# Physical Database Design and Referential Integrity

University of California, Berkeley

School of Information

*INFO 257: Database Management*

# Announcements

- Assignments 1 and 2a due
- Lecture
- Group Meeting (To discuss app stack selection for the next two class meetings)

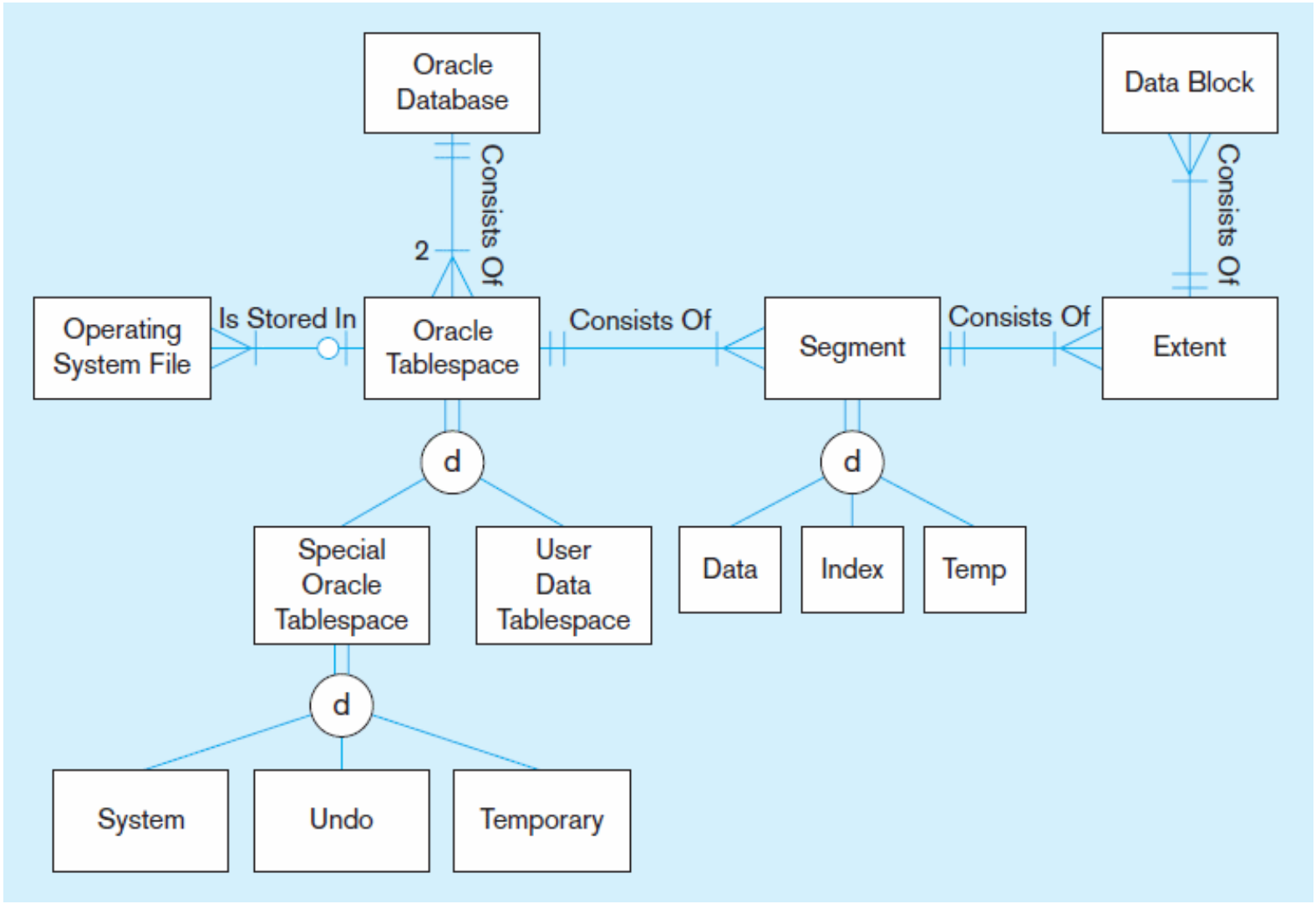
# Lecture Outline

- File and Access Methods
- Indexes and What to index
- Parallel storage systems (RAID)
- Integrity constraints
- Database Design Process Recap
- XML and databases – first look

# Designing Physical Database Files

- Physical File:
  - A named portion of secondary memory allocated for the purpose of storing physical records
  - Tablespace–named logical storage unit in which data from multiple tables/views/objects can be stored
- Tablespace components
  - Segment – a table, index, or partition
  - Extent–contiguous section of disk space
  - Data block – smallest unit of storage

# Figure 5-6 DBMS terminology in an Oracle 11g environment




© 2013 Pearson Education, Inc. Publishing as Prentice Hall

# File Organizations

- Technique for physically arranging records of a file on secondary storage
- Factors for selecting file organization:
  - Fast data retrieval and throughput
  - Efficient storage space utilization
  - Protection from failure and data loss
  - Minimizing need for reorganization
  - Accommodating growth
  - Security from unauthorized use
- Types of file organizations
  - Sequential
  - Indexed
  - Hashed

# Internal Model Access Methods

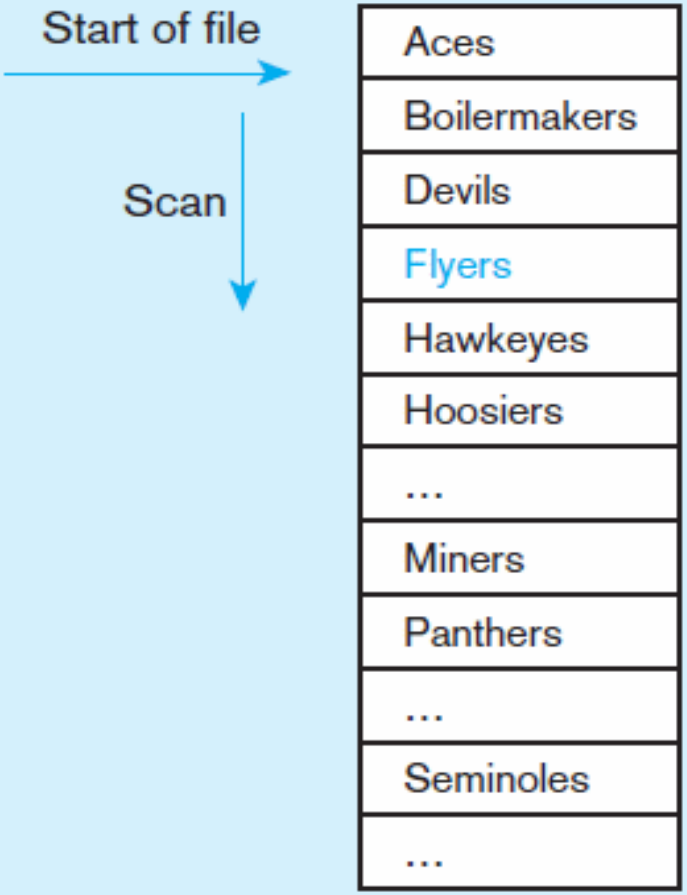
- Many types of access methods:
  - Physical Sequential
  - Indexed Sequential
  - Indexed Random
  - Inverted
  - Direct
  - Hashed
- Differences in
  - Access Efficiency
  - Storage Efficiency

# Physical Sequential

- Key values of the physical records are in logical sequence
- Main use is for “dump” and “restore”
- Access method may be used for storage as well as retrieval
- Storage Efficiency is near 100%
- Access Efficiency is poor (unless fixed size physical records)

# Sequential File Organization

Records of the file are stored in sequence by the primary key field values.




© 2013 Pearson Education, Inc. Publishing as Prentice Hall

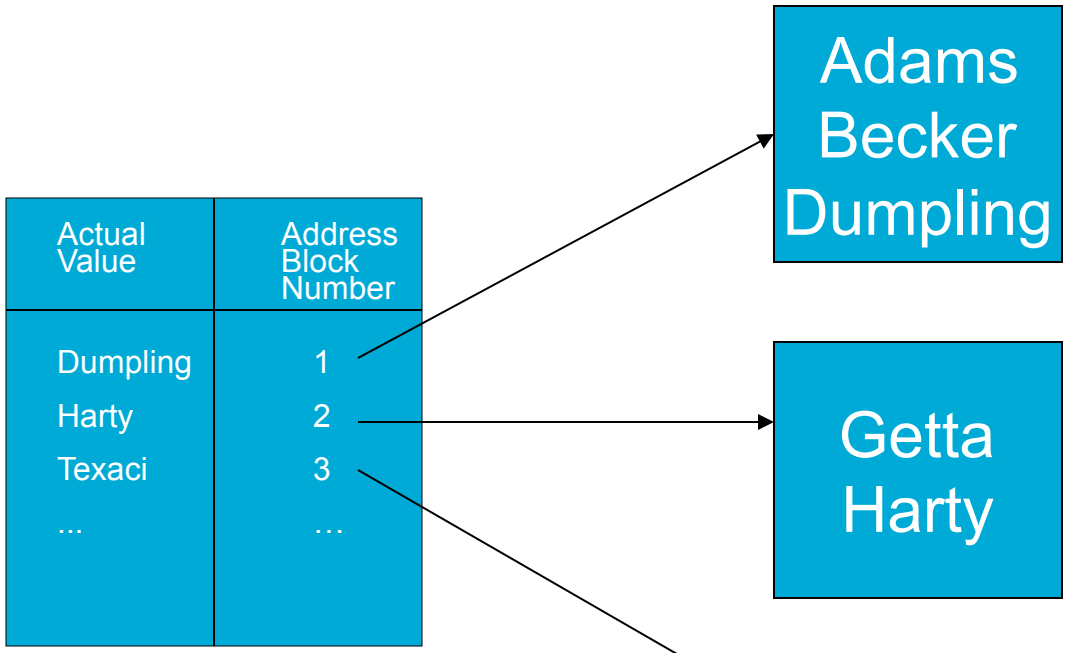
# Indexed File Organization

- Storage of records sequentially or non-sequentially with an index that allows software to locate individual records
- **Index**: a table or other data structure used to determine in a file the location of records that satisfy some condition
- Primary keys are automatically indexed
- Other fields or combinations of fields can also be indexed; these are called secondary keys (or non-unique keys)

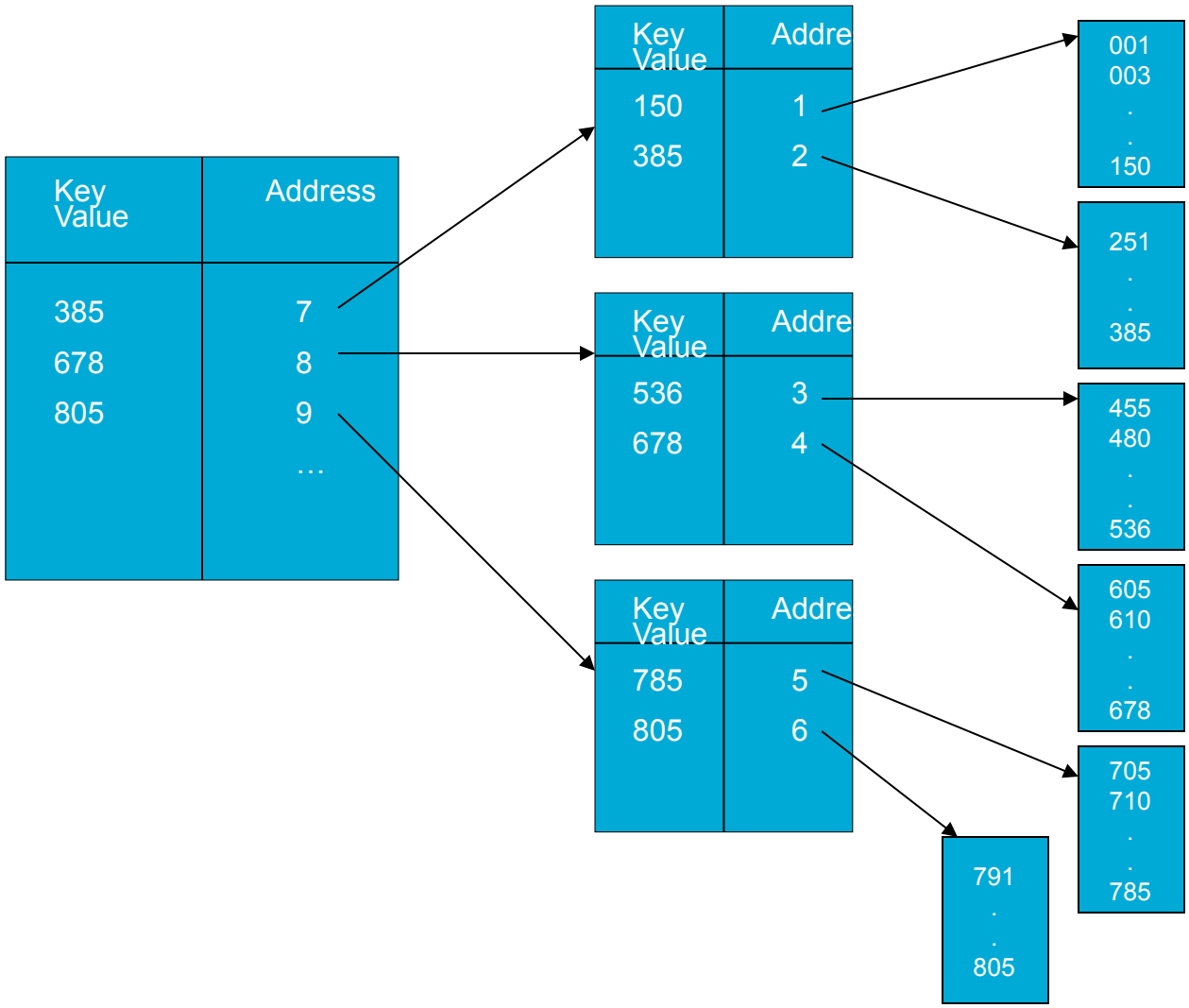
# Indexed Sequential

- Key values of the physical records are in logical sequence
- Access method may be used for storage and retrieval
- Index of key values is maintained with entries for the highest key values per block(s)
- Access Efficiency depends on the levels of index, storage allocated for index, number of database records, and amount of overflow
- Storage Efficiency depends on size of index and volatility of database

