



Database Management Systems



Chapter 1

Instructor: Raghu Ramakrishnan
raghu@cs.wisc.edu

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

1



What Is a DBMS?



- ❖ A very large, integrated collection of data.
- ❖ Models real-world enterprise.
 - Entities (e.g., students, courses)
 - Relationships (e.g., Madonna is taking CS564)
- ❖ A Database Management System (DBMS) is a software package designed to store and manage databases.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

2



Files vs. DBMS

- ❖ Application must stage large datasets between main memory and secondary storage (e.g., buffering, page-oriented access, 32-bit addressing, etc.)
- ❖ Special code for different queries
- ❖ Must protect data from inconsistency due to multiple concurrent users
- ❖ Crash recovery
- ❖ Security and access control

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

3

Why Use a DBMS?



- ❖ Data independence and efficient access.
- ❖ Reduced application development time.
- ❖ Data integrity and security.
- ❖ Uniform data administration.
- ❖ Concurrent access, recovery from crashes.

Why Study Databases??



- ❖ Shift from computation to information
 - at the "low end": scramble to webspace (a mess!)
 - at the "high end": scientific applications
- ❖ Datasets increasing in diversity and volume.
 - Digital libraries, interactive video, Human Genome project, EOS project
 - ... need for DBMS exploding
- ❖ DBMS encompasses most of CS
 - OS, languages, theory, "A"l, multimedia, logic

Data Models

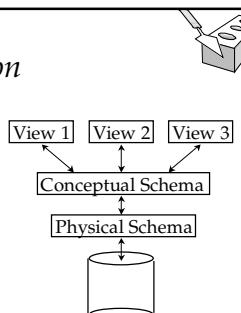


- ❖ A data model is a collection of concepts for describing data.
- ❖ A schema is a description of a particular collection of data, using a given data model.
- ❖ The relational model of data is the most widely used model today.
 - Main concept: relation, basically a table with rows and columns.
 - Every relation has a schema, which describes the columns, or fields.

Levels of Abstraction

- ❖ Many views, single conceptual (logical) schema and physical schema.

- Views describe how users see the data.
- Conceptual schema defines logical structure
- Physical schema describes the files and indexes used.



* Schemas are defined using DDL; data is modified/queried using DML.

Example: University Database

❖ Conceptual schema:

- *Students(sid: string, name: string, login: string, age: integer, gpa:real)*
- *Courses(cid: string, cname:string, credits:integer)*
- *Enrolled(sid:string, cid:string, grade:string)*

❖ Physical schema:

- Relations stored as unordered files.
- Index on first column of Students.

❖ External Schema (View):

- *Course_info(cid:string,enrollment:integer)*

Data Independence *

- ❖ Applications insulated from how data is structured and stored.
- ❖ Logical data independence: Protection from changes in *logical structure* of data.
- ❖ Physical data independence: Protection from changes in *physical structure* of data.

* One of the most important benefits of using a DBMS!

Concurrency Control

- ❖ Concurrent execution of user programs is essential for good DBMS performance.
 - Because disk accesses are frequent, and relatively slow, it is important to keep the CPU humming by working on several user programs concurrently.
- ❖ Interleaving actions of different user programs can lead to inconsistency: e.g., check is cleared while account balance is being computed.
- ❖ DBMS ensures such problems don't arise: users can pretend they are using a single-user system.

Transaction: An Execution of a DB Program

- ❖ Key concept is transaction, which is an *atomic* sequence of database actions (reads/writes).
- ❖ Each transaction, executed completely, must leave the DB in a consistent state if DB is consistent when the transaction begins.
 - Users can specify some simple integrity constraints on the data, and the DBMS will enforce these constraints.
 - Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
 - Thus, ensuring that a transaction (run alone) preserves consistency is ultimately the user's responsibility!

Scheduling Concurrent Transactions

- ❖ DBMS ensures that execution of $\{T_1, \dots, T_n\}$ is equivalent to some serial execution $T'_1 \dots T'_n$.
 - Before reading/writing an object, a transaction requests a lock on the object, and waits till the DBMS gives it the lock. All locks are released at the end of the transaction. (Strict 2PL locking protocol.)
 - Idea: If an action of T_i (say, writing X) affects T_j (which perhaps reads X), one of them, say T_i , will obtain the lock on X first and T_j is forced to wait until T_i completes; this effectively orders the transactions.
 - What if T_j already has a lock on Y and T_i later requests a lock on Y? (Deadlock!) T_i or T_j is aborted and restarted!

Ensuring Atomicity

- ❖ DBMS ensures *atomicity* (all-or-nothing property) even if system crashes in the middle of a Xact.
- ❖ Idea: Keep a log (history) of all actions carried out by the DBMS while executing a set of Xacts:
 - Before a change is made to the database, the corresponding log entry is forced to a safe location. (WAL protocol: OS support for this is often inadequate.)
 - After a crash, the effects of partially executed transactions are undone using the log. (Thanks to WAL, if log entry wasn't saved before the crash, corresponding change was not applied to database!)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

13

The Log



- ❖ The following actions are recorded in the log:
 - *Ti writes an object*: the old value and the new value.
 - Log record must go to disk *before* the changed page!
 - *Ti commits/aborts*: a log record indicating this action.
- ❖ Log records chained together by Xact id, so it's easy to undo a specific Xact (e.g., to resolve a deadlock).
- ❖ Log is often *duplexed* and *archived* on "stable" storage.
- ❖ All log related activities (and in fact, all CC related activities such as lock/unlock, dealing with deadlocks etc.) are handled transparently by the DBMS.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

14

Databases make these folks happy ...



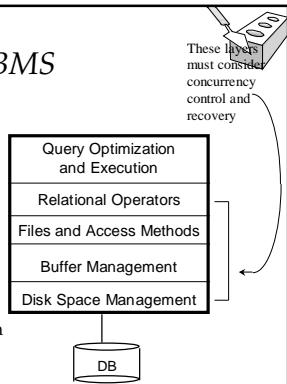
- ❖ End users and DBMS vendors
 - ❖ DB application programmers
 - E.g. smart webmasters
 - ❖ Database administrator (DBA)
 - Designs logical /physical schemas
 - Handles security and authorization
 - Data availability, crash recovery
 - Database tuning as needs evolve
- Must understand how a DBMS works!*

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

15

Structure of a DBMS

- ❖ A typical DBMS has a layered architecture.
- ❖ The figure does not show the concurrency control and recovery components.
- ❖ This is one of several possible architectures; each system has its own variations.



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

16

Summary

- ❖ DBMS used to maintain, query large datasets.
- ❖ Benefits include recovery from system crashes, concurrent access, quick application development, data integrity and security.
- ❖ Levels of abstraction give data independence.
- ❖ A DBMS typically has a layered architecture.
- ❖ DBAs hold responsible jobs and are well-paid!
- ❖ DBMS R&D is one of the broadest, most exciting areas in CS.



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

17

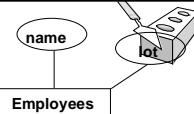
The Entity-Relationship Model

Chapter 2

Overview of Database Design

- ❖ Conceptual design: (ER Model is used at this stage.)
 - What are the *entities* and *relationships* in the enterprise?
 - What information about these entities and relationships should we store in the database?
 - What are the *integrity constraints* or *business rules* that hold?
 - A database ‘schema’ in the ER Model can be represented pictorially (*ER diagrams*).
 - Can map an ER diagram into a relational schema.

ER Model Basics



- ❖ Entity: Real-world object distinguishable from other objects. An entity is described (in DB) using a set of *attributes*.
- ❖ Entity Set: A collection of similar entities. E.g., all employees.
 - All entities in an entity set have the same set of attributes. (Until we consider ISA hierarchies, anyway!)
 - Each entity set has a *key*.
 - Each attribute has a *domain*.

ER Model Basics (Contd.)

```

    erDiagram
        class Employee {
            string ssn;
            string name;
            string lot;
        }
        class Department {
            string did;
            string dname;
            number budget;
        }
        Employee }o--o Department : "Works_In"
    
```

Relationship: Association among two or more entities.
E.g., Attishoo works in Pharmacy department.

Relationship Set: Collection of similar relationships.

- An n-ary relationship set R relates n entity sets E₁ ... E_n; each relationship in R involves entities e₁ ∈ E₁, ..., e_n ∈ E_n
- Same entity set could participate in different relationship sets, or in different “roles” in same set.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

4

Key Constraints

```

    erDiagram
        class Employee {
            string ssn;
            string name;
            string lot;
        }
        class Department {
            string did;
            string dname;
            number budget;
        }
        Employee }o--o Department : "Manages"
    
```

Consider Works_In:
An employee can work in many departments; a dept can have many employees.

In contrast, each dept has at most one manager, according to the key constraint on Manages.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

5

Participation Constraints

```

    erDiagram
        class Employee {
            string ssn;
            string name;
            string lot;
        }
        class Department {
            string did;
            string dname;
            number budget;
        }
        Employee }o--o Department : "Manages"
        Employee }o--o since : "Works_In"
    
```

Does every department have a manager?

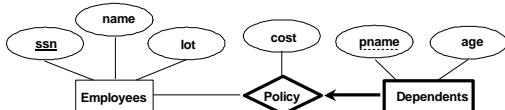
- If so, this is a participation constraint: the participation of Departments in Manages is said to be *total* (vs. *partial*).
 - Every did value in Departments table must appear in a row of the Manages table (with a non-null ssn value!)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

6

Weak Entities

- ❖ A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
 - Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
 - Weak entity set must have total participation in this *identifying* relationship set.



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

7

ISA ('is a') Hierarchies

- As in C++, or other PLs, attributes are inherited.
 - If we declare A ISA B, every A entity is also considered to be a B entity.
 - Overlap constraints:** Can Joe be an Hourly_Emps as well as a Contract_Emps entity? (*Allowed/disallowed*)
 - Covering constraints:** Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? (*Yes/no*)
 - Reasons for using ISA:
 - To add descriptive attributes specific to a subclass.
 - To identify entities that participate in a relationship.

Database Management Systems 3ed | R. Ramakrishnan and J. Gehrke

8

Aggregation

- Used when we have to model a relationship involving (entity sets and) a **relationship set**.
 - Aggregation allows us to treat a relationship set as an entity set for purposes of participation in (other) relationships.

```

classDiagram
    class Projects {
        <<pid>>
        <<pbudget>>
    }
    class Sponsors {
        <<since>>
        <<did>>
    }
    class Departments {
        <<dbname>>
        <<budget>>
    }

    Projects "1..*" o-- "0..1" Sponsors : pid
    Projects "*" --> "1..*" Departments : pbudget

    class Monitors
    class until
  
```

* *Aggregation vs. ternary relationship:*

 - v Monitors is a distinct relationship, with a descriptive attribute.
 - v Also, can say that each sponsorship is monitored by at most one employee.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

9

Conceptual Design Using the ER Model

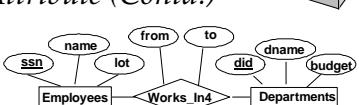
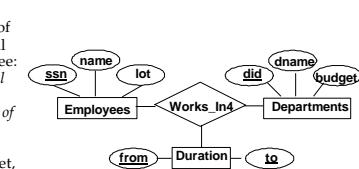
❖ Design choices:

- Should a concept be modeled as an entity or an attribute?
 - Should a concept be modeled as an entity or a relationship?
 - Identifying relationships: Binary or ternary? Aggregation?
- ❖ Constraints in the ER Model:
- A lot of data semantics can (and should) be captured.
 - But some constraints cannot be captured in ER diagrams.

Entity vs. Attribute

- ❖ Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
- ❖ Depends upon the use we want to make of address information, and the semantics of the data:
 - If we have several addresses per employee, *address* must be an entity (since attributes cannot be set-valued).
 - If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modeled as an entity (since attribute values are atomic).

Entity vs. Attribute (Contd.)

- ❖ Works_In4 does not allow an employee to work in a department for two or more periods.
- ❖ Similar to the problem of wanting to record several addresses for an employee: We want to record *several values of the descriptive attributes for each instance of this relationship*. Accomplished by introducing new entity set, Duration.

Entity vs. Relationship

- First ER diagram OK if a manager gets a separate discretionary budget for each dept.
 - What if a manager gets a discretionary budget that covers *all* managed depts?
 - Redundancy: *dbudget* stored for each dept managed by manager.
 - Misleading: Suggests *dbudget* associated with department-mgr combination.
-

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

13

Binary vs. Ternary Relationships

- If each policy is owned by just 1 employee, and each dependent is tied to the covering policy, first diagram is inaccurate.
 - What are the additional constraints in the 2nd diagram?
-

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

14

Binary vs. Ternary Relationships (Contd.)

- Previous example illustrated a case when two binary relationships were better than one ternary relationship.
- An example in the other direction: a ternary relation Contracts relates entity sets Parts, Departments and Suppliers, and has descriptive attribute *qty*. No combination of binary relationships is an adequate substitute:
 - S "can-supply" P, D "needs" P, and D "deals-with" S does not imply that D has agreed to buy P from S.
 - How do we record *qty*?

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

15

Summary of Conceptual Design

- ❖ *Conceptual design follows requirements analysis,*
 - Yields a high-level description of data to be stored
- ❖ ER model popular for conceptual design
 - Constructs are expressive, close to the way people think about their applications.
- ❖ Basic constructs: *entities, relationships, and attributes* (of entities and relationships).
- ❖ Some additional constructs: *weak entities, ISA hierarchies, and aggregation.*
- ❖ Note: There are many variations on ER model.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

16

Summary of ER (Contd.)

- ❖ Several kinds of integrity constraints can be expressed in the ER model: *key constraints, participation constraints, and overlap/covering constraints* for ISA hierarchies. Some *foreign key constraints* are also implicit in the definition of a relationship set.
 - Some constraints (notably, *functional dependencies*) cannot be expressed in the ER model.
 - Constraints play an important role in determining the best database design for an enterprise.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

17

Summary of ER (Contd.)

- ❖ ER design is *subjective*. There are often many ways to model a given scenario! Analyzing alternatives can be tricky, especially for a large enterprise. Common choices include:
 - Entity vs. attribute, entity vs. relationship, binary or n-ary relationship, whether or not to use ISA hierarchies, and whether or not to use aggregation.
- ❖ Ensuring good database design: resulting relational schema should be analyzed and refined further. FD information and normalization techniques are especially useful.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

18



The Relational Model

Chapter 3

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

1



Why Study the Relational Model?

- ❖ Most widely used model.
 - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- ❖ “Legacy systems” in older models
 - E.G., IBM’s IMS
- ❖ Recent competitor: object-oriented model
 - ObjectStore, Versant, Ontos
 - A synthesis emerging: *object-relational model*
 - Informix Universal Server, UniSQL, O2, Oracle, DB2

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

2



Relational Database: Definitions

- ❖ *Relational database*: a set of *relations*
- ❖ *Relation*: made up of 2 parts:
 - *Instance* : a *table*, with rows and columns.
#Rows = *cardinality*, #fields = *degree / arity*.
 - *Schema* : specifies name of relation, plus name and type of each column.
 - E.G. Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real).
- ❖ Can think of a relation as a *set* of rows or *tuples* (i.e., all rows are distinct).

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

3

Example Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

- ❖ Cardinality = 3, degree = 5, all rows distinct
- ❖ Do all columns in a relation instance have to be distinct?

Relational Query Languages

- ❖ A major strength of the relational model: supports simple, powerful *querying* of data.
- ❖ Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
 - The key: precise semantics for relational queries.
 - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

The SQL Query Language

- ❖ Developed by IBM (system R) in the 1970s
- ❖ Need for a standard since it is used by many vendors
- ❖ Standards:
 - SQL-86
 - SQL-89 (minor revision)
 - SQL-92 (major revision)
 - SQL-99 (major extensions, current standard)

The SQL Query Language

- ❖ To find all 18 year old students, we can write:

```
SELECT *      | sid   name    login   age  gpa |
FROM Students S | 53666 Jones   jones@cs  18   3.4 |
WHERE S.age=18  | 53688 Smith   smith@ee  18   3.2 |
```

- To find just names and logins, replace the first line:

```
SELECT S.name, S.login
```

Querying Multiple Relations

- ❖ What does the following query compute?

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade="A"
```

Given the following instance of Enrolled (is this possible if the DBMS ensures referential integrity?):

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

we get:

S.name	E.cid
Smith	Topology112

Creating Relations in SQL

- ❖ Creates the Students relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Students
(sid: CHAR(20),
name: CHAR(20),
login: CHAR(10),
age: INTEGER,
gpa: REAL)
```

- ❖ As another example, the Enrolled table holds information about courses that students take.

```
CREATE TABLE Enrolled
(sid: CHAR(20),
cid: CHAR(20),
grade: CHAR(2))
```

Destroying and Altering Relations

DROP TABLE Students

- ❖ Destroys the relation Students. The schema information *and* the tuples are deleted.

ALTER TABLE Students
ADD COLUMN firstYear: integer

- ❖ The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.

Adding and Deleting Tuples

- ❖ Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)
```

- ❖ Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = 'Smith'
```

* Powerful variants of these commands are available; more later!

Integrity Constraints (ICs)

- ❖ IC: condition that must be true for *any* instance of the database; e.g., domain constraints.
 - ICs are specified when schema is defined.
 - ICs are checked when relations are modified.
- ❖ A *legal* instance of a relation is one that satisfies all specified ICs.
 - DBMS should not allow illegal instances.
- ❖ If the DBMS checks ICs, stored data is more faithful to real-world meaning.
 - Avoids data entry errors, too!

Primary Key Constraints

- ❖ A set of fields is a *key* for a relation if :
 1. No two distinct tuples can have same values in all key fields, and
 2. This is not true for any subset of the key.
 - Part 2 false? A *superkey*.
 - If there's >1 key for a relation, one of the keys is chosen (by DBA) to be the *primary key*.
- ❖ E.g., *sid* is a key for Students. (What about *name*?) The set {*sid*, *gpa*} is a superkey.

Primary and Candidate Keys in SQL

- ❖ Possibly many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.
- ❖ "For a given student and course, there is a single grade." vs. "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."
- ❖ Used carelessly, an IC can prevent the storage of database instances that arise in practice!

```
CREATE TABLE Enrolled  
(sid CHAR(20)  
cid CHAR(20),  
grade CHAR(2),  
PRIMARY KEY (sid,cid) )  
  
CREATE TABLE Enrolled  
(sid CHAR(20)  
cid CHAR(20),  
grade CHAR(2),  
PRIMARY KEY (sid),  
UNIQUE (cid, grade) )
```

Foreign Keys, Referential Integrity

- ❖ *Foreign key* : Set of fields in one relation that is used to 'refer' to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a 'logical pointer'.
- ❖ E.g. *sid* is a foreign key referring to Students:
 - Enrolled(*sid*: string, *cid*: string, *grade*: string)
 - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.
 - Can you name a data model w/o referential integrity?
 - Links in HTML!

Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses.

```
CREATE TABLE Enrolled
  (sid CHAR(20), cid CHAR(20), grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students)
```

Enrolled			Students			
sid	cid	grade	sid	name	login	age
53666	Carnatic101	C	53666	Jones	jones@cs	18
53666	Reggae203	B	53688	Smith	smith@eecs	18
53650	Topology112	A	53650	Smith	smith@math	19
53666	History105	B				3.2
						3.8

Enforcing Referential Integrity

- Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)
- What should be done if a Students tuple is deleted?
 - Also delete all Enrolled tuples that refer to it.
 - Disallow deletion of a Students tuple that is referred to.
 - Set *sid* in Enrolled tuples that refer to it to a *default sid*.
 - (In SQL, also: Set *sid* in Enrolled tuples that refer to it to a special value *null*, denoting ‘unknown’ or ‘inapplicable’.)
- Similar if primary key of Students tuple is updated.

Referential Integrity in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and updates.
 - Default is NO ACTION (*delete/update is rejected*)
 - CASCADE (also delete all tuples that refer to deleted tuple)
 - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)
- ```
CREATE TABLE Enrolled
 (sid CHAR(20),
 cid CHAR(20),
 grade CHAR(2),
 PRIMARY KEY (sid,cid),
 FOREIGN KEY (sid)
 REFERENCES Students
 ON DELETE CASCADE
 ON UPDATE SET DEFAULT)
```

## Where do ICs Come From?

- ❖ ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
- ❖ We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about *all possible* instances!
  - For example, we know *name* is not a key, but the assertion that *sid* is a key is given to us.
- ❖ Key and foreign key ICs are the most common; more general ICs supported too.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

19

---

---

---

---

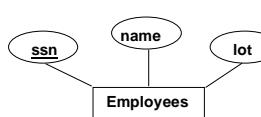
---

---

---

## Logical DB Design: ER to Relational

- ❖ Entity sets to tables:



```
CREATE TABLE Employees
(ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

20

---

---

---

---

---

---

---

## Relationship Sets to Tables

- ❖ In translating a relationship set to a relation, attributes of the relation must include:
  - Keys for each participating entity set (as foreign keys).
    - This set of attributes forms a *superkey* for the relation.
  - All descriptive attributes.

```
CREATE TABLE Works_In(
ssn CHAR(1),
did INTEGER,
since DATE,
PRIMARY KEY (ssn, did),
FOREIGN KEY (ssn)
REFERENCES Employees,
FOREIGN KEY (did)
REFERENCES Departments)
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

21

---

---

---

---

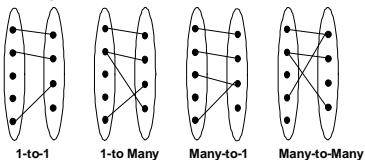
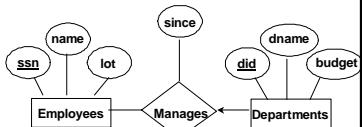
---

---

---

## *Review: Key Constraints*

- ❖ Each dept has at most one manager according to the key constraint on Manages.



## *Translation to relational model?*

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

22

Translating ER Diagrams with Key Constraints

- ❖ Map relationship to a table:
    - Note that did is the key now!
    - Separate tables for Employees and Departments.
  - ❖ Since each department has a unique manager, we could instead combine Manages and Departments.

```
CREATE TABLE Manages(
 ssn CHAR(11),
 did INTEGER,
 since DATE,
 PRIMARY KEY (did),
 FOREIGN KEY (ssn) REFERENCES Employees,
 FOREIGN KEY (did) REFERENCES Departments)
```

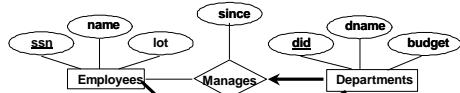
```
CREATE TABLE Dept_Mgr(
 did INTEGER,
 dname CHAR(20),
 budget REAL,
 ssn CHAR(11),
 since DATE,
 PRIMARY KEY (did),
 FOREIGN KEY (ssn) REFERENCES Employees(ssn))
```

Database Management Systems 3ed., R. Ramakrishnan and J. Gehrke

23

## *Review: Participation Constraints*

- ❖ Does every department have a manager?
    - If so, this is a participation constraint: the participation of Departments in Manages is said to be *total* (vs. *partial*).
      - Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!)



Database Management Systems 3ed., R. Ramakrishnan and J.Gehrke

24

## Participation Constraints in SQL

- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints).

```
CREATE TABLE Dept_Mgr(
 did INTEGER,
 dname CHAR(20),
 budget REAL,
 ssn CHAR(11) NOT NULL,
 since DATE,
 PRIMARY KEY (did),
 FOREIGN KEY (ssn) REFERENCES Employees,
 ON DELETE NO ACTION)
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

25

---

---

---

---

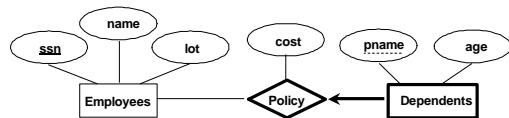
---

---

---

## Review: Weak Entities

- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

26

---

---

---

---

---

---

---

## Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - When the owner entity is deleted, all owned weak entities must also be deleted.

```
CREATE TABLE Dep_Policy (
 pname CHAR(20),
 age INTEGER,
 cost REAL,
 ssn CHAR(11) NOT NULL,
 PRIMARY KEY (pname, ssn),
 FOREIGN KEY (ssn) REFERENCES Employees,
 ON DELETE CASCADE)
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

27

---

---

---

---

---

---

---

## Review: ISA Hierarchies

- ❖ As in C++, or other PLs, attributes are inherited.
- ❖ If we declare A ISA B, every A entity is also considered to be a B entity.
- ❖ *Overlap constraints:* Can Joe be an Hourly\_Emps as well as a Contract\_Emps entity? (*Allowed/disallowed*)
- ❖ *Covering constraints:* Does every Employees entity also have to be an Hourly\_Emps or a Contract\_Emps entity? (*Yes/no*)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

28

## Translating ISA Hierarchies to Relations

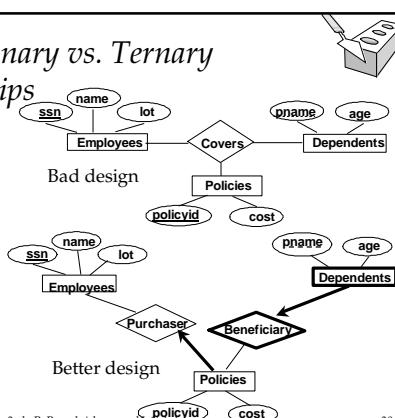
- ❖ **General approach:**
  - 3 relations: Employees, Hourly\_Emps and Contract\_Emps.
  - *Hourly\_Emps:* Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly\_Emps (*hourly\_wages, hours\_worked, ssn*); must delete Hourly\_Emps tuple if referenced Employees tuple is deleted).
  - Queries involving all employees easy, those involving just Hourly\_Emps require a join to get some attributes.
- ❖ Alternative: Just Hourly\_Emps and Contract\_Emps.
  - *Hourly\_Emps:* *ssn, name, lot, hourly\_wages, hours\_worked*.
  - Each employee must be in one of these two subclasses.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

29

## Review: Binary vs. Ternary Relationships

- ❖ What are the additional constraints in the 2nd diagram?



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

30

### *Binary vs. Ternary Relationships (Contd.)*

- ❖ The key constraints allow us to combine Purchaser with Policies and Beneficiary with Dependents.
  - ❖ Participation constraints lead to NOT NULL constraints.
  - ❖ What if Policies is a weak entity set?
- ```
CREATE TABLE Policies (
    policyid INTEGER,
    cost REAL,
    ssn CHAR(11) NOT NULL,
    PRIMARY KEY (policyid),
    FOREIGN KEY (ssn) REFERENCES Employees,
    ON DELETE CASCADE)
```
- ```
CREATE TABLE Dependents (
 pname CHAR(20),
 age INTEGER,
 policyid INTEGER,
 PRIMARY KEY (pname, policyid),
 FOREIGN KEY (policyid) REFERENCES Policies,
 ON DELETE CASCADE)
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

31

### *Views*

- ❖ A *view* is just a relation, but we store a *definition*, rather than a set of tuples.

```
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
```

- ❖ Views can be dropped using the DROP VIEW command.
  - How to handle DROP TABLE if there's a view on the table?
    - DROP TABLE command has options to let the user specify this.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

32

### *Views and Security*

- ❖ Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given YoungStudents, but not Students or Enrolled, we can find students s who have are enrolled, but not the *cids* of the courses they are enrolled in.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

33

## *Relational Model: Summary*



- ❖ A tabular representation of data.
- ❖ Simple and intuitive, currently the most widely used.
- ❖ Integrity constraints can be specified by the DBA, based on application semantics. DBMS checks for violations.
  - Two important ICs: primary and foreign keys
  - In addition, we *always* have domain constraints.
- ❖ Powerful and natural query languages exist.
- ❖ Rules to translate ER to relational model

---

---

---

---

---

---

---

---



## *Relational Algebra*

### Chapter 4, Part A

---

---

---

---

---

---



## *Relational Query Languages*

- ❖ Query languages: Allow manipulation and retrieval of data from a database.
- ❖ Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic.
  - Allows for much optimization.
- ❖ Query Languages  $\vdash$  programming languages!
  - QLs not expected to be “Turing complete”.
  - QLs not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

---

---

---

---

---

---



## *Formal Relational Query Languages*

- ❖ Two mathematical Query Languages form the basis for “real” languages (e.g. SQL), and for implementation:
  - Relational Algebra: More operational, very useful for representing execution plans.
  - Relational Calculus: Lets users describe what they want, rather than how to compute it. (Non-operational, declarative.)

---

---

---

---

---

---

## Preliminaries

- ❖ A query is applied to *relation instances*, and the result of a query is also a relation instance.
  - *Schemas* of input relations for a query are fixed (but query will run regardless of instance!)
  - The schema for the *result* of a given query is also fixed! Determined by definition of query language constructs.
- ❖ Positional vs. named-field notation:
  - Positional notation easier for formal definitions, named-field notation more readable.
  - Both used in SQL

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

4

## Example Instances

| R1 | sid | bid      | day |
|----|-----|----------|-----|
| 22 | 101 | 10/10/96 |     |
| 58 | 103 | 11/12/96 |     |

- ❖ "Sailors" and "Reserves" relations for our examples.
  - ❖ We'll use positional or named field notation, assume that names of fields in query results are 'inherited' from names of fields in query input relations.
- | S1 | sid    | sname | rating | age |
|----|--------|-------|--------|-----|
| 22 | dustin | 7     | 45.0   |     |
| 31 | lubber | 8     | 55.5   |     |
| 58 | rusty  | 10    | 35.0   |     |
| S2 | sid    | sname | rating | age |
| 28 | yuppy  | 9     | 35.0   |     |
| 31 | lubber | 8     | 55.5   |     |
| 44 | guppy  | 5     | 35.0   |     |
| 58 | rusty  | 10    | 35.0   |     |

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

5

## Relational Algebra

- ❖ Basic operations:
  - Selection ( $\sigma$ ) Selects a subset of rows from relation.
  - Projection ( $\pi$ ) Deletes unwanted columns from relation.
  - Cross-product ( $\times$ ) Allows us to combine two relations.
  - Set-difference ( $-$ ) Tuples in reln. 1, but not in reln. 2.
  - Union ( $\sqcup$ ) Tuples in reln. 1 and in reln. 2.
- ❖ Additional operations:
  - Intersection, join, division, renaming: Not essential, but (very!) useful.
- ❖ Since each operation returns a relation, operations can be *composed!* (Algebra is "closed".)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

6

## Projection

- Deletes attributes that are not in *projection list*.
- Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate *duplicates!* (Why??)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

| sname  | rating |
|--------|--------|
| yuppy  | 9      |
| lubber | 8      |
| guppy  | 5      |
| rusty  | 10     |

$$\pi_{sname, rating}(S2)$$

| age  |
|------|
| 35.0 |
| 55.5 |

$$\pi_{age}(S2)$$

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

7

## Selection

- Selects rows that satisfy *selection condition*.
- No duplicates in result! (Why?)
- Schema* of result identical to schema of (only) input relation.
- Result* relation can be the *input* for another relational algebra operation! (*Operator composition*.)

| sid | sname | rating | age  |
|-----|-------|--------|------|
| 28  | yuppy | 9      | 35.0 |
| 58  | rusty | 10     | 35.0 |

$$\sigma_{rating > 8}(S2)$$

| sname | rating |
|-------|--------|
| yuppy | 9      |
| rusty | 10     |

$$\pi_{sname, rating}(\sigma_{rating > 8}(S2))$$

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

8

## Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be union-compatible:
  - Same number of fields.
  - 'Corresponding' fields have the same type.
- What is the *schema* of result?

| sid | sname  | rating | age  |
|-----|--------|--------|------|
| 22  | dustin | 7      | 45.0 |
| 31  | lubber | 8      | 55.5 |
| 58  | rusty  | 10     | 35.0 |
| 44  | guppy  | 5      | 35.0 |
| 28  | yuppy  | 9      | 35.0 |

$$S1 \cup S2$$

| sid | sname  | rating | age  |
|-----|--------|--------|------|
| 31  | lubber | 8      | 55.5 |

$$S1 \cap S2$$

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

9

## Cross-Product

- ♦ Each row of S1 is paired with each row of R1.
- ♦ Result schema has one field per field of S1 and R1, with field names ‘inherited’ if possible.

- **Conflict:** Both S1 and R1 have a field called *sid*.

| (sid) | sname  | rating | age  | (sid) | bid | day      |
|-------|--------|--------|------|-------|-----|----------|
| 22    | dustin | 7      | 45.0 | 22    | 101 | 10/10/96 |
| 22    | dustin | 7      | 45.0 | 58    | 103 | 11/12/96 |
| 31    | lubber | 8      | 55.5 | 22    | 101 | 10/10/96 |
| 31    | lubber | 8      | 55.5 | 58    | 103 | 11/12/96 |
| 58    | rusty  | 10     | 35.0 | 22    | 101 | 10/10/96 |
| 58    | rusty  | 10     | 35.0 | 58    | 103 | 11/12/96 |

- **Renaming operator:**  $\rho(C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

10

## Joins

- ♦ Condition Join:  $R \bowtie_c S = \sigma_c(R \times S)$

| (sid) | sname  | rating | age  | (sid) | bid | day      |
|-------|--------|--------|------|-------|-----|----------|
| 22    | dustin | 7      | 45.0 | 58    | 103 | 11/12/96 |
| 31    | lubber | 8      | 55.5 | 58    | 103 | 11/12/96 |

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

- ♦ Result schema same as that of cross-product.
- ♦ Fewer tuples than cross-product, might be able to compute more efficiently
- ♦ Sometimes called a *theta-join*.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

11

## Joins

- ♦ Equi-Join: A special case of condition join where the condition *c* contains only *equalities*.

| sid | sname  | rating | age  | bid | day      |
|-----|--------|--------|------|-----|----------|
| 22  | dustin | 7      | 45.0 | 101 | 10/10/96 |
| 58  | rusty  | 10     | 35.0 | 103 | 11/12/96 |

$$S1 \bowtie_{sid} R1$$

- ♦ Result schema similar to cross-product, but only one copy of fields for which equality is specified.
- ♦ Natural Join: Equijoin on *all* common fields.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

12

## *Division*

- ❖ Not supported as a primitive operator, but useful for expressing queries like:  

$$\text{Find sailors who have reserved } \underline{\text{all}} \text{ boats.}$$
  - ❖ Let  $A$  have 2 fields,  $x$  and  $y$ ;  $B$  have only field  $y$ :
    - $A/B = \{\langle x \rangle \mid \exists \langle x, y \rangle \in A \ \forall \langle y \rangle \in B\}$
    - i.e.,  $A/B$  contains all  $x$  tuples (sailors) such that for every  $y$  tuple (boat) in  $B$ , there is an  $xy$  tuple in  $A$ .
    - Or: If the set of  $y$  values (boats) associated with an  $x$  value (sailor) in  $A$  contains all  $y$  values in  $B$ , the  $x$  value is in  $A/B$ .
  - ❖ In general,  $x$  and  $y$  can be any lists of fields;  $y$  is the list of fields in  $B$ , and  $x \cup y$  is the list of fields of  $A$ .

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

13



### *Examples of Division A/B*

| sno | pno |
|-----|-----|
| s1  | p1  |
| s1  | p2  |
| s1  | p3  |
| s1  | p4  |
| s2  | p1  |
| s2  | p2  |
| s3  | p2  |
| s4  | p2  |
| s4  | p4  |

A

|     |
|-----|
| pno |
| p2  |
| B1  |

|     |
|-----|
| sno |
| s1  |
| s2  |
| s3  |
| s4  |

A/B/T

|     |
|-----|
| pno |
| p2  |
| p4  |
| B2  |

|     |
|-----|
| sno |
| s1  |
| s4  |

A/B2

pno  
p1  
p2  
p4

A/B3

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

14



## *Expressing A/B Using Basic Operators*

- ❖ Division is not essential op; just a useful shorthand.
    - (Also true of joins, but joins are so common that systems implement joins specially.)
  - ❖ Idea: For  $A/B$ , compute all  $x$  values that are not ‘disqualified’ by some  $y$  value in  $B$ .
    - $x$  value is *disqualified* if by attaching  $y$  value from  $B$ , we obtain an  $xy$  tuple that is not in  $A$ .

Disqualified  $x$  values:  $\pi_x((\pi_x(A) \times B) - A)$

$A/B: \quad \pi_x(A) - \text{all disqualified tuples}$

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

15



*Find names of sailors who've reserved boat #103*

- ❖ Solution 1:  $\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \sqcup \text{Sailors})$
- ❖ Solution 2:  $\rho(\text{Temp1}, \sigma_{bid=103} \text{Reserves})$   
 $\rho(\text{Temp2}, \text{Temp1} \sqcup \text{Sailors})$   
 $\pi_{sname}(\text{Temp2})$
- ❖ Solution 3:  $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \sqcup \text{Sailors}))$

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

16

---

---

---

---

---

---

---

*Find names of sailors who've reserved a red boat*

- ❖ Information about boat color only available in Boats; so need an extra join:  
 $\pi_{sname}((\sigma_{color='red'} \text{Boats}) \sqcup \text{Reserves} \sqcup \text{Sailors})$
- ❖ A more efficient solution:  
 $\pi_{sname}(\pi_{sid}((\pi_{bid} \sigma_{color='red'} \text{Boats}) \sqcup \text{Reserves} \sqcup \text{Sailors}))$

*A query optimizer can find this, given the first solution!*

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

17

---

---

---

---

---

---

---

*Find sailors who've reserved a red or a green boat*

- ❖ Can identify all red or green boats, then find sailors who've reserved one of these boats:  
 $\rho(\text{Tempboats}, (\sigma_{color='red'} \vee color='green') \text{Boats})$   
 $\pi_{sname}(\text{Tempboats} \sqcup \text{Reserves} \sqcup \text{Sailors})$
- ❖ Can also define Tempboats using union! (How?)
- ❖ What happens if  $\vee$  is replaced by  $\wedge$  in this query?

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

18

---

---

---

---

---

---

---

*Find sailors who've reserved a red and a green boat*

- ❖ Previous approach won't work! Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that  $sid$  is a key for Sailors):

$$\begin{aligned}\rho(\text{Tempred}, \pi_{sid}((\sigma_{color='red'} \text{Boats}) \bowtie \text{Reserves})) \\ \rho(\text{Tempgreen}, \pi_{sid}((\sigma_{color='green'} \text{Boats}) \bowtie \text{Reserves})) \\ \pi_{sname}((\text{Tempred} \cap \text{Tempgreen}) \bowtie \text{Sailors})\end{aligned}$$

---

---

---

---

---

---

---

*Find the names of sailors who've reserved all boats*

- ❖ Uses division; schemas of the input relations to / must be carefully chosen:
- $$\begin{aligned}\rho(\text{Tempsids}, (\pi_{sid,bid} \text{Reserves}) / (\pi_{bid} \text{Boats})) \\ \pi_{sname}(\text{Tempsids} \bowtie \text{Sailors})\end{aligned}$$
- ❖ To find sailors who've reserved all 'Interlake' boats:  
..... /  $\pi_{bid}((\sigma_{bname='Interlake'} \text{Boats})$

---

---

---

---

---

---

---

## *Summary*

- ❖ The relational model has rigorously defined query languages that are simple and powerful.
- ❖ Relational algebra is more operational; useful as internal representation for query evaluation plans.
- ❖ Several ways of expressing a given query; a query optimizer should choose the most efficient version.

---

---

---

---

---

---

---



## *Relational Calculus*

### Chapter 4, Part B

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

1

---

---

---

---

---

---

---



## *Relational Calculus*

- ❖ Comes in two flavors: *Tuple relational calculus* (TRC) and *Domain relational calculus* (DRC).
- ❖ Calculus has *variables, constants, comparison ops, logical connectives and quantifiers*.
  - TRC: Variables range over (i.e., get bound to) *tuples*.
  - DRC: Variables range over *domain elements* (= field values).
  - Both TRC and DRC are simple subsets of first-order logic.
- ❖ Expressions in the calculus are called *formulas*. An answer tuple is essentially an assignment of constants to variables that make the formula evaluate to *true*.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

2

---

---

---

---

---

---

---



## *Domain Relational Calculus*

- ❖ *Query* has the form:  
 $\langle x_1, x_2, \dots, x_n \rangle | p(\langle x_1, x_2, \dots, x_n \rangle)$
- ❖ *Answer* includes all tuples  $\langle x_1, x_2, \dots, x_n \rangle$  that make the formula  $p(\langle x_1, x_2, \dots, x_n \rangle)$  be *true*.
- ❖ *Formula* is recursively defined, starting with simple *atomic formulas* (getting tuples from relations or making comparisons of values), and building bigger and better formulas using the *logical connectives*.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

3

---

---

---

---

---

---

---

## DRC Formulas

❖ *Atomic formula:*

- $\langle x_1, x_2, \dots, x_n \rangle \in Rname$ , or  $X op Y$ , or  $X op$  constant
- $op$  is one of  $<, >, =, \leq, \geq, \neq$

❖ *Formula:*

- an atomic formula, or
- $\neg p, p \wedge q, p \vee q$ , where  $p$  and  $q$  are formulas, or
- $\exists X(p(X))$ , where variable  $X$  is *free* in  $p(X)$ , or
- $\forall X(p(X))$ , where variable  $X$  is *free* in  $p(X)$

❖ The use of quantifiers  $\exists X$  and  $\forall X$  is said to *bind*  $X$ .

- A variable that is not bound is free.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

4

---

---

---

---

---

---

---

---

## Free and Bound Variables

❖ The use of quantifiers  $\exists X$  and  $\forall X$  in a formula is said to *bind*  $X$ .

- A variable that is not bound is *free*.

❖ Let us revisit the definition of a query:

$$\{ \langle x_1, x_2, \dots, x_n \rangle \mid p(\langle x_1, x_2, \dots, x_n \rangle) \}$$

❖ There is an important restriction: the variables  $x_1, \dots, x_n$  that appear to the left of ' $|$ ' must be the *only* free variables in the formula  $p(\dots)$ .

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

5

---

---

---

---

---

---

---

---

## Find all sailors with a rating above 7

$$\{ \langle I, N, T, A \rangle \mid \langle I, N, T, A \rangle \in Sailors \wedge T > 7 \}$$

❖ The condition  $\langle I, N, T, A \rangle \in Sailors$  ensures that the domain variables  $I, N, T$  and  $A$  are bound to fields of the same Sailors tuple.

❖ The term  $\langle I, N, T, A \rangle$  to the left of ' $|$ ' (which should be read as *such that*) says that every tuple  $\langle I, N, T, A \rangle$  that satisfies  $T > 7$  is in the answer.

❖ Modify this query to answer:

- Find sailors who are older than 18 or have a rating under 9, and are called 'Joe'.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

6

---

---

---

---

---

---

---

---

*Find sailors rated > 7 who've reserved boat #103*

$$\langle I, N, T, A \rangle | \langle I, N, T, A \rangle \in \text{Sailors} \wedge T > 7 \wedge \exists Ir, Br, D (\langle Ir, Br, D \rangle \in \text{Reserves} \wedge Ir = I \wedge Br = 103)$$

- ❖ We have used  $\exists Ir, Br, D (\dots)$  as a shorthand for  $\exists Ir (\exists Br (\exists D(\dots)))$
- ❖ Note the use of  $\exists$  to find a tuple in Reserves that ‘joins with’ the Sailors tuple under consideration.

---

---

---

---

---

---

---

*Find sailors rated > 7 who've reserved a red boat*

$$\langle I, N, T, A \rangle | \langle I, N, T, A \rangle \in \text{Sailors} \wedge T > 7 \wedge \exists Ir, Br, D (\langle Ir, Br, D \rangle \in \text{Reserves} \wedge Ir = I \wedge \exists B, BN, C (\langle B, BN, C \rangle \in \text{Boats} \wedge B = Br \wedge C = 'red'))$$

- ❖ Observe how the parentheses control the scope of each quantifier’s binding.
- ❖ This may look cumbersome, but with a good user interface, it is very intuitive. (MS Access, QBE)

---

---

---

---

---

---

---

*Find sailors who've reserved all boats*

$$\langle I, N, T, A \rangle | \langle I, N, T, A \rangle \in \text{Sailors} \wedge \forall B, BN, C (\neg (\langle B, BN, C \rangle \in \text{Boats}) \vee (\exists Ir, Br, D (\langle Ir, Br, D \rangle \in \text{Reserves} \wedge I = Ir \wedge Br = B)))$$

- ❖ Find all sailors  $I$  such that for each 3-tuple  $\langle B, BN, C \rangle$  either it is not a tuple in Boats or there is a tuple in Reserves showing that sailor  $I$  has reserved it.

---

---

---

---

---

---

---

*Find sailors who've reserved all boats (again!)*

$$\left\{ \langle I, N, T, A \rangle \mid \begin{array}{l} \langle I, N, T, A \rangle \in Sailors \wedge \\ \forall \langle B, BN, C \rangle \in Boats \\ \exists \langle Ir, Br, D \rangle \in Reserves (I = Ir \wedge Br = B) \end{array} \right\}$$

- ❖ Simpler notation, same query. (Much clearer!)
- ❖ To find sailors who've reserved all red boats:  
.....  $\left[ C \neq 'red' \vee \exists \langle Ir, Br, D \rangle \in Reserves (I = Ir \wedge Br = B) \right]$

---

---

---

---

---

---

---

---

---

### *Unsafe Queries, Expressive Power*

- ❖ It is possible to write syntactically correct calculus queries that have an infinite number of answers! Such queries are called unsafe.
  - e.g.,  $\{S \mid \neg(S \in Sailors)\}$
- ❖ It is known that every query that can be expressed in relational algebra can be expressed as a safe query in DRC / TRC; the converse is also true.
- ❖ Relational Completeness: Query language (e.g., SQL) can express every query that is expressible in relational algebra/calculus.

---

---

---

---

---

---

---

---

---

### *Summary*

- ❖ Relational calculus is non-operational, and users define queries in terms of what they want, not in terms of how to compute it. (Declarativeness.)
- ❖ Algebra and safe calculus have same expressive power, leading to the notion of relational completeness.

---

---

---

---

---

---

---

---

---



## SQL: Queries, Programming, Triggers

### Chapter 5

---

---

---

---

---

---

### Example Instances

| R1 | sid | bid | day      |
|----|-----|-----|----------|
|    | 22  | 101 | 10/10/96 |
|    | 58  | 103 | 11/12/96 |

- ❖ We will use these instances of the Sailors and Reserves relations in our examples.
- ❖ If the key for the Reserves relation contained only the attributes *sid* and *bid*, how would the semantics differ?

| S1 | sid | sname  | rating | age  |
|----|-----|--------|--------|------|
|    | 22  | dustin | 7      | 45.0 |
|    | 31  | lubber | 8      | 55.5 |
|    | 58  | rusty  | 10     | 35.0 |

| S2 | sid | sname  | rating | age  |
|----|-----|--------|--------|------|
|    | 28  | yuppy  | 9      | 35.0 |
|    | 31  | lubber | 8      | 55.5 |
|    | 44  | guppy  | 5      | 35.0 |
|    | 58  | rusty  | 10     | 35.0 |

---

---

---

---

---

---

### Basic SQL Query



```
SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification
```

- ❖ *relation-list* A list of relation names (possibly with a range-variable after each name).
- ❖ *target-list* A list of attributes of relations in *relation-list*
- ❖ *qualification* Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of *<*, *>*, *=*,  *$\leq$* ,  *$\geq$* ,  *$\neq$* ) combined using AND, OR and NOT.
- ❖ DISTINCT is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are *not* eliminated!

---

---

---

---

---

---

## Conceptual Evaluation Strategy

- ❖ Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*.
  - Discard resulting tuples if they fail *qualifications*.
  - Delete attributes that are not in *target-list*.
  - If DISTINCT is specified, eliminate duplicate rows.
- ❖ This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute *the same answers*.

---

---

---

---

---

---

---

---

## Example of Conceptual Evaluation

```
SELECT S.sname
 FROM Sailors S, Reserves R
 WHERE S.sid=R.sid AND R.bid=103
```

| (sid) | sname  | rating | age  | (sid) | bid | day      |
|-------|--------|--------|------|-------|-----|----------|
| 22    | dustin | 7      | 45.0 | 22    | 101 | 10/10/96 |
| 22    | dustin | 7      | 45.0 | 58    | 103 | 11/12/96 |
| 31    | lubber | 8      | 55.5 | 22    | 101 | 10/10/96 |
| 31    | lubber | 8      | 55.5 | 58    | 103 | 11/12/96 |
| 58    | rusty  | 10     | 35.0 | 22    | 101 | 10/10/96 |
| 58    | rusty  | 10     | 35.0 | 58    | 103 | 11/12/96 |

---

---

---

---

---

---

---

---

## A Note on Range Variables

- ❖ Really needed only if the same relation appears twice in the FROM clause. The previous query can also be written as:

```
SELECT S.sname
 FROM Sailors S, Reserves R
 WHERE S.sid=R.sid AND bid=103
OR SELECT sname
 FROM Sailors, Reserves
 WHERE Sailors.sid=Reserves.sid
 AND bid=103
```

*It is good style,  
however, to use  
range variables  
always!*

---

---

---

---

---

---

---

---

*Find sailors who've reserved at least one boat*

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

- ❖ Would adding DISTINCT to this query make a difference?
- ❖ What is the effect of replacing *S.sid* by *S.sname* in the SELECT clause? Would adding DISTINCT to this variant of the query make a difference?

---

---

---

---

---

---

*Expressions and Strings*

```
SELECT S.age, age1=S.age-5, 2*S.age AS age2
FROM Sailors S
WHERE S.sname LIKE 'B_%B'
```

- ❖ Illustrates use of arithmetic expressions and string pattern matching: *Find triples (of ages of sailors and two fields defined by expressions) for sailors whose names begin and end with B and contain at least three characters.*
- ❖ AS and = are two ways to name fields in result.
- ❖ LIKE is used for string matching. `\_` stands for any one character and `%` stands for 0 or more arbitrary characters.

---

---

---

---

---

---

*Find sid's of sailors who've reserved a red or a green boat*

- ❖ UNION: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).
- ❖ If we replace OR by AND in the first version, what do we get?
- ❖ Also available: EXCEPT (What do we get if we replace UNION by EXCEPT?)

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
AND (B.color='red' OR B.color='green')
```

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
AND B.color='red'
UNION
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
AND B.color='green'
```

---

---

---

---

---

---

*Find sid's of sailors who've reserved a red and a green boat*

```
SELECT S.sid
FROM Sailors S, Boats B1, Reserves R1,
 Boats B2, Reserves R2
WHERE S.sid=R1.sid AND R1.bid=B1.bid
 AND S.sid=R2.sid AND R2.bid=B2.bid
 AND (B1.color='red' AND B2.color='green')
```

- ❖ INTERSECT: Can be used to compute the intersection of any two *union-compatible* sets of tuples.
- ❖ Included in the SQL/92 standard, but some systems don't support it.
- ❖ Contrast symmetry of the UNION and INTERSECT queries with how much the other versions differ.

```
SELECT S.sid Key field!
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
 AND B.color='red'
INTERSECT
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
 AND B.color='green'
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

10

## Nested Queries

*Find names of sailors who've reserved boat #103:*

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
 FROM Reserves R
 WHERE R.bid=103)
```

- ❖ A very powerful feature of SQL: a WHERE clause can itself contain an SQL query! (Actually, so can FROM and HAVING clauses.)
- ❖ To find sailors who've *not* reserved #103, use NOT IN.
- ❖ To understand semantics of nested queries, think of a *nested loops* evaluation: *For each Sailors tuple, check the qualification by computing the subquery.*

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

11

## Nested Queries with Correlation

*Find names of sailors who've reserved boat #103:*

```
SELECT S.sname
FROM Sailors S
WHERE EXISTS (SELECT *
 FROM Reserves R
 WHERE R.bid=103 AND S.sid=R.sid)
```

- ❖ EXISTS is another set comparison operator, like IN.
- ❖ If UNIQUE is used, and \* is replaced by R.bid, finds sailors with at most one reservation for boat #103. (UNIQUE checks for duplicate tuples; \* denotes all attributes. Why do we have to replace \* by R.bid?)
- ❖ Illustrates why, in general, subquery must be recomputed for each Sailors tuple.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

12

## More on Set-Comparison Operators

- ❖ We've already seen IN, EXISTS and UNIQUE. Can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- ❖ Also available:  $op$  ANY,  $op$  ALL,  $op$  IN  $>,<,>=,>=,$
- ❖ Find sailors whose rating is greater than that of some sailor called Horatio:

```
SELECT *
 FROM Sailors S
 WHERE S.rating > ANY (SELECT S2.rating
 FROM Sailors S2
 WHERE S2.sname='Horatio')
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

13

---

---

---

---

---

---

---

---

## Rewriting INTERSECT Queries Using IN

Find sid's of sailors who've reserved both a red and a green boat:

```
SELECT S.sid
 FROM Sailors S, Boats B, Reserves R
 WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red'
 AND S.sid IN (SELECT S2.sid
 FROM Sailors S2, Boats B2, Reserves R2
 WHERE S2.sid=R2.sid AND R2.bid=B2.bid
 AND B2.color='green')
```

- ❖ Similarly, EXCEPT queries re-written using NOT IN.
- ❖ To find *names* (not sid's) of Sailors who've reserved both red and green boats, just replace *S.sid* by *S.sname* in SELECT clause. (What about INTERSECT query?)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

14

---

---

---

---

---

---

---

---

## Division in SQL

Find sailors who've reserved all boats.

- ❖ Let's do it the hard way, without EXCEPT:

(1) 

```
SELECT S.sname
 FROM Sailors S
 WHERE NOT EXISTS
 ((SELECT B.bid
 FROM Boats B)
 EXCEPT
 (SELECT R.bid
 FROM Reserves R
 WHERE R.sid=S.sid))
```

(2) 

```
SELECT S.sname
 FROM Sailors S
 WHERE NOT EXISTS (SELECT B.bid
 FROM Boats B
 WHERE NOT EXISTS (SELECT R.bid
 FROM Reserves R
 WHERE R.bid=B.bid
 AND R.sid=S.sid))
Sailors S such that ...
there is no boat B without ...
a Reserves tuple showing S reserved B
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

15

---

---

---

---

---

---

---

---

## Aggregate Operators

- Significant extension of relational algebra.

COUNT (\*)  
COUNT ([DISTINCT] A)  
SUM ([DISTINCT] A)  
AVG ([DISTINCT] A)  
MAX (A)  
MIN (A)

*single column*

```
SELECT COUNT (*) SELECT S.sname
FROM Sailors S FROM Sailors S
SELECT AVG (S.age) WHERE S.rating= (SELECT MAX(S2.rating)
FROM Sailors S FROM Sailors S2)
WHERE S.rating=10

SELECT COUNT (DISTINCT S.rating) SELECT AVG (DISTINCT S.age)
FROM Sailors S FROM Sailors S
WHERE S.sname='Bob' WHERE S.rating=10
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

16

## Find name and age of the oldest sailor(s)

- The first query is illegal! (We'll look into the reason a bit later, when we discuss GROUP BY.)
- The third query is equivalent to the second query, and is allowed in the SQL/92 standard, but is not supported in some systems.

```
SELECT S.sname, MAX (S.age)
FROM Sailors S

SELECT S.sname, S.age
FROM Sailors S
WHERE S.age =
 (SELECT MAX (S2.age)
 FROM Sailors S2)

SELECT S.sname, S.age
FROM Sailors S
WHERE (SELECT MAX (S2.age)
 FROM Sailors S2)
 = S.age
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

17

## GROUP BY and HAVING

- So far, we've applied aggregate operators to all (qualifying) tuples. Sometimes, we want to apply them to each of several *groups* of tuples.
- Consider: *Find the age of the youngest sailor for each rating level.*
  - In general, we don't know how many rating levels exist, and what the rating values for these levels are!
  - Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this (!):

```
For i = 1, 2, ..., 10:
 SELECT MIN (S.age)
 FROM Sailors S
 WHERE S.rating = i
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

18

## Queries With GROUP BY and HAVING

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

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

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

19

## Conceptual Evaluation

- The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded, `unnecessary' fields are deleted, and the remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.
- The *group-qualification* is then applied to eliminate some groups. Expressions in *group-qualification* must have a single value per group!
  - In effect, an attribute in *group-qualification* that is not an argument of an aggregate op also appears in *grouping-list*. (SQL does not exploit primary key semantics here!)
- One answer tuple is generated per qualifying group.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

20

*Find the age of the youngest sailor with age  $\geq 18$  for each rating with at least 2 such sailors*

```
SELECT S.rating, MIN (S.age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1
```

- Only *S.rating* and *S.age* are mentioned in the SELECT, GROUP BY or HAVING clauses; other attributes `unnecessary'.
- 2nd column of result is unnamed. (Use AS to name it.)

| sid | sname   | rating | age  |
|-----|---------|--------|------|
| 22  | dustin  | 7      | 45.0 |
| 31  | lubber  | 8      | 55.5 |
| 71  | zorba   | 10     | 16.0 |
| 64  | horatio | 7      | 35.0 |
| 29  | brutus  | 1      | 33.0 |
| 58  | rusty   | 10     | 35.0 |

| rating | age  |
|--------|------|
| 1      | 33.0 |
| 7      | 45.0 |
| 7      | 35.0 |
| 8      | 55.5 |
| 10     | 35.0 |

Answer relation

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

21

*For each red boat, find the number of reservations for this boat*



```
SELECT B.bid, COUNT (*) AS count
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red'
GROUP BY B.bid
```

- ❖ Grouping over a join of three relations.
- ❖ What do we get if we remove `B.color='red'` from the WHERE clause and add a HAVING clause with this condition?
- ❖ What if we drop Sailors and the condition involving `S.sid`?

