Databases: Introduction

P.J. McBrien

Imperial College London

Databases are Computer Stores of Data!

Tiny Bank Ltd Customer: McBrien, P.
Strand Branch Current Acc: 10000100
Sortcode: 55-66-67

Trans	Amount	Date
1000	2300.00	5/1/1999
1002	-223.45	8/1/1999
1006	10.23	15/1/1999

Tiny Bank Ltd Customer: Poulovassilis, A.
Wimbledon Branch Current Acc: 10000107
Sortcode: 55-66-56

Trans	Amount	Date
1004	-100.00	11/1/1999
1007	345.56	15/1/1999

Tiny Bank Ltd Customer: McBrien, P.
Strand Branch Deposit Acc: 10000101
Sortcode: 55-66-67

Trans	Amount	Date
1001	4000.00	5/1/1999
1008	1230.00	15/1/1999

Tiny Bank Ltd Customer: Poulovassilis, A.
Wimbledon Branch Deposit Acc: 10000119
Sortcode: 55-66-56

Trans	Amount	Date
1009	5600.00	18/1/1999

Tiny Bank Ltd Customer: Boyd, M.
Goodge St Branch Current Acc: 10000103
Sortcode: 55-66-34

Trans	Amount	Date
1005	145.50	12/1/1999

Tiny Bank Ltd Customer: Bailey, J.
Wimbledon Branch Current Acc: 10000125
Sortcode: 55-66-56

Trans	Amount	Date

No transactions this month

Deposit Rates

Account	Rate
101	5.25
119	5.50

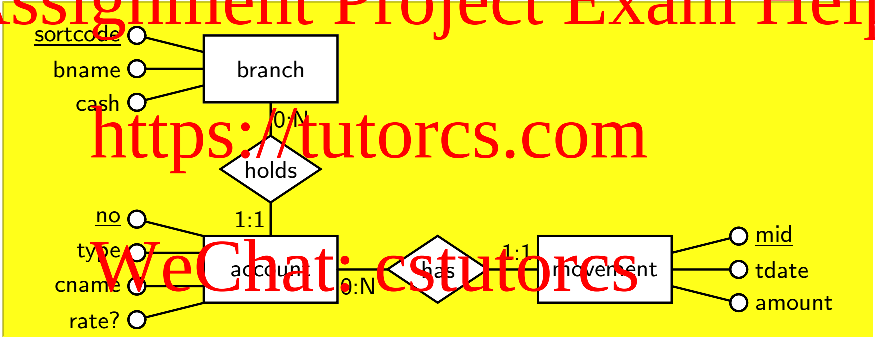
Relational Data Model

Relational Data Model

Roughly: storing data in tables

bank_data									
no	sortcode	bname	cash	type	cname	rate?	<u>mid</u>	amount	tdate
100	67	Strand	34005.00	current	McBrien, P.		1000	2300.00	1999-01-05
101	67	Strand	34005.00	deposit	McBrien, P.	5.25	1001	4000.00	1999-01-05
100	67	Strand	34005.00	current	McBrien, P.		1002	-223.45	1999-01-08
107	56	Wimbledon	84340.45	current	Poulovassilis, A.		1004	-100.00	1999-01-11
103	34	Geodge St	6900.67	current	Boyd, M.		1005	145.50	1999-01-12
100	67	Strand	34005.00	current	McBrien, P.		1006	10.23	1999-01-15
107	56	Wimbledon	84340.45	current	Poulovassilis, A.		1007	345.56	1999-01-15
101	67	Strand	34005.00	deposit	McBrien, P.	5.25	1008	1230.00	1999-01-15
119	56	Wimbledon	84340.45	deposit	Poulovassilis, A.	5.50	1009	5600.00	1999-01-18

Database Design: ER Modelling



Structured Data: Relational Model

branch		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

movement			
mid	no	amount	tdate
1000	100	2300.00	5/1/1999
1001	101	4000.00	5/1/1999
1002	100	-223.45	8/1/1999
1004	107	-100.00	11/1/1999
1005	103	145.50	12/1/1999
1006	100	10.23	15/1/1999
1007	107	345.56	15/1/1999
1008	101	1230.00	15/1/1999
1009	119	5600.00	18/1/1999

account				
no	type	cname	rate?	sortcode
100	'current'	'McBrien, P.'	NULL	67
101	'deposit'	'McBrien, P.'	5.25	67
103	'current'	'Boyd, M.'	NULL	34
107	'current'	'Poulovassilis, A.'	NULL	56
119	'deposit'	'Poulovassilis, A.'	5.50	56
125	'current'	'Bailey, J.'	NULL	56

key branch(sortcode)
key branch(bname)
key movement(mid)
key account(no)

movement(no) $\overset{fk}{\Rightarrow}$ account(no)

account(sortcode) $\overset{fk}{\Rightarrow}$ branch(sortcode)

