Overview of the next two classes

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)
- Potpourri

Levels of Abstraction

- A major purpose of a DB management system is to provide an abstract view of the data.

  Three abstraction levels:
  - Physical level: how data is actually stored
  - Conceptual (or Logical) level: how data is perceived by the users
  - External (or View) level: describes part of the database to different users
    - Convenience, security, etc.
    - E.g., views of student, registrar, & database admin.

Schema and Instances

- We’ll start with the schema – the logical structure of the database (e.g., students take courses)
  - Conceptual (or logical) schema: db design at the logical level
  - Physical schema: db design at the physical level; indexes, etc.
- Later we’ll populate instances – content of the database at a particular point in time
  - E.g., currently there are no grades for CPSC 504
- Physical Data Independence – ability to modify physical schema without changing logical schema
  - Applications depend on the conceptual schema
- Logical Data Independence – Ability to change conceptual scheme without changing applications
  - Provided by views

Conceptual Database Design

- What are the entities and relationships involved?
  - Entities are usually nouns, e.g., "course"-"prof"
- Relationships are statements about 2 or more objects. Often, verbs, e.g., “a prof teaches a course”
- What information about these entities and relationships should we store in the database?
- What integrity constraints or other rules hold?
- In relational databases, this is generally created in an Entity-Relationship (ER) Diagram
Entity / Relationship Diagrams

- **Entities**
  - Product

- **Attributes**
  - address

- **Relationships between entities**
  - buys

Keys in E/R Diagrams

- Every entity set must have a key which is identified by an underline

Roles in Relationships

- What if we need an entity set twice in one relationship?

Attributes on Relationships

Subclasses in E/R Diagrams

- Software Product
  - platforms

- Educational Product
  - Age Group
Summarizing ER diagrams

- Basics: entities, relationships, and attributes
- Also showed inheritance
- Has things other things like cardinality
- Used to design databases...

But how do you store data in them?

Overview of the next two classes

- Entity Relationship (ER) diagrams
- Relational databases
  - How did we get here?
  - What’s in a relational schema?
  - From ER to relational
  - Query Languages
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)
- Potpourri

How did we get the relational model?

- Before the relational model, there were two main contenders
  - Network databases
  - Hierarchical databases
- Network databases had a complex data model
- Hierarchical databases integrated the application in the data model

Example Hierarchical Model

Example IMS (Hierarchical) query: Print the names of all the provinces admitted during a Liberal Government

Relational model to the rescue!

- Introduced by Edgar Codd (IBM) in 1970
- Most widely used model today.
  - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- Former Competitor: object-oriented model
  - ObjectStore, Versant, Ontos
- A synthesis emerged: object-relational model
  - Informix Universal Server, UniSQL, O2, Oracle, DB2
- Recent competitor: XML data model
Key points of the relational model
- Exceedingly simple to understand – main abstraction is a table
- Query language separate from application language
  - General form is simple
  - Many bells and whistles

Structure of Relational Databases
- **Relational database**: a set of relations
- **Relation**: made up of 2 parts:
  - **Schema**: specifies name of relation, plus name and domain (type) of each field (or column or attribute), e.g., `Student(id: string, name: string, major: string)`.  
  - **Instance**: a table, with rows and columns.
  - #Rows = cardinality, #Fields = dimension / arity
- **Relational Database Schema**: collection of schemas in the database
- **Database Instance**: a collection of instances of its relations (e.g., currently no grades in CPSC 504)

Example of a Relation Instance

<table>
<thead>
<tr>
<th>Product</th>
<th>Attribute names or columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Price</td>
</tr>
<tr>
<td>gizmo</td>
<td>$19.99</td>
</tr>
<tr>
<td>Power gizmo</td>
<td>$29.99</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
</tr>
</tbody>
</table>

From E/R Diagrams to Relational Schema
- Entity set → relation
- Relationship → relation

Overview of the next two classes
- Entity Relationship (ER) diagrams
- Relational databases
  - How did we get here?
  - What’s in a relational schema?
- From ER to relational
- Query Languages
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)
- Potpourri

Entity Set to Relation
- **Product** (name, category, price)
- name | category | price
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>gizmo</td>
<td>gadgets</td>
<td>$19.99</td>
</tr>
</tbody>
</table>
Relationships to Relations