# Index Sequential



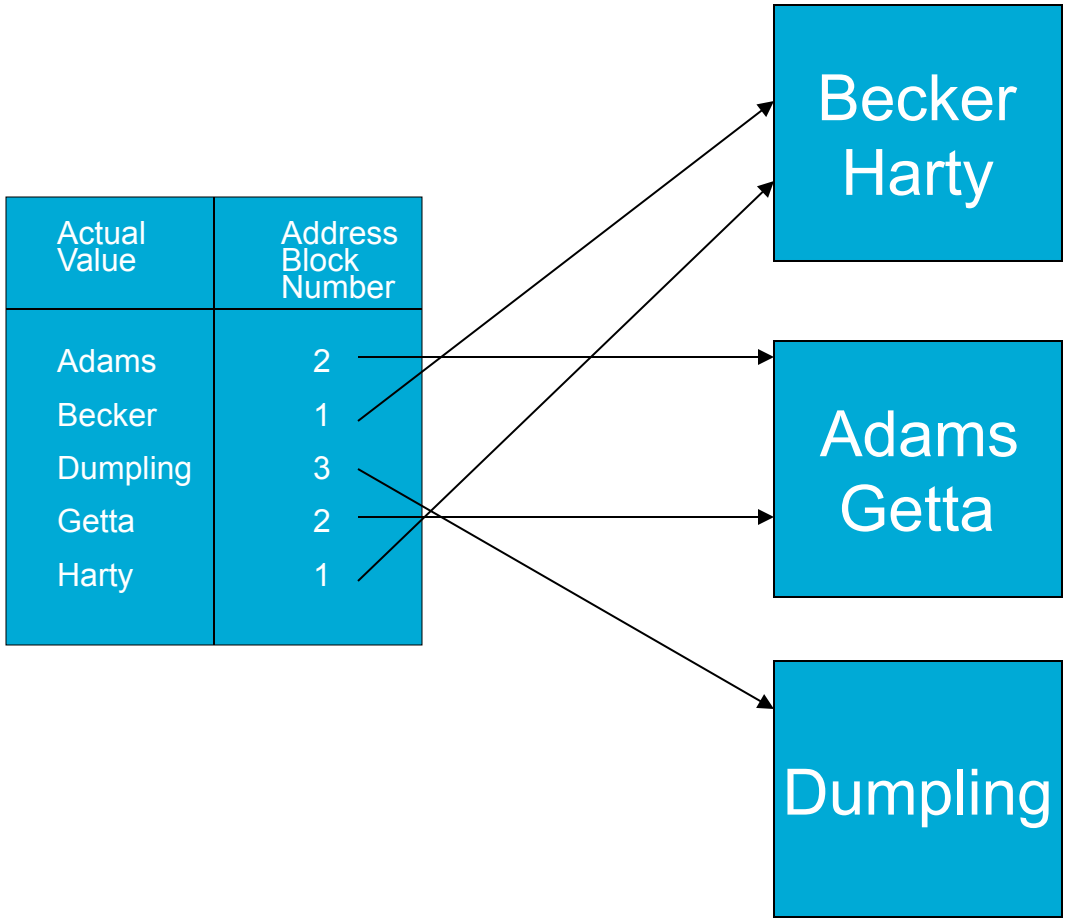
# Indexed Sequential: Two Levels



# Indexed Random

- Key values of the physical records are not necessarily in logical sequence
- Index may be stored and accessed with Indexed Sequential Access Method
- Index has an entry for every data base record. These are in ascending order. The index keys are in logical sequence. Database records are not necessarily in ascending sequence.
- Access method may be used for storage and retrieval

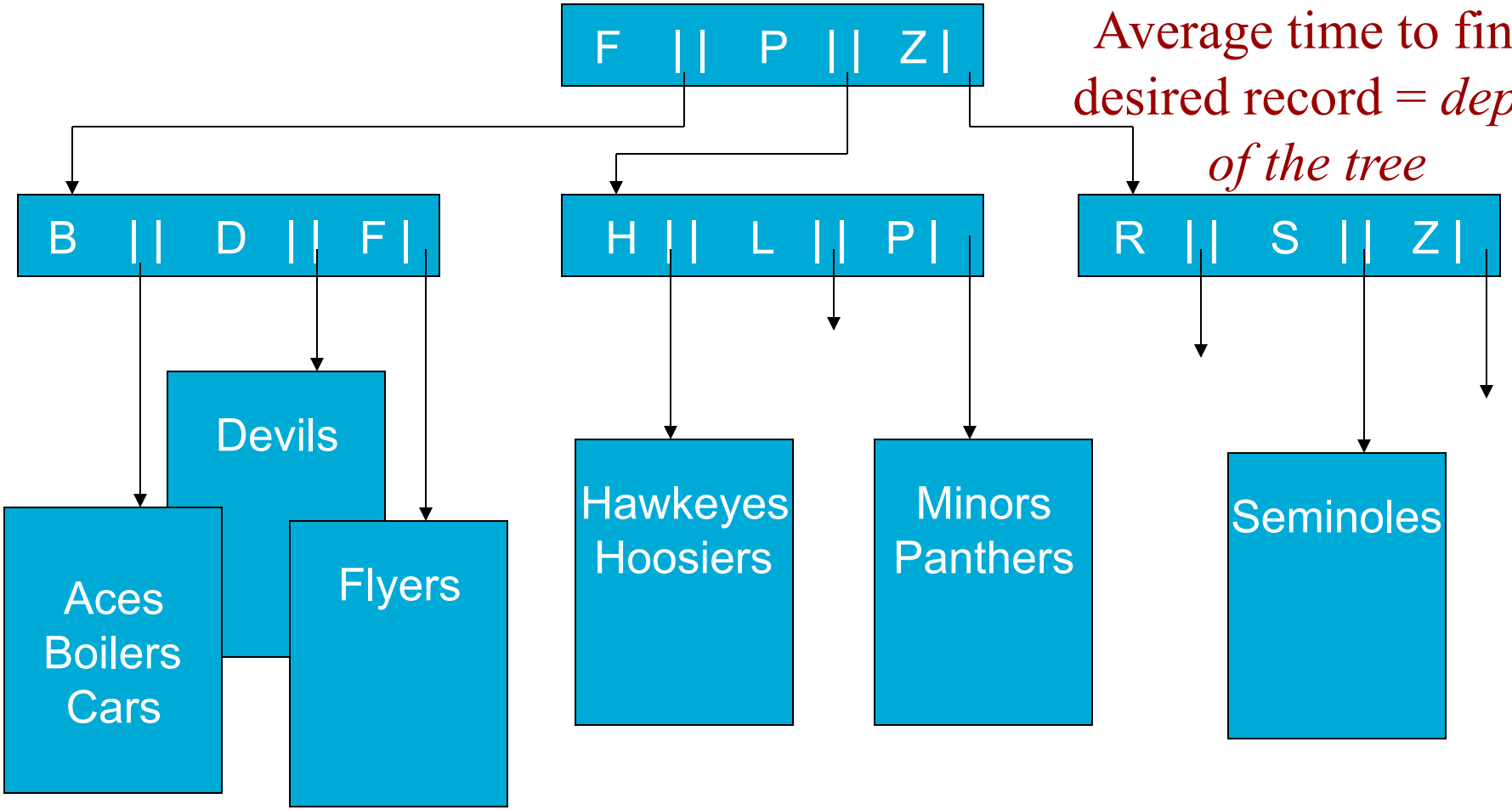
# Indexed Random



# Btree

**uses a *tree search***

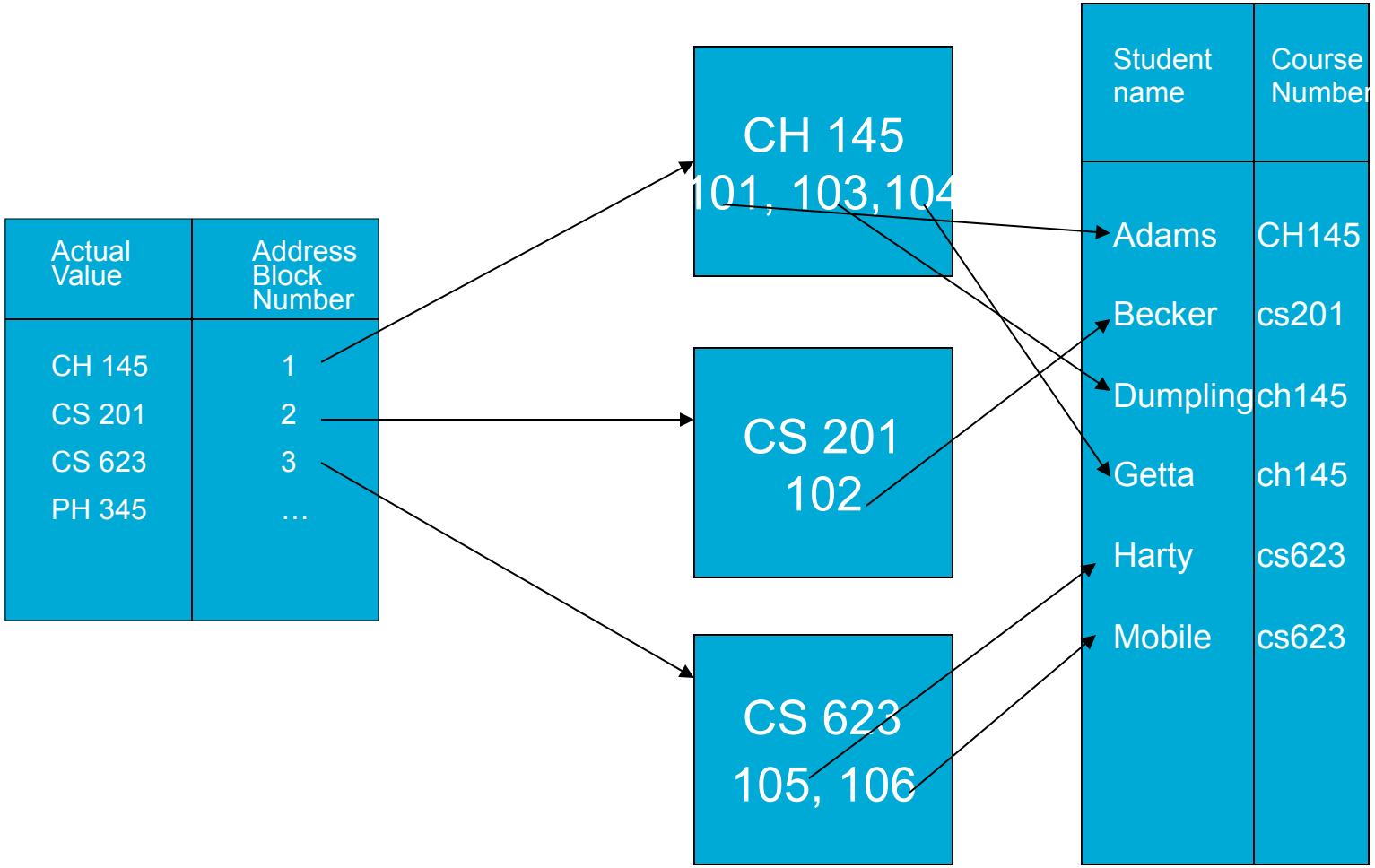
Average time to find desired record = *depth of the tree*