Data Model: CSV

branch.csv

```
sortcode,bname,cash
56,"Wimbledon",94340.45
34,"Goodge St", 8900.67
67,"Strand",34005.00
```

account.csv

```
no,type,cname,rate,sortcode
100,"current","McBrien, P.",,67
101,"deposit","McBrien, P.",5.25,67
103,"current","Boyd, M.",,34
107,"current","Poulovassilis, A.",,56
119,"deposit","Poulovassilis, A.",5.50,56
125,"current","Bailey, J.",,56
```

movement.csv

```
mid,no,amount,tdate
1000,100,2300.00,5/1/1999
1001,101,4000.00,5/1/1999
1002,100,-223.45,8/1/1999
1004,107,-100.00,11/1/1999
1005,103,145.50,12/1/1999
1006,100,10.23,15/1/1999
1007,107,345.56,15/1/1999
1008,101,1230.00,15/1/1999
1009,119,5600.00,18/1/1999
```

Semistructured Data: XML

```
⟨bank⟩
  ⟨branch sortcode="67" bname="Strand" cash="34005.00"⟩
    ⟨account no="100" type="current" cname="McBrien, P."⟩
      ⟨movement mid="1000" amount="2300.00" tdate="5/1/1999" /⟩
      ⟨movement mid="1002" amount="-223.45" tdate="8/1/1999" /⟩
      ⟨movement mid="1006" amount="10.23" tdate="15/1/1999" /⟩
    ⟨/account⟩
    ⟨account no="101" type="deposit" cname="McBrien, P." rate="5.25"⟩
      ⟨movement mid="1001" amount="4000.00" tdate="5/1/1999" /⟩
      ⟨movement mid="1008" amount="1230.00" tdate="15/1/1999" /⟩
    ⟨/account⟩
  ⟨/branch⟩
⟨/bank⟩
```

SQL DDL: Implementation of the Relational Model

```
CREATE TABLE branch
( sortcode INTEGER NOT NULL,
  bname VARCHAR(20) NOT NULL,
  cash DECIMAL(10,2) NOT NULL,
  CONSTRAINT branch_pk PRIMARY KEY (sortcode)
)

CREATE UNIQUE INDEX branch_bname_idx
  ON branch(bname)
```

```
CREATE TABLE account
( no INTEGER NOT NULL,
  type CHAR(8) NOT NULL,
  cname VARCHAR(20) NOT NULL,
  rate DECIMAL(4,2) NULL,
  sortcode INTEGER NOT NULL,
  CONSTRAINT account_pk
      PRIMARY KEY (no),
  CONSTRAINT account_fk
      FOREIGN KEY (sortcode) REFERENCES branch
)

CREATE INDEX account_type_idx ON account(type)
```

```
CREATE TABLE movement
( mid INTEGER NOT NULL,
  no INTEGER NOT NULL,
  amount DECIMAL(10,2) NOT NULL,
  tdate DATETIME NOT NULL,
  CONSTRAINT movement_pk
       PRIMARY KEY (mid),
  CONSTRAINT movement_fk
       FOREIGN KEY (no) REFERENCES account
)
```

SQL DML: Implementation of the Relational Algebra

Basic SQL SELECT statements

```
SELECT  no, cname, rate
FROM    account
WHERE   type='deposit'
```

SQL Joins

```
SELECT  bname, no, rate
FROM    branch JOIN account USING (sortcode)
WHERE   type='deposit'
```

Same as

```
SELECT  bname, no, rate
FROM    account JOIN branch ON branch.sortcode=account.sortcode
WHERE   type='deposit'
```

Same as

```
SELECT  bname, no, rate
FROM    account, branch
WHERE   branch.sortcode=account.sortcode
AND     type='deposit'
```

RDBMS Products

Product	SQL Language	Company
DB2	SQL PL	IBM
Oracle	PL/SQL	Oracle
Sybase	Transact-SQL	SAP
SQLServer	Transact-SQL	Microsoft
PostgreSQL	PL/pgSQL	Open Source
MySQL	MySQL	Open Source (Oracle)