Makes(product-name, product-category, company-name, year)

<table>
<thead>
<tr>
<th>Product-name</th>
<th>Product-Category</th>
<th>Company-name</th>
<th>Starting-year</th>
</tr>
</thead>
<tbody>
<tr>
<td>gizmo</td>
<td>gadgets</td>
<td>gizmoWorks</td>
<td>1963</td>
</tr>
</tbody>
</table>

(watch out for attribute name conflicts)

Overview of the next two classes
- Entity Relationship (ER) diagrams
- Relational databases
  - How did we get here?
  - What's in a relational schema?
  - From ER to relational
- Query Languages
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)
- Potpourri

Relational Query Languages
- A major strength of the relational model: simple, powerful querying of data.
- Queries can be written intuitively; DBMS is responsible for efficient evaluation.
  - Precise semantics for relational queries.
  - Optimizer can re-order operations, and still ensure that the answer does not change.
- We'll look at 3: relational algebra, SQL, and datalog

Find products where the manufacturer is GizmoWorks

<table>
<thead>
<tr>
<th>Product</th>
<th>Name</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
<td></td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
<td></td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
<td></td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
<td></td>
</tr>
</tbody>
</table>

Selection: $\sigma_{\text{Manufacturer} = \text{GizmoWorks}} \ Product$

Find the Name, Price, and Manufacturers of products whose price is greater than 100

<table>
<thead>
<tr>
<th>Product</th>
<th>Name</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
<td></td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
<td></td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
<td></td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
<td></td>
</tr>
</tbody>
</table>

Selection + Projection: $\pi_{\text{Name}, \text{Price}, \text{Manufacturer}} (\sigma_{\text{Price} > 100} \ Product)$
Find names and prices of products that cost less than $200 and have Japanese manufacturers

<table>
<thead>
<tr>
<th>Product</th>
<th>Name</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
<td></td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
<td></td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
<td></td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
<td></td>
</tr>
</tbody>
</table>

π\text{Product}(\text{Name, Price} < 200) \bowtie \text{Company}(\text{Country} = \text{Japan})

When are two relations related?
- You guess they are
- I tell you so
- Constraints say so
  - A key is a set of attributes whose values are unique; we underline a key
  - Foreign keys are a method for schema designers to tell you so
    - A foreign key states that an attribute is a reference to the key of another relation
      ex: Product.Manufacturer is foreign key of Company
    - Gives information and enforces constraint

The SQL Query Language
- Structured Query Language
- The standard relational query language
- Developed by IBM (System R) in the 1970s
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision, current standard)
  - SQL-99 (major extensions)

SQL basics
- Basic form: (many more bells and whistles in addition)

Select attributes
From relations (possibly multiple, joined)
Where conditions (selections)

SQL
- Data Manipulation Language (DML)
  - Query one or more tables
  - Insert/delete/modify tuples in tables
- Data Definition Language (DDL)
  - Create/alter/delete tables and their attributes
- Transact-SQL
  - Idea: package a sequence of SQL statements → server

SQL – Selections

```
SELECT * FROM Company WHERE country = “Canada” AND stockPrice > 50
```

Some things allowed in the WHERE clause:
- attribute names of the relation(s) used in the FROM
- comparison operators: =, <>, <, >, <=, >=
- apply arithmetic operations: stockPrice*2
- operations on strings (e.g., “” for concatenation)
- Lexicographic order on strings
- Pattern matching: s LIKE p
- Special stuff for comparing dates and times.
**SQL – Projections**

Select only a subset of the attributes

```sql
SELECT name, stockPrice AS price
FROM Company
WHERE country='Canada' AND stockPrice > 50
```

Rename the attributes in the resulting table

```sql
SELECT name AS company, stockPrice AS price
FROM Company
WHERE country='Canada' AND stockPrice > 50
```

**SQL – Joins**

```sql
SELECT name, stockPrice
FROM Company
WHERE country='Canada' AND stockPrice > 50
```

**Selection:**

\( \sigma_{\text{Manufacturer} = \text{GizmoWorks}}(\text{Product}) \)

**Product**

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

What’s the SQL?

```sql
\( \pi_{\text{Name, Price, Manufacturer}}(\sigma_{\text{Price} > 100}(\text{Product})) \)
```

**Product**

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

What’s the SQL?

```sql
\( \pi_{\text{Name, Price}}(\sigma_{\text{Price} <= 200}(\text{Product}) \bowtie \sigma_{\text{Country} = 'Japan'}(\text{Company})) \)
```

**Product**

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
</tr>
</tbody>
</table>

What’s the SQL?

