functions Example

Calculate the number of drivers, the total number of points and the average number of points.

driverID	name	address	points
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

```
SELECT COUNT(driverID), SUM(points), AVG(points)
FROM Drivers;
```

↓

COUNT(driverID)	SUM(points)	AVG(points)
5	28	5.6

10

Subquery with an Aggregate Function

```
SELECT staffNo, name, position,
  salary – (SELECT AVG(salary) FROM Staff) AS salDIF
FROM staff
WHERE salary > (SELECT AVG(salary) FROM Staff) ;
```

- N.B. cannot write

  WHERE salary > AVG(salary)

  Aggregate functions cannot be used in a where clause.

- The subquery finds the average salary.
- In other words, it returns a result table containing a single value £17,000.

11

Result of the Subquery

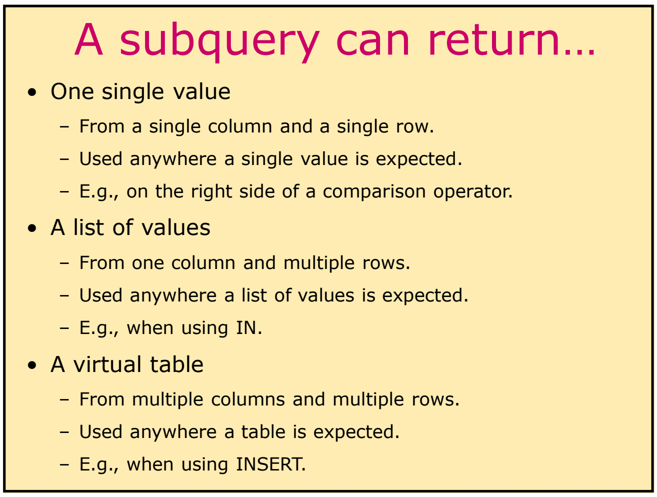
staffNo	name	position	dob	salary	branchNo
S1	Tom	Manager	1990-12-03	30000	B005
S2	Sarah	Assistant	1985-12-01	12000	B003
S3	Harry	Supervisor	1995-02-09	18000	B003
S4	Sophie	Assistant	1992-05-04	9000	B007
S5	Louise	Manager	1993-07-04	24000	B003
S6	Laura	Assistant	1998-11-07	9000	B005

```
SELECT staffNo, name, position,
  salary – (SELECT AVG(salary) FROM Staff) AS salDIF
FROM staff
WHERE salary > (SELECT AVG(salary) FROM Staff) ;
```

The outer query returns

staffNo	name	position	salDif
S1	Tom	Assistant	13000.00
S3	Harry	Supervisor	1000.00
S5	Louise	Manager	7000.00

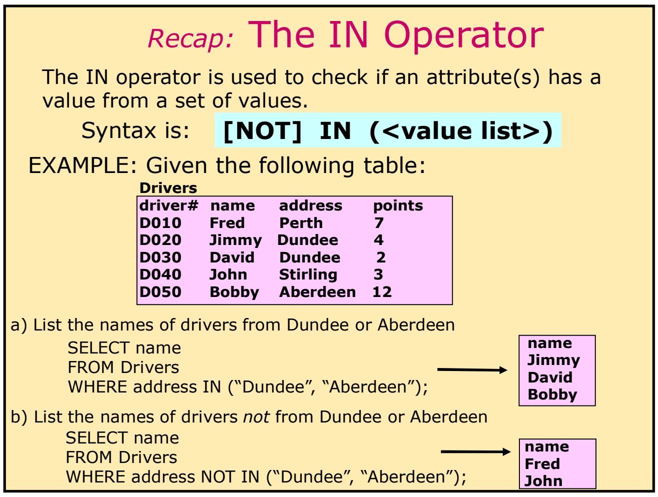
12



A subquery can return…

- One single value
  - From a single column and a single row.
  - Used anywhere a single value is expected.
  - E.g., on the right side of a comparison operator.
- A list of values
  - From one column and multiple rows.
  - Used anywhere a list of values is expected.
  - E.g., when using IN.
- A virtual table
  - From multiple columns and multiple rows.
  - Used anywhere a table is expected.
  - E.g., when using INSERT.

13



Recap: The IN Operator

The IN operator is used to check if an attribute(s) has a value from a set of values.

Syntax is: **[NOT] IN (<value list>)**

EXAMPLE: Given the following table:

Drivers

driver#	**name**	**address**	**points**
D010	**Fred**	**Perth**	**7**
D020	**Jimmy**	**Dundee**	**4**
D030	**David**	**Dundee**	**2**
D040	**John**	**Stirling**	**3**
D050	**Bobby**	**Aberdeen**	**12**

a) List the names of drivers from Dundee or Aberdeen

```
SELECT name
FROM Drivers
WHERE address IN (“Dundee”, “Aberdeen”);
```

→

name
Jimmy
David
Bobby

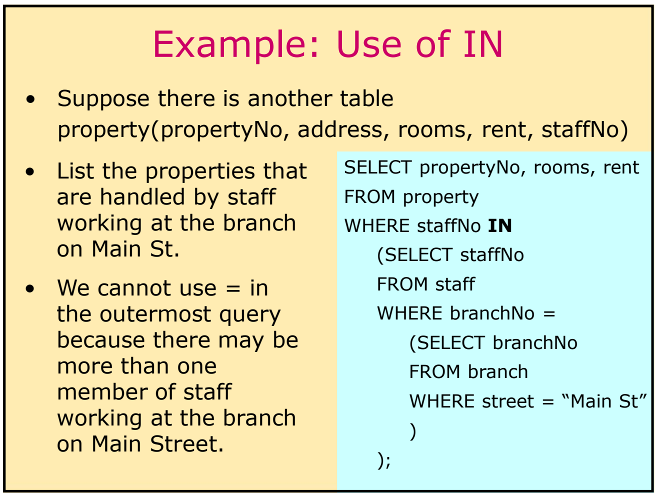
b) List the names of drivers *not* from Dundee or Aberdeen

```
SELECT name
FROM Drivers
WHERE address NOT IN (“Dundee”, “Aberdeen”);
```

→

name
Fred
John

14

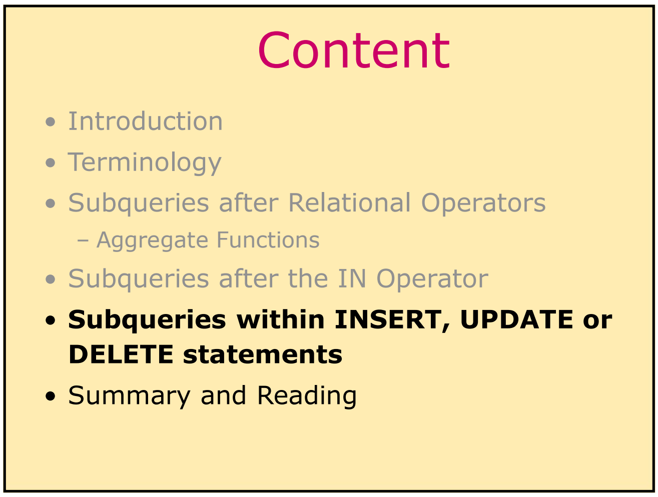


Example: Use of IN

- Suppose there is another table property(propertyNo, address, rooms, rent, staffNo)
- List the properties that are handled by staff working at the branch on Main St.
- We cannot use = in the outermost query because there may be more than one member of staff working at the branch on Main Street.

```
SELECT propertyNo, rooms, rent
FROM property
WHERE staffNo IN
    (SELECT staffNo
    FROM staff
    WHERE branchNo =
        (SELECT branchNo
        FROM branch
        WHERE street = “Main St”
        )
    );
```

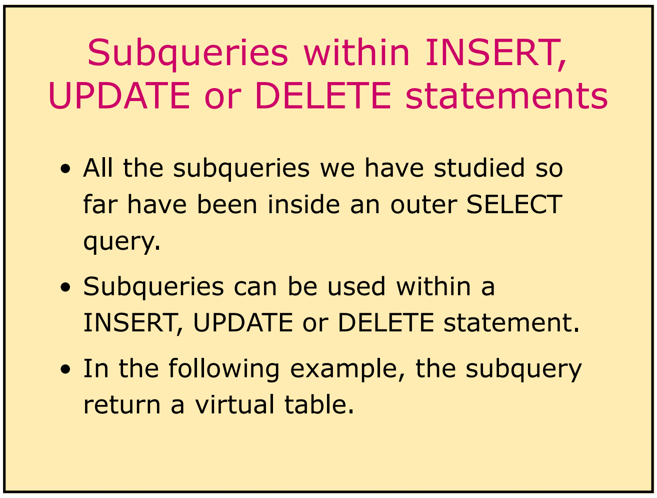
15



Content

- Introduction
- Terminology
- Subqueries after Relational Operators
  - Aggregate Functions
- Subqueries after the IN Operator
- **Subqueries within INSERT, UPDATE or DELETE statements**
- Summary and Reading

