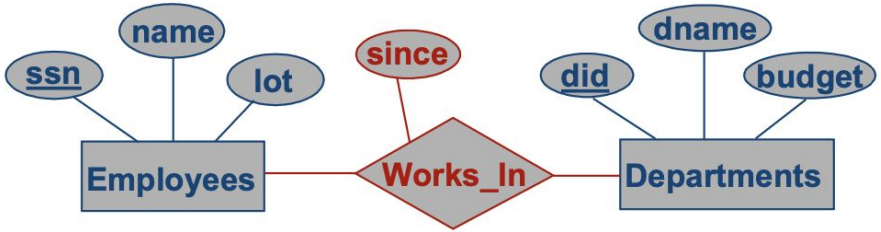
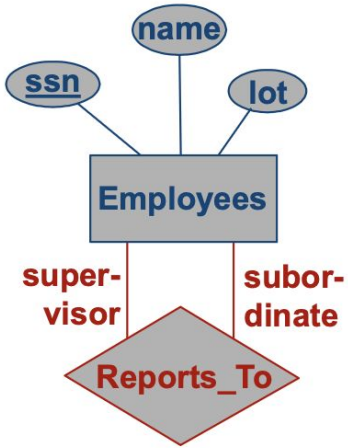
DBWS

ER Model Basics (Contd.)

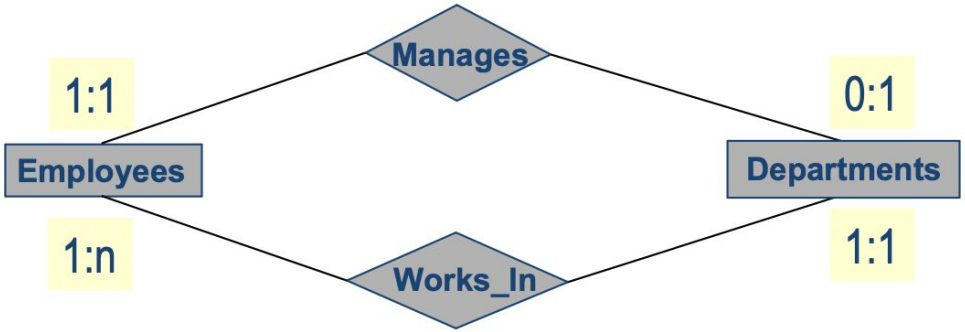
- Relationship: (unique!) association among two or more entities
  - E.g., Attishoo works_in Pharmacy department
- Relationship Set: Collection of similar relationships
  - An n-ary (binary, ternary, …) relationship set R relates n entity sets E1 ... En
  - each relationship in R involves entities e1 ∈ E1, ..., en ∈ En
  - Same entity set can participate in different relationship sets, or even in the same set (but then in different roles)





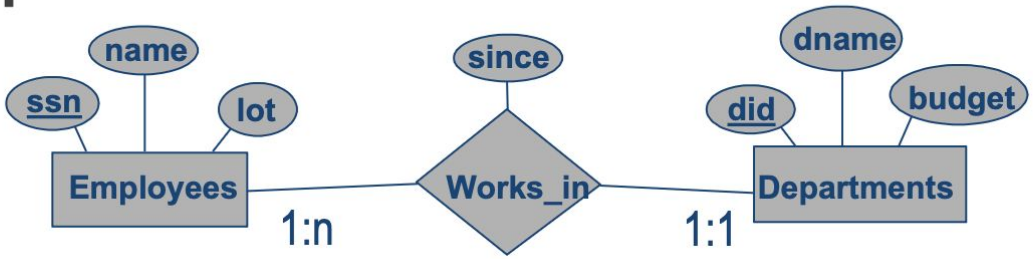
More Detail Wanted!

- Want to refine further: *how many connections on each leg of relship?*
- Attach intervals to leg:



- Read as:
  - *„an Employee sees, through its Manages tunnel, none or one Department“*
  - *„a Department sees, through its Works_In tunnel, at least one Employee“*

Example




```
Create table Employees(
  eid: int,
  ssn: int unique,
  name: char(100),
  lot: int
  primary key (eid)
)
```

```
Create table Works_in(
  eid: int unique,
  did_ int,
  since: date
  primary key(eid,did_)
  foreign key (eid) references Employees
  foreign key (did_) references Departments
)
```

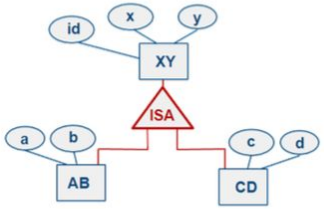
```
Create table Departments(
  did_: int,
  did: int unique,
  dname: char(100),
  budget: money
  primary key (did_)
)
```

eid	ssn	name	lot
1	123	John Doe	5
2	456	Jane Fox	17
3	789	Charlie Brown	42

eid	did_	since
1	2	2018-12-01
3	1	2017-01-01
2	2	2015-06-01

did_	did	name	budget
1	5	Sales	500
2	17	Accounting	170
3	99	Production	420

ISA → Relations: Schemas



- Alt 1: separate relation per entity set

  XY (id, x, y)
  AB (id, a, b, FOREIGN KEY (id) REFERENCES XY(id))
  CD (id, c, d, FOREIGN KEY (id) REFERENCES XY(id))