# Inverted

- Key values of the physical records are not necessarily in logical sequence
- Access Method is better used for retrieval
- An index for every field to be inverted may be built
- Access efficiency depends on number of database records, levels of index, and storage allocated for index

# Inverted


Actual Value
Address Block Number
CH 145
CS 201
CS 623
PH 345
1
2
3
…
CH 145
101, 103,104
CS 201
102
CS 623
105, 106
Student name
Course Number
Adams
CH145
Becker
cs201
Dumpling
ch145
Getta
ch145
Harty
cs623
Mobile
cs623

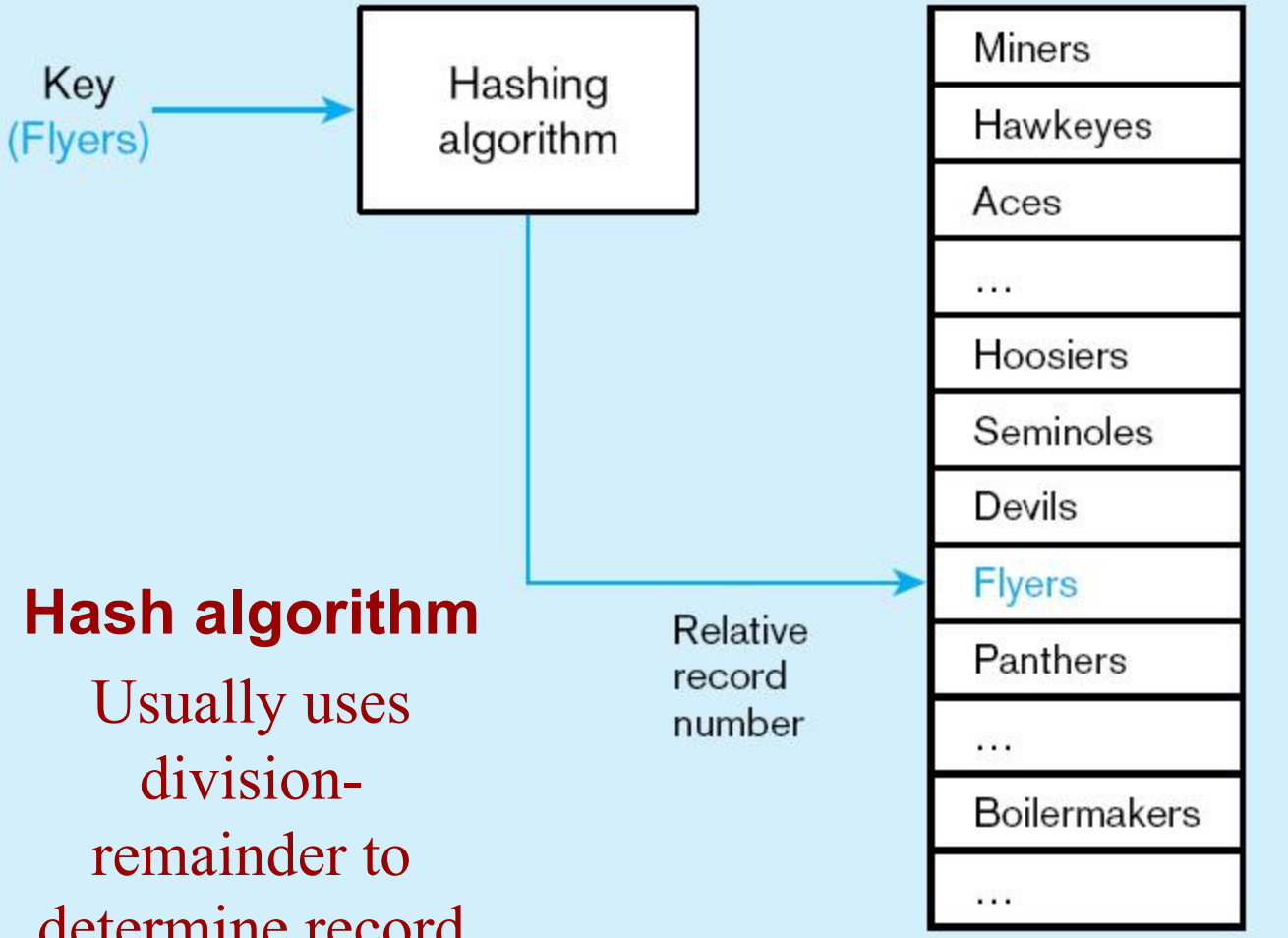
# Direct

- Key values of the physical records are not necessarily in logical sequence
- There is a one-to-one correspondence between a record key and the physical address of the record
- May be used for storage and retrieval
- Access efficiency always 1
- Storage efficiency depends on density of keys
- No duplicate keys permitted

# Hashing

- Key values of the physical records are not necessarily in logical sequence
- Many key values may share the same physical address (block)
- May be used for storage and retrieval
- Access efficiency depends on distribution of keys, algorithm for key transformation and space allocated
- Storage efficiency depends on distibution of keys and algorithm used for key transformation

Figure 5-7c **Hashed** file organization




© 2013 Pearson Education, Inc. Publishing as Prentice Hall

# Bitmap indexes

- Uses a single bit to represent whether or not a particular record has a specific value
- Useful for data with small numbers of possible values, with large numbers (N) of records

| Val | 1 | 2 | 3 | 4 | … | 100 | 101 | … | N |
|---|---|---|---|---|---|---|---|---|---|
| M | 1 | 0 | 0 | 1 | … | 1 | 0 | … | 0 |
| F | 0 | 1 | 1 | 0 | … | 0 | 1 | … | 1 |

# Comparative Access Methods

| Factor | Sequential | Indexed | Hashed |
|---|---|---|---|
| *Storage space* | No wasted space | No wasted space for data but extra space for index | more space needed for addition and deletion of records after initial load |
| *Sequential retrieval on primary key* | Very fast | Moderately Fast | Impractical |
| *Random Retr.* | Impractical | Moderately Fast | Very fast |
| *Multiple Key Retr.* | Possible but needs a full scan | Very fast with multiple indexes | Not possible |
| *Deleting records* | can create wasted space | OK if dynamic | very easy |
| *Adding records* | requires rewriting file | OK if dynamic | very easy |
| *Updating records* | usually requires rewriting file | Easy but requires Maintenance of indexes | very easy |

# Lecture Outline

- File and Access Methods
- Indexes and What to index
- Parallel storage systems (RAID)
- Integrity constraints
- Database Design Process Recap
- XML and databases – first look

# Indexes

- Most database applications require:
  - locating rows in tables that match some condition (e.g. SELECT operations)
  - Joining one table with another based on common values of attributes in each table
- Indexes can greatly speed up these processes and avoid having to do sequential scanning of database tables to resolve queries

# Figure 6-8 **Join** Indexes–speeds up join operations

a) Join index for common non-key columns

Customer

| RowID | Cust# | CustName | City | State |
|---|---|---|---|---|
| 10001 | C2027 | Hadley | Dayton | Ohio |
| 10002 | C1026 | Baines | Columbus | Ohio |
| 10003 | C0042 | Ruskin | Columbus | Ohio |
| 10004 | C3861 | Davies | Toledo | Ohio |
| . . . | | | | |

Store

| RowID | Store# | City | Size | Manager |
|---|---|---|---|---|
| 20001 | S4266 | Dayton | K2 | E2166 |
| 20002 | S2654 | Columbus | K3 | E0245 |
| 20003 | S3789 | Dayton | K4 | E3330 |
| 20004 | S1941 | Toledo | K1 | E0874 |
| . . . | | | | |

Join Index

| CustRowID | StoreRowID | Common Value* |
|---|---|---|
| 10001 | 20001 | Dayton |
| 10001 | 20003 | Dayton |
| 10002 | 20002 | Columbus |
| 10003 | 20002 | Columbus |
| 10004 | 20004 | Toledo |
| . . . | | |

*This column may or may not be included, as needed. Join index could be sorted on any of the three columns. Sometimes two join indexes are created, one as above and one with the two RowID columns reversed.

b) Join index for matching foreign key (FK) and primary key (PK)

Order

| RowID | Order# | Order Date | Cust#(FK) |
|---|---|---|---|
| 30001 | O5532 | 10/01/2001 | C3861 |
| 30002 | O3478 | 10/01/2001 | C1062 |
| 30003 | O8734 | 10/02/2001 | C1062 |
| 30004 | O9845 | 10/02/2001 | C2027 |
| . . . | | | |

Customer

| RowID | Cust#(PK) | CustName | City | State |
|---|---|---|---|---|
| 10001 | C2027 | Hadley | Dayton | Ohio |
| 10002 | C1062 | Baines | Columbus | Ohio |
| 10003 | C0042 | Ruskin | Columbus | Ohio |
| 10004 | C3861 | Davies | Toledo | Ohio |
| . . . | | | | |

Join Index

| CustRowID | OrderRowID | Cust# |
|---|---|---|
| 10001 | 30004 | C2027 |
| 10002 | 30002 | C1062 |
| 10002 | 30003 | C1062 |
| 10004 | 30001 | C3861 |
| . . . | | |

© 2013 Pearson Education, Inc. Publishing as Prentice Hall

# Type of Keys

- Primary keys -- as we have seen before -- uniquely identify a single row in a relational table
- Secondary keys -- are search keys that may occur multiple times in a table
- Bitmap Indexes
  - Table of bits where each row represents a distinct key value and each column is a bit – 0 or 1 for each record

# Primary Key Indexes

- In MySQL – will create a unique index
- In Access – also this will be created automatically when a field is selected as primary key
  - in the table design view select an attribute row (or rows) and clock on the key symbol in the toolbar.
  - The index is created automatically as one with (No Duplicates)
- In SQL
    - CREATE UNIQUE INDEX indexname ON tablename(attribute);

# Secondary Key Indexes

- In Access -- Secondary key indexes can be created on any field.
  - In the table design view, select the attribute to be indexed
  - In the "Indexed" box on the General field description information at the bottom of the window, select "Yes (Duplicates OK)"
- In SQL (including MySQL)
  - CREATE INDEX indxname on tablename(attribute);
- MySQL suggests that CREATE TABLE be used for most index creation. E.g. adding "*create table..,idnum int index using btree,*"

# MySQL Index Creation syntax

```
CREATE [ONLINE|OFFLINE] [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [index_option] ...

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
```

*CREATE INDEX cannot be used to create a PRIMARY KEY; use ALTER TABLE instead*

# When to Index

- Tradeoff between time and space:
  - Indexes permit faster processing for searching
  - But they take up space for the index
  - They also slow processing for insertions, deletions, and updates, because both the table and the index must be modified
- Thus they **SHOULD** be used for databases where search is the main mode of interaction
- The might be skipped if high rates of updating and insertions are expected, and access or retrieval operations are rare

# When to Use Indexes

- Rules of thumb
  - Indexes are most useful on larger tables
  - Specify a unique index for the primary key of each table (automatically done for many DBMS)
  - Indexes are most useful for attributes used as search criteria or for joining tables
  - Indexes are useful if *sorting* is often done on the attribute
  - Most useful when there are many different values for an attribute
  - Some DBMS limit the number of indexes and the size of the index key values
  - Some indexes will not retrieve NULL values

# Clustering Files

- In some relational DBMSs, related records from different tables can be stored together in the same disk area
- Useful for improving performance of join operations
- Primary key records of the main table are stored adjacent to associated foreign key records of the dependent table
- e.g. Oracle has a CREATE CLUSTER command

# Query Optimization

- Parallel query processing–possible when working in multiprocessor systems
- Overriding automatic query optimization–allows for query writers to preempt the automated optimization
- Oracle example:

```
SELECT /*+ FULL(Order_T) PARALLEL(Order_T,3) */ COUNT(*)
FROM Order_T
WHERE Salesperson = "Smith";
```

