Transactions Recovery

Distributed Databases

CSC 321/621 – 4/5/2012

Project

- Demo Day: 4/26/2012
- Checkin/progress report: 4/19/2012?

From Tuesday

- Notion of recovery
- Database buffers, Flushing
- Checkpoints
- Redo/Undo

- Immediate Update

Actual Recovery Protocols: Immediate Update

- Immediate update:
  - Updates are sent to the database as they occur (our common way of thinking about things)
  - Transaction handling:
    - When transaction starts, write trans start record
    - When a write occurs, write the record to the log file
    - After the log file is written, write the update to the database buffers
    - When a transaction commits, write the trans commit record (or for aborts, a trans abort record)
    - Database buffers flushed when checkpoint encountered

Actual Recovery Protocols: Immediate Update

- Using immediate update,
  - It is possible that data has been written to secondary storage for transactions not committed
  - Has two effects:
    - If an abort occurs for the transaction, use the log and write the before images back to the database, in reverse order (most recent first)
    - If a crash occurs,
      - Redo any transactions that indicate a commit since last checkpoint by re-writing after-image data, in normal oldest to newest order, as recorded in log file
      - Undo any transactions without a commit since last checkpoint by going in reverse and writing before-image data, request that thost transactions restart

Actual Recovery Protocols: Deferred Update

- Deferred update:
  - Updates are not written to the database (even to the database buffers) until a transaction commit occurs
  - Transaction handling:
    - When transaction starts, write trans start record
    - When a write occurs, write the record to the log file
    - When a transaction commits,
      - Write the trans commit record (or for aborts, a trans abort record)
      - Write all log records for that transaction to disk
      - Use the log records to write the actual updates to the database buffers
    - Database buffer writes to disk are performed periodically at checkpoints

Actual Recovery Protocols: Deferred Update

- Using deferred update has two effects:
  - If an abort occurs for the transaction, nothing needs to be done (as nothing has ever actually changed in the database proper)
  - If a crash occurs,
    - Redo any transactions that indicate a commit since last checkpoint by re-writing after-image data, in normal oldest to newest order, as recorded in log file
    - Nothing needs to be done, (i.e. no undo, just ask them to be restarted) for those transactions without a commit since last checkpoint, since no changes were ever really written

Backups Proper

- Helps in the case of secondary storage failures
- Store both copy of database and log file
- Could be incremental
  - Baseline written once, then only save incremental changes

Recovery Practice Problem

- Systems only allow transactions to commit the whole transaction at the end of the transaction, not on individual items as they are finished with in the middle of the transaction?
  - Committing some data in the middle seems like it would reduce later recovery costs – why not use it?

Recovery Practice Problem

- Systems only allow transactions to commit the whole transaction at the end of the transaction, not on individual items as they are finished with in the middle of the transaction?
  - Committing some data in the middle seems like it would reduce later recovery costs – why not use it?

Answer: Violates atomicity of transactions

What if a rollback had to occur? Those things committed couldn't be rolled back – so not ALL OR NOTHING.

Isolation Levels

- It is possible to specify that a transaction is willing to using a less strict transaction-isolation approach.
- SQL defines four transaction levels (not all DB implementations support all 4).
  - Serializable --- what we have discussed so far
  - Repeatable read
  - Read committed
  - Rean uncommitted

Isolation Levels

- Repeatable read
  - Allows phantom reads
    - If a transaction makes the same query twice, receiving multiple rows, it is OK if there are additional rows present in the second result set having been added from a committed other transaction
- Read committed
  - Allows phantom reads and non-repeatable reads
    - NR-read: If a transaction re-reads data that it has previously read, and it has been modified or deleted in the intervening period by another committed transaction
- Read uncommitted
  - Allows phantom reads, non-repeatable reads, and dirty reads
    - Dirty read: A transaction can read data that has been written by an uncommitted transaction

Transaction Support in SQL (MySQL)

- START TRANSACTION or BEGIN
- COMMIT
- ROLLBACK

Example: (Can use at command line, in stored procedures, etc)

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

Transaction Support in SQL (MySQL)

MySQL by default runs in “REPEATABLE READ” mode.

