Data Fragmentation

- Division of relation r into fragments r_1, r_2, …, r_n which contain sufficient information to reconstruct relation r.
- **Horizontal fragmentation**: each tuple of *r* is assigned to one fragment
- **Vertical fragmentation**: the schema for relation *r* is split into several smaller schemas
  - All schemas must contain a common candidate key (or superkey) to ensure lossless join property.
  - A special attribute, the tuple-id attribute may be added to each schema to serve as a candidate key.

- Example : relation account with following schema
- *Account* = (*account_number, branch_name , balance*)

Horizontal Fragmentation of *account* Relation

account_number	*branch_name*	*balance*
A-305	Hillside	500
A-226	Hillside	336
A-155	Hillside	62

$$account_1 = \sigma_{branch\_name=\text{“Hillside”}}(account)$$

account_number	*branch_name*	*balance*
A-177	Valleyview	205
A-402	Valleyview	10000
A-408	Valleyview	1123
A-639	Valleyview	750

$$account_2 = \sigma_{branch\_name=\text{“Valleyview”}}(account)$$

Vertical Fragmentation of *deposit* Relation

branch_name	*customer_name*	*tuple_id*
Hillside	Lowman	1
Hillside	Camp	2
Valleyview	Camp	3
Valleyview	Kahn	4
Hillside	Kahn	5
Valleyview	Kahn	6
Valleyview	Green	7

$deposit_1 = \Pi_{branch\_name,\ customer\_name,\ tuple\_id}\ (deposit\ )$

account_number	*balance*	*tuple_id*
A-305	500	1
A-226	336	2
A-177	205	3
A-402	10000	4
A-155	62	5
A-408	1123	6
A-639	750	7

$deposit_2 = \Pi_{account\_number,\ balance,\ tuple\_id}\ (deposit\ )$

Advantages of Fragmentation

- Horizontal:
  - allows parallel processing on fragments of a relation
  - allows a relation to be split so that tuples are located where they are most frequently accessed
- Vertical:
  - allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed
  - tuple-id attribute allows efficient joining of vertical fragments
- Vertical and horizontal fragmentation can be mixed.
  - Fragments may be successively fragmented to an arbitrary depth.
- Replication and fragmentation can be combined
  - Relation is partitioned into several fragments: system maintains several identical replicas of each such fragment.

Data Transparency

- **Data transparency**: Degree to which system user may remain unaware of the details of how and where the data items are stored in a distributed system
- Consider transparency issues in relation to:
  - Fragmentation transparency
  - Replication transparency
  - Location transparency
- Naming of data items: criteria
  1. Every data item must have a system-wide unique name.
  2. It should be possible to find the location of data items efficiently.
  3. It should be possible to change the location of data items transparently.
  4. Each site should be able to create new data items autonomously.

Query Transformation

- Translating algebraic queries on fragments.
  - It must be possible to construct relation *r* from its fragments
  - Replace relation *r* by the expression to construct relation *r* from its fragments
- Consider the horizontal fragmentation of the *account* relation into

  $account_1 = \sigma_{branch\_name = \text{“Hillside”}}(account)$

  $account_2 = \sigma_{branch\_name = \text{“Valleyview”}}(account)$
- The query $\sigma_{branch\_name = \text{“Hillside”}}(account)$ becomes

  $\sigma_{branch\_name = \text{“Hillside”}}(account_1 \cup account_2)$

  which is optimized into

  $\sigma_{branch\_name = \text{“Hillside”}}(account_1) \cup \sigma_{branch\_name = \text{“Hillside”}}(account_2)$

Example Query (Cont.)

- Since $account_1$ has only tuples pertaining to the Hillside branch, we can eliminate the selection operation.
- Apply the definition of $account_2$ to obtain

  $\sigma_{branch\_name = \text{“Hillside”}} (\sigma_{branch\_name = \text{“Valleyview”}} (account))$
- This expression is the empty set regardless of the contents of the *account* relation.
- Final strategy is for the Hillside site to return $account_1$ as the result of the query.