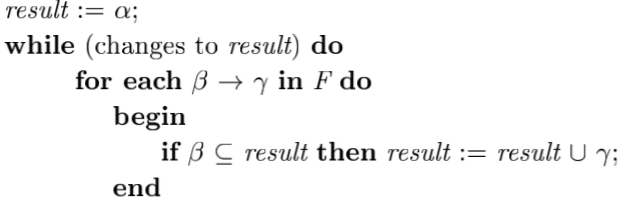
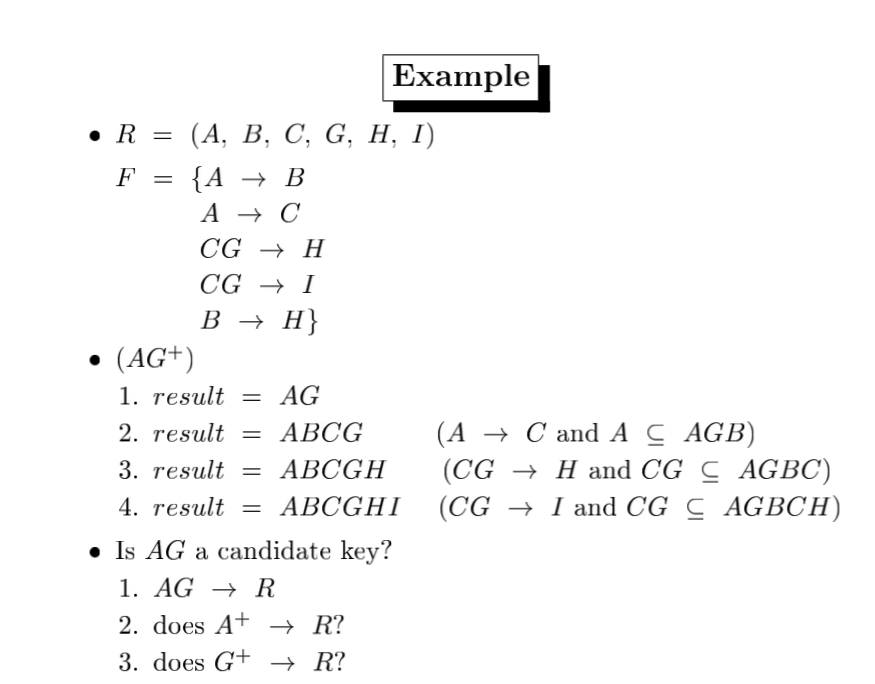
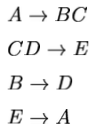
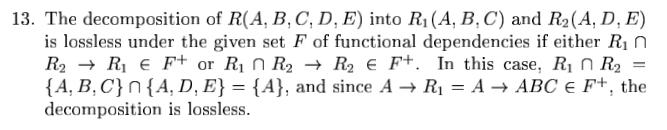
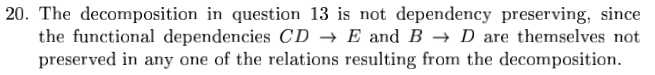
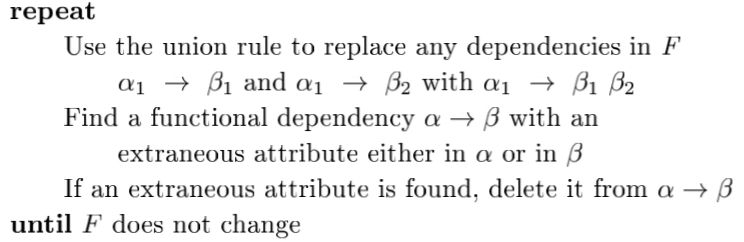
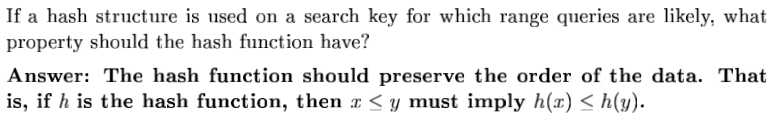
**Calculating Closure**