To change: (can’t do in middle of transaction)

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
  {
        REPEATABLE READ
      | READ COMMITTED
      | READ UNCOMMITTED
      | SERIALIZABLE
   }
```

Distributed Databases

- So far, we have thought of databases as such:
  - One machine, one DBMS system, though multi-user
- Desire to support distributing (data/processing) across machines

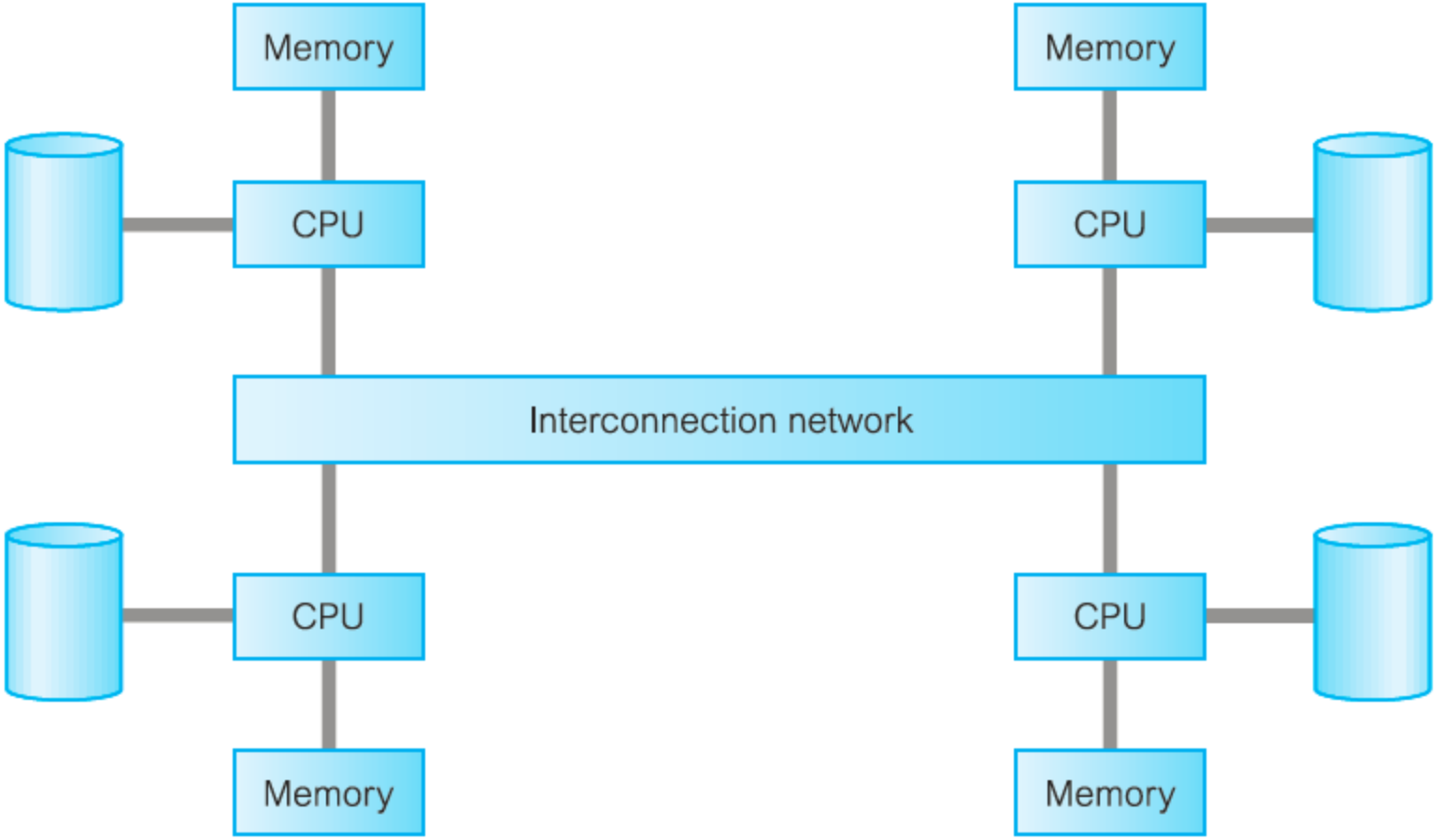
Distributed Databases

- An end-user should be unaware that the database is distributed
  - "Transparent"
  - Called "fundamental principle of distributed database design"
- Now that we have distributed databases, need to think about:
  - How such a design could affect table design issues
  - How such a design affects transaction support
    - Concurrency, Recovery

Distributed Databases vs Parallel Databases

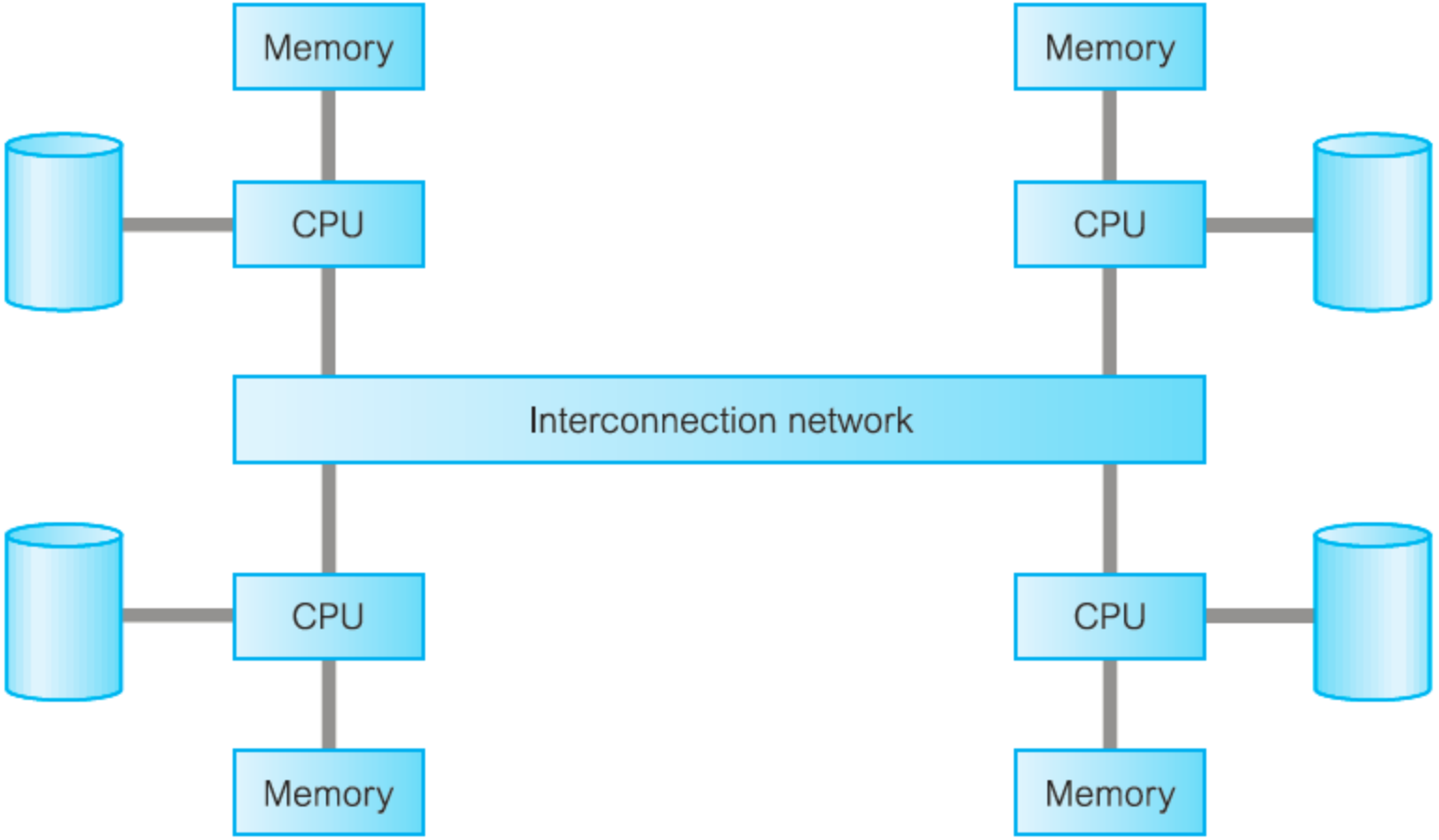
- Distributed databases:
  - Data is physically distributed across a number of sites in the network
  - There is inherent parallelism, but parallelism/extremely fast computation is not the primary goal
- Parallel databases:
  - A DBMS running across multiple processors and disks that is designed to execute operations in parallel, whenever possible, to improve performance

Database Architectures: Shared Nothing



Multiple independent computers connected via a network
Data distributed over disks
Standard model for distributed database, model for some parallel databases

Database Architectures: Shared Nothing



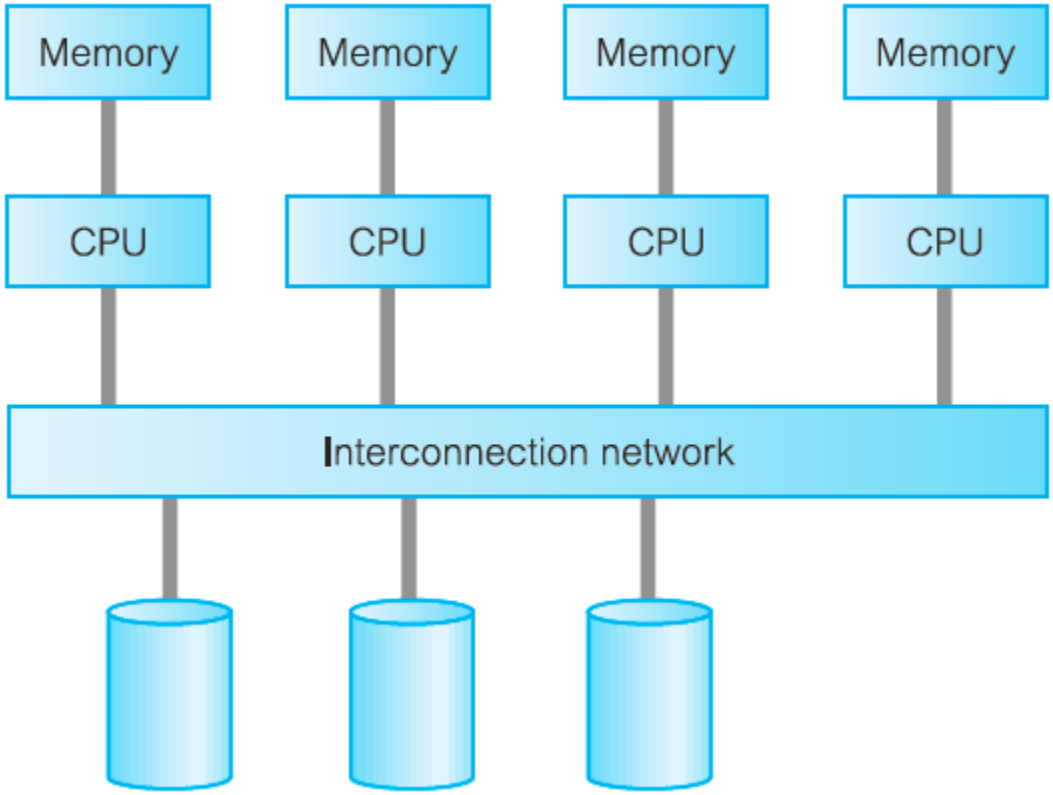
Queries are optimal if local

For parallel DBMS, design of interconnects and data distribution is based exclusively on efficiency

For distributed DBMs, may have arisen organically/fit organization needs

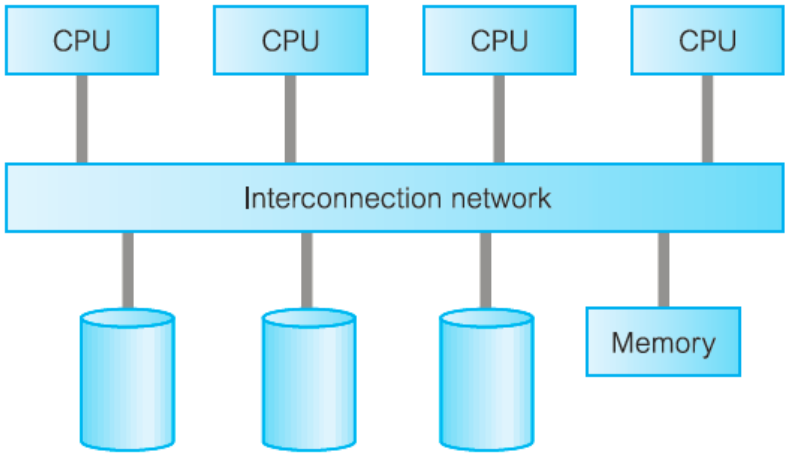
Database Architectures: Shared Disk

- This is the typical “cluster” model
- Common in parallel DBMS



Database Architectures: Shared Memory

- This is the “symmetric multi-processing” setup (one example, multi-core systems)
- Common for parallel DBMS
- Doesn’t scale as easily as shared disk (upper limit on number of processors can squeeze into a given physical space).