- Alt 2: relations only for subclass entity sets

  XYAB (id, x, y, a, b)
  XYCD (id, x, y, c, d)

- Alt 3: one big relation

  XYABCD (id, x, y, a, b c, d)

Views

- like a table, but stores query rather than data
- Definition:

```
CREATE  VIEW  YoungActiveStudents (name, grade)
AS  SELECT   S.name, E.grade
      FROM  Students S, Enrolled E
      WHERE  S.sid = E.sid and S.age < 21
```

- Use like any table:

```
SELECT name
FROM YoungActiveStudents
WHERE grade < 3.00
```

- Security: hiding details of underlying relation(s)
  - Given YoungActiveStudents, but not Students or Enrolled, can find students enrolled
  - …but not courses they are enrolled in

Conceptual Evaluation

- compute cross-product of *relation-list*
- discard tuples that fail *qualification*
- delete `*unnecessary*' attributes
- partition remaining tuples into groups by value of attributes in *grouping-list*
- apply *group-qualification* to eliminate some groups
  - Expressions in *group-qualification* must have a single value per group!
- generate one answer tuple per qualifying group

```
SELECT      [DISTINCT] target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING      group-qualification
```

Join

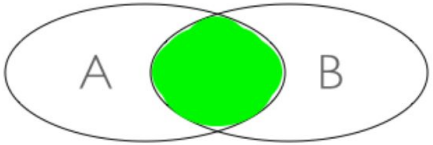
- Join = several tables addressed in one query

```
SELECT  target-list
FROM  Relation1 R1, Relation2 R2, …
WHERE  qualification
```

- List of relations in FROM clause determine cross product
- Frequently cross-relation conditions on attribute values to restrict results
- Most common: R1.attr1 = R2.attr2
  - ex:

```
SELECT  S.sid
FROM  Sailors S, Reserves R
WHERE  S.sid=R.sid
```

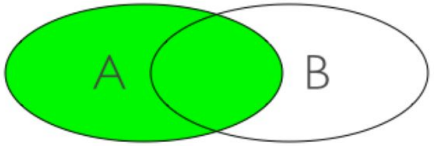
Even More on Joins



SELECT * FROM A JOIN B ON A.id=B.id;
SELECT * FROM A, B WHERE A.id=B.id;

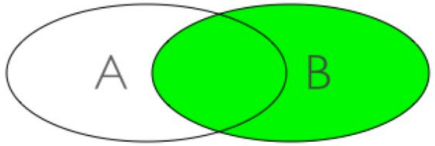
INNER JOIN

OUTER JOINS



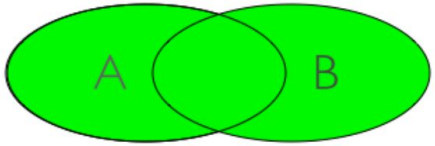
LEFT JOIN

SELECT * FROM A LEFT JOIN B
ON A.id=B.id



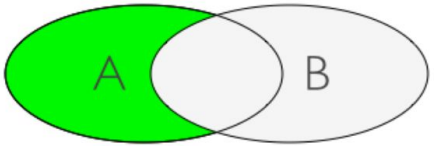
RIGHT JOIN

SELECT * FROM A RIGHT JOIN B
ON B.id=A.id

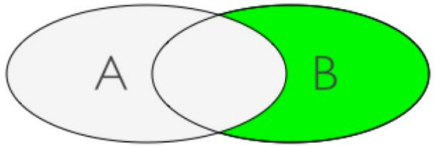


FULL JOIN

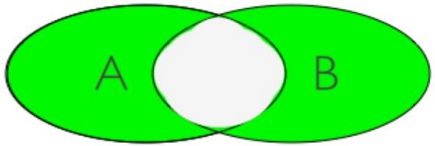
SELECT * FROM A FULL JOIN B
ON A.id=B.id



WHERE B.id IS NULL



WHERE A.id IS NULL



WHERE A.id IS NULL OR B.id is NULL

Queries With GROUP BY and HAVING

```
SELECT     [DISTINCT] target-list
FROM       relation-list
WHERE      qualification
GROUP BY   grouping-list
HAVING     group-qualification
```

- *target-list* contains (i) attribute names, (ii) aggregate terms (ex: MIN(S.age))
- *grouping-list*: list of attributes for grouping
- *group-qualification*: group selection criterion (predicate on *grouping-list*)
- *target-list* attributes must be subset of *grouping-list*
  - A group is a set of tuples that have the same value for all attributes in *grouping-list*
  - Intuitively, each answer tuple corresponds to a group, and these attributes must have a single value per group