****

****

**Lossless Decomposition ex.**

**Dependency Preserving ex.**

**Calculating Canonical Cover**

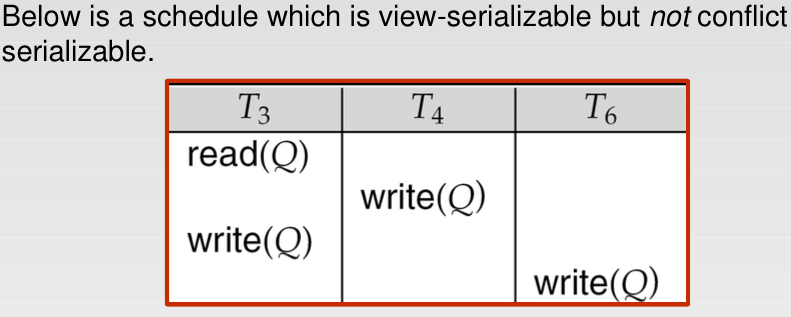
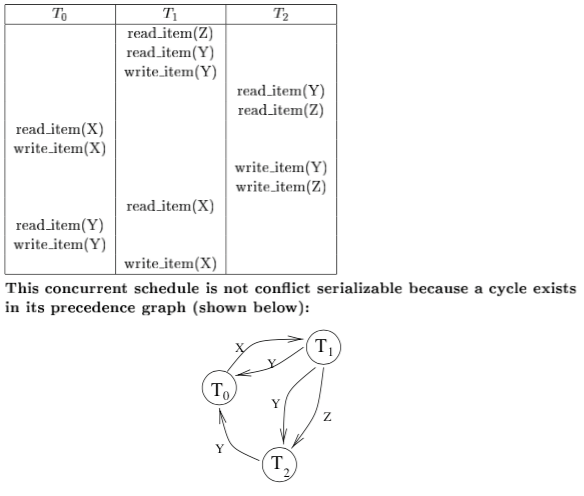
**For query trees (optimization)**

* Perform selection and projection early
* Perform most restrictive selection and join operations before other similar operations.

**Views**

* Pros: supports multiple user perspectives on the DB, avoiding data duplication and consistency problems.
  + Security + privacy concerns
* Cons: Join fail because join on NULL attr.
  + Problem when view is defined as an aggregate quantity
* Updates rarely allowed, b/c of consistency concerns

**ACID (Transactions)**

* Atomicity – either all operations of transitions are properly reflected in the DB, or none are.
* Consistency – execution of a transaction in isolation preserves DB’s consistency.
* Isolation – although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate results must be hidden from other executing transactions.
* Durability – after transaction succeeds, the change it made to the DB persists, even if DB fails.
* **Serial Schedule** – transaction set goes one after another in order.
* **Serializable Schedule** – transaction set that is not in chronological order, but is equivalent.
  + **Conflict Serializable** (conflicts if two instructions of two transactions access some item and at least one of them wrote to that item.) a schedule that is equivalent to serial schedule after swapping non-conflicting instructions.
  + **View Serializable**
  + Every conflict serializable schedule is also view serializable
* **Precedence Graph –** use to test for conflict serializability. Only CSeri if graph is acyclic.
* **Concurrency-control protocols** DB must provide way to ensure all schedules are either conflict or view serializable, and are recoverable and preferably cascadeless (no rollback if fail of transaction).