/* */ clause is a hint to override Oracle's default query plan

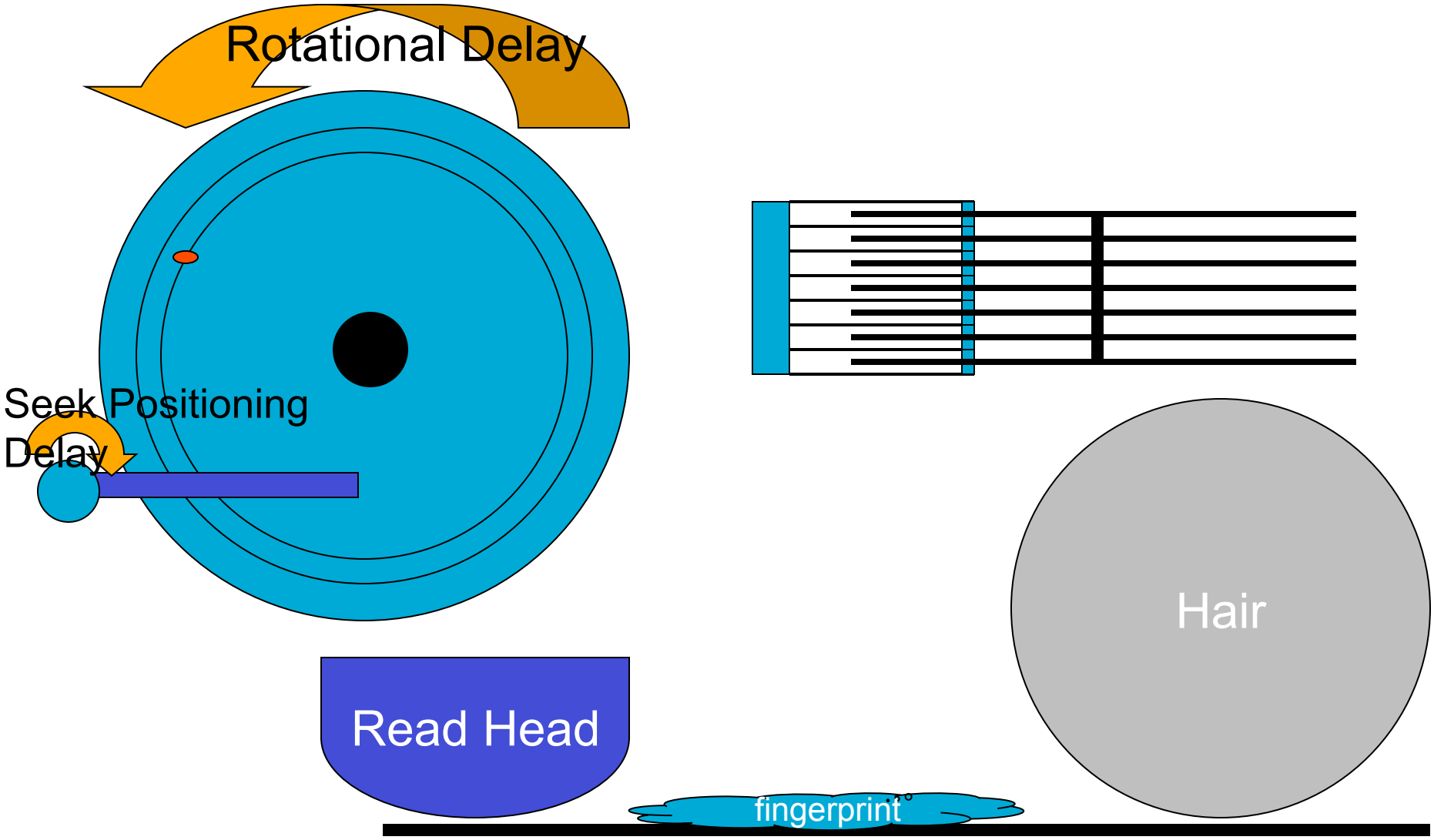
# Lecture Outline

- File and Access Methods
- Indexes and What to index
- Parallel storage systems (RAID)
- Integrity constraints
- Database Design Process Recap
- XML and databases – first look

# Parallel Processing with RAID

- In reading pages from secondary storage, there are often situations where the DBMS must retrieve multiple pages of data from storage -- and may often encounter
  - rotational delay
  - seek positioning delay
- In getting each page from the disk

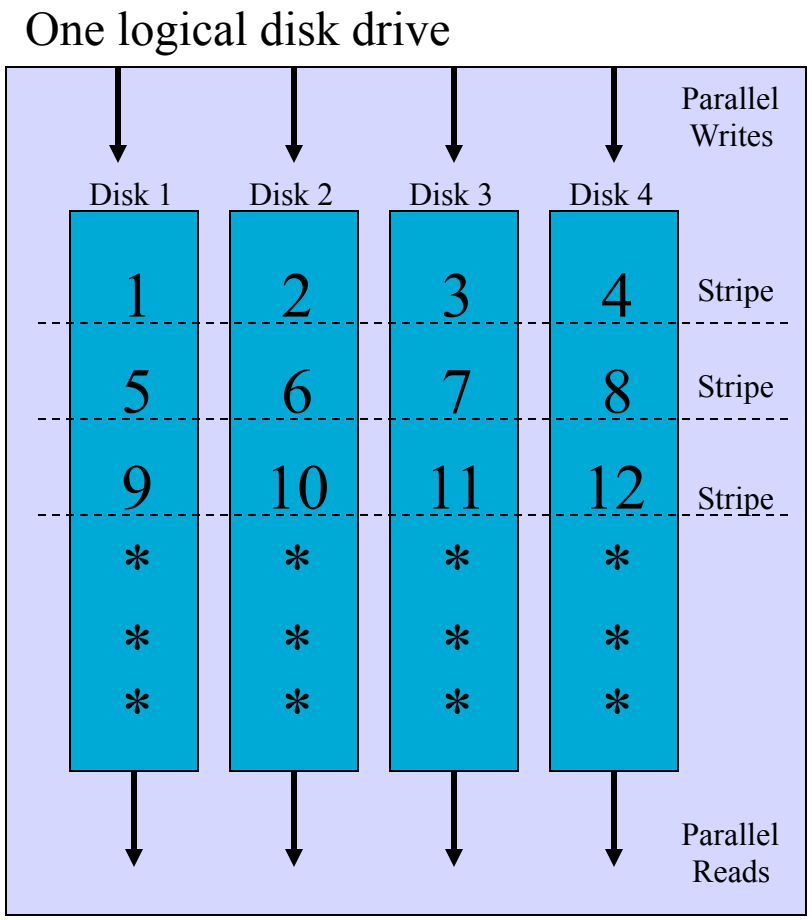
# Disk Timing (and Problems)



# RAID

- Provides parallel disks (and software) so that multiple pages can be retrieved simultaneously
- RAID stands for “Redundant Arrays of Inexpensive Disks”
  - invented by Randy Katz and Dave Patterson here at Berkeley
- Some manufacturers have renamed the “inexpensive” part (for obvious reasons)

# RAID Technology



# Lecture Outline

- File and Access Methods
- Indexes and What to index
- Parallel storage systems (RAID)
- Integrity constraints
- Database Design Process Recap
- XML and databases - first look

# Integrity Constraints

- The constraints we wish to impose in order to protect the database from becoming inconsistent.
- Five types
  - Required data
  - attribute domain constraints
  - entity integrity
  - referential integrity
  - enterprise constraints

# Integrity Constraints

- The constraints we wish to impose in order to protect the database from becoming inconsistent.
- Five types
  - Required data
  - attribute domain constraints
  - entity integrity
  - referential integrity
  - enterprise constraints

# Integrity constraints

- Usually set during table creation in RDBMS
- May also be set or modified by ALTER TABLE

**CREATE [TEMPORARY] TABLE [IF NOT EXISTS] *tbl_name* (*create_definition*,...) [*table_options*]**

# Required Data

- Some attributes must always contain a value -- they cannot have a null
- For example:
  - Every employee must have a job title.
  - Every diveshop diveitem must have an order number and an item number.

# Attribute Domain Constraints

- Every attribute has a domain, that is a set of values that are legal for it to use
- For example:
  - The domain of sex in the employee relation is "M" or "F"
- Domain ranges can be used to validate input to the database.

# E.g. – in SQLite

- sqlite> CREATE TABLE tst (num integer CHECK (num < 100));
- sqlite> insert into tst (num) values (1);
- sqlite> select * from tst;
- 1
- sqlite> insert into tst (num) values (80);
- sqlite> insert into tst (num) values (99);
- sqlite> insert into tst (num) values (100);
- Error: constraint failed

# Entity Integrity

- The primary key of any entity cannot be NULL.

# Column Definitions in MySQL

- *column_definition*:

  *data_type* [NOT NULL | NULL]

  [DEFAULT *default_value*]
  [AUTO_INCREMENT]

  [UNIQUE [KEY] | [PRIMARY] KEY]

  [COMMENT '*string*']

  [COLUMN_FORMAT {FIXED|DYNAMIC|
  DEFAULT}]

  [STORAGE {DISK|MEMORY|DEFAULT}]
  [*reference_definition*]

# Referential Integrity

- A “foreign key” links each occurrence in a relation representing a *child* entity to the occurrence of the *parent* entity containing the matching candidate key
- Referential Integrity means that if the foreign key contains a value, that value *must refer to an existing occurrence* in the parent entity
- For example:
  - Since the ‘Order ID’ in the diveitem relation refers to a particular diveords primary key, that key –and row- must exist for referential integrity to be satisfied

# Referential Integrity

- Referential integrity options are declared when tables are defined (in most systems)
- There are many issues having to do with how particular referential integrity constraints are to be implemented to deal with insertions and deletions of data from the parent and child tables.

# Insertion rules

- A row should not be inserted in the referencing (child) table unless there already exists a matching entry in the referenced table.
- Inserting into the parent table should *not* cause referential integrity problems
  - Unless it is itself a child…
- Sometimes a special NULL value may be used to create child entries without a parent or with a “dummy” parent.

# Deletion rules

- A row should not be deleted from the referenced table (parent) if there are matching rows in the referencing table (child).
- Three ways to handle this
  - **Restrict** -- disallow the delete
  - **Nullify** -- reset the foreign keys in the child to some NULL or dummy value
  - **Cascade** -- Delete all rows in the child where there is a foreign key matching the key in the parent row being deleted

# Referential Integrity

- This can be implemented using external programs that access the database
- newer databases implement executable rules or built-in integrity constraints
- For example in Diveshop…

# E.g. – in MySQL

- *reference_definition*:

  REFERENCES *tbl_name* (*index_col_name*,...)
[MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]

  [ON DELETE *reference_option*]

  [ON UPDATE *reference_option*]

- *reference_option*:

  RESTRICT | CASCADE | SET NULL | NO ACTION

# DIVEORDS SQL

```
CREATE TABLE `DIVEORDS` (
`Order_No` int(11) NOT NULL,
`Customer_No` int(11) default NULL,
`Sale_Date` datetime default NULL,
`Ship_Via` varchar(255) default NULL,
`Ship_Cost` double default NULL,
…some things deleted for space…,
`VacationCost` double default NULL,
PRIMARY KEY (`Order_No`),
KEY `Customer_No` (`Customer_No`),
KEY `DESTDIVEORDS` (`Destination`),
KEY `DIVECUSTDIVEORDS` (`Customer_No`),
KEY `DIVEORDSShip_Via` (`Ship_Via`),
KEY `SHIPVIADIVEORDS` (`Ship_Via`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

# DIVEITEM SQL

Note that only the InnoDB or NDB Engines in MySQL support actual actions and checking for Foreign Keys

```
CREATE TABLE `DIVEITEM` (
`Order_No` int(11) NOT NULL,
`Item_No` int(11) default NULL,
 `Rental_Sale` varchar(255) default NULL,
`Qty` smallint(6) default NULL,
`Line_Note` varchar(255) default NULL,
KEY `DIVEORDSDIVEITEM` (`Order_No`),
KEY `DIVESTOKDIVEITEM` (`Item_No`),
KEY `Item_No` (`Item_No`),
FOREIGN KEY (`Order_No`) REFERENCES
        DIVEORDS(`Order_No`) ON DELETE CASCADE )
ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

# Handling Missing Data

- Substitute an estimate of the missing value (e.g., using a formula)
- Construct a report listing missing values
- In programs, ignore missing data unless the value is significant (sensitivity testing)