HTTP Sample Request/Response

- Client sends:

```
GET ~dbbook/index.html HTTP/1.1
User-agent: Mozilla/4.0
Accept: text/*, image/gif, image/jpeg
```

- Server responds:

```
HTTP/1.1 200 OK
Date: Mon, 04 Mar 2002 12:00:00 GMT
Server: Apache/1.3.0 (Linux)
Last-Modified: Mon, 01 Mar 2002 09:23:24 GMT
Content-Length: 1024
Content-Type: text/html

<html> <head></head>
<body>
<h1>Burns and Nobble Internet Bookstore</h1>
Our inventory:
<h3>Science</h3>
<b>The Character of Physical Law</b>
...
</body></html>
```

Try this:
$ telnet google.com 80
GET / HTTP/1.1
<3x newline>

HTML Primer (contd.)

```
<a name="top">

<h1>An important heading</h1>

<h2>A slightly less important heading</h2>

<p>This is the <em>first</em> paragraph.</p>

<img src="peter.jpg" width="150" height="200" alt="me">

My link list:
<ul>
  <li>This is a link to <a href="http://www.w3.org/">W3C</a>
  <li>This a link to <a href="peter.html">Peter's page</a>
  <li>Go to <a href="#top">top</a>
  <li><a href="/"><img src="logo.gif" alt="home page"></a>
</ul>
```

HTML Primer (contd.)

- Text structuring (contd.)
  - tables
  - row
  - column heading
  - regular column

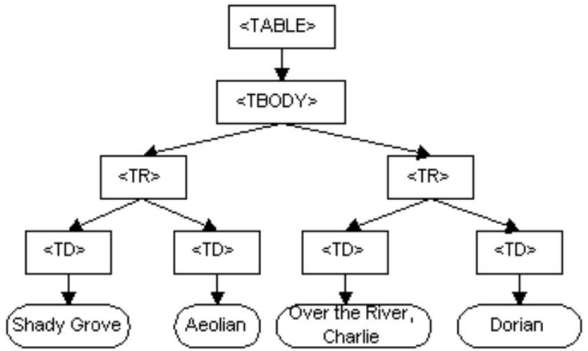
Year	Sales
2000	$18M
2001	$25M
2002	$36M

```
<table>
  <tr>
    <th>Year</th>
    <th>Sales</th>
  </tr>
  <tr>
    <td>2000</td>
    <td>$18M</td>
  </tr>
  <tr>
    <td>2001</td>
    <td>$25M</td>
  </tr>
  <tr>
    <td>2002</td>
    <td>$36M</td>
  </tr>
</table>
```

HTML and DOM

```
<TABLE>
  <TBODY>
    <TR>
      <TD>Shady Grove</TD>
      <TD>Aeolian</TD>
    </TR>
    <TR>
      <TD>Over the River, Charlie</TD>
      <TD>Dorian</TD>
    </TR>
  </TBODY>
</TABLE>
```

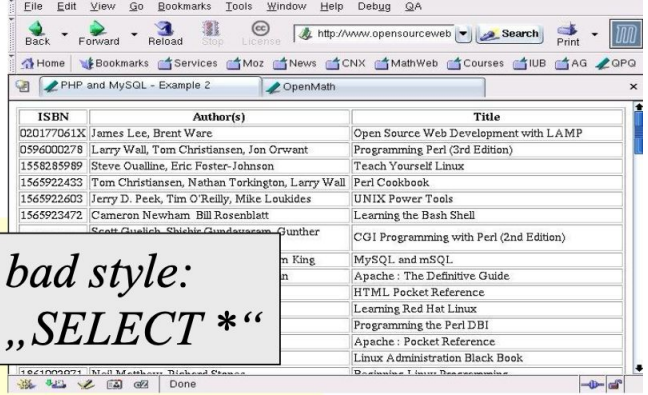
Exercise:
draw DOM tree
for some HTML snippet



CONSTRUCTOR UNIVERSITY

Document Object Model

PHP, HTML, and (My)SQL



bad style:
*„SELECT *“*

```
<html>
  <head>
    <title>PHP and MySQL Example</title>
  </head>
  <body>
    <?php $mysql = mysqli_connect( “localhost” );
      $result = mysql_db_query( “books”, “SELECT isbn, author, title FROM book_info" )
        or die( “query failed - “ . mysql_errno() . “: “ . mysql_error(); )
    ?>
    <table>
      <tr>  <th>ISBN</th>  <th>Author(s)</th>  <th>Title</th>  </tr>
      <?php while ( $array = mysql_fetch_array($result) ); ?>
      <tr><td><?php echo $array[ "isbn"    ]; ?></td>
          <td><?php echo $array[ "author" ]; ?></td>
          <td><?php echo $array[ "title"    ]; ?></td>
      </tr>
      <?php endwhile; ?>
    </table>
    <?php mysql_close($mysql); ?>
  </body>
</html>
```

open

query

iterate over result set

close

AJAX

- AJAX = Asynchronous Javascript and XML
- web development technique for creating more interactive web applications
  - Goal: increase interactivity, speed, functionality, usability
  - not complete page reload → small data loads → more responsive
- asynchronous: c/s communication independent from normal page loading
  - JavaScript
  - XML
  - any server-side PL