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

22

---

---

---

---

---

---

---

*Find the age of the youngest sailor with age > 18, for each rating with at least 2 sailors (of any age)*



```
SELECT S.rating, MIN (S.age)
FROM Sailors S
WHERE S.age > 18
GROUP BY S.rating
HAVING 1 < (SELECT COUNT (*)
 FROM Sailors S2
 WHERE S.rating=S2.rating)
```

- ❖ Shows HAVING clause can also contain a subquery.
- ❖ Compare this with the query where we considered only ratings with 2 sailors over 18!
- ❖ What if HAVING clause is replaced by:

- HAVING COUNT(\*) >1

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

23

---

---

---

---

---

---

---

*Find those ratings for which the average age is the minimum over all ratings*



- ❖ Aggregate operations cannot be nested! WRONG:

```
SELECT S.rating
FROM Sailors S
WHERE S.age = (SELECT MIN (AVG (S2.age)) FROM Sailors S2)
```

- ❖ Correct solution (in SQL/92):

```
SELECT Temp.rating, Temp.avgage
FROM (SELECT S.rating, AVG (S.age) AS avgage
 FROM Sailors S
 GROUP BY S.rating) AS Temp
WHERE Temp.avgage = (SELECT MIN (Temp.avgage)
 FROM Temp)
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

24

---

---

---

---

---

---

---

## Null Values

- ❖ Field values in a tuple are sometimes *unknown* (e.g., a rating has not been assigned) or *inapplicable* (e.g., no spouse's name).
  - SQL provides a special value *null* for such situations.
- ❖ The presence of *null* complicates many issues. E.g.:
  - Special operators needed to check if value is/is not *null*.
  - Is *rating>8* true or false when *rating* is equal to *null*? What about AND, OR and NOT connectives?
  - We need a 3-valued logic (true, false and *unknown*).
  - Meaning of constructs must be defined carefully. (e.g., WHERE clause eliminates rows that don't evaluate to true.)
  - New operators (in particular, *outer joins*) possible/needed.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

25

---

---

---

---

---

---

---

## Integrity Constraints (Review)

- ❖ An IC describes conditions that every *legal instance* of a relation must satisfy.
  - Inserts/deletes/updates that violate IC's are disallowed.
  - Can be used to ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)
- ❖ Types of IC's: Domain constraints, primary key constraints, foreign key constraints, general constraints.
  - *Domain constraints:* Field values must be of right type. Always enforced.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

26

---

---

---

---

---

---

---

## General Constraints

- ❖ Useful when more general ICs than keys are involved.
- ❖ Can use queries to express constraint.
- ❖ Constraints can be named.

```
CREATE TABLE Sailors
 (sid INTEGER,
 sname CHAR(10),
 rating INTEGER,
 age REAL,
 PRIMARY KEY (sid),
 CHECK (rating >= 1
 AND rating <= 10)

CREATE TABLE Reserves
 (sname CHAR(10),
 bid INTEGER,
 day DATE,
 PRIMARY KEY (bid,day),
 CONSTRAINT noInterlakeRes
 CHECK ('Interlake' <>
 (SELECT B.bname
 FROM Boats B
 WHERE B.bid=bid)))
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

27

---

---

---

---

---

---

---

## Constraints Over Multiple Relations

```
CREATE TABLE Sailors
```

( sid INTEGER,  
sname CHAR(10),  
rating INTEGER,  
age REAL,

*Number of boats  
plus number of  
sailors is < 100*

- ❖ Awkward and wrong!
  - ❖ If Sailors is empty, the number of Boats tuples can be anything!
  - ❖ ASSERTION is the right solution; not associated with either table.
- ```
PRIMARY KEY (sid),  
CHECK  
( (SELECT COUNT (S.sid) FROM Sailors S)  
+ (SELECT COUNT (B.bid) FROM Boats B) < 100  
CREATE ASSERTION smallClub  
CHECK  
( (SELECT COUNT (S.sid) FROM Sailors S)  
+ (SELECT COUNT (B.bid) FROM Boats B) < 100
```

Triggers

- ❖ Trigger: procedure that starts automatically if specified changes occur to the DBMS
- ❖ Three parts:
 - Event (activates the trigger)
 - Condition (tests whether the triggers should run)
 - Action (what happens if the trigger runs)

Triggers: Example (SQL:1999)

```
CREATE TRIGGER youngSailorUpdate  
AFTER INSERT ON SAILORS  
REFERENCING NEW TABLE NewSailors  
FOR EACH STATEMENT  
INSERT  
INTO YoungSailors(sid, name, age, rating)  
SELECT sid, name, age, rating  
FROM NewSailors N  
WHERE N.age <= 18
```

Summary

- ❖ SQL was an important factor in the early acceptance of the relational model; more natural than earlier, procedural query languages.
- ❖ Relationally complete; in fact, significantly more expressive power than relational algebra.
- ❖ Even queries that can be expressed in RA can often be expressed more naturally in SQL.
- ❖ Many alternative ways to write a query; optimizer should look for most efficient evaluation plan.
 - In practice, users need to be aware of how queries are optimized and evaluated for best results.



Summary (Contd.)

- ❖ NULL for unknown field values brings many complications
- ❖ SQL allows specification of rich integrity constraints
- ❖ Triggers respond to changes in the database



Database Application Development

Chapter 6

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

1



Overview

Concepts covered in this lecture:

- ❖ SQL in application code
- ❖ Embedded SQL
- ❖ Cursors
- ❖ Dynamic SQL
- ❖ JDBC
- ❖ SQLJ
- ❖ Stored procedures

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

2



SQL in Application Code

❖ SQL commands can be called from within a host language (e.g., C++ or Java) program.

- SQL statements can refer to **host variables** (including special variables used to return status).
- Must include a statement to **connect** to the right database.

❖ Two main integration approaches:

- Embed SQL in the host language (Embedded SQL, SQLJ)
- Create special API to call SQL commands (JDBC)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

3

SQL in Application Code (Contd.)



Impedance mismatch:

- ❖ SQL relations are (multi-) sets of records, with no *a priori* bound on the number of records.
No such data structure exist traditionally in procedural programming languages such as C++. (Though now: STL)
 - SQL supports a mechanism called a *cursor* to handle this.

Embedded SQL



- ❖ Approach: Embed SQL in the host language.
 - A preprocessor converts the SQL statements into special API calls.
 - Then a regular compiler is used to compile the code.
- ❖ Language constructs:
 - Connecting to a database:
EXEC SQL CONNECT
 - Declaring variables:
EXEC SQL BEGIN (END) DECLARE SECTION
 - Statements:
EXEC SQL Statement;

Embedded SQL: Variables



```
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20];
long c_sid;
short c_rating;
float c_age;
EXEC SQL END DECLARE SECTION
```

- ❖ Two special “error” variables:
 - SQLCODE (long, is negative if an error has occurred)
 - SQLSTATE (char[6], predefined codes for common errors)

Cursors



- ❖ Can declare a cursor on a relation or query statement (which generates a relation).
- ❖ Can *open* a cursor, and repeatedly *fetch* a tuple then *move* the cursor, until all tuples have been retrieved.
 - Can use a special clause, called **ORDER BY**, in queries that are accessed through a cursor, to control the order in which tuples are returned.
 - Fields in ORDER BY clause must also appear in SELECT clause.
 - The **ORDER BY** clause, which orders answer tuples, is *only* allowed in the context of a cursor.
- ❖ Can also modify/delete tuple pointed to by a cursor.

Cursor that gets names of sailors who've reserved a red boat, in alphabetical order



```
EXEC SQL DECLARE sinfo CURSOR FOR
  SELECT S.sname
    FROM Sailors S, Boats B, Reserves R
   WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red'
 ORDER BY S.sname
```

- ❖ Note that it is illegal to replace *S.sname* by, say, *S.sid* in the **ORDER BY** clause! (Why?)
- ❖ Can we add *S.sid* to the **SELECT** clause and replace *S.sname* by *S.sid* in the **ORDER BY** clause?

Embedding SQL in C: An Example



```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
  SELECT S.sname, S.age   FROM Sailors S
  WHERE S.rating > :c_minrating
  ORDER BY S.sname;
do {
  EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
  printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

Dynamic SQL



- ❖ SQL query strings are now always known at compile time (e.g., spreadsheet, graphical DBMS frontend): Allow construction of SQL statements on-the-fly

- ❖ Example:

```
char c_sqstring[]=  
    {"DELETE FROM Sailors WHERE rating>5"};  
EXEC SQL PREPARE readytogo FROM :c_sqstring;  
EXEC SQL EXECUTE readytogo;
```

Database APIs: Alternative to embedding



Rather than modify compiler, add library with database calls (API)

- ❖ Special standardized interface: procedures/objects
- ❖ Pass SQL strings from language, presents result sets in a language-friendly way
- ❖ Sun's JDBC: Java API
- ❖ Supposedly DBMS-neutral
 - a "driver" traps the calls and translates them into DBMS-specific code
 - database can be across a network

JDBC: Architecture



❖ Four architectural components:

- Application (initiates and terminates connections, submits SQL statements)
- Driver manager (load JDBC driver)
- Driver (connects to data source, transmits requests and returns/translates results and error codes)
- Data source (processes SQL statements)

JDBC Architecture (Contd.)



Four types of drivers:

Bridge:

- Translates SQL commands into non-native API.
Example: JDBC-ODBC bridge. Code for ODBC and JDBC driver needs to be available on each client.

Direct translation to native API, non-Java driver:

- Translates SQL commands to native API of data source.
Need OS-specific binary on each client.

Network bridge:

- Send commands over the network to a middleware server that talks to the data source. Needs only small JDBC driver at each client.

Direction translation to native API via Java driver:

- Converts JDBC calls directly to network protocol used by DBMS. Needs DBMS-specific Java driver at each client.

JDBC Classes and Interfaces



Steps to submit a database query:

- ❖ Load the JDBC driver
- ❖ Connect to the data source
- ❖ Execute SQL statements

JDBC Driver Management



- ❖ All drivers are managed by the DriverManager class
- ❖ Loading a JDBC driver:
 - In the Java code:
`Class.forName("oracle/jdbc.driver.OracleDriver");`
 - When starting the Java application:
`-Djdbc.drivers=oracle/jdbc.driver`

Connections in JDBC



We interact with a data source through sessions. Each connection identifies a logical session.

- ❖ JDBC URL:
`jdbc:<subprotocol>:<otherParameters>`

Example:

```
String url="jdbc:oracle:www.bookstore.com:3083";
Connection con;
try{
    con = DriverManager.getConnection(url,userId,password);
} catch SQLException expt { ...}
```

Connection Class Interface



- ❖ `public int getTransactionIsolation()` and
`void setTransactionIsolation(int level)`
Sets isolation level for the current connection.
- ❖ `public boolean getReadOnly()` and
`void setReadOnly(boolean b)`
Specifies whether transactions in this connection are read-only
- ❖ `public boolean getAutoCommit()` and
`void setAutoCommit(boolean b)`
If autocommit is set, then each SQL statement is considered its own transaction. Otherwise, a transaction is committed using `commit()`, or aborted using `rollback()`.
- ❖ `public boolean isClosed()`
Checks whether connection is still open.

Executing SQL Statements



- ❖ Three different ways of executing SQL statements:
 - Statement (both static and dynamic SQL statements)
 - PreparedStatement (semi-static SQL statements)
 - CallableStatement (stored procedures)
- ❖ **PreparedStatement class:**
Precompiled, parametrized SQL statements:
 - Structure is fixed
 - Values of parameters are determined at run-time

Executing SQL Statements (Contd.)

```
String sql="INSERT INTO Sailors VALUES(?, ?, ?, ?);  
PreparedStatement pstmt=con.prepareStatement(sql);  
pstmt.clearParameters();  
pstmt.setInt(1,sid);  
pstmt.setString(2,sname);  
pstmt.setInt(3, rating);  
pstmt.setFloat(4,age);  
  
// we know that no rows are returned, thus we use  
// executeUpdate()  
int numRows = pstmt.executeUpdate();
```



ResultSets

- ❖ `PreparedStatement.executeUpdate` only returns the number of affected records
- ❖ `PreparedStatement.executeQuery` returns data, encapsulated in a `ResultSet` object (a cursor)

```
ResultSet rs=pstmt.executeQuery(sql);  
// rs is now a cursor  
While (rs.next()) {  
    // process the data  
}
```



ResultSets (Contd.)

- A `ResultSet` is a very powerful cursor:
- ❖ `previous()`: moves one row back
 - ❖ `absolute(int num)`: moves to the row with the specified number
 - ❖ `relative (int num)`: moves forward or backward
 - ❖ `first()` and `last()`

Matching Java and SQL Data Types



SQL Type	Java class	ResultSet get method
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.Timestamp	getTimestamp()

JDBC: Exceptions and Warnings



- ❖ Most of java.sql can throw and SQLException if an error occurs.
- ❖ SQLWarning is a subclass of SQLException; not as severe (they are not thrown and their existence has to be explicitly tested)

Warning and Exceptions (Contd.)



```
try {
    stmt=con.createStatement();
    warning=con.getWarnings();
    while(warning != null) {
        // handle SQLWarnings;
        warning = warning.getNextWarning();
    }
    con.clearWarnings();
    stmt.executeUpdate(queryString);
    warning = con.getWarnings();
    ...
} //end try
catch( SQLException SQLe) {
    // handle the exception
}
```

Examining Database Metadata

DatabaseMetaData object gives information about the database system and the catalog.

```
DatabaseMetaData md = con.getMetaData();
// print information about the driver:
System.out.println(
    "Name:" + md.getDriverName() +
    "version:" + md.getDriverVersion());
```



Database Metadata (Contd.)

```
DatabaseMetaData md=con.getMetaData();
ResultSet trs=md.getTables(null,null,null,null);
String tableName;
While(trs.next()) {
    tableName = trs.getString("TABLE_NAME");
    System.out.println("Table: " + tableName);
    //print all attributes
    ResultSet crs = md.getColumns(null,null,tableName, null);
    while (crs.next()) {
        System.out.println(crs.getString("COLUMN_NAME") + ", ");
    }
}
```



A (Semi-)Complete Example

```
Connection con = // connect
    DriverManager.getConnection(url, "login", "pass");
Statement stmt = con.createStatement(); // set up stmt
String query = "SELECT name, rating FROM Sailors";
ResultSet rs = stmt.executeQuery(query);
try { // handle exceptions
    // loop through result tuples
    while (rs.next()) {
        String s = rs.getString("name");
        Int n = rs.getFloat("rating");
        System.out.println(s + " " + n);
    }
} catch(SQLException ex) {
    System.out.println(ex.getMessage ())
    + ex.getSQLState () + ex.getErrorCode ());
}
```



SQLJ



Complements JDBC with a (semi-)static query model:

Compiler can perform syntax checks, strong type checks, consistency of the query with the schema

- All arguments always bound to the same variable:

```
#sql = {  
    SELECT name, rating INTO :name, :rating  
    FROM Books WHERE sid = :sid;  
}  
  
Compare to JDBC:  
sid=rs.getInt(1);  
if (sid==1) {name=rs.getString(2);}  
else { name2=rs.getString(2);}
```

- SQLJ (part of the SQL standard) versus embedded SQL (vendor-specific)

SQLJ Code



```
Int sid; String name; Int rating;  
// named iterator  
#sql iterator Sailors(Int sid, String name, Int rating);  
Sailors sailors;  
// assume that the application sets rating  
#sailors = {  
    SELECT sid, sname INTO :sid, :name  
    FROM Sailors WHERE rating = :rating  
};  
// retrieve results  
while (sailors.next()) {  
    System.out.println(sailors.sid + " " + sailors.sname);  
}  
sailors.close();
```

SQLJ Iterators



Two types of iterators (“cursors”):

- Named iterator

- Need both variable type and name, and then allows retrieval of columns by name.
- See example on previous slide.

- Positional iterator

- Need only variable type, and then uses FETCH .. INTO construct:

```
#sql iterator Sailors(Int, String, Int);  
Sailors sailors;  
#sailors = ...  
while (true) {  
    #sql {FETCH :sailors INTO :sid, :name};  
    if (sailors.endFetch()) { break; }  
    // process the sailor  
}
```

Stored Procedures



❖ What is a stored procedure:

- Program executed through a single SQL statement
- Executed in the process space of the server

❖ Advantages:

- Can encapsulate application logic while staying “close” to the data
- Reuse of application logic by different users
- Avoid tuple-at-a-time return of records through cursors

Stored Procedures: Examples



```
CREATE PROCEDURE ShowNumReservations
  SELECT S.sid, S.sname, COUNT(*)
  FROM Sailors S, Reserves R
  WHERE S.sid = R.sid
  GROUP BY S.sid, S.sname
```

Stored procedures can have parameters:

- ❖ Three different modes: IN, OUT, INOUT

```
CREATE PROCEDURE IncreaseRating(
  IN sailor_sid INTEGER, IN increase INTEGER)
UPDATE Sailors
  SET rating = rating + increase
  WHERE sid = sailor_sid
```

Stored Procedures: Examples (Contd.)



Stored procedure do not have to be written in SQL:

```
CREATE PROCEDURE TopSailors(
  IN num INTEGER)
LANGUAGE JAVA
EXTERNAL NAME "file:///c:/storedProcs/rank.jar"
```

Calling Stored Procedures

```
EXEC SQL BEGIN DECLARE SECTION
Int sid;
Int rating;
EXEC SQL END DECLARE SECTION

// now increase the rating of this sailor
EXEC CALL IncreaseRating(:sid,:rating);
```



Calling Stored Procedures (Contd.)

IDBC: CallableStatement cstmt=con.prepareCall("{call ShowSailors}"); ResultSet rs = cstmt.executeQuery(); while (rs.next()) { ... }	SQL: #sql iterator ShowSailors(...); ShowSailors showsailors; #sql showsailors={CALL ShowSailors}; while (showsailors.next()) { ... }
---	---



SQL/PSM

Most DBMSs allow users to write stored procedures in a simple, general-purpose language (close to SQL) à SQL/PSM standard is a representative

Declare a stored procedure:

```
CREATE PROCEDURE name(p1, p2, ..., pn)
```

```
local variable declarations  
procedure code;
```

Declare a function:

```
CREATE FUNCTION name (p1, ..., pn) RETURNS  
sqlDataType  
local variable declarations  
function code;
```



Main SQL/PSM Constructs

```
CREATE FUNCTION rate Sailor  
  (IN sailorId INTEGER)  
  RETURNS INTEGER  
DECLARE rating INTEGER  
DECLARE numRes INTEGER  
SET numRes = (SELECT COUNT(*)  
              FROM Reserves R  
              WHERE R.sid = sailorId)  
IF (numRes > 10) THEN rating =1;  
ELSE rating = 0;  
END IF;  
RETURN rating;
```



Main SQL/PSM Constructs (Contd.)

- ❖ Local variables (DECLARE)
- ❖ RETURN values for FUNCTION
- ❖ Assign variables with SET
- ❖ Branches and loops:
 - IF (condition) THEN statements;
 - ELSEIF (condition) statements;
 - ... ELSE statements; END IF;
 - LOOP statements; END LOOP
- ❖ Queries can be parts of expressions
- ❖ Can use cursors naturally without “EXEC SQL”



Summary

- ❖ Embedded SQL allows execution of parametrized static queries within a host language
- ❖ Dynamic SQL allows execution of completely ad-hoc queries within a host language
- ❖ Cursor mechanism allows retrieval of one record at a time and bridges impedance mismatch between host language and SQL
- ❖ APIs such as JDBC introduce a layer of abstraction between application and DBMS



Summary (Contd.)



- ❖ SQLJ: Static model, queries checked at compile-time.
- ❖ Stored procedures execute application logic directly at the server
- ❖ SQL/PSM standard for writing stored procedures



Internet Applications

Chapter 7

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

1



Lecture Overview

- ❖ **Internet Concepts**
- ❖ Web data formats
 - HTML, XML, DTDs
- ❖ Introduction to three-tier architectures
- ❖ The presentation layer
 - HTML forms; HTTP Get and POST, URL encoding; Javascript; Stylesheets. XSLT
- ❖ The middle tier
 - CGI, application servers, Servlets, JavaServerPages, passing arguments, maintaining state (cookies)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

2



Uniform Resource Identifiers

- ❖ Uniform naming schema to identify *resources* on the Internet
- ❖ A resource can be anything:
 - Index.html
 - mysong.mp3
 - picture.jpg
- ❖ Example URIs:
<http://www.cs.wisc.edu/~dbbook/index.html>
<mailto:webmaster@bookstore.com>

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

3

Structure of URIs



<http://www.cs.wisc.edu/~dbbook/index.html>

❖ URI has three parts:

- Naming schema ([http](#))
- Name of the host computer ([www.cs.wisc.edu](#))
- Name of the resource ([~dbbook/index.html](#))

❖ URLs are a subset of URIs

Hypertext Transfer Protocol



❖ What is a communication protocol?

- Set of standards that defines the structure of messages
- Examples: TCP, IP, [HTTP](#)

❖ What happens if you click on [www.cs.wisc.edu/~dbbook/index.html](#)?

- ❖ Client (web browser) sends HTTP request to server
- ❖ Server receives request and replies
- ❖ Client receives reply; makes new requests

HTTP (Contd.)



Client to Server:

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

Server replies:

```
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>Barns and Noble Internet
Bookstores</H1>
Our inventory:
<H3>Science</H3>
<B>The Character of Physical Law</B>
...
```

HTTP Protocol Structure



HTTP Requests

- ❖ Request line: **GET ~/index.html HTTP/1.1**
 - **GET**: Http method field (possible values are GET and POST, more later)
 - **~/index.html**: URI field
 - **HTTP/1.1**: HTTP version field
- ❖ Type of client: **User-agent: Mozilla/4.0**
- ❖ What types of files will the client accept:
Accept: text/html, image/gif, image/jpeg

HTTP Protocol Structure (Contd.)



HTTP Responses

- ❖ Status line: **HTTP/1.1 200 OK**
 - HTTP version: **HTTP/1.1**
 - Status code: **200**
 - Server message: **OK**
 - Common status code/server message combinations:
 - **200 OK**: Request succeeded
 - **400 Bad Request**: Request could not be fulfilled by the server
 - **404 Not Found**: Requested object does not exist on the server
 - **505 HTTP Version not Supported**
- ❖ Date when the object was created:
Last-Modified: Mon, 01 Mar 2002 09:23:24 GMT
- ❖ Number of bytes being sent: **Content-Length: 1024**
- ❖ What type is the object being sent: **Content-Type: text/html**
- ❖ Other information such as the server type, server time, etc.

Some Remarks About HTTP



- ❖ **HTTP is stateless**
 - No “sessions”
 - Every message is completely self-contained
 - No previous interaction is “remembered” by the protocol
 - Tradeoff between ease of implementation and ease of application development: Other functionality has to be built on top
- ❖ **Implications for applications:**
 - Any state information (shopping carts, user login-information) need to be encoded in every HTTP request and response!
 - Popular methods on how to maintain state:
 - Cookies (later this lecture)
 - Dynamically generate unique URL's at the server level (later this lecture)

Web Data Formats



- ❖ **HTML**
 - The presentation language for the Internet
- ❖ **Xml**
 - A self-describing, hierachal data model
- ❖ **DTD**
 - Standardizing schemas for Xml
- ❖ **XSLT** (not covered in the book)

HTML: An Example



```
<HTML>
<HEAD></HEAD>
<BODY>
<H1>Barns and Nobble Internet
Bookstore</H1>
Our inventory:
<H3>Science</H3>
<B>The Character of Physical
Law</B>
<UL>
<LI>Author: Richard
Feynman</LI>
<LI>Published 1980</LI>
<LI>Hardcover</LI>
</UL>
<H3>Fiction</H3>
<B>Waiting for the Mahatma</B>
<UL>
<LI>Author: R.K. Narayan</LI>
<LI>Published 1981</LI>
</UL>
<B>The English Teacher</B>
<UL>
<LI>Author: R.K. Narayan</LI>
<LI>Published 1980</LI>
<LI>Paperback</LI>
</UL>
</BODY>
</HTML>
```

HTML: A Short Introduction



- ❖ **HTML is a markup language**
- ❖ **Commands are tags:**
 - Start tag and end tag
 - Examples:
 - <HTML> ... </HTML>
 - ...
- ❖ **Many editors automatically generate HTML directly from your document (e.g., Microsoft Word has an “Save as html” facility)**

HTML: Sample Commands



- ❖ <HTML>
- ❖ : unordered list
- ❖ : list entry
- ❖ <h1>: largest heading
- ❖ <h2>: second-level heading, <h3>, <h4> analogous
- ❖ Title: Bold

XML: An Example



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<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>
  <BOOK genre="Fiction">
    <AUTHOR>
      <firstname>R.K.</firstname><lastname>Narayan</lastname>
    </AUTHOR>
    <TITLE>Waiting for the Mahatma</TITLE>
    <PUBLISHED>1981</PUBLISHED>
  </BOOK>
  <BOOK genre="Fiction">
    <AUTHOR>
      <firstname>R.K.</firstname><lastname>Narayan</lastname>
    </AUTHOR>
    <TITLE>The English Teachers</TITLE>
    <PUBLISHED>1980</PUBLISHED>
  </BOOK>
</BOOKLIST>
```

XML – The Extensible Markup Language



- ❖ Language
 - A way of communicating information
- ❖ Markup
 - Notes or meta-data that describe your data or language
- ❖ Extensible
 - Limitless ability to define new languages or data sets

XML – What's The Point?



- ❖ You can include your data and a description of what the data represents
 - This is useful for defining your own language or protocol
- ❖ Example: Chemical Markup Language

```
<molecule>
    <weight>234.5</weight>
    <Spectra>...</Spectra>
    <Figures>...</Figures>
</molecule>
```
- ❖ XML design goals:
 - XML should be compatible with SGML
 - It should be easy to write XML processors
 - The design should be formal and precise

XML – Structure



- ❖ XML: Confluence of SGML and HTML
- ❖ XML looks like HTML
- ❖ XML is a hierarchy of user-defined tags called elements with attributes and data
- ❖ Data is described by elements, elements are described by attributes

```
<BOOK genre="Science" format="Hardcover">...</BOOK>
```

XML – Elements



- ```
<BOOK genre="Science" format="Hardcover">...</BOOK>
```
- 
- ❖ XML is case and space sensitive
  - ❖ Element opening and closing tag names must be identical
  - ❖ Opening tags: "<" + element name + ">"
  - ❖ Closing tags: "</>" + element name + ">"
  - ❖ Empty Elements have no data and no closing tag:
    - They begin with a "<" and end with a ">"

---

---

---

---

---

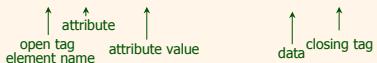
---

---

## XML – Attributes



```
<BOOK genre="Science" format="Hardcover">...</BOOK>
```



- ❖ Attributes provide additional information for element tags.
- ❖ There can be zero or more attributes in every element; each one has the form:  
*attribute\_name="attribute\_value"*
  - There is no space between the name and the “=“
  - Attribute values must be surrounded by ““ characters
- ❖ Multiple attributes are separated by white space (one or more spaces or tabs).

---

---

---

---

---

---

## XML – Data and Comments



```
<BOOK genre="Science" format="Hardcover">...</BOOK>
```



- ❖ Xml data is any information between an opening and closing tag
- ❖ Xml data must not contain the ‘<‘ or ‘>‘ characters
- ❖ Comments:  
  <!-- comment -->

---

---

---

---

---

---

## XML – Nesting & Hierarchy



- ❖ Xml tags can be nested in a tree hierarchy
- ❖ Xml documents can have only one root tag
- ❖ Between an opening and closing tag you can insert:
  1. Data
  2. More Elements
  3. A combination of data and elements

```
<root>
 <tag1>
 Some Text
 <tag2>More</tag2>
 </tag1>
</root>
```

---

---

---

---

---

---

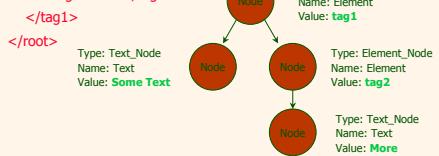
## Xml – Storage



- ❖ Storage is done just like an n-ary tree (DOM)

```
<root>
 <tag1>
 Some Text
 <tag2>More</tag2>
 </tag1>
</root>
```

Type: Text\_Node  
Name: Text  
Value: Some Text



---

---

---

---

---

---

---

## DTD – Document Type Definition



- ❖ A DTD is a schema for Xml data
- ❖ Xml protocols and languages can be standardized with DTD files
- ❖ A DTD says what elements and attributes are required or optional
  - Defines the formal structure of the language

---

---

---

---

---

---

---

## DTD – An Example



```
<?xml version='1.0?'>
<!ELEMENT Basket (Cherry+, (Apple | Orange)*)>
<!ELEMENT Cherry EMPTY>
 <!ATTLIST Cherry flavor CDATA #REQUIRED>
<!ELEMENT Apple EMPTY>
 <!ATTLIST Apple color CDATA #REQUIRED>
<!ELEMENT Orange EMPTY>
 <!ATTLIST Orange location 'Florida'>
```

  &gt; <Basket>  
    <Cherry flavor='good'/>    <Basket>  
    <Apple color='red'/>     <Apple/>  
    <Apple color='green'/>    <Cherry flavor='good'/>  
  </Basket>                <Orange/>  
                          </Basket>

---

---

---

---

---

---

---

## DTD - !ELEMENT



```
<!ELEMENT Basket (Cherry+, (Apple | Orange)*) >
```

- ❖ !ELEMENT Name Children
- ❖ !ELEMENT declares an element name, and what children elements it should have
- ❖ Content types:
  - Other elements
  - #PCDATA (parsed character data)
  - EMPTY (no content)
  - ANY (no checking inside this structure)
  - A regular expression

---

---

---

---

---

---

---

## DTD - !ELEMENT (Contd.)



- ❖ A regular expression has the following structure:
  - $\text{exp}_1, \text{exp}_2, \text{exp}_3, \dots, \text{exp}_k$ : A list of regular expressions
  - $\text{exp}^*$ : An optional expression with zero or more occurrences
  - $\text{exp}^+$ : An optional expression with one or more occurrences
  - $\text{exp}_1 \mid \text{exp}_2 \mid \dots \mid \text{exp}_k$ : A disjunction of expressions

---

---

---

---

---

---

---

## DTD - !ATTLIST



```
<!ATTLIST Cherry flavor CDATA #REQUIRED>
 Element Attribute Type Flag
<!ATTLIST Orange location CDATA #REQUIRED
 color 'orange'>
```

- ❖ !ATTLIST defines a list of attributes for an element
- ❖ Attributes can be of different types, can be required or not required, and they can have default values.

---

---

---

---

---

---

---

## DTD – Well-Formed and Valid

```
<?xml version='1.0'?>
<!ELEMENT Basket (Cherry+)>
<!ELEMENT Cherry EMPTY>
<!ATTLIST Cherry flavor CDATA #REQUIRED>
```

Not Well-Formed	Well-Formed but Invalid
<basket>	<Job>
<Cherry flavor=good>	<Location>Home</Location>
</Basket>	</Job>

Well-Formed and Valid
<Basket>
<Cherry flavor='good'/>
</Basket>



## XML and DTDs

- ❖ More and more standardized DTDs will be developed
  - MathML
  - Chemical Markup Language
- ❖ Allows light-weight exchange of data with the same semantics
- ❖ Sophisticated query languages for XML are available:
  - Xquery
  - XPath



## Lecture Overview

- ❖ Internet Concepts
- ❖ Web data formats
  - HTML, XML, DTDs
- ❖ Introduction to three-tier architectures
- ❖ The presentation layer
  - HTML forms; HTTP Get and POST, URL encoding; Javascript; Stylesheets. XSLT
- ❖ The middle tier
  - CGI, application servers, Servlets, JavaServerPages, passing arguments, maintaining state (cookies)

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## *Components of Data-Intensive Systems*



Three separate types of functionality:

- ❖ Data management
- ❖ Application logic
- ❖ Presentation

- ❖ The system architecture determines whether these three components reside on a single system ("tier") or are distributed across several tiers

---

---

---

---

---

---

## *Single-Tier Architectures*



All functionality combined into a single tier, usually on a mainframe

- User access through dumb terminals

Advantages:

- Easy maintenance and administration

Disadvantages:

- Today, users expect graphical user interfaces.
- Centralized computation of all of them is too much for a central system

---

---

---

---

---

---

## *Client-Server Architectures*



Work division: Thin client

- Client implements only the graphical user interface
- Server implements business logic and data management

❖ Work division: Thick client

- Client implements both the graphical user interface and the business logic
- Server implements data management

---

---

---

---

---

---

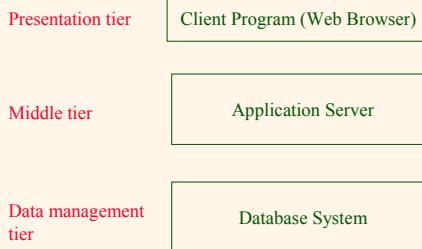
## *Client-Server Architectures (Contd.)*

### Disadvantages of thick clients

- No central place to update the business logic
- Security issues: Server needs to trust clients
  - Access control and authentication needs to be managed at the server
  - Clients need to leave server database in consistent state
  - One possibility: Encapsulate all database access into stored procedures
- Does not scale to more than several 100s of clients
  - Large data transfer between server and client
  - More than one server creates a problem: x clients, y servers:  $x \times y$  connections



## *The Three-Tier Architecture*



---

---

---

---

---

---

---

---

---

---

---

---

## *The Three Layers*



### **Presentation tier**

- Primary interface to the user
- Needs to adapt to different display devices (PC, PDA, cell phone, voice access?)

### **Middle tier**

- Implements business logic (implements complex actions, maintains state between different steps of a workflow)
- Accesses different data management systems

### **Data management tier**

- One or more standard database management systems

---

---

---

---

---

---

---

---

---

---

---

---

## *Example 1: Airline reservations*



- ❖ Build a system for making airline reservations
- ❖ What is done in the different tiers?
- ❖ Database System
  - Airline info, available seats, customer info, etc.
- ❖ Application Server
  - Logic to make reservations, cancel reservations, add new airlines, etc.
- ❖ Client Program
  - Log in different users, display forms and human-readable output

---

---

---

---

---

---

---

## *Example 2: Course Enrollment*



- ❖ Build a system using which students can enroll in courses
- ❖ Database System
  - Student info, course info, instructor info, course availability, pre-requisites, etc.
- ❖ Application Server
  - Logic to add a course, drop a course, create a new course, etc.
- ❖ Client Program
  - Log in different users (students, staff, faculty), display forms and human-readable output

---

---

---

---

---

---

---

## *Technologies*



Client Program ( <i>Web Browser</i> )
Application Server ( <i>Tomcat, Apache</i> )
Database System ( <i>DB2</i> )

*HTML  
Javascript  
XSLT*

*JSP  
Servlets  
Cookies  
CGI*

*XML  
Stored Procedures*

---

---

---

---

---

---

---

## *Advantages of the Three-Tier Architecture*



- ❖ Heterogeneous systems
  - Tiers can be independently maintained, modified, and replaced
- ❖ Thin clients
  - Only presentation layer at clients (web browsers)
- ❖ Integrated data access
  - Several database systems can be handled transparently at the middle tier
  - Central management of connections
- ❖ Scalability
  - Replication at middle tier permits scalability of business logic
- ❖ Software development
  - Code for business logic is centralized
  - Interaction between tiers through well-defined APIs: Can reuse standard components at each tier

## *Lecture Overview*



- ❖ Internet Concepts
- ❖ Web data formats
  - HTML, XML, DTDs
- ❖ Introduction to three-tier architectures
- ❖ The presentation layer
  - HTML forms; HTTP Get and POST, URL encoding; Javascript; Stylesheets. XSLT
- ❖ The middle tier
  - CGI, application servers, Servlets, JavaServerPages, passing arguments, maintaining state (cookies)

## *Overview of the Presentation Tier*



- ❖ Recall: Functionality of the presentation tier
  - Primary interface to the user
  - Needs to adapt to different display devices (PC, PDA, cell phone, voice access?)
  - Simple functionality, such as field validity checking
- ❖ We will cover:
  - HTML Forms: How to pass data to the middle tier
  - JavaScript: Simple functionality at the presentation tier
  - Style sheets: Separating data from formatting

## HTML Forms



- ❖ Common way to communicate data from client to middle tier
- ❖ General format of a form:
  - <FORM ACTION="page.jsp" METHOD="GET" NAME="LoginForm">
  - ...</FORM>
- ❖ Components of an HTML FORM tag:
  - **ACTION:** Specifies URI that handles the content
  - **METHOD:** Specifies HTTP GET or POST method
  - **NAME:** Name of the form; can be used in client-side scripts to refer to the form

---

---

---

---

---

---

---

## Inside HTML Forms



- ❖ INPUT tag
  - Attributes:
    - TYPE: text (text input field), password (text input field where input is, reset (resets all input fields)
    - NAME: symbolic name, used to identify field value at the middle tier
    - VALUE: default value
  - Example: <INPUT TYPE="text" Name="title">
- ❖ Example form:

```
<form method="POST" action="TableOfContents.jsp">
<input type="text" name="userid">
<input type="password" name="password">
<input type="submit" value="Login" name="submit">
<input type="reset" value="Clear">
</form>
```

---

---

---

---

---

---

---

## Passing Arguments



Two methods: GET and POST

- ❖ GET
  - Form contents go into the submitted URI
  - Structure:  
`action?name1=value1&name2=value2&name3=value3`
    - Action: name of the URI specified in the form
    - (name,value)-pairs come from INPUT fields in the form; empty fields have empty values ("name=")
  - Example from previous password form:  
`TableOfContents.jsp?userid=john&password=johnpw`
  - Note that the page named action needs to be a program, script, or page that will process the user input

---

---

---

---

---

---

---

## *HTTP GET: Encoding Form Fields*

- ❖ Form fields can contain general ASCII characters that cannot appear in an URI
- ❖ A special encoding convention converts such field values into “URI-compatible” characters:
  - Convert all “special” characters to %xyz, where xyz is the ASCII code of the character. Special characters include &, =, +, %, etc.
  - Convert all spaces to the “+” character
  - Glue (name,value)-pairs from the form INPUT tags together with “&” to form the URI



---

---

---

---

---

---

---

## *HTML Forms: A Complete Example*

```
<form method="POST" action="TableOfContents.jsp">
<table align = "center" border="0" width="300">
<tr>
 <td>UserId</td>
 <td><input type="text" name="userId" size="20"></td>
</tr>
<tr>
 <td>Password</td>
 <td><input type="password" name="password" size="20"></td>
</tr>
<tr>
 <td align = "center"><input type="submit" value="Login"
 name="submit"></td>
</tr>
</table>
</form>
```



---

---

---

---

---

---

---

## *JavaScript*

- ❖ Goal: Add functionality to the presentation tier.
- ❖ Sample applications:
  - Detect browser type and load browser-specific page
  - Form validation: Validate form input fields
  - Browser control: Open new windows, close existing windows (example: pop-up ads)
- ❖ Usually embedded directly inside the HTML with the **<SCRIPT> ... </SCRIPT>** tag.
- ❖ **<SCRIPT>** tag has several attributes:
  - LANGUAGE: specifies language of the script (such as javascript)
  - SRC: external file with script code
  - Example:  
**<SCRIPT LANGUAGE="JavaScript" SRC="validate.js>**  
**</SCRIPT>**



---

---

---

---

---

---

---

## JavaScript (Contd.)



- ❖ If <SCRIPT> tag does not have a SRC attribute, then the JavaScript is directly in the HTML file.
- ❖ Example:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- alert("Welcome to our bookstore")
//-->
</SCRIPT>
```
- ❖ Two different commenting styles
  - <!-- comment for HTML, since the following JavaScript code should be ignored by the HTML processor
  - // comment for JavaScript in order to end the HTML comment

---

---

---

---

---

---

## JavaScript (Contd.)



- ❖ JavaScript is a complete scripting language
  - Variables
  - Assignments (=, +=, ...)
  - Comparison operators (<, >, ...), boolean operators (&&, ||, !)
  - Statements
    - if (condition) {statements;} else {statements;}
    - for loops, do-while loops, and while-loops
  - Functions with return values
    - Create functions using the function keyword
    - f(arg1, ..., argk) {statements;}

---

---

---

---

---

---

## JavaScript: A Complete Example



### HTML Form:

```
<form method="POST"
action="TableOfContents.jsp">
<input type="text"
name="userid">
<input type="password"
name="password">
<input type="submit"
value="Login"
name="submit">
<input type="reset"
value="Clear">
</form>
```

### Associated JavaScript:

```
<script language="javascript">
function testLoginEmpty()
{
 loginForm = document.LoginForm
 if ((loginForm.userid.value == "") ||
 (loginForm.password.value == ""))
 {
 alert('Please enter values for userid and
 password');
 return false;
 }
 else return true;
}
</script>
```

---

---

---

---

---

---

## Stylesheets



- ❖ Idea: Separate display from contents, and adapt display to different presentation formats
- ❖ Two aspects:
  - Document transformations to decide what parts of the document to display in what order
  - Document rendering to decide how each part of the document is displayed
- ❖ Why use stylesheets?
  - Reuse of the same document for different displays
  - Tailor display to user's preferences
  - Reuse of the same document in different contexts
- ❖ Two stylesheet languages
  - Cascading style sheets (CSS): For HTML documents
  - Extensible stylesheet language (XSL): For XML documents

---

---

---

---

---

---

---

## CSS: Cascading Style Sheets



- ❖ Defines how to display HTML documents
- ❖ Many HTML documents can refer to the same CSS
  - Can change format of a website by changing a single style sheet
  - Example:  
`<LINK REL="style sheet" TYPE="text/css" HREF="books.css"/>`

Each line consists of three parts:

- ❖ Selector {property: value}
- ❖ Selector: Tag whose format is defined
- ❖ Property: Tag's attribute whose value is set
- ❖ Value: value of the attribute

---

---

---

---

---

---

---

## CSS: Cascading Style Sheets



Example style sheet:

```
body {background-color: yellow}
h1 {font-size: 36pt}
h3 {color: blue}
p {margin-left: 50px; color: red}
```

The first line has the same effect as:

```
<body background-color="yellow">
```

---

---

---

---

---

---

---

## XSL



- ❖ Language for expressing style sheets
  - More at: <http://www.w3.org/Style/XSL/>
- ❖ Three components
  - XSLT: XSL Transformation language
    - Can transform one document to another
    - More at <http://www.w3.org/TR/xslt>
  - XPath XML Path Language
    - Selects parts of an XML document
    - More at <http://www.w3.org/TR/xpath>
  - XSL Formatting Objects
    - Formats the output of an XSL transformation
    - More at [http://www.w3.org/TR/xsl/](http://www.w3.org/TR/xsl)

## Lecture Overview



- ❖ Internet Concepts
- ❖ Web data formats
  - HTML, XML, DTDs
- ❖ Introduction to three-tier architectures
- ❖ The presentation layer
  - HTML forms; HTTP Get and POST, URL encoding; Javascript; Stylesheets. XSLT
- ❖ The middle tier
  - CGI, application servers, Servlets, JavaServerPages, passing arguments, maintaining state (cookies)

## Overview of the Middle Tier



- ❖ Recall: Functionality of the middle tier
  - Encodes business logic
  - Connects to database system(s)
  - Accepts form input from the presentation tier
  - Generates output for the presentation tier
- ❖ We will cover
  - CGI: Protocol for passing arguments to programs running at the middle tier
  - Application servers: Runtime environment at the middle tier
  - Servlets: Java programs at the middle tier
  - JavaServerPages: Java scripts at the middle tier
  - Maintaining state: How to maintain state at the middle tier. Main focus: Cookies.

## CGI: Common Gateway Interface



- ❖ Goal: Transmit arguments from HTML forms to application programs running at the middle tier
- ❖ Details of the actual CGI protocol unimportant à libraries implement high-level interfaces
- ❖ Disadvantages:
  - The application program is invoked in a new process at every invocation (remedy: FastCGI)
  - No resource sharing between application programs (e.g., database connections)
  - Remedy: Application servers

---

---

---

---

---

---

---

## CGI: Example



- ❖ HTML form:

```
<form action="findbooks.cgi" method=POST>
Type an author name:
<input type="text" name="authorName">
<input type="submit" value="Send it">
<input type="reset" value="Clear form">
</form>
```
- ❖ Perl code:

```
use CGI;
$dataIn=new CGI;
$dataIn->header();
$authorName=$dataIn->param('authorName');
print("<HTML><TITLE>Argument passing test</TITLE>");
print("The author name is " + $authorName);
print("</HTML>");
```

---

---

---

---

---

---

---

## Application Servers



- ❖ Idea: Avoid the overhead of CGI
  - Main pool of threads of processes
  - Manage connections
  - Enable access to heterogeneous data sources
  - Other functionality such as APIs for session management

---

---

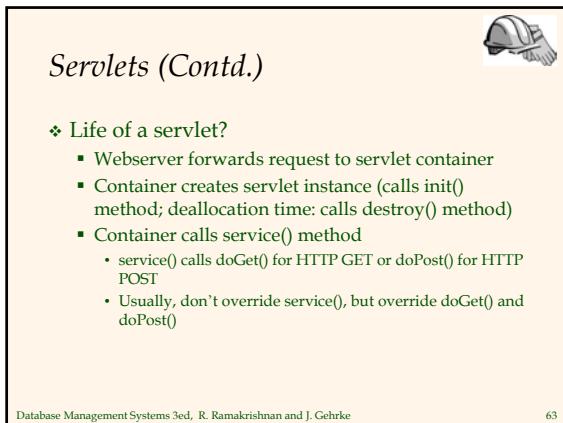
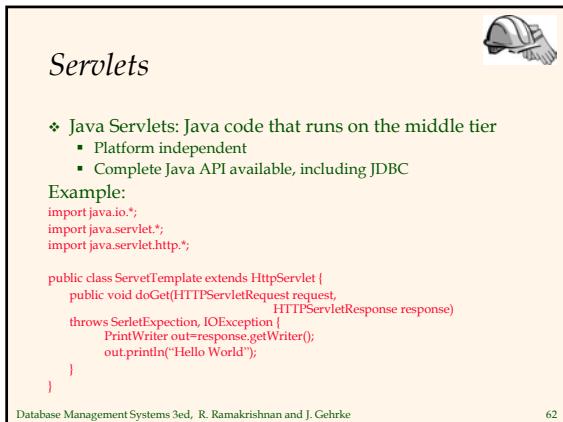
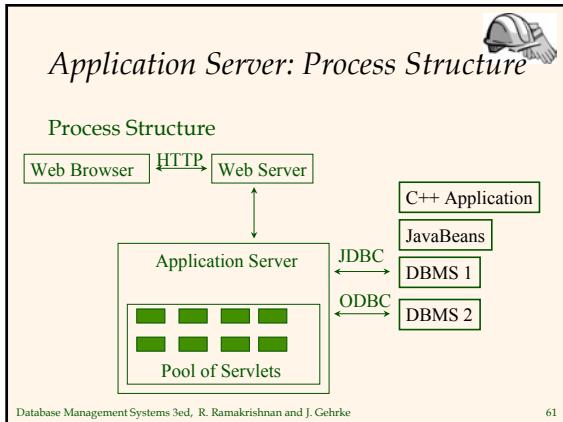
---

---

---

---

---



## Servlets: A Complete Example



```
public class ReadUserName extends HttpServlet {
 public void doGet(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out=response.getWriter();
 out.println("<HTML><BODY>\n \n" +
 "<LD>" + request.getParameter("userid") + "\n" +
 "<LD>" + request.getParameter("password") + "\n" +
 "\n<BODY></HTML>");
 }
 public void doPost(HttpServletRequest request,
 HttpServletResponse response)
 throws ServletException, IOException {
 doGet(request,response);
 }
}
```

## Java Server Pages



### ❖ Servlets

- Generate HTML by writing it to the “PrintWriter” object
- Code first, webpage second

### ❖ JavaServerPages

- Written in HTML, Servlet-like code embedded in the HTML
- Webpage first, code second
- They are usually compiled into a Servlet

## JavaServerPages: Example



```
<html><head><title>Welcome to B&N</title></head>
<body>
 <h1>Welcome back!</h1>
 <% String name="NewUser";
 if (request.getParameter("username") != null) {
 name=request.getParameter("username");
 }
 %>
 You are logged on as user <%=name%>
 <p>
</body>
</html>
```

## Maintaining State



HTTP is stateless.

❖ Advantages

- Easy to use: don't need anything
  - Great for static-information applications
  - Requires no extra memory space
- ❖ Disadvantages
- No record of previous requests means
    - No shopping baskets
    - No user logins
    - No custom or dynamic content
    - Security is more difficult to implement

---

---

---

---

---

---

## Application State



❖ Server-side state

- Information is stored in a database, or in the application layer's local memory

❖ Client-side state

- Information is stored on the client's computer in the form of a cookie

❖ Hidden state

- Information is hidden within dynamically created web pages

---

---

---

---

---

---

## Application State



So many kinds of  
state...  
...how will I choose?



---

---

---

---

---

---

## Server-Side State



- ❖ Many types of Server side state:
  1. Store information in a database
    - Data will be safe in the database
    - BUT: requires a database access to query or update the information
  2. Use application layer's local memory
    - Can map the user's IP address to some state
    - BUT: this information is volatile and takes up lots of server main memory

5 million IPs = 20 MB

---

---

---

---

---

---

---

## Server-Side State



- ❖ Should use Server-side state maintenance for information that needs to persist
  - Old customer orders
  - "Click trails" of a user's movement through a site
  - Permanent choices a user makes

---

---

---

---

---

---

---

## Client-side State: Cookies



- ❖ Storing text on the client which will be passed to the application with every HTTP request.
  - Can be disabled by the client.
  - Are wrongfully perceived as "dangerous", and therefore will scare away potential site visitors if asked to enable cookies<sup>1</sup>
- ❖ Are a collection of (Name, Value) pairs

---

---

---

---

---

---

---

## Client State: Cookies



- ❖ Advantages
  - Easy to use in Java Servlets / JSP
  - Provide a simple way to persist non-essential data on the client even when the browser has closed
- ❖ Disadvantages
  - Limit of 4 kilobytes of information
  - Users can (and often will) disable them
- ❖ Should use cookies to store interactive state
  - The current user's login information
  - The current shopping basket
  - Any non-permanent choices the user has made

---

---

---

---

---

---

---

## Creating A Cookie



```
Cookie myCookie =
 new Cookie("username", "jeffd");
response.addCookie(userCookie);
```

- ❖ You can create a cookie at any time



---

---

---

---

---

---

---

## Accessing A Cookie



```
Cookie[] cookies = request.getCookies();
String theUser;
for(int i=0; i<cookies.length; i++) {
 Cookie cookie = cookies[i];
 if(cookie.getName().equals("username")) theUser =
 cookie.getValue();
}
// at this point theUser == "username"