**Querying – Datalog**

(Our final query language)
- Enables recursive queries
- More convenient for analysis
- Some people find it easier to understand
- Without recursion but with negation it is equivalent in power to relational algebra and SQL
- Limited version of Prolog (no functions)
**Datalog Rules and Queries**

A datalog rule has the following form:

```
head :- atom1, atom2, ..., atom, ...
```

You can read this as:

```
then :- if...
```

ExpensiveProduct(N) :- Product(N,M,P) & P > $100

CanadianProduct(N) :- Product(N,M,P) & Company(M, "Canada", SP)

IntlProd(N) :- Product(N,M,P) & NOT Company(M, "Canada", SP)

**Conjunctive Queries**

- A subset of Datalog
- Only relations appear in the right hand side of rules
- No negation
- Functionally equivalent to Select, Project, Join queries
- Very popular in modeling relationships between databases

**Selection:**

\[ \sigma_{\text{Manufacturer} = \text{GizmoWorks}}(\text{Product}) \]

**Selection + Projection:**

\[ \pi_{\text{Name}, \text{Price}, \text{Manufacturer}}(\sigma_{\text{Price} > 100}(\text{Product})) \]

**Administrative notes**

- If you have not yet signed up for papers, I'll be doing some redistributing AFTER drop/add ends
- Remember the 1st homework is due beginning of class Monday
- General theory – trying to make sure you understand basics and have thought about it – not looking for one, true, answer.
- State any assumptions you make
- If you can't figure out a detail, write an explanation as to what you did and why.
- Office hours?
Summarizing/Rehashing Relational DBs

- Relational perspective: Data is stored in relations. Relations have attributes. Data instances are tuples.
- SQL perspective: Data is stored in tables. Tables have columns. Data instances are rows.
- Query languages
  - Relational algebra – mathematical base for understanding query languages
  - SQL – most commonly used
  - Datalog – based on Prolog, very popular with theoreticians
- Bonus! Views allow complex queries to be written simply

Bonus Relational Goodness: Views

Views are stored queries treated as relations. Virtual views are not physically stored. Materialized views are stored. They are used (1) to define conceptually different views of the database and (2) to write complex queries simply.

View: purchases of telephony products:

```
CREATE VIEW telephony-purchases AS
  SELECT product, buyer, seller, store
  FROM Purchase, Product
  WHERE Purchase.product = Product.name
  AND Product.category = "telephony"
```

Outline

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)
- Potpourri

Object-Oriented DBMS’s

- Started late 80’s
- Main idea:
  - Toss the relational model!
  - Use the OO model – e.g., C++ classes
- Standards group: ODMG = Object Data Management Group.
- OQL = Object Query Language, tries to imitate SQL in an OO framework.

The OO Plan

ODMG imagines OO-DBMS vendors implementing an OO language like C++ with extensions (OQL) that allow the programmer to transfer data between the database and “host language” seamlessly.

A brief diversion: the impedance mismatch

OO Implementation Options

- Build a new database from scratch (O2)
  - Elegant extension of SQL
  - Later adopted by ODMG in the OQL language
  - Used to help build XML query languages
- Make a programming language persistent (ObjectStore)
  - No query language
  - Niche market
- We’ll see a few others
ODL

- ODL defines persistent classes, whose objects may be stored permanently in the database.
- ODL classes look like Entity sets with binary relationships, plus methods.
- ODL class definitions are part of the extended, OO host language.

ODL – remind you of anything?

interface Person
    (extent People key sin)
    { attribute string sin;
      attribute string dept;
      attribute string name;}

interface Student
    extends Person
    (extent Students)
    { attribute string major;
      relationship Set<Course> stds;
      inverse takes;}

Why did OO Fail?