**Triggers can be used to perform these operations.**

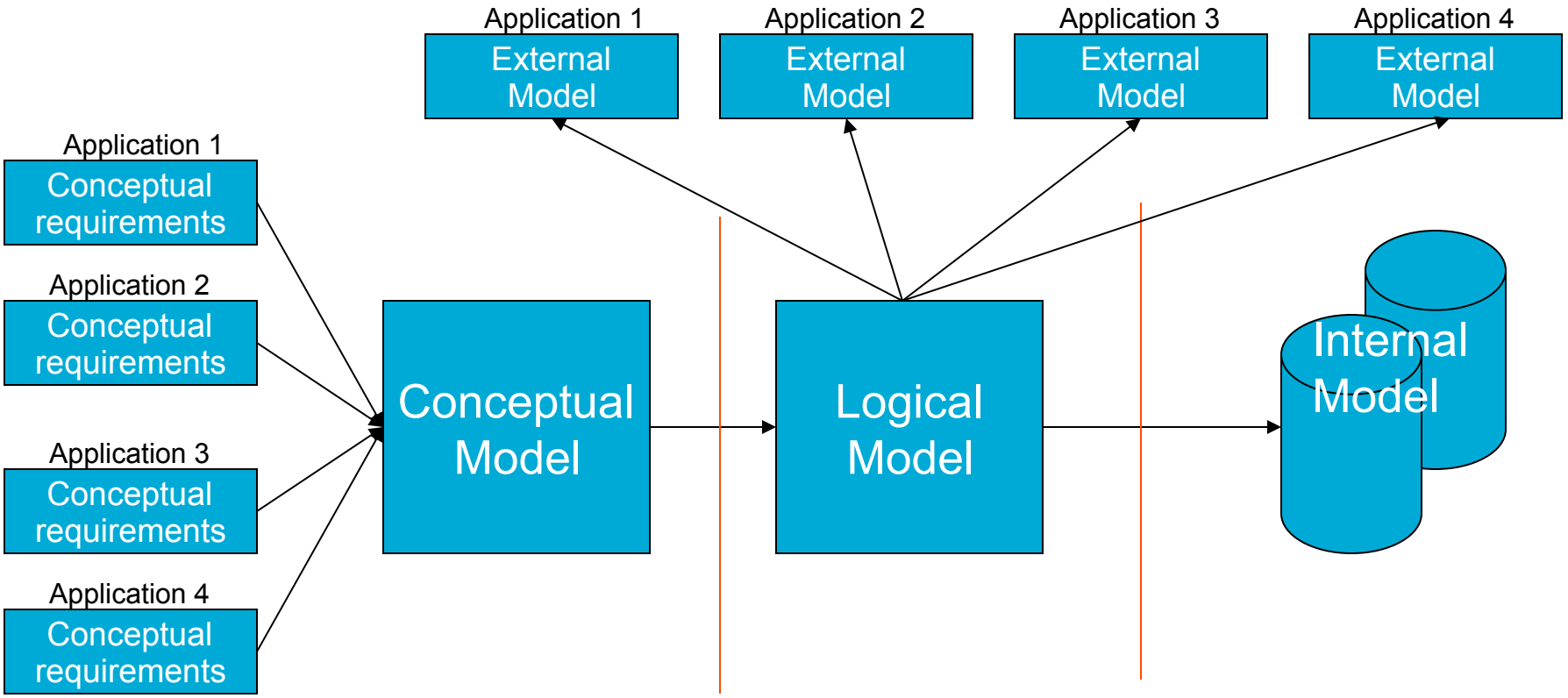
# Enterprise Constraints

- These are business rule that may affect the database and the data in it
  - for example, if a manager is only permitted to manage 10 employees then it would violate an enterprise constraint to manage more

# Lecture Outline

- File and Access Methods
- Indexes and What to index
- Parallel storage systems (RAID)
- Integrity constraints
- Database Design Process Recap
- XML and databases – first look

# Database Design Process



# Today: New Design

- Today we will build the COOKIE database from (rough) needs assessment through the conceptual model, logical model and finally physical implementation in Access.

# Cookie Requirements

- Cookie is a bibliographic database that contains information about a hypothetical union catalog of several libraries.
- Need to record which books are held by which libraries
- Need to search on bibliographic information
  - Author, title, subject, call number for a given library, etc.
- Need to know who publishes the books for ordering, etc.

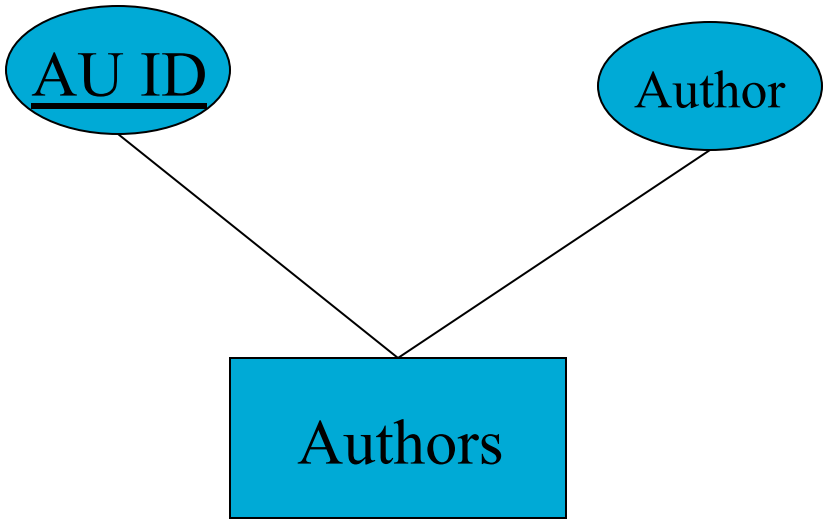
# Cookie Database

- There are currently 6 main types of entities in the database
  - Authors (Authors)
    - Note: we created authors from the former design when talking about normalization (a few weeks ago)
  - Books (bibfile)
  - Local Call numbers (callfile)
  - Libraries (libfile)
  - Publishers (pubfile)
  - Subject headings (subfile)
  - Additional entities
    - Links between subject and books (indxfile)
    - Links between authors and books (AU_BIB)

# AUTHORS

- Author -- The author's name (We do not distinguish between Personal and Corporate authors)
- Au_id – a unique id for the author

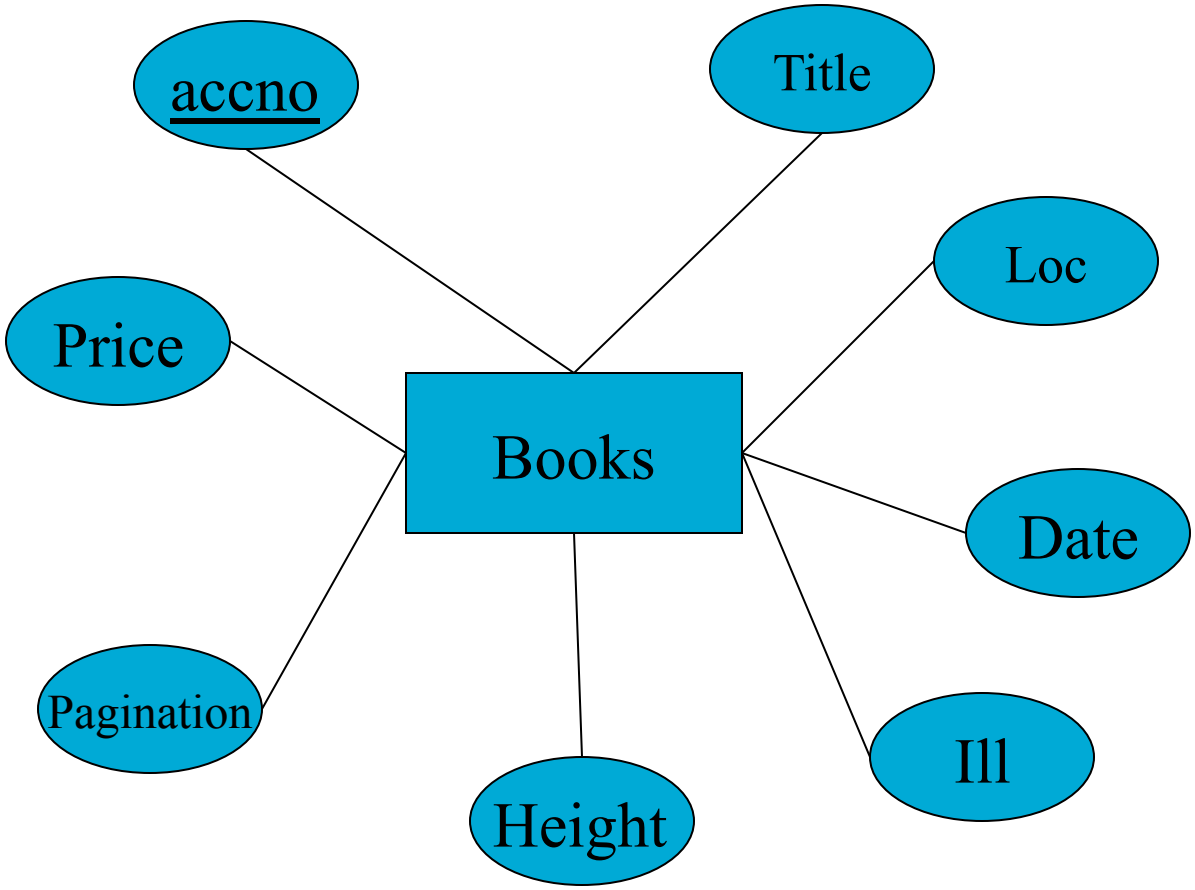
# AUTHORS



# BIBFILE

- Books (BIBFILE) contains information about particular books. It includes one record for each book. The attributes are:
  - accno -- an “accession” or serial number
  - title -- The title of the book
  - loc -- Location of publication (where published)
  - date -- Date of publication
  - price -- Price of the book
  - pagination -- Number of pages
  - ill -- What type of illustrations (maps, etc) if any
  - height -- Height of the book in centimeters

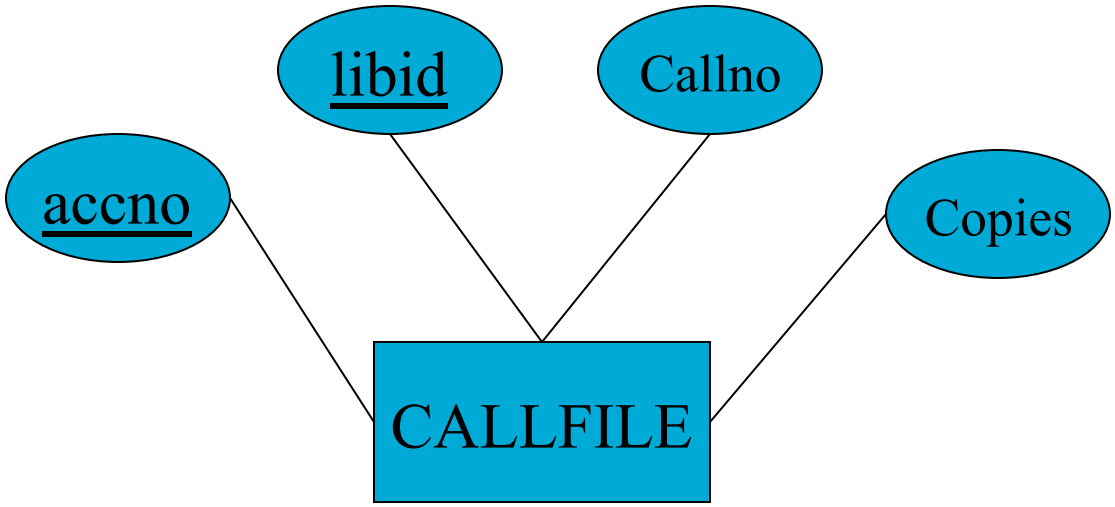
# Books/BIBFILE



# CALLFILE

- CALLFILE contains call numbers and holdings information linking particular books with particular libraries. Its attributes are:
  - accno -- the book accession number
  - libid -- the id of the holding library
  - callno -- the call number of the book in the particular library
  - copies -- the number of copies held by the particular library