❖ Cookies need to be accessed BEFORE you set your response header:
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
```

---

---

---

---

---

---

---

## Cookie Features



- ❖ Cookies can have
  - A duration (expire right away or persist even after the browser has closed)
  - Filters for which domains/directory paths the cookie is sent to
- ❖ See the Java Servlet API and Servlet Tutorials for more information

---

---

---

---

---

---

---

## Hidden State



- ❖ Often users will disable cookies
- ❖ You can “hide” data in two places:
  - Hidden fields within a form
  - Using the path information
- ❖ Requires no “storage” of information because the state information is passed inside of each web page

---

---

---

---

---

---

---

## Hidden State: Hidden Fields



- ❖ Declare hidden fields within a form:
  - <input type='hidden' name='user' value='username' />
- ❖ Users will not see this information (unless they view the HTML source)
- ❖ If used prolifically, it's a killer for performance since EVERY page must be contained within a form.

---

---

---

---

---

---

---

## *Hidden State: Path Information*



- ❖ Path information is stored in the URL request:  
`http://server.com/index.htm?user=jeffd`
- ❖ Can separate ‘fields’ with an & character:  
`index.htm?user=jeffd&preference=pepsi`
- ❖ There are mechanisms to parse this field in Java. Check out the `javax.servlet.http.HttpUtils parserQueryString()` method.

---

---

---

---

---

---

---

## *Multiple state methods*



- ❖ Typically all methods of state maintenance are used:
  - User logs in and this information is stored in a cookie
  - User issues a query which is stored in the path information
  - User places an item in a shopping basket cookie
  - User purchases items and credit-card information is stored/retrieved from a database
  - User leaves a click-stream which is kept in a log on the web server (which can later be analyzed)

---

---

---

---

---

---

---

## *Summary*



### We covered:

- ❖ Internet Concepts (URLs, HTTP)
- ❖ Web data formats
  - HTML, XML, DTDs
- ❖ Three-tier architectures
- ❖ The presentation layer
  - HTML forms; HTTP Get and POST, URL encoding; Javascript; Stylesheets, XSLT
- ❖ The middle tier
  - CGI, application servers, Servlets, JavaServerPages, passing arguments, maintaining state (cookies)

---

---

---

---

---

---

---



## *Overview of Storage and Indexing*

### Chapter 8

"How index-learning turns no student pale  
Yet holds the eel of science by the tail."  
-- Alexander Pope (1688-1744)

---

---

---

---

---

---



## *Data on External Storage*

- ❖ **Disks:** Can retrieve random page at fixed cost
  - But reading several consecutive pages is much cheaper than reading them in random order
- ❖ **Tapes:** Can only read pages in sequence
  - Cheaper than disks; used for archival storage
- ❖ **File organization:** Method of arranging a file of records on external storage.
  - Record id (rid) is sufficient to physically locate record
  - Indexes are data structures that allow us to find the record ids of records with given values in index search key fields
- ❖ **Architecture:** Buffer manager stages pages from external storage to main memory buffer pool. File and index layers make calls to the buffer manager.

---

---

---

---

---

---



## *Alternative File Organizations*

Many alternatives exist, *each ideal for some situations, and not so good in others:*

- **Heap (random order) files:** Suitable when typical access is a file scan retrieving all records.
- **Sorted Files:** Best if records must be retrieved in some order, or only a 'range' of records is needed.
- **Indexes:** Data structures to organize records via trees or hashing.
  - Like sorted files, they speed up searches for a subset of records, based on values in certain ("search key") fields
  - Updates are much faster than in sorted files.

---

---

---

---

---

---

## Indexes



- ❖ An *index* on a file speeds up selections on the *search key fields* for the index.
  - Any subset of the fields of a relation can be the search key for an index on the relation.
  - *Search key* is not the same as *key* (minimal set of fields that uniquely identify a record in a relation).
- ❖ An index contains a collection of *data entries*, and supports efficient retrieval of all data entries  $k^*$  with a given key value  $k$ .

---

---

---

---

---

---

## Alternatives for Data Entry $k^*$ in Index



- ❖ Three alternatives:
  - Data record with key value  $k$
  - $\langle k, \text{rid of data record with search key value } k \rangle$
  - $\langle k, \text{list of rids of data records with search key } k \rangle$
- ❖ Choice of alternative for data entries is orthogonal to the indexing technique used to locate data entries with a given key value  $k$ .
  - Examples of indexing techniques: B+ trees, hash-based structures
  - Typically, index contains auxiliary information that directs searches to the desired data entries

---

---

---

---

---

---

## Alternatives for Data Entries (Contd.)



- ❖ Alternative 1:
  - If this is used, index structure is a file organization for data records (instead of a Heap file or sorted file).
  - At most one index on a given collection of data records can use Alternative 1. (Otherwise, data records are duplicated, leading to redundant storage and potential inconsistency.)
  - If data records are very large, # of pages containing data entries is high. Implies size of auxiliary information in the index is also large, typically.

---

---

---

---

---

---

## Alternatives for Data Entries (Contd.)

### ❖ Alternatives 2 and 3:

- Data entries typically much smaller than data records. So, better than Alternative 1 with large data records, especially if search keys are small. (Portion of index structure used to direct search, which depends on size of data entries, is much smaller than with Alternative 1.)
- Alternative 3 more compact than Alternative 2, but leads to variable sized data entries even if search keys are of fixed length.

---

---

---

---

---

---

---

## Index Classification

- ❖ *Primary vs. secondary*: If search key contains primary key, then called primary index.
  - Unique index: Search key contains a candidate key.
- ❖ *Clustered vs. unclustered*: If order of data records is the same as, or 'close to', order of data entries, then called clustered index.
  - Alternative 1 implies clustered; in practice, clustered also implies Alternative 1 (since sorted files are rare).
  - A file can be clustered on at most one search key.
  - Cost of retrieving data records through index varies greatly based on whether index is clustered or not!

---

---

---

---

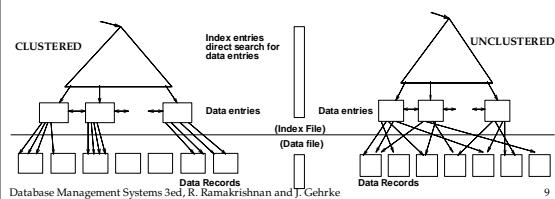
---

---

---

## Clustered vs. Unclustered Index

- ❖ Suppose that Alternative (2) is used for data entries, and that the data records are stored in a Heap file.
  - To build clustered index, first sort the Heap file (with some free space on each page for future inserts).
  - Overflow pages may be needed for inserts. (Thus, order of data recs is 'close to', but not identical to, the sort order.)



---

---

---

---

---

---

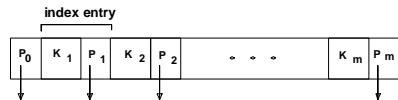
---

## Hash-Based Indexes

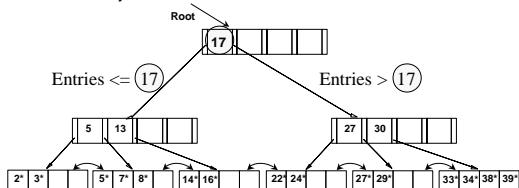
- ❖ Good for equality selections.
  - Index is a collection of *buckets*. Bucket = *primary* page plus zero or more *overflow* pages.
  - Hashing function  $h$ :  $h(r)$  = bucket in which record  $r$  belongs.  $h$  looks at the *search key* fields of  $r$ .
- ❖ If Alternative (1) is used, the buckets contain the data records; otherwise, they contain  $\langle \text{key}, \text{rid} \rangle$  or  $\langle \text{key}, \text{rid-list} \rangle$  pairs.

## B+ Tree Indexes

- ❖ Leaf pages contain *data entries*, and are chained (prev & next)
- ❖ Non-leaf pages contain *index entries* and direct searches:



## Example B+ Tree



- ❖ Find 28\*? 29\*? All  $> 15^*$  and  $< 30^*$
- ❖ Insert/delete: Find data entry in leaf, then change it. Need to adjust parent sometimes.
  - And change sometimes bubbles up the tree

## *Cost Model for Our Analysis*



We ignore CPU costs, for simplicity:

- **B**: The number of data pages
- **R**: Number of records per page
- **D**: (Average) time to read or write disk page
- Measuring number of page I/O's ignores gains of pre-fetching a sequence of pages; thus, even I/O cost is only approximated.
- Average-case analysis; based on several simplistic assumptions.

\* Good enough to show the overall trends!

---

---

---

---

---

---

---

---

## *Comparing File Organizations*



- ❖ Heap files (random order; insert at eof)
- ❖ Sorted files, sorted on  $\langle age, sal \rangle$
- ❖ Clustered B+ tree file, Alternative (1), search key  $\langle age, sal \rangle$
- ❖ Heap file with unclustered B + tree index on search key  $\langle age, sal \rangle$
- ❖ Heap file with unclustered hash index on search key  $\langle age, sal \rangle$

---

---

---

---

---

---

---

---

## *Operations to Compare*



- ❖ Scan: Fetch all records from disk
- ❖ Equality search
- ❖ Range selection
- ❖ Insert a record
- ❖ Delete a record

---

---

---

---

---

---

---

---

### *Assumptions in Our Analysis*



- ❖ Heap Files:
    - Equality selection on key; exactly one match.
  - ❖ Sorted Files:
    - Files compacted after deletions.
  - ❖ Indexes:
    - Alt (2), (3): data entry size = 10% size of record
    - Hash: No overflow buckets.
      - 80% page occupancy => File size = 1.25 data size
    - Tree: 67% occupancy (this is typical).
      - Implies file size = 1.5 data size

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

16



## *Cost of Operations*

	(a) Scan	(b) Equality	(c) Range	(d) Insert	(e) Delete
(1) Heap					
(2) Sorted					
(3) Clustered					
(4) Unclustered Tree index					
(5) Unclustered Hash index					

\* Several assumptions underlie these (rough) estimates!

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

17



## *Cost of Operations*

	(a) Scan	(b) Equality	(c) Range	(d) Insert	(e) Delete
(1) Heap	BD	0.5BD	BD	2D	Search +D
(2) Sorted	BD	Dlog zB	Dlog zB + # matches	Search + BD	Search +BD
(3) Clustered	1.5BD	Dlog f 1.5B	Dlog f 1.5B + # matches	Search + D	Search +D
(4) Unclustered Tree index	BD(R+0.15)	D(1 + log f 0.15B)	Dlog f 0.15B + # matches	D(3 + log f 0.15B)	Search + 2D
(5) Unclustered Hash index	BD(R+0.125)	2D	BD	4D	Search + 2D

\* Several assumptions underlie these (rough) estimates!

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

18

## *Understanding the Workload*



- ❖ For each query in the workload:
  - Which relations does it access?
  - Which attributes are retrieved?
  - Which attributes are involved in selection/join conditions?  
How selective are these conditions likely to be?
- ❖ For each update in the workload:
  - Which attributes are involved in selection/join conditions?  
How selective are these conditions likely to be?
  - The type of update (INSERT/DELETE/UPDATE), and the attributes that are affected.

---

---

---

---

---

---

---

## *Choice of Indexes*



- ❖ What indexes should we create?
  - Which relations should have indexes? What field(s) should be the search key? Should we build several indexes?
- ❖ For each index, what kind of an index should it be?
  - Clustered? Hash/tree?

---

---

---

---

---

---

---

## *Choice of Indexes (Contd.)*



- ❖ One approach: Consider the most important queries in turn. Consider the best plan using the current indexes, and see if a better plan is possible with an additional index. If so, create it.
  - Obviously, this implies that we must understand how a DBMS evaluates queries and creates query evaluation plans!
  - For now, we discuss simple 1-table queries.
- ❖ Before creating an index, must also consider the impact on updates in the workload!
  - Trade-off: Indexes can make queries go faster, updates slower. Require disk space, too.

---

---

---

---

---

---

---

## *Index Selection Guidelines*



- ❖ Attributes in WHERE clause are candidates for index keys.
    - Exact match condition suggests hash index.
    - Range query suggests tree index.
      - Clustering is especially useful for range queries; can also help on equality queries if there are many duplicates.
  - ❖ Multi-attribute search keys should be considered when a WHERE clause contains several conditions.
    - Order of attributes is important for range queries.
    - Such indexes can sometimes enable index-only strategies for important queries.
      - For index-only strategies, clustering is not important!
  - ❖ Try to choose indexes that benefit as many queries as possible. Since only one index can be clustered per relation, choose it based on important queries that would benefit the most from clustering.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

22

## *Examples of Clustered Indexes*



- ❖ B+ tree index on  $E.age$  can be used to get qualifying tuples.
    - How selective is the condition?
    - Is the index clustered?
  - ❖ Consider the GROUP BY query
    - If many tuples have  $E.age > 10$ ,  $E.age$  index and sorting the retrieved tuples may be costly.
    - Clustered  $E.dno$  index may be better.
  - ❖ Equality queries and duplicates
    - Clustering on  $E.hobby$  helps!

```
SELECT E.dno
FROM Emp E
WHERE E.age>40
```

```
SELECT E.dno, COUNT (*)
FROM Emp E
WHERE E.age>10
GROUP BY E.dno
```

```
SELECT E.dno
FROM Emp E
WHERE E.hobby=Stamps
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

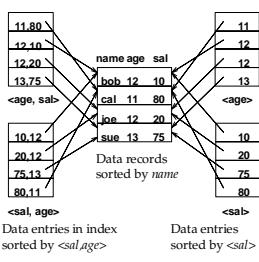
-23-

## *Indexes with Composite Search Keys*



- ❖ **Composite Search Keys:** Search on a combination of fields.
    - Equality query: Every field value is equal to a constant value. E.g. wrt  $\langle$ sal,age $\rangle$  index:
      - age=20 and sal =75
    - Range query: Some field value is not a constant. E.g.:
      - age =20; or age=20 and sal > 10
  - ❖ Data entries in index sorted by search key to support range queries.
    - Lexicographic order, or
    - Spatial order.

Examples of composite key indexes using lexicographic order.



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

24

## *Composite Search Keys*



- ❖ To retrieve Emp records with  $age=30$  AND  $sal=4000$ , an index on  $\langle age, sal \rangle$  would be better than an index on  $age$  or an index on  $sal$ .
    - Choice of index key orthogonal to clustering etc.
  - ❖ If condition is:  $20 < age < 30$  AND  $3000 < sal < 5000$ :
    - Clustered tree index on  $\langle age, sal \rangle$  or  $\langle sal, age \rangle$  is best.
  - ❖ If condition is:  $age=30$  AND  $3000 < sal < 5000$ :
    - Clustered  $\langle age, sal \rangle$  index much better than  $\langle sal, age \rangle$  index!
  - ❖ Composite indexes are larger, updated more often.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

25

## *Index-Only Plans*



- ❖ A number of queries can be answered without retrieving any tuples from one or more of the relations involved if a suitable index is available.

<i>Plans</i>	SELECT D.mgr FROM Dept D, Emp E WHERE D.dno=E.dno
$<E.dno>$	SELECT D.mgr, E.eid FROM Dept D, Emp E WHERE D.dno=E.dno
$<E.dno, E.eid>$ <i>Tree index!</i>	SELECT E.dno, COUNT(*) FROM Emp E GROUP BY E.dno
$<E.dno>$	SELECT E.dno, MIN(E.sal) FROM Emp E GROUP BY E.dno
$E.dno, E.sal>$ <i>Tree index!</i>	SELECT AVG(E.sal) FROM Emp E WHERE E.age=25 AND E.sal BETWEEN 3000 AND 5000
$E.e>$ $E.age>$ <i>Tree!</i>	

Database Management Systems 2 ed., P. Ramachandran and S. Jayachandran, PHI Learning Private Limited, New Delhi, 2011.

### *Index-Only Plans (Contd.)*



- ❖ Index-only plans are possible if the key is  $\langle \text{dno}, \text{age} \rangle$  or we have a tree index with key  $\langle \text{age}, \text{dno} \rangle$ 
    - Which is better?
    - What if we consider the second query?

```
SELECT E.dno, COUNT (*)
FROM Emp E
WHERE E.age=30
GROUP BY E.dno
```

```
SELECT E.dno, COUNT (*)
FROM Emp E
WHERE E.age>30
GROUP BY E.dno
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

27

## Summary



- ❖ Many alternative file organizations exist, each appropriate in some situation.
- ❖ If selection queries are frequent, sorting the file or building an *index* is important.
  - Hash-based indexes only good for equality search.
  - Sorted files and tree-based indexes best for range search; also good for equality search. (Files rarely kept sorted in practice; B+ tree index is better.)
- ❖ Index is a collection of data entries plus a way to quickly find entries with given key values.

---

---

---

---

---

---

---

## Summary (Contd.)



- ❖ Data entries can be actual data records, <key, rid> pairs, or <key, rid-list> pairs.
  - Choice orthogonal to *indexing technique* used to locate data entries with a given key value.
- ❖ Can have several indexes on a given file of data records, each with a different search key.
- ❖ Indexes can be classified as clustered vs. unclustered, primary vs. secondary, and dense vs. sparse. Differences have important consequences for utility/performance.

---

---

---

---

---

---

---

## Summary (Contd.)



- ❖ Understanding the nature of the *workload* for the application, and the performance goals, is essential to developing a good design.
  - What are the important queries and updates? What attributes/relations are involved?
- ❖ Indexes must be chosen to speed up important queries (and perhaps some updates!).
  - Index maintenance overhead on updates to key fields.
  - Choose indexes that can help many queries, if possible.
  - Build indexes to support index-only strategies.
  - Clustering is an important decision; only one index on a given relation can be clustered!
  - Order of fields in composite index key can be important.

---

---

---

---

---

---

---



## Storing Data: Disks and Files

### Chapter 7

"Yea, from the table of my memory  
I'll wipe away all trivial fond records."  
— Shakespeare, *Hamlet*

---

---

---

---

---

---



## Disks and Files

- ❖ DBMS stores information on (“hard”) disks.
- ❖ This has major implications for DBMS design!
  - READ: transfer data from disk to main memory (RAM).
  - WRITE: transfer data from RAM to disk.
  - Both are high-cost operations, relative to in-memory operations, so must be planned carefully!

---

---

---

---

---

---



## Why Not Store Everything in Main Memory

- ❖ *Costs too much.* \$1000 will buy you either 128MB of RAM or 7.5GB of disk today.
- ❖ *Main memory is volatile.* We want data to be saved between runs. (Obviously!)
- ❖ Typical storage hierarchy:
  - Main memory (RAM) for currently used data.
  - Disk for the main database (secondary storage).
  - Tapes for archiving older versions of the data (tertiary storage).

---

---

---

---

---

---

## Disks



- ❖ Secondary storage device of choice.
- ❖ Main advantage over tapes: random access vs. sequential.
- ❖ Data is stored and retrieved in units called *disk blocks* or *pages*.
- ❖ Unlike RAM, time to retrieve a disk page varies depending upon location on disk.
  - Therefore, relative placement of pages on disk has major impact on DBMS performance!

---

---

---

---

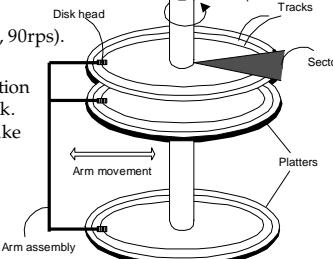
---

---

## Components of a Disk



- ❖ The platters spin (say, 90rps).
- ❖ The arm assembly is moved in or out to position a head on a desired track. Tracks under heads make a *cylinder* (imaginary!).
- ❖ Only one head reads/writes at any one time.
- ❖ Block size is a multiple of sector size (which is fixed).



---

---

---

---

---

---

## Accessing a Disk Page



- ❖ Time to access (read/write) a disk block:
  - *seek time* (moving arms to position disk head on track)
  - *rotational delay* (waiting for block to rotate under head)
  - *transfer time* (actually moving data to/from disk surface)
- ❖ Seek time and rotational delay dominate.
  - Seek time varies from about 1 to 20msec
  - Rotational delay varies from 0 to 10msec
  - Transfer rate is about 1msec per 4KB page
- ❖ Key to lower I/O cost: reduce seek/rotation delays! Hardware vs. software solutions?

---

---

---

---

---

---

## Arranging Pages on Disk



- ❖ `Next' block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder
- ❖ Blocks in a file should be arranged sequentially on disk (by `next'), to minimize seek and rotational delay.
- ❖ For a sequential scan, pre-fetching several pages at a time is a big win!

---

---

---

---

---

---

---

## RAID



- ❖ Disk Array: Arrangement of several disks that gives abstraction of a single, large disk.
- ❖ Goals: Increase performance and reliability.
- ❖ Two main techniques:
  - Data striping: Data is partitioned; size of a partition is called the striping unit. Partitions are distributed over several disks.
  - Redundancy: More disks => more failures. Redundant information allows reconstruction of data if a disk fails.

---

---

---

---

---

---

---

## RAID Levels



- ❖ Level 0: No redundancy
- ❖ Level 1: Mirrored (two identical copies)
  - Each disk has a mirror image (check disk)
  - Parallel reads, a write involves two disks.
  - Maximum transfer rate = transfer rate of one disk
- ❖ Level 0+1: Striping and Mirroring
  - Parallel reads, a write involves two disks.
  - Maximum transfer rate = aggregate bandwidth

---

---

---

---

---

---

---

## RAID Levels (Contd.)



- ❖ Level 3: Bit-Interleaved Parity
  - Striping Unit: One bit. One check disk.
  - Each read and write request involves all disks; disk array can process one request at a time.
- ❖ Level 4: Block-Interleaved Parity
  - Striping Unit: One disk block. One check disk.
  - Parallel reads possible for small requests, large requests can utilize full bandwidth
  - Writes involve modified block and check disk
- ❖ Level 5: Block-Interleaved Distributed Parity
  - Similar to RAID Level 4, but parity blocks are distributed over all disks

---

---

---

---

---

---

---

## Disk Space Management



- ❖ Lowest layer of DBMS software manages space on disk.
- ❖ Higher levels call upon this layer to:
  - allocate/de-allocate a page
  - read/write a page
- ❖ Request for a *sequence* of pages must be satisfied by allocating the pages sequentially on disk! Higher levels don't need to know how this is done, or how free space is managed.

---

---

---

---

---

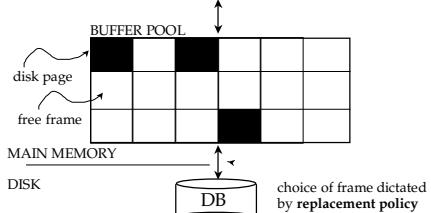
---

---

## Buffer Management in a DBMS



Page Requests from Higher Levels



- ❖ Data must be in RAM for DBMS to operate on it!
- ❖ Table of  $\langle \text{frame\#}, \text{pageid} \rangle$  pairs is maintained.

---

---

---

---

---

---

---

## When a Page is Requested ...



- ❖ If requested page is not in pool:
    - Choose a frame for *replacement*
    - If frame is dirty, write it to disk
    - Read requested page into chosen frame
  - ❖ *Pin* the page and return its address.
- \* *If requests can be predicted (e.g., sequential scans) pages can be pre-fetched several pages at a time!*

---

---

---

---

---

---

---

## More on Buffer Management



- ❖ Requestor of page must unpin it, and indicate whether page has been modified:
  - *dirty* bit is used for this.
- ❖ Page in pool may be requested many times,
  - a *pin count* is used. A page is a candidate for replacement iff *pin count* = 0.
- ❖ CC & recovery may entail additional I/O when a frame is chosen for replacement.  
(*Write-Ahead Log* protocol; more later.)

---

---

---

---

---

---

---

## Buffer Replacement Policy



- ❖ Frame is chosen for replacement by a *replacement policy*:
  - Least-recently-used (LRU), Clock, MRU etc.
- ❖ Policy can have big impact on # of I/O's; depends on the *access pattern*.
- ❖ Sequential flooding: Nasty situation caused by LRU + repeated sequential scans.
  - # buffer frames < # pages in file means each page request causes an I/O. MRU much better in this situation (but not in all situations, of course).

---

---

---

---

---

---

---

## DBMS vs. OS File System



OS does disk space & buffer mgmt: why not let OS manage these tasks?

- ❖ Differences in OS support: portability issues
- ❖ Some limitations, e.g., files can't span disks.
- ❖ Buffer management in DBMS requires ability to:
  - pin a page in buffer pool, force a page to disk (important for implementing CC & recovery),
  - adjust *replacement policy*, and pre-fetch pages based on access patterns in typical DB operations.

---

---

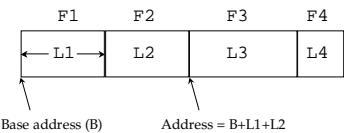
---

---

---

---

## Record Formats: Fixed Length



- ❖ Information about field types same for all records in a file; stored in *system catalogs*.
- ❖ Finding *i'th* field requires scan of record.

---

---

---

---

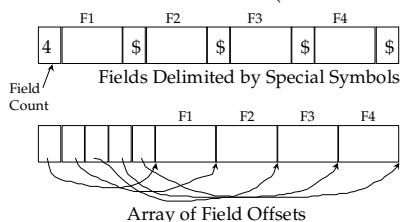
---

---

## Record Formats: Variable Length



- ❖ Two alternative formats (# fields is fixed):



\* Second offers direct access to *i'th* field, efficient storage of nulls (special *don't know* value); small directory overhead.

---

---

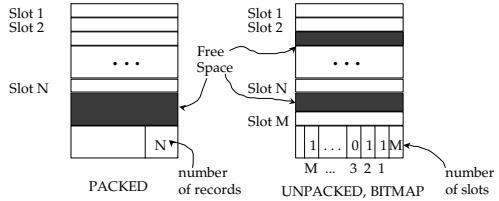
---

---

---

---

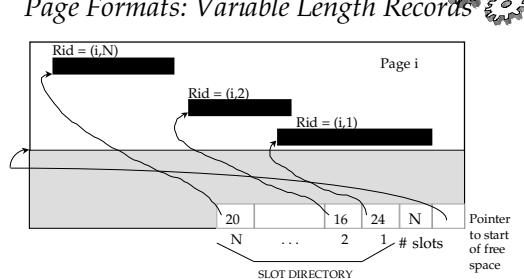
## *Page Formats: Fixed Length Records*



- \* Record id = <page id, slot #>. In first alternative, moving records for free space management changes rid; may not be acceptable.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

1



- \* Can move records on page without changing rid; so, attractive for fixed-length records too.

Database Management Systems 3ed. R. Ramakrishnan and J. Gehrke

2

## *Files of Records*

- ❖ Page or block is OK when doing I/O, but higher levels of DBMS operate on *records*, and *files of records*.
  - ❖ FILE: A collection of pages, each containing a collection of records. Must support:
    - insert/delete/modify record
    - read a particular record (specified using *record id*)
    - scan all records (possibly with some conditions on the records to be retrieved)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

2

## *Unordered (Heap) Files*



- ❖ Simplest file structure contains records in no particular order.
- ❖ As file grows and shrinks, disk pages are allocated and de-allocated.
- ❖ To support record level operations, we must:
  - keep track of the *pages* in a file
  - keep track of *free space* on pages
  - keep track of the *records* on a page
- ❖ There are many alternatives for keeping track of this.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

22

---

---

---

---

---

---

---

## *Heap File Implemented as a List*



- ❖ The header page id and Heap file name must be stored someplace.
- ❖ Each page contains 2 `pointers' plus data.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

23

---

---

---

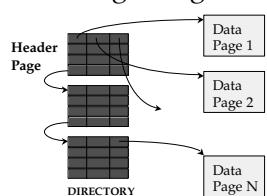
---

---

---

---

## *Heap File Using a Page Directory*



- ❖ The entry for a page can include the number of free bytes on the page.
- ❖ The directory is a collection of pages; linked list implementation is just one alternative.

▪ *Much smaller than linked list of all HF pages!*

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

24

---

---

---

---

---

---

---

## System Catalogs



- ❖ For each index:
  - structure (e.g., B+ tree) and search key fields
- ❖ For each relation:
  - name, file name, file structure (e.g., Heap file)
  - attribute name and type, for each attribute
  - index name, for each index
  - integrity constraints
- ❖ For each view:
  - view name and definition
- ❖ Plus statistics, authorization, buffer pool size, etc.
  - \* Catalogs are themselves stored as relations!

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

25

## Attr\_Cat(attr\_name, rel\_name, type, position)



attr_name	rel_name	type	position
attr_name	Attribute_Cat	string	1
rel_name	Attribute_Cat	string	2
type	Attribute_Cat	string	3
position	Attribute_Cat	integer	4
sid	Students	string	1
name	Students	string	2
login	Students	string	3
age	Students	integer	4
gpa	Students	real	5
fid	Faculty	string	1
fname	Faculty	string	2
sal	Faculty	real	3

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

26

## Summary



- ❖ Disks provide cheap, non-volatile storage.
  - Random access, but cost depends on location of page on disk; important to arrange data sequentially to minimize *seek* and *rotation* delays.
- ❖ Buffer manager brings pages into RAM.
  - Page stays in RAM until released by requestor.
  - Written to disk when frame chosen for replacement (which is sometime after requestor releases the page).
  - Choice of frame to replace based on *replacement policy*.
  - Tries to *pre-fetch* several pages at a time.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

27

## Summary (Contd.)



- ❖ DBMS vs. OS File Support
  - DBMS needs features not found in many OS's, e.g., forcing a page to disk, controlling the order of page writes to disk, files spanning disks, ability to control pre-fetching and page replacement policy based on predictable access patterns, etc.
- ❖ Variable length record format with field offset directory offers support for direct access to i'th field and null values.
- ❖ Slotted page format supports variable length records and allows records to move on page.

---

---

---

---

---

---

---

---

## Summary (Contd.)



- ❖ File layer keeps track of pages in a file, and supports abstraction of a collection of records.
  - Pages with free space identified using linked list or directory structure (similar to how pages in file are kept track of).
- ❖ Indexes support efficient retrieval of records based on the values in some fields.
- ❖ Catalog relations store information about relations, indexes and views. (*Information that is common to all records in a given collection.*)

---

---

---

---

---

---

---

---

## Tree-Structured Indexes

### Chapter 9



---

---

---

---

---

---

## Introduction



- ❖ As for any index, 3 alternatives for data entries  $k^*$ :
  - Data record with key value  $k$
  - $\langle k, \text{rid} \text{ of data record with search key } k \rangle$
  - $\langle k, \text{list of rids of data records with search key } k \rangle$
- ❖ Choice is orthogonal to the *indexing technique* used to locate data entries  $k^*$ .
- ❖ Tree-structured indexing techniques support both *range searches* and *equality searches*.
- ❖ ISAM: static structure; B+ tree: dynamic, adjusts gracefully under inserts and deletes.

---

---

---

---

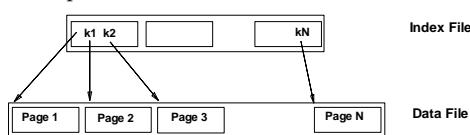
---

---

## Range Searches



- ❖ ``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.



\* Can do binary search on (smaller) index file!

---

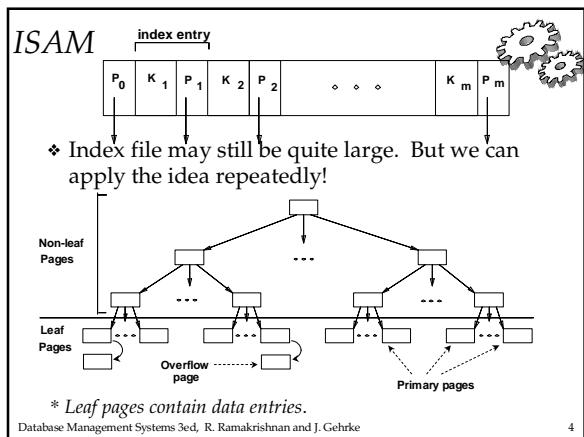
---

---

---

---

---



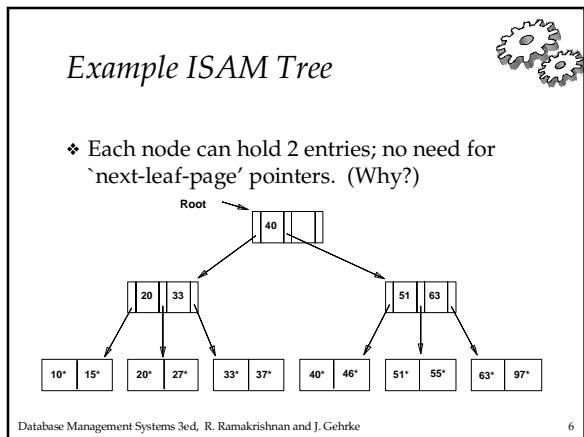
**Comments on ISAM**

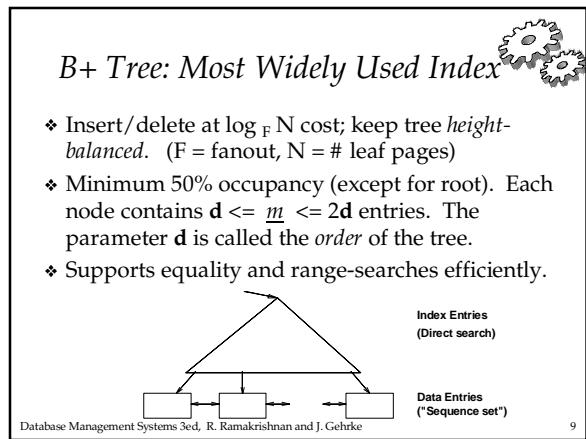
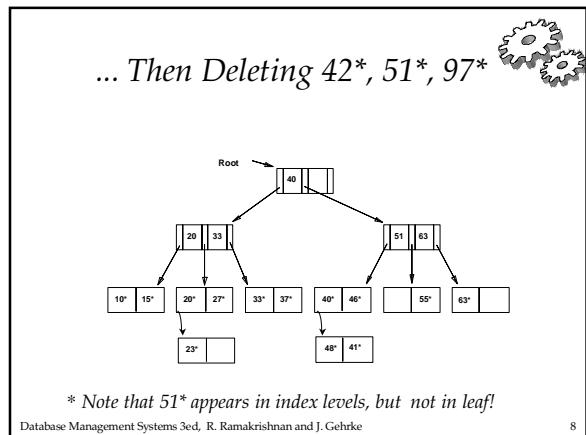
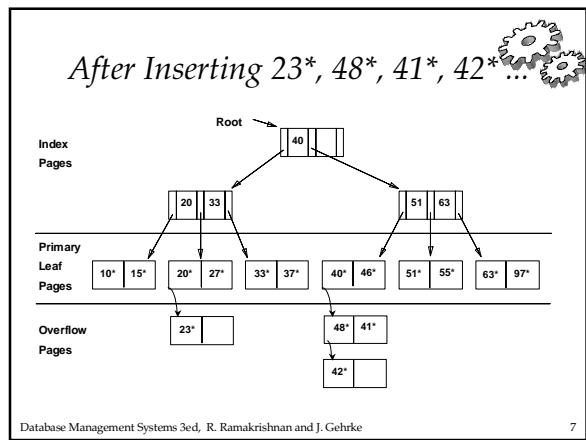
The comments on ISAM include:

- File creation: Leaf (data) pages allocated sequentially, sorted by search key; then index pages allocated, then space for overflow pages.
- Index entries: <search key value, page id>; they `direct' search for *data entries*, which are in leaf pages.
- Search: Start at root; use key comparisons to go to leaf. Cost  $\propto \log_f N$ ;  $F = \# \text{ entries/index pg}$ ,  $N = \# \text{ leaf pgs}$
- Insert: Find leaf data entry belongs to, and put it there.
- Delete: Find and remove from leaf; if empty overflow page, de-allocate.
- Static tree structure:** *inserts/deletes affect only leaf pages.*

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

5

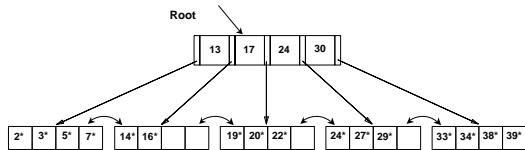




## Example B+ Tree



- ❖ Search begins at root, and key comparisons direct it to a leaf (as in ISAM).
- ❖ Search for  $5^*$ ,  $15^*$ , all data entries  $\geq 24^*$  ...



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

10

## B+ Trees in Practice



- ❖ Typical order: 100. Typical fill-factor: 67%.
  - average fanout = 133
- ❖ Typical capacities:
  - Height 4:  $133^4 = 312,900,700$  records
  - Height 3:  $133^3 = 2,352,637$  records
- ❖ Can often hold top levels in buffer pool:
  - Level 1 = 1 page = 8 Kbytes
  - Level 2 = 133 pages = 1 Mbyte
  - Level 3 = 17,689 pages = 133 MBbytes

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

11

## Inserting a Data Entry into a B+ Tree



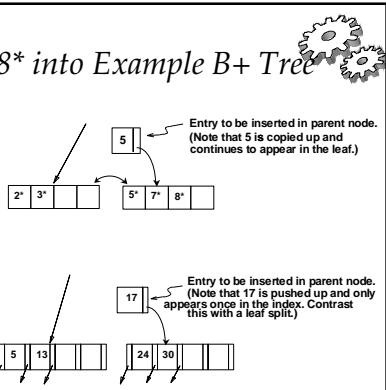
- ❖ Find correct leaf  $L$ .
- ❖ Put data entry onto  $L$ .
  - If  $L$  has enough space, done!
  - Else, must split  $L$  (into  $L$  and a new node  $L_2$ )
    - Redistribute entries evenly, copy up middle key.
    - Insert index entry pointing to  $L_2$  into parent of  $L$ .
- ❖ This can happen recursively
  - To split index node, redistribute entries evenly, but push up middle key. (Contrast with leaf splits.)
- ❖ Splits “grow” tree; root split increases height.
  - Tree growth: gets wider or one level taller at top

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

12

## Inserting 8\* into Example B+ Tree

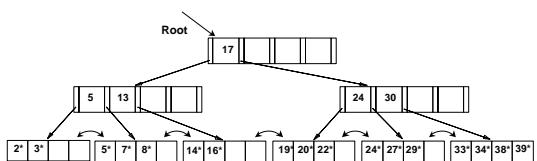
- ❖ Observe how minimum occupancy is guaranteed in both leaf and index pg splits.
  - ❖ Note difference between *copy-up* and *push-up*; be sure you understand the reasons for this.



Database Management Systems 3ed. R. Ramakrishnan and J. Gehrke

13

### *Example B+ Tree After Inserting 8*



- v Notice that root was split, leading to increase in height.
  - v In this example, we can avoid split by re-distributing entries; however, this is usually not done in practice.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

14

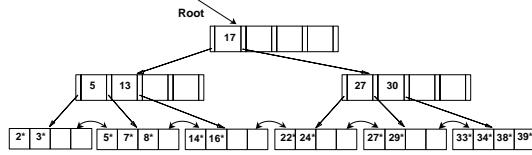
## *Deleting a Data Entry from a B+ Tree*

- ❖ Start at root, find leaf  $L$  where entry belongs.
  - ❖ Remove the entry.
    - If  $L$  is at least half-full, *done!*
    - If  $L$  has only  $d-1$  entries,
      - Try to re-distribute, borrowing from *sibling* (*adjacent node with same parent as  $L$* ).
      - If re-distribution fails, *merge*  $L$  and sibling.
  - ❖ If merge occurred, must delete entry (pointing to  $L$  or sibling) from parent of  $L$ .
  - ❖ Merge could propagate to root, decreasing height.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

15

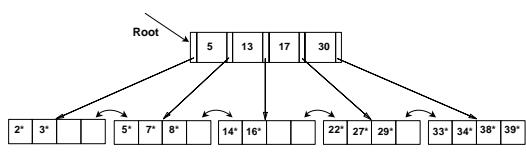
*Example Tree After (Inserting 8\*) Then) Deleting 19\* and 20\* ...*



- ❖ Deleting 19\* is easy.
- ❖ Deleting 20\* is done with re-distribution.  
Notice how middle key is *copied up*.

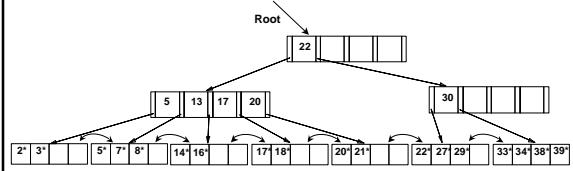
*... And Then Deleting 24\**

❖ Must merge.  
❖ Observe 'toss' of index entry (on right), and 'pull down' of index entry (below).



*Example of Non-leaf Re-distribution*

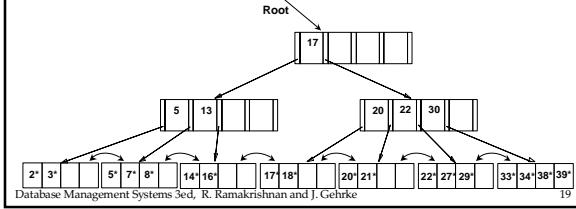
- ❖ Tree is shown below during deletion of 24\*. (What could be a possible initial tree?)
- ❖ In contrast to previous example, can re-distribute entry from left child of root to right child.



## After Re-distribution



- ♦ Intuitively, entries are re-distributed by ‘pushing through’ the splitting entry in the parent node.
- ♦ It suffices to re-distribute index entry with key 20; we’ve re-distributed 17 as well for illustration.



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

19

## Prefix Key Compression



- ♦ Important to increase fan-out. (Why?)
- ♦ Key values in index entries only ‘direct traffic’; can often compress them.
  - E.g., If we have adjacent index entries with search key values *Dannon Yogurt*, *David Smith* and *Devarakonda Murthy*, we can abbreviate *David Smith* to *Dav*. (The other keys can be compressed too ...)
    - Is this correct? Not quite! What if there is a data entry *Davey Jones*? (Can only compress *David Smith* to *Davi*)
    - In general, while compressing, must leave each index entry greater than every key value (in any subtree) to its left.
- ♦ Insert/delete must be suitably modified.

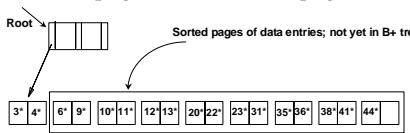
Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

20

## Bulk Loading of a B+ Tree



- ♦ If we have a large collection of records, and we want to create a B+ tree on some field, doing so by repeatedly inserting records is very slow.
- ♦ Bulk Loading can be done much more efficiently.
- ♦ Initialization: Sort all data entries, insert pointer to first (leaf) page in a new (root) page.

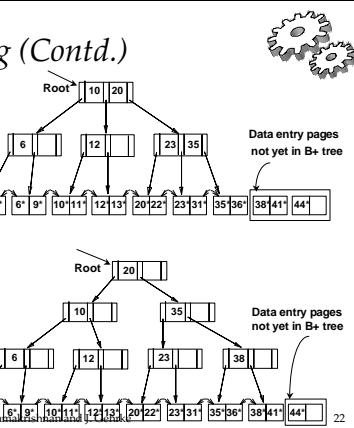


Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

21

## Bulk Loading (Contd.)

- Index entries for leaf pages always entered into right-most index page just above leaf level.  
When this fills up, it splits. (Split may go up right-most path to the root.)



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

22

## Summary of Bulk Loading

- Option 1: multiple inserts.
  - Slow.
  - Does not give sequential storage of leaves.
- Option 2: Bulk Loading
  - Has advantages for concurrency control.
  - Fewer I/Os during build.
  - Leaves will be stored sequentially (and linked, of course).
  - Can control “fill factor” on pages.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

23

## A Note on `Order'

- Order (d)** concept replaced by physical space criterion in practice ('at least half-full').
  - Index pages can typically hold many more entries than leaf pages.
  - Variable sized records and search keys mean different nodes will contain different numbers of entries.
  - Even with fixed length fields, multiple records with the same search key value (*duplicates*) can lead to variable-sized data entries (if we use Alternative (3)).

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

24

## Summary



- ❖ Tree-structured indexes are ideal for range-searches, also good for equality searches.
- ❖ ISAM is a static structure.
  - Only leaf pages modified; overflow pages needed.
  - Overflow chains can degrade performance unless size of data set and data distribution stay constant.
- ❖ B+ tree is a dynamic structure.
  - Inserts/deletes leave tree height-balanced;  $\log_F N$  cost.
  - High fanout ( $F$ ) means depth rarely more than 3 or 4.
  - Almost always better than maintaining a sorted file.

---

---

---

---

---

---

---

## Summary (Contd.)



- Typically, 67% occupancy on average.
- Usually preferable to ISAM, modulo *locking* considerations; adjusts to growth gracefully.
- If data entries are data records, splits can change rids!
- ❖ Key compression increases fanout, reduces height.
- ❖ Bulk loading can be much faster than repeated inserts for creating a B+ tree on a large data set.
- ❖ Most widely used index in database management systems because of its versatility. One of the most optimized components of a DBMS.

---

---

---

---

---

---

---



# Hash-Based Indexes

## Chapter 10

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

1

---



---



---



---



---



---



---



---



---



---



## Introduction

- ❖ As for any index, 3 alternatives for data entries  $k^*$ :
  - Data record with key value  $k$
  - $\langle k, \text{rid of data record with search key value } k \rangle$
  - $\langle k, \text{list of rids of data records with search key } k \rangle$
  - Choice orthogonal to the *indexing technique*
- ❖ Hash-based indexes are best for *equality selections*.  
Cannot support range searches.
- ❖ Static and dynamic hashing techniques exist; trade-offs similar to ISAM vs. B+ trees.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

2

---



---



---



---



---



---



---



---



---

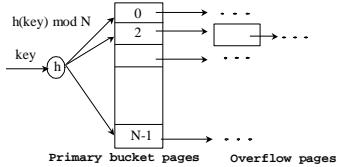


---



## Static Hashing

- ❖ # primary pages fixed, allocated sequentially, never de-allocated; overflow pages if needed.
- ❖  $h(k) \bmod M$  = bucket to which data entry with key  $k$  belongs. ( $M = \# \text{ of buckets}$ )



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

3

---



---



---



---



---



---



---



---



---



---

## *Static Hashing (Contd.)*



- ❖ Buckets contain *data entries*.
  - ❖ Hash fn works on *search key* field of record  $r$ . Must distribute values over range 0 ... M-1.
    - $h(key) = (a * key + b)$  usually works well.
    - a and b are constants; lots known about how to tune  $h$ .
  - ❖ Long overflow chains can develop and degrade performance.
    - *Extendible and Linear Hashing*: Dynamic techniques to fix this problem.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

4

Extendible Hashing

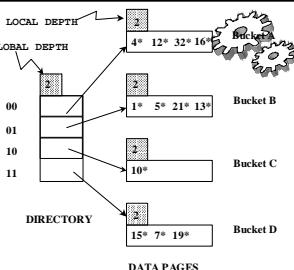


- ❖ Situation: Bucket (primary page) becomes full.  
Why not re-organize file by *doubling* # of buckets?
    - Reading and writing all pages is expensive!
    - Idea: Use *directory of pointers to buckets*, double # of buckets by *doubling the directory*, splitting just the bucket that overflowed!
    - Directory much smaller than file, so doubling it is much cheaper. Only one page of data entries is split. *No overflow page!*
    - Trick lies in how hash function is adjusted!

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

5

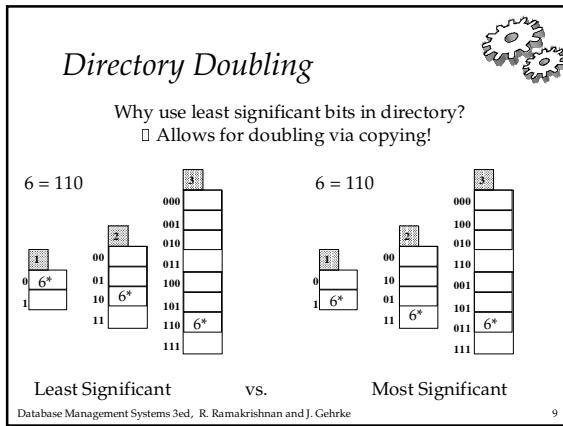
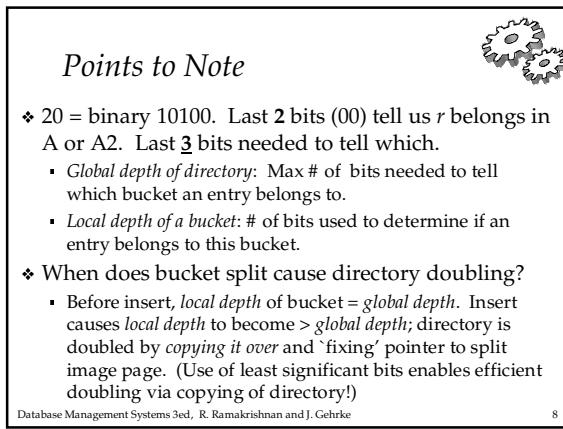
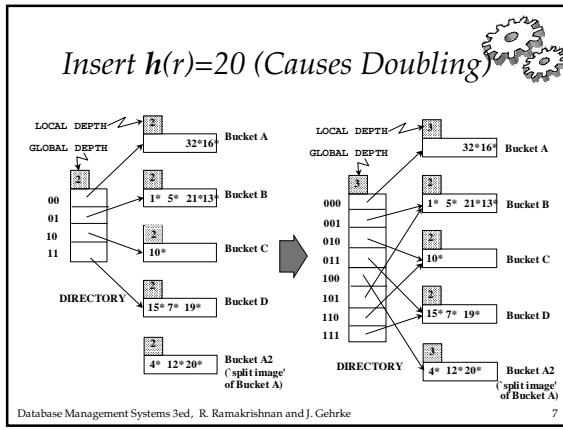
### *Example*



- ❖ Directory is array of size 4.
  - ❖ To find bucket for  $r$ , take last `global depth' # bits of  $h(r)$ ; we denote  $r$  by  $h(r)$ .
    - If  $h(r) = 5 = \text{binary } 101$ , it is in bucket pointed to by 01.
  - ❖ Insert: If bucket is full, *split* it (*allocate new page, re-distribute*).
  - ❖ If necessary, double the directory. (As we will see, splitting a bucket does not always require doubling; we can tell by comparing *global depth* with *local depth* for the split bucket.)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

6



## Comments on Extendible Hashing

- ❖ If directory fits in memory, equality search answered with one disk access; else two.
  - 100MB file, 100 bytes/rec, 4K pages contains 1,000,000 records (as data entries) and 25,000 directory elements; chances are high that directory will fit in memory.
  - Directory grows in spurts, and, if the distribution of *hash values* is skewed, directory can grow large.
  - Multiple entries with same hash value cause problems!
- ❖ **Delete:** If removal of data entry makes bucket empty, can be merged with 'split image'. If each directory element points to same bucket as its split image, can halve directory.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

10



---

---

---

---

---

---

---

---

## Linear Hashing



- ❖ This is another dynamic hashing scheme, an alternative to Extendible Hashing.
- ❖ LH handles the problem of long overflow chains without using a directory, and handles duplicates.
- ❖ Idea: Use a family of hash functions  $h_0, h_1, h_2, \dots$ 
  - $h_i(key) = h(key) \text{ mod } (2^i N)$ ;  $N$  = initial # buckets
  - $h$  is some hash function (range is *not* 0 to  $N-1$ )
  - If  $N = 2^{d0}$ , for some  $d0$ ,  $h_i$  consists of applying  $h$  and looking at the last  $di$  bits, where  $di = d0 + i$ .
  - $h_{i+1}$  doubles the range of  $h_i$  (similar to directory doubling)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

11

---

---

---

---

---

---

---

---

## Linear Hashing (Contd.)



- ❖ Directory avoided in LH by using overflow pages, and choosing bucket to split round-robin.
  - Splitting proceeds in 'rounds'. Round ends when all  $N_R$  initial (for round  $R$ ) buckets are split. Buckets 0 to  $Next-1$  have been split;  $Next$  to  $N_R$  yet to be split.
  - Current round number is *Level*.
  - **Search:** To find bucket for data entry  $r$ , find  $h_{Level}(r)$ :
    - If  $h_{Level}(r)$  in range ' $Next$  to  $N_R$ ',  $r$  belongs here.
    - Else,  $r$  could belong to bucket  $h_{Level}(r)$  or bucket  $h_{Level}(r) + N_R$ ; must apply  $h_{Level+1}(r)$  to find out.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

12

---

---

---

---

---

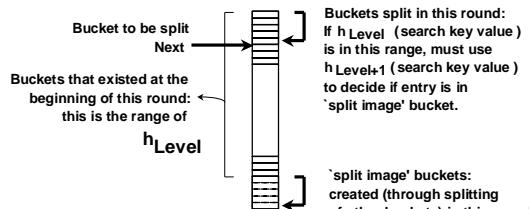
---

---

---

## Overview of LH File

- In the middle of a round.



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

13

## Linear Hashing (Contd.)

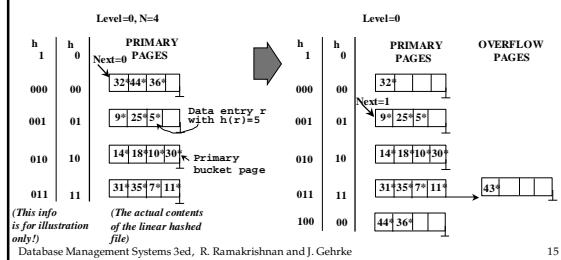
- Insert:** Find bucket by applying  $h_{Level} / h_{Level+1}$ :
  - If bucket to insert into is full:
    - Add overflow page and insert data entry.
    - (Maybe) Split Next bucket and increment Next.
- Can choose any criterion to 'trigger' split.
- Since buckets are split round-robin, long overflow chains don't develop!
- Doubling of directory in Extendible Hashing is similar; switching of hash functions is *implicit* in how the # of bits examined is increased.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

14

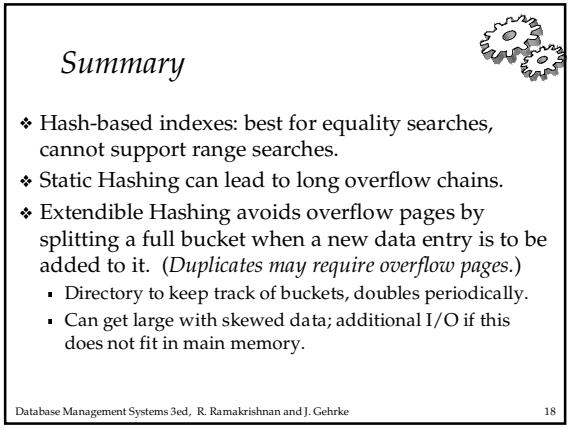
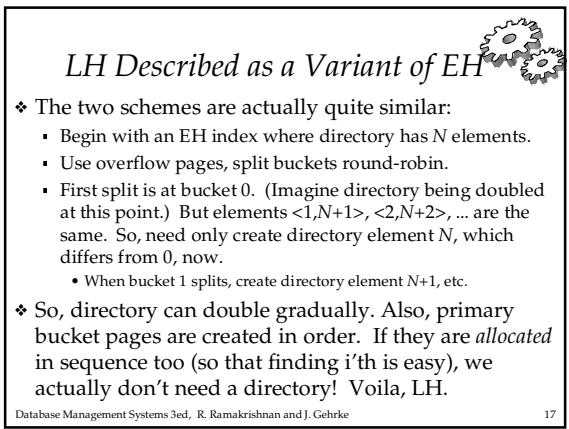
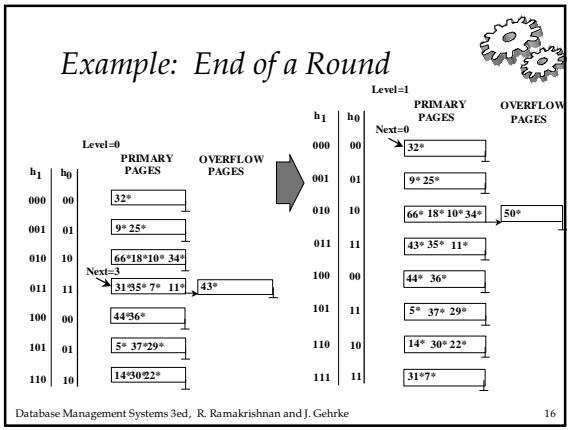
## Example of Linear Hashing

- On split,  $h_{Level+1}$  is used to re-distribute entries.



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

15



## Summary (Contd.)



- ❖ Linear Hashing avoids directory by splitting buckets round-robin, and using overflow pages.
  - Overflow pages not likely to be long.
  - Duplicates handled easily.
  - Space utilization could be lower than Extendible Hashing, since splits not concentrated on 'dense' data areas.
    - Can tune criterion for triggering splits to trade-off slightly longer chains for better space utilization.
- ❖ For hash-based indexes, a *skewed* data distribution is one in which the *hash values* of data entries are not uniformly distributed!



## *Overview of Query Evaluation*

### Chapter 12

---

---

---

---

---

---



## *Overview of Query Evaluation*

- ❖ Plan: Tree of R.A. ops, with choice of alg for each op.
  - Each operator typically implemented using a ‘pull’ interface: when an operator is ‘pulled’ for the next output tuples, it ‘pulls’ on its inputs and computes them.
- ❖ Two main issues in query optimization:
  - 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!
- ❖ We will study the System R approach.

---

---

---

---

---

---



## *Some Common Techniques*

- ❖ Algorithms for evaluating relational operators use some simple ideas extensively:
  - Indexing: Can use WHERE conditions to retrieve small set of tuples (selections, joins)
  - Iteration: Sometimes, faster to scan all tuples even if there is an index. (And sometimes, we can scan the data entries in an index instead of the table itself.)
  - Partitioning: By using sorting or hashing, we can partition the input tuples and replace an expensive operation by similar operations on smaller inputs.

\* Watch for these techniques as we discuss query evaluation!

---

---

---

---

---

---

## Statistics and Catalogs



- ❖ Need information about the relations and indexes involved. *Catalogs* typically contain at least:
  - # tuples (NTuples) and # pages (NPages) for each relation.
  - # distinct key values (NKeys) and NPages for each index.
  - Index height, low/high key values (Low / High) for each tree index.
- ❖ Catalogs updated periodically.
  - Updating whenever data changes is too expensive; lots of approximation anyway, so slight inconsistency ok.
- ❖ More detailed information (e.g., histograms of the values in some field) are sometimes stored.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

4

---

---

---

---

---

---

---

---

## Access Paths



- ❖ An access path is a method of retrieving tuples:
  - File scan, or index that matches a selection (in the query)
- ❖ A tree index matches (a conjunction of) terms that involve only attributes in a *prefix* of the search key.
  - E.g., Tree index on  $\langle a, b, c \rangle$  matches the selection  $a=5 \text{ AND } b=3$ , and  $a=5 \text{ AND } b>6$ , but not  $b=3$ .
- ❖ A hash index matches (a conjunction of) terms that has a term *attribute = value* for every attribute in the search key of the index.
  - E.g., Hash index on  $\langle a, b, c \rangle$  matches  $a=5 \text{ AND } b=3 \text{ AND } c=5$ ; but it does not match  $b=3$ , or  $a=5 \text{ AND } b=3$ , or  $a>5 \text{ AND } b=3 \text{ AND } c=5$ .

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

5

---

---

---

---

---

---

---

---

## A Note on Complex Selections



`(day<8/9/94 AND rname='Paul') OR bid=5 OR sid=3`

- ❖ Selection conditions are first converted to conjunctive normal form (CNF):  
$$(day < 8/9/94 \text{ OR } bid = 5 \text{ OR } sid = 3) \text{ AND } (rname = 'Paul' \text{ OR } bid = 5 \text{ OR } sid = 3)$$
- ❖ We only discuss case with no ORs; see text if you are curious about the general case.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

6

---

---

---

---

---

---

---

---

## One Approach to Selections



- ❖ Find the *most selective access path*, retrieve tuples using it, and apply any remaining terms that don't match the index:
  - *Most selective access path:* An index or file scan that we estimate will require the fewest page I/Os.
  - Terms that match this index reduce the number of tuples *retrieved*; other terms are used to discard some retrieved tuples, but do not affect number of tuples/pages fetched.
  - Consider  $\text{day} < 8/9/94 \text{ AND } \text{bid} = 5 \text{ AND } \text{sid} = 3$ . A B+ tree index on *day* can be used; then, *bid=5* and *sid=3* must be checked for each retrieved tuple. Similarly, a hash index on  $\langle \text{bid}, \text{sid} \rangle$  could be used; *day < 8/9/94* must then be checked.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

7

---

---

---

---

---

---

---

## Using an Index for Selections



- ❖ Cost depends on #qualifying tuples, and clustering.
  - Cost of finding qualifying data entries (typically small) plus cost of retrieving records (could be large w/o clustering).
  - In example, assuming uniform distribution of names, about 10% of tuples qualify (100 pages, 10000 tuples). With a clustered index, cost is little more than 100 I/Os; if unclustered, upto 10000 I/Os!