- Why are relational databases so popular?
  - Very simple abstraction; don’t have to think about programming when storing data.
  - Very well optimized
- Relational db are very well entrenched – OODBs had not enough advantages, and no good exit strategy (we’ll see more about this)

Merging Relational and OODBs

- Object-oriented models support interesting data types – not just flat files.
  - Maps, multimedia, etc.
- The relational model supports very-high-level queries.
- Object-relational databases are an attempt to get the best of both.
- All major commercial DBs today have OR versions – full spec in SQL99, but your mileage may vary.

Outline

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)
- Potpourri

XML

- eXtensible Markup Language
- XML 1.0 – a recommendation from W3C, 1998
- Roots: SGML (from document community - works great for them; from db perspective, very nasty).
- After the roots: a format for sharing data
XML is self-describing

- Schema elements become part of the data
  - In XML `<persons>`, `<name>`, `<phone>` are part of the data, and are repeated many times
- Relational schema: `<persons(name,phone)>` defined separately for the data and is fixed
- Consequence: XML is much more flexible

Why XML is of Interest to Us

- XML is semistructured and hierarchical
- XML is just syntax for data
  - Note: we have no syntax for relational data
- This is exciting because:
  - Can translate any data to XML
  - Can ship XML over the Web (HTTP)
  - Can input XML into any application
  - Thus: data sharing and exchange on the Web

XML Data Sharing and Exchange

From HTML to XML

HTML describes the presentation

XML describes the content
XML Document

XML Terminology

- **Elements**
  - enclosed within tags: `<person> ... </person>`
  - nested within other elements: `<person> <address> ... </address> </person>`
  - can be empty `<married></married>` abbreviated as `<married/>`
  - can have Attributes `<person id="0005"> ... </person>`

- **XML document has as single ROOT element**

XML as a Tree !!

Relational Data as XML

XML is **semi-structured**

- Missing elements:
  
  ```
  <person>  <name>John</name>  </person>
  <person>  <name>Joe</name>  </person>
  ```

  → no phone !

- Could represent in a table with nulls

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>1234</td>
</tr>
<tr>
<td>Joe</td>
<td></td>
</tr>
</tbody>
</table>

XML is **semi-structured**

- Repeated elements

  ```
  <person>  <name>Mary</name>  <phone>2345</phone>  <phone>3456</phone>  </person>
  ```

  → two phones !

- Impossible in tables:

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>2345</td>
</tr>
<tr>
<td>Joe</td>
<td>3456</td>
</tr>
<tr>
<td>Mary</td>
<td>??</td>
</tr>
</tbody>
</table>
XML is semi-structured

- Elements with different types in different objects

```
<person><name><first>John</first><last>Smith</last></name><phone>1234</phone></person>
```

- Heterogeneous collections:
  - `<persons>` can contain both `<person>`s and `<customer>`s

Summarizing XML

- XML has first class elements and second class attributes
- XML is semi-structured
- XML is nested
- XML is a tree
- XML is a huge buzzword

Will XML replace relational databases?

Outline

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)
- Potpourri

Other data formats

- Makefiles
- Forms
- Application code

What format is your data in?

Outline

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)
  - Query Optimization & Execution
  - Transaction Processing
- Potpourri

How SQL Gets Executed:

**Query Execution Plans**

```
Select Name, Price
From Product, Company
Where Manufacturer = Cname
AND Price <= 200
AND Country = 'Japan'
```

Query optimization also specifies the algorithms for each operator; then queries can be executed.
Overview of Query Optimization

- **Plan**: Tree of ordered Relational Algebra operators and choice of algorithm for each operator
- Two main issues:
  - For a given query, what plans are considered?
  - Algorithm to search plan space for cheapest (estimated) plan.
  - How is the cost of a plan estimated?
- Ideally: Want to find best plan.
- Practically: Avoid worst plans.
- Some tactics
  - Do selections early
  - Use materialized views
  - Use Indexes

Example B+ Tree

- Search begins at root, and key comparisons direct it to a leaf.
- Search for 5*, 15*, all data entries >= 24*

![Example B+ Tree Diagram]

Tree-Based Indexes

- “Find all students with gpa > 3.0”
  - If data is in sorted file, do binary search to find first such student, then scan to find others.
  - Cost of binary search can be quite high.
- Simple idea: Create an ‘index’ file.