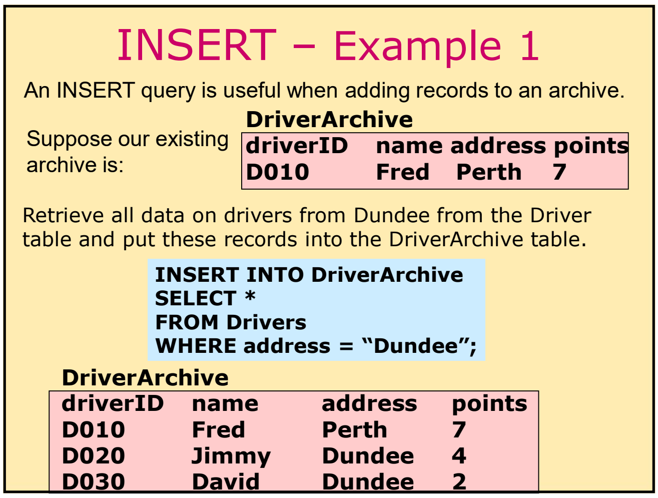
16



Subqueries within INSERT, UPDATE or DELETE statements

- All the subqueries we have studied so far have been inside an outer SELECT query.
- Subqueries can be used within a INSERT, UPDATE or DELETE statement.
- In the following example, the subquery return a virtual table.

17



INSERT – Example 1

An INSERT query is useful when adding records to an archive.

Suppose our existing archive is:

DriverArchive

driverID	**name**	**address**	**points**
D010	**Fred**	**Perth**	**7**

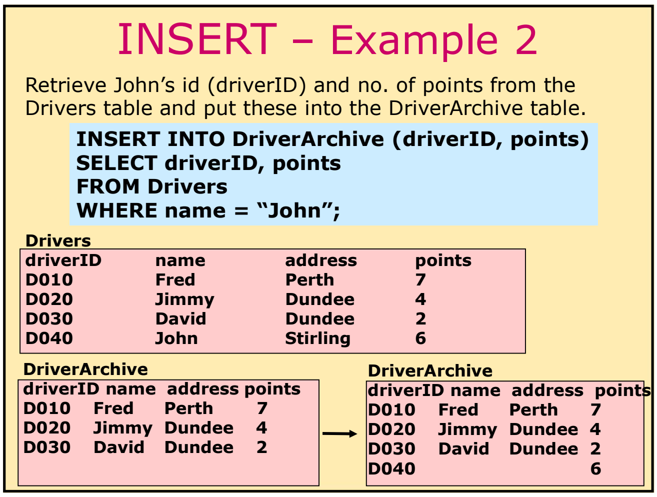
Retrieve all data on drivers from Dundee from the Driver table and put these records into the DriverArchive table.

```
INSERT INTO DriverArchive
SELECT *
FROM Drivers
WHERE address = “Dundee”;
```

DriverArchive

driverID	**name**	**address**	**points**
D010	**Fred**	**Perth**	**7**
D020	**Jimmy**	**Dundee**	**4**
D030	**David**	**Dundee**	**2**

18



INSERT – Example 2

Retrieve John's id (driverID) and no. of points from the Drivers table and put these into the DriverArchive table.

```
INSERT INTO DriverArchive (driverID, points)
SELECT driverID, points
FROM Drivers
WHERE name = "John";
```

Drivers

driverID	**name**	**address**	**points**
D010	**Fred**	**Perth**	**7**
D020	**Jimmy**	**Dundee**	**4**
D030	**David**	**Dundee**	**2**
D040	**John**	**Stirling**	**6**

DriverArchive

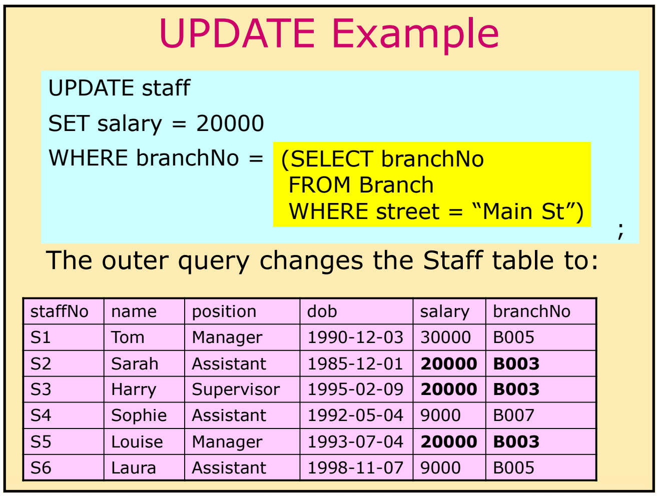
driverID	**name**	**address**	**points**
D010	**Fred**	**Perth**	**7**
D020	**Jimmy**	**Dundee**	**4**
D030	**David**	**Dundee**	**2**

→

DriverArchive

driverID	**name**	**address**	**points**
D010	**Fred**	**Perth**	**7**
D020	**Jimmy**	**Dundee**	**4**
D030	**David**	**Dundee**	**2**
D040			**6**

19



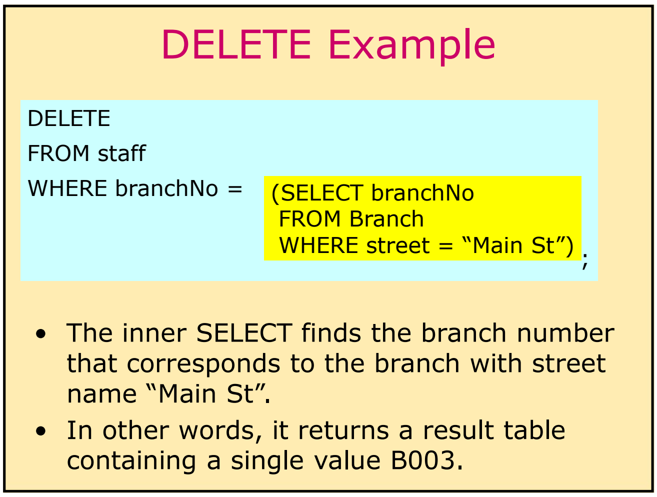
UPDATE Example

```
UPDATE staff
SET salary = 20000
WHERE branchNo = (SELECT branchNo
                  FROM Branch
                  WHERE street = "Main St")
                                            ;
```

The outer query changes the Staff table to:

staffNo	name	position	dob	salary	branchNo
S1	Tom	Manager	1990-12-03	30000	B005
S2	Sarah	Assistant	1985-12-01	**20000**	**B003**
S3	Harry	Supervisor	1995-02-09	**20000**	**B003**
S4	Sophie	Assistant	1992-05-04	9000	B007
S5	Louise	Manager	1993-07-04	**20000**	**B003**
S6	Laura	Assistant	1998-11-07	9000	B005

20

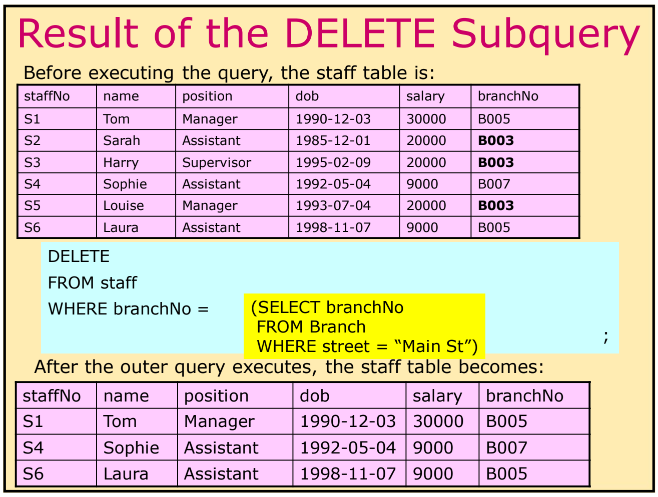


DELETE Example

```
DELETE
FROM staff
WHERE branchNo = (SELECT branchNo
                  FROM Branch
                  WHERE street = "Main St");
```

- The inner SELECT finds the branch number that corresponds to the branch with street name "Main St".
- In other words, it returns a result table containing a single value B003.