All partially implement ANSI SQL

Transactions

```
BEGIN TRANSACTION
     UPDATE branch
     SET cash=cash-10000.00
     WHERE sortcode=56

     UPDATE branch
     SET cash=cash+10000.00
     WHERE sortcode=34
COMMIT TRANSACTION
```

database management systems (**DBMS**) implements indivisible tasks called **transactions**

The ACID Properties

- **Atomicity** all or nothing
- **Consistency** consistent before → consistent after
- **Isolation** independent of any other transaction
- **Durability** completed transaction are durable

Transaction Properties: Atomicity

```
BEGIN TRANSACTION
    UPDATE branch
    SET    cash=cash −10000.00
    WHERE  sortcode=56
```

CRASH

Failure to maintain Atomicity

Suppose that the system crashes half way through processing a cash transfer, and the first part of the transfer has been written to disc

- The database on disc is left in an inconsistent state: the sum of cash should be £137,246.12 but only £127,246.12 recorded
- A DBMS implementing **Atomicity** of transactions would on restart undo the change to branch 56

Transaction Properties: Consistency

```
BEGIN TRANSACTION
    DELETE FROM branch
    WHERE sortcode=56

    INSERT INTO account
    VALUES (100,'Smith, J','deposit',5.00,34)
END TRANSACTION
```

Failure to maintain Consistency

Suppose that a user deletes branch with sortcode 56, and inserts a desposit account number 100 for John Smith at branch sortcode 34

- The database is left in an inconsistent state for two reasons
  - it has three accounts recorded for a branch that appears not to exist, and
  - it has two records for account number 100, with different details for the account
- A DBMS implementing **Consistency** of transactions would forbid both of these changes to the database

Transaction Properties: Isolation

```
BEGIN TRANSACTION                        BEGIN TRANSACTION
   UPDATE branch
   SET    cash=cash −10000.00
   WHERE  sortcode=56

                                            SELECT SUM(cash) AS net_cash
                                            FROM   branch

   UPDATE branch
   SET    cash=cash +10000.00
   WHERE  sortcode=34
END TRANSACTION                          END TRANSACTION
```

Failure to maintain Isolation

Suppose that the system sums the cash in the bank in one transaction, half way through processing a cash transfer in another transaction

- The result of the summation of cash in the bank erroneously reports £127,246.12, whereas the movement of cash always leaves a total of £137,246.12
- A DBMS implementing **Isolation** of transactions ensures that transactions always report results based on the values of committed transactions

Transaction Properties: Durability

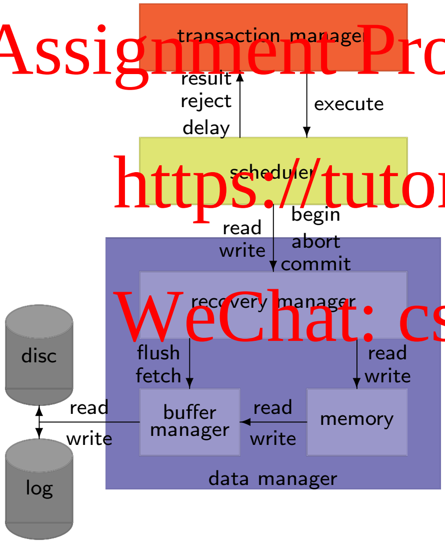
```
BEGIN TRANSACTION
   UPDATE branch
   SET    cash=cash −10000.00
   WHERE  sortcode=56

   UPDATE branch
   SET cash=cash +10000.00
   WHERE sortcode=34
END TRANSACTION
CRASH
```

Failure to maintain Durability

Suppose that the system crashes after informing the user that it has committed the transfer of cash, but has not yet written to disc the update to branch 34

- The database on disc is left in an inconsistent state, with £10,000 ‘missing’
- A DBMS implementing **Durability** of transactions would on restart complete the change to branch 34 (or alternatively never inform a user of commitment with writing the results to disc).

DBMS Architecture



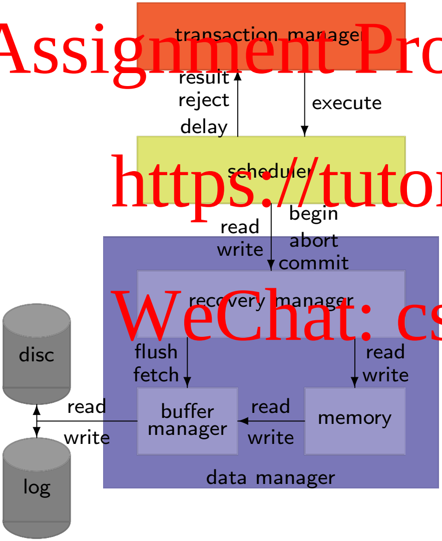
Transaction Manager

```
BEGIN TRANSACTION
    UPDATE branch
    SET    cash=cash −10000.00
    WHERE sortcode=56

    UPDATE branch
    SET cash=cash +10000.00
    WHERE sortcode=34
END TRANSACTION
```