```
SELECT *
FROM Reserves R
WHERE R.name < 'C%'
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

8

---

---

---

---

---

---

---

## Projection

```
SELECT DISTINCT
 R.sid, R.bid
 FROM Reserves R
```



- ❖ The expensive part is removing duplicates.
  - SQL systems don't remove duplicates unless the keyword DISTINCT is specified in a query.
- ❖ Sorting Approach: Sort on  $\langle \text{sid}, \text{bid} \rangle$  and remove duplicates. (Can optimize this by dropping unwanted information while sorting.)
- ❖ Hashing Approach: Hash on  $\langle \text{sid}, \text{bid} \rangle$  to create partitions. Load partitions into memory one at a time, build in-memory hash structure, and eliminate duplicates.
- ❖ If there is an index with both R.sid and R.bid in the search key, may be cheaper to sort data entries!

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

9

---

---

---

---

---

---

---

## Join: Index Nested Loops

```
foreach tuple r in R do
 foreach tuple s in S where ri == sj do
 add <r, s> to result
```

- ❖ If there is an index on the join column of one relation (say S), can make it the inner and exploit the index.
  - Cost:  $M + (M^*p_R) * \text{cost of finding matching } S \text{ tuples}$
- ❖ For each R tuple, cost of probing S index is about 1.2 for hash index, 2-4 for B+ tree. Cost of then finding S tuples (assuming Alt. (2) or (3) for data entries) depends on clustering.
  - Clustered index: 1 I/O (typical), unclustered: upto 1 I/O per matching S tuple.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

10

## Examples of Index Nested Loops

- ❖ Hash-index (Alt. 2) on *sid* of Sailors (as inner):
  - Scan Reserves: 1000 page I/Os, 100\*1000 tuples.
  - For each Reserves tuple: 1.2 I/Os to get data entry in index, plus 1 I/O to get (the exactly one) matching Sailors tuple. Total: 220,000 I/Os.
- ❖ Hash-index (Alt. 2) on *sid* of Reserves (as inner):
  - Scan Sailors: 500 page I/Os, 80\*500 tuples.
  - For each Sailors tuple: 1.2 I/Os to find index page with data entries, plus cost of retrieving matching Reserves tuples. Assuming uniform distribution, 2.5 reservations per sailor (100,000 / 40,000). Cost of retrieving them is 1 or 2.5 I/Os depending on whether the index is clustered.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

11

## Join: Sort-Merge ( $R \sqcup\!\!\sqcup_{i=j} S$ )

- ❖ Sort R and S on the join column, then scan them to do a ``merge'' (on join col.), and output result tuples.
  - Advance scan of R until current R-tuple  $\geq$  current S tuple, then advance scan of S until current S-tuple  $\geq$  current R tuple; do this until current R tuple = current S tuple.
  - At this point, all R tuples with same value in  $R_i$  (*current R group*) and all S tuples with same value in  $S_j$  (*current S group*) match; output  $\langle r, s \rangle$  for all pairs of such tuples.
  - Then resume scanning R and S.
- ❖ R is scanned once; each S group is scanned once per matching R tuple. (Multiple scans of an S group are likely to find needed pages in buffer.)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

12

## Example of Sort-Merge Join

sid	sname	rating	age	sid	bid	day	rname
22	dustin	7	45.0	28	103	12/4/96	guppy
28	yuppy	9	35.0	28	103	11/3/96	yuppy
31	lubber	8	55.5	31	101	10/10/96	dustin
44	guppy	5	35.0	31	102	10/12/96	lubber
58	rusty	10	35.0	31	101	10/11/96	lubber
				58	103	11/12/96	dustin

- ❖ Cost:  $M \log M + N \log N + (M+N)$ 
  - The cost of scanning,  $M+N$ , could be  $M^N$  (very unlikely!)
- ❖ With 35, 100 or 300 buffer pages, both Reserves and Sailors can be sorted in 2 passes; total join cost: 7500.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

13

## Highlights of System R Optimizer

- ❖ Impact:
  - Most widely used currently; works well for < 10 joins.
- ❖ Cost estimation: Approximate art at best.
  - Statistics, maintained in system catalogs, used to estimate cost of operations and result sizes.
  - Considers combination of CPU and I/O costs.
- ❖ Plan Space: Too large, must be pruned.
  - Only the space of *left-deep plans* is considered.
    - Left-deep plans allow output of each operator to be *pipelined* into the next operator without storing it in a temporary relation.
  - Cartesian products avoided.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

14

## Cost Estimation

- ❖ For each plan considered, must estimate cost:
  - Must estimate *cost* of each operation in plan tree.
    - Depends on input cardinalities.
    - We've already discussed how to estimate the cost of operations (sequential scan, index scan, joins, etc.)
  - Must also estimate *size of result* for each operation in tree!
    - Use information about the input relations.
    - For selections and joins, assume independence of predicates.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

15

## Size Estimation and Reduction Factors

```

SELECT attribute list
FROM relation list
WHERE term1 AND ... AND termk

```

- ❖ Consider a query block:
- ❖ Maximum # tuples in result is the product of the cardinalities of relations in the FROM clause.
- ❖ *Reduction factor (RF)* associated with each *term* reflects the impact of the *term* in reducing result size. *Result cardinality* = Max # tuples \* product of all RF's.
  - Implicit assumption that *terms* are independent!
  - Term *col=value* has RF  $1/N\text{Keys}(I)$ , given index I on *col*
  - Term *col1=col2* has RF  $1/\text{MAX}(N\text{Keys}(I1), N\text{Keys}(I2))$
  - Term *col>value* has RF  $(\text{High}(I)-\text{value})/(\text{High}(I)-\text{Low}(I))$

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

16

## Schema for Examples

Sailors (*sid: integer, sname: string, rating: integer, age: real*)  
Reserves (*sid: integer, bid: integer, day: dates, rname: string*)

- ❖ Similar to old schema; *rname* added for variations.
- ❖ Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
- ❖ Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

17

## Motivating Example

```

SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND
R.bid=100 AND S.rating>5

```

- ❖ Cost:  $500+500*1000$  I/Os
- ❖ By no means the worst plan! Plan:
  - $\pi_{sname}$  (On-the-fly)
  - $\sigma_{bid=100 \wedge rating > 5}$  (On-the-fly)
  - $\bowtie_{sid=sid}$  (Simple Nested Loops)
- ❖ Misses several opportunities:
  - selections could have been 'pushed' earlier, no use is made of any available indexes, etc.
- ❖ *Goal of optimization:* To find more efficient plans that compute the same answer.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

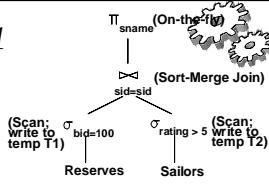
18

## Alternative Plans 1 (No Indexes)

- ❖ **Main difference:** push selects.
- ❖ With 5 buffers, cost of plan:
  - Scan Reserves (1000) + write temp T1 (10 pages, if we have 100 boats, uniform distribution).
  - Scan Sailors (500) + write temp T2 (250 pages, if we have 10 ratings).
  - Sort T1 ( $2^2 \cdot 10$ ), sort T2 ( $2^3 \cdot 250$ ), merge ( $10+250$ )
  - Total: 3560 page I/Os.
- ❖ If we used BNL join, join cost =  $10+4 \cdot 250$ , total cost = 2770.
- ❖ If we 'push' projections, T1 has only *sid*, T2 only *sid* and *sname*:
  - T1 fits in 3 pages, cost of BNL drops to under 250 pages, total < 2000.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

19

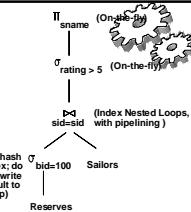


## Alternative Plans 2 With Indexes

- ❖ With clustered index on *bid* of Reserves, we get  $100,000/100 = 1000$  tuples on  $1000/100 = 10$  pages.
- ❖ INL with pipelining (outer is not materialized).
  - Projecting out unnecessary fields from outer doesn't help.
- ❖ Join column *sid* is a key for Sailors.
  - At most one matching tuple, unclustered index on *sid* OK.
- ❖ Decision not to push *rating>5* before the join is based on availability of *sid* index on Sailors.
- ❖ Cost: Selection of Reserves tuples (10 I/Os); for each, must get matching Sailors tuple ( $1000 \cdot 1.2$ ); total 1210 I/Os.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

20



## Summary

- ❖ There are several alternative evaluation algorithms for each relational operator.
- ❖ A query is evaluated by converting it to a tree of operators and evaluating the operators in the tree.
- ❖ Must understand query optimization in order to fully understand the performance impact of a given database design (relations, indexes) on a workload (set of queries).
- ❖ Two parts to optimizing a query:
  - Consider a set of alternative plans.
    - Must prune search space; typically, left-deep plans only.
  - Must estimate cost of each plan that is considered.
    - Must estimate size of result and cost for each plan node.
    - *Key issues:* Statistics, indexes, operator implementations.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

21

# *External Sorting*

## Chapter 13

---

---

---

---

---

---

## *Why Sort?*

- ❖ A classic problem in computer science!
- ❖ Data requested in sorted order
  - e.g., find students in increasing *gpa* order
- ❖ Sorting is first step in *bulk loading* B+ tree index.
- ❖ Sorting useful for eliminating *duplicate copies* in a collection of records (Why?)
- ❖ *Sort-merge* join algorithm involves sorting.
- ❖ Problem: sort 1Gb of data with 1Mb of RAM.
  - why not virtual memory?

---

---

---

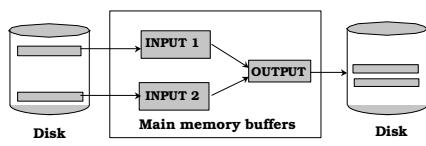
---

---

---

## *2-Way Sort: Requires 3 Buffers*

- ❖ Pass 1: Read a page, sort it, write it.
  - only one buffer page is used
- ❖ Pass 2, 3, ..., etc.:
  - three buffer pages used.



---

---

---

---

---

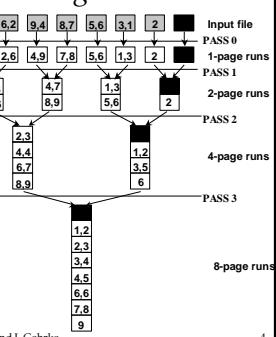
---

## Two-Way External Merge Sort

- Each pass we read + write each page in file.
- $N$  pages in the file  $\Rightarrow$  the number of passes  $= \lceil \log_2 N \rceil + 1$
- So total cost is:  $2N(\lceil \log_2 N \rceil + 1)$
- Idea: Divide and conquer:** sort subfiles and merge

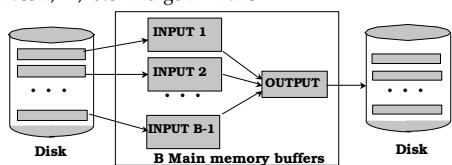
Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

4



## General External Merge Sort

- \* More than 3 buffer pages. How can we utilize them?
- To sort a file with  $N$  pages using  $B$  buffer pages:
  - Pass 0: use  $B$  buffer pages. Produce  $\lceil N / B \rceil$  sorted runs of  $B$  pages each.
  - Pass 2, ..., etc.: merge  $B-1$  runs.



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

5

## Cost of External Merge Sort

- Number of passes:  $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
- Cost =  $2N * (\# \text{ of passes})$
- E.g., with 5 buffer pages, to sort 108 page file:
  - Pass 0:  $\lceil 108 / 5 \rceil = 22$  sorted runs of 5 pages each (last run is only 3 pages)
  - Pass 1:  $\lceil 22 / 4 \rceil = 6$  sorted runs of 20 pages each (last run is only 8 pages)
  - Pass 2: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

6

## Number of Passes of External Sort

N	B=3	B=5	B=9	B=17	B=129	B=257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

7

## Internal Sort Algorithm

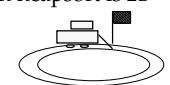
- ❖ Quicksort is a fast way to sort in memory.
- ❖ An alternative is “tournament sort” (a.k.a. “heapsort”)
  - **Top:** Read in  $B$  blocks
  - **Output:** move smallest record to output buffer
  - Read in a new record  $r$
  - insert  $r$  into “heap”
  - if  $r$  not smallest, then **GOTO Output**
  - else remove  $r$  from “heap”
  - output “heap” in order; **GOTO Top**

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

8

## More on Heapsort

- ❖ Fact: average length of a run in heapsort is  $2B$ 
  - The “snowplow” analogy
- ❖ Worst-Case:
  - What is min length of a run?
  - How does this arise?
- ❖ Best-Case:
  - What is max length of a run?
  - How does this arise?
- ❖ Quicksort is faster, but ...



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

9

## I/O for External Merge Sort

- ❖ ... longer runs often means fewer passes!
- ❖ Actually, do I/O a page at a time
- ❖ In fact, read a block of pages sequentially!
- ❖ Suggests we should make each buffer (input/output) be a block of pages.
  - But this will reduce fan-out during merge passes!
  - In practice, most files still sorted in 2-3 passes.

---

---

---

---

---

---

---

---

## Number of Passes of Optimized Sort

N	B=1,000	B=5,000	B=10,000
100	1	1	1
1,000	1	1	1
10,000	2	2	1
100,000	3	2	2
1,000,000	3	2	2
10,000,000	4	3	3
100,000,000	5	3	3
1,000,000,000	5	4	3

\* Block size = 32, initial pass produces runs of size  $2B$ .

---

---

---

---

---

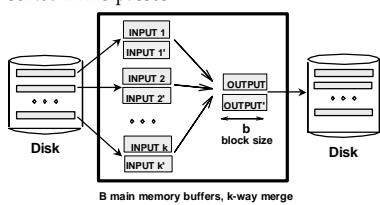
---

---

---

## Double Buffering

- ❖ To reduce wait time for I/O request to complete, can prefetch into 'shadow block'.
  - Potentially, more passes; in practice, most files still sorted in 2-3 passes.



---

---

---

---

---

---

---

---

## Sorting Records!

- ❖ Sorting has become a blood sport!
  - Parallel sorting is the name of the game ...
- ❖ Datamation: Sort 1M records of size 100 bytes
  - Typical DBMS: 15 minutes
  - World record: **3.5 seconds**
    - 12-CPU SGI machine, 96 disks, 2GB of RAM
- ❖ New benchmarks proposed:
  - Minute Sort: How many can you sort in 1 minute?
  - Dollar Sort: How many can you sort for \$1.00?

---

---

---

---

---

---

---

## Using B+ Trees for Sorting

- ❖ Scenario: Table to be sorted has B+ tree index on sorting column(s).
- ❖ Idea: Can retrieve records in order by traversing leaf pages.
- ❖ *Is this a good idea?*
- ❖ Cases to consider:
  - B+ tree is clustered      *Good idea!*
  - B+ tree is not clustered      *Could be a very bad idea!*

---

---

---

---

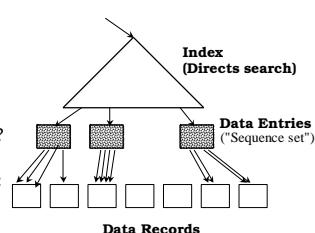
---

---

---

## Clustered B+ Tree Used for Sorting

- ❖ Cost: root to the left-most leaf, then retrieve all leaf pages (Alternative 1)
- ❖ If Alternative 2 is used? Additional cost of retrieving data records: each page fetched just once.



\* *Always better than external sorting!*

---

---

---

---

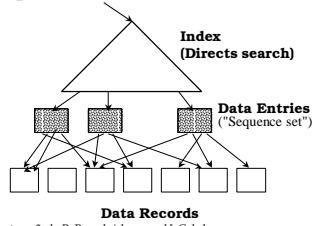
---

---

---

## Unclustered B+ Tree Used for Sorting

- ❖ Alternative (2) for data entries; each data entry contains *rid* of a data record. In general, one I/O per data record!



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

16

---

---

---

---

---

---

---

---

## External Sorting vs. Unclustered Index

N	Sorting	p=1	p=10	p=100
100	200	100	1,000	10,000
1,000	2,000	1,000	10,000	100,000
10,000	40,000	10,000	100,000	1,000,000
100,000	600,000	100,000	1,000,000	10,000,000
1,000,000	8,000,000	1,000,000	10,000,000	100,000,000
10,000,000	80,000,000	10,000,000	100,000,000	1,000,000,000

\* *p*: # of records per page  
\* *B*=1,000 and block size=32 for sorting  
\* *p*=100 is the more realistic value.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

17

---

---

---

---

---

---

---

---

## Summary

- ❖ External sorting is important; DBMS may dedicate part of buffer pool for sorting!
- ❖ External merge sort minimizes disk I/O cost:
  - Pass 0: Produces sorted *runs* of size *B* (# buffer pages). Later passes: *merge* runs.
  - # of runs merged at a time depends on *B*, and *block size*.
  - Larger block size means less I/O cost per page.
  - Larger block size means smaller # runs merged.
  - In practice, # of runs rarely more than 2 or 3.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

18

---

---

---

---

---

---

---

---



## *Summary, cont.*

- ❖ Choice of internal sort algorithm may matter:
  - Quicksort: Quick!
  - Heap/tournament sort: slower (2x), longer runs
- ❖ The best sorts are wildly fast:
  - Despite 40+ years of research, we're still improving!
- ❖ Clustered B+ tree is good for sorting;  
unclustered tree is usually very bad.

---

---

---

---

---

---

---



## *Evaluation of Relational Operations*

### Chapter 14, Part A (Joins)

---

---

---

---

---

---



## *Relational Operations*

- ❖ We will consider how to implement:
  - Selection ( $\sigma$ ) Selects a subset of rows from relation.
  - Projection ( $\pi$ ) Deletes unwanted columns from relation.
  - Join ( $\sqcup\sqcup$ ) Allows us to combine two relations.
  - Set-difference ( $-$ ) Tuples in reln. 1, but not in reln. 2.
  - Union ( $\sqcup$ ) Tuples in reln. 1 and in reln. 2.
  - Aggregation (SUM, MIN, etc.) and GROUP BY
- ❖ Since each op returns a relation, ops can be *composed*!  
After we cover the operations, we will discuss how to *optimize* queries formed by composing them.

---

---

---

---

---

---



## *Schema for Examples*

Sailors (sid: integer, sname: string, rating: integer, age: real)  
Reserves (sid: integer, bid: integer, day: dates, rname: string)

- ❖ Similar to old schema; *rname* added for variations.
- ❖ Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
- ❖ Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.

---

---

---

---

---

---

## Equality Joins With One Join Column

```
SELECT *
 FROM Reserves R1, Sailors S1
 WHERE R1.sid=S1.sid
```

- ❖ In algebra:  $R \sqcup S$ . Common! Must be carefully optimized.  $R \times S$  is large; so,  $\times S$  followed by a selection is inefficient.
- ❖ Assume: M tuples in R,  $p_R$  tuples per page, N tuples in S,  $p_S$  tuples per page.
  - In our examples, R is Reserves and S is Sailors.
- ❖ We will consider more complex join conditions later.
- ❖ *Cost metric*: # of I/Os. We will ignore output costs.

---

---

---

---

---

---

---

## Simple Nested Loops Join

```
foreach tuple r in R do
 foreach tuple s in S do
 if $r_i == s_j$ then add $\langle r, s \rangle$ to result
```

- ❖ For each tuple in the *outer* relation R, we scan the entire *inner* relation S.
  - Cost:  $M + p_R * M * N = 1000 + 100*1000*500$  I/Os.
- ❖ Page-oriented Nested Loops join: For each *page* of R, get each *page* of S, and write out matching pairs of tuples  $\langle r, s \rangle$ , where r is in R-page and S is in S-page.
  - Cost:  $M + M*N = 1000 + 1000*500$

---

---

---

---

---

---

---

## Index Nested Loops Join

```
foreach tuple r in R do
 foreach tuple s in S where $r_i == s_j$ do
 add $\langle r, s \rangle$ to result
```

- ❖ If there is an index on the join column of one relation (say S), can make it the inner and exploit the index.
  - Cost:  $M + (M*p_R) * \text{cost of finding matching S tuples}$
- ❖ For each R tuple, cost of probing S index is about 1.2 for hash index, 2-4 for B+ tree. Cost of then finding S tuples (assuming Alt. (2) or (3) for data entries) depends on clustering.
  - Clustered index: 1 I/O (typical), unclustered: upto 1 I/O per matching S tuple.

---

---

---

---

---

---

---

## Examples of Index Nested Loops

- ❖ Hash-index (Alt. 2) on *sid* of Sailors (as inner):
  - Scan Reserves: 1000 page I/Os, 100\*1000 tuples.
  - For each Reserves tuple: 1.2 I/Os to get data entry in index, plus 1 I/O to get (the exactly one) matching Sailors tuple. Total: 220,000 I/Os.
- ❖ Hash-index (Alt. 2) on *sid* of Reserves (as inner):
  - Scan Sailors: 500 page I/Os, 80\*500 tuples.
  - For each Sailors tuple: 1.2 I/Os to find index page with data entries, plus cost of retrieving matching Reserves tuples. Assuming uniform distribution, 2.5 reservations per sailor (100,000 / 40,000). Cost of retrieving them is 1 or 2.5 I/Os depending on whether the index is clustered.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

7

---

---

---

---

---

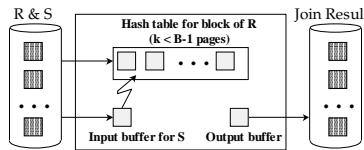
---

---

---

## Block Nested Loops Join

- ❖ Use one page as an input buffer for scanning the inner S, one page as the output buffer, and use all remaining pages to hold ``block'' of outer R.
  - For each matching tuple r in R-block, s in S-page, add  $\langle r, s \rangle$  to result. Then read next R-block, scan S, etc.



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

8

---

---

---

---

---

---

---

---

## Examples of Block Nested Loops

- ❖ Cost: Scan of outer + #outer blocks \* scan of inner
  - #outer blocks =  $\lceil \# \text{ of pages of outer} / \text{blocksize} \rceil$
- ❖ With Reserves (R) as outer, and 100 pages of R:
  - Cost of scanning R is 1000 I/Os; a total of 10 blocks.
  - Per block of R, we scan Sailors (S); 10\*500 I/Os.
  - If space for just 90 pages of R, we would scan S 12 times.
- ❖ With 100-page block of Sailors as outer:
  - Cost of scanning S is 500 I/Os; a total of 5 blocks.
  - Per block of S, we scan Reserves; 5\*1000 I/Os.
- ❖ With sequential reads considered, analysis changes: may be best to divide buffers evenly between R and S.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

9

---

---

---

---

---

---

---

---

## *Sort-Merge Join* ( $R \sqcup_{i=j} S$ )

- ❖ Sort R and S on the join column, then scan them to do a “merge” (on join col.), and output result tuples.
    - Advance scan of R until current R-tuple  $\geq$  current S tuple, then advance scan of S until current S-tuple  $\geq$  current R tuple; do this until current R tuple = current S tuple.
    - At this point, all R tuples with same value in  $R_i$  (*current R group*) and all S tuples with same value in  $S_j$  (*current S group*) *match*; output  $\langle r, s \rangle$  for all pairs of such tuples.
    - Then resume scanning R and S.
  - ❖ R is scanned once; each S group is scanned once per matching R tuple. (Multiple scans of an S group are likely to find needed pages in buffer.)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

10



## *Example of Sort-Merge Join*

				sid	bid	day	rname
sid	sname	rating	age	28	103	12/4/96	guppy
22	dustin	7	45.0	28	103	11/3/96	yuppy
28	yuppy	9	35.0	31	101	10/10/96	dustin
31	lubber	8	55.5	31	102	10/12/96	lubber
44	guppy	5	35.0	31	101	10/11/96	lubber
58	rusty	10	35.0	58	103	11/12/96	dustin

- ❖ Cost:  $M \log M + N \log N + (M+N)$ 
    - The cost of scanning,  $M+N$ , could be  $M*N$  (very unlikely!)
  - ❖ With 35, 100 or 300 buffer pages, both Reserves and Sailors can be sorted in 2 passes; total join cost: 7500.   

$$(PNL \text{ costs: } 2500 \text{ to } 15000 \text{ UOs})$$

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke (BNL cost: 2500 to 15000 I/Os) 11

1



## *Refinement of Sort-Merge Join*

- We can combine the merging phases in the *sorting* of R and S with the merging required for the join.
    - With  $B > \sqrt{L}$ , where  $L$  is the size of the larger relation, using the sorting refinement that produces runs of length  $2B$  in Pass 0, #runs of each relation is  $< B/2$ .
    - Allocate 1 page per run of each relation, and ‘merge’ while checking the join condition.
    - Cost: read+write each relation in Pass 0 + read each relation in (only) merging pass (+ writing of result tuples).
    - In example, cost goes down from 7500 to 4500 I/Os.
  - In practice, cost of sort-merge join, like the cost of external sorting, is *linear*.

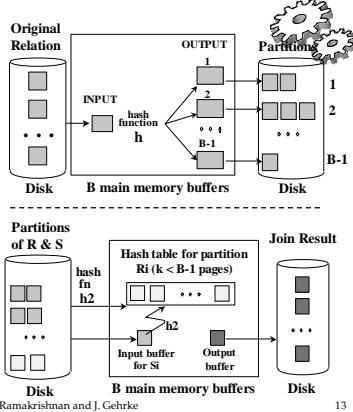
Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

12



## Hash-Join

- Partition both relations using hash fn  $h$ : R tuples in partition i will only match S tuples in partition i.
- Read in a partition of R, hash it using  $h_2 \ll h_1$ . Scan matching partition of S, search for matches.



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

13

## Observations on Hash-Join

- # partitions  $k < B-1$  (why?), and  $B-2 >$  size of largest partition to be held in memory. Assuming uniformly sized partitions, and maximizing  $k$ , we get:
  - $k = B-1$ , and  $M/(B-1) < B-2$ , i.e.,  $B$  must be  $> \sqrt{M}$
- If we build an in-memory hash table to speed up the matching of tuples, a little more memory is needed.
- If the hash function does not partition uniformly, one or more R partitions may not fit in memory. Can apply hash-join technique recursively to do the join of this R-partition with corresponding S-partition.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

14

## Cost of Hash-Join

- In partitioning phase, read+write both relns;  $2(M+N)$ . In matching phase, read both relns;  $M+N$  I/Os.
- In our running example, this is a total of 4500 I/Os.
- Sort-Merge Join vs. Hash Join:
  - Given a minimum amount of memory (*what is this, for each?*) both have a cost of  $3(M+N)$  I/Os. Hash Join superior on this count if relation sizes differ greatly. Also, Hash Join shown to be highly parallelizable.
  - Sort-Merge less sensitive to data skew; result is sorted.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

15

## *General Join Conditions*



- ❖ Equalities over several attributes (e.g.,  $R.sid=S.sid$  AND  $R.rname=S.sname$ ):
  - For Index NL, build index on  $\langle sid, sname \rangle$  (if S is inner); or use existing indexes on *sid* or *sname*.
  - For Sort-Merge and Hash Join, sort/partition on combination of the two join columns.
- ❖ Inequality conditions (e.g.,  $R.rname < S.sname$ ):
  - For Index NL, need (clustered!) B+ tree index.
    - Range probes on inner; # matches likely to be much higher than for equality joins.
  - Hash Join, Sort Merge Join not applicable.
  - Block NL quite likely to be the best join method here.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

16

---

---

---

---

---

---

---



## *Evaluation of Relational Operations: Other Techniques*

Chapter 12, Part B

---

---

---

---

---

---



## *Using an Index for Selections*

- ❖ Cost depends on #qualifying tuples, and clustering.
  - Cost of finding qualifying data entries (typically small) plus cost of retrieving records (could be large w/o clustering).
  - In example, assuming uniform distribution of names, about 10% of tuples qualify (100 pages, 10000 tuples). With a clustered index, cost is little more than 100 I/Os; if unclustered, upto 10000 I/Os!
- ❖ *Important refinement for unclustered indexes:*
  1. Find qualifying data entries.
  2. Sort the rid's of the data records to be retrieved.
  3. Fetch rids in order. This ensures that each data page is looked at just once (though # of such pages likely to be higher than with clustering).

---

---

---

---

---

---



## *Two Approaches to General Selections*

- ❖ First approach: Find the *most selective access path*, retrieve tuples using it, and apply any remaining terms that don't match the index:
  - *Most selective access path:* An index or file scan that we estimate will require the fewest page I/Os.
  - Terms that match this index reduce the number of tuples *retrieved*; other terms are used to discard some retrieved tuples, but do not affect number of tuples/pages fetched.
  - Consider  $day < 8/9/94 \text{ AND } bid = 5 \text{ AND } sid = 3$ . A B+ tree index on *day* can be used; then, *bid=5* and *sid=3* must be checked for each retrieved tuple. Similarly, a hash index on  $<bid, sid>$  could be used;  $day < 8/9/94$  must then be checked.

---

---

---

---

---

---

## Intersection of Rids

- ❖ Second approach (if we have 2 or more matching indexes that use Alternatives (2) or (3) for data entries):
  - Get sets of rids of data records using each matching index.
  - Then *intersect* these sets of rids (we'll discuss intersection soon!)
  - Retrieve the records and apply any remaining terms.
  - Consider  $day < 8/9/94 \text{ AND } bid = 5 \text{ AND } sid = 3$ . If we have a B+ tree index on *day* and an index on *sid*, both using Alternative (2), we can retrieve rids of records satisfying  $day < 8/9/94$  using the first, rids of recs satisfying  $sid = 3$  using the second, intersect, retrieve records and check *bid* = 5.

Database Management Systems 3ed, R. Ramakrishnan and Johannes Gehrke

4

---

---

---

---

---

---

---

---

## The Projection Operation

```
SELECT DISTINCT R.sid, R.bid
FROM Reserves R
```

- ❖ An approach based on sorting:
  - Modify Pass 0 of external sort to eliminate unwanted fields. Thus, runs of about 2B pages are produced, but tuples in runs are smaller than input tuples. (Size ratio depends on # and size of fields that are dropped.)
  - Modify merging passes to eliminate duplicates. Thus, number of result tuples smaller than input. (Difference depends on # of duplicates.)
  - Cost: In Pass 0, read original relation (size M), write out same number of smaller tuples. In merging passes, fewer tuples written out in each pass. Using Reserves example, 1000 input pages reduced to 250 in Pass 0 if size ratio is 0.25

Database Management Systems 3ed, R. Ramakrishnan and Johannes Gehrke

5

---

---

---

---

---

---

---

---

## Projection Based on Hashing

- ❖ *Partitioning phase*: Read R using one input buffer. For each tuple, discard unwanted fields, apply hash function  $h1$  to choose one of  $B-1$  output buffers.
  - Result is  $B-1$  partitions (of tuples with no unwanted fields). 2 tuples from different partitions guaranteed to be distinct.
- ❖ *Duplicate elimination phase*: For each partition, read it and build an in-memory hash table, using hash fn  $h2$  ( $\leftrightarrow h1$ ) on all fields, while discarding duplicates.
  - If partition does not fit in memory, can apply hash-based projection algorithm recursively to this partition.
- ❖ *Cost*: For partitioning, read R, write out each tuple, but with fewer fields. This is read in next phase.

Database Management Systems 3ed, R. Ramakrishnan and Johannes Gehrke

6

---

---

---

---

---

---

---

---

## Discussion of Projection



- ❖ Sort-based approach is the standard; better handling of skew and result is sorted.
- ❖ If an index on the relation contains all wanted attributes in its search key, can do *index-only* scan.
  - Apply projection techniques to data entries (much smaller!)
- ❖ If an ordered (i.e., tree) index contains all wanted attributes as *prefix* of search key, can do even better:
  - Retrieve data entries in order (index-only scan), discard unwanted fields, compare adjacent tuples to check for duplicates.

---

---

---

---

---

---

---

## Set Operations



- ❖ Intersection and cross-product special cases of join.
- ❖ Union (Distinct) and Except similar; we'll do union.
- ❖ Sorting based approach to union:
  - Sort both relations (on combination of all attributes).
  - Scan sorted relations and merge them.
  - *Alternative:* Merge runs from Pass 0 for *both* relations.
- ❖ Hash based approach to union:
  - Partition R and S using hash function  $h$ .
  - For each S-partition, build in-memory hash table (using  $h_2$ ), scan corr. R-partition and add tuples to table while discarding duplicates.

---

---

---

---

---

---

---

## Aggregate Operations (AVG, MIN, etc.)



- ❖ Without grouping:
  - In general, requires scanning the relation.
  - Given index whose search key includes all attributes in the SELECT or WHERE clauses, can do index-only scan.
- ❖ With grouping:
  - Sort on group-by attributes, then scan relation and compute aggregate for each group. (Can improve upon this by combining sorting and aggregate computation.)
  - Similar approach based on hashing on group-by attributes.
  - Given tree index whose search key includes all attributes in SELECT, WHERE and GROUP BY clauses, can do index-only scan; if group-by attributes form prefix of search key, can retrieve data entries/tuples in group-by order.

---

---

---

---

---

---

---

## *Impact of Buffering*



- ❖ If several operations are executing concurrently, estimating the number of available buffer pages is guesswork.
- ❖ Repeated access patterns interact with buffer replacement policy.
  - e.g., Inner relation is scanned repeatedly in Simple Nested Loop Join. With enough buffer pages to hold inner, replacement policy does not matter. Otherwise, MRU is best, LRU is worst (*sequential flooding*).
  - Does replacement policy matter for Block Nested Loops?
  - What about Index Nested Loops? Sort-Merge Join?

Database Management Systems 3ed, R. Ramakrishnan and Johannes Gehrke

10

---

---

---

---

---

---

---

---

---

## *Summary*



- ❖ A virtue of relational DBMSs: *queries are composed of a few basic operators*; the implementation of these operators can be carefully tuned (and it is important to do this!).
- ❖ Many alternative implementation techniques for each operator; no universally superior technique for most operators.
- ❖ Must consider available alternatives for each operation in a query and choose best one based on system statistics, etc. This is part of the broader task of optimizing a query composed of several ops.

Database Management Systems 3ed, R. Ramakrishnan and Johannes Gehrke

11

---

---

---

---

---

---

---

---

---

## *Relational Query Optimization*

### Chapter 15

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

1

---

---

---

---

---

---

## *Highlights of System R Optimizer*

### ❖ Impact:

- Most widely used currently; works well for < 10 joins.
- ❖ Cost estimation: Approximate art at best.
  - Statistics, maintained in system catalogs, used to estimate cost of operations and result sizes.
  - Considers combination of CPU and I/O costs.
- ❖ Plan Space: Too large, must be pruned.
  - Only the space of *left-deep plans* is considered.
    - Left-deep plans allow output of each operator to be *pipelined* into the next operator without storing it in a temporary relation.
    - Cartesian products avoided.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

2

---

---

---

---

---

---

## *Overview of Query Optimization*

- ❖ Plan: Tree of R.A. ops, with choice of alg for each op.
  - Each operator typically implemented using a ‘pull’ interface: when an operator is ‘pulled’ for the next output tuples, it ‘pulls’ on its inputs and computes them.
- ❖ 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!
- ❖ We will study the System R approach.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

3

---

---

---

---

---

---

## Schema for Examples

Sailors (sid: integer, sname: string, rating: integer, age: real)  
Reserves (sid: integer, bid: integer, day: dates, rname: string)

- ❖ Similar to old schema; rname added for variations.
- ❖ Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
- ❖ Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.

---

---

---

---

---

---

---

---

## Query Blocks: Units of Optimization

- ❖ An SQL query is parsed into a collection of *query blocks*, and these are optimized one block at a time.
- ❖ Nested blocks are usually treated as calls to a subroutine, made once per outer tuple. (This is an over-simplification, but serves for now.)
- ❖ For each block, the plans considered are:
  - All available access methods, for each reln in FROM clause.
  - All *left-deep join trees* (i.e., all ways to join the relations one-at-a-time, with the inner reln in the FROM clause, considering all reln permutations and join methods.)

```
SELECT S.sname
FROM Sailors S
WHERE S.age IN
 (SELECT MAX(S2.age)
 FROM Sailors S2
 GROUP BY S2.rating)
```

Outer block    Nested block

---

---

---

---

---

---

---

---

## Relational Algebra Equivalences

- ❖ Allow us to choose different join orders and to ‘push’ selections and projections ahead of joins.
- ❖ Selections:  $\sigma_{c_1 \wedge \dots \wedge c_n}(R) \equiv \sigma_{c_1}(\dots \sigma_{c_n}(R))$  (Cascade)  
 $\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$  (Commute)
- ❖ Projections:  $\pi_{a_1}(R) \equiv \pi_{a_1}(\dots(\pi_{a_n}(R)))$  (Cascade)
- ❖ Joins:  $R \sqcup\!\sqcup (S \sqcup\!\sqcup T) \equiv (R \sqcup\!\sqcup S) \sqcup\!\sqcup T$  (Associative)  
 $(R \sqcup\!\sqcup S) \equiv (S \sqcup\!\sqcup R)$  (Commute)

+ Show that:  $R \sqcap\!\sqcap (S \sqcap\!\sqcap T) \equiv (T \sqcap\!\sqcap R) \sqcap\!\sqcap S$

---

---

---

---

---

---

---

---

## *More Equivalences*

- ❖ A projection commutes with a selection that only uses attributes retained by the projection.
- ❖ Selection between attributes of the two arguments of a cross-product converts cross-product to a join.
- ❖ A selection on just attributes of R commutes with  $R \bowtie S$ . (i.e.,  $\sigma(R \bowtie S) \equiv \sigma(R) \bowtie S$ )
- ❖ Similarly, if a projection follows a join  $R \bowtie S$ , we can ‘push’ it by retaining only attributes of R (and S) that are needed for the join or are kept by the projection.

---

---

---

---

---

---

---

## *Enumeration of Alternative Plans*

- ❖ There are two main cases:
  - Single-relation plans
  - Multiple-relation plans
- ❖ For queries over a single relation, queries consist of a combination of selects, projects, and aggregate ops:
  - Each available access path (file scan / index) is considered, and the one with the least estimated cost is chosen.
  - The different operations are essentially carried out together (e.g., if an index is used for a selection, projection is done for each retrieved tuple, and the resulting tuples are *pipelined* into the aggregate computation).

---

---

---

---

---

---

---

## *Cost Estimation*

- ❖ For each plan considered, must estimate cost:
  - Must estimate *cost* of each operation in plan tree.
    - Depends on input cardinalities.
    - We’ve already discussed how to estimate the cost of operations (sequential scan, index scan, joins, etc.)
  - Must also estimate *size of result* for each operation in tree!
    - Use information about the input relations.
    - For selections and joins, assume independence of predicates.

---

---

---

---

---

---

---

## Cost Estimates for Single-Relation Plans

- ❖ Index I on primary key matches selection:
  - Cost is  $\text{Height}(I)+1$  for a B+ tree, about 1.2 for hash index.
- ❖ Clustered index I matching one or more selects:
  - $(N\text{Pages}(I)+N\text{Pages}(R)) * \text{product of RF's of matching selects}$ .
- ❖ Non-clustered index I matching one or more selects:
  - $(N\text{Pages}(I)+N\text{Tuples}(R)) * \text{product of RF's of matching selects}$ .
- ❖ Sequential scan of file:
  - $N\text{Pages}(R)$ .
- + **Note:** Typically, no duplicate elimination on projections!  
(Exception: Done on answers if user says DISTINCT.)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

10

## Example

```
SELECT S.sid
FROM Sailors S
WHERE S.rating=8
```

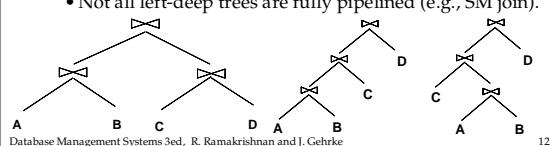
- ❖ If we have an index on *rating*:
  - $(1/\text{NKeys}(I)) * \text{NTuples}(R) = (1/10) * 40000$  tuples retrieved.
  - Clustered index:  $(1/\text{NKeys}(I)) * (N\text{Pages}(I)+N\text{Pages}(R)) = (1/10) * (50+500)$  pages are retrieved. (This is the *cost*.)
  - Unclustered index:  $(1/\text{NKeys}(I)) * (N\text{Pages}(I)+\text{NTuples}(R)) = (1/10) * (50+40000)$  pages are retrieved.
- ❖ If we have an index on *sid*:
  - Would have to retrieve all tuples/pages. With a clustered index, the cost is 50+500, with unclustered index, 50+40000.
- ❖ Doing a file scan:
  - We retrieve all file pages (500).

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

11

## Queries Over Multiple Relations

- ❖ Fundamental decision in System R: only left-deep join trees are considered.
  - As the number of joins increases, the number of alternative plans grows rapidly; we need to restrict the search space.
  - Left-deep trees allow us to generate all *fully pipelined* plans.
    - Intermediate results not written to temporary files.
    - Not all left-deep trees are fully pipelined (e.g., SM join).



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

12

### *Enumeration of Left-Deep Plans*

- ❖ Left-deep plans differ only in the order of relations, the access method for each relation, and the join method for each join.
- ❖ Enumerated using N passes (if N relations joined):
  - Pass 1: Find best 1-relation plan for each relation.
  - Pass 2: Find best way to join result of each 1-relation plan (as outer) to another relation. (*All 2-relation plans.*)
  - Pass N: Find best way to join result of a (N-1)-relation plan (as outer) to the N'th relation. (*All N-relation plans.*)
- ❖ For each subset of relations, retain only:
  - Cheapest plan overall, plus
  - Cheapest plan for each *interesting order* of the tuples.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

13

---

---

---

---

---

---

---

---

---

### *Enumeration of Plans (Contd.)*

- ❖ ORDER BY, GROUP BY, aggregates etc. handled as a final step, using either an ‘interestingly ordered’ plan or an additional sorting operator.
- ❖ An N-1 way plan is not combined with an additional relation unless there is a join condition between them, unless all predicates in WHERE have been used up.
  - i.e., avoid Cartesian products if possible.
- ❖ In spite of pruning plan space, this approach is still exponential in the # of tables.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

14

---

---

---

---

---

---

---

---

---

### *Cost Estimation for Multirelation Plans*

SELECT attribute list  
FROM relation list

- ❖ Consider a query block: 

SELECT attribute list FROM relation list WHERE term1 AND ... AND termk
------------------------------------------------------------------------------
- ❖ Maximum # tuples in result is the product of the cardinalities of relations in the FROM clause.
- ❖ *Reduction factor (RF)* associated with each *term* reflects the impact of the *term* in reducing result size. *Result cardinality* = Max # tuples \* product of all RF's.
- ❖ Multirelation plans are built up by joining one new relation at a time.
  - Cost of join method, plus estimation of join cardinality gives us both cost estimate and result size estimate

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

15

---

---

---

---

---

---

---

---

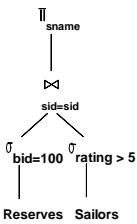
---

## Example

- ❖ Pass1:
  - *Sailors*: B+ tree matches  $rating > 5$ , and is probably cheapest. However, if this selection is expected to retrieve a lot of tuples, and index is unclustered, file scan may be cheaper.
    - Still, B+ tree plan kept (because tuples are in *rating* order).
  - *Reserves*: B+ tree on *bid* matches  $bid = 500$ ; cheapest.
- ❖ Pass 2:
  - We consider each plan retained from Pass 1 as the outer, and consider how to join it with the (only) other relation.
    - e.g., *Reserves as outer*: Hash index can be used to get *Sailors* tuples that satisfy  $sid =$  outer tuple's *sid* value.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

16



## Nested Queries

- ❖ Nested block is optimized independently, with the outer tuple considered as providing a selection condition.
- ❖ Outer block is optimized with the cost of 'calling' nested block computation taken into account.
- ❖ Implicit ordering of these blocks means that some good strategies are not considered. *The non-nested version of the query is typically optimized better.*

```

SELECT S.sname
FROM Sailors S
WHERE EXISTS
 (SELECT *
 FROM Reserves R
 WHERE R.bid=103
 AND R.sid=S.sid)

```

Nested block to optimize:  
`SELECT *  
FROM Reserves R  
WHERE R.bid=103  
AND S.sid= outer value`

Equivalent non-nested query:  
`SELECT S.sname  
FROM Sailors S, Reserves R  
WHERE S.sid=R.sid  
AND R.bid=103`

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

17

## Summary

- ❖ Query optimization is an important task in a relational DBMS.
- ❖ Must understand optimization in order to understand the performance impact of a given database design (relations, indexes) on a workload (set of queries).
- ❖ Two parts to optimizing a query:
  - Consider a set of alternative plans.
    - Must prune search space; typically, left-deep plans only.
  - Must estimate cost of each plan that is considered.
    - Must estimate size of result and cost for each plan node.
    - Key issues: Statistics, indexes, operator implementations.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

18

## *Summary (Contd.)*

### ❖ Single-relation queries:

- All access paths considered, cheapest is chosen.
- *Issues:* Selections that *match* index, whether index key has all needed fields and/or provides tuples in a desired order.

### ❖ Multiple-relation queries:

- All single-relation plans are first enumerated.
  - Selections/projections considered as early as possible.
- Next, for each 1-relation plan, all ways of joining another relation (as inner) are considered.
- Next, for each 2-relation plan that is ‘retained’, all ways of joining another relation (as inner) are considered, etc.
- At each level, for each subset of relations, only best plan for each interesting order of tuples is ‘retained’.

---

---

---

---

---

---

---

---

---

---



## *Transaction Management Overview*

### Chapter 16

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

1

---

---

---

---

---

---

---



## *Transactions*

- ❖ Concurrent execution of user programs is essential for good DBMS performance.
  - Because disk accesses are frequent, and relatively slow, it is important to keep the cpu humming by working on several user programs concurrently.
- ❖ A user's program may carry out many operations on the data retrieved from the database, but the DBMS is only concerned about what data is read/written from/to the database.
- ❖ A transaction is the DBMS's abstract view of a user program: a sequence of reads and writes.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

2

---

---

---

---

---

---

---



## *Concurrency in a DBMS*

- ❖ Users submit transactions, and can think of each transaction as executing by itself.
  - Concurrency is achieved by the DBMS, which interleaves actions (reads/writes of DB objects) of various transactions.
  - Each transaction must leave the database in a consistent state if the DB is consistent when the transaction begins.
    - DBMS will enforce some ICs, depending on the ICs declared in CREATE TABLE statements.
    - Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
- ❖ Issues: Effect of *interleaving* transactions, and *crashes*.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

3

---

---

---

---

---

---

---

## *Atomicity of Transactions*



- ❖ A transaction might *commit* after completing all its actions, or it could *abort* (or be aborted by the DBMS) after executing some actions.
- ❖ A very important property guaranteed by the DBMS for all transactions is that they are atomic. That is, a user can think of a Xact as always executing all its actions in one step, or not executing any actions at all.
  - DBMS logs all actions so that it can *undo* the actions of aborted transactions.

---

---

---

---

---

---

---

## *Example*



- ❖ Consider two transactions (*Xacts*):

T1:	BEGIN	A=A+100,	B=B-100	END
T2:	BEGIN	A=1.06*A,	B=1.06*B	END
- ❖ Intuitively, the first transaction is transferring \$100 from B's account to A's account. The second is crediting both accounts with a 6% interest payment.
- ❖ There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together. However, the net effect *must* be equivalent to these two transactions running serially in some order.

---

---

---

---

---

---

---

## *Example (Contd.)*



- ❖ Consider a possible interleaving (*schedule*):

T1:	A=A+100,	B=B-100
T2:	A=1.06*A,	B=1.06*B
- ❖ This is OK. But what about:

T1:	A=A+100,	B=B-100
T2:	A=1.06*A, B=1.06*B	
- ❖ The DBMS's view of the second schedule:

T1:	R(A), W(A),	R(B), W(B)
T2:	R(A), W(A), R(B), W(B)	

---

---

---

---

---

---

---

## Scheduling Transactions



- ❖ Serial schedule: Schedule that does not interleave the actions of different transactions.
- ❖ Equivalent schedules: For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second schedule.
- ❖ Serializable schedule: A schedule that is equivalent to some serial execution of the transactions.  
(Note: If each transaction preserves consistency, every serializable schedule preserves consistency.)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

7

---

---

---

---

---

---

---

## Anomalies with Interleaved Execution



- ❖ Reading Uncommitted Data (WR Conflicts, "dirty reads"):

T1: R(A), W(A),	R(B), W(B), Abort
T2:	R(A), W(A), C

- ❖ Unrepeatable Reads (RW Conflicts):

T1: R(A),	R(A), W(A), C
T2:	R(A), W(A), C

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

8

---

---

---

---

---

---

---

## Anomalies (Continued)



- ❖ Overwriting Uncommitted Data (WW Conflicts):

T1: W(A),	W(B), C
T2:	W(A), W(B), C

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

9

---

---

---

---

---

---

---

## Lock-Based Concurrency Control

### ❖ Strict Two-phase Locking (Strict 2PL) Protocol:

- Each Xact must obtain a S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing.
  - All locks held by a transaction are released when the transaction completes.
  - If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.
- ❖ Strict 2PL allows only serializable schedules.

---

---

---

---

---

---

---

---

## Aborting a Transaction

- ❖ If a transaction  $T_i$  is aborted, all its actions have to be undone. Not only that, if  $T_j$  reads an object last written by  $T_i$ ,  $T_j$  must be aborted as well!
- ❖ Most systems avoid such *cascading aborts* by releasing a transaction's locks only at commit time.
  - If  $T_i$  writes an object,  $T_j$  can read this only after  $T_i$  commits.
- ❖ In order to *undo* the actions of an aborted transaction, the DBMS maintains a *log* in which every write is recorded. This mechanism is also used to recover from system crashes: all active Xacts at the time of the crash are aborted when the system comes back up.

---

---

---

---

---

---

---

---

## The Log

- ❖ The following actions are recorded in the log:
  - $T_i$  writes an object: the old value and the new value.
    - Log record must go to disk *before* the changed page!
    - $T_i$  commits/aborts: a log record indicating this action.
- ❖ Log records are chained together by Xact id, so it's easy to undo a specific Xact.
- ❖ Log is often *duplexed* and *archived* on stable storage.
- ❖ All log related activities (and in fact, all CC related activities such as lock/unlock, dealing with deadlocks etc.) are handled transparently by the DBMS.

---

---

---

---

---

---

---

---

## Recovering From a Crash



- ❖ There are 3 phases in the *Aries* recovery algorithm:
  - Analysis: Scan the log forward (from the most recent *checkpoint*) to identify all Xacts that were active, and all dirty pages in the buffer pool at the time of the crash.
  - Redo: Redoes all updates to dirty pages in the buffer pool, as needed, to ensure that all logged updates are in fact carried out and written to disk.
  - Undo: The writes of all Xacts that were active at the crash are undone (by restoring the *before value* of the update, which is in the log record for the update), working backwards in the log. (Some care must be taken to handle the case of a crash occurring during the recovery process!)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

13

## Summary



- ❖ Concurrency control and recovery are among the most important functions provided by a DBMS.
- ❖ Users need not worry about concurrency.
  - System automatically inserts lock/unlock requests and schedules actions of different Xacts in such a way as to ensure that the resulting execution is equivalent to executing the Xacts one after the other in some order.
- ❖ Write-ahead logging (WAL) is used to undo the actions of aborted transactions and to restore the system to a consistent state after a crash.
  - *Consistent state*: Only the effects of committed Xacts seen.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

14

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

# Concurrency Control

## Chapter 17

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

1

---

---

---

---

---

---

# Conflict Serializable Schedules

- ❖ Two schedules are conflict equivalent if:
  - Involve the same actions of the same transactions
  - Every pair of conflicting actions is ordered the same way
- ❖ Schedule S is conflict serializable if S is conflict equivalent to some serial schedule

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

2

---

---

---

---

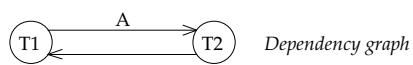
---

---

# Example

- ❖ A schedule that is not conflict serializable:

T1:	R(A), W(A),	R(B), W(B)
T2:	R(A), W(A), R(B), W(B)	



- ❖ The cycle in the graph reveals the problem. The output of T1 depends on T2, and vice-versa.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

3

---

---

---

---

---

---

## Dependency Graph



- ❖ Dependency graph: One node per Xact; edge from  $T_i$  to  $T_j$  if  $T_j$  reads/writes an object last written by  $T_i$ .
- ❖ Theorem: Schedule is conflict serializable if and only if its dependency graph is acyclic

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

4

---

---

---

---

---

---

## Review: Strict 2PL



- ❖ Strict Two-phase Locking (Strict 2PL) Protocol:
  - Each Xact must obtain a S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing.
  - All locks held by a transaction are released when the transaction completes
  - If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.
- ❖ Strict 2PL allows only schedules whose precedence graph is acyclic

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

5

---

---

---

---

---

---

## Two-Phase Locking (2PL)



- ❖ Two-Phase Locking Protocol
  - Each Xact must obtain a S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing.
  - A transaction can not request additional locks once it releases any locks.
  - If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

6

---

---

---

---

---

---

## *View Serializability*



- ❖ Schedules S1 and S2 are view equivalent if:
  - If Ti reads initial value of A in S1, then Ti also reads initial value of A in S2
  - If Ti reads value of A written by Tj in S1, then Ti also reads value of A written by Tj in S2
  - If Ti writes final value of A in S1, then Ti also writes final value of A in S2

T1: R(A)	W(A)
T2:	W(A)
T3:	W(A)

T1: R(A), W(A)	
T2:	W(A)
T3:	W(A)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

7

---

---

---

---

---

---

---

## *Lock Management*



- ❖ Lock and unlock requests are handled by the lock manager
- ❖ Lock table entry:
  - Number of transactions currently holding a lock
  - Type of lock held (shared or exclusive)
  - Pointer to queue of lock requests
- ❖ Locking and unlocking have to be atomic operations
- ❖ Lock upgrade: transaction that holds a shared lock can be upgraded to hold an exclusive lock

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

8

---

---

---

---

---

---

---

## *Deadlocks*



- ❖ Deadlock: Cycle of transactions waiting for locks to be released by each other.
- ❖ Two ways of dealing with deadlocks:
  - Deadlock prevention
  - Deadlock detection

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

9

---

---

---

---

---

---

---

## Deadlock Prevention

- ❖ Assign priorities based on timestamps.  
Assume  $T_i$  wants a lock that  $T_j$  holds. Two policies are possible:
  - Wait-Die: If  $T_i$  has higher priority,  $T_i$  waits for  $T_j$ ; otherwise  $T_i$  aborts
  - Wound-wait: If  $T_i$  has higher priority,  $T_j$  aborts; otherwise  $T_i$  waits
- ❖ If a transaction re-starts, make sure it has its original timestamp

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

10



---

---

---

---

---

## Deadlock Detection

- ❖ Create a waits-for graph:
  - Nodes are transactions
  - There is an edge from  $T_i$  to  $T_j$  if  $T_i$  is waiting for  $T_j$  to release a lock
- ❖ Periodically check for cycles in the waits-for graph

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

11



---

---

---

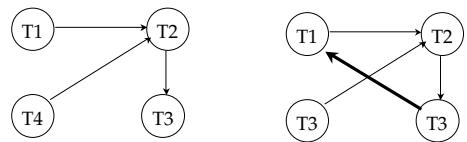
---

---

## Deadlock Detection (Continued)

Example:

T1: S(A), R(A),                                   S(B)  
T2:                                                   X(B), W(B)  
T3:                                                   S(C), R(C)                           X(C)  
T4:                                                   X(A)  
                                                         X(B)



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

12



---

---

---

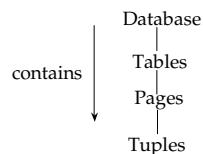
---

---

## Multiple-Granularity Locks



- ❖ Hard to decide what granularity to lock (tuples vs. pages vs. tables).
- ❖ Shouldn't have to decide!
- ❖ Data "containers" are nested:



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

13

---

---

---

---

---

---

---

## Solution: New Lock Modes, Protocol



- ❖ Allow Xacts to lock at each level, but with a special protocol using new "intention" locks:
  - v Before locking an item, Xact must set "intention locks" on all its ancestors.
  - v For unlock, go from specific to general (i.e., bottom-up).
  - v SIX mode: Like S & IX at the same time.