21



Result of the DELETE Subquery

Before executing the query, the staff table is:

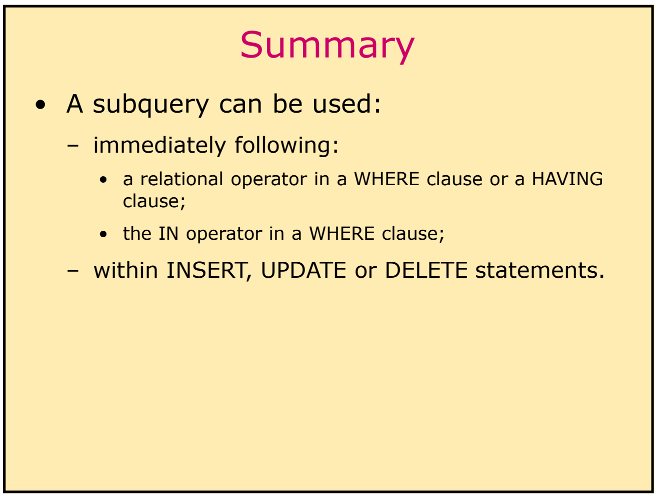
staffNo	name	position	dob	salary	branchNo
S1	Tom	Manager	1990-12-03	30000	B005
S2	Sarah	Assistant	1985-12-01	20000	**B003**
S3	Harry	Supervisor	1995-02-09	20000	**B003**
S4	Sophie	Assistant	1992-05-04	9000	B007
S5	Louise	Manager	1993-07-04	20000	**B003**
S6	Laura	Assistant	1998-11-07	9000	B005

```
DELETE
FROM staff
WHERE branchNo = (SELECT branchNo
                  FROM Branch
                  WHERE street = "Main St")
                                            ;
```

After the outer query executes, the staff table becomes:

staffNo	name	position	dob	salary	branchNo
S1	Tom	Manager	1990-12-03	30000	B005
S4	Sophie	Assistant	1992-05-04	9000	B007
S6	Laura	Assistant	1998-11-07	9000	B005

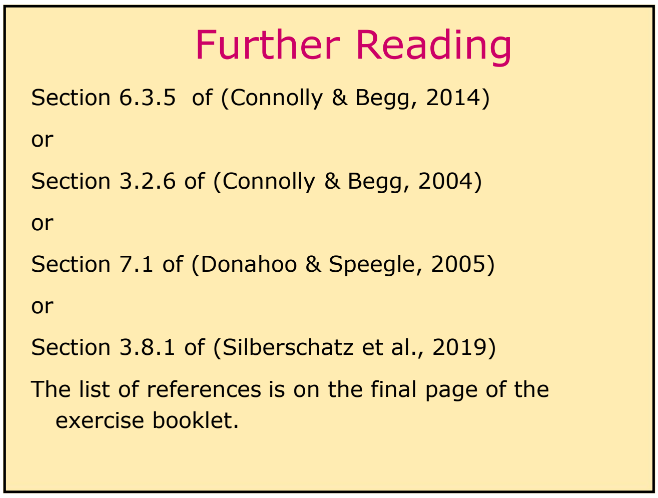
22



Summary

- A subquery can be used:
  - – immediately following:
    - a relational operator in a WHERE clause or a HAVING clause;
    - the IN operator in a WHERE clause;
  - – within INSERT, UPDATE or DELETE statements.

23



Further Reading

Section 6.3.5 of (Connolly & Begg, 2014)

or

Section 3.2.6 of (Connolly & Begg, 2004)

or

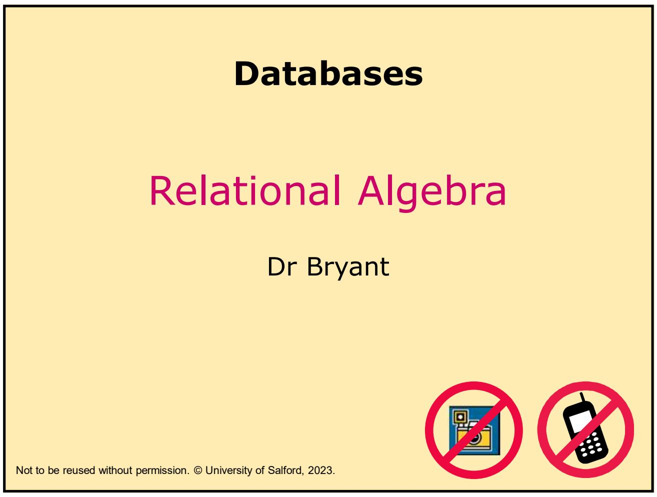
Section 7.1 of (Donahoo & Speegle, 2005)

or

Section 3.8.1 of (Silberschatz et al., 2019)

The list of references is on the final page of the exercise booklet.

24

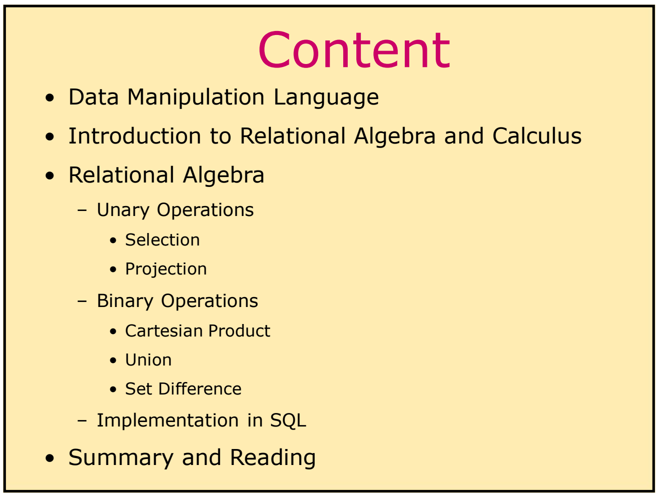


Databases

Relational Algebra

Dr Bryant

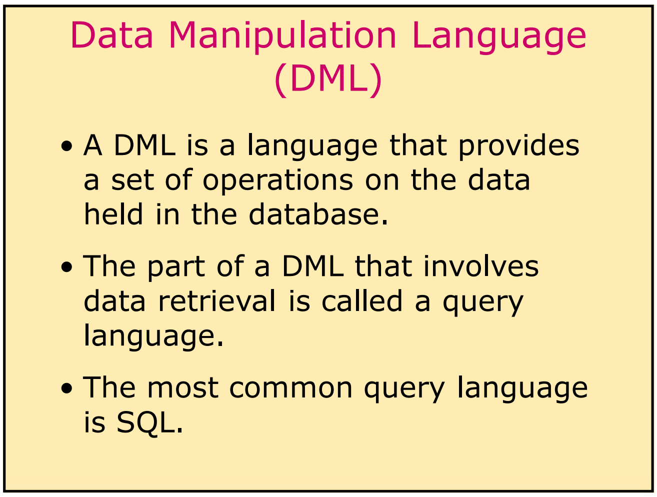
1



Content

- Data Manipulation Language
- Introduction to Relational Algebra and Calculus
- Relational Algebra
  - Unary Operations
    - Selection
    - Projection
  - Binary Operations
    - Cartesian Product
    - Union
    - Set Difference
  - Implementation in SQL
- Summary and Reading

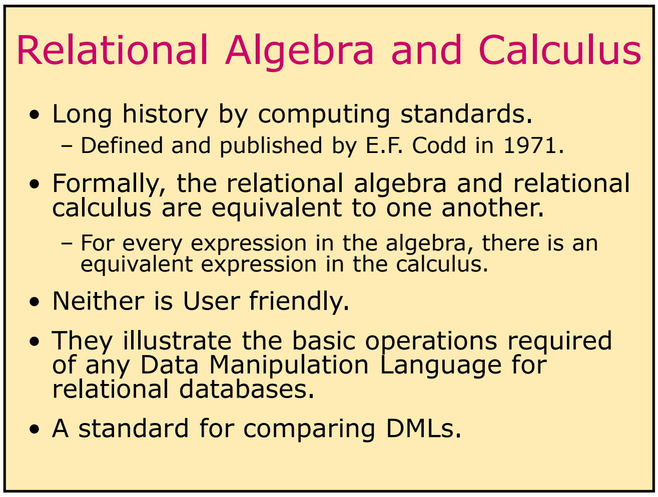
2



Data Manipulation Language (DML)

- A DML is a language that provides a set of operations on the data held in the database.
- The part of a DML that involves data retrieval is called a query language.
- The most common query language is SQL.