Distributed Databases: Advantages

- A few key advantages of using distributed databases:
  - Organizational:
    - Better reflection of organizational structure
    - Local data management/autonomy
  - Computational:
    - Improved availability because of redundant computers and connections
    - Improved reliability due to replication of data
    - Improved performance with local queries over forcing to centralized server
    - Supports modular growth
  - Economic:
    - Monetary cost of transporting data over network and buying commodity machines is << buying massive/mainframe computers

Distributed Databases: Disadvantages

- A few key disadvantages of using distributed databases:
  - Computational:
    - Increased complexity in underlying algorithms & processes
    - Increased potential for security issues – data, local computers, and network must be secure
    - Integrity constraint management usually involves checking against large datasets – not optimal problem for distributed data
    - Requires re-thinking how relations are designed
  - Economic:
    - Typically increased costs in networking and maintenance (labor, parts)
    - There are limited standards and experience in place for distributed databases

Distributed Databases: Terms

- A few more key terms:
  - *Homogenous database*
  - *Heterogenous database*

  - A homogeneous distributed database employs the same DBMS across all nodes, while heterogeneous may employ multiple DBMS
  - Homogeneous typically result from: designing a distributed DBMS from ground up
  - Heterogeneous: cobbling a system together from DBMS already present

We will focus on homogeneous databases…

Distributed Databases: Architectures

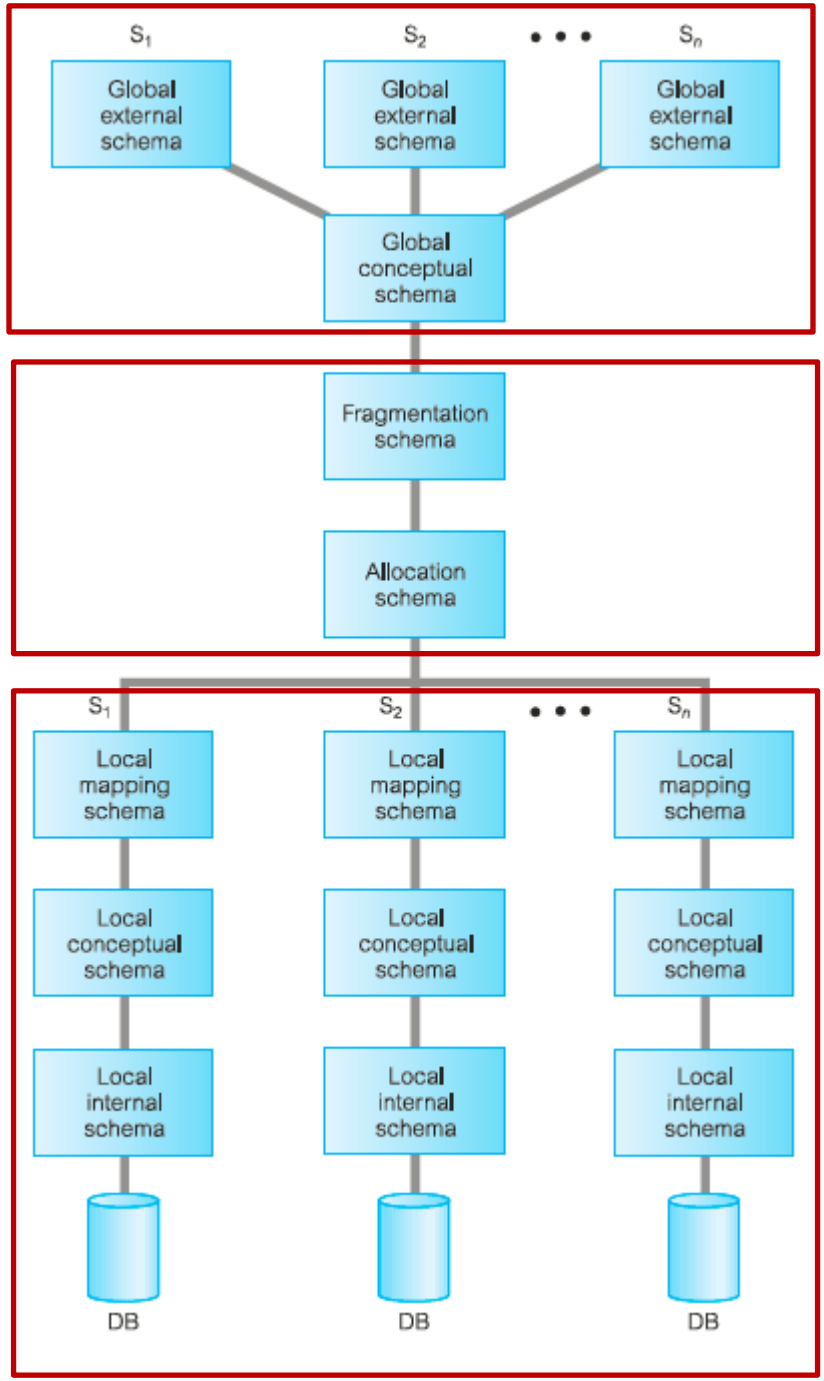
- In addition to supporting all centralized DBMS functions, a DBMS must also support:
  - Network access for remote querying and updates
  - Maintenance of a map of where data is stored
  - Conversion of queries/updates into appropriate distributed queries
  - Extended authorization scheme, supporting multiple hosts
  - Extended concurrency support to maintain consistency of distributed AND replicated data
