Recovery, cont

Checkpointing

- How far should we go back in the log while constructing redo and undo lists??
  - It is possible that a transaction made an update at the very beginning of the system, and that update never made it to disk
    - very very unlikely, but possible (because we don’t do force)
  - For correctness, we have to go back all the way to the beginning of the log
  - Bad idea !!

- Checkpointing is a mechanism to reduce this
Checkpointing

- Periodically, the database system writes out everything in the memory to disk
  - Goal is to get the database in a state that we know (not necessarily consistent state)
- Steps:
  - Stop all other activity in the database system
  - Write out the entire contents of the memory to the disk
    - Only need to write updated pages, so not so bad
    - Entire === all updates, whether committed or not
  - Write out all the log records to the disk
  - Write out a special log record to disk
    - <CHECKPOINT LIST_OF_ACTIVE_TRANSACTIONS>
    - The second component is the list of all active transactions in the system right now
  - Continue with the transactions again

Restart Recovery w/ checkpoints

- Key difference: Only need to go back till the last checkpoint
- Steps:
  - undo_list:
    - Go back till the checkpoint as before.
    - Add all the transactions that were active at that time, and that didn’t commit
      - e.g. possible that a transactions started before the checkpoint, but didn’t finish till the crash
  - redo_list:
    - Similarly, go back till the checkpoint constructing the redo_list
    - Add all the transactions that were active at that time, and that did commit
  - Do UNDOs and REDOas as before
Recap so far …

- **Log-based recovery**
  - Uses a log to aid during recovery

- **UNDO()**
  - Used for normal transaction abort/rollback, as well as during restart recovery

- **REDO()**
  - Used during restart recovery

- **Checkpoints**
  - Used to reduce the restart recovery time

Other issues

- **ARIES**: Considered the canonical description of log-based recovery
  - Used in most systems
  - Has many other types of log records that simplify recovery significantly

- **Loss of disk**:
  - Can use a scheme similar to checkpointing to periodically dump the database onto tapes or optical storage
  - Techniques exist for doing this while the transactions are executing (called fuzzy dumps)

- **Shadow paging**:
  - Read up
Recap

- STEAL vs NO STEAL, FORCE vs NO FORCE
  - We studied how to do STEAL and NO FORCE through log-based recovery scheme

<table>
<thead>
<tr>
<th>Force</th>
<th>No Force</th>
<th>Force</th>
<th>No Force</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Steal</td>
<td>Desired</td>
<td>No Steal</td>
<td>UNDO</td>
</tr>
<tr>
<td>Steal</td>
<td>Trivial</td>
<td>Steal</td>
<td>REDO</td>
</tr>
</tbody>
</table>

Write-ahead logging

- So far assumed that log records are written to disk as soon as generated
  - Too restrictive
- Write-ahead logging:
  - Before an update on a data item (say A) makes it to disk, the log records referring to the update must be forced to disk
  - How?
    - Each log record has a log sequence number (LSN)
      - Monotonically increasing
    - For each page in the memory, we maintain the LSN of the last log record that updated a record on this page
      - pageLSN
    - If a page $P$ is to be written to disk, all the log records till $\text{pageLSN}(P)$ are forced to disk first
Write-ahead logging

- Write-ahead logging (WAL) is sufficient for all our purposes
  - All the algorithms discussed before work

- Note the special case:
  - A transaction is not considered committed unless the \(<T, \text{commit}>\) record is on disk

Other issues

- The system halts during checkpointing
  - Not acceptable
  - Advanced recovery techniques allow the system to continue processing while checkpointing is going on

- System may crash during recovery
  - Our simple protocol is actually fine
  - In general, this can be painful to handle

- B+-Tree and other indexing techniques
  - Strict 2PL is typically not followed (we didn’t cover this)
  - So physical logging is not sufficient; must have logical logging
    - Read 16.7 if interested.
Recap

- **ACID Properties**
  - Atomicity and Durability:
    - Logs, undo(), redo(), WAL etc
  - Consistency and Isolation:
    - Concurrency schemes
  - Strong interactions:
    - We had to assume Strict 2PL for proving correctness of recovery

Topics

- OLAP/Data Warehouses
- Distributed Systems
Topics

- Object Oriented, Object Relational
- Client-server, Parallel, Distributed Systems
- OLAP/Data Warehouses
- Information Retrieval
- Cloud Computing
  - Data centers, Map-reduce, NoSQL Systems

OLAP

- On-line Analytical Processing
- Why?
  - Exploratory analysis
  - Interactive
  - Different queries than typical SPJ SQL queries
  - Data CUBE
    - A summary structure used for this purpose
      - E.g. *give me total sales by zipcode; now show me total sales by customer employment category*
    - Much much faster than using SQL queries against the raw data
      - The tables are *huge*
- Applications:
  - Sales reporting, Marketing, Forecasting etc etc
Data Warehouses