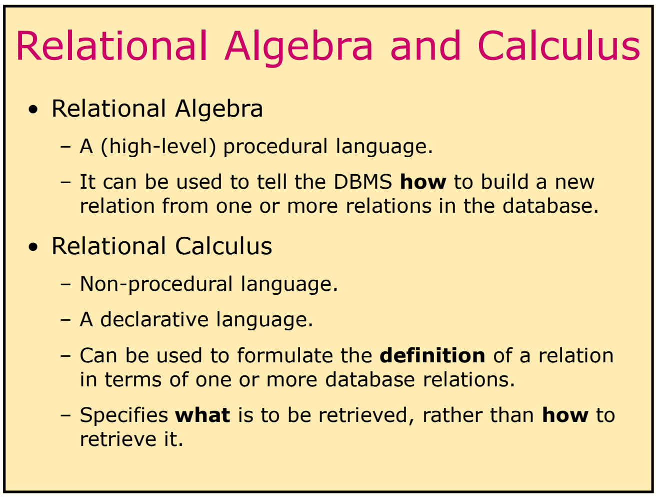
3



Relational Algebra and Calculus

- Long history by computing standards.
  - Defined and published by E.F. Codd in 1971.
- Formally, the relational algebra and relational calculus are equivalent to one another.
  - For every expression in the algebra, there is an equivalent expression in the calculus.
- Neither is User friendly.
- They illustrate the basic operations required of any Data Manipulation Language for relational databases.
- A standard for comparing DMLs.

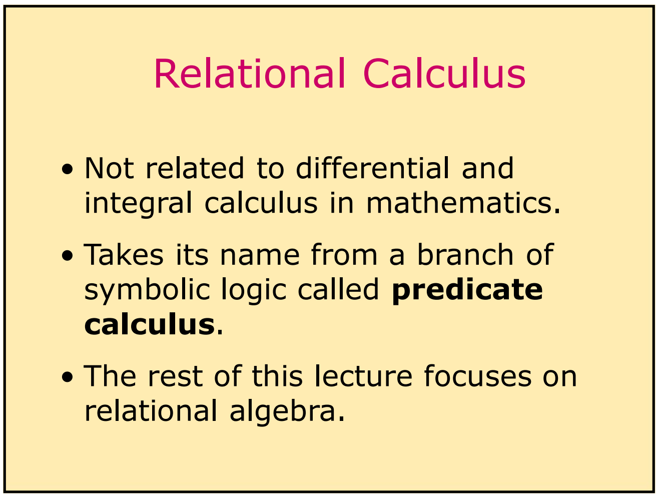
4



Relational Algebra and Calculus

- Relational Algebra
  - A (high-level) procedural language.
  - It can be used to tell the DBMS **how** to build a new relation from one or more relations in the database.
- Relational Calculus
  - Non-procedural language.
  - A declarative language.
  - Can be used to formulate the **definition** of a relation in terms of one or more database relations.
  - Specifies **what** is to be retrieved, rather than **how** to retrieve it.

5



Relational Calculus

- Not related to differential and integral calculus in mathematics.
- Takes its name from a branch of symbolic logic called **predicate calculus**.
- The rest of this lecture focuses on relational algebra.

6

Content

- Data Manipulation Language
- Introduction to Relational Algebra and Calculus
- **Relational Algebra**
  - Unary Operations
    - Selection
    - Projection
  - Binary Operations
    - Cartesian Product
    - Union
    - Set Difference
  - Implementation in SQL
- Summary and Reading

7

Relational Algebra: 5 Fundamental Operations

- A theoretical language.
- Many variations of the operations that are included in it.
- 5 fundamental operations.

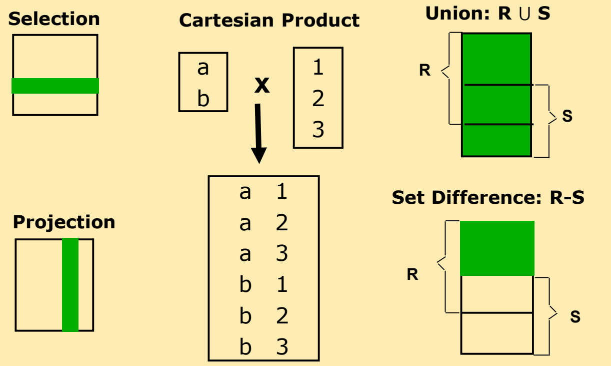
Summarised on the next slide.

- Additional operations are defined as a combination of two or more of the 5 basic operations.

Beyond the scope of the Semester 1 part of this module.

8

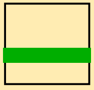
Relational Algebra: 5 Fundamental Operations



a	1
a	2
a	3
b	1
b	2
b	3

9

Selection (or Restriction)



- Selects a subset of the tuples in a relation that satisfy a selection condition.

σ <selection condition> (<relation name>)

- σ is the lower-case letter of the Greek alphabet called sigma.
- The selection condition is a Boolean expression specified on the attributes of <relation-name>.
  - Can include AND, NOT, OR.
- Result is a relation that has the same attributes as the relation specified in <relation-name>.

10

Example of Selection

staffNo	name	position	salary	branchNo
S1	Tom	Manager	30000	B005
S2	Sarah	Assistant	12000	B003
S3	Harry	Supervisor	18000	B003
S4	Sophie	Assistant	9000	B007
S5	Louise	Manager	24000	B003
S6	Laura	Assistant	9000	B005

List all staff with a salary greater than 20,000.

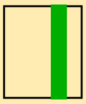
Relational Algebra: $\sigma_{salary>20000}$ (Staff)

SQL: SELECT * FROM Staff WHERE salary > 20000;

staffNo	name	position	salary	branchNo
S1	Tom	Manager	30000	B005
S5	Louise	Manager	24000	B003

11

Projection



- Selects a subset of the attributes of a relation.

Π <attribute list> (<relation name>)

- Π is the capital letter of the Greek alphabet called Pi.
- The order of the attributes in the result is the same as that in the <attribute list>.
- Duplicates tuples are removed.

12

Example of Projection

staffNo	name	position	salary	branchNo
S1	Tom	Manager	30000	B005
S2	Sarah	Assistant	12000	B003
S3	Harry	Supervisor	18000	B003

List the number, name and salary of all staff.

Relational Algebra: $\Pi_{\text{staffNo, name, salary}}$ (Staff)

SQL: SELECT DISTINCT staffNo, name, salary
FROM Staff;

staffNo	name	salary
S1	Tom	30000
S2	Sarah	12000
S3	Harry	18000

13

Relational Algebra: Closure

- Operations work on one or more relations to define another relation without changing the original.
- Both operands and results are relations.
- So the output from one operation can become the input to another operation.
- This property is called closure.
- Notice the parallel with algebraic operations in arithmetic, which take one or more numbers as operands and return a number as output.

14

Example of Closure

staffNo	name	position	salary	branchNo
S1	Tom	Manager	30000	B005
S2	Sarah	Assistant	12000	B003
S3	Harry	Supervisor	18000	B003
S4	Sophie	Assistant	9000	B007
S5	Louise	Manager	24000	B003
S6	Laura	Assistant	9000	B005

Π_{name} ($\sigma_{\text{salary>20000}}$ (Staff))

i.e., list the names of all staff with a salary > 20,000.

name
Tom
Louise

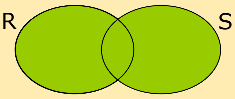
15

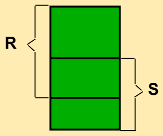
Unary and Binary Operators

- Unary Operations
  - operate on one relation
  - selection
  - projection
- Binary Operations
  - operate on two relations
  - Cartesian product
  - union
  - set difference

16

Union R ∪ S





- The union of relations R and S is a relation that contains all the tuples of both R and S.
- Duplicates are removed.
- R and S must be union compatible.