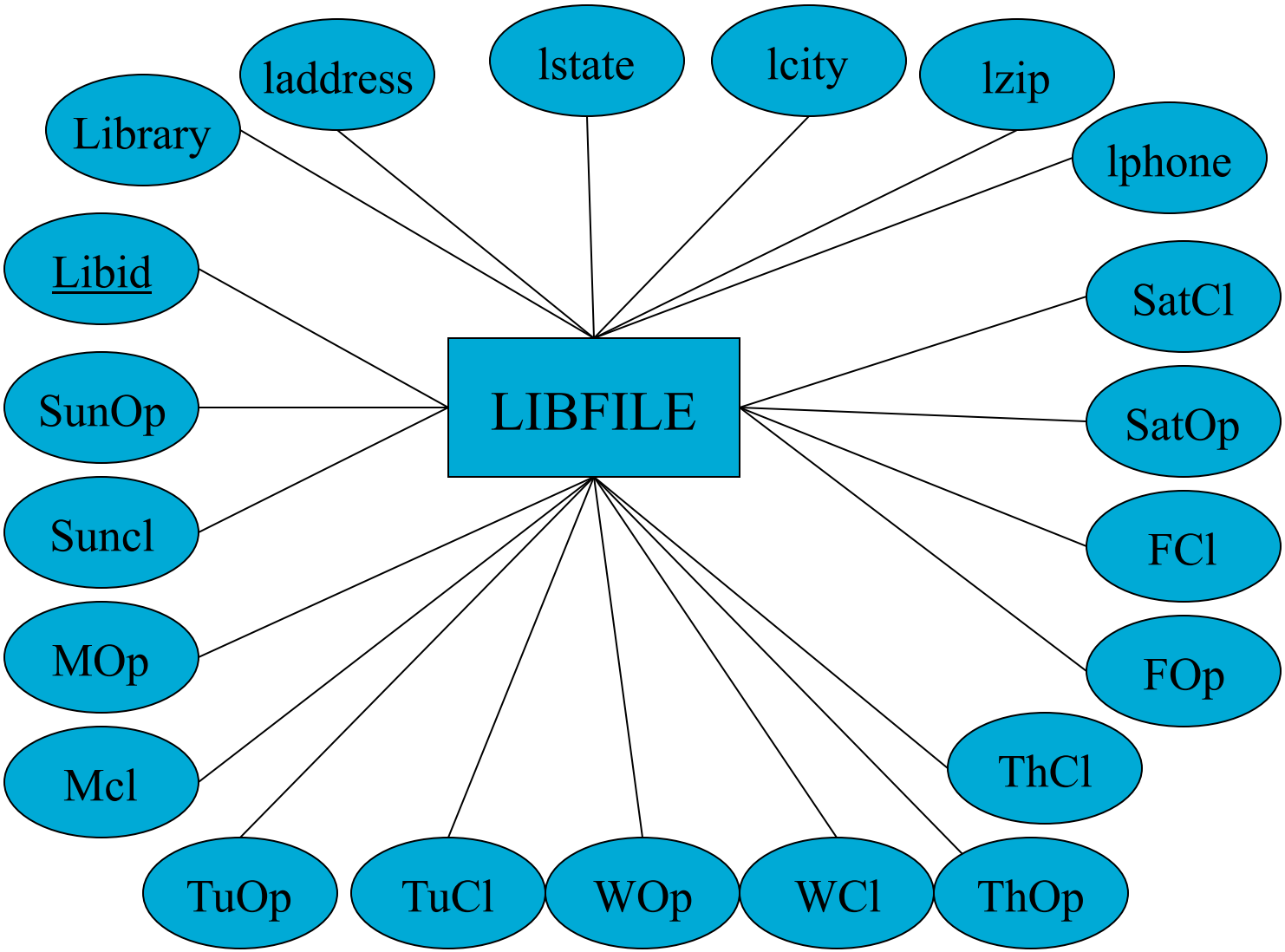
# LocalInfo/CALLFILE



# LIBFILE

- LIBFILE contain information about the libraries participating in this union catalog. Its attributes include:
  - libid -- Library id number
  - library -- Name of the library
  - laddress -- Street address for the library
  - lcity -- City name
  - lstate -- State code (postal abbreviation)
  - lzip -- zip code
  - lphone -- Phone number
  - mop - suncl -- Library opening and closing times for each day of the week.

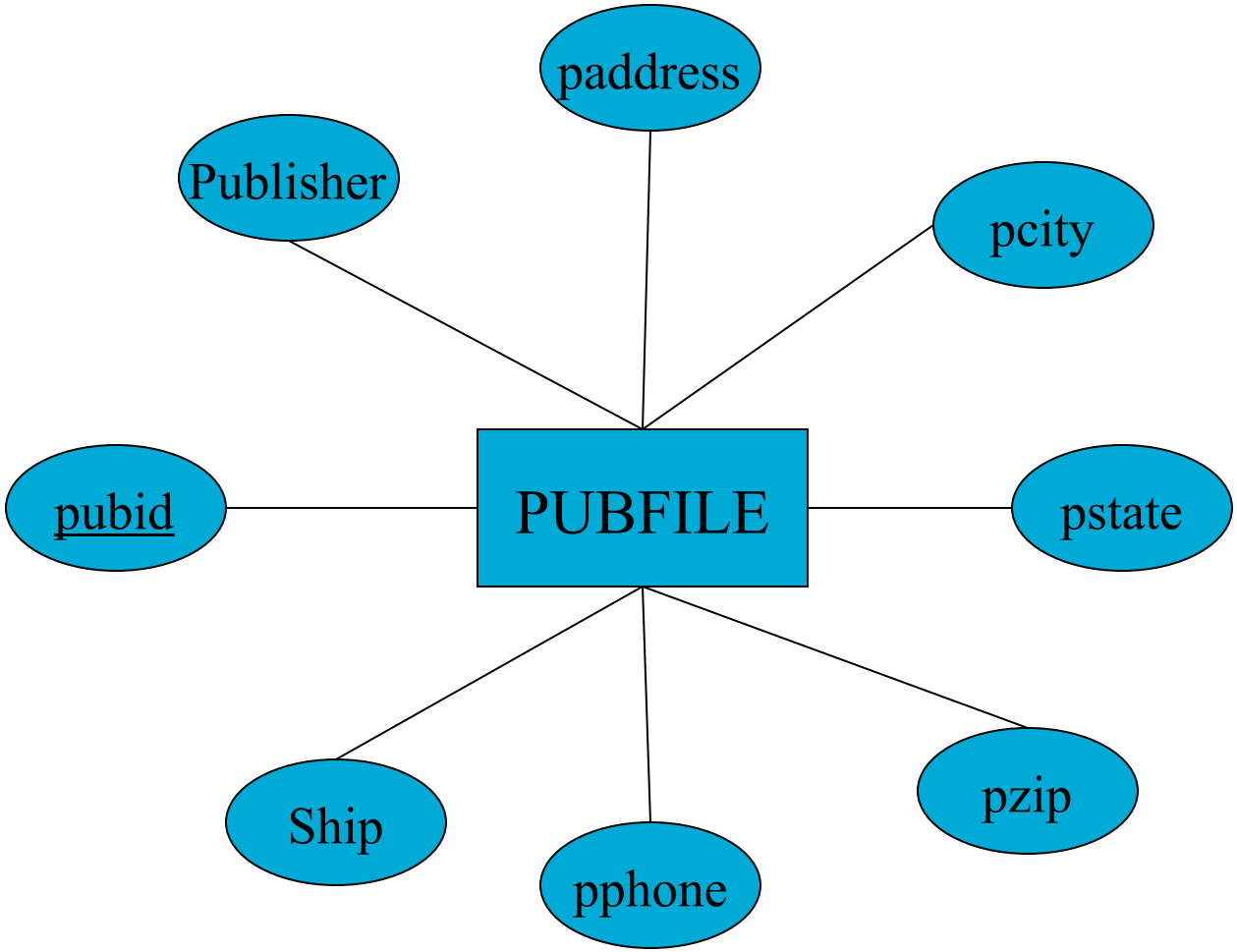
# Libraries/LIBFILE



# PUBFILE

- PUBFILE contain information about the publishers of books. Its attributes include
  - pubid -- The publisher's id number
  - publisher -- Publisher name
  - paddress -- Publisher street address
  - pcity -- Publisher city
  - pstate -- Publisher state
  - pzip -- Publisher zip code
  - pphone -- Publisher phone number
  - ship -- standard shipping time in days

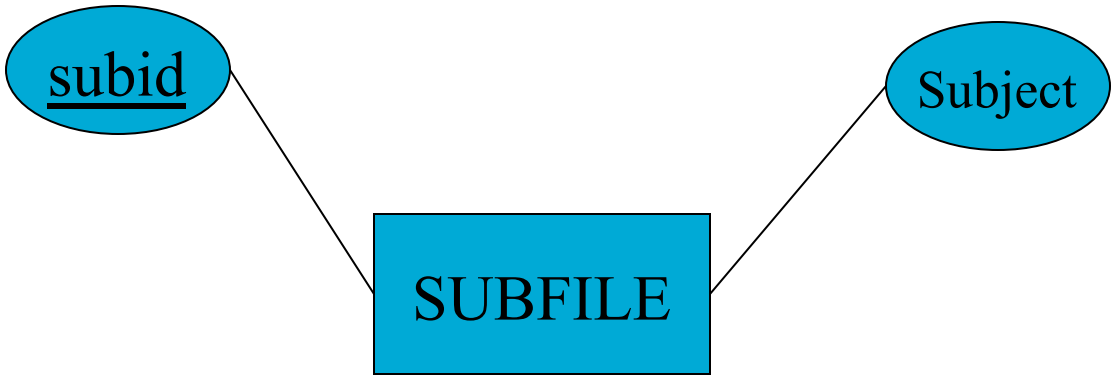
# Publisher/PUBFILE



# SUBFILE

- SUBFILE contains each unique subject heading that can be assigned to books. Its attributes are
  - subcode -- Subject identification number
  - subject -- the subject heading/description

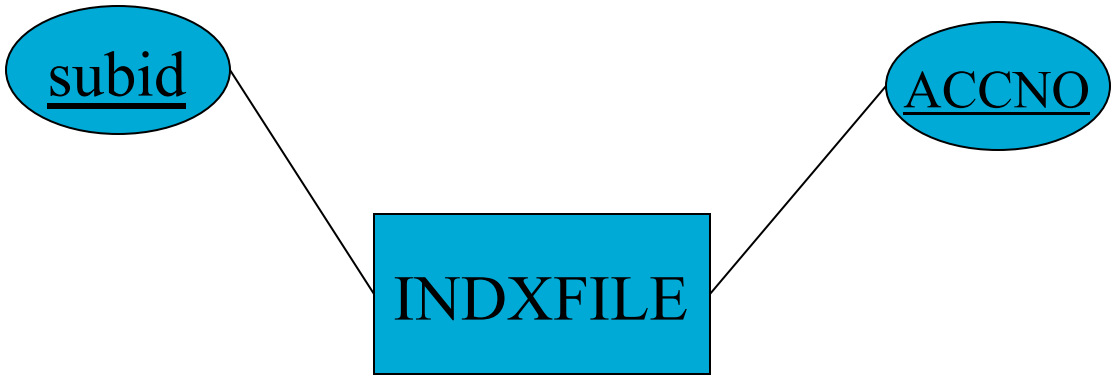
# Subjects/SUBFILE



# INDXFILE

- INDXFILE provides a way to allow many-to-many mapping of subject headings to books. Its attributes consist entirely of links to other tables
  - subcode -- link to subject id
  - accno -- link to book accession number

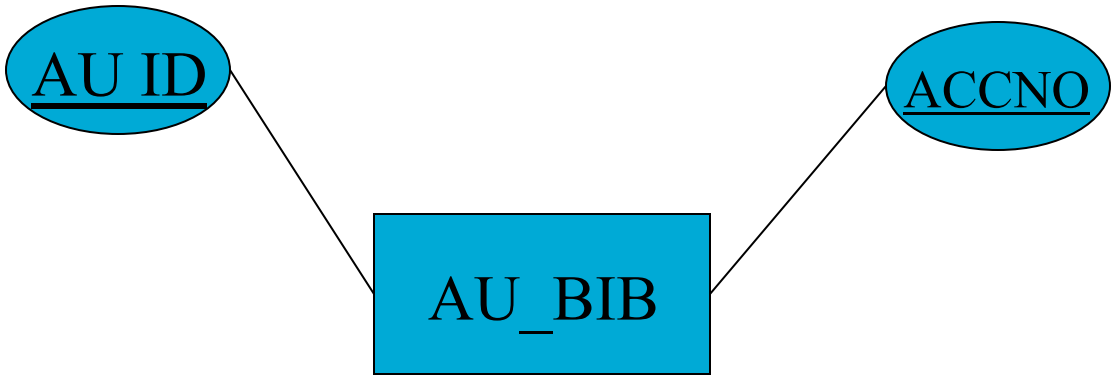
# Linking Subjects and Books



# AU_BIB

- AU_BIB provides a way to allow many to many mapping between books and authors. It also consists only of links to other tables
  - AU_ID – link to the AUTHORS table
  - ACCNO – link to the BIBFILE table