JSON

- JSON = JavaScript Object Notation
  - Lightweight data interchange format
  - MIME type: **`application/json`** (RFC 4627)
  - text-based, human-readable
- alternative to XML use
  - Subset of JavaScript's object literal notation
  - 10x faster than XML parsing
  - _way_ easier to handle
  - JSON parsing / generating code readily available for many languages

"JSON is XML without garbage"

JSON Example

- Server sends:

```
req.onreadystatechange=function()
{   if(req.readyState==4)
    {   var p = eval( "(" + req.responseText + ")" );
        document.myForm.firstName.value = p.firstName;
    }
}
```

- JSON string sent from server:

```
{   "firstName": "John",
    "lastName": "Smith",
    "address":
    {   "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [ "212 732-1234", "646 123-4567" ]
 }
```

- response parsing code:

```
<?  echo '{' + '"firstName":' + obj.firstName + ','
             + '"lastName":'  + obj.lastName  + ','
     … + '}'
?>
```

Relational Algebra: Summary

= Mathematical definition of relations + operators

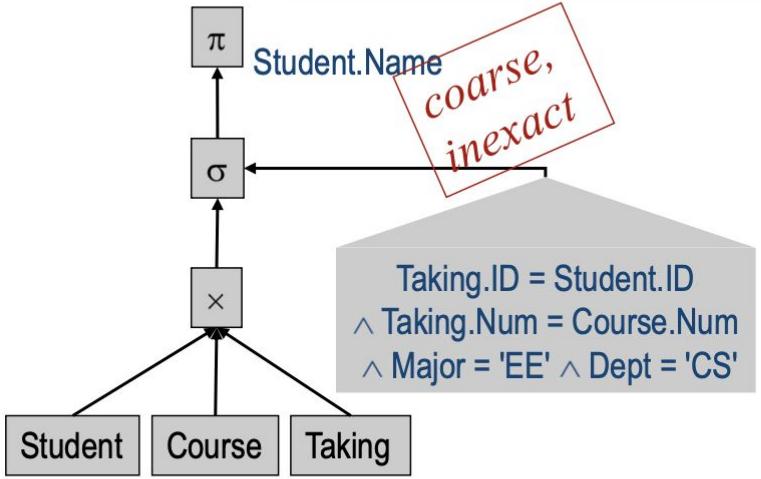
- Query = Algebraic expression

- Relational algebra A = (R,OP) with relation $R = A_1 \times ... \times A_n$, $OP=\{\pi,\sigma,\times\}$
  - Projection: $\pi_{attr}(R) = \{ r.attr \mid r \in R \}$
  - Selection: $\sigma_p(R) = \{ r \mid r \in R, p(r) \}$
  - Cross product: $R_1 \times R_2 = \{(r_{11}, r_{12}, ..., r_{21}, r_{22}, ...) \mid (r_{11}, r_{12}, ...) \in R_1, (r_{21}, r_{22}, ...) \in R_2 \}$
  - Further: set operations, join, ...

Logical Query Plan

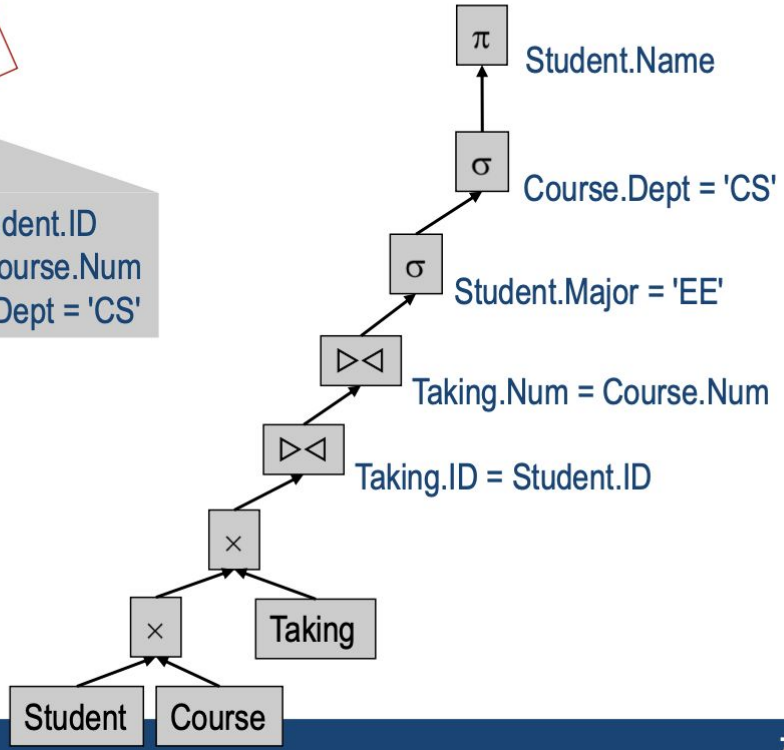
JACOBS UNIVERSITY

Parser – Checker - Views - Logical plan – Optim1 - Physical plan – Optim2 - Execution



coarse, inexact

```
SELECT  Name
FROM    Student, Course, Taking
WHERE   Taking.ID = Student.ID
   AND  Taking.Num = Course.Num
   AND  Major = 'EE'
   AND  Dept = 'CS'
```