	--	IS	IX	S	X
--	✓	✓	✓	✓	✓
IS	✓	✓	✓	✓	
IX	✓	✓	✓		
S	✓	✓		✓	
X	✓				

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

14

---

---

---

---

---

---

---

## Multiple Granularity Lock Protocol



- ❖ Each Xact starts from the root of the hierarchy.
- ❖ To get S or IS lock on a node, must hold IS or IX on parent node.
  - What if Xact holds SIX on parent? S on parent?
- ❖ To get X or IX or SIX on a node, must hold IX or SIX on parent node.
- ❖ Must release locks in bottom-up order.

Protocol is correct in that it is equivalent to directly setting locks at the leaf levels of the hierarchy.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

15

---

---

---

---

---

---

---

## Examples

- ❖ T1 scans R, and updates a few tuples:
  - T1 gets an S lock on R, then repeatedly gets an S lock on tuples of R, and occasionally upgrades to X on the tuples.
- ❖ T2 uses an index to read only part of R:
  - T2 gets an IS lock on R, and repeatedly gets an S lock on tuples of R.
- ❖ T3 reads all of R:
  - T3 gets an S lock on R.
  - OR, T3 could behave like T2; can use lock escalation to decide which.

	-	IS	IX	S	X
-	✓	✓	✓	✓	✓
IS	✓	✓	✓	✓	
IX		✓	✓		
S	✓	✓		✓	
X	✓				

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

16

## Dynamic Databases

- ❖ If we relax the assumption that the DB is a fixed collection of objects, even Strict 2PL will not assure serializability:
  - T1 locks all pages containing sailor records with *rating* = 1, and finds oldest sailor (say, *age* = 71).
  - Next, T2 inserts a new sailor; *rating* = 1, *age* = 96.
  - T2 also deletes oldest sailor with *rating* = 2 (and, say, *age* = 80), and commits.
  - T1 now locks all pages containing sailor records with *rating* = 2, and finds oldest (say, *age* = 63).
- ❖ No consistent DB state where T1 is "correct"!

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

17

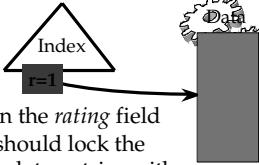
## The Problem

- ❖ T1 implicitly assumes that it has locked the set of all sailor records with *rating* = 1.
  - Assumption only holds if no sailor records are added while T1 is executing!
  - Need some mechanism to enforce this assumption. (Index locking and predicate locking.)
- ❖ Example shows that conflict serializability guarantees serializability only if the set of objects is fixed!

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

18

## *Index Locking*



- ❖ If there is a dense index on the *rating* field using Alternative (2), T1 should lock the index page containing the data entries with *rating* = 1.
  - If there are no records with *rating* = 1, T1 must lock the index page where such a data entry *would* be, if it existed!
- ❖ If there is no suitable index, T1 must lock all pages, and lock the file/table to prevent new pages from being added, to ensure that no new records with *rating* = 1 are added.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

19

---

---

---

---

---

---

## *Predicate Locking*



- ❖ Grant lock on all records that satisfy some logical predicate, e.g.  $age > 2 * salary$ .
- ❖ Index locking is a special case of predicate locking for which an index supports efficient implementation of the predicate lock.
  - What is the predicate in the sailor example?
- ❖ In general, predicate locking has a lot of locking overhead.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

20

---

---

---

---

---

---

## *Locking in B+ Trees*



- ❖ How can we efficiently lock a particular leaf node?
  - Btw, don't confuse this with multiple granularity locking!
- ❖ One solution: Ignore the tree structure, just lock pages while traversing the tree, following 2PL.
- ❖ This has terrible performance!
  - Root node (and many higher level nodes) become bottlenecks because every tree access begins at the root.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

21

---

---

---

---

---

---

## Two Useful Observations

- ❖ Higher levels of the tree only direct searches for leaf pages.
- ❖ For inserts, a node on a path from root to modified leaf must be locked (in X mode, of course), only if a split can propagate up to it from the modified leaf. (Similar point holds w.r.t. deletes.)
- ❖ We can exploit these observations to design efficient locking protocols that guarantee serializability even though they violate 2PL.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

22

---

---

---

---

---

---

## A Simple Tree Locking Algorithm

- ❖ Search: Start at root and go down; repeatedly, S lock child then unlock parent.
- ❖ Insert/Delete: Start at root and go down, obtaining X locks as needed. Once child is locked, check if it is safe:
  - If child is safe, release all locks on ancestors.
- ❖ Safe node: Node such that changes will not propagate up beyond this node.
  - Inserts: Node is not full.
  - Deletes: Node is not half-empty.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

23

---

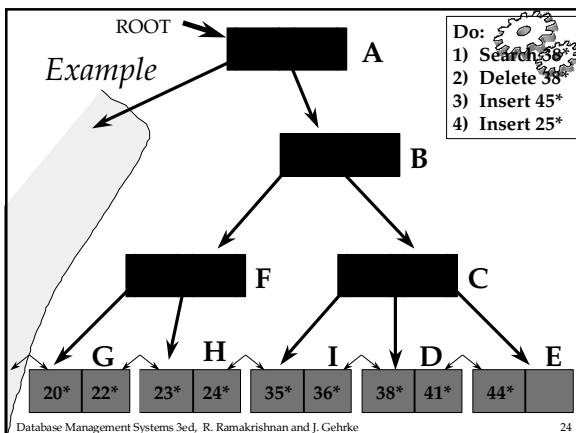
---

---

---

---

---



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

24

---

---

---

---

---

---

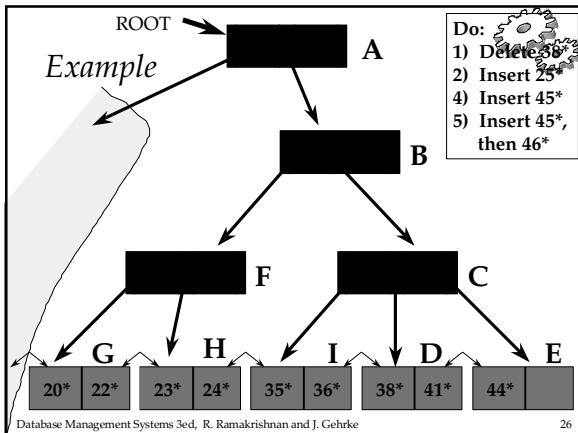
## A Better Tree Locking Algorithm

(See Bayer-Schkolnick paper)

- ❖ Search: As before.
- ❖ Insert/Delete:
  - Set locks as if for search, get to leaf, and set X lock on leaf.
  - If leaf is not safe, release all locks, and restart Xact using previous Insert/Delete protocol.
- ❖ Gambles that only leaf node will be modified; if not, S locks set on the first pass to leaf are wasteful. In practice, better than previous alg.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

25



## Even Better Algorithm

- ❖ Search: As before.
- ❖ Insert/Delete:
  - Use original Insert/Delete protocol, but set IX locks instead of X locks at all nodes.
  - Once leaf is locked, convert all IX locks to X locks top-down: i.e., starting from node nearest to root. (Top-down reduces chances of deadlock.)

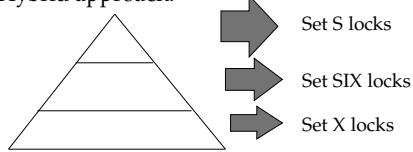
(Contrast use of IX locks here with their use in multiple-granularity locking.)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

27

## Hybrid Algorithm

- ❖ The likelihood that we really need an X lock decreases as we move up the tree.
- ❖ Hybrid approach:



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

28

---

---

---

---

---

---

## Optimistic CC (Kung-Robinson)

- ❖ Locking is a conservative approach in which conflicts are prevented. Disadvantages:
  - Lock management overhead.
  - Deadlock detection/resolution.
  - Lock contention for heavily used objects.
- ❖ If conflicts are rare, we might be able to gain concurrency by not locking, and instead checking for conflicts before Xacts commit.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

29

---

---

---

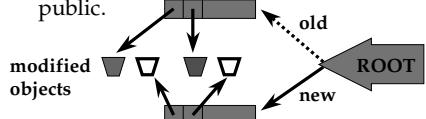
---

---

---

## Kung-Robinson Model

- ❖ Xacts have three phases:
  - READ: Xacts read from the database, but make changes to private copies of objects.
  - VALIDATE: Check for conflicts.
  - WRITE: Make local copies of changes public.



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

30

---

---

---

---

---

---

## Validation

- ❖ Test conditions that are sufficient to ensure that no conflict occurred.
- ❖ Each Xact is assigned a numeric id.
  - Just use a timestamp.
- ❖ Xact ids assigned at end of READ phase, just before validation begins. (Why then?)
- ❖ ReadSet(Ti): Set of objects read by Xact Ti.
- ❖ WriteSet(Ti): Set of objects modified by Ti.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

31

---

---

---

---

---

---

---

## Test 1

- ❖ For all i and j such that  $T_i < T_j$ , check that  $T_i$  completes before  $T_j$  begins.



Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

32

---

---

---

---

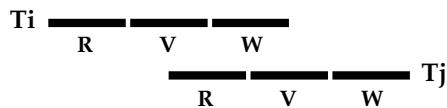
---

---

---

## Test 2

- ❖ For all i and j such that  $T_i < T_j$ , check that:
  - $T_i$  completes before  $T_j$  begins its Write phase +
  - $\text{WriteSet}(T_i) \cap \text{ReadSet}(T_j)$  is empty.



Does  $T_j$  read dirty data? Does  $T_i$  overwrite  $T_j$ 's writes?

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

33

---

---

---

---

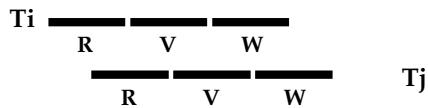
---

---

---

### Test 3

- ❖ For all i and j such that  $T_i < T_j$ , check that:
  - $T_i$  completes Read phase before  $T_j$  does +
  - $WriteSet(T_i) \cap ReadSet(T_j)$  is empty +
  - $WriteSet(T_i) \cap WriteSet(T_j)$  is empty.



Does  $T_j$  read dirty data? Does  $T_i$  overwrite  $T_j$ 's writes?

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

34

### Applying Tests 1 & 2: Serial Validation

- ❖ To validate Xact T:

```
valid = true;
// S = set of Xacts that committed after Begin(T)
< foreach Ts in S do {
 if ReadSet(Ts) does not intersect WriteSet(Ts)
 then valid = false;
 }
 if valid then { install updates; // Write phase
 Commit T } >
 else Restart T
end of critical section
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

35

### Comments on Serial Validation

- ❖ Applies Test 2, with T playing the role of  $T_j$  and each Xact in Ts (in turn) being  $T_i$ .
- ❖ Assignment of Xact id, validation, and the Write phase are inside a **critical section!**
  - I.e., Nothing else goes on concurrently.
  - If Write phase is long, major drawback.
- ❖ Optimization for Read-only Xacts:
  - Don't need critical section (because there is no Write phase).

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

36

## *Serial Validation (Contd.)*



- ❖ Multistage serial validation: Validate in stages, at each stage validating T against a subset of the Xacts that committed after Begin(T).
  - Only last stage has to be inside critical section.
- ❖ Starvation: Run starving Xact in a critical section (!!)
- ❖ Space for WriteSets: To validate  $T_j$ , must have WriteSets for all  $T_i$  where  $T_i < T_j$  and  $T_i$  was active when  $T_j$  began. There may be many such Xacts, and we may run out of space.
  - $T_j$ 's validation fails if it requires a missing WriteSet.
  - No problem if Xact ids assigned at start of Read phase.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

37

---

---

---

---

---

---

---

## *Overheads in Optimistic CC*



- ❖ Must record read/write activity in ReadSet and WriteSet per Xact.
  - Must create and destroy these sets as needed.
- ❖ Must check for conflicts during validation, and must make validated writes ``global''.
  - Critical section can reduce concurrency.
  - Scheme for making writes global can reduce clustering of objects.
- ❖ Optimistic CC restarts Xacts that fail validation.
  - Work done so far is wasted; requires clean-up.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

38

---

---

---

---

---

---

---

## *``Optimistic'' 2PL*



- ❖ If desired, we can do the following:
  - Set S locks as usual.
  - Make changes to private copies of objects.
  - Obtain all X locks at end of Xact, make writes global, then release all locks.
- ❖ In contrast to Optimistic CC as in Kung-Robinson, this scheme results in Xacts being blocked, waiting for locks.
  - However, no validation phase, no restarts (modulo deadlocks).

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

39

---

---

---

---

---

---

---

## Timestamp CC



- ❖ **Idea:** Give each object a read-timestamp (RTS) and a write-timestamp (WTS), give each Xact a timestamp (TS) when it begins:
  - If action  $a_i$  of Xact  $T_i$  conflicts with action  $a_j$  of Xact  $T_j$ , and  $TS(T_i) < TS(T_j)$ , then  $a_i$  must occur before  $a_j$ . Otherwise, restart violating Xact.

---

---

---

---

---

---

## When Xact T wants to read Object O



- ❖ If  $TS(T) < WTS(O)$ , this violates timestamp order of T w.r.t. writer of O.
  - So, abort T and restart it with a new, larger TS. (If restarted with same TS, T will fail again! Contrast use of timestamps in 2PL for ddlk prevention.)
- ❖ If  $TS(T) > WTS(O)$ :
  - Allow T to read O.
  - Reset RTS(O) to  $\max(RTS(O), TS(T))$
- ❖ Change to RTS(O) on reads must be written to disk! This and restarts represent overheads.

---

---

---

---

---

---

## When Xact T wants to Write Object O



- ❖ If  $TS(T) < RTS(O)$ , this violates timestamp order of T w.r.t. writer of O; abort and restart T.
- ❖ If  $TS(T) < WTS(O)$ , violates timestamp order of T w.r.t. writer of O.
  - Thomas Write Rule: We can safely ignore such outdated writes; need not restart T! (T's write is effectively followed by another write, with no intervening reads.) Allows some serializable but non conflict serializable schedules:
- ❖ Else, allow T to write O.

T1	T2
R(A)	W(A) Commit
W(A) Commit	

---

---

---

---

---

---

## Timestamp CC and Recoverability

- ❖ Unfortunately, unrecoverable schedules are allowed:
- ❖ Timestamp CC can be modified to allow only recoverable schedules:
  - Buffer all writes until writer commits (but update WTS(O) when the write is allowed.)
  - Block readers T (where TS(T) > WTS(O)) until writer of O commits.
- ❖ Similar to writers holding X locks until commit, but still not quite 2PL.

T1	T2
W(A)	R(A) W(B) Commit

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

43

---

---

---

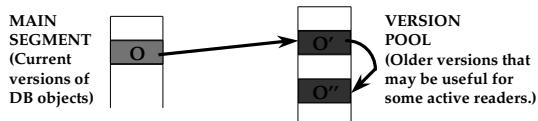
---

---

---

## Multiversion Timestamp CC

- ❖ Idea: Let writers make a “new” copy while readers use an appropriate “old” copy:



- ❖ Readers are always allowed to proceed.
  - But may be blocked until writer commits.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

44

---

---

---

---

---

---

## Multiversion CC (Contd.)

- ❖ Each version of an object has its writer’s TS as its WTS, and the TS of the Xact that most recently read this version as its RTS.
- ❖ Versions are chained backward; we can discard versions that are “too old to be of interest”.
- ❖ Each Xact is classified as Reader or Writer.
  - Writer *may* write some object; Reader never will.
  - Xact declares whether it is a Reader when it begins.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

45

---

---

---

---

---

---

**Reader Xact**

- ❖ For each object to be read:
  - Finds **newest version** with  $WTS < TS(T)$ .  
(Starts with current version in the main segment and chains backward through earlier versions.)
- ❖ Assuming that some version of every object exists from the beginning of time, Reader Xacts are never restarted.
  - However, might block until writer of the appropriate version commits.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

46

---

---

---

---

---

---

**Writer Xact**

- ❖ To read an object, follows reader protocol.
- ❖ To write an object:
  - Finds **newest version V** s.t.  $WTS < TS(T)$ .
  - If  $RTS(V) < TS(T)$ , T makes a copy CV of V, with a pointer to V, with  $WTS(CV) = TS(T)$ ,  $RTS(CV) = TS(T)$ . (Write is buffered until T commits; other Xacts can see TS values but can't read version CV.)
  - Else, reject write.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

47

---

---

---

---

---

---

**Transaction Support in SQL-92**

- ❖ Each transaction has an access mode, a diagnostics size, and an isolation level.

Isolation Level	Dirty Read	Unrepeatable Read	Phantom Problem
Read Uncommitted	Maybe	Maybe	Maybe
Read Committed	No	Maybe	Maybe
Repeatable Reads	No	No	Maybe
Serializable	No	No	No

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

48

---

---

---

---

---

---

## Summary



- ❖ There are several lock-based concurrency control schemes (Strict 2PL, 2PL). Conflicts between transactions can be detected in the dependency graph
- ❖ The lock manager keeps track of the locks issued. Deadlocks can either be prevented or detected.
- ❖ Naïve locking strategies may have the phantom problem

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

49

---

---

---

---

---

---

## Summary (Contd.)



- ❖ Index locking is common, and affects performance significantly.
  - Needed when accessing records via index.
  - Needed for locking logical sets of records (index locking/predicate locking).
- ❖ Tree-structured indexes:
  - Straightforward use of 2PL very inefficient.
  - Bayer-Schkolnick illustrates potential for improvement.
- ❖ In practice, better techniques now known; do record-level, rather than page-level locking.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

50

---

---

---

---

---

---

## Summary (Contd.)



- ❖ Multiple granularity locking reduces the overhead involved in setting locks for nested collections of objects (e.g., a file of pages); should not be confused with tree index locking!
- ❖ Optimistic CC aims to minimize CC overheads in an “optimistic” environment where reads are common and writes are rare.
- ❖ Optimistic CC has its own overheads however; most real systems use locking.
- ❖ SQL-92 provides different isolation levels that control the degree of concurrency

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

51

---

---

---

---

---

---

## *Summary (Contd.)*



- ❖ Timestamp CC is another alternative to 2PL; allows some serializable schedules that 2PL does not (although converse is also true).
- ❖ Ensuring recoverability with Timestamp CC requires ability to block Xacts, which is similar to locking.
- ❖ Multiversion Timestamp CC is a variant which ensures that read-only Xacts are never restarted; they can always read a suitable older version. Additional overhead of version maintenance.

---

---

---

---

---

---

---



# Crash Recovery

## Chapter 18

Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

1

---

---

---

---

---

---



### Review: The ACID properties

- ❖ **A**tomicity: All actions in the Xact happen, or none happen.
- ❖ **C**onsistency: If each Xact is consistent, and the DB starts consistent, it ends up consistent.
- ❖ **I**solation: Execution of one Xact is isolated from that of other Xacts.
- ❖ **D**urability: If a Xact commits, its effects persist.
- ❖ The **Recovery Manager** guarantees Atomicity & Durability.

Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

2

---

---

---

---

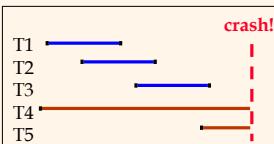
---

---



### Motivation

- ❖ Atomicity:
  - Transactions may abort ("Rollback").
- ❖ Durability:
  - What if DBMS stops running? (Causes?)
- ❖ Desired Behavior after system restarts:
  - T1, T2 & T3 should be durable.
  - T4 & T5 should be aborted (effects not seen).



The diagram illustrates a timeline of transaction execution. Five transactions, T1 through T5, are shown. T1, T2, and T3 are represented by blue horizontal bars. T4 and T5 are represented by red horizontal bars. A vertical dashed red line marks the 'crash!' point. T1 and T2 have completed before the crash. T3 has partially executed, with its bar ending before the crash line. T4 and T5 are still executing at the time of the crash, indicated by their bars extending past the crash line.

Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

3

---

---

---

---

---

---

## Assumptions

- ❖ Concurrency control is in effect.
  - **Strict 2PL**, in particular.
- ❖ Updates are happening “in place”.
  - i.e. data is overwritten on (deleted from) the disk.
- ❖ A simple scheme to guarantee Atomicity & Durability?



---

---

---

---

---

---

## Handling the Buffer Pool

- ❖ Force every write to disk?
  - Poor response time.
  - But provides durability.
- ❖ Steal buffer-pool frames from uncommitted Xacts?
  - If not, poor throughput.
  - If so, how can we ensure atomicity?

	No Steal	Steal
Force	Trivial	
No Force		Desired



---

---

---

---

---

---

## More on Steal and Force

- ❖ **STEAL** (why enforcing Atomicity is hard)
  - *To steal frame F:* Current page in F (say P) is written to disk; some Xact holds lock on P.
    - What if the Xact with the lock on P aborts?
    - Must remember the old value of P at steal time (to support **UNDO**ing the write to page P).
- ❖ **NO FORCE** (why enforcing Durability is hard)
  - What if system crashes before a modified page is written to disk?
  - Write as little as possible, in a convenient place, at commit time, to support **REDO**ing modifications.

---

---

---

---

---

---

## Basic Idea: Logging



- ❖ Record REDO and UNDO information, for every update, in a *log*.
  - Sequential writes to log (put it on a separate disk).
  - Minimal info (diff) written to log, so multiple updates fit in a single log page.
- ❖ Log: An ordered list of REDO/UNDO actions
  - Log record contains:  
 $\langle XID, \text{pageID}, \text{offset}, \text{length}, \text{old data}, \text{new data} \rangle$
  - and additional control info (which we'll see soon).

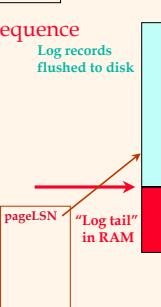
## Write-Ahead Logging (WAL)

- ❖ The Write-Ahead Logging Protocol:
  - ① Must **force** the log record for an update *before* the corresponding data page gets to disk.
  - ② Must **write all log records** for a Xact *before commit*.
- ❖ #1 guarantees Atomicity.
- ❖ #2 guarantees Durability.
- ❖ Exactly how is logging (and recovery!) done?
  - We'll study the ARIES algorithms.

## WAL & the Log



Log records flushed to disk



- ❖ Each log record has a unique **Log Sequence Number (LSN)**.
  - LSNs always increasing.
- ❖ Each **data page** contains a **pageLSN**.
  - The LSN of the most recent *log record* for an update to that page.
- ❖ System keeps track of **flushedLSN**.
  - The max LSN flushed so far.
- ❖ WAL: *Before* a page is written,
  - $\text{pageLSN} \leq \text{flushedLSN}$

## Log Records

Possible log record types:

- ❖ Update

- ❖ Commit

- ❖ Abort

- ❖ End (signifies end of commit or abort)

- ❖ Compensation Log Records (CLRs)

- for UNDO actions

### LogRecord fields:

update records only {  
prevLSN  
XID  
type  
pageID  
length  
offset  
before-image  
after-image

---

---

---

---

---

---

---

## Other Log-Related State

### ❖ Transaction Table:

- One entry per active Xact.
- Contains **XID**, **status** (running/committed/aborted), and **lastLSN**.

### ❖ Dirty Page Table:

- One entry per dirty page in buffer pool.
- Contains **recLSN** -- the LSN of the log record which first caused the page to be dirty.

---

---

---

---

---

---

---

## Normal Execution of an Xact

### ❖ Series of **reads & writes**, followed by **commit** or **abort**.

- We will assume that write is atomic on disk.
  - In practice, additional details to deal with non-atomic writes.

### ❖ Strict 2PL.

### ❖ STEAL, NO-FORCE buffer management, with **Write-Ahead Logging**.

---

---

---

---

---

---

---

## Checkpointing

- Periodically, the DBMS creates a **checkpoint**, in order to minimize the time taken to recover in the event of a system crash. Write to log:
  - begin\_checkpoint** record: Indicates when chkpt began.
  - end\_checkpoint** record: Contains current *Xact table* and *dirty page table*. This is a '**fuzzy checkpoint**'.
    - Other Xacts continue to run; so these tables accurate only as of the time of the **begin\_checkpoint** record.
    - No attempt to force dirty pages to disk; effectiveness of checkpoint limited by oldest unwritten change to a dirty page. (So it's a good idea to periodically flush dirty pages to disk!)
  - Store LSN of chkpt record in a safe place (**master** record).

---

---

---

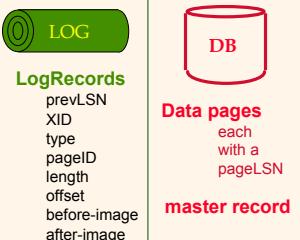
---

---

---

---

## The Big Picture: What's Stored Where



---

---

---

---

---

---

---

## Simple Transaction Abort

- For now, consider an explicit abort of a Xact.
  - No crash involved.
- We want to "play back" the log in reverse order, UNDOing updates.
  - Get **lastLSN** of Xact from Xact table.
  - Can follow chain of log records backward via the **prevLSN** field.
  - Before starting UNDO, write an **Abort log record**.
    - For recovering from crash during UNDO!

---

---

---

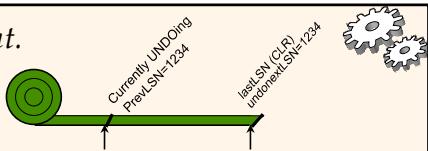
---

---

---

---

## Abort, cont.



- ❖ To perform UNDO, must have a lock on data!
  - No problem!
- ❖ Before restoring old value of a page, write a CLR:
  - You continue logging while you UNDO!!
  - CLR has one extra field: **undonetLSN**
    - Points to the next LSN to undo (i.e. the prevLSN of the record we're currently undoing).
  - CLRs **never** Undo (but they might be Redone when repeating history: guarantees Atomicity!)
- ❖ At end of UNDO, write an “end” log record.

Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

16

---

---

---

---

---

---

## Transaction Commit

- ❖ Write **commit** record to log.
- ❖ All log records up to Xact's **lastLSN** are flushed.
  - Guarantees that **flushedLSN  $\geq$  lastLSN**.
  - Note that log flushes are sequential, synchronous writes to disk.
  - Many log records per log page.
- ❖ **Commit()** returns.
- ❖ Write **end** record to log.

Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

17

---

---

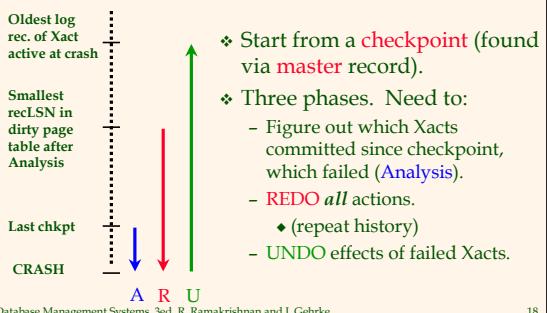
---

---

---

---

## Crash Recovery: Big Picture



Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

18

---

---

---

---

---

---

## Recovery: The Analysis Phase



- ❖ Reconstruct state at checkpoint.
  - via **end\_checkpoint** record.
- ❖ Scan log forward from checkpoint.
  - **End** record: Remove Xact from Xact table.
  - Other records: Add Xact to Xact table, set **lastLSN=LSN**, change Xact status on **commit**.
  - **Update** record: If P not in Dirty Page Table,
    - Add P to D.P.T., set its **recLSN=LSN**.

---

---

---

---

---

---

---

## Recovery: The REDO Phase



- ❖ We **repeat History** to reconstruct state at crash:
  - Reapply **all** updates (even of aborted Xacts!), redo CLRs.
- ❖ Scan forward from log rec containing smallest **recLSN** in D.P.T. For each CLR or update log rec **LSN**, REDO the action unless:
  - Affected page is not in the Dirty Page Table, or
  - Affected page is in D.P.T., but has **recLSN > LSN**, or
  - **pageLSN** (in DB)  $\geq$  **LSN**.
- ❖ To **REDO** an action:
  - Reapply logged action.
  - Set **pageLSN** to **LSN**. No additional logging!

---

---

---

---

---

---

---

## Recovery: The UNDO Phase



**ToUndo**={ l | l a lastLSN of a “loser” Xact}

### Repeat:

- Choose largest LSN among ToUndo.
- If this LSN is a **CLR** and **undonextLSN==NULL**
  - Write an **End** record for this Xact.
- If this LSN is a **CLR**, and **undonextLSN != NULL**
  - Add **undonextLSN** to **ToUndo**
- Else this LSN is an **update**. Undo the update, write a **CLR**, add **prevLSN** to **ToUndo**.

Until **ToUndo** is empty.

---

---

---

---

---

---

---

## Example of Recovery



Xact Table  
lastLSN  
status  
Dirty Page Table  
reclSN  
flushedLSN  
  
ToUndo

LSN	LOG
00	begin_checkpoint
05	end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40	CLR: Undo T1 LSN 10
45	T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART

## Example: Crash During Restart!



Xact Table  
lastLSN  
status  
Dirty Page Table  
reclSN  
flushedLSN  
  
ToUndo

LSN	LOG
00,05	begin_checkpoint, end_checkpoint
10	update: T1 writes P5
20	update T2 writes P3
30	T1 abort
40,45	CLR: Undo T1 LSN 10, T1 End
50	update: T3 writes P1
60	update: T2 writes P5
	CRASH, RESTART
70	CLR: Undo T2 LSN 60
80,85	CLR: Undo T3 LSN 50, T3 end
	CRASH, RESTART
90	CLR: Undo T2 LSN 20, T2 end

## Additional Crash Issues

- ❖ What happens if system crashes during Analysis? During REDO?
- ❖ How do you limit the amount of work in REDO?
  - Flush asynchronously in the background.
  - Watch “hot spots”!
- ❖ How do you limit the amount of work in UNDO?
  - Avoid long-running Xacts.

## *Summary of Logging/Recovery*



- ❖ **Recovery Manager** guarantees Atomicity & Durability.
- ❖ Use WAL to allow STEAL/NO-FORCE w/o sacrificing correctness.
- ❖ LSNs identify log records; linked into backwards chains per transaction (via prevLSN).
- ❖ pageLSN allows comparison of data page and log records.

---

---

---

---

---

---

---

## *Summary, Cont.*



- ❖ **Checkpointing:** A quick way to limit the amount of log to scan on recovery.
- ❖ Recovery works in 3 phases:
  - **Analysis:** Forward from checkpoint.
  - **Redo:** Forward from oldest recLSN.
  - **Undo:** Backward from end to first LSN of oldest Xact alive at crash.
- ❖ Upon Undo, write CLRs.
- ❖ Redo “repeats history”: Simplifies the logic!

---

---

---

---

---

---

---



## Schema Refinement and Normal Forms

### Chapter 19

---

---

---

---

---

---



## The Evils of Redundancy

- ❖ **Redundancy** is at the root of several problems associated with relational schemas:
  - redundant storage, insert/delete/update anomalies
- ❖ Integrity constraints, in particular **functional dependencies**, can be used to identify schemas with such problems and to suggest refinements.
- ❖ Main refinement technique: **decomposition** (replacing ABCD with, say, AB and BCD, or ACD and ABD).
- ❖ Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?

---

---

---

---

---

---



## Functional Dependencies (FDs)

- ❖ A **functional dependency**  $X \rightarrow Y$  holds over relation R if, for every allowable instance  $r$  of R:
  - $t1 \in r, t2 \in r, \pi_X(t1) = \pi_X(t2)$  implies  $\pi_Y(t1) = \pi_Y(t2)$
  - i.e., given two tuples in  $r$ , if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)
- ❖ An FD is a statement about *all* allowable relations.
  - Must be identified based on semantics of application.
  - Given some allowable instance  $r1$  of R, we can check if it violates some FD  $f$ , but we cannot tell if  $f$  holds over R!
- ❖ K is a candidate key for R means that  $K \rightarrow R$ 
  - However,  $K \rightarrow R$  does not require K to be *minimal*!

---

---

---

---

---

---

## Example: Constraints on Entity Set

- ❖ Consider relation obtained from Hourly\_Emps:
  - Hourly\_Emps (*ssn, name, lot, rating, hrly\_wages, hrs\_worked*)
- ❖ Notation: We will denote this relation schema by listing the attributes: SNLRWH
  - This is really the *set* of attributes {S,N,L,R,W,H}.
  - Sometimes, we will refer to all attributes of a relation by using the relation name. (e.g., Hourly\_Emps for SNLRWH)
- ❖ Some FDs on Hourly\_Emps:
  - *ssn* is the key:  $S \rightarrow \text{SNLRWH}$
  - *rating* determines *hrly\_wages*:  $R \rightarrow W$

## Example (Contd.)

- ❖ Problems due to  $R \rightarrow W$  :
  - Update anomaly: Can we change W in just the 1st tuple of SNLRWH?
  - Insertion anomaly: What if we want to insert an employee and don't know the hourly wage for his rating?
  - Deletion anomaly: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

Wages		R	W	
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

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

Will 2 smaller tables be better?

## Reasoning About FDs

- ❖ Given some FDs, we can usually infer additional FDs:
  - $ssn \rightarrow did$ ,  $did \rightarrow lot$  implies  $ssn \rightarrow lot$
- ❖ An FD  $f$  is implied by a set of FDs  $F$  if  $f$  holds whenever all FDs in  $F$  hold.
  - $F^+ = \text{closure of } F$  is the set of all FDs that are implied by  $F$ .
- ❖ Armstrong's Axioms (X, Y, Z are sets of attributes):
  - Reflexivity: If  $X \subseteq Y$ , then  $Y \rightarrow X$
  - Augmentation: If  $X \rightarrow Y$ , then  $XZ \rightarrow YZ$  for any  $Z$
  - Transitivity: If  $X \rightarrow Y$  and  $Y \rightarrow Z$ , then  $X \rightarrow Z$
- ❖ These are *sound* and *complete* inference rules for FDs!

## Reasoning About FDs (Contd.)



- ❖ Couple of additional rules (that follow from AA):
  - **Union:** If  $X \rightarrow Y$  and  $X \rightarrow Z$ , then  $X \rightarrow YZ$
  - **Decomposition:** If  $X \rightarrow YZ$ , then  $X \rightarrow Y$  and  $X \rightarrow Z$
- ❖ Example: Contracts(*cid,sid,jid,did,pid,qty,value*), and:
  - C is the key:  $C \rightarrow CSJDPQV$
  - Project purchases each part using single contract:  $JP \rightarrow C$
  - Dept purchases at most one part from a supplier:  $SD \rightarrow P$
- ❖  $JP \rightarrow C$ ,  $C \rightarrow CSJDPQV$  imply  $JP \rightarrow CSJDPQV$
- ❖  $SD \rightarrow P$  implies  $SDJ \rightarrow JP$
- ❖  $SDJ \rightarrow JP$ ,  $JP \rightarrow CSJDPQV$  imply  $SDJ \rightarrow CSJDPQV$

Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

7

## Reasoning About FDs (Contd.)



- ❖ Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in # attrs!)
- ❖ Typically, we just want to check if a given FD  $X \rightarrow Y$  is in the closure of a set of FDs F. An efficient check:
  - Compute **attribute closure** of X (denoted  $X^+$ ) wrt F:
    - Set of all attributes A such that  $X \rightarrow A$  is in  $F^+$
    - There is a linear time algorithm to compute this.
  - Check if Y is in  $X^+$
- ❖ Does  $F = \{A \rightarrow B, B \rightarrow C, C D \rightarrow E\}$  imply  $A \rightarrow E$ ?
  - i.e., is  $A \rightarrow E$  in the closure  $F^+$ ? Equivalently, is E in  $A^+$ ?

Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

8

## Normal Forms



- ❖ Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- ❖ If a relation is in a certain **normal form** (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.
- ❖ Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - **No FDs hold:** There is no redundancy here.
    - **Given  $A \rightarrow B$ :** Several tuples could have the same A value, and if so, they'll all have the same B value!

Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

9

## Boyce-Codd Normal Form (BCNF)



- ❖ Reln R with FDs  $F$  is in BCNF if, for all  $X \rightarrow A$  in  $F^+$ 
  - $A \in X$  (called a *trivial* FD), or
  - $X$  contains a key for R.
- ❖ In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints.
  - No dependency in R that can be predicted using FDs alone.
  - If we are shown two tuples that agree upon the X value, we cannot infer the A value in one tuple from the A value in the other.
  - If example relation is in BCNF, the 2 tuples must be identical (since X is a key).

X	Y	A
x	y1	a
x	y2	?

---

---

---

---

---

---

---

## Third Normal Form (3NF)



- ❖ Reln R with FDs  $F$  is in 3NF if, for all  $X \rightarrow A$  in  $F^+$ 
  - $A \in X$  (called a *trivial* FD), or
  - $X$  contains a key for R, or
  - A is part of some key for R.
- ❖ *Minimality* of a key is crucial in third condition above!
- ❖ If R is in BCNF, obviously in 3NF.
- ❖ If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no "good" decomp, or performance considerations).
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

---

---

---

---

---

---

---

## What Does 3NF Achieve?



- ❖ If 3NF violated by  $X \rightarrow A$ , one of the following holds:
  - $X$  is a subset of some key K
    - We store  $(X, A)$  pairs redundantly.
  - $X$  is not a proper subset of any key.
    - There is a chain of FDs  $K \rightarrow X \rightarrow A$ , which means that we cannot associate an X value with a K value unless we also associate an A value with an X value.
- ❖ **But:** even if reln is in 3NF, these problems could arise.
  - e.g., Reserves SBDC,  $S \rightarrow C$ ,  $C \rightarrow S$  is in 3NF, but for each reservation of sailor S, same (S, C) pair is stored.
- ❖ Thus, 3NF is indeed a compromise relative to BCNF.

---

---

---

---

---

---

---

## Decomposition of a Relation Scheme

- ❖ Suppose that relation R contains attributes  $A_1 \dots A_n$ .  
A *decomposition* of R consists of replacing R by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of R (and no attributes that do not appear in R), and
  - Every attribute of R appears as an attribute of one of the new relations.
- ❖ Intuitively, decomposing R means we will store instances of the relation schemes produced by the decomposition, instead of instances of R.
- ❖ E.g., Can decompose SNLRWH into SNLRH and RW.

Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

13

---

---

---

---

---

---

---

---

## Example Decomposition

- ❖ Decompositions should be used only when needed.
  - SNLRWH has FDs  $S \rightarrow SNLRWH$  and  $R \rightarrow W$
  - Second FD causes violation of 3NF; W values repeatedly associated with R values. Easiest way to fix this is to create a relation RW to store these associations, and to remove W from the main schema:
    - i.e., we decompose SNLRWH into SNLRH and RW
- ❖ The information to be stored consists of SNLRWH tuples. If we just store the projections of these tuples onto SNLRH and RW, are there any potential problems that we should be aware of?

Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

14

---

---

---

---

---

---

---

---

## Problems with Decompositions

- ❖ There are three potential problems to consider:
  - Some queries become more expensive.
    - e.g., How much did sailor Joe earn? ( $\text{salary} = W \cdot H$ )
  - Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
    - Fortunately, not in the SNLRWH example.
  - Checking some dependencies may require joining the instances of the decomposed relations.
    - Fortunately, not in the SNLRWH example.
- ❖ Tradeoff. Must consider these issues vs. redundancy.

Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

15

---

---

---

---

---

---

---

---

## Lossless Join Decompositions



- ❖ Decomposition of R into X and Y is ***lossless-join*** w.r.t. a set of FDs F if, for every instance r that satisfies F:
  - $\pi_X(r) \bowtie \pi_Y(r) = r$
- ❖ It is always true that  $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$ 
  - In general, the other direction does not hold! If it does, the decomposition is lossless-join.
- ❖ Definition extended to decomposition into 3 or more relations in a straightforward way.
- ❖ *It is essential that all decompositions used to deal with redundancy be lossless! (Avoids Problem (2).)*

## More on Lossless Join



- ❖ The decomposition of R into X and Y is ***lossless-join wrt F*** if and only if the closure of F contains:
  - $X \cap Y \rightarrow X$ , or
  - $X \cap Y \rightarrow Y$
- ❖ In particular, the decomposition of R into UV and R - V is lossless-join if  $U \rightarrow V$  holds over R.

A	B	C
1	2	3
4	5	6
7	2	8

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

## Dependency Preserving Decomposition



- ❖ Consider CSJDPQV, C is key,  $JP \rightarrow C$  and  $SD \rightarrow P$ .
  - BCNF decomposition: CSJDQV and SDP
  - Problem: Checking  $JP \rightarrow C$  requires a join!
- ❖ **Dependency preserving decomposition (Intuitive):**
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, on Y and on Z, then all FDs that were given to hold on R must also hold. (*Avoids Problem (3)*)
- ❖ ***Projection of set of FDs F:*** If R is decomposed into X, ... projection of F onto X (denoted  $F_X$ ) is the set of FDs  $U \rightarrow V$  in  $F^*$  (closure of F) such that  $U, V$  are in X.

## Dependency Preserving Decompositions (Contd.)

- ❖ Decomposition of R into X and Y is dependency preserving if  $(F_X \cup F_Y)^+ = F^+$ 
  - i.e., if we consider only dependencies in the closure  $F^+$  that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in  $F^+$ .
- ❖ Important to consider  $F^+$ , not  $F$ , in this definition:
  - ABC,  $A \rightarrow B$ ,  $B \rightarrow C$ ,  $C \rightarrow A$ , decomposed into AB and BC.
  - Is this dependency preserving? Is  $C \rightarrow A$  preserved?????
- ❖ Dependency preserving does not imply lossless join:
  - ABC,  $A \rightarrow B$ , decomposed into AB and BC.
- ❖ And vice-versa! (Example?)

Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

19



---

---

---

---

---

---

## Decomposition into BCNF

- ❖ Consider relation R with FDs F. If  $X \rightarrow Y$  violates BCNF, decompose R into R - Y and XY.
  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
  - e.g., CSJDPQV, key C,  $JP \rightarrow C$ ,  $SD \rightarrow P$ ,  $J \rightarrow S$
  - To deal with  $SD \rightarrow P$ , decompose into SDP, CSJDQV.
  - To deal with  $J \rightarrow S$ , decompose CSJDQV into JS and CJDQV.
- ❖ In general, several dependencies may cause violation of BCNF. The order in which we ``deal with'' them could lead to very different sets of relations!

Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

20



---

---

---

---

---

---

## BCNF and Dependency Preservation

- ❖ In general, there may not be a dependency preserving decomposition into BCNF.
  - e.g., CSZ,  $CS \rightarrow Z$ ,  $Z \rightarrow C$
  - Can't decompose while preserving 1st FD; not in BCNF.
- ❖ Similarly, decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs  $JP \rightarrow C$ ,  $SD \rightarrow P$  and  $J \rightarrow S$ ).
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
  - JPC tuples stored only for checking FD! (*Redundancy!*)

Database Management Systems, 3ed, R. Ramakrishnan and J. Gehrke

21



---

---

---

---

---

---

## Decomposition into 3NF



- ❖ Obviously, the algorithm for lossless join decom into BCNF can be used to obtain a lossless join decom into 3NF (typically, can stop earlier).
- ❖ To ensure dependency preservation, one idea:
  - If  $X \rightarrow Y$  is not preserved, add relation XY.
  - Problem is that XY may violate 3NF! e.g., consider the addition of CJP to 'preserve'  $JP \rightarrow C$ . What if we also have  $J \rightarrow C$ ?
- ❖ Refinement: Instead of the given set of FDs F, use a *minimal cover for F*.

---

---

---

---

---

---

## Minimal Cover for a Set of FDs



- ❖ Minimal cover G for a set of FDs F:
  - Closure of F = closure of G.
  - Right hand side of each FD in G is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- ❖ Intuitively, every FD in G is needed, and ``*as small as possible*'' in order to get the same closure as F.
- ❖ e.g.,  $A \rightarrow B$ ,  $ABCD \rightarrow E$ ,  $EF \rightarrow GH$ ,  $ACDF \rightarrow EG$  has the following minimal cover:
  - $A \rightarrow B$ ,  $ACD \rightarrow E$ ,  $EF \rightarrow G$  and  $EF \rightarrow H$
- ❖ M.C.  $\rightarrow$  Lossless-Join, Dep. Pres. Decomp!!! (in book)

---

---

---

---

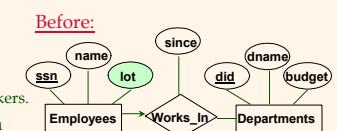
---

---

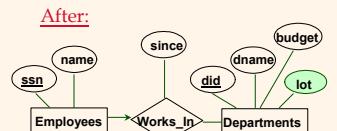
## Refining an ER Diagram



- ❖ 1st diagram translated:  
Workers(S,N,L,D,S)  
Departments(D,M,B)



- Lots associated with workers.
- ❖ Suppose all workers in a dept are assigned the same lot:  $D \rightarrow L$



- ❖ Redundancy; fixed by:  
Workers2(S,N,D,S)  
Dept\_Lots(D,L)
- ❖ Can fine-tune this:  
Workers2(S,N,D,S)  
Departments(D,M,B,L)

---

---

---

---

---

---

## *Summary of Schema Refinement*



- ❖ If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in BCNF is a good heuristic.
- ❖ If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  - Must consider whether all FDs are preserved. If a lossless-join, dependency preserving decomposition into BCNF is not possible (or unsuitable, given typical queries), should consider decomposition into 3NF.
  - Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind.

---

---

---

---

---

---

---

---



## Physical Database Design

### Chapter 16, Part A

---

---

---

---

---

---



## Overview

- ❖ After ER design, schema refinement, and the definition of views, we have the *conceptual* and *external* schemas for our database.
- ❖ The next step is to choose indexes, make clustering decisions, and to refine the conceptual and external schemas (if necessary) to meet performance goals.
- ❖ We must begin by understanding the workload:
  - The most important queries and how often they arise.
  - The most important updates and how often they arise.
  - The desired performance for these queries and updates.

---

---

---

---

---

---



## Decisions to Make

- ❖ What indexes should we create?
  - Which relations should have indexes? What field(s) should be the search key? Should we build several indexes?
- ❖ For each index, what kind of an index should it be?
  - Clustered? Hash/tree?
- ❖ Should we make changes to the conceptual schema?
  - Consider alternative normalized schemas? (Remember, there are many choices in decomposing into BCNF, etc.)
  - Should we ``undo'' some decomposition steps and settle for a lower normal form? (*Denormalization*.)
  - Horizontal partitioning, replication, views ...

---

---

---

---

---

---

## *Index Selection for Joins*



- ❖ When considering a join condition:
  - Hash index on inner is very good for Index Nested Loops.
    - Should be clustered if join column is not key for inner, and inner tuples need to be retrieved.
  - *Clustered* B+ tree on join column(s) good for Sort-Merge.

---

---

---

---

---

---

---

---

## *Example 1*

```
SELECT E.ename, D.mgr
FROM Emp E, Dept D
WHERE D.dname='Toy' AND E.dno=D.dno
```



- ❖ Hash index on *D.dname* supports ‘Toy’ selection.
  - Given this, index on *D.dno* is not needed.
- ❖ Hash index on *E.dno* allows us to get matching (inner) Emp tuples for each selected (outer) Dept tuple.
- ❖ What if WHERE included: “... AND *E.age*=25” ?
  - Could retrieve Emp tuples using index on *E.age*, then join with Dept tuples satisfying *dname* selection. Comparable to strategy that used *E.dno* index.
  - So, if *E.age* index is already created, this query provides much less motivation for adding an *E.dno* index.

---

---

---

---

---

---

---

---

## *Example 2*

```
SELECT E.ename, D.mgr
FROM Emp E, Dept D
WHERE E.sal BETWEEN 10000 AND 20000
AND E.hobby='Stamps' AND E.dno=D.dno
```



- ❖ Clearly, Emp should be the outer relation.
  - Suggests that we build a hash index on *D.dno*.
- ❖ What index should we build on Emp?
  - B+ tree on *E.sal* could be used, OR an index on *E.hobby* could be used. Only one of these is needed, and which is better depends upon the selectivity of the conditions.
    - As a rule of thumb, equality selections more selective than range selections.
- ❖ As both examples indicate, our choice of indexes is guided by the plan(s) that we expect an optimizer to consider for a query. *Have to understand optimizers!*

---

---

---

---

---

---

---

---

## Clustering and Joins

```
SELECT E.ename, Dmgr
FROM Emp E, Dept D
WHERE D.dname='Toy' AND E.dno=D.dno
```

- ❖ Clustering is especially important when accessing inner tuples in INL.
  - Should make index on *E.dno* clustered.
- ❖ Suppose that the WHERE clause is instead:  
*WHERE E.hobby='Stamps' AND E.dno=D.dno*
  - If many employees collect stamps, Sort-Merge join may be worth considering. A *clustered* index on *D.dno* would help.
- ❖ Summary: Clustering is useful whenever many tuples are to be retrieved.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

7

---

---

---

---

---

---

---

## Tuning the Conceptual Schema

- ❖ The choice of conceptual schema should be guided by the workload, in addition to redundancy issues:
  - We may settle for a 3NF schema rather than BCNF.
  - Workload may influence the choice we make in decomposing a relation into 3NF or BCNF.
  - We may further decompose a BCNF schema!
  - We might *denormalize* (i.e., undo a decomposition step), or we might add fields to a relation.
  - We might consider *horizontal decompositions*.
- ❖ If such changes are made after a database is in use, called *schema evolution*; might want to mask some of these changes from applications by defining *views*.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

8

---

---

---

---

---

---

---

## Example Schemas

```
Contracts (Cid, Sid, Jid, Did, Pid, Qty, Val)
Depts (Did, Budget, Report)
Suppliers (Sid, Address)
Parts (Pid, Cost)
Projects (Jid, Mgr)
```

- ❖ We will concentrate on Contracts, denoted as CSJDPQV. The following ICs are given to hold:  
 $JP \rightarrow C$ ,  $SD \rightarrow P$ ,  $C$  is the primary key.
  - What are the candidate keys for CSJDPQV?
  - What normal form is this relation schema in?

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

9

---

---

---

---

---

---

---

## Settling for 3NF vs BCNF



- ❖ CSJDPQV can be decomposed into SDP and CSJDQV, and both relations are in BCNF. (Which FD suggests that we do this?)
  - Lossless decomposition, but not dependency-preserving.
  - Adding CJP makes it dependency-preserving as well.
- ❖ Suppose that this query is very important:
  - *Find the number of copies Q of part P ordered in contract C.*
  - Requires a join on the decomposed schema, but can be answered by a scan of the original relation CSJDPQV.
  - Could lead us to settle for the 3NF schema CSJDPQV.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

10

---

---

---

---

---

---

---

## Denormalization



- ❖ Suppose that the following query is important:
  - *Is the value of a contract less than the budget of the department?*
- ❖ To speed up this query, we might add a field *budget* B to Contracts.
  - This introduces the FD  $D \rightarrow B$  wrt Contracts.
  - Thus, Contracts is no longer in 3NF.
- ❖ We might choose to modify Contracts thus if the query is sufficiently important, and we cannot obtain adequate performance otherwise (i.e., by adding indexes or by choosing an alternative 3NF schema.)

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

11

---

---

---

---

---

---

---

## Choice of Decompositions



- ❖ There are 2 ways to decompose CSJDPQV into BCNF:
  - SDP and CSJDQV; lossless-join but not dep-preserving.
  - SDP, CSJDQV and CJP; dep-preserving as well.
- ❖ The difference between these is really the cost of enforcing the FD  $JP \rightarrow C$ .
  - 2nd decomposition: Index on JP on relation CJP.
  - 1st:

```
CREATE ASSERTION CheckDep
CHECK (NOT EXISTS (SELECT *
FROM PartInfo P,ContractInfo C
WHERE P.sid=C.sid AND P.did=C.did
GROUP BY C.jid, P.pid
HAVING COUNT (C.cid) > 1))
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

12

---

---

---

---

---

---

---

## Choice of Decompositions (Contd.)



- ❖ The following ICs were given to hold:  
 $JP \rightarrow C$ ,  $SD \rightarrow P$ ,  $C$  is the primary key.
- ❖ Suppose that, in addition, a given supplier always charges the same price for a given part:  $SPQ \rightarrow V$ .
- ❖ If we decide that we want to decompose CSJDPQV into BCNF, we now have a third choice:
  - Begin by decomposing it into SPQV and CSJDPQ.
  - Then, decompose CSJDPQ (not in 3NF) into SDP, CSJDQ.
  - This gives us the lossless-join decomp: SPQV, SDP, CSJDQ.
  - To preserve  $JP \rightarrow C$ , we can add CJP, as before.
- ❖ Choice: { SPQV, SDP, CSJDQ } or { SDP, CSJDQV } ?

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

13

---

---

---

---

---

---

---

## Decomposition of a BCNF Relation



- ❖ Suppose that we choose { SDP, CSJDQV }. This is in BCNF, and there is no reason to decompose further (assuming that all known ICs are FDs).
- ❖ However, suppose that these queries are important:
  - Find the contracts held by supplier S.
  - Find the contracts that department D is involved in.
- ❖ Decomposing CSJDQV further into CS, CD and CJQV could speed up these queries. (Why?)
- ❖ On the other hand, the following query is slower:
  - Find the total value of all contracts held by supplier S.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

14

---

---

---

---

---

---

---

## Horizontal Decompositions



- ❖ Our definition of decomposition: Relation is replaced by a collection of relations that are *projections*. Most important case.
- ❖ Sometimes, might want to replace relation by a collection of relations that are *selections*.
  - Each new relation has same schema as the original, but a subset of the rows.
  - Collectively, new relations contain all rows of the original. Typically, the new relations are disjoint.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

15

---

---

---

---

---

---

---

## Horizontal Decompositions (Contd.)

- ❖ Suppose that contracts with value > 10000 are subject to different rules. This means that queries on Contracts will often contain the condition  $val > 10000$ .
- ❖ One way to deal with this is to build a clustered B+ tree index on the *val* field of Contracts.
- ❖ A second approach is to replace contracts by two new relations: LargeContracts and SmallContracts, with the same attributes (CSJDPQV).
  - Performs like index on such queries, but no index overhead.
  - Can build clustered indexes on other attributes, in addition!

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

16

---

---

---

---

---

---

---

---

## Masking Conceptual Schema Changes

```
CREATE VIEW Contracts(cid, sid, jid, did, pid, qty, val)
AS SELECT *
FROM LargeContracts
UNION
SELECT *
FROM SmallContracts
```

- ❖ The replacement of Contracts by LargeContracts and SmallContracts can be masked by the view.
- ❖ However, queries with the condition  $val > 10000$  must be asked wrt LargeContracts for efficient execution: so users concerned with performance have to be aware of the change.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

17

---

---

---

---

---

---

---

---

## Tuning Queries and Views

- ❖ If a query runs slower than expected, check if an index needs to be re-built, or if statistics are too old.
- ❖ Sometimes, the DBMS may not be executing the plan you had in mind. Common areas of weakness:
  - Selections involving null values.
  - Selections involving arithmetic or string expressions.
  - Selections involving OR conditions.
  - Lack of evaluation features like index-only strategies or certain join methods or poor size estimation.
- ❖ Check the plan that is being used! Then adjust the choice of indexes or rewrite the query / view.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

18

---

---

---

---

---

---

---

---

## Rewriting SQL Queries



- ❖ Complicated by interaction of:
  - NULLs, duplicates, aggregation, subqueries.