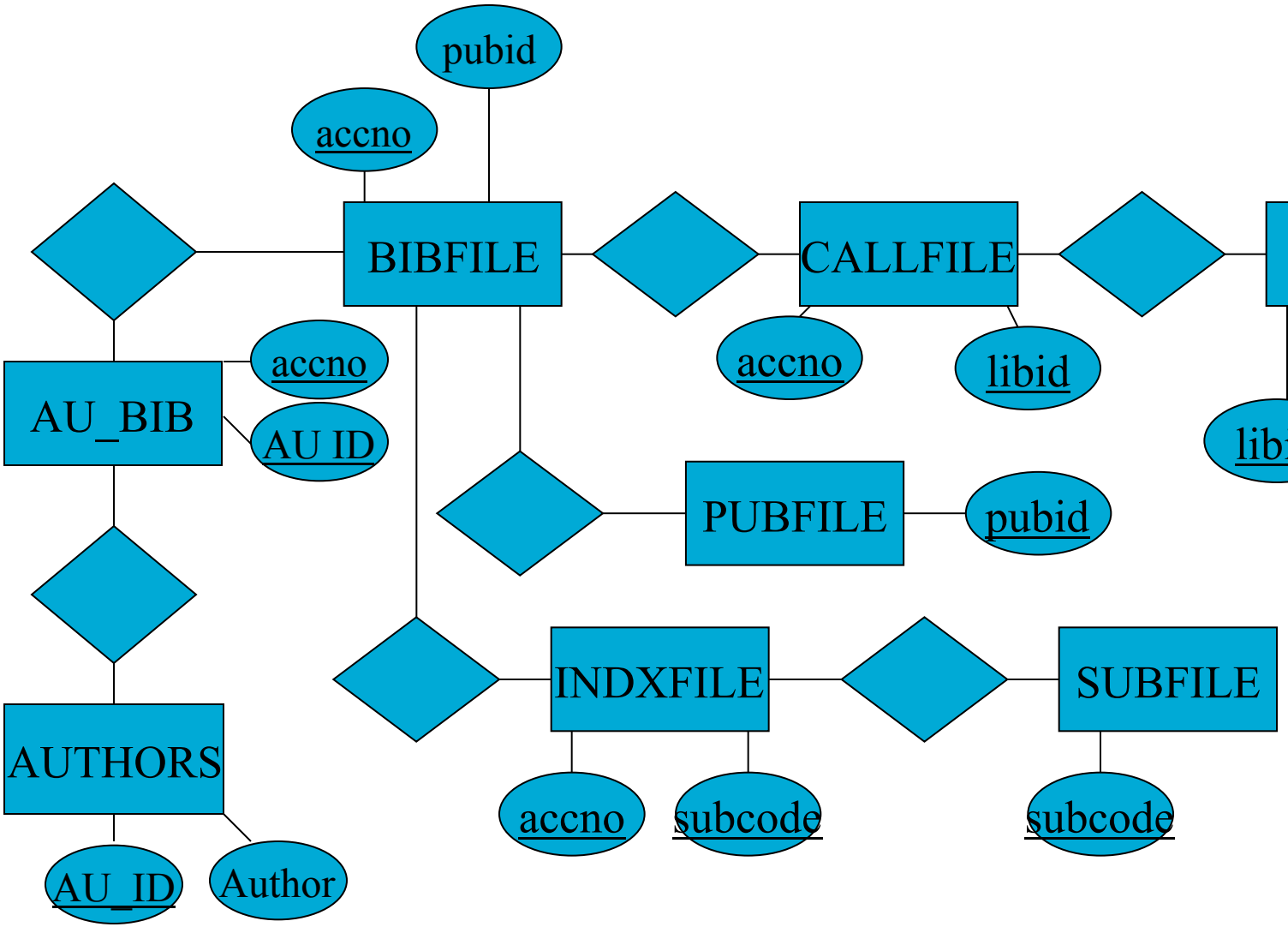
# Linking Authors and Books



# Some examples of Cookie Searches

- Who wrote Microcosmographia Academica?
- How many pages long is Alfred Whitehead's The Aims of Education and Other Essays?
- Which branches in Berkeley's public library system are open on Sunday?
- What is the call number of Moffitt Library's copy of Abraham Flexner's book Universities: American, English, German?
- What books on the subject of higher education are among the holdings of Berkeley (both UC and City) libraries?
- Print a list of the Mechanics Library holdings, in descending order by height.
- What would it cost to replace every copy of each book that contains illustrations (including graphs, maps, portraits, etc.)?
- Which library closes earliest on Friday night?

# Cookie ER Diagram




Note: diagram contains only attributes used for linking

# What Problems?

- What sorts of problems and missing features arise given the previous ER diagram?

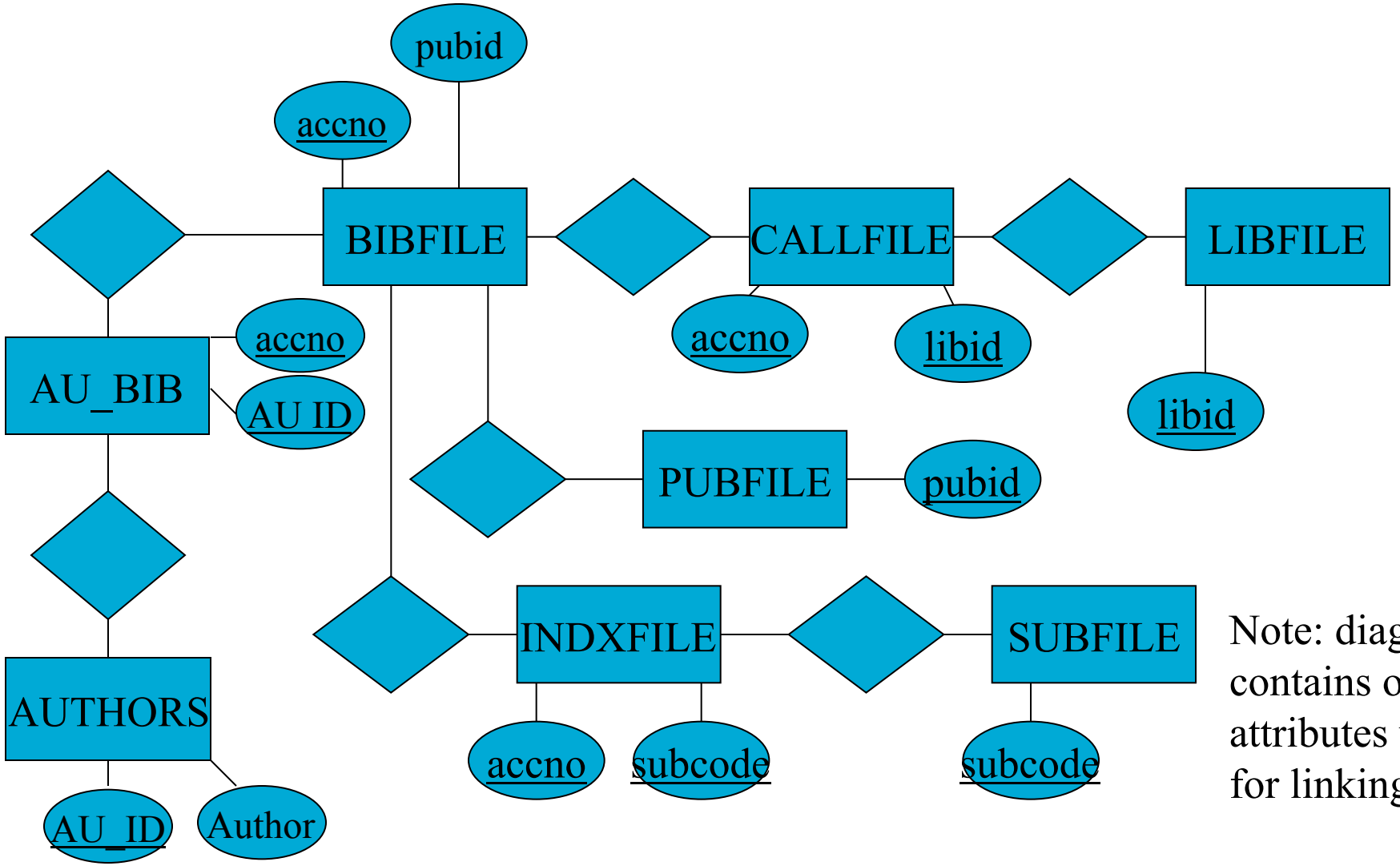
# Problems Identified

- Subtitles, parallel titles?
- Edition information
- Series information
- lending status
- material type designation
- Genre, class information
- Better codes (ISBN?)
- Missing information (ISBN)
- Authority control for authors
- Missing/incomplete data
- Data entry problems
- Ordering information
- Illustrations
- Subfield separation (such as last_name, first_name)
- Separate personal and corporate authors

# Problems (Cont.)

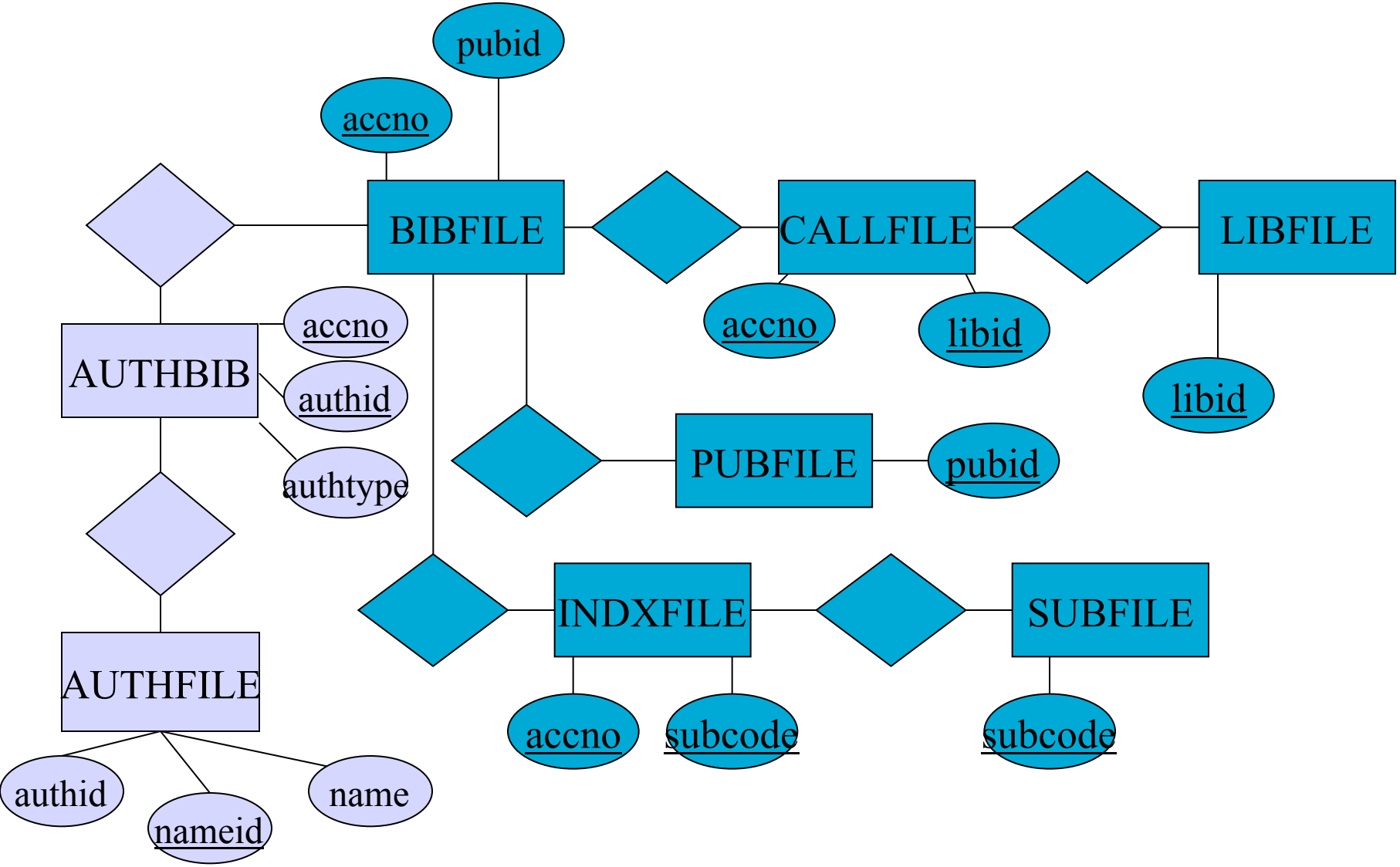
- Location field inconsistent
- No notes field
- No language field
- Zipcode doesn't support plus-4
- No publisher shipping addresses
- No (indexable) keyword search capability
- No support for multivolume works
- No support for URLs
  - to online version
  - to libraries
  - to publishers