Example Decomposition

- SNLRWH has FDs
  S → SNLRWH, R ↔ W, N → SN
- 2nd FD causes 3NF violation:
  W values repeatedly associated with R values (and vice versa)!

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

- Easiest fix: create relation RW to store assocs w/o dups, remove W from main schema
  = decompose SNLRWH into SNLRH and RW

Wages

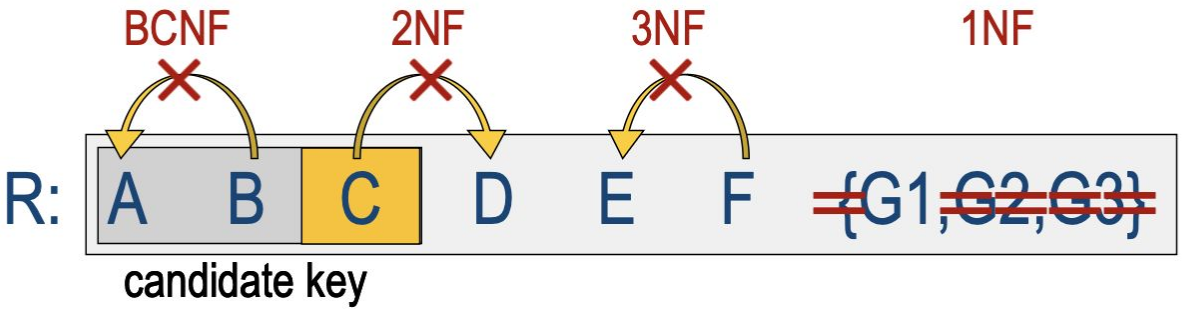
R	W
8	10
5	7

Hourly_Emps2

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

If we just store projections of SNLRWH tuples onto SNLRH and RW, are there any potential problems?

Pocket Guide to NFs

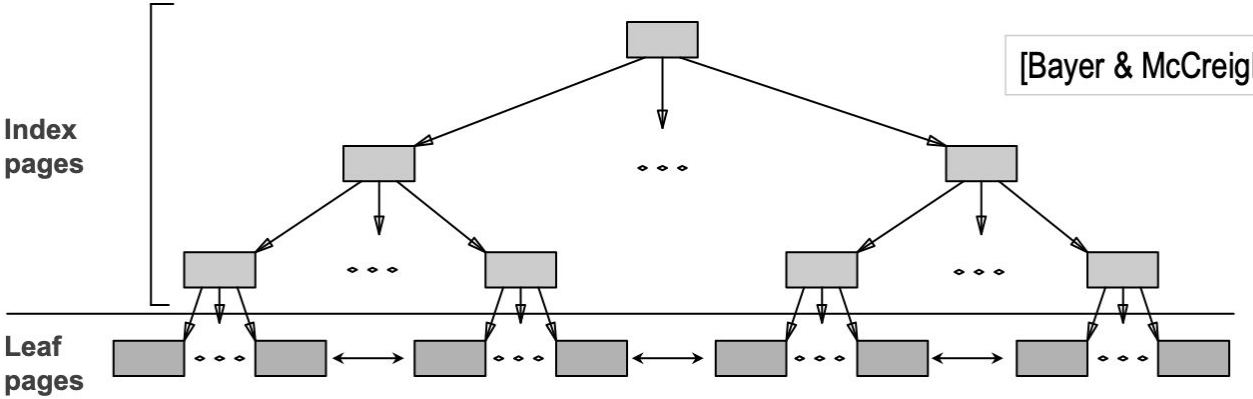


- 1NF = no repeating groups
- 2NF = 1NF + no partial key → non-key
- 3NF = 2NF + no non-key → anything
- BCNF = 3NF + no key → key

Normalization of table R with FD set :

- For all FDs F = „X → Y“:
  - Create additional table $R_F(X,Y)$
  - Remove Y from R, but keep X
- Drop duplicate tables arising from „X → Y, Y → X“ cycles
- Crosscheck all new tables created against all FDs for decomposition need