(This is defined two slides later.)

- The resulting relation might have the same attribute names as the 1st or 2nd relation.

17

Example of Union

Employee

EmpNo	Ename	Sal
7782 7839	clark king	2450 5000

Manager

EmpNo	Ename	Sal
8034 8044	smith yao	7000 6000

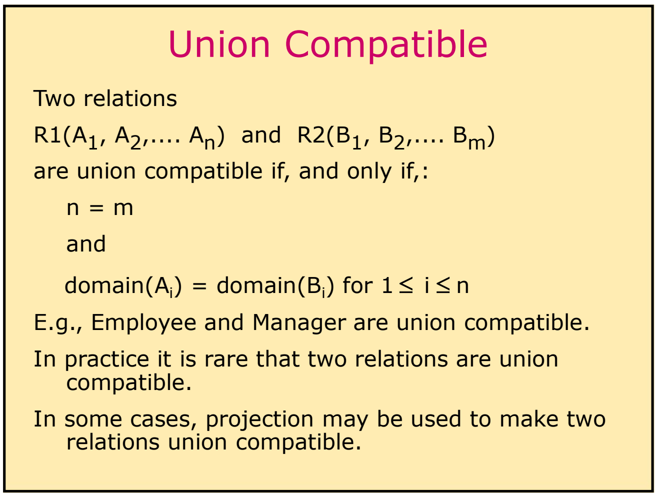
- List all the employee numbers of staff who are employees or managers or both.
- Relational Algebra:

Π_{EmpNo}(Employee) ∪ Π_{EmpNo}(Manager)

SQL:
SELECT EmpNo FROM Employee
UNION
SELECT EmpNo FROM Manager

EmpNo
7782
7839
8034
8044

18



Union Compatible

Two relations

$R1(A_1, A_2,\dots A_n)$ and $R2(B_1, B_2,\dots B_m)$

are union compatible if, and only if,:

$n = m$

and

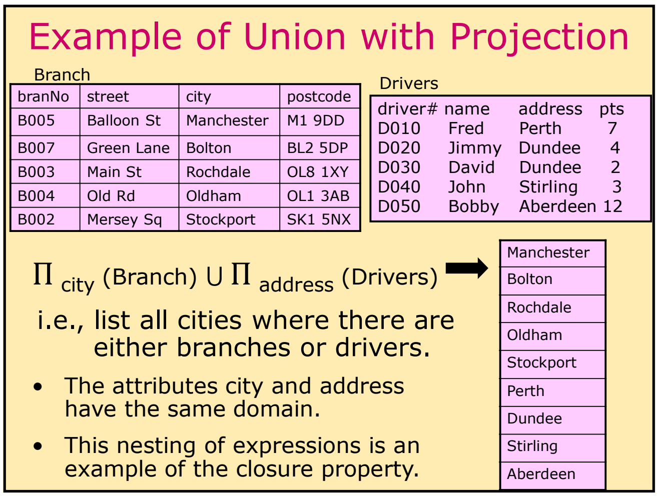
$\text{domain}(A_i) = \text{domain}(B_i)$ for $1 \leq i \leq n$

E.g., Employee and Manager are union compatible.

In practice it is rare that two relations are union compatible.

In some cases, projection may be used to make two relations union compatible.

19



Example of Union with Projection

Branch

branNo	street	city	postcode
B005	Balloon St	Manchester	M1 9DD
B007	Green Lane	Bolton	BL2 5DP
B003	Main St	Rochdale	OL8 1XY
B004	Old Rd	Oldham	OL1 3AB
B002	Mersey Sq	Stockport	SK1 5NX

Drivers

driver#	name	address	pts
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

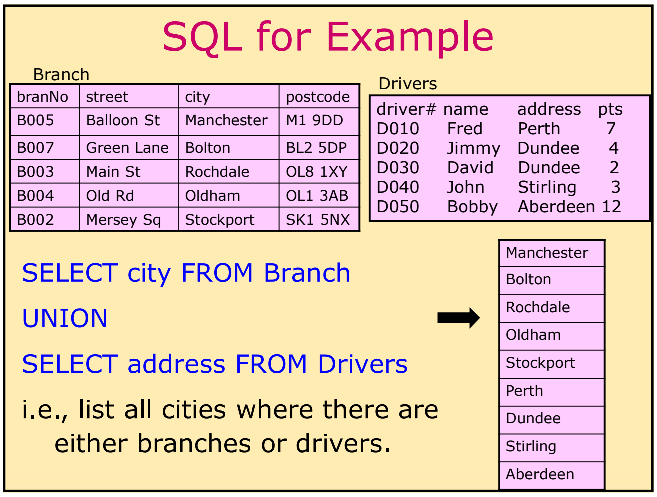
Π_{city} (Branch) ∪ Π_{address} (Drivers) ➡

Manchester
Bolton
Rochdale
Oldham
Stockport
Perth
Dundee
Stirling
Aberdeen

i.e., list all cities where there are either branches or drivers.

- The attributes city and address have the same domain.
- This nesting of expressions is an example of the closure property.

20



SQL for Example

Branch

branNo	street	city	postcode
B005	Balloon St	Manchester	M1 9DD
B007	Green Lane	Bolton	BL2 5DP
B003	Main St	Rochdale	OL8 1XY
B004	Old Rd	Oldham	OL1 3AB
B002	Mersey Sq	Stockport	SK1 5NX

Drivers

driver#	name	address	pts
D010	Fred	Perth	7
D020	Jimmy	Dundee	4
D030	David	Dundee	2
D040	John	Stirling	3
D050	Bobby	Aberdeen	12

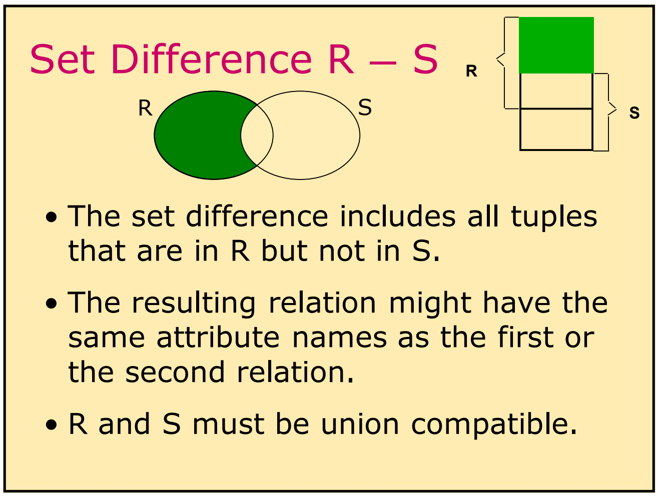
```
SELECT city FROM Branch
UNION
SELECT address FROM Drivers
```

➡

Manchester
Bolton
Rochdale
Oldham
Stockport
Perth
Dundee
Stirling
Aberdeen

i.e., list all cities where there are either branches or drivers.

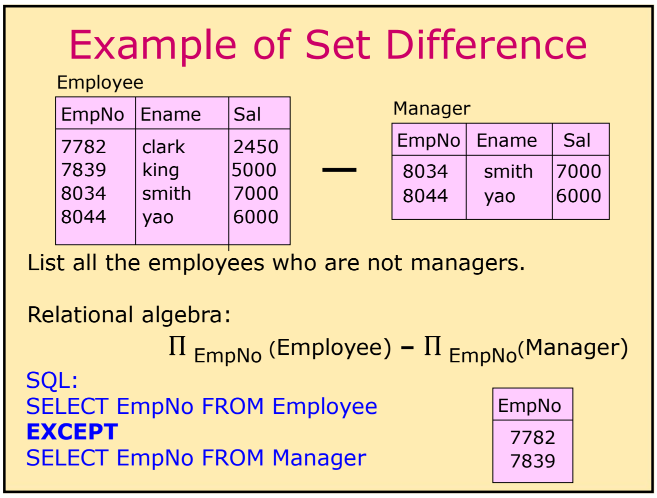
21



Set Difference R – S

- The set difference includes all tuples that are in R but not in S.
- The resulting relation might have the same attribute names as the first or the second relation.
- R and S must be union compatible.

22



Example of Set Difference

Employee

EmpNo	Ename	Sal
7782	clark	2450
7839	king	5000
8034	smith	7000
8044	yao	6000

—

Manager

EmpNo	Ename	Sal
8034	smith	7000
8044	yao	6000

List all the employees who are not managers.