# Original Cookie ER Diagram

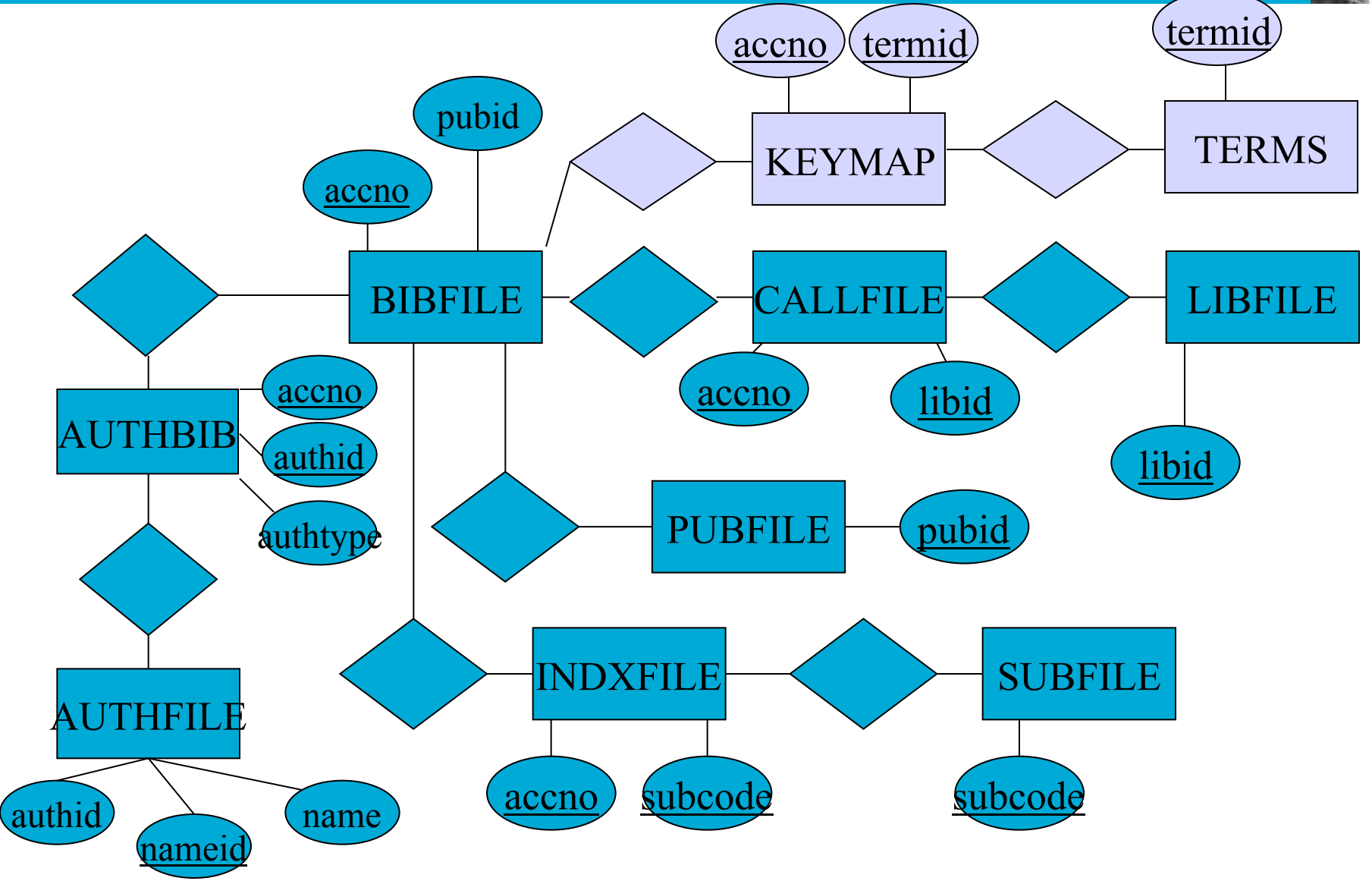



Note: diagram contains only attributes used for linking

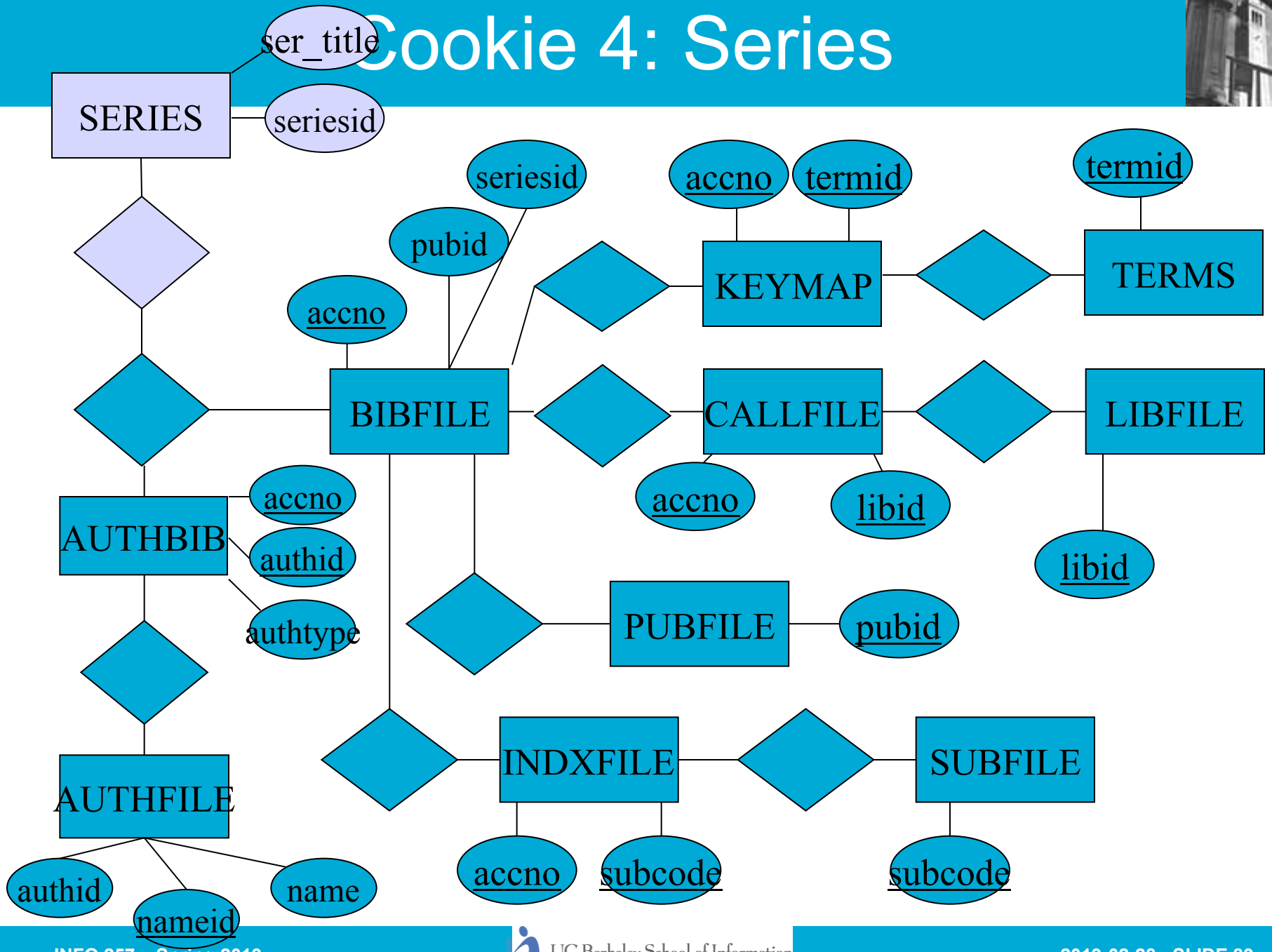
# Cookie2: Separate Name Authorities



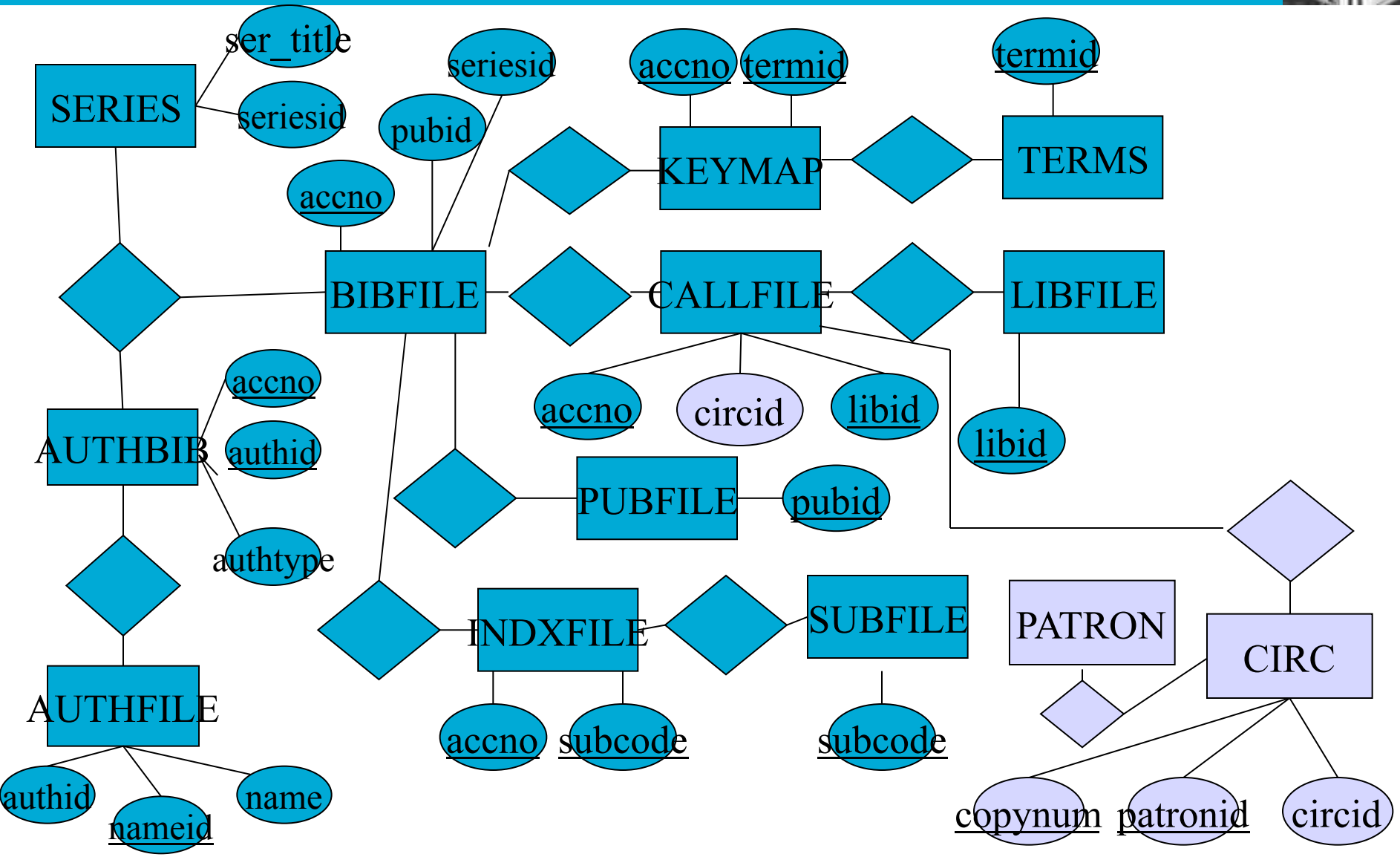
# Cookie 3: Keywords



# Cookie 4: Series



# Cookie 5: Circulation



# Logical Model: Mapping to Relations

- Take each entity
  - Authors
  - BIBFILE
  - LIBFILE
  - CALLFILE
  - SUBFILE
  - PUBFILE
  - INDXFILE
  - AU_BIB
- And make it a table...

# Implementing the Physical Database...

- For each of the entities, we will build a table…
- Loading data
- Entering data
- Data entry forms

# Lecture Outline

- File and Access Methods
- Indexes and What to index
- Parallel storage systems (RAID)
- Integrity constraints
- Database Design Process Recap
- XML and databases – first look

# Why XML?

- As part of the SQL Standards there is an extension providing a mapping from XML to DBMS is being created called XML/SQL
- The (draft) standard is very complex, but the ideas are actually pretty simple
- Suppose we have a table called EMPLOYEE that has columns EMPNO, FIRSTNAME, LASTNAME, BIRTHDATE, SALARY

# Standards: XML/SQL

- That table can be mapped to:

```
<EMPLOYEE>
<row><EMPNO>000020</EMPNO>
<FIRSTNAME>John</FIRSTNAME>
<LASTNAME>Smith</LASTNAME>
<BIRTHDATE>1955-08-21</BIRTHDATE>
<SALARY>52300.00</SALARY>
</row>
 <row> … etc. …
 </EMPLOYEE>
```

# Standards: XML/SQL

- In addition the standard says that XMLSchemas must be generated for each table, and also allows relations to be managed by nesting records from tables in the XML.
- Variants of this are incorporated into the latest versions of ORACLE and in MySQL
- But what if you want to deal with more complex XML schemas (beyond “flat” structures)?

# Native XML Database (NXD)

- Native XML databases have an XML-based internal model
  - That is, their fundamental unit of storage is XML
- However, different native XML databases differ in what they consider the fundamental unit of storage
  - Document vs element or segment
- And how that information or its subelements are accessed, indexed and queried
  - E.g., SQL vs. Xquery or a special query language

# Database Systems supporting XQuery

- The following database systems offer XQuery support:
  - *Native XML Databases:*
    - Berkeley DB XML
    - eXist
    - MarkLogic
    - Software AG Tamino
    - Raining Data TigerLogic
    - Documentum xDb (X-Hive/DB) (now EMC)
  - *Relational Databases (also support SQL):*
    - IBM DB2
    - Microsoft SQL Server
    - Oracle

# Further comments on NXD

- Native XML databases are most often used for storing “document-centric” XML document
  - I.e. the unit of retrieval would typically be the entire document and not a particular node or subelement
- This supports query languages like Xquery
  - Able to ask for “all documents where the third chapter contains a page that has boldfaced word”
  - Very difficult to do that kind of query in SQL