❖ **Guideline:** Use only one "query block", if possible.

```
SELECT DISTINCT * SELECT DISTINCT S.*
 FROM Sailors S FROM Sailors S,
 WHERE S.sname IN YoungSailors Y
 (SELECT Y.sname WHERE S.sname = Y.sname
 FROM YoungSailors Y)
```

❖ Not always possible ...

```
SELECT * SELECT S.*
 FROM Sailors S FROM Sailors S,
 WHERE S.sname IN YoungSailors Y
 (SELECT DISTINCT Y.sname WHERE S.sname = Y.sname
 FROM YoungSailors Y)
```

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

19

---

---

---

---

---

---

---

---

---

## The Notorious COUNT Bug



```
SELECT dname FROM Department D
WHERE D.num_emps >
 (SELECT COUNT(*) FROM Employee E
 WHERE D.building = E.building)

CREATE VIEW Temp (empcount, building) AS
 SELECT COUNT(*), E.building
 FROM Employee E
 GROUP BY E.building

SELECT dname
 FROM Department D,Temp
 WHERE D.building = Temp.building
 AND D.num_emps > Temp.empcount;
```

❖ What happens when Employee is empty??

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

20

---

---

---

---

---

---

---

---

---

## Summary on Unnesting Queries



- ❖ DISTINCT at top level: *Can ignore duplicates.*
  - Can sometimes infer DISTINCT at top level! (e.g. subquery clause matches at most one tuple)
- ❖ DISTINCT in subquery w/o DISTINCT at top: *Hard to convert.*
- ❖ Subqueries inside OR: *Hard to convert.*
- ❖ ALL subqueries: *Hard to convert.*
  - EXISTS and ANY are just like IN.
- ❖ Aggregates in subqueries: *Tricky.*
- ❖ **Good news:** Some systems now rewrite under the covers (e.g. DB2).

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

21

---

---

---

---

---

---

---

---

---

## More Guidelines for Query Tuning

- ❖ Minimize the use of DISTINCT: don't need it if duplicates are acceptable, or if answer contains a key.
- ❖ Minimize the use of GROUP BY and HAVING:

```
SELECT MIN (E.age)
FROM Employee E
GROUP BY E.dno
HAVING E.dno=102
```

```
SELECT MIN (E.age)
FROM Employee E
WHERE E.dno=102
```

- ❖ Consider DBMS use of index when writing arithmetic expressions:  $E.age=2*D.age$  will benefit from index on  $E.age$ , but might not benefit from index on  $D.age$ !

---

---

---

---

---

---

---

---

---

## Guidelines for Query Tuning (Contd.)

- ❖ Avoid using intermediate relations:

vs.

```
SELECT E.dno, AVG(E.sal)
FROM Emp E, Dept D
WHERE E.dno=D.dno
AND D.mgrname='Joe'
GROUP BY E.dno
```

```
SELECT * INTO Temp
FROM Emp E, Dept D
WHERE E.dno=D.dno
AND D.mgrname='Joe'
and
SELECT T.dno, AVG(T.sal)
FROM Temp T
GROUP BY T.dno
```

- ❖ Does not materialize the intermediate reln Temp.
- ❖ If there is a dense B+ tree index on  $<dno, sal>$ , an index-only plan can be used to avoid retrieving Emp tuples in the second query!

---

---

---

---

---

---

---

---

---

## Summary

- ❖ Database design consists of several tasks: *requirements analysis, conceptual design, schema refinement, physical design and tuning*.
  - In general, have to go back and forth between these tasks to refine a database design, and decisions in one task can influence the choices in another task.
- ❖ Understanding the nature of the *workload* for the application, and the performance goals, is essential to developing a good design.
  - What are the important queries and updates? What attributes/relations are involved?

---

---

---

---

---

---

---

---

---

## Summary



- ❖ The conceptual schema should be refined by considering performance criteria and workload:
  - May choose 3NF or lower normal form over BCNF.
  - May choose among alternative decompositions into BCNF (or 3NF) based upon the workload.
  - May *denormalize*, or undo some decompositions.
  - May decompose a BCNF relation further!
  - May choose a *horizontal decomposition* of a relation.
  - Importance of dependency-preservation based upon the dependency to be preserved, and the cost of the IC check.
    - Can add a relation to ensure dep-preservation (for 3NF, not BCNF!); or else, can check dependency using a join.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

25

---

---

---

---

---

---

---

---

## Summary (Contd.)



- ❖ Over time, indexes have to be fine-tuned (dropped, created, re-built, ...) for performance.
  - Should determine the plan used by the system, and adjust the choice of indexes appropriately.
- ❖ System may still not find a good plan:
  - Only left-deep plans considered!
  - Null values, arithmetic conditions, string expressions, the use of ORS, etc. can confuse an optimizer.
- ❖ So, may have to rewrite the query/view:
  - Avoid nested queries, temporary relations, complex conditions, and operations like DISTINCT and GROUP BY.

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke

26

---

---

---

---

---

---

---

---



## Security and Authorization

### Chapter 21

---

---

---

---

---

---



## Introduction to DB Security

- ❖ **Secrecy:** Users should not be able to see things they are not supposed to.
  - E.g., A student can't see other students' grades.
- ❖ **Integrity:** Users should not be able to modify things they are not supposed to.
  - E.g., Only instructors can assign grades.
- ❖ **Availability:** Users should be able to see and modify things they are allowed to.

---

---

---

---

---

---



## Access Controls

- ❖ A **security policy** specifies who is authorized to do what.
- ❖ A **security mechanism** allows us to enforce a chosen security policy.
- ❖ Two main mechanisms at the DBMS level:
  - Discretionary access control
  - Mandatory access control

---

---

---

---

---

---

## Discretionary Access Control



- ❖ Based on the concept of access rights or **privileges** for objects (tables and views), and mechanisms for giving users privileges (and revoking privileges).
- ❖ Creator of a table or a view automatically gets all privileges on it.
  - DBMS keeps track of who subsequently gains and loses privileges, and ensures that only requests from users who have the necessary privileges (at the time the request is issued) are allowed.

---

---

---

---

---

---

---

## GRANT Command



**GRANT privileges ON object TO users [WITH GRANT OPTION]**

- ❖ The following **privileges** can be specified:
  - ❖ SELECT: Can read all columns (including those added later via ALTER TABLE command).
  - ❖ INSERT(col-name): Can insert tuples with non-null or non-default values in this column.
    - ❖ INSERT means same right with respect to all columns.
  - ❖ DELETE: Can delete tuples.
  - ❖ REFERENCES (col-name): Can define foreign keys (in other tables) that refer to this column.
- ❖ If a user has a privilege with the **GRANT OPTION**, can pass privilege on to other users (with or without passing on the **GRANT OPTION**).
- ❖ Only owner can execute CREATE, ALTER, and DROP.

---

---

---

---

---

---

---

## GRANT and REVOKE of Privileges



- ❖ GRANT INSERT, SELECT ON Sailors TO Horatio
  - Horatio can query Sailors or insert tuples into it.
- ❖ GRANT DELETE ON Sailors TO Yuppy WITH GRANT OPTION
  - Yuppy can delete tuples, and also authorize others to do so.
- ❖ GRANT UPDATE (*rating*) ON Sailors TO Dustin
  - Dustin can update (only) the *rating* field of Sailors tuples.
- ❖ GRANT SELECT ON ActiveSailors TO Guppy, Yuppy
  - This does NOT allow the 'uppies to query Sailors directly!
- ❖ **REVOKE:** When a privilege is revoked from X, it is also revoked from all users who got it *solely* from X.

---

---

---

---

---

---

---

## *GRANT/REVOKE on Views*



- ❖ If the creator of a view loses the SELECT privilege on an underlying table, the view is dropped!
- ❖ If the creator of a view loses a privilege held with the grant option on an underlying table, (s)he loses the privilege on the view as well; so do users who were granted that privilege on the view!

---

---

---

---

---

---

---

## *Views and Security*



- ❖ Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given ActiveSailors, but not Sailors or Reserves, we can find sailors who have a reservation, but not the bid's of boats that have been reserved.
- ❖ Creator of view has a privilege on the view if (s)he has the privilege on all underlying tables.
- ❖ Together with GRANT/REVOKE commands, views are a very powerful access control tool.

---

---

---

---

---

---

---

## *Role-Based Authorization*



- ❖ In SQL-92, privileges are actually assigned to **authorization ids**, which can denote a single user or a group of users.
- ❖ In SQL:1999 (and in many current systems), privileges are assigned to **roles**.
  - Roles can then be granted to users and to other roles.
  - Reflects how real organizations work.
  - Illustrates how standards often catch up with "de facto" standards embodied in popular systems.

---

---

---

---

---

---

---

## Security to the Level of a Field!



- ❖ Can create a view that only returns one field of one tuple. (How?)
- ❖ Then grant access to that view accordingly.
- ❖ Allows for *arbitrary* granularity of control, *but*:
  - Clumsy to specify, though this can be hidden under a good UI
  - Performance is unacceptable if we need to define field-granularity access frequently. (Too many view creations and look-ups.)

---

---

---

---

---

---

---

## Internet-Oriented Security



- ❖ **Key Issues: User authentication and trust.**
  - When DB must be accessed from a secure location, password-based schemes are usually adequate.
- ❖ For access over an external network, trust is hard to achieve.
  - If someone with Sam's credit card wants to buy from you, how can you be sure it is not someone who stole his card?
  - How can Sam be sure that the screen for entering his credit card information is indeed yours, and not some rogue site spoofing you (to steal such information)? How can he be sure that sensitive information is not "sniffed" while it is being sent over the network to you?
- ❖ *Encryption* is a technique used to address these issues.

---

---

---

---

---

---

---

## Encryption



- ❖ "Masks" data for secure transmission or storage
  - Encrypt(data, encryption key) = encrypted data
  - Decrypt(encrypted data, decryption key) = original data
  - Without decryption key, the encrypted data is meaningless gibberish
- ❖ **Symmetric Encryption:**
  - Encryption key = decryption key; all authorized users know decryption key (a weakness).
  - DES, used since 1977, has 56-bit key; AES has 128-bit (optionally, 192-bit or 256-bit) key
- ❖ **Public-Key Encryption:** Each user has two keys:
  - User's public encryption key: Known to all
  - Decryption key: Known only to this user
  - Used in RSA scheme (Turing Award!)

---

---

---

---

---

---

---

## RSA Public-Key Encryption



- ❖ Let the data be an integer  $I$
- ❖ Choose a large ( $>> I$ ) integer  $L = p * q$ 
  - $p, q$  are large, say 1024-bit, distinct prime numbers
- ❖ Encryption: Choose a random number  $1 < e < L$  that is relatively prime to  $(p-1) * (q-1)$ 
  - Encrypted data  $S = I^e \text{ mod } L$
- ❖ Decryption key  $d$ : Chosen so that
  - $d * e = 1 \text{ mod } ((p-1) * (q-1))$
  - We can then show that  $I = S^d \text{ mod } L$
- ❖ It turns out that the roles of  $e$  and  $d$  can be reversed; so they are simply called the public and private keys

---

---

---

---

---

---

---

## Certifying Servers: SSL, SET



- ❖ If Amazon distributes their public key, Sam's browser will encrypt his order using it.
  - So, only Amazon can decipher the order, since no one else has Amazon's private key.
- ❖ But how can Sam (or his browser) know that the public key for Amazon is genuine? The SSL protocol covers this.
  - Amazon contracts with, say, Verisign, to issue a certificate <Verisign, Amazon, amazon.com, public-key>
  - This certificate is stored in encrypted form, encrypted with Verisign's *private* key, known only to Verisign.
  - Verisign's public key is known to all browsers, which can therefore decrypt the certificate and obtain Amazon's public key and be confident that it is genuine.
  - The browser then generates a temporary *session key*, encodes it using Amazon's public key, and sends it to Amazon.
  - All subsequent msgs between the browser and Amazon are encoded using symmetric encryption (e.g., DES), which is more efficient than public-key encryption.
- ❖ What if Sam doesn't trust Amazon with his credit card information?
  - Secure Electronic Transaction protocol: 3-way communication between Amazon, Sam, and a trusted server, e.g., Visa.

---

---

---

---

---

---

---

## Authenticating Users



- ❖ Amazon can simply use password authentication, i.e., ask Sam to log into his Amazon account.
  - Done after SSL is used to establish a session key, so that the transmission of the password is secure!
  - Amazon is still at risk if Sam's card is stolen and his password is hacked. Business risk ...
- ❖ Digital Signatures:
  - Sam encrypts the order using his *private* key, then encrypts the result using Amazon's public key.
  - Amazon decrypts the msg with their private key, and then decrypts the result using Sam's public key, which yields the original order!
  - Exploits interchangeability of public/private keys for encryption/decryption
  - Now, no one can forge Sam's order, and Sam cannot claim that someone else forged the order.

---

---

---

---

---

---

---

## Mandatory Access Control



- ❖ Based on system-wide policies that cannot be changed by individual users.
  - Each **DB object** is assigned a **security class**.
  - Each **subject** (user or user program) is assigned a **clearance** for a security class.
  - Rules based on security classes and clearances govern who can read/write which objects.
- ❖ Most commercial systems do not support mandatory access control. Versions of some DBMSs do support it; used for specialized (e.g., military) applications.

---

---

---

---

---

---

## Why Mandatory Control?



- ❖ Discretionary control has some flaws, e.g., the *Trojan horse* problem:
  - Dick creates Horsie and gives INSERT privileges to Justin (who doesn't know about this).
  - Dick modifies the code of an application program used by Justin to additionally write some secret data to table Horsie.
  - Now, Justin can see the secret info.
- ❖ The modification of the code is beyond the DBMSs control, but it can try and prevent the use of the database as a **channel** for secret information.

---

---

---

---

---

---

## Bell-LaPadula Model



- ❖ Objects (e.g., tables, views, tuples)
- ❖ Subjects (e.g., users, user programs)
- ❖ Security classes:
  - Top secret (TS), secret (S), confidential (C), unclassified (U): TS > S > C > U
- ❖ Each object and subject is assigned a class.
  - Subject S can read object O only if class(S)  $\geq$  class(O) (**Simple Security Property**)
  - Subject S can write object O only if class(S)  $\leq$  class(O) (**\*-Property**)

---

---

---

---

---

---

## Intuition



- ❖ Idea is to ensure that information can never flow from a higher to a lower security level.
- ❖ E.g., If Dick has security class C, Justin has class S, and the secret table has class S:
  - Dick's table, Horsie, has Dick's clearance, C.
  - Justin's application has his clearance, S.
  - So, the program cannot write into table Horsie.
- ❖ The mandatory access control rules are applied in addition to any discretionary controls that are in effect.

---

---

---

---

---

---

## Multilevel Relations



bid	bname	color	class
101	Salsa	Red	S
102	Pinto	Brown	C

- ❖ Users with S and TS clearance will see both rows; a user with C will only see the 2<sup>nd</sup> row; a user with U will see no rows.
- ❖ If user with C tries to insert <101,Pasta,Blue,C>:
  - Allowing insertion violates key constraint
  - Disallowing insertion tells user that there is another object with key 101 that has a class > C!
  - Problem resolved by treating class field as part of key.

---

---

---

---

---

---

## Statistical DB Security



- ❖ Statistical DB: Contains information about individuals, but allows only aggregate queries (e.g., average age, rather than Joe's age).
- ❖ New problem: It may be possible to **infer** some secret information!
  - E.g., If I know Joe is the oldest sailor, I can ask "How many sailors are older than X?" for different values of X until I get the answer 1; this allows me to infer Joe's age.
- ❖ Idea: Insist that each query must involve at least N rows, for some N. Will this work? (No!)

---

---

---

---

---

---

## Why Minimum N is Not Enough



- ❖ By asking “How many sailors older than X?” until the system rejects the query, can identify a set of N sailors, including Joe, that are older than X; let X=55 at this point.
- ❖ Next, ask “What is the sum of ages of sailors older than X?” Let result be S1.
- ❖ Next, ask “What is sum of ages of sailors other than Joe who are older than X, plus my age?” Let result be S2.
- ❖ S1-S2 is Joe’s age!

---

---

---

---

---

---

---

## Summary



- ❖ Three main security objectives: secrecy, integrity, availability.
- ❖ DB admin is responsible for overall security.
  - Designs security policy, maintains an [audit trail](#), or history of users’ accesses to DB.
- ❖ Two main approaches to DBMS security: discretionary and mandatory access control.
  - Discretionary control based on notion of privileges.
  - Mandatory control based on notion of security classes.
- ❖ Statistical DBs try to protect individual data by supporting only aggregate queries, but often, individual information can be inferred.

---

---

---

---

---

---

---



## Parallel DBMS

Chapter 22, Part A

Slides by Joe Hellerstein, UCB, with some material from Jim Gray, Microsoft Research. See also:  
<http://www.research.microsoft.com/research/BARC/Gray/PDB95.ppt>

Database Management Systems, 2<sup>nd</sup> Edition. Raghu Ramakrishnan and Johannes Gehrke

1

---

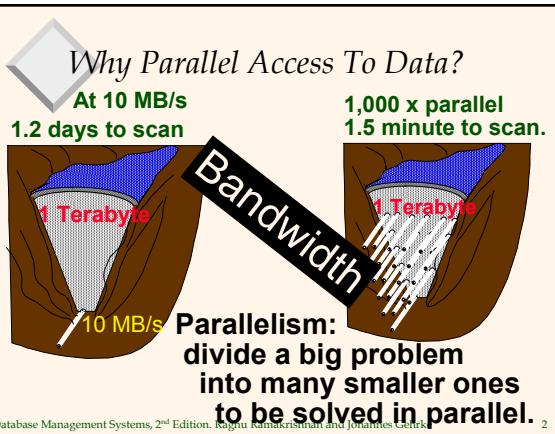
---

---

---

---

---



---

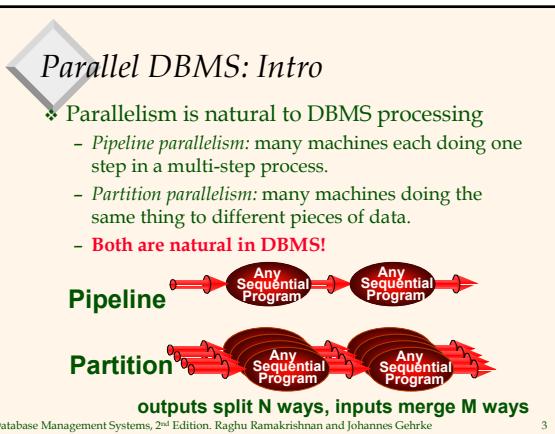
---

---

---

---

---



---

---

---

---

---

---

## DBMS: The || Success Story

- ❖ DBMSs are the most (only?) successful application of parallelism.
  - Teradata, Tandem vs. Thinking Machines, KSR..
  - Every major DBMS vendor has some || server
  - Workstation manufacturers now depend on || DB server sales.
- ❖ Reasons for success:
  - Bulk-processing (= partition ||-ism).
  - Natural pipelining.
  - Inexpensive hardware can do the trick!
  - Users/app-programmers don't need to think in ||

Database Management Systems, 2<sup>nd</sup> Edition. Raghu Ramakrishnan and Johannes Gehrke

4

---

---

---

---

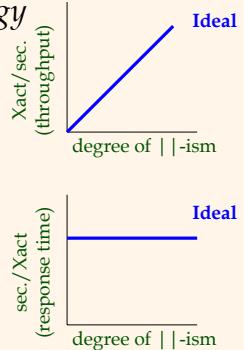
---

---

## Some || Terminology

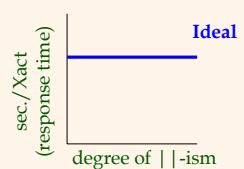
### ❖ Speed-Up

- More resources means proportionally less time for given amount of data.



### ❖ Scale-Up

- If resources increased in proportion to increase in data size, time is constant.



Database Management Systems, 2<sup>nd</sup> Edition. Raghu Ramakrishnan and Johannes Gehrke

5

---

---

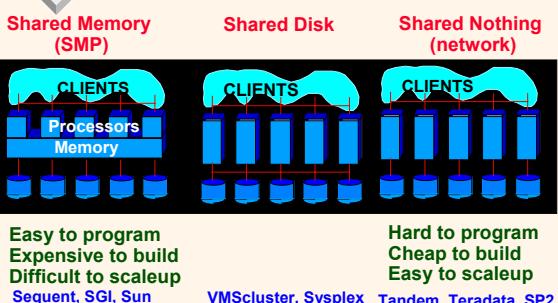
---

---

---

---

## Architecture Issue: Shared What?



Database Management Systems, 2<sup>nd</sup> Edition. Raghu Ramakrishnan and Johannes Gehrke

6

---

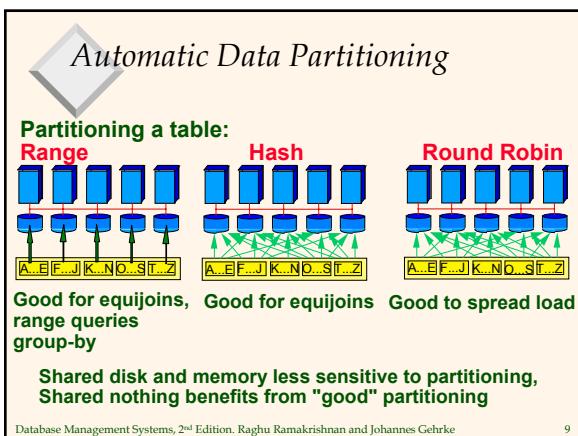
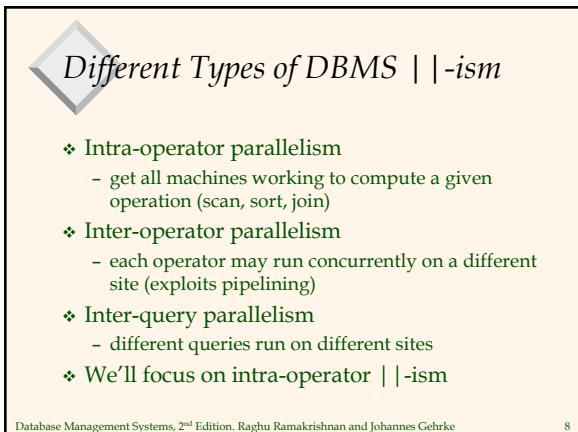
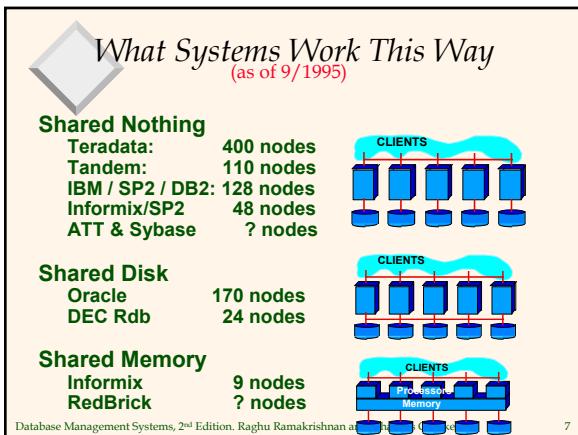
---

---

---

---

---



## Parallel Scans

- ❖ Scan in parallel, and merge.
- ❖ Selection may not require all sites for range or hash partitioning.
- ❖ Indexes can be built at each partition.
- ❖ Question: How do indexes differ in the different schemes?
  - Think about both lookups and inserts!
  - What about unique indexes?

---

---

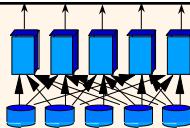
---

---

---

---

## Parallel Sorting



- ❖ Current records:
  - 8.5 Gb/minute, shared-nothing; Datamation benchmark in 2.41 secs (UCB students! <http://now.cs.berkeley.edu/NowSort/>)
- ❖ Idea:
  - Scan in parallel, and range-partition as you go.
  - As tuples come in, begin "local" sorting on each
  - Resulting data is sorted, and range-partitioned.
  - Problem: **skew!**
  - Solution: "sample" the data at start to determine partition points.

---

---

---

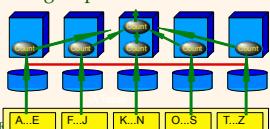
---

---

---

## Parallel Aggregates

- ❖ For each aggregate function, need a decomposition:
  - $\text{count}(S) = \sum \text{count}(s(i))$ , ditto for **sum()**
  - $\text{avg}(S) = (\sum \text{sum}(s(i))) / \sum \text{count}(s(i))$
  - and so on...
- ❖ For groups:
  - Sub-aggregate groups close to the source.
  - Pass each sub-aggregate to its group's site.
    - ◆ Chosen via a hash fn.



---

---

---

---

---

---

## Parallel Joins

### ❖ Nested loop:

- Each outer tuple must be compared with each inner tuple that might join.
- Easy for range partitioning on join cols, hard otherwise!

### ❖ Sort-Merge (or plain Merge-Join):

- Sorting gives range-partitioning.
  - ◆ But what about handling 2 skews?
- Merging partitioned tables is local.

---

---

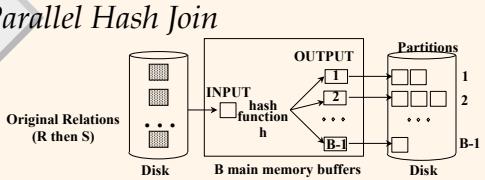
---

---

---

## Parallel Hash Join

Phase 1



### ❖ In first phase, partitions get distributed to different sites:

- A good hash function *automatically* distributes work evenly!

### ❖ Do second phase at each site.

### ❖ Almost always the winner for equi-join.

---

---

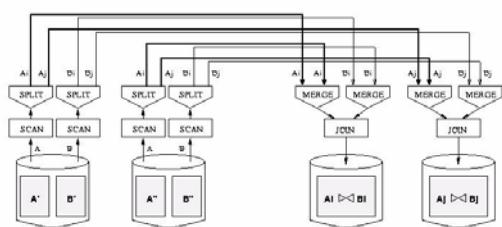
---

---

---

## Dataflow Network for || Join

Phase 2



### ❖ Good use of split/merge makes it easier to build parallel versions of sequential join code.

---

---

---

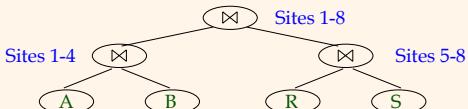
---

---

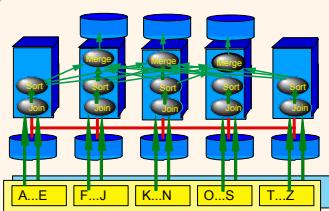
## Complex Parallel Query Plans

- ❖ Complex Queries: Inter-Operator parallelism

- Pipelining between operators:
  - note that sort and phase 1 of hash-join block the pipeline!!
- Bushy Trees



## $N \times M$ -way Parallelism



**N inputs, M outputs, no bottlenecks.**

## Partitioned Data Partitioned and Pipelined Data Flows

## Observations

- ❖ It is relatively easy to build a fast parallel query executor
  - S.M.O.P.
- ❖ It is hard to write a robust and world-class parallel query optimizer.
  - There are many tricks.
  - One quickly hits the complexity barrier.
  - Still open research!

## Parallel Query Optimization

- ❖ Common approach: 2 phases
  - Pick best sequential plan (System R algorithm)
  - Pick degree of parallelism based on current system parameters.
- ❖ “Bind” operators to processors
  - Take query tree, “decorate” as in previous picture.

---

---

---

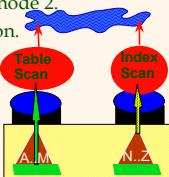
---

---

---

## What's Wrong With That?

- ❖ Best serial plan != Best || plan! Why?
- ❖ Trivial counter-example:
  - Table partitioned with local secondary index at two nodes
  - Range query: all of node 1 and 1% of node 2.
  - Node 1 should do a scan of its partition.
  - Node 2 should use secondary index.
- ❖ SELECT \*  
FROM telephone\_book  
WHERE name < "NoGood";



---

---

---

---

---

---

## Parallel DBMS Summary

- ❖ || -ism natural to query processing:
  - Both pipeline and partition || -ism!
- ❖ Shared-Nothing vs. Shared-Mem
  - Shared-disk too, but less standard
  - Shared-mem easy, costly. Doesn't scaleup.
  - Shared-nothing cheap, scales well, harder to implement.
- ❖ Intra-op, Inter-op, & Inter-query || -ism all possible.

---

---

---

---

---

---



## || DBMS Summary, cont.

- ❖ Data layout choices important!
- ❖ Most DB operations can be done partition-||
  - Sort.
  - Sort-merge join, hash-join.
- ❖ Complex plans.
  - Allow for pipeline-||ism, but sorts, hashes block the pipeline.
  - Partition ||-ism achieved via bushy trees.

---

---

---

---

---

---



## || DBMS Summary, cont.

- ❖ Hardest part of the equation: optimization.
  - 2-phase optimization simplest, but can be ineffective.
  - More complex schemes still at the research stage.
- ❖ We haven't said anything about Xacts, logging.
  - Easy in shared-memory architecture.
  - Takes some care in shared-nothing.

---

---

---

---

---

---



## *Distributed Databases*

### Chapter 22, Part B

---

---

---

---

---

---



## *Introduction*

- ❖ Data is stored at several sites, each managed by a DBMS that can run independently.
- ❖ **Distributed Data Independence:** Users should not have to know where data is located (extends Physical and Logical Data Independence principles).
- ❖ **Distributed Transaction Atomicity:** Users should be able to write Xacts accessing multiple sites just like local Xacts.

---

---

---

---

---

---



## *Recent Trends*

- ❖ Users have to be aware of where data is located, i.e., Distributed Data Independence and Distributed Transaction Atomicity are not supported.
- ❖ These properties are hard to support efficiently.
- ❖ For globally distributed sites, these properties may not even be desirable due to administrative overheads of making location of data transparent.

---

---

---

---

---

---

## Types of Distributed Databases

- ❖ **Homogeneous:** Every site runs same type of DBMS.
- ❖ **Heterogeneous:** Different sites run different DBMSs (different RDBMSs or even non-relational DBMSs).

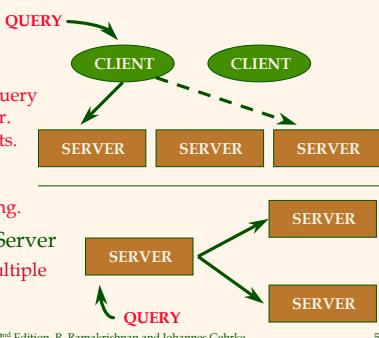


Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

4

## Distributed DBMS Architectures

- ❖ **Client-Server**  
Client ships query to single site. All query processing at server.
  - *Thin vs. fat clients.*
  - Set-oriented communication, client side caching.
- ❖ **Collaborating-Server**  
Query can span multiple sites.

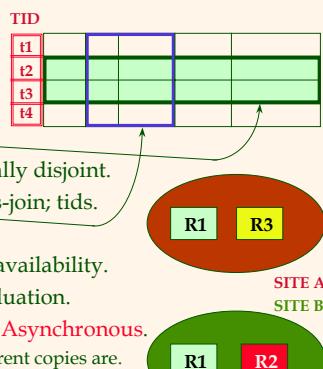


Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

5

## Storing Data

- ❖ **Fragmentation**
  - Horizontal: Usually disjoint.
  - Vertical: Lossless-join; tids.
- ❖ **Replication**
  - Gives increased availability.
  - Faster query evaluation.
  - **Synchronous vs. Asynchronous.**
    - ◆ Vary in how current copies are.



Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

6

## Distributed Catalog Management

- ❖ Must keep track of how data is distributed across sites.
- ❖ Must be able to name each replica of each fragment. To preserve local autonomy:
  - <local-name, birth-site>
- ❖ **Site Catalog:** Describes all objects (fragments, replicas) at a site + Keeps track of replicas of relations created at this site.
  - To find a relation, look up its birth-site catalog.
  - Birth-site never changes, even if relation is moved.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

7

---

---

---

---

---

---

---

## Distributed Queries

```
SELECT AVG(S.age)
FROM Sailors S
WHERE S.rating > 3
 AND S.rating < 7
```

- ❖ **Horizontally Fragmented:** Tuples with rating < 5 at Shanghai, >= 5 at Tokyo.
  - Must compute SUM(age), COUNT(age) at both sites.
  - If WHERE contained just S.rating>6, just one site.
- ❖ **Vertically Fragmented:** sid and rating at Shanghai, sname and age at Tokyo, tid at both.
  - Must reconstruct relation by join on tid, then evaluate the query.
- ❖ **Replicated:** Sailors copies at both sites.
  - Choice of site based on local costs, shipping costs.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

8

---

---

---

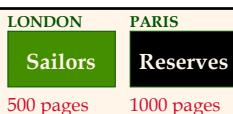
---

---

---

---

## Distributed Joins



- ❖ **Fetch as Needed**, Page NL, Sailors as outer:
  - **Cost:**  $500 D + 500 * 1000 (D+S)$
  - **D** is cost to read/write page; **S** is cost to ship page.
  - If query was not submitted at London, must add cost of shipping result to query site.
  - Can also do INL at London, fetching matching Reserves tuples to London as needed.
- ❖ **Ship to One Site:** Ship Reserves to London.
  - Cost:  $1000 S + 4500 D$  (SM Join; cost =  $3*(500+1000)$ )
  - If result size is very large, may be better to ship both relations to result site and then join them!

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

9

---

---

---

---

---

---

---

## *Semijoin*

- ❖ At London, project Sailors onto join columns and ship this to Paris.
- ❖ At Paris, join Sailors projection with Reserves.
  - Result is called **reduction** of Reserves wrt Sailors.
- ❖ Ship reduction of Reserves to London.
- ❖ At London, join Sailors with reduction of Reserves.
- ❖ **Idea:** Tradeoff the cost of computing and shipping projection and computing and shipping projection for cost of shipping full Reserves relation.
- ❖ Especially useful if there is a selection on Sailors, and answer desired at London.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

10

---

---

---

---

---

---

---

---

## *Bloomjoin*

- ❖ At London, compute a bit-vector of some size k:
  - Hash join column values into range 0 to k-1.
  - If some tuple hashes to I, set bit I to 1 (I from 0 to k-1).
  - Ship bit-vector to Paris.
- ❖ At Paris, hash each tuple of Reserves similarly, and discard tuples that hash to 0 in Sailors bit-vector.
  - Result is called **reduction** of Reserves wrt Sailors.
- ❖ Ship bit-vector reduced Reserves to London.
- ❖ At London, join Sailors with reduced Reserves.
- ❖ Bit-vector cheaper to ship, almost as effective.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

11

---

---

---

---

---

---

---

---

## *Distributed Query Optimization*

- ❖ Cost-based approach; consider all plans, pick cheapest; similar to centralized optimization.
  - **Difference 1:** Communication costs must be considered.
  - **Difference 2:** Local site autonomy must be respected.
  - **Difference 3:** New distributed join methods.
- ❖ Query site constructs **global plan**, with **suggested local plans** describing processing at each site.
  - If a site can improve suggested local plan, free to do so.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

12

---

---

---

---

---

---

---

---

## Updating Distributed Data

- ❖ **Synchronous Replication:** All copies of a modified relation (fragment) must be updated before the modifying Xact commits.
  - Data distribution is made transparent to users.
- ❖ **Asynchronous Replication:** Copies of a modified relation are only periodically updated; different copies may get out of synch in the meantime.
  - Users must be aware of data distribution.
  - Current products follow this approach.

---

---

---

---

---

---

## Synchronous Replication

- ❖ **Voting:** Xact must write a majority of copies to modify an object; must read enough copies to be sure of seeing at least one most recent copy.
  - E.g., 10 copies; 7 written for update; 4 copies read.
  - Each copy has version number.
  - Not attractive usually because reads are common.
- ❖ **Read-any Write-all:** Writes are slower and reads are faster, relative to Voting.
  - Most common approach to synchronous replication.
- ❖ Choice of technique determines *which* locks to set.

---

---

---

---

---

---

## Cost of Synchronous Replication

- ❖ Before an update Xact can commit, it must obtain locks on all modified copies.
  - Sends lock requests to remote sites, and while waiting for the response, holds on to other locks!
  - If sites or links fail, Xact cannot commit until they are back up.
  - Even if there is no failure, committing must follow an expensive **commit protocol** with many msgs.
- ❖ So the alternative of *asynchronous replication* is becoming widely used.

---

---

---

---

---

---

## Asynchronous Replication

- ❖ Allows modifying Xact to commit before all copies have been changed (and readers nonetheless look at just one copy).
  - Users must be aware of which copy they are reading, and that copies may be out-of-sync for short periods of time.
- ❖ Two approaches: Primary Site and Peer-to-Peer replication.
  - Difference lies in how many copies are "updatable" or "master copies".

---

---

---

---

---

---

## Peer-to-Peer Replication

- ❖ More than one of the copies of an object can be a master in this approach.
- ❖ Changes to a master copy must be propagated to other copies somehow.
- ❖ If two master copies are changed in a conflicting manner, this must be resolved. (e.g., Site 1: Joe's age changed to 35; Site 2: to 36)
- ❖ Best used when conflicts do not arise:
  - E.g., Each master site owns a disjoint fragment.
  - E.g., Updating rights owned by one master at a time.

---

---

---

---

---

---

## Primary Site Replication

- ❖ Exactly one copy of a relation is designated the primary or master copy. Replicas at other sites cannot be directly updated.
  - The primary copy is published.
  - Other sites subscribe to (fragments of) this relation; these are secondary copies.
- ❖ Main issue: How are changes to the primary copy propagated to the secondary copies?
  - Done in two steps. First, capture changes made by committed Xacts; then apply these changes.

---

---

---

---

---

---

## Implementing the Capture Step

- ❖ **Log-Based Capture:** The log (kept for recovery) is used to generate a Change Data Table (CDT).
  - If this is done when the log tail is written to disk, must somehow remove changes due to subsequently aborted Xacts.
- ❖ **Procedural Capture:** A procedure that is automatically invoked (**trigger**; more later!) does the capture; typically, just takes a snapshot.
- ❖ Log-Based Capture is better (cheaper, faster) but relies on proprietary log details.

---

---

---

---

---

---

## Implementing the Apply Step

- ❖ The Apply process at the secondary site periodically obtains (a snapshot or) changes to the CDT table from the primary site, and updates the copy.
  - Period can be timer-based or user/application defined.
- ❖ Replica can be a view over the modified relation!
  - If so, the replication consists of incrementally updating the materialized view as the relation changes.
- ❖ Log-Based Capture plus continuous Apply minimizes delay in propagating changes.
- ❖ Procedural Capture plus application-driven Apply is the most flexible way to process changes.

---

---

---

---

---

---

## Data Warehousing and Replication

- ❖ **A hot trend:** Building giant “warehouses” of data from many sites.
  - Enables complex **decision support queries** over data from across an organization.
- ❖ Warehouses can be seen as an instance of asynchronous replication.
  - Source data typically controlled by different DBMSs; emphasis on “cleaning” data and removing mismatches (\$ vs. rupees) while creating replicas.
- ❖ Procedural capture and application Apply best for this environment.

---

---

---

---

---

---

## Distributed Locking

- ❖ How do we manage locks for objects across many sites?
  - **Centralized:** One site does all locking.
    - ◆ Vulnerable to single site failure.
  - **Primary Copy:** All locking for an object done at the primary copy site for this object.
    - ◆ Reading requires access to locking site as well as site where the object is stored.
  - **Fully Distributed:** Locking for a copy done at site where the copy is stored.
    - ◆ Locks at all sites while writing an object.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

22

---

---

---

---

---

---

---

---

## Distributed Deadlock Detection

- ❖ Each site maintains a **local waits-for graph**.
- ❖ A global deadlock might exist even if the local graphs contain no cycles:

```
graph LR; subgraph SITE_A [SITE A]; T1A((T1)) --> T2A((T2)); end; subgraph SITE_B [SITE B]; T1B((T1)) --> T2B((T2)); end; subgraph GLOBAL [GLOBAL]; T1G((T1)) <--> T2G((T2)); end;
```
- ❖ Three solutions: **Centralized** (send all local graphs to one site); **Hierarchical** (organize sites into a hierarchy and send local graphs to parent in the hierarchy); **Timeout** (abort Xact if it waits too long).

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

23

---

---

---

---

---

---

---

---

## Distributed Recovery

- ❖ Two new issues:
  - New kinds of failure, e.g., links and remote sites.
  - If “sub-transactions” of an Xact execute at different sites, all or none must commit.  
Need a **commit protocol** to achieve this.
- ❖ A log is maintained at each site, as in a centralized DBMS, and commit protocol actions are additionally logged.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

24

---

---

---

---

---

---

---

---

## Two-Phase Commit (2PC)

- ❖ Site at which Xact originates is **coordinator**; other sites at which it executes are **subordinates**.
- ❖ When an Xact wants to commit:
  - ① Coordinator sends **prepare** msg to each subordinate.
  - ② Subordinate force-writes an **abort** or **prepare** log record and then sends a **no** or **yes** msg to coordinator.
  - ③ If coordinator gets unanimous yes votes, force-writes a **commit** log record and sends **commit** msg to all subs.  
Else, force-writes **abort** log rec, and sends **abort** msg.
  - ④ Subordinates force-write **abort/commit** log rec based on msg they get, then send **ack** msg to coordinator.
  - ⑤ Coordinator writes **end** log rec after getting all acks.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

25

---

---

---

---

---

---

---

## Comments on 2PC

- ❖ Two rounds of communication: first, **voting**; then, **termination**. Both initiated by coordinator.
- ❖ Any site can decide to abort an Xact.
- ❖ Every msg reflects a decision by the sender; to ensure that this decision survives failures, it is first recorded in the local log.
- ❖ All commit protocol log recs for an Xact contain Xactid and Coordinatorid. The coordinator's abort/commit record also includes ids of all subordinates.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

26

---

---

---

---

---

---

---

## Restart After a Failure at a Site

- ❖ If we have a **commit** or **abort** log rec for Xact T, but not an end rec, must redo/undo T.
  - If this site is the coordinator for T, keep sending **commit/abort** msgs to subs until **acks** received.
- ❖ If we have a **prepare** log rec for Xact T, but not **commit/abort**, this site is a subordinate for T.
  - Repeatedly contact the coordinator to find status of T, then write **commit/abort** log rec; redo/undo T; and write **end** log rec.
- ❖ If we don't have even a **prepare** log rec for T, unilaterally abort and undo T.
  - This site may be coordinator! If so, subs may send msgs.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and Johannes Gehrke

27

---

---

---

---

---

---

---

## Blocking

- ❖ If coordinator for Xact T fails, subordinates who have voted **yes** cannot decide whether to commit or abort T until coordinator recovers.
  - T is **blocked**.
  - Even if all subordinates know each other (extra overhead in **prepare** msg) they are blocked unless one of them voted **no**.

---

---

---

---

---

---

## Link and Remote Site Failures

- ❖ If a remote site does not respond during the commit protocol for Xact T, either because the site failed or the link failed:
  - If the current site is the coordinator for T, it should abort T.
  - If the current site is a subordinate, and has not yet voted **yes**, it should abort T.
  - If the current site is a subordinate and has voted **yes**, it is blocked until the coordinator responds.

---

---

---

---

---

---

## Observations on 2PC

- ❖ **Ack** msgs used to let coordinator know when it can “forget” an Xact; until it receives all **acks**, it must keep T in the Xact Table.
- ❖ If coordinator fails after sending **prepare** msgs but before writing **commit/abort** log recs, when it comes back up it aborts the Xact.
- ❖ If a subtransaction does no updates, its **commit** or **abort** status is irrelevant.

---

---

---

---

---

---

## 2PC with Presumed Abort

- ❖ When coordinator aborts T, it undoes T and removes it from the Xact Table immediately.
  - Doesn't wait for **acks**; "presumes abort" if Xact not in Xact Table. Names of subs not recorded in **abort** log rec.
- ❖ Subordinates do not send **acks** on **abort**.
- ❖ If subxact does not do updates, it responds to **prepare** msg with **reader** instead of **yes/no**.
- ❖ Coordinator subsequently ignores readers.
- ❖ If all subxacts are readers, 2nd phase not needed.

---

---

---

---

---

---

## Summary

- ❖ Parallel DBMSs designed for scalable performance. Relational operators very well-suited for parallel execution.
  - Pipeline and partitioned parallelism.
- ❖ Distributed DBMSs offer site autonomy and distributed administration. Must revisit storage and catalog techniques, concurrency control, and recovery issues.

---

---

---

---

---

---



## Data Warehousing and Decision Support

### Chapter 23, Part A

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

1

---

---

---

---

---

---



### Introduction

- ❖ Increasingly, organizations are analyzing current and historical data to identify useful patterns and support business strategies.
- ❖ Emphasis is on complex, interactive, exploratory analysis of very large datasets created by integrating data from across all parts of an enterprise; data is fairly static.
  - Contrast such **On-Line Analytic Processing (OLAP)** with traditional **On-line Transaction Processing (OLTP)**: mostly long queries, instead of short update Xacts.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

2

---

---

---

---

---

---



### Three Complementary Trends

- ❖ **Data Warehousing:** Consolidate data from many sources in one large repository.
  - Loading, periodic synchronization of replicas.
  - Semantic integration.
- ❖ **OLAP:**
  - Complex SQL queries and views.
  - Queries based on spreadsheet-style operations and “multidimensional” view of data.
  - Interactive and “online” queries.
- ❖ **Data Mining:** Exploratory search for interesting trends and anomalies. (Another lecture!)

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

3

---

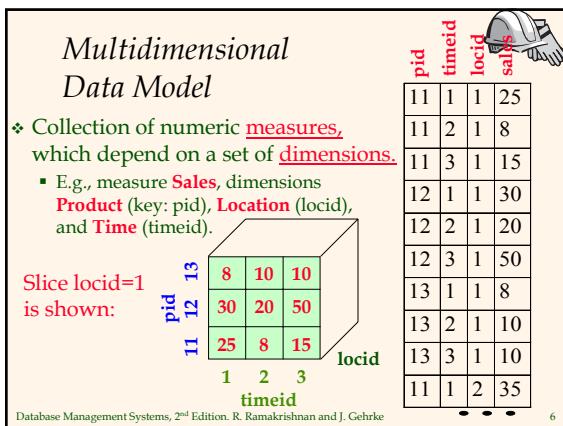
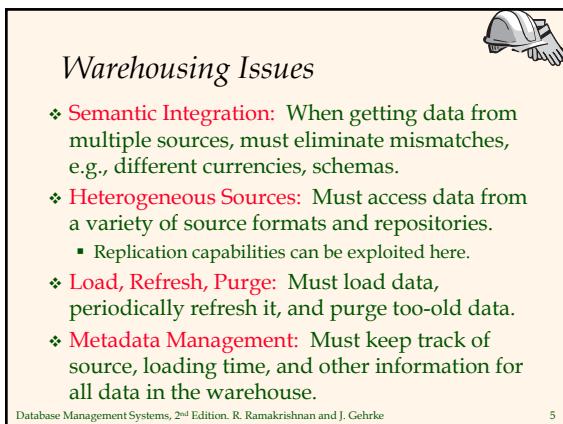
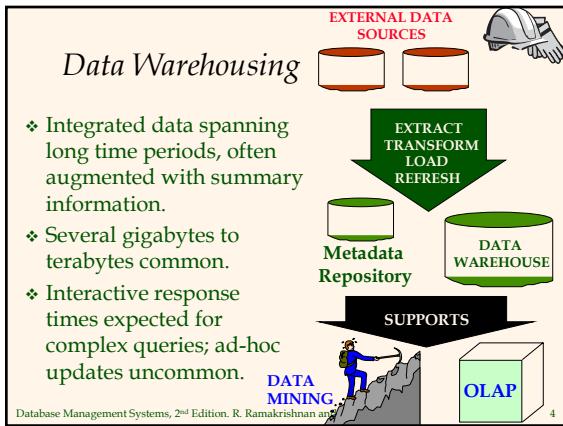
---

---

---

---

---





MOLAP vs ROLAP

- ❖ Multidimensional data can be stored physically in a (disk-resident, persistent) array; called **MOLAP** systems. Alternatively, can store as a relation; called **ROLAP** systems.
  - ❖ The main relation, which relates dimensions to a measure, is called the **fact table**. Each dimension can have additional attributes and an associated **dimension table**.
    - E.g., **Products(pid, pname, category, price)**
    - Fact tables are *much* larger than dimensional tables.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

7



## *Dimension Hierarchies*

- ❖ For each dimension, the set of values can be organized in a hierarchy:



Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

8



## *OLAP Queries*

- ❖ Influenced by SQL and by spreadsheets.
  - ❖ A common operation is to aggregate a measure over one or more dimensions.
    - Find total sales.
    - Find total sales for each city, or for each state.
    - Find top five products ranked by total sales.
  - ❖ Roll-up: Aggregating at different levels of a dimension hierarchy.
    - E.g., Given total sales by city, we can roll-up to get sales by state.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

9

## OLAP Queries



### ❖ Drill-down: The inverse of roll-up.

- E.g., Given total sales by state, can drill-down to get total sales by city.
- E.g., Can also drill-down on different dimension to get total sales by product for each state.

### ❖ Pivoting: Aggregation on selected dimensions.

- E.g., Pivoting on Location and Time yields this cross-tabulation:

	WI	CA	Total
1995	63	81	144
1996	38	107	145
1997	75	35	110
Total	176	223	339

10

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

## Comparison with SQL Queries



### ❖ The cross-tabulation obtained by pivoting can also be computed using a collection of SQL queries:

```
SELECT SUM(S.sales)
FROM Sales S, Times T, Locations L
WHERE S.timeid=T.timeid AND S.timeid=L.timeid
GROUP BY T.year, L.state
```

```
SELECT SUM(S.sales)
FROM Sales S, Times T
WHERE S.timeid=T.timeid
GROUP BY T.year
```

```
SELECT SUM(S.sales)
FROM Sales S, Location L
WHERE S.timeid=L.timeid
GROUP BY L.state
```

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

11

## The CUBE Operator



### ❖ Generalizing the previous example, if there are k dimensions, we have $2^k$ possible SQL GROUP BY queries that can be generated through pivoting on a subset of dimensions.

### ❖ CUBE pid, locid, timeid BY SUM Sales

- Equivalent to rolling up Sales on all eight subsets of the set {pid, locid, timeid}; each roll-up corresponds to an SQL query of the form:

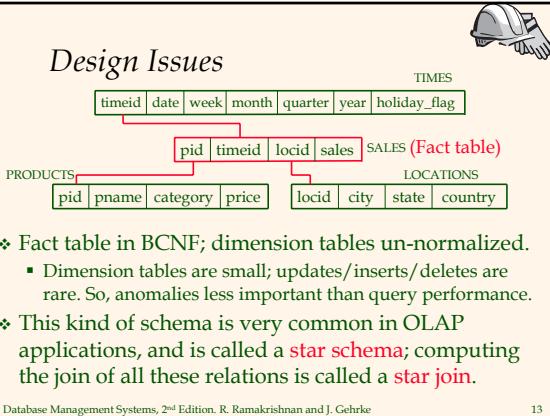
Lots of work on optimizing  
the CUBE operator!

```
SELECT SUM(S.sales)
FROM Sales S
GROUP BY grouping-list
```

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

12

## *Design Issues*



- ❖ Fact table in BCNF; dimension tables un-normalized.
    - Dimension tables are small; updates/inserts/deletes are rare. So, anomalies less important than query performance.
  - ❖ This kind of schema is very common in OLAP applications, and is called a **star schema**; computing the join of all these relations is called a **star join**.

Database Management Systems, 2nd Edition, R. Ramakrishnan and J. Gehrke

13

## *Implementation Issues*

- ❖ New indexing techniques: Bitmap indexes, Join indexes, array representations, compression, precomputation of aggregations, etc.
  - ❖ E.g., Bitmap index:

**Bit-vector:**  **sex**

**1 bit for each possible value.**

*Many queries can be answered using bit-vector ops!*

10
10
01
10

custid	name	sex	rating	rating
112	Joe	M	3	00100
115	Ram	M	5	00001
119	Sue	F	5	00001
112	Woo	M	4	00010

Database Management Systems, 2<sup>nd</sup> Edition, R. Ramakrishnan and J. Gehrke

14

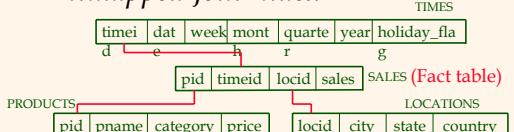
## *Join Indexes*

- ❖ Consider the join of Sales, Products, Times, and Locations, possibly with additional selection conditions (e.g., country="USA").
    - A **join index** can be constructed to speed up such joins. The index contains **[s,p,t,l]** if there are tuples (with sid **s**) in Sales, **p** in Products, **t** in Times and **l** in Locations that satisfy the join (and selection) conditions.
  - ❖ **Problem:** Number of join indexes can grow rapidly.
    - A variation addresses this problem: For each column with an additional selection (e.g., country), build an index with **[c,s]** in it if a dimension table tuple with value **c** in the selection column joins with a Sales tuple with sid **s**; if indexes are bitmaps, called **bitmapped join index**.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

15

## Bitmapped Join Index



- ❖ Consider a query with conditions  $\text{price}=10$  and  $\text{country}=\text{"USA"}$ . Suppose tuple (with sid) s in Sales joins with a tuple p with  $\text{price}=10$  and a tuple l with  $\text{country}=\text{"USA"}$ . There are two join indexes; one containing [10,s] and the other [USA,s].
- ❖ Intersecting these indexes tells us which tuples in Sales are in the join and satisfy the given selection.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

16

## Querying Sequences in SQL:1999

- ❖ Trend analysis is difficult to do in SQL-92:
  - Find the % change in monthly sales
  - Find the top 5 product by total sales
  - Find the trailing  $n$ -day moving average of sales
  - The first two queries can be expressed with difficulty, but the third cannot even be expressed in SQL-92 if  $n$  is a parameter of the query.
- ❖ The WINDOW clause in SQL:1999 allows us to write such queries over a table viewed as a sequence (implicitly, based on user-specified sort keys)

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

17

## The WINDOW Clause

```
SELECT L.state, T.month, AVG(S.sales) OVER W AS movavg
FROM Sales S, Times T, Locations L
WHERE S.timeid=T.timeid AND S.locid=L.locid
WINDOW W AS (PARTITION BY L.state
 ORDER BY T.month
 RANGE BETWEEN INTERVAL '1' MONTH PRECEDING
 AND INTERVAL '1' MONTH FOLLOWING)
```

- ❖ Let the result of the FROM and WHERE clauses be "Temp".
- ❖ (Conceptually) Temp is partitioned according to the PARTITION BY clause.
  - Similar to GROUP BY, but the answer has one row for each row in a partition, not one row per partition!
- ❖ Each partition is sorted according to the ORDER BY clause.
- ❖ For each row in a partition, the WINDOW clause creates a "window" of nearby (preceding or succeeding) tuples.
  - Can be value-based, as in example, using RANGE
  - Can be based on number of rows to include in the window, using ROWS clause
- ❖ The aggregate function is evaluated for each row in the partition using the corresponding window.
  - New aggregate functions that are useful with windowing include RANK (position of a row within its partition) and its variants DENSE\_RANK, PERCENT\_RANK, CUME\_DIST.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

18

## Top N Queries



- ❖ If you want to find the 10 (or so) cheapest cars, it would be nice if the DB could avoid computing the costs of all cars before sorting to determine the 10 cheapest.
  - **Idea:** Guess at a cost  $c$  such that the 10 cheapest all cost less than  $c$ , and that not too many more cost less. Then add the selection  $\text{cost} < c$  and evaluate the query.
    - If the guess is right, great, we avoid computation for cars that cost more than  $c$ .
    - If the guess is wrong, need to reset the selection and recompute the original query.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

19

---

---

---

---

---

---

---

---

---

---

## Top N Queries



```
SELECT P.pid, P.pname, S.sales
FROM Sales S, Products P
WHERE S.pid=P.pid AND S.locid=1 AND S.timeid=3
ORDER BY S.sales DESC
OPTIMIZE FOR 10 ROWS

SELECT P.pid, P.pname, S.sales
FROM Sales S, Products P
WHERE S.pid=P.pid AND S.locid=1 AND S.timeid=3
 AND S.sales > c
ORDER BY S.sales DESC
```

- ❖ **OPTIMIZE FOR** construct is not in SQL:1999!
- ❖ Cut-off value  $c$  is chosen by optimizer.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

20

---

---

---

---

---

---

---

---

---

---

## Online Aggregation



- ❖ Consider an aggregate query, e.g., finding the average sales by state. Can we provide the user with some information before the exact average is computed for all states?
  - Can show the current “running average” for each state as the computation proceeds.
  - Even better, if we use statistical techniques and sample tuples to aggregate instead of simply scanning the aggregated table, we can provide bounds such as “the average for Wisconsin is  $2000 \pm 102$  with 95% probability.
  - Should also use nonblocking algorithms!

