Advanced Databases

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: 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: 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: 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: Poulovassilis, A.	
Wimbledon Branch	Deposit Acc: 10000119	
Sortcode: 55-66-56		
Trans	Amount	Date
1009	5600.00	18/1/1999

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

Data Models

- unstructured-data
- flat files
- semistructured data
- structured data

Flat file: 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: JSON

JSON file for DataTables Javascript Library

```
{ "aoColumns": [{"sTitle":"no","sClass":"right"},
    {"sTitle":"type","sClass":"left"},
    {"sTitle":"cname","sClass":"left"},
    { "sTitle":"rate","sClass":"right"},
    {"sTitle":"sortcode","sClass":"right"}],
  "aaData" : [["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"]]
}
```

Semistructured Data: XML

Fragment of Bank Branch Data as Nested 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⟩
```

Structured Data: Relational Model

branch		
<u>sortcode</u>	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

movement			
<u>mid</u>	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				
<u>no</u>	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) $\stackrel{fk}{\Rightarrow}$ account(no)

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

SQL 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)
```

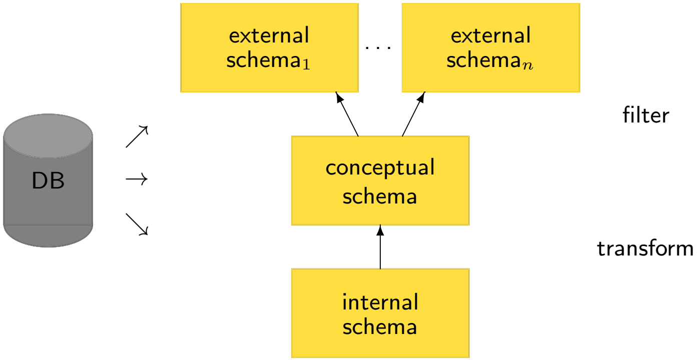
```
CREATE TABLE movement
( mid INTEGER NOT NULL,
  no INTEGER NOT NULL,
  amount DECIMAL(10,2) NOT NULL,
  tdate DATETIME NOT NULL,
  CONSTRAINT movement_pk
      PRIMARY KEY (mid),
  CONSTRAINT movement_fk
      FOREIGN KEY (no) REFERENCES account
)
```

RDBMS Products

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

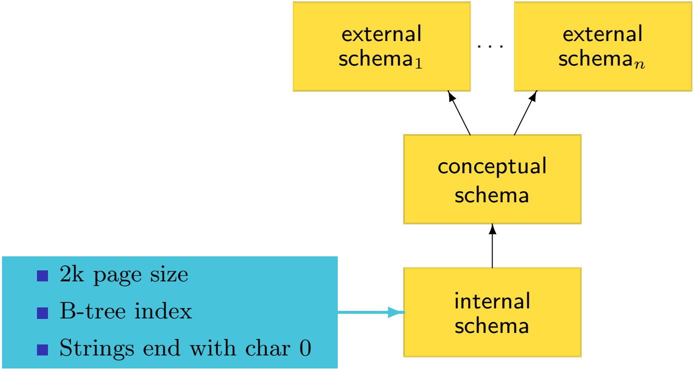
All partially implement ANSI SQL

ANSI/SPARC Model



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

ANSI/SPARC Model (Internal Schema)

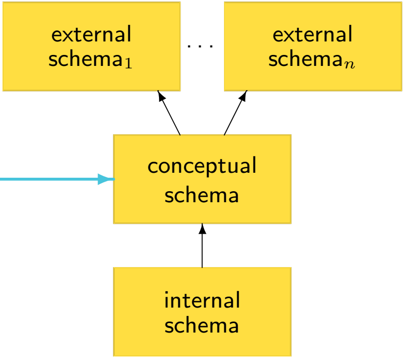


- Describes the physical layout of data

ANSI/SPARC Model (Conceptual Schema)

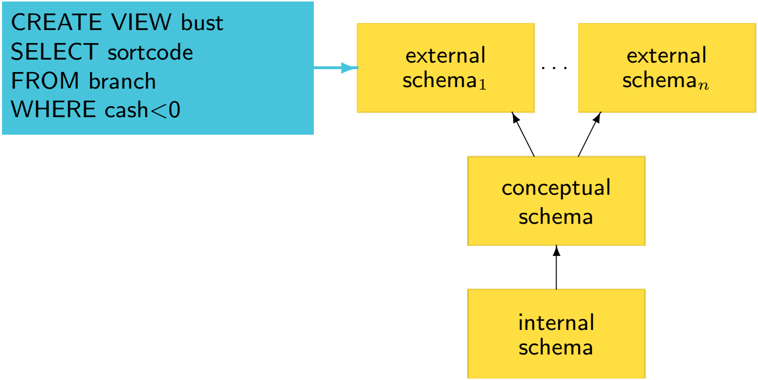
```
CREATE TABLE branch
(    sortcode INTEGER,
     bname VARCHAR(20),
     cash DECIMAL(10,2)
)

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

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 of Transactions

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

Transaction Execution

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

```
BEGIN TRANSACTION T1
    UPDATE branch
    SET cash=cash-10000.00
    WHERE sortcode=56