B+ Tree Indexes



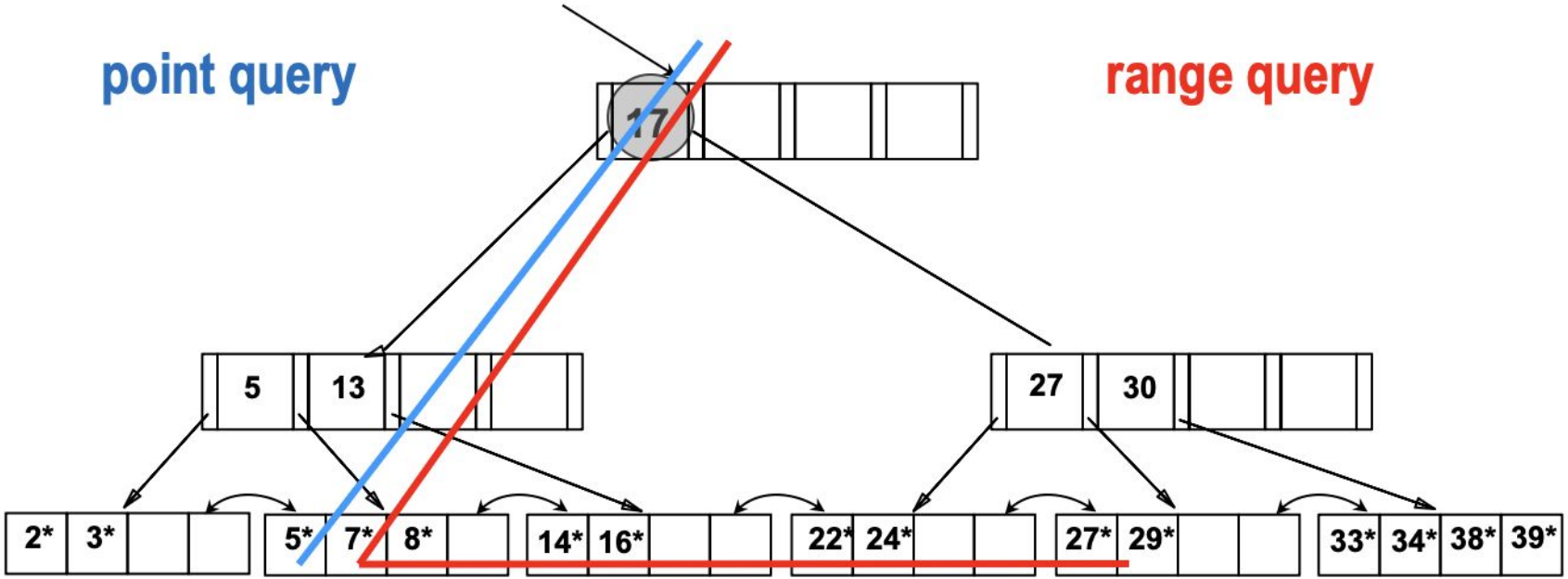
[Bayer & McCreight, 1972]

- Ordered Tree
- Leaf pages contain data entries, are chained (prev & next)
- Non-leaf pages have index entries to direct searches:

- Complexity: O($\log_F N$) where F = fan-out, N = # leaf pages

CONSTRUCTOR UNIVERSITY

Example B+ Tree: Traversal Pattern



ACID

- TA concept includes four basic properties:
- Atomic
  - all TA actions will be completed, or nothing
- Consistent
  - after commit/abort, data satisfy all integrity constraints
- Isolation
  - any changes are invisible to other TAs until commit
- Durable
  - nothing lost in future; failures occurring after commit cause no loss of data

Anomalies from Interleaved Execution

- Reading uncommitted data (R/W conflicts, "dirty reads"):

```
T1:    R(A), W(A),              R(B), W(B), Abort
T2:              R(A), W(A), Commit
```

- Unrepeatable reads (R/W conflicts):

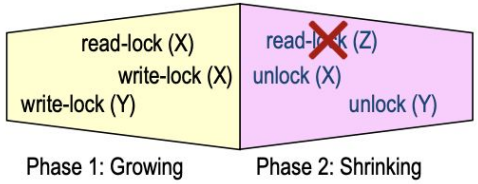
```
T1:    R(A),                    R(A), W(A), Commit
T2:           R(A), W(A), Commit
```

- Overwriting uncommitted data (W/W conflicts):

```
T1:    W(A),                    W(B), Commit
T2:           W(A), W(B), Commit
```

Two-Phase Locking Protocol

- 2PL
  - All locks acquired before first release
  - cannot acquire locks after releasing first lock
- allows only serializable schedules ☺
  - but complex abort processing