  - Extended recovery support to manage localized-site and network failures

Distributed Database: Architectures

- Usually requires three levels of schemas:
  - Global schemas
  - Fragmentation/allocation schemas
  - Local schemas

- External vs Internal vs Conceptual Schema
  - External: End-users view of the system (defined by views, security, etc)
  - Conceptual: DBA view – entire set of entities, attributes, relationships; constraints; security information
  - Internal: Physical storage (files, indexes, records)

Distributed Databases: Architecture



Distributed Databases: Architecture

- Global Schemas:
  - Logical description of the entire database, as if not distributed
    - End-user view of the system
  - Entities, relationships, constraints, security, integrity information
- Fragmentation/allocation schemas:
  - Fragmentation: how the data is to be partitioned (broken up)
  - Allocation: where the data resides, including any replication
- Local schemas:
  - Conceptual/internal schemas: Local databases data representation
  - Mapping schemas: Maps fragments in the allocation schema into objects in the database that have been “exposed”

Fragments

- A fragment is a decomposition of a relation (so at the table level)

  Reasonable types of fragments?

  Horizontal (subsets of tuples)

  Vertical (subsets of attributes)

- Fragments are distributed across local databases

Fragmenting Relations

- Already introduced notion of fragmenting relations and distributing over network
  - Horizontal fragmentation – separate by tuples/rows
  - Vertical fragmentation – separate by attributes/columns

Fragmenting Relations

- Any fragmentation of relation R into fragments R1..Rk must obey the following:
  - *Completeness:* Each data item (row, column entry) in R must be able to be found in Ri for some I
  - *Reconstruction:* It must be possible to employ a relational operation that can reconstruct R from R1..Rk
  - *Disjointness:* If a data item appears in fragment Ri, it should not appear in any other fragment (except for primary keys in a vertical fragmentation)

Fragmenting Relations

- Horizontal fragmentation:
  - A fragment consists of a subset of rows/tuples