⇩

$r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}]$

DBMS Architecture



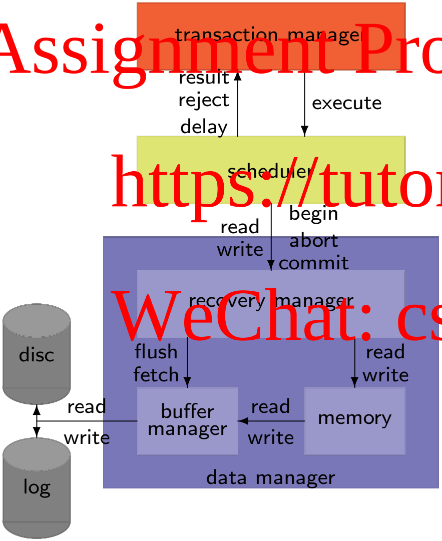
Transaction Manager

```
BEGIN TRANSACTION
    SELECT SUM(cash) AS net_cash
    FROM   branch
END TRANSACTION
```

$\Downarrow$

$r_2[b_{56}], r_2[b_{34}], r_2[b_{67}]$

DBMS Architecture



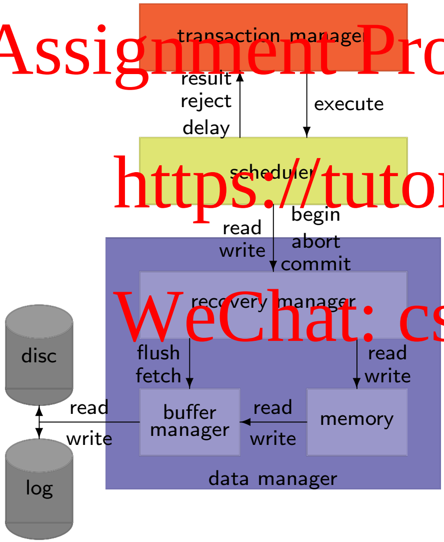
Scheduler

execute($r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}]$)
execute($r_2[b_{56}], r_2[b_{34}], r_2[b_{67}]$)

$$\Downarrow$$

$b_1, r_1[b_{56}], w_1[b_{56}], b_2, r_2[b_{56}]$
$r_1[b_{34}], w_1[b_{34}], c_1, r_2[b_{34}], r_2[b_{67}], c_2$

DBMS Architecture



Data Manager

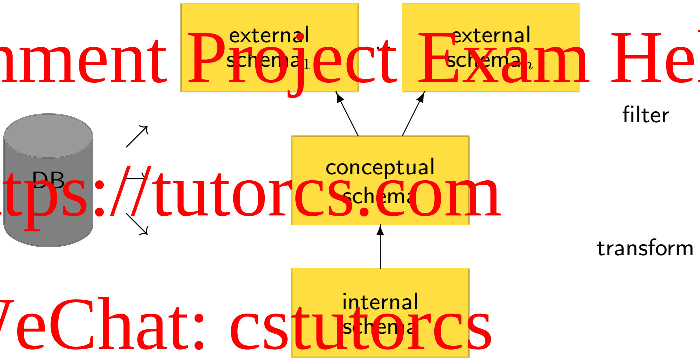
$b_1, r_1[b_{56}], w_1[b_{56}], b_2,$
$r_2[b_{56}]$
$r_1[b_{34}], w_1[b_{34}],$
$c_1, r_2[b_{34}],$
$r_2[b_{67}], c_2$