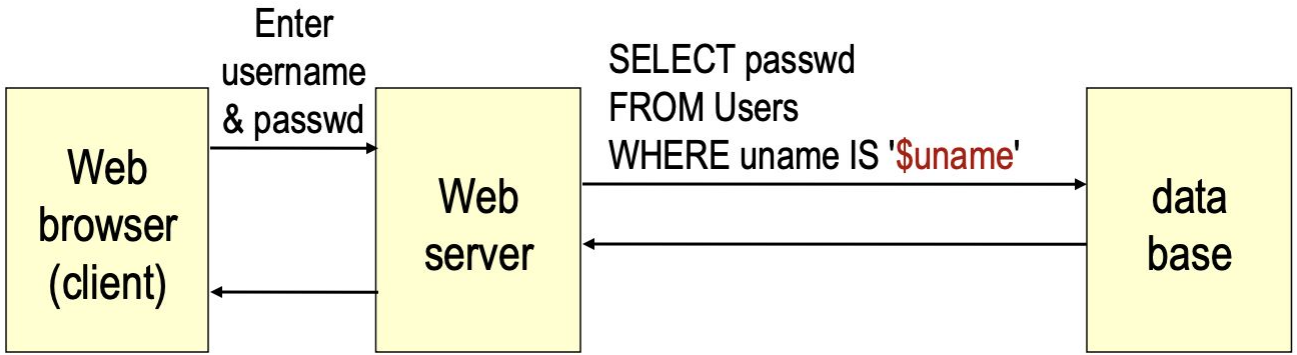
- Strict 2PL
  - All locks released when TA completes
- Strict 2PL simplifies TA aborts ☺☺

Write-Ahead Logging (WAL)

- All change actions recorded in log file(s)
  - Not single tuples, but complete pages affected
  - Before-Image (BFIM) + After-Image (AFIM) allow choice of redo or undo
  - Ti writes an object: TA identifier + BFIM + AFIM
  - Ti commits/aborts: TA identifier + commit/abort indicator
  - Log records chained by TA id → easy to undo specific TA
- Log written before database update = "write ahead"
  - Simply append to log file, so fast
- Log is beating heart of DBMS!
  - Use fast storage
  - often duplexed & archived on stable storage

How To Hack a Database (contd.)

- Most common: SQL injection
  - Compromise database query



- What will happen at input of '; DROP TABLE Users; -- ? (keyword: DoS)
- *Name 2 independent techniques to prevent!*

Outlook: ACID vs BASE

- BASE = Basically Available Soft-state Eventual Consistency
  - availability over consistency, relaxing ACID
  - ACID model promotes consistency over availability,
    BASE promotes availability over consistency
- Comparison:
  - Traditional RDBMSs: Strong consistency over availability under a partition
  - Cassandra: Eventual (weak) consistency, availability, partition-tolerance
- CAP Theorem [proposed: Eric Brewer; proven: Gilbert & Lynch]:
  In a distributed system you can satisfy at most 2 out of the 3 guarantees
  - Consistency: all nodes have same data at any time
  - Availability: system allows operations all the time
  - Partition-tolerance: system continues to work in spite of network partitions

Pattern Expressions

- Walk document, return (sub)tree
  - „CD prices in catalog“:
    /catalog/cd/price
  - „all titles, artists“:
    /catalog/cd/title | /catalog/cd/artist
  - „all titles and artists“:
    //title | //artist
  - *„all CDs in catalog with price 10.90“*:
    /catalog/cd[price=10.90]
  - „all CD countries“:
    //cd/@country

```
<price>10.90</price>
<price>9.90</price>
<price>9.90</price>
```

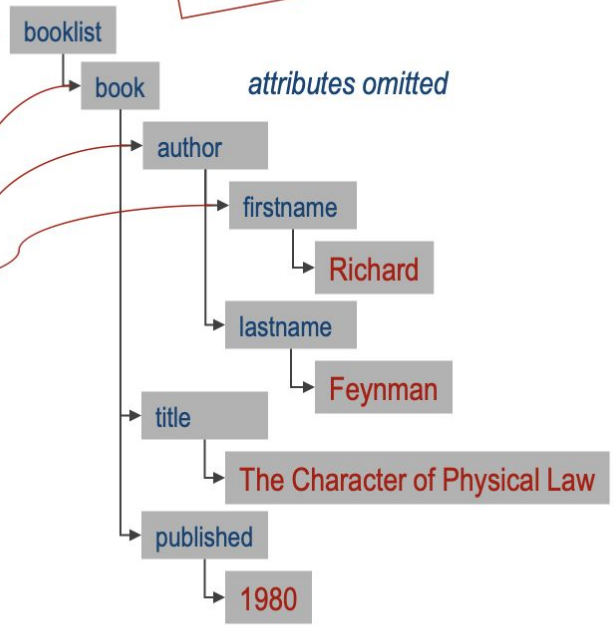
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

XML Document Tree

- Tree nodes = [element | attribute | text] infoset items

www.w3schools.com

```
<?xml version="1.0" encoding="UTF-8">
<!DOCTYPE booklist SYSTEM "library.std">
<booklist>
  <book genre="Science" format="Hardcover">
    <author>
      <firstname>Richard</firstname>
      <lastname>Feynman</lastname>
    </author>
    <title>The Character of Physical Law</title>
    <published>1980</published>
  </book>
</booklist>
```