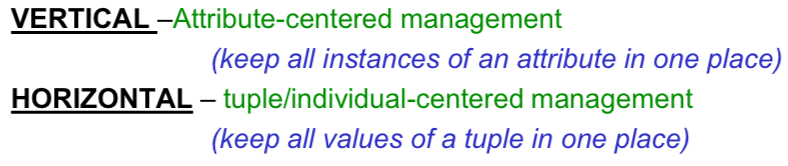
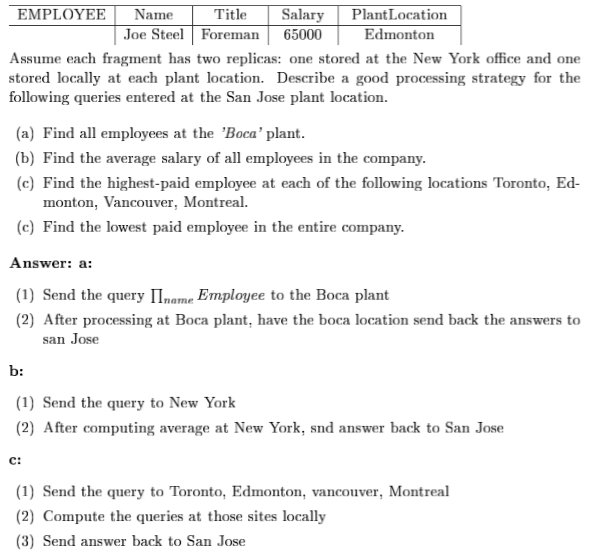
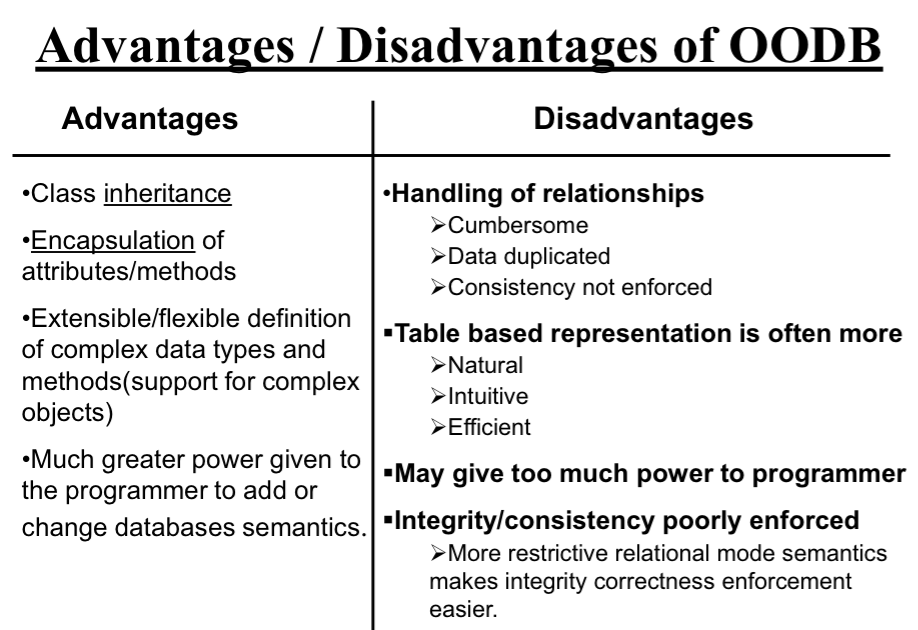
**Recovery**

* **Log Based Recovery** log file kept on stable storage, when transaction starts, registers on log. Records info about transaction, then before committing, will log commit.
* **Checkpoints** streamline recovery procedure by periodically checkpointing. Think video game.
* **Shadow Paging** have two page tables for a transaction, current and shadow, shadow on nonvolatile storage. State of DB prior to transaction thus can be recovered. Cons: data fragmentation, garbage collection.

**Advantages of Distributed DBs**

* **Interconnectivity** of pre-existing databases
* **Expandability** – doesn’t need to replace whole system to grow
* **Cost** – many small engines cheaper than a whole mainframe
* **Performance** – place data near where it is used
* **Availability** – and reliability

**Disadvantages**

* **Maintaining data consistency**
* **Distributed directory management**
* **Security**
* **Heterogeneous databases**
* **Data replication and/or fragmentation**
  + **Replication** increases availability and parallelism, but overhead and consistency issues.
  + **Fragmentation** issues are completeness and reconstruction.