    UPDATE branch
    SET cash=cash+10000.00
    WHERE sortcode=34
COMMIT TRANSACTION T1
```

⇒

$H_1 = r_1[b_{56}]$, cash=94340.45,
$w_1[b_{56}]$, cash=84340.45,
$r_1[b_{34}]$, cash=8900.67,
$w_1[b_{34}]$, cash=18900.67, c_1

```
BEGIN TRANSACTION T2
    UPDATE branch
    SET cash=cash-2000.00
    WHERE sortcode=34

    UPDATE branch
    SET cash=cash+2000.00
    WHERE sortcode=67
COMMIT TRANSACTION T2
```

⇒

$H_2 = r_2[b_{34}]$, cash=18900.67,
$w_2[b_{34}]$, cash=16900.67,
$r_2[b_{67}]$, cash=34005.00,
$w_2[b_{67}]$, cash=36005.00, c_2

Maintaining the ACID Properties

Some executions maintain the ACID Properties …

$H_{ok} = r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, r_2[b_{34}], w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2$

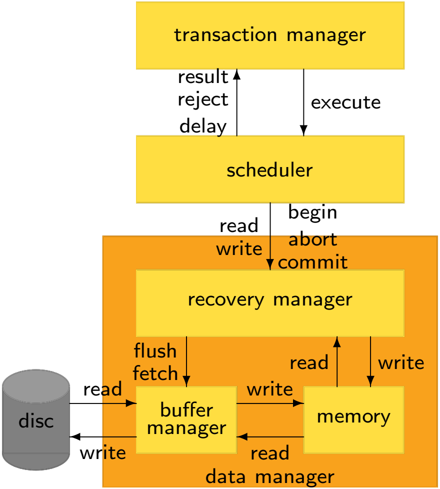
$H_{ok} = r_2[b_{34}], w_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2, c_1$

…others do not

$H_{bad} = r_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, w_2[b_{34}], r_2[b_{67}], w_2[b_{67}], c_2$

$H_{bad} = r_2[b_{34}], w_2[b_{34}], r_1[b_{56}], w_1[b_{56}], r_1[b_{34}], w_1[b_{34}], c_1, r_2[b_{67}], w_2[b_{67}], c_2$

DBMS Architecture



Do you always want a full DBMS?

Providing a full relational DBMS has an implementation overhead.

A **No-SQL** database (there are no standards for No-SQL!) omits one or more of the following:

- ACID properties
- Relational data structures
- Query language

In return, No-SQL systems promise the ability to

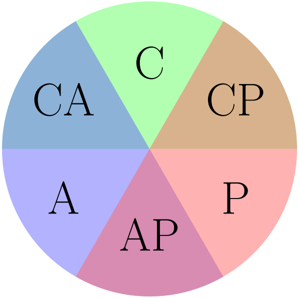
- replicate and fragment data of the large number of servers
- adapt the schema at runtime without performance degradation
- scale to handle very large datasets
- handle network partitions

CAP Theorem

CAP Theorem

No distributed system may maintain all three of

- **Consistency**: all nodes see the same version of data
- **Availability**: the system always responds within fixed upper limits of time
- **Partition Tolerance**: the system always is available even when messages are lost or network failures occur



- CA
  e.g. Centralised Database
- CP
  e.g. Distributed RDBMS
- AP
  e.g. DNS

No-SQL Example: Google Bigtable

Data model is a single large table with names given to each row and column

	contents:	anchor:uk.ac.imperial	anchor:uk.ac.bbk.dcs	language:
uk.ac.ic.doc/pjm	<HTML><HEAD><TITLE>Peter ...	Home Page	Personal website	en
uk.ac.imperial.www/people/p.mcbrien	<HTML><HEAD><TITLE>p.mcbrien			en

⋮

- referencing data involves requesting the value held at a (row,column,timestamp) location in the table
- columns divided into families with names taking form family_name:qualifier
- access to rows is **atomic**
- rows stored in clusters called **tablets**, making access to a group of rows with a lexically close name more efficient

Course Format

Schedule

- Four hours combined lectures/tutorials per week, for Weeks 2 to 9
- Four short courseworks
- Week 11 has the exam (full 2 hour paper)

Books

No one text book covers all the material

- *Fundamentals of Database Systems*,
  7th Ed, Elmasri and Navathe, Pearson
- *Database Systems: The Complete Book*,
  2nd Ed, Garcia-Molina, Ullman and Widom, Pearson
- *Database Systems*,
  6th Ed, Connolly and Begg, Pearson

Course Resources

Course Web Site

`http://www.doc.ic.ac.uk/~pjm/adb/`

- Lecture slides
- Example Databases
  - Postgres
  - SQL Server

Resources

- **CATe** course work handout and submission
- **Panopto** recordings of lectures
- **Piazza** discussion forum
- **email** course email list

If you registered at Level 2 things might not work!

Course Content: Physical Layer

Physical Storage & Indexing

- File organisation & record layout
- B^+-tree Indexes
- Bit-map Indexes

Transaction Processing

- Serialisability
- Recovery
- Checkpointing

Query processing & optimization

- evaluation of relational operators: join algorithms
- query plans and plan selection

Data Distribution

- making a collection of databases behave as one database
- distributed transaction processing
- distributed query processing

Course Content: Conceptual Layer

Data Models + their query languages

- relational
- ER modelling
- temporal data

Distributed Data Processing

- Map Reduce
- Pig