Relational algebra:

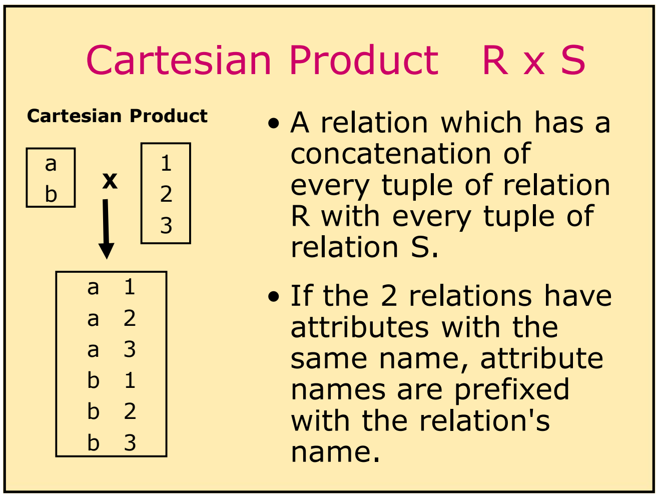
$$\Pi_{\text{EmpNo}}(\text{Employee}) - \Pi_{\text{EmpNo}}(\text{Manager})$$

SQL:

```
SELECT EmpNo FROM Employee
EXCEPT
SELECT EmpNo FROM Manager
```

EmpNo
7782
7839

23



Cartesian Product R x S

Cartesian Product

a
b

x

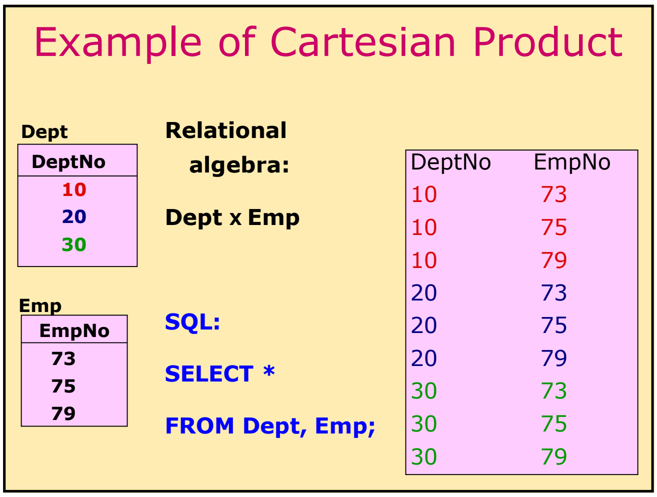
1
2
3

a	1
a	2
a	3
b	1
b	2
b	3

- A relation which has a concatenation of every tuple of relation R with every tuple of relation S.
- If the 2 relations have attributes with the same name, attribute names are prefixed with the relation's name.

24

Dr C.H. Bryant, School of SEE, Salford, UK.



Example of Cartesian Product

Dept

DeptNo
10
20
30

Emp

EmpNo
73
75
79

Relational algebra:

Dept x Emp

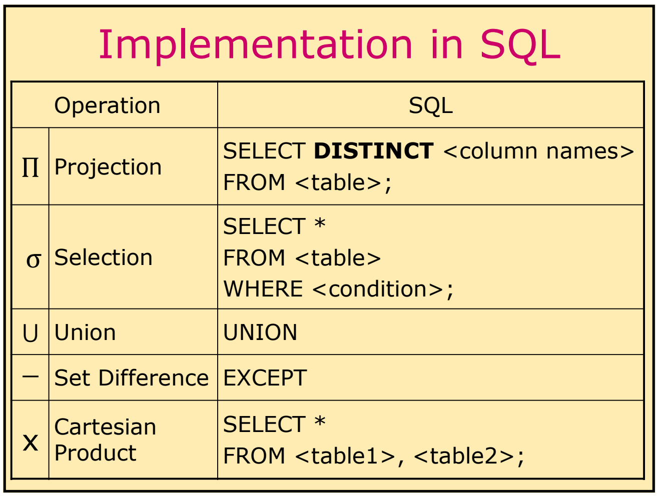
SQL:

SELECT *

FROM Dept, Emp;

DeptNo	EmpNo
10	73
10	75
10	79
20	73
20	75
20	79
30	73
30	75
30	79

25



Implementation in SQL

	Operation	SQL

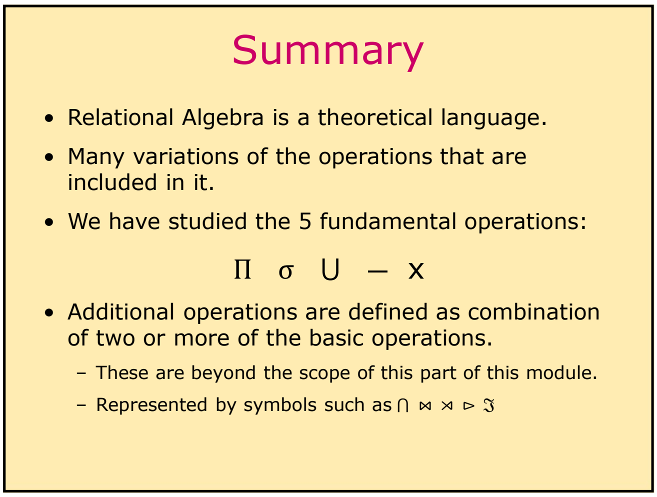
Π	Projection	SELECT **DISTINCT** <column names>
		FROM <table>;

σ	Selection	SELECT *
		FROM <table>
		WHERE <condition>;

U	Union	UNION
–	Set Difference	EXCEPT

x	Cartesian Product	SELECT *
		FROM <table1>, <table2>;

26



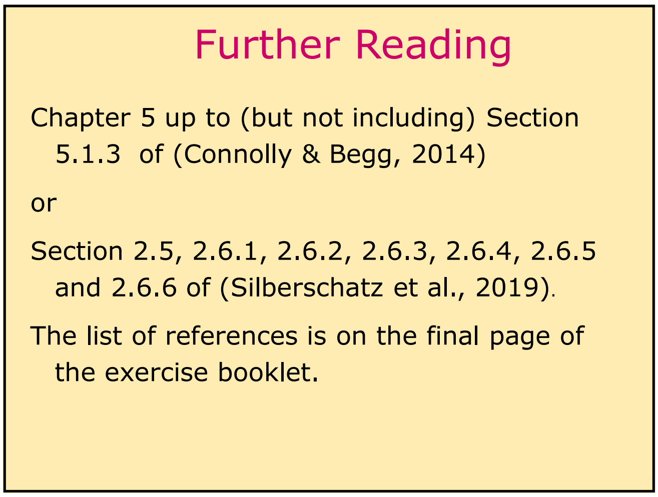
Summary

- Relational Algebra is a theoretical language.
- Many variations of the operations that are included in it.
- We have studied the 5 fundamental operations:

  Π σ U – x

- Additional operations are defined as combination of two or more of the basic operations.
  - These are beyond the scope of this part of this module.
  - Represented by symbols such as ∩ ⋈ ⋊ ⊳ ℑ

27



Further Reading

Chapter 5 up to (but not including) Section 5.1.3 of (Connolly & Begg, 2014)

or

Section 2.5, 2.6.1, 2.6.2, 2.6.3, 2.6.4, 2.6.5 and 2.6.6 of (Silberschatz et al., 2019).

The list of references is on the final page of the exercise booklet.

28

Databases

Enhanced Entity Relationship (EER) Modelling

Dr Bryant

1

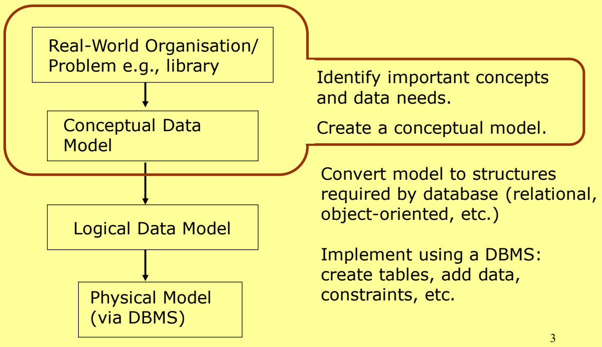
Contents

- Recap: ER modelling
- Additional Features of ER models
- Motivation for Enhancement
- Enhanced ER models
- Summary and Reading

2

2

Recap: Role of Conceptual Modelling Within the Design Process



Real-World Organisation/ Problem e.g., library

Conceptual Data Model

Logical Data Model

Physical Model (via DBMS)

Identify important concepts and data needs.