XQuery

- XQuery – retrieving information from XML data
  - XQuery is to XML what SQL is to tables
  - XPath: extract from DOM tree; XQuery: derive new structure
  - Stored in files or in database
- FOR-LET-WHERE-ORDERBY-RETURN = FLWOR ("*flower*")
- Ex: "*all book titles published after 1995*"

```
FOR $x IN document("bib.xml")/bib/book
WHERE $x/year > 1995
RETURN $x/title
```

```
<title> abc </title>
<title> def </title>
<title> ghi </title>
```

RDF: Resource Description Framework

- Framework for resources on the Web = very simple assertion language
  - Conceptual model: directed, labeled graphs
  - designed for computers, not people
  - part of W3C's Semantic Web Activity
- RDF statement = resource with properties with values

- Ex: "*http://www.w3.org/TR/REC-rdf-syntax/ has the author Ora Lassila*"

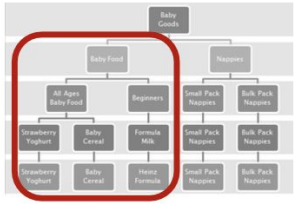
SPARQL

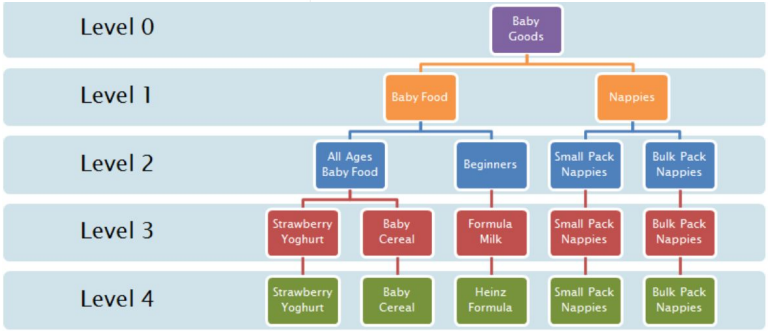
- = Simple Protocol and RDF Query Language
  - W3C recommendation
- = QL for RDF
  - extract information in the form of URIs, blank nodes, plain and typed literals
  - extract RDF subgraphs
  - construct new RDF graphs based on information in the queried graphs

- *Let's taste the flavor...*

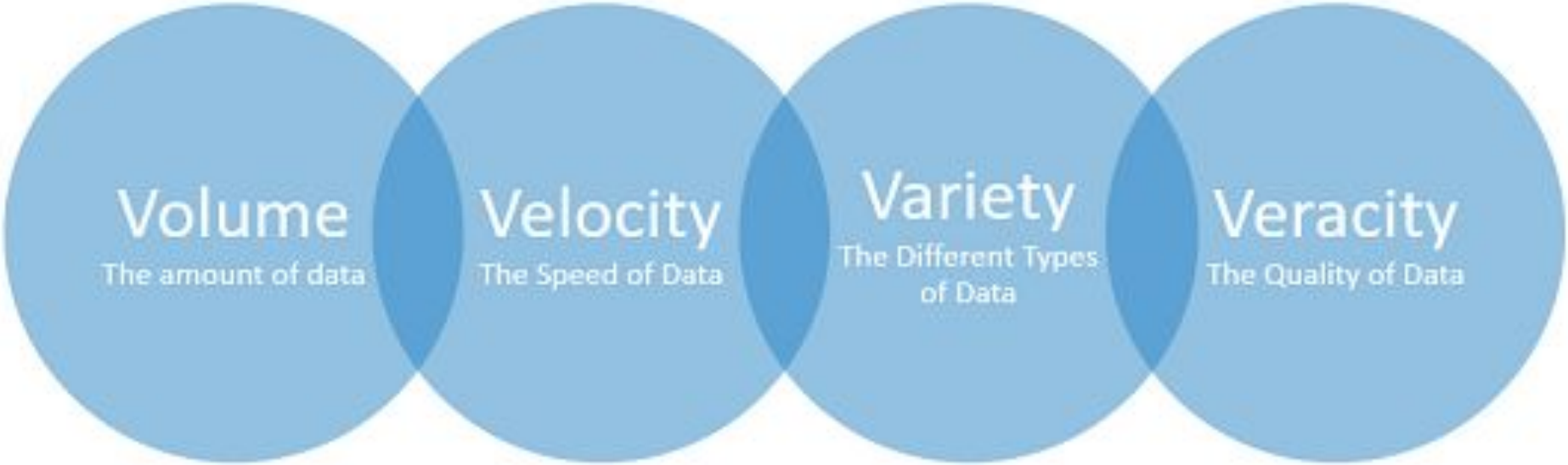
Dimension Hierarchies

- Dimension enumerates values along an axis
  - Ex: time (predefined, ordered), product (custom, unordered)
- Dimension hierarchy = generalization levels of a dimension
  - „zoom levels“ into datacube
  - Roll-up done based on hierarchies
- Strict nesting: Lower bins roll up neatly into higher bins
  - Not always strict! ex: week vs month





4 V's of Big Data



Visual Summary: Datacube Ops