  - A given fragment is defined by a selection predicate that that performs a restriction on the tuples
  - A fragmentation is thus defined by a set of selection predicates/selection operations

Horizontal Fragmentation Example

Student

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1129	Jones	Douglas	3	MTH	2.9
1145	Brady	Susan	4	CSC	3.8

Student1 – SELECT * FROM Student WHERE major='CSC';

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1145	Brady	Susan	4	CSC	3.8

Student2 – SELECT * FROM Student WHERE major!='CSC';

studentID	lastName	firstName	year	major	GPA
1129	Jones	Douglas	3	MTH	2.9

Horizontal Fragmentation Example

Student

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1129	Jones	Douglas	3	MTH	2.9
1145	Brady	Susan	4	CSC	3.8

What are other reasonable fragmentations of such a relation?

Horizontal Fragmentation Correctness

Student

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1129	Jones	Douglas	3	MTH	2.9
1145	Brady	Susan	4	CSC	3.8

Student1 – SELECT * FROM Student WHERE major='CSC';

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1145	Brady	Susan	4	CSC	3.8

Student2 – SELECT * FROM Student WHERE major!='CSC';

studentID	lastName	firstName	year	major	GPA
1129	Jones	Douglas	3	MTH	2.9

Is this fragmentation:

Complete?

Reconstructable?

Disjoint?

Horizontal Fragmentation Correctness

Student

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1129	Jones	Douglas	3	MTH	2.9
1145	Brady	Susan	4	CSC	3.8

Student1 – SELECT * FROM Student WHERE major='CSC';

studentID	lastName	firstName	year	major	GPA
1123	Smith	Robert	4	CSC	3.5
1145	Brady	Susan	4	CSC	3.8

Student2 – SELECT * FROM Student WHERE major!='CSC';

studentID	lastName	firstName	year	major	GPA
1129	Jones	Douglas	3	MTH	2.9

Is this fragmentation:

Complete? Yes (all 3 tuples in Student appear in Student1 or 2)

Reconstructable? Yes (using UNION relational operation)

Disjoint? Yes, given selection condition (=, !=)

Fragmenting Relations

- Vertical fragmentation:
  - A fragment consists of a subset of columns/attributes

  - A given fragment is defined by a projection
  - A fragmentation is thus defined by a set of projections