Create a conceptual model.

Convert model to structures required by database (relational, object-oriented, etc.)

Implement using a DBMS: create tables, add data, constraints, etc.

3

3

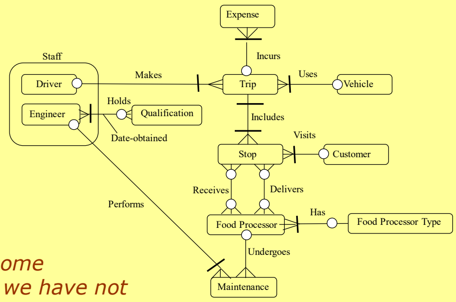
Recap: Entity Relationship (ER) Modelling

- A conceptual data model:
  - identifies the important elements and the relationships between them;
  - is independent of the type of logical model, the choice of DBMS or the type of database.
- One form is the Entity-Relationship Model.
  - Contains entities, attributes, relationships and constraints.
  - Can be represented graphically using the Crow's foot notation.

4

4

Example of more realistic ER diagram



N.B. This has some features which we have not studied previously. By the end of this lecture you should be able to understand these.

5

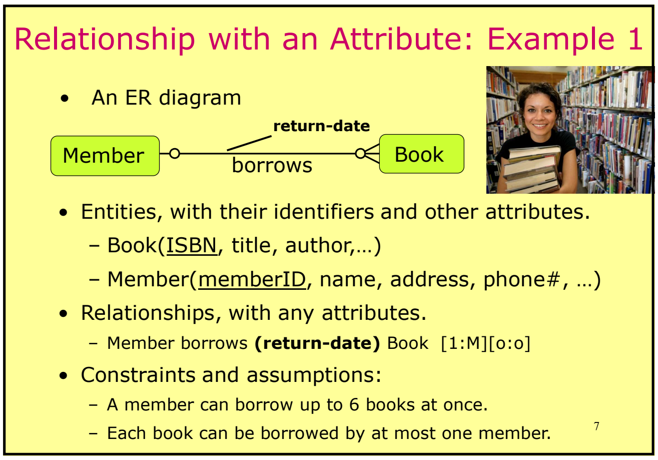
5

Additional Features of ER Diagrams

- Relationships with attributes
- Complex relationships
  - Ternary
  - Quaternary
  - Recursive

6

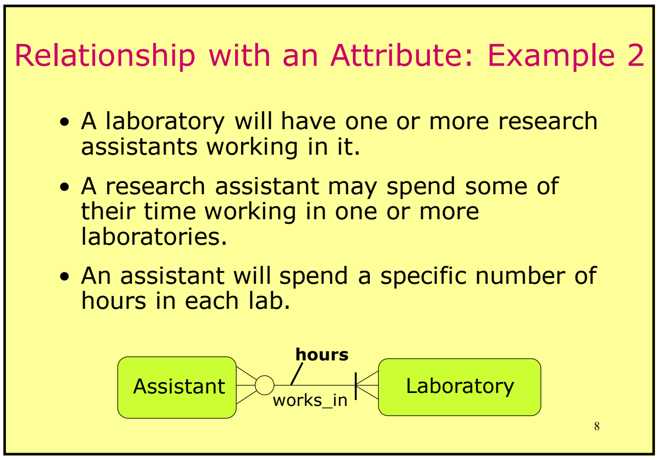
6



Relationship with an Attribute: Example 1

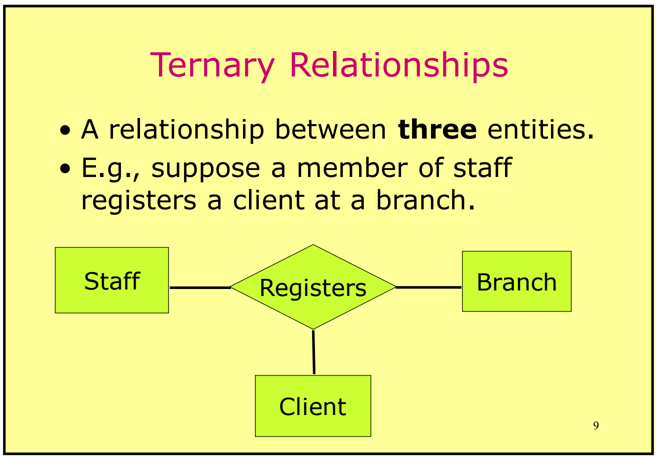
- An ER diagram

- Entities, with their identifiers and other attributes.
  - Book(<u>ISBN</u>, title, author,…)
  - Member(<u>memberID</u>, name, address, phone#, …)
- Relationships, with any attributes.
  - Member borrows **(return-date)** Book [1:M][o:o]
- Constraints and assumptions:
  - A member can borrow up to 6 books at once.
  - Each book can be borrowed by at most one member.



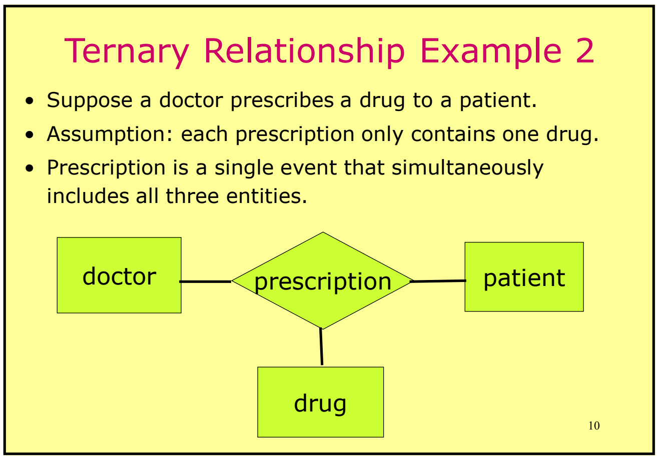
Relationship with an Attribute: Example 2

- A laboratory will have one or more research assistants working in it.
- A research assistant may spend some of their time working in one or more laboratories.
- An assistant will spend a specific number of hours in each lab.



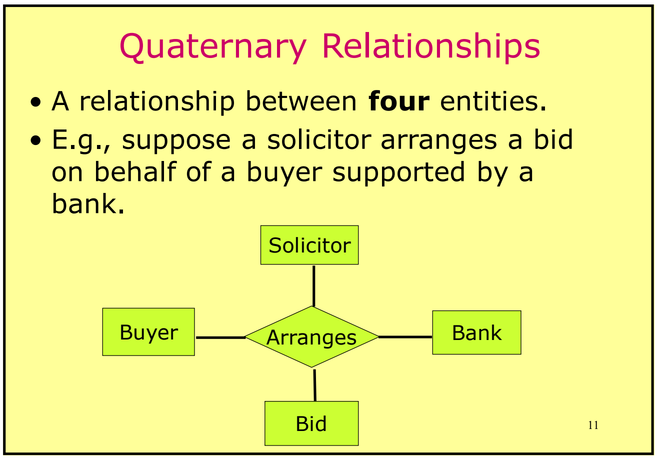
Ternary Relationships

- A relationship between **three** entities.
- E.g., suppose a member of staff registers a client at a branch.



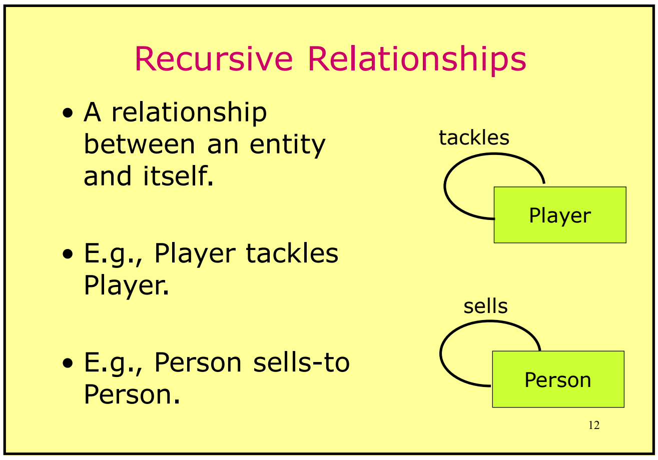
Ternary Relationship Example 2

- Suppose a doctor prescribes a drug to a patient.
- Assumption: each prescription only contains one drug.
- Prescription is a single event that simultaneously includes all three entities.