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

21

---

---

---

---

---

---

---

---

---

---



## Summary

- ❖ Decision support is an emerging, rapidly growing subarea of databases.
- ❖ Involves the creation of large, consolidated data repositories called data warehouses.
- ❖ Warehouses exploited using sophisticated analysis techniques: complex SQL queries and OLAP “multidimensional” queries (influenced by both SQL and spreadsheets).
- ❖ New techniques for database design, indexing, view maintenance, and interactive querying need to be supported.

---

---

---

---

---

---

---



## Data Warehousing and Decision Support

### Chapter 23, Part B

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

1

---

---

---

---

---

---



## Views and Decision Support

- ❖ OLAP queries are typically aggregate queries.
  - Precomputation is essential for interactive response times.
  - The CUBE is in fact a collection of aggregate queries, and precomputation is especially important: lots of work on what is best to precompute given a limited amount of space to store precomputed results.
- ❖ Warehouses can be thought of as a collection of asynchronously replicated tables and periodically maintained views.
  - Has renewed interest in view maintenance!

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

2

---

---

---

---

---

---



## View Modification (Evaluate On Demand)

View

```
CREATE VIEW RegionalSales(category,sales,state)
AS SELECT P.category, S.sales, L.state
 FROM Products P, Sales S, Locations L
 WHERE P.pid=S.pid AND S.locid=L.locid
```

Query

```
SELECT R.category, R.state, SUM(R.sales)
 FROM RegionalSales AS R GROUP BY R.category, R.state
```

Modified Query

```
SELECT R.category, R.state, SUM(R.sales)
 FROM (SELECT P.category, S.sales, L.state
 FROM Products P, Sales S, Locations L
 WHERE P.pid=S.pid AND S.locid=L.locid) AS R
 GROUP BY R.category, R.state
```

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

3

---

---

---

---

---

---

## *View Materialization (Precomputation)*



- ❖ Suppose we precompute RegionalSales and store it with a clustered B+ tree index on [category,state,sales].
  - Then, previous query can be answered by an index-only scan.

```
SELECT R.state, SUM(R.sales)
FROM RegionalSales R
WHERE R.category="Laptop"
GROUP BY R.state
```

Index on precomputed view  
is great!

```
SELECT R.state, SUM(R.sales)
FROM RegionalSales R
WHERE R.state="Wisconsin"
GROUP BY R.category
```

Index is less useful (must  
scan entire leaf level).

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

4

## *Materialized Views*



- ❖ A view whose tuples are stored in the database is said to be **materialized**.
  - Provides fast access, like a (very high-level) cache.
  - Need to **Maintain** the view as the underlying tables change.
  - Ideally, we want incremental view maintenance algorithms.
- ❖ Close relationship to **Data Warehousing**, **OLAP**, (asynchronously) maintaining **Distributed Databases**, checking **Integrity Constraints**, and evaluating **Rules and Triggers**.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

5

## *Issues in View Materialization*



- ❖ What views should we materialize, and what indexes should we build on the precomputed results?
- ❖ Given a query and a set of materialized views, can we use the materialized views to answer the query?
- ❖ How frequently should we refresh materialized views to make them consistent with the underlying tables? (And how can we do this incrementally?)

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

6

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## *View Maintenance*



- ❖ Two steps:
  - **Propagate:** Compute changes to view when data changes.
  - **Refresh:** Apply changes to the materialized view table.
- ❖ **Maintenance policy:** Controls when we do refresh.
  - **Immediate:** As part of the transaction that modifies the underlying data tables. (+ Materialized view is always consistent; - updates are slowed)
  - **Deferred:** Some time later, in a separate transaction. (- View becomes inconsistent; + can scale to maintain many views without slowing updates)

## *Deferred Maintenance*



- ❖ Three flavors:
  - **Lazy:** Delay refresh until next query on view; then refresh before answering the query.
  - **Periodic (Snapshot):** Refresh periodically. Queries possibly answered using outdated version of view tuples. Widely used, especially for asynchronous replication in distributed databases, and for warehouse applications.
  - **Event-based:** E.g., Refresh after a fixed number of updates to underlying data tables.

## *Snapshots in Oracle 7*



- ❖ A **snapshot** is a local materialization of a view on data stored at a master site.
  - Periodically refreshed by re-computing view entirely.
  - Incremental “fast refresh” for “simple snapshots” (each row in view based on single row in a single underlying data table; no DISTINCT, GROUP BY, or aggregate ops; no sub-queries, joins, or set ops)
    - Changes to master recorded in a log by a trigger to support this.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Issues in View Maintenance (1)



```
expensive_parts(pno) :- parts(pno, cost), cost > 1000
```

- ❖ **What information is available?** (Base relations, materialized view, ICs). Suppose parts(p5,5000) is inserted:

- **Only materialized view available:** Add p5 if it isn't there.
- **Parts table is available:** If there isn't already a parts tuple p5 with cost >1000, add p5 to view.
  - May not be available if the view is in a data warehouse!
- **If we know pno is key for parts:** Can infer that p5 is not already in view, must insert it.

---

---

---

---

---

---

## Issues in View Maintenance (2)



```
expensive_parts(pno) :- parts(pno, cost), cost > 1000
```

- ❖ **What changes are propagated?** (Inserts, deletes, updates). Suppose parts(p1,3000) is deleted:

- **Only materialized view available:** If p1 is in view, no way to tell whether to delete it. (Why?)
  - If count(#derivations) is maintained for each view tuple, can tell whether to delete p1 (decrement count and delete if = 0).
- **Parts table is available:** If there is no other tuple p1 with cost >1000 in parts, delete p1 from view.
- **If we know pno is key for parts:** Can infer that p1 is currently in view, and must be deleted.

---

---

---

---

---

---

## Issues in View Maintenance (3)



- ❖ **View definition language?** (Conjunctive queries, SQL subset, duplicates, aggregates, recursion)

```
Supp_parts(pno) :- suppliers(sno, pno), parts(pno, cost)
```

- ❖ Suppose parts(p5,5000) is inserted:

- Can't tell whether to insert p5 into view if we're only given the materialized view.

---

---

---

---

---

---

### *Incremental Maintenance Alg: One Rule, Inserts*



`View(X,Y) :- Rel1(X,Z), Rel2(Z,Y)`

- ❖ **Step 0:** For each tuple in the materialized view, store a “derivation count”.
- ❖ **Step 1:** Rewrite this rule using Seminaive rewriting, set “*delta\_old*” relations for Rel1 and Rel2 to be the inserted tuples.
- ❖ **Step 2:** Compute the “*delta\_new*” relations for the view relation.
  - Important: Don’t remove duplicates! For each new tuple, maintain a “derivation count”.
- ❖ **Step 3:** Refresh the stored view by doing “multiset union” of the new and old view tuples. (I.e., update the derivation counts of existing tuples, and add the new tuples that weren’t in the view earlier.)

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

13

---

---

---

---

---

---

---

### *Incremental Maintenance Alg: One Rule, Deletes*



`View(X,Y) :- Rel1(X,Z), Rel2(Z,Y)`

- ❖ **Steps 0 - 2:** As for inserts.
- ❖ **Step 3:** Refresh the stored view by doing “multiset difference ” of the new and old view tuples.
  - To update the derivation counts of existing tuples, we must now subtract the derivation counts of the new tuples from the counts of existing tuples.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

14

---

---

---

---

---

---

---

### *Incremental Maintenance Alg: General*



- ❖ The “counting” algorithm can be generalized to views defined by multiple rules. In fact, it can be generalized to SQL queries with duplicate semantics, negation, and aggregation.
  - Try and do this! The extension is straightforward.

Database Management Systems, 2<sup>nd</sup> Edition. R. Ramakrishnan and J. Gehrke

15

---

---

---

---

---

---

---

## Maintaining Warehouse Views



```
view(sno) :- r1(sno, pno), r2(pno, cost)
```

**Problem:**  
New source  
updates  
between  
Steps 1 and  
3!

- ❖ **Main twist:** The views are in the data warehouse, and the source tables are somewhere else (operational DBMS, legacy sources, ...).
  - 1) Warehouse is notified whenever source tables are updated. (e.g., when a tuple is added to r2)
  - 2) Warehouse may need additional information about source tables to process the update (e.g., what is in r1 currently?)
  - 3) The source responds with the additional info, and the warehouse incrementally refreshes the view.

---

---

---

---

---

---

```
view(sno) :- r1(sno, pno), r2(pno, cost)
```



## Example of Warehouse View Maint.

- ❖ Initially, we have  $r1(1,2)$ ,  $r2$  empty
- ❖ **insert  $r2(2,3)$**  at source; notify warehouse
- ❖ Warehouse asks  $?r1(sno,2)$ 
  - Checking to find sno's to insert into view
- ❖ **insert  $r1(4,2)$**  at source; notify warehouse
- ❖ Warehouse asks  $?r2(2,cost)$ 
  - Checking to see if we need to increment count for view(4)
- ❖ Source gets first warehouse query, and returns  $sno=1$ ,  $sno=4$ ; these values go into view (with derivation counts of 1 each)
- ❖ Source gets second query, and says Yes, so count for 4 is incremented in the view
  - **But this is wrong! Correct count for view(4) is 1.**

---

---

---

---

---

---

---

---

## Warehouse View Maintenance



- ❖ **Alternative 1:** Evaluate view from scratch
  - On every source update, or periodically
- ❖ **Alternative 2:** Maintain a copy of each source table at warehouse
- ❖ **Alternative 3:** More fancy algorithms
  - Generate queries to the source that take into account the anomalies due to earlier conflicting updates.

---

---

---

---

---

---

---

---



## Deductive Databases

### Chapter 25

---

---

---

---

---

---



## Motivation

- ❖ SQL-92 cannot express some queries:
  - Are we running low on any parts needed to build a ZX600 sports car?
  - What is the total component and assembly cost to build a ZX600 at today's part prices?
- ❖ Can we extend the query language to cover such queries?
  - Yes, by adding **recursion**.

---

---

---

---

---

---



## Datalog

- ❖ SQL queries can be read as follows:  
“If some tuples exist in the From tables that satisfy the Where conditions,  
then the Select tuple is in the answer.”
- ❖ Datalog is a query language that has the same if-then flavor:
  - **New:** The answer table can appear in the From clause, i.e., be defined recursively.
  - Prolog style syntax is commonly used.

---

---

---

---

---

---

**Example**

part	subpart	number
trike	wheel	3
trike	frame	1
frame	seat	1
frame	pedal	1
wheel	spoke	2
wheel	tire	1
tire	rim	1
tire	tube	1

**Assembly instance**

---



---



---



---



---



---



---



---

**The Problem with R.A. and SQL-92**

- Intuitively, we must join Assembly with itself to deduce that trike contains spoke and tire.
  - Takes us one level down Assembly hierarchy.
  - To find components that are one level deeper (e.g., rim), need another join.
  - To find all components, need as many joins as there are levels in the given instance!
- For any relational algebra expression, we can create an Assembly instance for which some answers are not computed by including more levels than the number of joins in the expression!

---



---



---



---



---



---



---



---

**A Datalog Query that Does the Job**

```
Comp(Part, Subpt) :- Assembly(Part, Subpt, Qty).
Comp(Part, Subpt) :- Assembly(Part, Part2, Qty),
 Comp(Part2, Subpt).
```

head of rule      implication      body of rule

Can read the second rule as follows:  
**"For all values of Part, Subpt and Qty,  
 if there is a tuple (Part, Part2, Qty) in Assembly  
 and a tuple (Part2, Subpt) in Comp,  
 then there must be a tuple (Part, Subpt) in Comp."**

---



---



---



---



---



---



---



---

## Using a Rule to Deduce New Tuples

- Each rule is a **template**: by assigning constants to the variables in such a way that each body "literal" is a tuple in the corresponding relation, we identify a tuple that must be in the head relation.
  - By setting Part=trike, Subpt=wheel, Qty=3 in the first rule, we can deduce that the tuple <trike,wheel> is in the relation Comp.
  - This is called an **inference** using the rule.
  - Given a set of tuples, we **apply** the rule by making all possible inferences with these tuples in the body.

## Example

- For any instance of Assembly, we can compute all Comp tuples by repeatedly applying the two rules. (Actually, we can apply Rule 1 just once, then apply Rule 2 repeatedly.)

trike	spoke
trike	tire
trike	seat
trike	pedal
wheel	rim
wheel	tube

Comp tuples  
got by applying  
Rule 2 once

trike	spoke
trike	tire
trike	seat
trike	pedal
wheel	rim
wheel	tube

Comp tuples  
got by applying  
Rule 2 twice

## Datalog vs. SQL Notation

- Don't let the rule syntax of Datalog fool you: a collection of Datalog rules can be rewritten in SQL syntax, if recursion is allowed.

WITH RECURSIVE Comp(Part, Subpt) AS

```
(SELECT A1.Part, A1.Subpt FROM Assembly A1)
UNION
(SELECT A2.Part, C1.Subpt
FROM Assembly A2, Comp C1
WHERE A2.Subpt=C1.Part)
```

SELECT \* FROM Comp C2

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Fixpoints

- ❖ Let  $f$  be a function that takes values from domain  $D$  and returns values from  $D$ . A value  $v$  in  $D$  is a **fixpoint** of  $f$  if  $f(v)=v$ .
- ❖ Consider the fn  $double^+$ , which is applied to a set of integers and returns a set of integers (I.e.,  $D$  is the set of all sets of integers).
  - E.g.,  $double^+(\{1,2,5\})=\{2,4,10\} \text{ Union } \{1,2,5\}$
  - The set of all integers is a fixpoint of  $double^+$ .
  - The set of all even integers is another fixpoint of  $double^+$ ; it is smaller than the first fixpoint.



## Least Fixpoint Semantics for Datalog

- ❖ The **least fixpoint** of a function  $f$  is a fixpoint  $v$  of  $f$  such that every other fixpoint of  $f$  is smaller than or equal to  $v$ .
- ❖ In general, there may be no least fixpoint (we could have two minimal fixpoints, neither of which is smaller than the other).
- ❖ If we think of a Datalog program as a function that is applied to a set of tuples and returns another set of tuples, this function (fortunately!) always has a least fixpoint.



## Negation

```
Big(Part) :- Assembly(Part, Subpt, Qty),
 Qty > 2, not Small(Part).
Small(Part) :- Assembly(Part, Subpt, Qty),
 not Big(Part).
```

- ❖ If rules contain **not** there may not be a least fixpoint. Consider the Assembly instance; **trike** is the only part that has 3 or more copies of some subpart. Intuitively, it should be in **Big**, and it will be if we apply Rule 1 first.
  - But we have **Small(trike)** if Rule 2 is applied first!
  - There are two minimal fixpoints for this program: **Big** is empty in one, and contains **trike** in the other (and all other parts are in **Small** in both fixpoints).
- ❖ Need a way to choose the intended fixpoint.

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Stratification



- ❖ T depends on S if some rule with T in the head contains S or (recursively) some predicate that depends on S, in the body.
- ❖ Stratified program: If T depends on not S, then S cannot depend on T (or not T).
- ❖ If a program is stratified, the tables in the program can be partitioned into strata:
  - Stratum 0: All database tables.
  - Stratum I: Tables defined in terms of tables in Stratum I and lower strata.
  - If T depends on not S, S is in lower stratum than T.

---

---

---

---

---

---

## Fixpoint Semantics for Stratified Pgms



- ❖ The semantics of a stratified program is given by one of the minimal fixpoints, which is identified by the following operational defn:
  - First, compute the least fixpoint of all tables in Stratum 1. (Stratum 0 tables are fixed.)
  - Then, compute the least fixpoint of tables in Stratum 2; then the lfp of tables in Stratum 3, and so on, stratum-by-stratum.
- ❖ Note that Big/Small program is not stratified.

---

---

---

---

---

---

## Aggregate Operators

SELECT A.Part, SUM(<Qty>)  
FROM Assembly A  
GROUP BY A.Part



NumParts(Part, SUM(<Qty>)) :- Assembly(Part, Subpt, Qty).

- ❖ The  $< \dots >$  notation in the head indicates grouping; the remaining arguments (Part, in this example) are the GROUP BY fields.
- ❖ In order to apply such a rule, must have all of Assembly relation available.
- ❖ Stratification with respect to use of  $< \dots >$  is the usual restriction to deal with this problem; similar to negation.

---

---

---

---

---

---

## Evaluation of Datalog Programs

- ❖ **Repeated inferences:** When recursive rules are repeatedly applied in the naïve way, we make the same inferences in several iterations.
- ❖ **Unnecessary inferences:** Also, if we just want to find the components of a particular part, say **wheel**, computing the fixpoint of the Comp program and then selecting tuples with **wheel** in the first column is wasteful, in that we compute many irrelevant facts.



## Avoiding Repeated Inferences

- ❖ **Seminaive Fixpoint Evaluation:** Avoid repeated inferences by ensuring that when a rule is applied, at least one of the body facts was generated in the most recent iteration. (Which means this inference could not have been carried out in earlier iterations.)
  - For each recursive table P, use a table **delta\_P** to store the P tuples generated in the previous iteration.
  - Rewrite the program to use the delta tables, and update the delta tables between iterations.

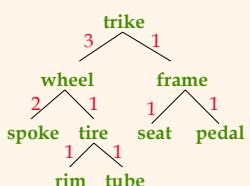
**Comp(Part, Subpt) :- Assembly(Part, Part2, Qty),  
delta\_Comp(Part2, Subpt).**



## Avoiding Unnecessary Inferences

```
SameLev(S1,S2) :- Assembly(P1,S1,Q1), Assembly(P2,S2,Q2).
SameLev(S1,S2) :- Assembly(P1,S1,Q1),
 SameLev(P1,P2), Assembly(P2,S2,Q2).
```

- ❖ There is a tuple (S1,S2) in SameLev if there is a path up from S1 to some node and down to S2 with the same number of up and down edges.



---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Avoiding Unnecessary Inferences



- ❖ Suppose that we want to find all SameLev tuples with **spoke** in the first column. We should “push” this selection into the fixpoint computation to avoid unnecessary inferences.
- ❖ But we can’t just compute SameLev tuples with **spoke** in the first column, because some other SameLev tuples are needed to compute all such tuples:

```
SameLev(spoke,seat) :- Assembly(wheel,spoke,2),
 SameLev(wheel,frame), Assembly(frame,seat,1).
```

---

---

---

---

---

---

## “Magic Sets” Idea



- ❖ Idea: Define a “filter” table that computes all relevant values, and restrict the computation of SameLev to infer only tuples with a relevant value in the first column.

```
Magic_SL(P1) :- Magic_SL(S1), Assembly(P1,S1,Q1).
Magic(spoke).
```

```
SameLev(S1,S2) :- Magic_SL(S1), Assembly(P1,S1,Q1),
 Assembly(P2,S2,Q2).
SameLev(S1,S2) :- Magic_SL(S1), Assembly(P1,S1,Q1),
 SameLev(P1,P2), Assembly(P2,S2,Q2).
```

---

---

---

---

---

---

## The Magic Sets Algorithm



- ❖ Generate an “adorned” program
  - Program is rewritten to make the pattern of bound and free arguments in the query explicit; evaluating SameLevel with the first argument bound to a constant is quite different from evaluating it with the second argument bound
  - This step was omitted for simplicity in previous slide
- ❖ Add filters of the form “Magic\_P” to each rule in the adorned program that defines a predicate P to restrict these rules
- ❖ Define new rules to define the filter tables of the form Magic\_P

---

---

---

---

---

---

## Generating Adorned Rules



- The adorned program for the query pattern  $\text{SameLev}^{\text{bf}}$ , assuming a right-to-left order of rule evaluation :

```
SameLevbf (S1,S2) :- Assembly(P1,S1,Q1), Assembly(P2,S2,Q2).
 |
 +-- SameLevbf (S1,S2) :- Assembly(P1,S1,Q1),
 |
 +-- SameLevbf (P1,P2), Assembly(P2,S2,Q2).
```

- An argument of (a given body occurrence of)  $\text{SameLev}$  is **b** if it appears to the left in the body, or in a **b** arg of the head of the rule.
- Assembly is not adorned because it is an explicitly stored table.

---

---

---

---

---

---

## Defining Magic Tables



- After modifying each rule in the adorned program by adding filter “Magic” predicates, a rule for  $\text{Magic\_P}$  is generated from each occurrence  $O$  of  $P$  in the body of such a rule:
  - Delete everything to the right of  $O$
  - Add the prefix “Magic” and delete the free columns of  $O$
  - Move  $O$ , with these changes, into the head of the rule

```
SameLevbf (S1,S2) :- Magic_SL(S1), Assembly(P1,S1,Q1),
 |
 +-- SameLevbf (P1,P2), Assembly(P2,S2,Q2).
```

$\text{Magic\_SL}(P1) :- \text{Magic\_SL}(S1)$ ,  $\text{Assembly}(P1,S1,Q1)$ .

---

---

---

---

---

---

## Summary



- Adding recursion extends relational algebra and SQL-92 in a fundamental way; included in SQL:1999, though not the core subset.
- Semantics based on iterative fixpoint evaluation. Programs with negation are restricted to be stratified to ensure that semantics is intuitive and unambiguous.
- Evaluation must avoid repeated and unnecessary inferences.
  - “Seminaive” fixpoint evaluation
  - “Magic Sets” query transformation

---

---

---

---

---

---

## Chapter 26: Data Mining

---

(Some slides courtesy of  
Rich Caruana, Cornell University)

---

---

---

---

---

---

---

---

---

---

### Definition

---

Data mining is the exploration and analysis of large quantities of data in order to discover valid, novel, potentially useful, and ultimately understandable patterns in data.

Example pattern (Census Bureau Data):  
If (relationship = husband), then (gender = male). 99.6%

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

---

---

### Definition (Cont.)

---

Data mining is the exploration and analysis of large quantities of data in order to discover valid, novel, potentially useful, and ultimately understandable patterns in data.

**Valid:** The patterns hold in general.

**Novel:** We did not know the pattern beforehand.

**Useful:** We can devise actions from the patterns.

**Understandable:** We can interpret and comprehend the patterns.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

---

---

## Why Use Data Mining Today?

Human analysis skills are inadequate:

- Volume and dimensionality of the data
- High data growth rate

Availability of:

- Data
- Storage
- Computational power
- Off-the-shelf software
- Expertise

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## An Abundance of Data

- Supermarket scanners, POS data
- Preferred customer cards
- Credit card transactions
- Direct mail response
- Call center records
- ATM machines
- Demographic data
- Sensor networks
- Cameras
- Web server logs
- Customer web site trails

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

## Evolution of Database Technology

- 1960s: IMS, network model
- 1970s: The relational data model, first relational DBMS implementations
- 1980s: Maturing RDBMS, application-specific DBMS, (spatial data, scientific data, image data, etc.), OODBMS
- 1990s: Mature, high-performance RDBMS technology, parallel DBMS, terabyte data warehouses, object-relational DBMS, middleware and web technology
- 2000s: High availability, zero-administration, seamless integration into business processes
- 2010: Sensor database systems, databases on embedded systems, P2P database systems, large-scale pub/sub systems, ???

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

## Computational Power

- Moore's Law:  
In 1965, Intel Corporation cofounder Gordon Moore predicted that the density of transistors in an integrated circuit would double every year. (Later changed to reflect 18 months progress.)
- Experts on ants estimate that there are  $10^{16}$  to  $10^{17}$  ants on earth. In the year 1997, we produced one transistor per ant.



Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Much Commercial Support

- Many data mining tools
  - <http://www.kdnuggets.com/software>
- Database systems with data mining support
- Visualization tools
- Data mining process support
- Consultants

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Why Use Data Mining Today?

Competitive pressure!

"The secret of success is to know something that nobody else knows."

Aristotle Onassis

- Competition on service, not only on price (Banks, phone companies, hotel chains, rental car companies)
- Personalization, CRM
- The real-time enterprise
- "Systemic listening"
- Security, homeland defense

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## The Knowledge Discovery Process

Steps:

1. Identify business problem
2. Data mining
3. Action
4. Evaluation and measurement
5. Deployment and integration into businesses processes

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

## Data Mining Step in Detail

### 2.1 Data preprocessing

- Data selection: Identify target datasets and relevant fields
- Data cleaning
  - Remove noise and outliers
  - Data transformation
  - Create common units
  - Generate new fields

### 2.2 Data mining model construction

### 2.3 Model evaluation

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

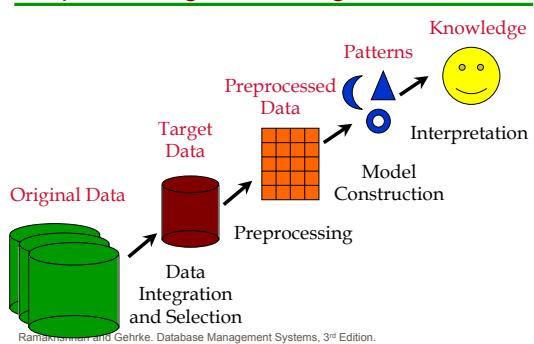
---

---

---

---

## Preprocessing and Mining



Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

## Example Application: Sports

IBM Advanced Scout analyzes  
NBA game statistics

- Shots blocked
  - Assists
  - Fouls
- Google: "IBM Advanced Scout"



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Advanced Scout

- Example pattern: An analysis of the data from a game played between the New York Knicks and the Charlotte Hornets revealed that "*When Glenn Rice played the shooting guard position, he shot 5/6 (83%) on jump shots.*"
- Pattern is interesting:  
The average shooting percentage for the Charlotte Hornets during that game was 54%.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Example Application: Sky Survey

- Input data: 3 TB of image data with 2 billion sky objects, took more than six years to complete
- Goal: Generate a catalog with all objects and their type
- Method: Use decision trees as data mining model
- Results:
  - 94% accuracy in predicting sky object classes
  - Increased number of faint objects classified by 300%
  - Helped team of astronomers to discover 16 new high red-shift quasars in one order of magnitude less observation time

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## **Gold Nuggets?**

- Investment firm mailing list: Discovered that old people do not respond to IRA mailings
- Bank clustered their customers. One cluster: Older customers, no mortgage, less likely to have a credit card
- “Bank of 1911”
- Customer churn example

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## **What is a Data Mining Model?**

A data mining model is a description of a specific aspect of a dataset. It produces output values for an assigned set of input values.

### **Examples:**

- Linear regression model
- Classification model
- Clustering

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## **Data Mining Models (Contd.)**

A data mining model can be described at two levels:

- Functional level:
  - Describes model in terms of its intended usage.  
Examples: Classification, clustering
- Representational level:
  - Specific representation of a model.  
Example: Log-linear model, classification tree, nearest neighbor method.
- Black-box models versus transparent models

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Data Mining: Types of Data

- Relational data and transactional data
  - Spatial and temporal data, spatio-temporal observations
  - Time-series data
  - Text
  - Images, video
  - Mixtures of data
  - Sequence data
- Features from processing other data sources

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Types of Variables

- *Numerical*: Domain is ordered and can be represented on the real line (e.g., age, income)
- *Nominal* or *categorical*: Domain is a finite set without any natural ordering (e.g., occupation, marital status, race)
- *Ordinal*: Domain is ordered, but absolute differences between values is unknown (e.g., preference scale, severity of an injury)

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Data Mining Techniques

- **Supervised learning**
  - Classification and regression
- **Unsupervised learning**
  - Clustering
- Dependency modeling
  - Associations, summarization, causality
- Outlier and deviation detection
- Trend analysis and change detection

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Supervised Learning

- $F(x)$ : true function (usually not known)
  - $D$ : training sample drawn from  $F(x)$

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

## Supervised Learning



Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

## Supervised Learning

### Well-defined goal:

Learn  $G(x)$  that is a good approximation to  $F(x)$  from training sample D

Well-defined error metrics:

Accuracy, RMSE, ROC, ...

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition

## Supervised Learning

Training dataset:

57,M,195,0,125,95,39,25,0,1,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0	0
78,M,160,1,130,100,37,40,1,0,0,0,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
69,F,180,0,115,85,40,22,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0
18,M,165,0,110,80,41,30,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
54,F,135,0,115,95,39,35,1,1,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0	1
84,F,210,1,135,105,39,24,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0
89,F,135,0,120,95,36,28,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,1,0,0	1
49,M,195,0,115,85,39,32,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0	0
40,M,205,0,115,90,37,18,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0
74,M,250,1,130,100,38,26,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
77,F,140,0,125,100,40,30,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1	0

Test dataset:

71,M,160,1,130,105,38,20,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,?

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

---

---

## Un-Supervised Learning

Training dataset:

57,M,195,0,125,95,39,25,0,1,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0	0
78,M,160,1,130,100,37,40,1,0,0,0,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
69,F,180,0,115,85,40,22,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0
18,M,165,0,110,80,41,30,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
54,F,135,0,115,95,39,35,1,1,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0	1
84,F,210,1,135,105,39,24,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0
89,F,135,0,120,95,36,28,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,1,0,0	1
49,M,195,0,115,85,39,32,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0	0
40,M,205,0,115,90,37,18,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0
74,M,250,1,130,100,38,26,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
77,F,140,0,125,100,40,30,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1	0



Test dataset:

71,M,160,1,130,105,38,20,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,?

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

---

---

## Un-Supervised Learning

Training dataset:

57,M,195,0,125,95,39,25,0,1,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0	0
78,M,160,1,130,100,37,40,1,0,0,0,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
69,F,180,0,115,85,40,22,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0
18,M,165,0,110,80,41,30,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
54,F,135,0,115,95,39,35,1,1,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0	1
84,F,210,1,135,105,39,24,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0
89,F,135,0,120,95,36,28,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,1,0,0	1
49,M,195,0,115,85,39,32,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0	0
40,M,205,0,115,90,37,18,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0
74,M,250,1,130,100,38,26,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
77,F,140,0,125,100,40,30,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1	0



Test dataset:

71,M,160,1,130,105,38,20,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,?

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

---

---

## Un-Supervised Learning

Data Set:

```
57,M,195,0,125,95,39,25,0,1,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0
78,M,160,1,130,100,37,40,1,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0
69,F,180,0,115,85,40,22,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0
18,M,165,0,110,80,41,30,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
54,F,135,0,115,95,39,35,1,1,0,0,0,1,0,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0
84,F,210,1,135,105,39,24,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
89,F,135,0,120,95,36,28,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,0
49,M,195,0,115,85,39,32,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0
40,M,205,0,115,90,37,18,0
74,M,250,1,130,100,38,26,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
77,F,140,0,125,100,40,30,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1
```

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

## Lecture Overview

- Data Mining I: Decision Trees
- Data Mining II: Clustering
- Data Mining III: Association Analysis

---

---

---

---

---

---

---

---

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

## Classification Example

- Example training database
  - Two predictor attributes: Age and Car-type (**S**port, **M**inivan and **T**ruck)
  - Age is ordered, Car-type is categorical attribute
  - Class label indicates whether person bought product
  - Dependent attribute is *categorical*

Age	Car	Class
20	M	Yes
30	M	Yes
25	T	No
30	S	Yes
40	S	Yes
20	T	No
30	M	Yes
25	M	Yes
40	M	Yes
20	S	No

---

---

---

---

---

---

---

---

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

## Regression Example

- Example training database
  - Two predictor attributes:  
Age and Car-type (**S**port, **M**inivan and **T**ruck)
  - Spent indicates how much person spent during a recent visit to the web site
  - Dependent attribute is *numerical*

Age	Car	Spent
20	M	\$200
30	M	\$150
25	T	\$300
30	S	\$220
40	S	\$400
20	T	\$80
30	M	\$100
25	M	\$125
40	M	\$500
20	S	\$420

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

## Types of Variables (Review)

- Numerical*: Domain is ordered and can be represented on the real line (e.g., age, income)
- Nominal* or *categorical*: Domain is a finite set without any natural ordering (e.g., occupation, marital status, race)
- Ordinal*: Domain is ordered, but absolute differences between values is unknown (e.g., preference scale, severity of an injury)

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

## Definitions

- Random variables  $X_1, \dots, X_k$  (*predictor variables*) and  $Y$  (*dependent variable*)
- $X_i$  has domain  $\text{dom}(X_i)$ ,  $Y$  has domain  $\text{dom}(Y)$
- $P$  is a probability distribution on  $\text{dom}(X_1) \times \dots \times \text{dom}(X_k) \times \text{dom}(Y)$   
Training database  $D$  is a random sample from  $P$
- A *predictor*  $d$  is a function  
 $d: \text{dom}(X_1) \dots \text{dom}(X_k) \rightarrow \text{dom}(Y)$

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

## Classification Problem

- If Y is categorical, the problem is a *classification problem*, and we use C instead of Y.  
 $|\text{dom}(C)| = J$ .
- C is called the *class label*, d is called a *classifier*.
- Take r be record randomly drawn from P.  
Define the *misclassification rate* of d:  
 $RT(d, P) = P(d(r.X_1, \dots, r.X_k) \neq r.C)$
- Problem definition: Given dataset D that is a random sample from probability distribution P, find classifier d such that  $RT(d, P)$  is minimized.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Regression Problem

- If Y is numerical, the problem is a *regression problem*.
- Y is called the dependent variable, d is called a *regression function*.
- Take r be record randomly drawn from P.  
Define mean squared error rate of d:  
 $RT(d, P) = E(r.Y - d(r.X_1, \dots, r.X_k))^2$
- Problem definition: Given dataset D that is a random sample from probability distribution P, find regression function d such that  $RT(d, P)$  is minimized.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Goals and Requirements

- **Goals:**
  - To produce an accurate classifier/regression function
  - To understand the structure of the problem
- **Requirements on the model:**
  - High accuracy
  - Understandable by humans, interpretable
  - Fast construction for very large training databases

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Different Types of Classifiers

- Linear discriminant analysis (LDA)
- Quadratic discriminant analysis (QDA)
- Density estimation methods
- Nearest neighbor methods
- Logistic regression
- Neural networks
- Fuzzy set theory
- Decision Trees

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

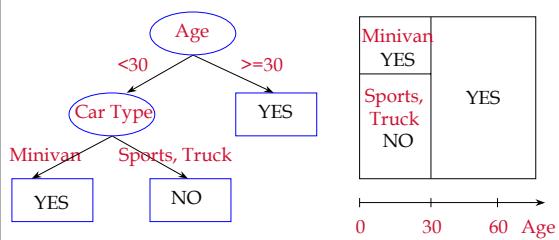
---

---

---

---

## What are Decision Trees?



Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Decision Trees

- A *decision tree*  $T$  encodes  $d$  (a classifier or regression function) in form of a tree.
- A node  $t$  in  $T$  without children is called a *leaf node*. Otherwise  $t$  is called an *internal node*.

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Internal Nodes

- Each internal node has an associated *splitting predicate*. Most common are binary predicates.  
Example predicates:
  - Age  $\leq 20$
  - Profession in {student, teacher}
  - $5000 * \text{Age} + 3 * \text{Salary} - 10000 > 0$

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

## Internal Nodes: Splitting Predicates

- Binary Univariate splits:
  - Numerical or ordered X:  $X \leq c$ ,  $c$  in  $\text{dom}(X)$
  - Categorical X:  $X$  in  $A$ ,  $A$  subset  $\text{dom}(X)$
- Binary Multivariate splits:
  - Linear combination split on numerical variables:
$$\sum a_i X_i \leq c$$
- k-ary ( $k > 2$ ) splits analogous

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

## Leaf Nodes

Consider leaf node  $t$

- Classification problem: Node  $t$  is labeled with one class label  $c$  in  $\text{dom}(C)$
- Regression problem: Two choices
  - Piecewise constant model:  
 $t$  is labeled with a constant  $y$  in  $\text{dom}(Y)$ .
  - Piecewise linear model:  
 $t$  is labeled with a linear model  
$$Y = y_t + \sum a_i X_i$$

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

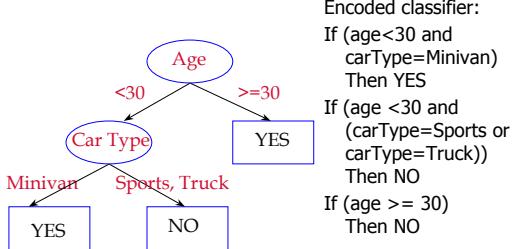
---

---

---

---

## Example



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

Encoded classifier:

```
If (age < 30 and
 carType = Minivan)
 Then YES
If (age < 30 and
 (carType = Sports or
 carType = Truck))
 Then NO
If (age >= 30)
 Then NO
```

## Evaluation of Misclassification Error

### Problem:

- In order to quantify the quality of a classifier  $d$ , we need to know its misclassification rate  $RT(d, P)$ .
- But unless we know  $P$ ,  $RT(d, P)$  is unknown.
- Thus we need to estimate  $RT(d, P)$  as good as possible.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## Resubstitution Estimate

The *Resubstitution estimate*  $R(d, D)$  estimates  $RT(d, P)$  of a classifier  $d$  using  $D$ :

- Let  $D$  be the training database with  $N$  records.
- $R(d, D) = 1/N \sum I(d(r.X) \neq r.C)$
- Intuition:  $R(d, D)$  is the proportion of training records that is misclassified by  $d$
- Problem with resubstitution estimate:  
Overly optimistic; classifiers that overfit the training dataset will have very low resubstitution error.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## Test Sample Estimate

- Divide D into  $D_1$  and  $D_2$
- Use  $D_1$  to construct the classifier  $d$
- Then use resubstitution estimate  $R(d, D_2)$  to calculate the estimated misclassification error of  $d$
- Unbiased and efficient, but removes  $D_2$  from training dataset D

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## V-fold Cross Validation

Procedure:

- Construct classifier  $d$  from D
- Partition D into V datasets  $D_1, \dots, D_V$
- Construct classifier  $d_i$  using  $D \setminus D_i$
- Calculate the estimated misclassification error  $R(d_i, D_i)$  of  $d_i$  using test sample  $D_i$

Final misclassification estimate:

- Weighted combination of individual misclassification errors:  
$$R(d, D) = 1/V \sum R(d_i, D_i)$$

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

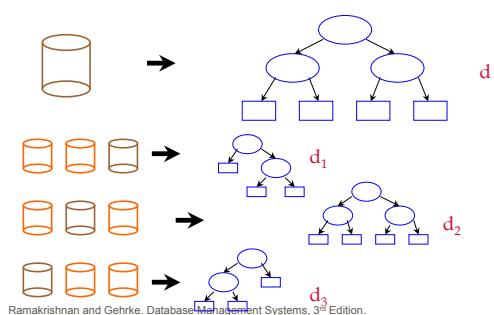
---

---

---

---

## Cross-Validation: Example



Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Cross-Validation

---

- Misclassification estimate obtained through cross-validation is usually nearly unbiased
- Costly computation (we need to compute  $d$ , and  $d_1, \dots, d_V$ ); computation of  $d_i$  is nearly as expensive as computation of  $d$
- Preferred method to estimate quality of learning algorithms in the machine learning literature

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

---

## Decision Tree Construction

---

- Top-down tree construction schema:
  - Examine training database and find best splitting predicate for the root node
  - Partition training database
  - Recurse on each child node

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

---

## Top-Down Tree Construction

---

**BuildTree**(Node  $t$ , Training database  $D$ ,  
Split Selection Method  $\mathcal{S}$ )

- (1) Apply  $\mathcal{S}$  to  $D$  to find splitting criterion
- (2) **if** ( $t$  is not a leaf node)
- (3) Create children nodes of  $t$
- (4) Partition  $D$  into children partitions
- (5) Recurse on each partition
- (6) **endif**

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

---

## Decision Tree Construction

- Three algorithmic components:
  - Split selection (CART, C4.5, QUEST, CHAID, CRUISE, ...)
  - Pruning (direct stopping rule, test dataset pruning, cost-complexity pruning, statistical tests, bootstrapping)
  - Data access (CLOUDS, SLIQ, SPRINT, RainForest, BOAT, UnPivot operator)

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

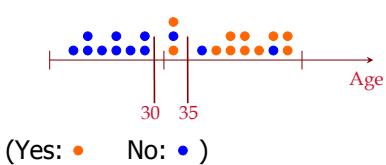
---

---

---

## Split Selection Method

- Numerical or ordered attributes: Find a split point that separates the (two) classes



Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

## Split Selection Method (Contd.)

- Categorical attributes: How to group?  
Sport: ●○○      Truck: ○○○      Minivan: ○○  
(Sport, Truck) -- (Minivan)      ●○○○○○      ○○  
(Sport) --- (Truck, Minivan)      ●○○      ●○○○○  
(Sport, Minivan) --- (Truck)      ●○○○○      ○○

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

## **Pruning Method**

---

- For a tree  $T$ , the misclassification rate  $R(T, P)$  and the mean-squared error rate  $R(T, P)$  depend on  $P$ , but not on  $D$ .
- The goal is to do well on records randomly drawn from  $P$ , not to do well on the records in  $D$
- If the tree is too large, it overfits  $D$  and does not model  $P$ . The pruning method selects the tree of the right size.

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

## **Data Access Method**

---

- Recent development: Very large training databases, both in-memory and on secondary storage
- Goal: Fast, efficient, and scalable decision tree construction, using the complete training database.

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

## **Split Selection Methods**

---

- Multitude of split selection methods in the literature
- In this workshop:
  - CART

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

## Split Selection Methods: CART

- Classification And Regression Trees (Breiman, Friedman, Ohlson, Stone, 1984; considered “the” reference on decision tree construction)
- Commercial version sold by Salford Systems ([www.salford-systems.com](http://www.salford-systems.com))
- Many other, slightly modified implementations exist (e.g., IBM Intelligent Miner implements the CART split selection method)

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## CART Split Selection Method

Motivation: We need a way to choose quantitatively between different splitting predicates

- Idea: Quantify the *impurity* of a node
- Method: Select splitting predicate that generates children nodes with minimum impurity from a space of possible splitting predicates

Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

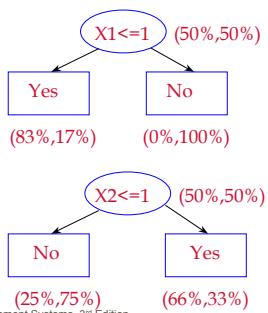
---

---

---

## Intuition: Impurity Function

X1	X2	Class
1	1	Yes
1	2	Yes
1	1	No
2	1	No
2	1	No
2	2	No
2	2	No



Ramakrishnan and Gehrke. Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Impurity Function

- Let  $p(j|t)$  be the proportion of class  $j$  training records at node  $t$
- Node impurity measure at node  $t$ :  
 $i(t) = \text{phi}(p(1|t), \dots, p(J|t))$
- $\text{phi}$  is symmetric
- Maximum value at arguments  $(J^{-1}, \dots, J^{-1})$  (maximum impurity)
- $\text{phi}(1,0,\dots,0) = \dots = \text{phi}(0,\dots,0,1) = 0$  (node has records of only one class; "pure" node)

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

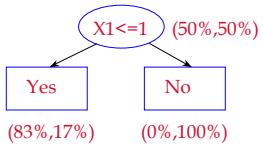
---

---

---

## Example

- Root node  $t$ :  
 $p(1|t)=0.5; p(2|t)=0.5$   
Left child node  $t_L$ :  
 $P(1|t)=0.83; p(2|t)=-.17$
- Impurity of root node:  
 $\text{phi}(0.5,0.5)$
- Impurity of left child node:  
 $\text{phi}(0.83,0.17)$
- Impurity of right child node:  
 $\text{phi}(0.0,1.0)$



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Goodness of a Split

Consider node  $t$  with impurity  $\text{phi}(t)$

The *reduction in impurity* through splitting predicate  $s$  ( $t$  splits into children nodes  $t_L$  with impurity  $\text{phi}(t_L)$  and  $t_R$  with impurity  $\text{phi}(t_R)$ ) is:

$$\Delta_{\text{phi}}(s,t) = \text{phi}(t) - p_L \text{phi}(t_L) - p_R \text{phi}(t_R)$$

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

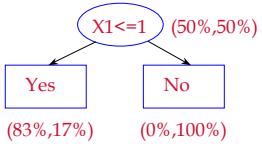
---

---

---

### Example (Contd.)

- Impurity of root node:  
 $\phi(0.5, 0.5)$
- Impurity of whole tree:  
 $0.6 * \phi(0.83, 0.17) + 0.4 * \phi(0, 1)$
- Impurity reduction:  
 $\phi(0.5, 0.5) - 0.6 * \phi(0.83, 0.17) - 0.4 * \phi(0, 1)$



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

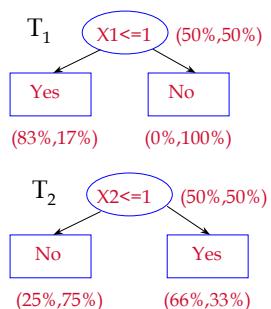
---

---

---

### Error Reduction as Impurity Function

- Possible impurity function:  
Resubstitution error  
 $R(T, D)$ .
- Example:  
 $R(\text{no tree}, D) = 0.5$   
 $R(T_1, D) = 0.6 * 0.17$   
 $R(T_2, D) = 0.4 * 0.25 + 0.6 * 0.33$



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

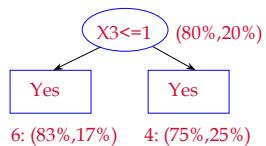
---

---

---

### Problems with Resubstitution Error

- Obvious problem:  
There are situations where no split can decrease impurity
- Example:  
 $R(\text{no tree}, D) = 0.2$   
 $R(T_1, D) = 0.6 * 0.17 + 0.4 * 0.25 = 0.2$



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

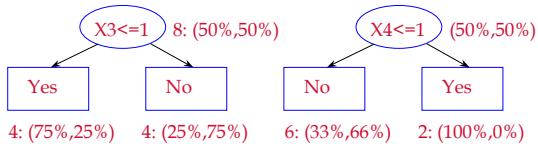
---

---

---

## Problems with Resubstitution Error

- More subtle problem:



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

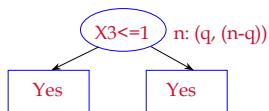
---

## Problems with Resubstitution Error

Root node:  $n$  records,  $q$  of class 1

Left child node:  $n_1$  records,  $q'$  of class 1

Right child node:  $n_2$  records,  $(q-q')$  of class 1,  
 $n_1+n_2 = n$



$n_1: (q'/n_1, (n_1-q')/n_1)$      $n_2: ((q-q')/n_2, (n_2-(q-q'))/n_2)$

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Problems with Resubstitution Error

Tree structure:

Root node:  $n$  records  $(q/n, (n-q))$

Left child:  $n_1$  records  $(q'/n_1, (n_1-q')/n_1)$

Right child:  $n_2$  records  $((q-q')/n_2, (n_2-q')/n_2)$

Impurity before split:

Error:  $q/n$

Impurity after split:

Left child:  $n_1/n * q'/n_1 = q'/n$

Right child:  $n_2/n * (q-q')/n_2 = (q-q')/n$

Total error:  $q/n + (q-q')/n = q/n$

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Problems with Resubstitution Error

Heart of the problem:

Assume two classes:

$$\begin{aligned}\text{phi}(p(1|t), p(2|t)) &= \text{phi}(p(1|t), 1-p(1|t)) \\ &= \text{phi}(p(1|t))\end{aligned}$$

Resubstitution error has the following property:

$$\text{phi}(p_1 + p_2) = \text{phi}(p_1) + \text{phi}(p_2)$$

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

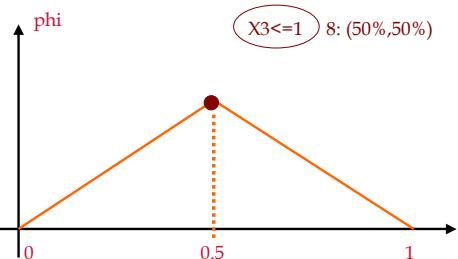
---

---

---

---

## Example: Only Root Node



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

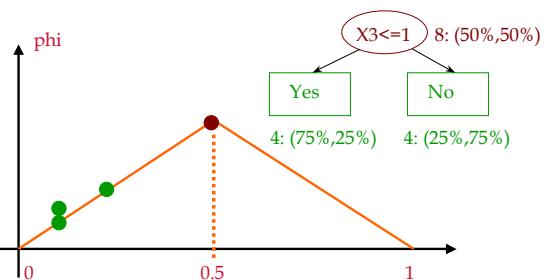
---

---

---

---

## Example: Split (75,25), (25,75)



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

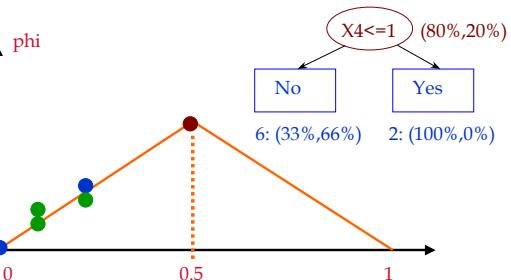
---

---

---

---

### Example: Split (33,66), (100,0)



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

### Remedy: Concavity

Use impurity functions that are concave:

$$\phi'' < 0$$

Example impurity functions

- Entropy:  
 $\phi(t) = - \sum p(j|t) \log(p(j|t))$
- Gini index:  
 $\phi(t) = \sum p(j|t)^2$

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

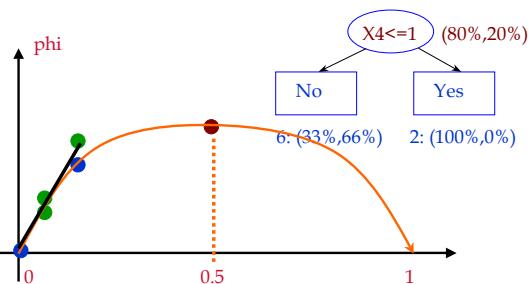
---

---

---

---

### Example Split With Concave Phi



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Nonnegative Decrease in Impurity

**Theorem:** Let  $\phi(p_1, \dots, p_j)$  be a strictly concave function on  $j=1, \dots, J$ ,  $\sum_j p_j = 1$ .

Then for any split  $s$ :

$$\Delta_{\phi}(s, t) \geq 0$$

With equality if and only if:

$$p(j|t_L) = p(j|t_R) = p(j|t), j = 1, \dots, J$$

**Note:** Entropy and gini-index are concave.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## CART Univariate Split Selection

- Use gini-index as impurity function
- For each numerical or ordered attribute  $X$ , consider all binary splits  $s$  of the form  
$$X \leq x$$
 where  $x \in \text{dom}(X)$
- For each categorical attribute  $X$ , consider all binary splits  $s$  of the form  
$$X \in A, \text{ where } A \subset \text{dom}(X)$$
- At a node  $t$ , select split  $s^*$  such that  $\Delta_{\phi}(s^*, t)$  is maximal over all  $s$  considered

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## CART: Shortcut for Categorical Splits

Computational shortcut if  $|Y|=2$ .

- **Theorem:** Let  $X$  be a categorical attribute with  $\text{dom}(X) = \{b_1, \dots, b_k\}$ ,  $|Y|=2$ ,  $\phi$  be a concave function, and let  
$$p(X=b_1) \leq \dots \leq p(X=b_k).$$
 Then the best split is of the form:  
$$X \in \{b_1, b_2, \dots, b_l\} \text{ for some } l < k$$
- **Benefit:** We need only to check  $k-1$  subsets of  $\text{dom}(X)$  instead of  $2^{(k-1)}-1$  subsets

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## CART Multivariate Split Selection

- For numerical predictor variables, examine splitting predicates  $s$  of the form:  
 $\sum_i a_i X_i \leq c$   
with the constraint:  
 $\sum_i a_i^2 = 1$
- Select splitting predicate  $s^*$  with maximum decrease in impurity.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Problems with CART Split Selection

- Biased towards variables with more splits ( $M$ -category variable has  $2^{M-1}-1$ ) possible splits, an  $M$ -valued ordered variable has  $(M-1)$  possible splits
- Computationally expensive for categorical variables with large domains

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Pruning Methods

- Test dataset pruning
- Direct stopping rule
- Cost-complexity pruning
- MDL pruning
- Pruning by randomization testing

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Top-Down and Bottom-Up Pruning

Two classes of methods:

- Top-down pruning: Stop growth of the tree at the right size. Need a statistic that indicates when to stop growing a subtree.
- Bottom-up pruning: Grow an overly large tree and then chop off subtrees that “overfit” the training data.

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Stopping Policies

A stopping policy indicates when further growth of the tree at a node  $t$  is counterproductive.