  - OK to share primary key across projections (actually need this to support reconstruction)

Vertical Fragmentation Example

Fragment S_1

staffNo	position	sex	DOB	salary
SL21	Manager	M	1-Oct-45	30000
SG37	Assistant	F	10-Nov-60	12000
SG14	Supervisor	M	24-Mar-58	18000
SA9	Assistant	F	19-Feb-70	9000
SG5	Manager	F	3-Jun-40	24000
SL41	Assistant	F	13-Jun-65	9000

Fragment S_2

staffNo	fName	lName	branchNo
SL21	John	White	B005
SG37	Ann	Beech	B003
SG14	David	Ford	B003
SA9	Mary	Howe	B007
SG5	Susan	Brand	B003
SL41	Julie	Lee	B005

Original Staff table had 8 attributes, and the six rows represented:

staffNo, position, sex, DOB,salary, fname, lName, branchNo

Payroll needs access to S1
BuildingSecurity needs access to S2

Fragmenting Relations

The definition of fragments makes me immediately think of something other topic we have covered where we dealt with subsetted data….

Which topic, and how is this different?

VIEWS!

Vertical Fragmentation Example

Fragment S_1

staffNo	position	sex	DOB	salary
SL21	Manager	M	1-Oct-45	30000
SG37	Assistant	F	10-Nov-60	12000
SG14	Supervisor	M	24-Mar-58	18000
SA9	Assistant	F	19-Feb-70	9000
SG5	Manager	F	3-Jun-40	24000
SL41	Assistant	F	13-Jun-65	9000

Fragment S_2

staffNo	fName	lName	branchNo
SL21	John	White	B005
SG37	Ann	Beech	B003
SG14	David	Ford	B003
SA9	Mary	Howe	B007
SG5	Susan	Brand	B003
SL41	Julie	Lee	B005

Is this fragmentation valid?

Complete?
Reconstructable?
Disjoint?

Vertical Fragmentation Example

Fragment S_1

staffNo	position	sex	DOB	salary
SL21	Manager	M	1-Oct-45	30000
SG37	Assistant	F	10-Nov-60	12000
SG14	Supervisor	M	24-Mar-58	18000
SA9	Assistant	F	19-Feb-70	9000
SG5	Manager	F	3-Jun-40	24000
SL41	Assistant	F	13-Jun-65	9000

Fragment S_2

staffNo	fName	lName	branchNo
SL21	John	White	B005
SG37	Ann	Beech	B003
SG14	David	Ford	B003
SA9	Mary	Howe	B007
SG5	Susan	Brand	B003
SL41	Julie	Lee	B005

Is this fragmentation valid?

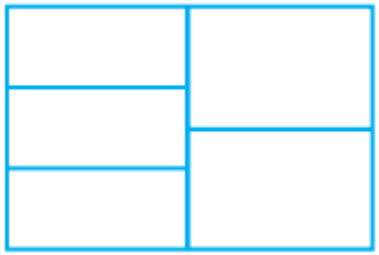
Complete? Yes, as each of the original attributes appears in at least one of the decomposed relations

Reconstructable? Yes, via a join

Disjoint? Yes for data items, as the attribute sets are disjoint (no attribute appears in both)
Ignore primary key redundancy

Fragmenting Relations