Quaternary Relationships

- A relationship between **four** entities.
- E.g., suppose a solicitor arranges a bid on behalf of a buyer supported by a bank.



Recursive Relationships

- A relationship between an entity and itself.
- E.g., Player tackles Player.
- E.g., Person sells-to Person.

Contents

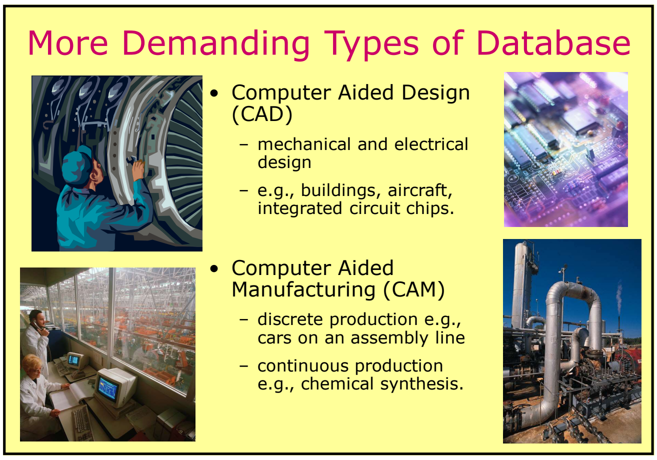
- Recap: ER modelling
- Additional Features of ER models
- **Motivation for Enhancement**
- Enhanced ER models
- Summary and Reading

13

Motivation for Enhancement

- The basic concepts of the Entity-Relationship (ER) model are normally adequate for building data models of traditional, administrative based database systems such as:
  - stock control
  - product ordering
  - customer invoicing
- Since 1980s there has been a rapid increase in the development of many new database systems that have more demanding database requirements.

14



More Demanding Types of Database

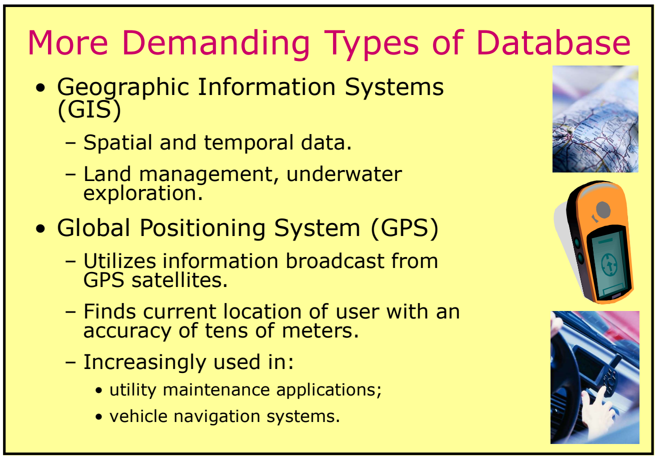
- Computer Aided Design (CAD)
  - mechanical and electrical design
  - e.g., buildings, aircraft, integrated circuit chips.
- Computer Aided Manufacturing (CAM)
  - discrete production e.g., cars on an assembly line
  - continuous production e.g., chemical synthesis.

15

More Demanding Types of Database

- Computer Aided Software Engineering (CASE)
  - stages of the software development life cycle.
- Network Management Systems
  - coordinate the delivery of communication services across a computer network.
- Multimedia Databases
  - free form text, photographs, diagrams, audio, video, spreadsheets.
- Digital Publishing
  - books, journals, articles.
- Interactive and Dynamic Websites

16



More Demanding Types of Database

- Geographic Information Systems (GIS)
  - Spatial and temporal data.
  - Land management, underwater exploration.
- Global Positioning System (GPS)
  - Utilizes information broadcast from GPS satellites.
  - Finds current location of user with an accuracy of tens of meters.
  - Increasingly used in:
    - utility maintenance applications;
    - vehicle navigation systems.

17

Contents

- Recap: ER modelling
- Additional Features of ER models
- Motivation for Enhancement
- **Enhanced ER models**
- Summary and Reading

18

Enhanced Entity-Relationship (EER) model

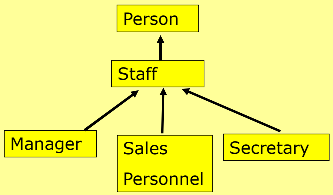
- Basic concepts of ER modelling are often not sufficient to represent the requirements of the newer, more complex applications.
- The EER model is the ER model supported with additional semantic concepts:
  - Specialisation/generalisation
    - The rest of this lecture introduces these.
  - Aggregation and composition.
    - Beyond the scope of this module.

19

19

Subclasses and Superclasses

- Inheritance allows one class to be defined as a special case of a more general class.
- Special case is called a subclass.
- More general cases are called superclasses.
- All members of a subclass are also members of superclass.



Person is a superclass.

Staff is a subclass of Person.

Manager, SalesStaff and Secretary are subclasses of Staff.

20

Superclasses and Subclasses

- Some superclasses may contain overlapping subclasses.
  - E.g., there may be a member of staff who is both a Manager and a member of Sales Personnel.
- Not every member of a superclass need be a member of a subclass.
  - E.g., some members of staff may not have a distinct job role.

21

21

Why Bother with Superclasses and Subclasses?

- Unshared attributes can cause problems if we try to represent all members with a single entity.
- Can result in many members having NULL values.

See example on next slide.

22

22

Example

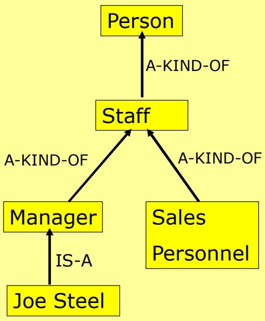
- Sales Personnel have special attributes such as salesArea and carAllowance.
- Only secretaries have a typing speed.
- Only managers can earn a bonus.

staffNo	name	position	salary	bonus	Sales Area	CarAllowance	Typing speed
S1	Tom	Manager	30000	2000			
S2	Sarah	Assistant	12000				
S3	Harry	Sales Assistant	27000		Bury	5000	
S4	Sophie	Assistant	9000				
S11	Jane	Secretary	8500				100
S12	Paula	Sales Assistant	17000		York	3700	

23

Inheritance

- Generalisation
  - process of forming a superclass.
- Specialisation
  - process of forming a subclass.
- By default, a subclass inherits all the properties of its superclass(es) and defines its own unique properties.



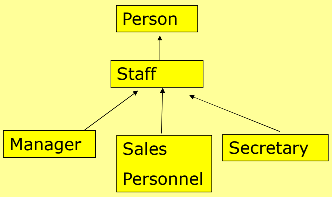
24

24

Example of Inheritance

A member of the SalesPersonnel subclass inherits all the attributes of Staff such as staffNo, name, position and salary.

It also inherits those specifically associated with the SalesPersonnel subclass such as salesArea and carAllowance.



Why Bother with Superclasses and Subclasses?

- Some **relationships** are only associated with particular subclasses and not with a superclass.
- E.g., relationships that are only associated with particular types of staff and not with staff in general.
  - Sales Personnel may have distinct relationships that are not appropriate for all staff.
  - SalesPersonnel Uses Car

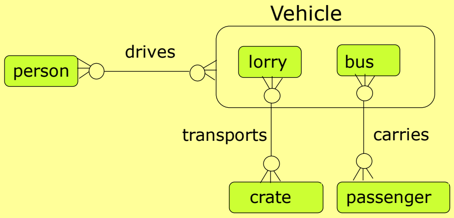
Another Example

People may drive one or more vehicles.

Vehicles may be driven by more than one person.

Lorries transport crates.

Buses carry passengers.



Summary

- E-R models can include relationships with their own attributes.
- E-R models can include a relationship:
  - involving more than two entities;
  - between an entity and itself.
- E-R models can be enhanced by modelling hierarchies of entities using:
  - specialisation and generalisation;
  - subclasses and superclasses;
  - inheritance.

References

The "crow's foot" notation is denoted in:

- Appendix C.2 of (Connolly & Begg, 2014);
- Appendix A.2 of (Connolly & Begg, 2004)
- (Barker, 1989)

The list of references is on the final page of the exercise booklet.

Dr C.H. Bryant, School of SSE, Salford, UK.