![Example Index File Diagram]

Query Execution

- Now that we have the plan, what do we do with it?
  - How do joins work?
  - How do deal with paging in data, etc.
- New research covers new paradigms where interleaved with optimization

Outline

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)
  - Query Optimization & Execution
  - Transaction Processing
- Potpourri

Transactions

Address two issues:

- Access by multiple users
- Protection against crashes
Transactions

- Transaction = group of statements that must be executed atomically

- Transaction properties: ACID
  - **Atomicity**: either all or none of the operations are completed
  - **Consistency**: preserves database integrity
  - **Isolation**: concurrent transactions must not interfere with each other
  - **Durability**: changes from successful transactions must persist through failures

Transactions: Serializability

Serializability = the technical term for isolation

- An execution is **serial** if it is completely before or completely after any other function's execution
- An execution is **serializable** if it equivalent to one that is serial
- DBMS can offer serializability guarantees

Transaction Example

- Consider two transactions:

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ(A)</td>
<td>READ(A)</td>
</tr>
<tr>
<td>A=A+100</td>
<td>A=1.1*A</td>
</tr>
<tr>
<td>WRITE(A)</td>
<td>WRITE(A)</td>
</tr>
<tr>
<td>READ(B)</td>
<td>READ(B)</td>
</tr>
<tr>
<td>B=B-100</td>
<td>B=1.1*B</td>
</tr>
<tr>
<td>WRITE(B)</td>
<td>WRITE(B)</td>
</tr>
</tbody>
</table>

  - Intuitively, T1 transfers $100 to A's account from B's account. T2 credits both accounts with a 10% interest payment.
  - No guarantee that T1 executes before T2 or vice-versa. However, the end effect must be equivalent to these two transactions running serially in some order: T1, T2 or T2, T1

Serializability Example

- Enforced with locks, like in Operating Systems!
- But this is not enough:

```
User 1
LOCK A [write A=1]
UNLOCK A
...
...
...
...
LOCK B [write B=2]
UNLOCK B

User 2
LOCK A [write A=3]
UNLOCK A
LOCK B [write B=4]
UNLOCK B
```

What is wrong?

Okay, but what if it crashes?

Transaction States

- A transaction can be in one of the following states:
  - **active**: makes progress or waits for resources; the initial state
  - **committed**: after successful completing a "commit" command
  - to undo its effects we need to run a compensating transaction
  - A few others we won’t go into

Enforcing Atomicity & Durability

- **Atomicity**:
  - Transactions may abort; Need to rollback changes
- **Durability**:
  - What if DBMS stops running? Need to "remember" committed changes.

```
T1
T2
T3
T4
T5
```

- Desired Behaviour after system restarts:
  - T1, T2, & T3 should be durable.
  - T4 & T5 should be aborted (effects not seen)
Handling the Buffer Pool

- Transactions modify pages in memory buffers
- Writing to disk is more permanent
- When should updated pages be written to disk?
  - Force every write to disk?
    - Poor response time.
    - But provides durability.
  - Steal buffer-pool frames from uncommitted Xacts? (resulting in write to disk)
    - If not, poor throughput.
    - If so, how can we ensure atomicity?

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

What to do?

- Basic idea: use steal and no-force
- Keep a log that tracks what’s happened
- Make checkpoints where write down everything that’s actually happened
- After a crash: assure Atomicity and Durability by keeping all committed transactions and getting rid of actions of uncommitted transactions

Outline

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)
- Potpourri
  - Complexity

Complexity

- Characterize algorithms by how much time they take
- The first major distinction: Polynomial (P) vs. Non-deterministic Polynomial (NP)
- Algorithms in P can be solved in P. time in size of input
  - E.g., merge sort is O(n log n) (where n = # of items)
- NP algorithms can be solved in NP time; equivalently, they can be verified in in polynomial time
- NP-complete = a set of algorithms that is as hard as possible but still in NP
  - E.g., Traveling Salesperson Problem
- Co-NP refers to algorithms whose converses are NP complete

Complexity Ice Cream Cone

P  NP  Co-NP

Outline

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)
- Potpourri
Now what?

- Time to read papers
- Prepare paper responses – it'll help you focus on the paper, and allow for the discussion leader to prepare better discussion
- You all have different backgrounds, interests, and insights. Bring them into class!