- Mixed fragmentation:
  - A horizontal fragmentation of a vertical fragmentation OR
  - A vertical fragmentation of a horizontal fragmentation



  - A given fragment is defined by a projection and selection
  - A fragmentation is thus defined by a set of (projection, selection pairs)
  - OK to share primary key across projections (actually need this to support reconstruction)

Mixed Fragmentation

Fragment S_1

staffNo	position	sex	DOB	salary
SL21	Manager	M	1-Oct-45	30000
SG37	Assistant	F	10-Nov-60	12000
SG14	Supervisor	M	24-Mar-58	18000
SA9	Assistant	F	19-Feb-70	9000
SG5	Manager	F	3-Jun-40	24000
SL41	Assistant	F	13-Jun-65	9000

Fragment S_{21}

staffNo	fName	lName	branchNo
SG37	Ann	Beech	B003
SG14	David	Ford	B003
SG5	Susan	Brand	B003

Fragment S_{22}

staffNo	fName	lName	branchNo
SL21	John	White	B005
SL41	Julie	Lee	B005

Fragment S_{23}

staffNo	fName	lName	branchNo
SA9	Mary	Howe	B007

Original Staff table had 8 attributes, and the six rows represented:

staffNo, position, sex, DOB,salary, fname, lName, branchNo

Payroll needs access to S1 for everyone
BuildingSecurity needs access to S2,
Security provided per-branch

S21 is defined by Projection to {staffNo, fName, lName, branchNo} then Selection on branchNo='B003'

Mixed Fragmentation

Fragment S_1

staffNo	position	sex	DOB	salary
SL21	Manager	M	1-Oct-45	30000
SG37	Assistant	F	10-Nov-60	12000
SG14	Supervisor	M	24-Mar-58	18000
SA9	Assistant	F	19-Feb-70	9000
SG5	Manager	F	3-Jun-40	24000
SL41	Assistant	F	13-Jun-65	9000

Fragment S_{21}

staffNo	fName	lName	branchNo
SG37	Ann	Beech	B003
SG14	David	Ford	B003
SG5	Susan	Brand	B003

Fragment S_{22}

staffNo	fName	lName	branchNo
SL21	John	White	B005
SL41	Julie	Lee	B005

Fragment S_{23}

staffNo	fName	lName	branchNo
SA9	Mary	Howe	B007

Is this a valid fragmentation?

Completeness?
Reconstructable?
Disjoint?

Mixed Fragmentation

Fragment S_1

staffNo	position	sex	DOB	salary
SL21	Manager	M	1-Oct-45	30000
SG37	Assistant	F	10-Nov-60	12000
SG14	Supervisor	M	24-Mar-58	18000
SA9	Assistant	F	19-Feb-70	9000
SG5	Manager	F	3-Jun-40	24000
SL41	Assistant	F	13-Jun-65	9000

Fragment S_{21}

staffNo	fName	lName	branchNo
SG37	Ann	Beech	B003
SG14	David	Ford	B003