$\Downarrow$

Say $P_1 = [b_{34}], P_2 = [b_{56}, b_{67}]$

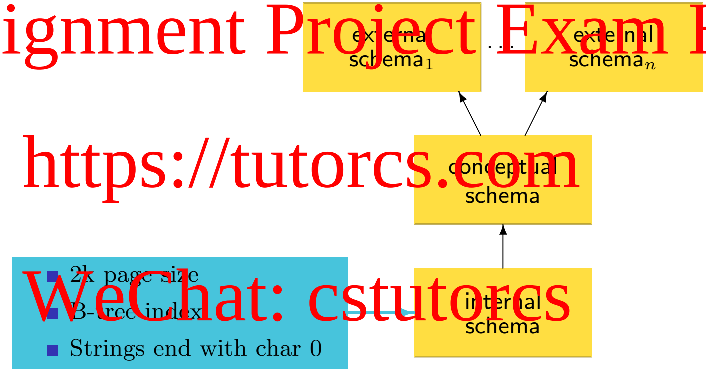
$M_r(P_2), B_r(P_2), D_r(P_2), M_w(P_2), L_w(P_2),$
$M_r(P_2),$
$M_r(P_1), B_r(P_1), D_r(P_1), M_w(P_1), L_w(P_1),$
$L_w(c_1), M_r(P_1),$
$M_r(P_2), L_w(c_2), D_w(P_1), D_w(P_2)$

ANSI/SPARC Model



- ANSI/SPARC model views three levels of abstractions
- **schema** means *structure of the database*

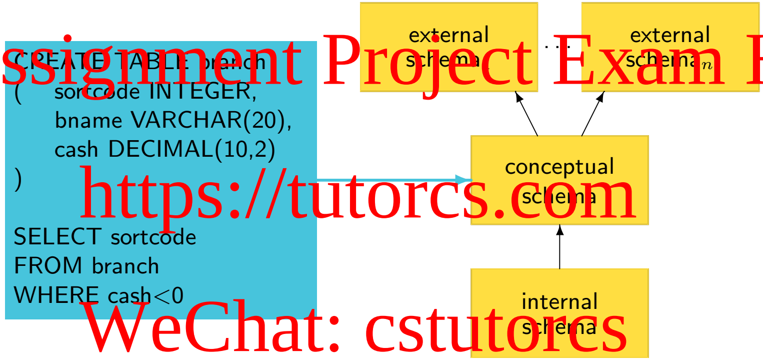
ANSI/SPARC Model (Internal Schema)



- 2k page size
- B-tree index
- Strings end with char 0

- Describes the physical layout of data

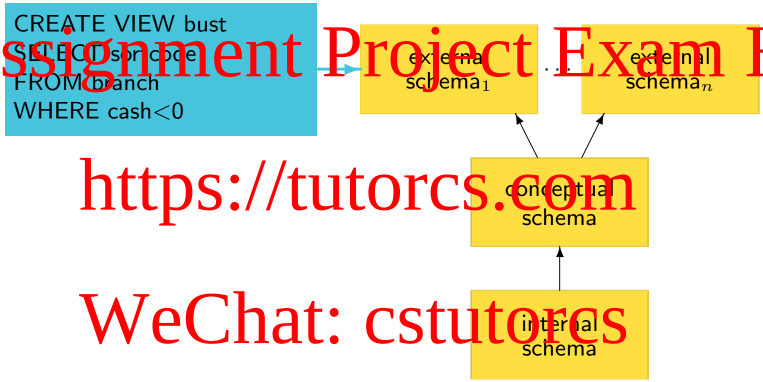
ANSI/SPARC Model (Conceptual Schema)



```
CREATE TABLE branch
(   sortcode INTEGER,
    bname VARCHAR(20),
    cash DECIMAL(10,2)
)

SELECT sortcode
FROM branch
WHERE cash<0
```

- defined in **data definition language** (**DDL**)
- queried using **data manipulation language** (**DML**)
- controlled by **database administrator** (**DBA**)

ANSI/SPARC Model (External Schema)



- Define a schema for a particular user/application

Course Format

Schedule

- Three hours combined lectures/tutorials per week, running into week 10
- Coursework that helps you prepare for the exam
- May Exam

Books

Several good text books on the market. Some that will also cover material in more advanced courses are:

- *Fundamentals of Database Systems*,
  6th Ed, Elmasri and Navathe, Addison Wesley
- *Database Systems: The Complete Book*,
  2nd Ed, Garcia-Molina, Ullman and Widom, Pearson
- *Database Systems*,
  5th Ed, Connolly and Begg, Addison Wesley

Course Resources

Course Web Site

http://www.doc.ic.ac.uk/~pjm/db/

- Lecture slides
- Example Databases
  - Postgres
  - SQL Server

Resources

- **CATe** course work handout and submission
- **Piazza** discussion forum
- **email** course email list

If you are not on Level 2 on CATe, nothing works!

Course Content

Conceptual Layer: Relational Algebra

- SQL
- Datalog

Conceptual Layer: Relational Data Model

- Properties of a ‘good’ schema: keys and normalisation
- Database design using ER models

Physical Layer: Transaction Processing

- Serialisability
- Recovery and Checkpointing