- A repository of integrated information for querying and analysis purposes
- A (usually) stand-alone system that integrates data from everywhere
  - Read-only, typically not kept up-to-date with the real data
  - Geared toward business analytics, data mining etc...
  - HUGE market today
- Heavily optimized
  - Specialized query processing and indexing techniques are used
  - High emphasis on pre-computed data structures like summary tables, data cubes
- Analysis cycle:
  - Extract data from databases with queries, visualize/analyze with desktop tools
  - E.g., Tableau

![Data Warehousing Architecture](image)

**Figure 1. Data Warehousing Architecture**
Data Warehouses

Query processing algorithms heavily optimized for these types of schemas

**Many queries of the type:**

- Selections on dimension tables (e.g., state = ‘MD’)
- Join fact table with dimension tables
- Aggregate on a “measure” attribute (e.g., Quantity, TotalPrice)

**For example:**

```
select c_city, o_year, SUM(quantity)
from Fact, Customer, Product
where p_category = 'Tablet';
```

Need Generalized SQL Groupbys

- **drill-down and roll-up**

**Table 3:** Sales Roll-up by Model by Year by Color

<table>
<thead>
<tr>
<th>Model</th>
<th>Year</th>
<th>Color</th>
<th>Sales by Model by Year by Color</th>
<th>Sales by Model by Year</th>
<th>Sales by Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chevy</td>
<td>1994</td>
<td>black</td>
<td>00</td>
<td>50</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td></td>
<td>white</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1995</td>
<td>black</td>
<td>85</td>
<td></td>
<td>115</td>
</tr>
<tr>
<td></td>
<td></td>
<td>white</td>
<td>115</td>
<td></td>
<td>290</td>
</tr>
</tbody>
</table>

**Table 4:** Sales Summary

<table>
<thead>
<tr>
<th>Model</th>
<th>Year</th>
<th>Color</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chevy</td>
<td>1994</td>
<td>black</td>
<td>50</td>
</tr>
<tr>
<td>Chevy</td>
<td>1994</td>
<td>white</td>
<td>40</td>
</tr>
<tr>
<td>Chevy</td>
<td>1994</td>
<td>ALL</td>
<td>90</td>
</tr>
<tr>
<td>Chevy</td>
<td>1995</td>
<td>black</td>
<td>85</td>
</tr>
<tr>
<td>Chevy</td>
<td>1995</td>
<td>white</td>
<td>115</td>
</tr>
<tr>
<td>Chevy</td>
<td>1995</td>
<td>ALL</td>
<td>200</td>
</tr>
<tr>
<td>Chevy</td>
<td>ALL</td>
<td>ALL</td>
<td>290</td>
</tr>
</tbody>
</table>

Not relational (null values in the keys)

```
SELECT Model, ALL, ALL, SUM(Sales)
FROM Sales
WHERE Model = 'Chevy'
GROUP BY Model
UNION
SELECT Model, Year, ALL, SUM(Sales)
FROM Sales
WHERE Model = 'Chevy'
GROUP BY Model, Year
UNION
SELECT Model, Year, Color, SUM(Sales)
FROM Sales
WHERE Model = 'Chevy'
GROUP BY Model, Year, Color;
```
More problems with Groupbys

- cross-tabulation (spreadsheets)

<table>
<thead>
<tr>
<th></th>
<th>1994</th>
<th>1995</th>
<th>total (ALL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>black</td>
<td>50</td>
<td>85</td>
<td>135</td>
</tr>
<tr>
<td>white</td>
<td>40</td>
<td>115</td>
<td>155</td>
</tr>
<tr>
<td>total (ALL)</td>
<td>90</td>
<td>200</td>
<td>290</td>
</tr>
</tbody>
</table>

- even if SQL syntax can be devised, a 6D cross-tab requires 64 groupby queries to generate it and 64 scans and sorts of the data

- most of these are not relational expressions but are in many report writers

CUBE: Aggregate Operator Generalizing Group By
Data Mining

- **Searching for patterns in data**
  - Typically done in data warehouses

- **Association Rules:**
  - When a customer buys X, she also typically buys Y
  - Use ?
    - Move X and Y together in supermarkets
    - A customer buys a lot of shirts
    - Send him a catalogue of shirts
  - Patterns are not always obvious
    - Classic example: It was observed that men tend to buy *beer* and *diapers* together (may be an urban legend)

- **Other types of mining**
  - Classification
  - Decision Trees
Data Warehouses

● Data analytics a major industry right now, and likely to grow in near future
  ● BIG Data !!
  ● Extracting (actionable) knowledge from data really critical
    ● Especially in real-time

● Some key technologies:
  ● Parallelism – pretty much required
  ● Column-oriented design
    ● Lay out the data column-by-column, rather than row-by-row
  ● Heavy pre-computation (like Cubes)
  ● New types of indexes
    ● Focusing on bitmap representations
  ● Heavy compression
  ● Map-reduce??