SG5	Susan	Brand	B003

Fragment S_{22}

staffNo	fName	lName	branchNo
SL21	John	White	B005
SL41	Julie	Lee	B005

Fragment S_{23}

staffNo	fName	lName	branchNo
SA9	Mary	Howe	B007

Is this a valid fragmentation?

Completeness? Yes all information from all six tuples, 8 attributes still present

Reconstructable? Yes, using union and join (S21 U S22 U S23) JOIN S1

Disjoint? Yes, S1 and S2 are disjoint except primary key; S21, S22, S23 are disjoint assuming no staff member works at two branches

Derived Horizontal Fragmentation

- A fragmentation based off of another fragmentation
- Example:

Hotel

hotelNumber	hotelName	city

Booking

hotelNumber	guestNumber	dateFrom	dateTo	roomNumber

We may horizontally fragment hotel by city, so that our Winston-Salem HQ manages W-S hotels.

Makes sense to fragment Booking by hotelNumber so that W-S hotel bookings are stored at the same site as the hotel information.

Derived Horizontal Fragmentation

- Assume parent relation is the first fragmented and child relation is the relation to fragment next.
  - Child relation has a FK reference to parent.

- If parent is fragmented into S1..Sk, then define child fragmentation as:
- Ri = R semijoin Si
  - *What is a semijoin again? Cartesian product, followed by selection based on shared values for a column (so a join), followed by dropping the attributes related to the RHS (the Si attributes) to leave only attributes from Ri*

Derived Horizontal Fragmentation

- Is a fragmentation (on the child relationship) designed in such a way guaranteed to be valid?
  - *Complete?*
  - *Reconstructable?*
  - *Disjoint?*

Derived Horizontal Fragmentation

- Is a fragmentation (on the child relationship) designed in such a way guaranteed to be valid?
  - *Complete? Yes, under definition of foreign keys (must tie back to value in parent table) and an assumption of no NULL foreign keys*
  - *Reconstructable? Yes, using union (since this is just a horizontal fragmentation)*
  - *Disjoint? Yes, under the fact that a foreign key reference can be to only one value*

STOP!

- That’s it for today – we will cover next week how concurrency and recovery work for distributed databases.