- All records are of the same class
- The attribute values of all records are identical
- All records have missing values
- At most one class has a number of records larger than a user-specified number
- All records go to the same child node if  $t$  is split (only possible with some split selection methods)

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Test Dataset Pruning

- Use an independent test sample  $D'$  to estimate the misclassification cost using the resubstitution estimate  $R(T, D')$  at each node
- Select the subtree  $T'$  of  $T$  with the smallest expected cost

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

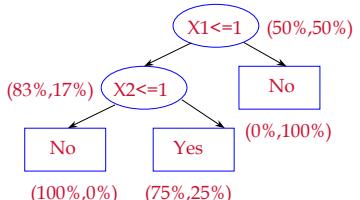
---

---

## Test Dataset Pruning Example

Test set:

X1	X2	Class
1	1	Yes
1	2	Yes
1	2	Yes
1	2	Yes
1	1	Yes
1	2	No
2	1	No
2	1	No
2	2	No
2	2	No



Only root: 10% misclassification

Full tree: 30% misclassification

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## Cost Complexity Pruning

(Breiman, Friedman, Olshen, Stone, 1984)

Some more tree notation

- $t$ : node in tree  $T$
- $\text{leaf}(T)$ : set of leaf nodes of  $T$
- $|\text{leaf}(T)|$ : number of leaf nodes of  $T$
- $T_t$ : subtree of  $T$  rooted at  $t$
- $\{t\}$ : subtree of  $T_t$  containing only node  $t$

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## Notation: Example

$$\text{leaf}(T) = \{t_1, t_2, t_3\}$$

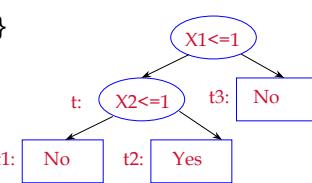
$$|\text{leaf}(T)| = 3$$

Tree rooted  
at node  $t$ :  $T_t$

Tree consisting  
of only node  $t$ :  $\{t\}$

$$\text{leaf}(T_t) = \{t_1, t_2\}$$

$$\text{leaf}(\{t\}) = \{t\}$$



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## Cost-Complexity Pruning

- Test dataset pruning is the ideal case, if we have a large test dataset. But:
  - We might not have a large test dataset
  - We want to use all available records for tree construction
- If we do not have a test dataset, we do not obtain "honest" classification error estimates
- Remember cross-validation: Re-use training dataset in a clever way to estimate the classification error.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Cost-Complexity Pruning

1. /\* cross-validation step \*/  
Construct tree T using D
2. Partition D into V subsets  $D_1, \dots, D_V$
3. for ( $i=1; i <= V; i++$ )  
    Construct tree  $T_i$  from  $(D \setminus D_i)$   
    Use  $D_i$  to calculate the estimate  $R(T_i, D \setminus D_i)$   
endfor
4. /\* estimation step \*/  
Calculate  $R(T, D)$  from  $R(T_i, D \setminus D_i)$

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

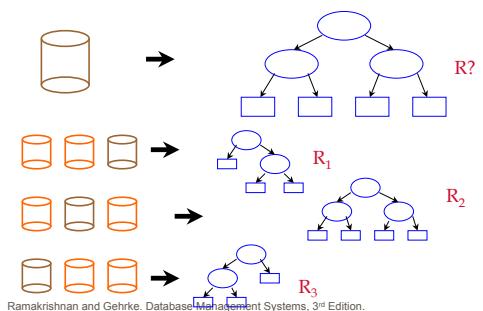
---

---

---

---

## Cross-Validation Step



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Cost-Complexity Pruning

- Problem: How can we relate the misclassification error of the CV-trees to the misclassification error of the large tree?
- Idea: Use a parameter that has the same meaning over different trees, and relate trees with similar parameter settings.
- Such a parameter is the cost-complexity of the tree.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

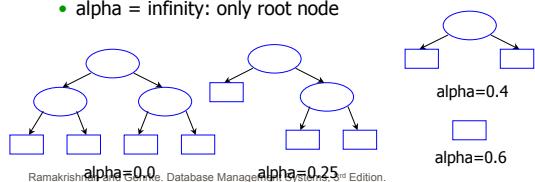
---

---

---

## Cost-Complexity Pruning

- Cost complexity of a tree  $T$ :  
 $R_{\alpha}(T) = R(T) + \alpha |\text{leaf}(T)|$
- For each  $\alpha$ , there is a tree that minimizes the cost complexity:
  - $\alpha = 0$ : full tree
  - $\alpha = \infty$ : only root node



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Cost-Complexity Pruning

- When should we prune the subtree rooted at  $t$ ?
  - $R_{\alpha}(\{t\}) = R(t) + \alpha$
  - $R_{\alpha}(T_t) = R(T_t) + \alpha |\text{leaf}(T_t)|$
  - Define  
$$g(t) = (R(t) - R(T_t)) / (|\text{leaf}(T_t)| - 1)$$
- Each node has a critical value  $g(t)$ :
  - $\alpha < g(t)$ : leave subtree  $T_t$  rooted at  $t$
  - $\alpha \geq g(t)$ : prune subtree rooted at  $t$  to  $\{t\}$
- For each  $\alpha$  we obtain a unique minimum cost-complexity tree.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

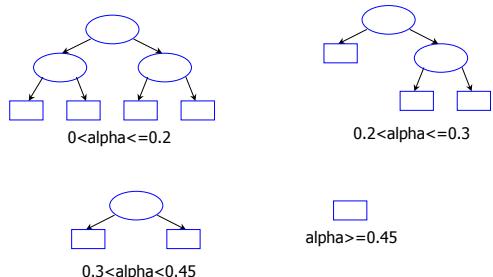
---

---

---

---

## Example Revisited



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Cost Complexity Pruning

1. Let  $T^1 > T^2 > \dots > \{t\}$  be the nested cost-complexity sequence of subtrees of  $T$  rooted at  $t$ .  
Let  $\alpha_1 < \dots < \alpha_k$  be the sequence of associated critical values of  $\alpha$ . Define  $\alpha_k = \sqrt{\alpha_k * \alpha_{k+1}}$
2. Let  $T_i$  be the tree grown from  $D \setminus D_i$
3. Let  $T_i(\alpha_k)$  be the minimal cost-complexity tree for  $\alpha_k$

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Cost Complexity Pruning

4. Let  $R'(T_i)(\alpha_k)$  be the misclassification cost of  $T_i(\alpha_k)$  based on  $D_i$
5. Define the V-fold cross-validation misclassification estimate as follows:  
$$R^*(T^k) = \frac{1}{V} \sum_i R'(T_i)(\alpha_k)$$
6. Select the subtree with the smallest estimated CV error

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## k-SE Rule

- Let  $T^*$  be the subtree of  $T$  that minimizes the misclassification error  $R(T_k)$  over all  $k$
- But  $R(T_k)$  is only an estimate:
  - Estimate the estimated standard error  $SE(R(T^*))$  of  $R(T^*)$
  - Let  $T^{**}$  be the smallest tree such that  $R(T^{**}) \leq R(T^*) + k * SE(R(T^*))$ ; use  $T^{**}$  instead of  $T^*$
- Intuition: A smaller tree is easier to understand.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Cost Complexity Pruning

Advantages:

- No independent test dataset necessary
- Gives estimate of misclassification error, and chooses tree that minimizes this error

Disadvantages:

- Originally devised for small datasets; is it still necessary for large datasets?
- Computationally very expensive for large datasets (need to grow  $V$  trees from nearly all the data)

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Missing Values

- What is the problem?
  - During computation of the splitting predicate, we can selectively ignore records with missing values (note that this has some problems)
  - But if a record  $r$  misses the value of the variable in the splitting attribute,  $r$  can not participate further in tree construction

Algorithms for missing values address this problem.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Mean and Mode Imputation

Assume record  $r$  has missing value  $r.X$ , and splitting variable is  $X$ .

- Simplest algorithm:
  - If  $X$  is numerical (categorical), impute the overall mean (mode)
- Improved algorithm:
  - If  $X$  is numerical (categorical), impute the  $\text{mean}(X|t.C)$  (the  $\text{mode}(X|t.C)$ )

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Decision Trees: Summary

- Many applications of decision trees
- There are many algorithms available for:
  - Split selection
  - Pruning
  - Handling Missing Values
  - Data Access
- Decision tree construction still active research area (after 20+ years!)
- Challenges: Performance, scalability, evolving datasets, new applications

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Lecture Overview

- Data Mining I: Decision Trees
- Data Mining II: Clustering
- Data Mining III: Association Analysis

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Supervised Learning

- $F(x)$ : true function (usually not known)
  - $D$ : training sample drawn from  $F(x)$

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

## Supervised Learning

- $F(x)$ : true function (usually not known)
  - $D$ : training sample  $(x, F(x))$



Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

## Supervised Learning

### Well-defined goal:

Learn  $G(x)$  that is a good approximation to  $F(x)$  from training sample D

Well-defined error metrics:

---

## Accuracy, RMSE, ROC, ...

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition

## Supervised Learning

Training dataset:

57,M,195,0,125,95,39,25,0,1,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0	0
78,M,160,1,130,100,37,40,1,0,0,0,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0	1
69,F,180,0,115,85,40,22,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0	0
18,M,165,0,110,80,41,30,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
54,F,135,0,115,95,39,35,1,1,0,0,0,1,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,0,0	1
84,F,210,1,135,105,39,24,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0
89,F,135,0,120,95,36,28,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0,0	1
49,M,195,0,115,85,39,32,0,0,0,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0	0
40,M,205,0,115,90,37,18,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0
74,M,250,1,130,100,38,26,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
77,F,140,0,125,100,40,30,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0	0

Test dataset:

71,M,160,1,130,105,38,20,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0

?

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## Un-Supervised Learning

Training dataset:

57,M,195,0,125,95,39,25,0,1,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0	0
78,M,160,1,130,100,37,40,1,0,0,0,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0	1
69,F,180,0,115,85,40,22,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0	0
18,M,165,0,110,80,41,30,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
54,F,135,0,115,95,39,35,1,1,0,0,0,1,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,0	1
84,F,210,1,135,105,39,24,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
89,F,135,0,120,95,36,28,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0	1
49,M,195,0,115,85,39,32,0,0,0,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0	0
40,M,205,0,115,90,37,18,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0
74,M,250,1,130,100,38,26,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
77,F,140,0,125,100,40,30,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0	0

Test dataset:

71,M,160,1,130,105,38,20,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0

?

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## Un-Supervised Learning

Training dataset:

57,M,195,0,125,95,39,25,0,1,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0	0
78,M,160,1,130,100,37,40,1,0,0,0,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0	1
69,F,180,0,115,85,40,22,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0	0
18,M,165,0,110,80,41,30,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
54,F,135,0,115,95,39,35,1,1,0,0,0,1,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,0	1
84,F,210,1,135,105,39,24,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
89,F,135,0,120,95,36,28,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,0,0	1
49,M,195,0,115,85,39,32,0,0,0,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0	0
40,M,205,0,115,90,37,18,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0
74,M,250,1,130,100,38,26,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	1
77,F,140,0,125,100,40,30,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0	0

Test dataset:

71,M,160,1,130,105,38,20,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0

?

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## Un-Supervised Learning

Data Set:

```
57,M,195,0,125,95,39,25,0,1,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0
78,M,160,1,130,100,37,40,1,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
69,F,180,0,115,85,40,22,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
18,M,165,0,110,80,41,30,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
54,F,135,0,115,95,39,35,1,1,0,0,0,1,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0,0
84,F,210,1,135,105,39,24,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
89,F,135,0,120,95,36,28,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,0
49,M,195,0,115,85,39,32,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0
40,M,205,0,115,90,37,18,0
74,M,250,1,130,100,38,26,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
77,F,140,0,125,100,40,30,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1
```

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

---

---

## Supervised vs. Unsupervised Learning

### Supervised

- $y=F(x)$ : true function
- D: labeled training set
- D:  $\{x_i, F(x_i)\}$
- Learn:  
 $G(x)$ : model trained to predict labels D
- Goal:  
 $E[(F(x)-G(x))^2] \approx 0$
- Well defined criteria:  
Accuracy, RMSE, ...

### Unsupervised

- Generator: true model
- D: unlabeled data sample
- D:  $\{x_i\}$
- Learn  
???????????
- Goal:  
???????????
- Well defined criteria:  
???????????

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

---

---

## What to Learn/Discover?

- Statistical Summaries
- Generators
- Density Estimation
- Patterns/Rules
- Associations (see previous segment)
- Clusters/Groups (this segment)
- Exceptions/Outliers
- Changes in Patterns Over Time or Location

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

---

---

## Clustering: Unsupervised Learning

- Given:
  - Data Set D (training set)
  - Similarity/distance metric/information
- Find:
  - Partitioning of data
  - Groups of similar/close items

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

## Similarity?

- Groups of similar customers
  - Similar demographics
  - Similar buying behavior
  - Similar health
- Similar products
  - Similar cost
  - Similar function
  - Similar store
  - ...
- Similarity usually is domain/problem specific

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

## Distance Between Records

- $d$ -dim vector space representation and distance metric

$r_1:$  57,M,195,0,125,95,39,25,0,1,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0  
 $r_2:$  78,M,160,1,130,100,37,40,1,0,0,0,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0  
...  
 $r_N:$  18,M,165,0,110,80,41,30,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Distance ( $r_1, r_2$ ) = ???

- Pairwise distances between points (no  $d$ -dim space)

- Similarity/dissimilarity matrix  
(upper or lower diagonal)

1	2	3	4	5	6	7	8	9	10
-	d	d	d	d	d	d	d	d	d
- Distance:      0 = near,       $\infty$  = far
- Similarity:      0 = far,       $\infty$  = near

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

## Properties of Distances: Metric Spaces

- A metric space is a set  $S$  with a global distance function  $d$ . For every two points  $x, y$  in  $S$ , the distance  $d(x, y)$  is a nonnegative real number.
- A metric space must also satisfy
  - $d(x, y) = 0$  iff  $x = y$
  - $d(x, y) = d(y, x)$  (**symmetry**)
  - $d(x, y) + d(y, z) \geq d(x, z)$  (**triangle inequality**)

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Minkowski Distance ( $L_p$ Norm)

- Consider two records  $x=(x_1, \dots, x_d), y=(y_1, \dots, y_d)$ :  
$$d(x, y) = \sqrt[p]{|x_1 - y_1|^p + |x_2 - y_2|^p + \dots + |x_d - y_d|^p}$$

Special cases:

- $p=1$ : Manhattan distance  
$$d(x, y) = |x_1 - y_1| + |x_2 - y_2| + \dots + |x_p - y_p|$$
- $p=2$ : Euclidean distance  
$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_d - y_d)^2}$$

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Only Binary Variables

2x2 Table:

	0	1	Sum
0	a	b	a+b
1	c	d	c+d
Sum	a+c	b+d	a+b+c+d

- Simple matching coefficient:  
$$d(x, y) = \frac{b + c}{a + b + c + d}$$
 (symmetric)
- Jaccard coefficient:  
$$d(x, y) = \frac{b + c}{b + c + d}$$
 (asymmetric)

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

## Nominal and Ordinal Variables

- **Nominal:** Count number of matching variables
  - m: # of matches, d: total # of variables

$$d(x, y) = \frac{d - m}{d}$$

- **Ordinal:** Bucketize and transform to numerical:
  - Consider record x with value  $x_i$  for  $i^{\text{th}}$  attribute of record x; new value  $x'_i$ :

$$x'_i = \frac{x_i - 1}{\text{dom}(X_i) - 1}$$

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Mixtures of Variables

- Weigh each variable differently
- Can take "importance" of variable into account (although usually hard to quantify in practice)

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Clustering: Informal Problem Definition

### Input:

- A data set of  $N$  records each given as a  $d$ -dimensional data feature vector.

### Output:

- Determine a natural, useful "partitioning" of the data set into a number of ( $k$ ) clusters and noise such that we have:
  - High similarity of records within each cluster (intra-cluster similarity)
  - Low similarity of records between clusters (inter-cluster similarity)

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Types of Clustering

- Hard Clustering:
  - Each object is in one and only one cluster
- Soft Clustering:
  - Each object has a probability of being in each cluster

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Clustering Algorithms

- Partitioning-based clustering
  - K-means clustering
  - K-medoids clustering
  - EM (expectation maximization) clustering
- Hierarchical clustering
  - Divisive clustering (top down)
  - Agglomerative clustering (bottom up)
- Density-Based Methods
  - Regions of dense points separated by sparser regions of relatively low density

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## K-Means Clustering Algorithm

Initialize k cluster centers

**Do**

**Assignment step:** Assign each data point to its closest cluster center

**Re-estimation step:** Re-compute cluster centers

**While** (there are still changes in the cluster centers)

Visualization at:

- <http://www.delft-cluster.nl/textminer/theory/kmeans/kmeans.html>

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Issues

Why is K-Means working:

- How does it find the cluster centers?
- Does it find an optimal clustering
- What are good starting points for the algorithm?
- What is the right number of cluster centers?
- How do we know it will terminate?

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## K-Means: Distortion

- Communication between sender and receiver
- Sender encodes dataset:  $x_i \rightarrow \{1, \dots, k\}$
- Receiver decodes dataset:  $j \rightarrow \text{center}_j$
- Distortion:  $D = \sum_1^N (x_i - \text{center}_{\text{encode}(x_i)})^2$
- A good clustering has minimal distortion.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Properties of the Minimal Distortion

- Recall: Distortion  $D = \sum_1^N (x_i - \text{center}_{\text{encode}(x_i)})^2$
- Property 1: Each data point  $x_i$  is encoded by its nearest cluster center  $\text{center}_j$ . (Why?)
- Property 2: When the algorithm stops, the partial derivative of the Distortion with respect to each center attribute is zero.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Property 2 Followed Through

- Calculating the partial derivative:

$$D = \sum_{i=1}^N (x_i - center_{encode(x)})^2 = \sum_{j=1}^k \sum_{i \in Cluster(center_j)} (x_i - center_j)^2$$
$$\frac{\partial D}{\partial center_j} = \frac{\partial}{\partial center_j} \sum_{i \in Cluster(center_j)} (x_i - center_j)^2 = -2 \sum_{i \in Cluster(center_j)} (x_i - center_j) = 0$$

- Thus at the minimum:

$$center_j = \frac{1}{|\{i \in Cluster(center_j)\}|} \sum_{i \in Cluster(center_j)} x_i$$

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

## K-Means Minimal Distortion Property

- Property 1: Each data point  $x_i$  is encoded by its nearest cluster center  $center_j$
  - Property 2: Each center is the centroid of its cluster.
- 
- How do we improve a configuration:
    - Change encoding (encode a point by its nearest cluster center)
    - Change the cluster center (make each center the centroid of its cluster)

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

## K-Means Minimal Distortion Property (Contd.)

- Termination? Count the number of distinct configurations ...
- Optimality? We might get stuck in a local optimum.
  - Try different starting configurations.
  - Choose the starting centers smart.
- Choosing the number of centers?
  - Hard problem. Usually choose number of clusters that minimizes some criterion.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

## K-Means: Summary

---

- Advantages:
  - Good for exploratory data analysis
  - Works well for low-dimensional data
  - Reasonably scalable
- Disadvantages
  - Hard to choose k
  - Often clusters are non-spherical

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

---

---

## K-Medoids

---

- Similar to K-Means, but for categorical data or data in a non-vector space.
- Since we cannot compute the cluster center (think text data), we take the “most representative” data point in the cluster.
- This data point is called the medoid (the object that “lies in the center”).

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

---

---

## Agglomerative Clustering

---

### Algorithm:

- Put each item in its own cluster (all singletons)
- Find all pairwise distances between clusters
- Merge the two *closest* clusters
- Repeat until everything is in one cluster

### Observations:

- Results in a hierarchical clustering
- Yields a clustering for each possible number of clusters
- Greedy clustering: Result is not “optimal” for any cluster size

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

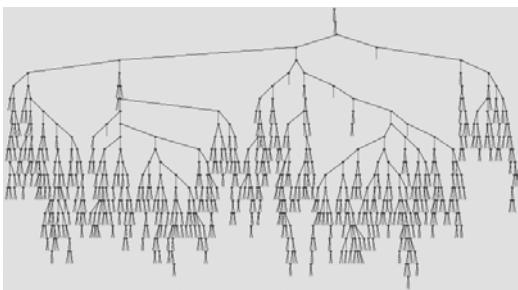
---

---

---

---

## Agglomerative Clustering Example



Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Density-Based Clustering

- A cluster is defined as a connected dense component.
- Density is defined in terms of number of neighbors of a point.
- We can find clusters of arbitrary shape



Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## DBSCAN

### E-neighborhood of a point

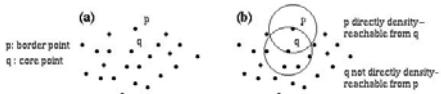
- $\text{NE}(p) = \{q \in D \mid \text{dist}(p,q) \leq E\}$

### Core point

- $|\text{NE}(q)| \geq \text{MinPts}$

### Directly density-reachable

- A point  $p$  is *directly density-reachable* from a point  $q$  wrt.  $E$ ,  $\text{MinPts}$  if
  - 1)  $p \in \text{NE}(q)$  and
  - 2)  $|\text{NE}(q)| \geq \text{MinPts}$  (core point condition).



Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

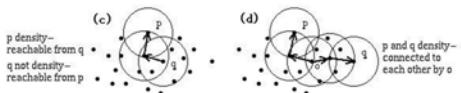
## DBSCAN

### Density-reachable

- A point  $p$  is density-reachable from a point  $q$  wrt.  $E$  and  $\text{MinPts}$  if there is a chain of points  $p_1, \dots, p_n$ ,  $p_1 = q$ ,  $p_n = p$  such that  $p_{i+1}$  is directly density-reachable from  $p_i$ .

### Density-connected

- A point  $p$  is density-connected to a point  $q$  wrt.  $E$  and  $\text{MinPts}$  if there is a point  $o$  such that both,  $p$  and  $q$  are density-reachable from  $o$  wrt.  $E$  and  $\text{MinPts}$ .



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## DBSCAN

### Cluster

- A cluster  $C$  satisfies:
  - $\forall p, q: \text{if } p \in C \text{ and } q \text{ is density-reachable from } p \text{ wrt. } E \text{ and } \text{MinPts}, \text{ then } q \in C$ . **(Maximality)**
  - $\forall p, q \in C: p \text{ is density-connected to } q \text{ wrt. } E \text{ and } \text{MinPts}$ . **(Connectivity)**

### Noise

Those points not belonging to any cluster

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## DBSCAN

Can show

- (1) Every density-reachable set is a cluster:

The set  
 $O = \{o \mid o \text{ is density-reachable from } p \text{ wrt. } \epsilon\text{-eps and } \text{MinPts}\}$   
is a cluster wrt.  $\epsilon\text{-eps}$  and  $\text{MinPts}$ .

- (2) Every cluster is a density-reachable set:

Let  $C$  be a cluster wrt.  $\epsilon\text{-eps}$  and  $\text{MinPts}$  and let  $p$  be any point in  $C$  with  $|N_{\epsilon\text{-eps}}(p)| \geq \text{MinPts}$ . Then  $C$  equals to the set  
 $O = \{o \mid o \text{ is density-reachable from } p \text{ wrt. } \epsilon\text{-eps and } \text{MinPts}\}$ .

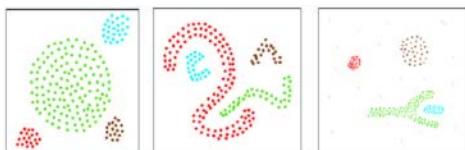
This motivates the following algorithm:

- For each point, DBSCAN determines the  $\epsilon$ -environment and checks whether it contains more than  $\text{MinPts}$  data points
- If so, it labels it with a cluster number
- If a neighbor  $q$  of a point  $p$  has already a cluster number, associate this number with  $p$

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## DBSCAN

---



Arbitrary shape clusters found by DBSCAN

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

---

---

## DBSCAN: Summary

---

- Advantages:
  - Finds clusters of arbitrary shapes
- Disadvantages:
  - Targets low dimensional spatial data
  - Hard to visualize for >2-dimensional data
  - Needs clever index to be scalable
  - How do we set the magic parameters?

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

---

---

## Lecture Overview

---

- Data Mining I: Decision Trees
- Data Mining II: Clustering
- Data Mining III: Association Analysis

Ramakrishnan and Gehrke. Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

---

---

---

## Market Basket Analysis

- Consider shopping cart filled with several items
- Market basket analysis tries to answer the following questions:
  - Who makes purchases?
  - What do customers buy together?
  - In what order do customers purchase items?

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Market Basket Analysis

Given:

- A database of customer transactions
- Each transaction is a set of items
- Example:  
Transaction with TID 111 contains items {Pen, Ink, Milk, Juice}

TID	CID	Date	Item	Qty
111	201	5/1/99	Pen	2
111	201	5/1/99	Ink	1
111	201	5/1/99	Milk	3
111	201	5/1/99	Juice	6
112	105	6/3/99	Pen	1
112	105	6/3/99	Ink	1
112	105	6/3/99	Milk	1
113	106	6/5/99	Pen	1
113	106	6/5/99	Milk	1
114	201	7/1/99	Pen	2
114	201	7/1/99	Ink	2
114	201	7/1/99	Juice	4

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Market Basket Analysis (Contd.)

- Cooccurrences
  - 80% of all customers purchase items X, Y and Z together.
- Association rules
  - 60% of all customers who purchase X and Y also buy Z.
- Sequential patterns
  - 60% of customers who first buy X also purchase Y within three weeks.

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Confidence and Support

We prune the set of all possible association rules using two interestingness measures:

- **Confidence** of a rule:
  - $X \rightarrow Y$  has confidence  $c$  if  $P(Y|X) = c$
- **Support** of a rule:
  - $X \rightarrow Y$  has support  $s$  if  $P(XY) = s$

We can also define

- **Support** of an itemset (a cooccurrence)  $XY$ :
  - $XY$  has support  $s$  if  $P(XY) = s$

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

## Example

Examples:

- $\{\text{Pen}\} \Rightarrow \{\text{Milk}\}$   
Support: 75%  
Confidence: 75%
- $\{\text{Ink}\} \Rightarrow \{\text{Pen}\}$   
Support: 100%  
Confidence: 100%

TID	CID	Date	Item	Qty
111	201	5/1/99	Pen	2
111	201	5/1/99	Ink	1
111	201	5/1/99	Milk	3
111	201	5/1/99	Juice	6
112	105	6/3/99	Pen	1
112	105	6/3/99	Ink	1
112	105	6/3/99	Milk	1
113	106	6/5/99	Pen	1
113	106	6/5/99	Milk	1
114	201	7/1/99	Pen	2
114	201	7/1/99	Ink	2
114	201	7/1/99	Juice	4

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

## Example

- Find all itemsets with support  $\geq 75\%$ ?

TID	CID	Date	Item	Qty
111	201	5/1/99	Pen	2
111	201	5/1/99	Ink	1
111	201	5/1/99	Milk	3
111	201	5/1/99	Juice	6
112	105	6/3/99	Pen	1
112	105	6/3/99	Ink	1
112	105	6/3/99	Milk	1
113	106	6/5/99	Pen	1
113	106	6/5/99	Milk	1
114	201	7/1/99	Pen	2
114	201	7/1/99	Ink	2
114	201	7/1/99	Juice	4

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

---

## Example

- Can you find all association rules with support  $\geq 50\%$ ?

TID	CID	Date	Item	Qty
111	201	5/1/99	Pen	2
111	201	5/1/99	Ink	1
111	201	5/1/99	Milk	3
111	201	5/1/99	Juice	6
112	105	6/3/99	Pen	1
112	105	6/3/99	Ink	1
112	105	6/3/99	Milk	1
113	106	6/5/99	Pen	1
113	106	6/5/99	Milk	1
114	201	7/1/99	Pen	2
114	201	7/1/99	Ink	2
114	201	7/1/99	Juice	4

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## Market Basket Analysis: Applications

- Sample Applications
  - Direct marketing
  - Fraud detection for medical insurance
  - Floor/shelf planning
  - Web site layout
  - Cross-selling

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## Applications of Frequent Itemsets

- Market Basket Analysis
- Association Rules
- Classification (especially: text, rare classes)
- Seeds for construction of Bayesian Networks
- Web log analysis
- Collaborative filtering

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## Association Rule Algorithms

- More abstract problem redux
- Breadth-first search
- Depth-first search

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

---

---

---

---

## Problem Redux

### Abstract:

- A set of items  $\{1, 2, \dots, k\}$
- A database of transactions (itemsets)  $D = \{T_1, T_2, \dots, T_n\}$ ,  $T_j$  subset  $\{1, 2, \dots, k\}$

### GOAL:

Find all itemsets that appear in at least  $x$  transactions  
("appear in" == "are subsets of")

I subset T: T **supports** I

For an itemset I, the number of transactions it appears in is called the **support** of I.

$x$  is called the **minimum support**.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

### Concrete:

- $I = \{\text{milk, bread, cheese, ...}\}$
- $D = \{\{\text{milk, bread, cheese}\}, \{\text{bread, cheese, juice}\}, \dots\}$

### GOAL:

Find all itemsets that appear in at least 1000 transactions

$\{\text{milk, bread, cheese}\}$  supports  $\{\text{milk, bread}\}$

---

---

---

---

---

---

---

## Problem Redux (Contd.)

### Definitions:

- An itemset is **frequent** if it is a subset of at least  $x$  transactions. (FI.)
- An itemset is **maximally frequent** if it is frequent and it does not have a frequent superset. (MFI.)

GOAL: Given  $x$ , find all frequent (maximally frequent) itemsets (to be stored in the **FI (MFI)**).

Obvious relationship:  
MFI subset FI

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

### Example:

$D = \{\{1, 2, 3\}, \{1, 2, 3\}, \{1, 2, 3\}, \{1, 2, 4\}\}$

Minimum support  $x = 3$

$\{1, 2\}$  is frequent

$\{1, 2, 3\}$  is maximal frequent

$\text{Support}(\{1, 2\}) = 4$

All maximal frequent itemsets:  
 $\{1, 2, 3\}$

---

---

---

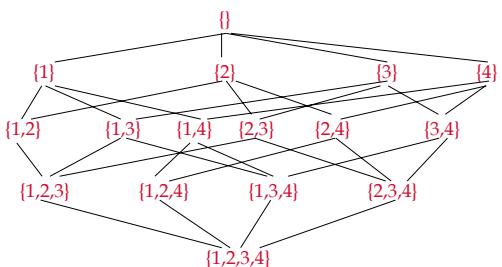
---

---

---

---

## The Itemset Lattice



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

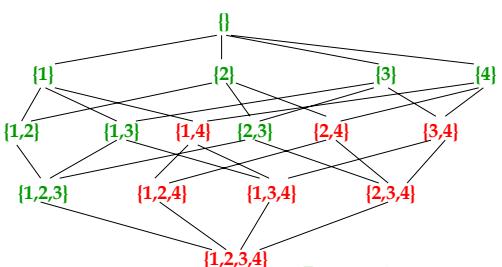
---

---

---

---

## Frequent Itemsets



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

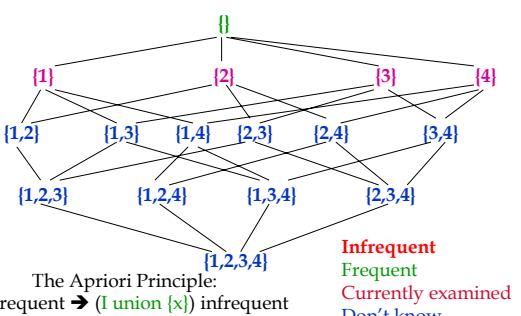
---

---

---

---

## Breath First Search: 1-Itemsets



The Apriori Principle:  
I infrequent  $\rightarrow$  (I union {x}) infrequent  
Infrequent  
Frequent  
Currently examined  
Don't know

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

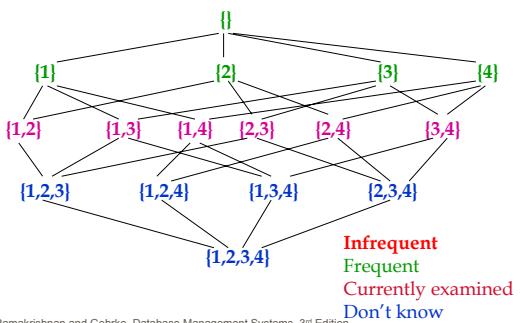
---

---

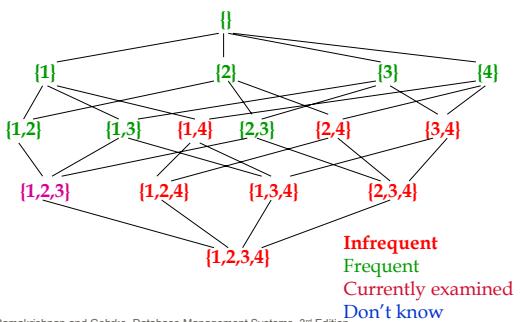
---

---

### Breath First Search: 2-Itemsets



### Breath First Search: 3-Itemsets

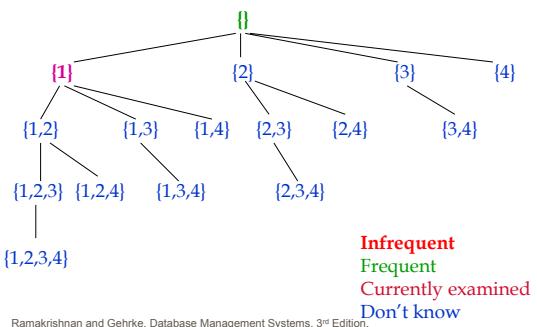


### Breadth First Search: Remarks

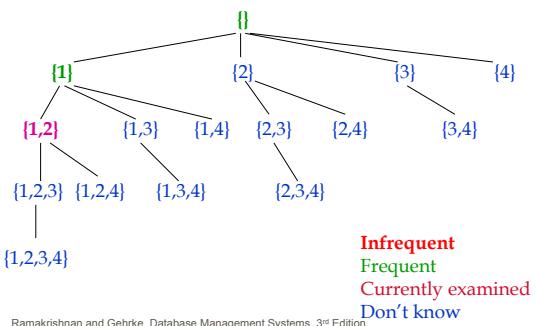
- We prune infrequent itemsets and avoid to count them
- To find an itemset with  $k$  items, we need to count all  $2^k$  subsets

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

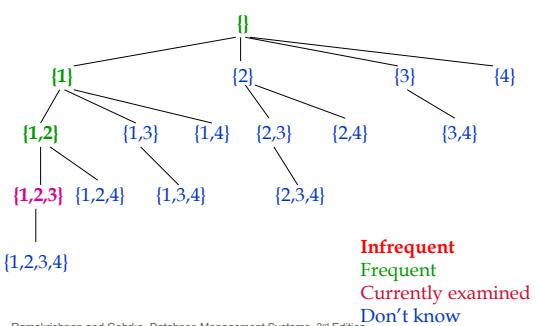
### Depth First Search (1)



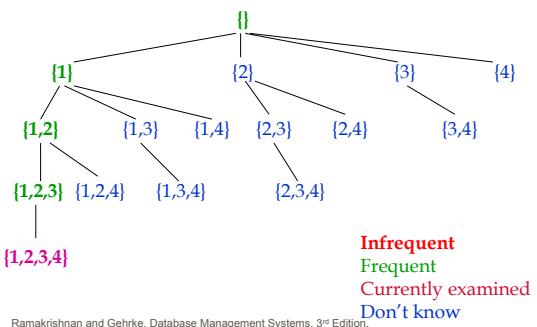
### Depth First Search (2)



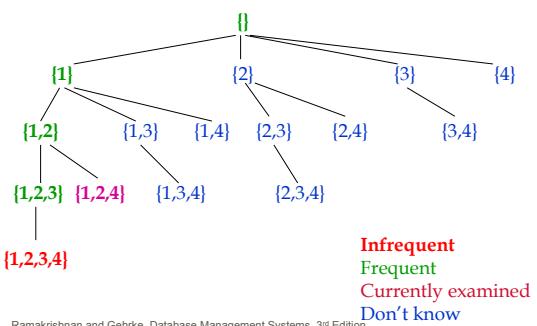
### Depth First Search (3)



### Depth First Search (4)



### Depth First Search (5)



### Depth First Search: Remarks

- We prune frequent itemsets and avoid counting them (works only for maximal frequent itemsets)
- To find an itemset with  $k$  items, we need to count  $k$  prefixes

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## BFS Versus DFS

### Breadth First Search

- Prunes infrequent itemsets
- Uses anti-monotonicity: Every superset of an infrequent itemset is infrequent

### Depth First Search

- Prunes frequent itemsets
- Uses monotonicity: Every subset of a frequent itemset is frequent

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Extensions

### • Imposing constraints

- Only find rules involving the dairy department
- Only find rules involving expensive products
- Only find “expensive” rules
- Only find rules with “whiskey” on the right hand side
- Only find rules with “milk” on the left hand side
- Hierarchies on the items
- Calendars (every Sunday, every 1<sup>st</sup> of the month)

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Itemset Constraints

### Definition:

- A *constraint* is an arbitrary property of itemsets.

### Examples:

- The itemset has support greater than 1000.
- No element of the itemset costs more than \$40.
- The items in the set average more than \$20.

### Goal:

- Find all itemsets satisfying a given constraint  $\mathbf{P}$ .

### “Solution”:

- If  $\mathbf{P}$  is a support constraint, use the *Apriori* Algorithm.

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

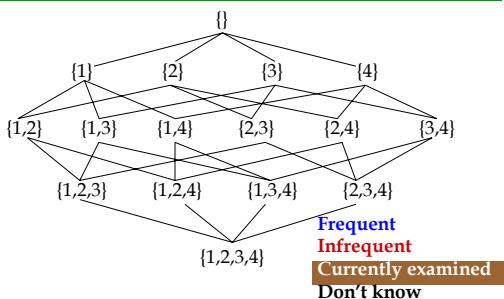
---

---

---

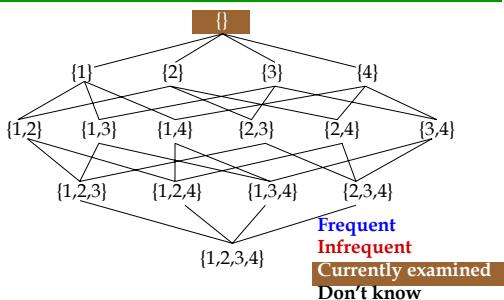
---

### Negative Pruning in Apriori



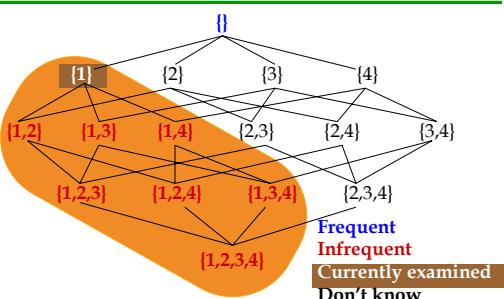
Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

### Negative Pruning in Apriori



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

### Negative Pruning in Apriori



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## Two Trivial Observations

- *Apriori* can be applied to any constraint **P** that is **antimonotone**.
  - Start from the empty set.
  - Prune **supersets** of sets that do not satisfy **P**.
- Itemset lattice is a **boolean algebra**, so *Apriori* also applies to a **monotone Q**.
  - Start from set of all items instead of empty set.
  - Prune **subsets** of sets that do not satisfy **Q**.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

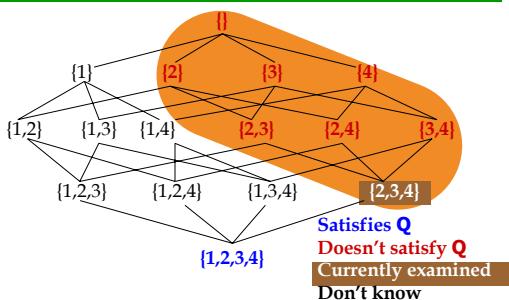
---

---

---

---

## Negative Pruning a Monotone Q



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

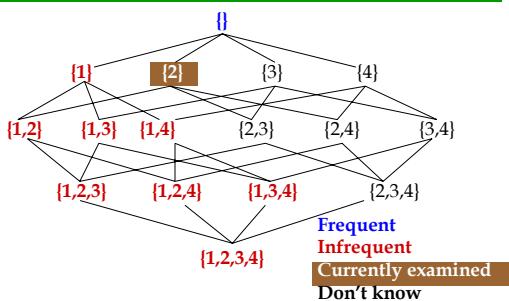
---

---

---

---

## Positive Pruning in *Apriori*



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

---

---

---

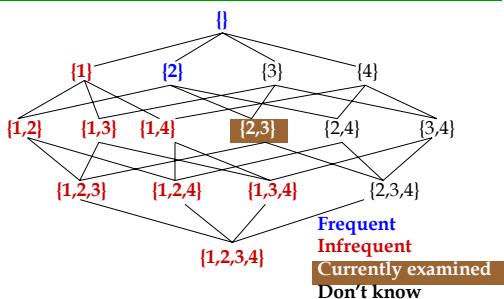
---

---

---

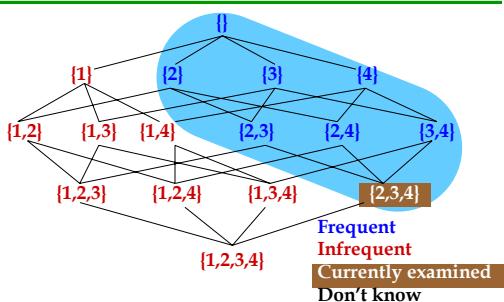
---

### Positive Pruning in Apriori



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

### Positive Pruning in Apriori



Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

### Classifying Constraints

Antimonotone:      Monotone:

- $\text{support}(I) > 1000$
- $\text{max}(I) < 100$

  - $\text{sum}(I) > 3$
  - $\text{min}(I) < 40$

Neither:

- $\text{average}(I) > 50$
- $\text{variance}(I) < 2$
- $3 < \text{sum}(I) < 50$

These are the constraints we really want.

Ramakrishnan and Gehrke, Database Management Systems, 3rd Edition.

## The Problem Redux

### Current Techniques:

- Approximate the difficult constraints.
- **Monotone** approximations are common.

### New Goal:

- Given constraints **P** and **Q**, with **P antimonotone** (support) and **Q monotone** (statistical constraint).
- Find all itemsets that satisfy both **P** and **Q**.

### Recent solutions:

- Newer algorithms can handle **both P and Q**

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

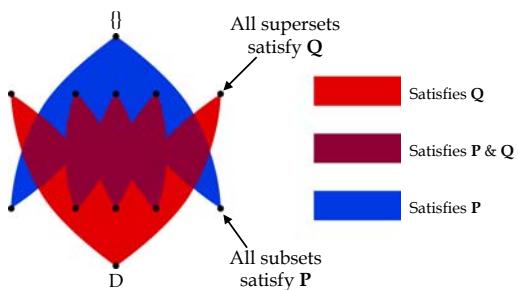
---

---

---

---

## Conceptual Illustration of Problem



Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Applications

- Spatial association rules
- Web mining
- Market basket analysis
- User/customer profiling

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.

---

---

---

---

---

---

---

## Extensions: Sequential Patterns

Customer ID (CID)	Transaction ID (TID)	Itemset
1	1	$\{a, b, d\}$
1	3	$\{c, d\}$
1	6	$\{b, c, d\}$
2	2	$\{b\}$
2	4	$\{a, b, c\}$
3	5	$\{a, b\}$
3	7	$\{b, c, d\}$

Customer ID (CID)	Sequence
1	( $\{a, b, d\}, \{c, d\}, \{b, c, d\}$ )
2	( $\{b\}, \{a, b, c\}$ )
3	( $\{a, b\}, \{b, c, d\}$ )

Ramakrishnan and Gehrke, Database Management Systems, 3<sup>rd</sup> Edition.



## *Introduction to IR Systems: Supporting Boolean Text Search*

### Chapter 27, Part A

---

---

---

---

---

---

---



## *Information Retrieval*

- ❖ A research field traditionally separate from Databases
  - Goes back to IBM, Rand and Lockheed in the 50's
  - G. Salton at Cornell in the 60's
  - Lots of research since then
- ❖ Products traditionally separate
  - Originally, document management systems for libraries, government, law, etc.
  - Gained prominence in recent years due to web search

---

---

---

---

---

---

---



## *IR vs. DBMS*

- ❖ Seem like very different beasts:

IR	DBMS
Imprecise Semantics	Precise Semantics
Keyword search	SQL
Unstructured data format	Structured data
Read-Mostly: Add docs occasionally	Expect reasonable number of updates
Page through top $k$ results	Generate full answer

- ❖ Both support queries over large datasets, use indexing.
  - In practice, you currently have to choose between the two.

---

---

---

---

---

---

---

## IR's "Bag of Words" Model

- ❖ Typical IR data model:
  - Each document is just a bag (multiset) of words ("terms")
- ❖ Detail 1: "Stop Words"
  - Certain words are considered irrelevant and not placed in the bag
    - e.g., "the"
    - e.g., HTML tags like <H1>
- ❖ Detail 2: "Stemming" and other content analysis
  - Using English-specific rules, convert words to their basic form
    - e.g., "surfing", "surfed" --> "surf"

---

---

---

---

---

---

## Boolean Text Search

- ❖ Find all documents that match a Boolean containment expression:

"Windows"  
AND ("Glass" OR "Door")  
AND NOT "Microsoft"
- ❖ Note: Query terms are also filtered via stemming and stop words.
- ❖ When web search engines say "10,000 documents found", that's the Boolean search result size (subject to a common "max # returned' cutoff").

---

---

---

---

---

---

## Text "Indexes"

- ❖ When IR folks say "text index" ...
  - Usually mean more than what DB people mean
- ❖ In our terms, both "tables" and indexes
  - Really a logical schema (i.e., tables)
  - With a physical schema (i.e., indexes)
  - Usually not stored in a DBMS
    - Tables implemented as files in a file system
    - We'll talk more about this decision soon

---

---

---

---

---

---

## A Simple Relational Text Index

- ❖ Create and populate a table  
`InvertedFile(term string, docURL string)`
- ❖ Build a B+-tree or Hash index on `InvertedFile.term`
  - Alternative 3 (<Key, list of URLs> as entries in index) critical here for efficient storage!!
    - Fancy list compression possible, too
    - Note: URL instead of RID, the web is your "heap file"!
    - Can also *cache* pages and use RIDs
- ❖ This is often called an "inverted file" or "inverted index"
  - Maps from `words -> docs`
- ❖ Can now do single-word text search queries!

---

---

---

---

---

---

---

---

---

## An Inverted File

term	docURL
data	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>
database	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>
date	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>
day	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>
dbms	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>
decision	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>
demonstrate	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>
description	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>
design	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>
desire	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>
developer	<a href="http://www.microsoft.com">http://www.microsoft.com</a>
differ	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>
disability	<a href="http://www.microsoft.com">http://www.microsoft.com</a>
discussion	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>
division	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>
do	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>
document	<a href="http://www-inst.eecs.berkeley.edu/~cs186">http://www-inst.eecs.berkeley.edu/~cs186</a>

---

---

---

---

---

---

---

---

---

## Handling Boolean Logic

- ❖ How to do "term1" OR "term2"?
  - Union of two DocURL sets!
- ❖ How to do "term1" AND "term2"?
  - Intersection of two DocURL sets!
    - Can be done by sorting both lists alphabetically and merging the lists
- ❖ How to do "term1" AND NOT "term2"?
  - Set subtraction, also done via sorting
- ❖ How to do "term1" OR NOT "term2"
  - Union of "term1" and "NOT term2".
    - "Not term2" = all docs not containing term2. Large set!!
    - Usually not allowed!
- ❖ Refinement: What order to handle terms if you have many ANDs/NOTs?

---

---

---

---

---

---

---

---

---

## Boolean Search in SQL

"Windows" AND ("Glass" OR "Door")  
AND NOT "Microsoft"

```
❖ (SELECT docURL FROM InvertedFile
 WHERE word = "windows"
 INTERSECT
 SELECT docURL FROM InvertedFile
 WHERE word = "glass" OR word =
 "door")
 EXCEPT
 SELECT docURL FROM InvertedFile
 WHERE word="Microsoft"
 ORDER BY relevance()
```

---

---

---

---

---

---

---

## Boolean Search in SQL

- ❖ Really only one SQL query in Boolean Search IR:
  - Single-table selects, UNION, INTERSECT, EXCEPT
- ❖ relevance () is the “secret sauce” in the search engines:
  - Combos of statistics, linguistics, and graph theory tricks!
  - Unfortunately, not easy to compute this efficiently using typical DBMS implementation.

---

---

---

---

---

---

---

## Computing Relevance

- ❖ Relevance calculation involves how often search terms appear in doc, and how often they appear in collection:
  - More search terms found in doc → doc is more relevant
  - Greater importance attached to finding *rare* terms
- ❖ Doing this efficiently in current SQL engines is not easy:
  - “Relevance of a doc wrt a search term” is a function that is called once per doc the term appears in (docs found via inv. index):
    - For efficient fn computation, for each term, we can store the # times it appears in each doc, as well as the # docs it appears in.
    - Must also sort retrieved docs by their relevance value.
    - Also, think about Boolean operators (if the search has multiple terms) and how they affect the relevance computation!
  - An object-relational or object-oriented DBMS with good support for function calls is better, but you still have long execution path-lengths compared to optimized search engines.

---

---

---

---

---

---

---

## Fancier: Phrases and “Near”

- ❖ Suppose you want a phrase
  - E.g., “Happy Days”
- ❖ Different schema:
  - InvertedFile (`term` string, count int, position int, DocURL string)
  - Alternative 3 index on `term`
- ❖ Post-process the results
  - Find “Happy” AND “Days”
  - Keep results where positions are 1 off
    - Doing this well is like *join processing*
- ❖ Can do a similar thing for “term1” NEAR “term2”
  - Position  $< k$  off

---

---

---

---

---

---

---

## Updates and Text Search

- ❖ Text search engines are designed to be query-mostly:
  - Deletes and modifications are rare
  - Can postpone updates (nobody notices, no transactions!)
    - Updates done in batch (rebuild the index)
  - Can’t afford to go off-line for an update?
    - Create a 2nd index on a separate machine
    - Replace the 1st index with the 2nd!
  - So no concurrency control problems
  - Can compress to search-friendly, update-unfriendly format
- ❖ Main reason why text search engines and DBMSs are usually separate products.
  - Also, text-search engines tune that one SQL query to death!

---

---

---

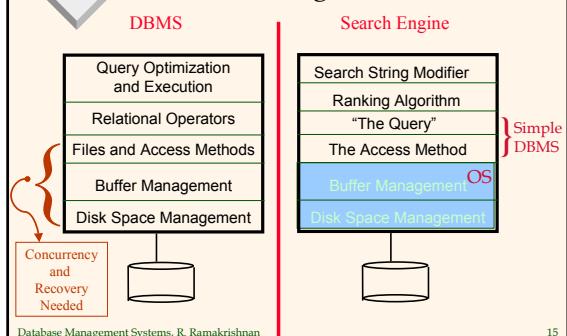
---

---

---

---

## DBMS vs. Search Engine Architecture



---

---

---

---

---

---

---

## IR vs. DBMS Revisited

### ❖ Semantic Guarantees

- DBMS guarantees transactional semantics
  - If inserting Xact commits, a later query *will see* the update
  - Handles multiple concurrent updates correctly
- IR systems do not do this; nobody notices!
  - Postpone insertions until convenient
  - No model of correct concurrency

### ❖ Data Modeling & Query Complexity

- DBMS supports any schema & queries
  - Requires you to define schema
  - Complex query language hard to learn
- IR supports only one schema & query
  - No schema design required (unstructured text)
  - Trivial to learn query language

---

---

---

---

---

---

---

## IR vs. DBMS, Contd.

### ❖ Performance goals

- DBMS supports general SELECT
  - Plus mix of INSERT, UPDATE, DELETE
  - General purpose engine must always perform “well”
- IR systems expect only one stylized SELECT
  - Plus delayed INSERT, unusual DELETE, no UPDATE
  - Special purpose, must run super-fast on “The Query”
  - Users rarely look at the full answer in Boolean Search

---

---

---

---

---

---

---

## Lots More in IR ...

- ❖ How to “rank” the output? I.e., how to compute relevance of each result item w.r.t. the query?
  - Doing this well / efficiently is hard!
- ❖ Other ways to help users paw through the output?
  - Document “clustering”, document visualization
- ❖ How to take advantage of hyperlinks?
  - Really cute tricks here!
- ❖ How to use compression for better I/O performance?
  - E.g., making RID lists smaller
  - Try to make things fit in RAM!
- ❖ How to deal with synonyms, misspelling, abbreviations?
- ❖ How to write a good web crawler?

---

---

---

---

---

---

---



## Computing Relevance, Similarity: The Vector Space Model

Chapter 27, Part B

Based on Larson and Hearst's slides at  
UC-Berkeley

<http://www.sims.berkeley.edu/courses/is202/f00/>

---

---

---

---

---

---

---



## Document Vectors

- ❖ Documents are represented as “bags of words”
- ❖ Represented as vectors when used computationally
  - A vector is like an array of floating point
  - Has direction and magnitude
  - Each vector holds a place for every term in the collection
  - Therefore, most vectors are sparse

---

---

---

---

---

---

---



## Document Vectors: One location for each word.

nova	galaxy	heat	h'wood	film	role	diet	fur
A	10	5	3				

“Nova” occurs 10 times in text A  
“Galaxy” occurs 5 times in text A  
“Heat” occurs 3 times in text A  
(Blank means 0 occurrences.)

---

---

---

---

---

---

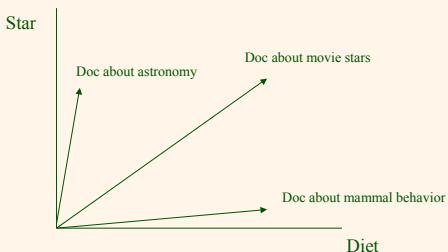
---

## Document Vectors

Document ids

	nova	galaxy	heat	h'wood	film	role	diet	fur
A	10	5	3					
B	5	10						
C				10	8	7		
D				9	10	5		
E							10	10
F							9	10
G	5	7			9			
H		6	10	2	8			
I				7	5		1	3

## We Can Plot the Vectors



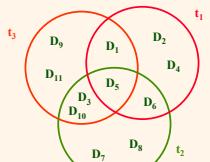
**Assumption:** Documents that are “close” in space are similar.

## Vector Space Model

- ❖ Documents are represented as *vectors* in term space
  - Terms are usually stems
  - Documents represented by binary vectors of terms
- ❖ Queries represented the same as documents
- ❖ A vector distance measure between the query and documents is used to rank retrieved documents
  - Query and Document similarity is based on length and direction of their vectors
  - Vector operations to capture boolean query

## Vector Space Documents and Queries

docs	<i>t1</i>	<i>t2</i>	<i>t3</i>	RSV=Q.Di
D1	1	0	1	4
D2	1	0	0	1
D3	0	1	1	5
D4	1	0	0	1
D5	1	1	1	6
D6	1	1	0	3
D7	0	1	0	2
D8	0	1	0	2
D9	0	0	1	3
D10	0	1	1	5
D11	1	0	1	3
Q	1	2	3	
	<i>q1</i>	<i>q2</i>	<i>q3</i>	



Boolean term combinations

Q is a query – also represented as a vector

## Assigning Weights to Terms

### ① Binary Weights

### ② Raw term frequency

### ③ tf x idf

- Recall the Zipf distribution
- Want to weight terms highly if they are
  - frequent in relevant documents ... BUT
  - infrequent in the collection as a whole

## Binary Weights

- ❖ Only the presence (1) or absence (0) of a term is included in the vector

docs	<i>t1</i>	<i>t2</i>	<i>t3</i>
D1	1	0	1
D2	1	0	0
D3	0	1	1
D4	1	0	0
D5	1	1	1
D6	1	1	0
D7	0	1	0
D8	0	1	0
D9	0	0	1
D10	0	1	1
D11	1	0	1

## Raw Term Weights

- The frequency of occurrence for the term in each document is included in the vector

docs	t1	t2	t3
D1	2	0	3
D2	1	0	0
D3	0	4	7
D4	3	0	0
D5	1	6	3
D6	3	5	0
D7	0	8	0
D8	0	10	0
D9	0	0	1
D10	0	3	5
D11	4	0	1

---

---

---

---

---

---

---

---

---

---

## TF x IDF Weights

- tf x idf measure:**
  - Term Frequency (tf)
  - Inverse Document Frequency (idf) -- a way to deal with the problems of the Zipf distribution
- Goal:** Assign a tf \* idf weight to each term in each document

---

---

---

---

---

---

---

---

---

---

## TF x IDF Calculation

$$w_{ik} = tf_{ik} * \log(N / n_k)$$

$T_k$  = term  $k$  in document  $D_i$

$tf_{ik}$  = frequency of term  $T_k$  in document  $D_i$

$idf_k$  = inverse document frequency of term  $T_k$  in  $C$

$N$  = total number of documents in the collection  $C$

$n_k$  = the number of documents in  $C$  that contain  $T_k$

$$idf_k = \log\left(\frac{N}{n_k}\right)$$

---

---

---

---

---

---

---

---

---

---

## Inverse Document Frequency

- IDF provides high values for rare words and low values for common words

$$\log\left(\frac{10000}{10000}\right) = 0$$

For a collection of 10000 documents

$$\log\left(\frac{10000}{5000}\right) = 0.301$$

$$\log\left(\frac{10000}{20}\right) = 2.698$$

$$\log\left(\frac{10000}{1}\right) = 4$$

---

---

---

---

---

---

---

## TF $\times$ IDF Normalization

- Normalize the term weights (so longer documents are not unfairly given more weight)
  - The longer the document, the more likely it is for a given term to appear in it, and the more often a given term is likely to appear in it. So, we want to reduce the importance attached to a term appearing in a document based on the length of the document.

$$w_{ik} = \frac{tf_{ik} \log(N/n_k)}{\sqrt{\sum_{k=1}^t (tf_{ik})^2 [\log(N/n_k)]^2}}$$

---

---

---

---

---

---

---

## Pair-wise Document Similarity

	nova	galaxy	heat	h'wood	film	role	diet	fur
A	1	3	1					
B	5	2						
C			2	1	5			
D				4	1			

How to compute document similarity?

---

---

---

---

---

---

---



## Pair-wise Document Similarity

$$D_1 = w_{11}, w_{12}, \dots, w_{1t}$$

$$\text{sim}(A, B) = (1 * 5) + (2 * 3) = 11$$

$$D_2 = w_{21}, w_{22}, \dots, w_{2t}$$

$$\text{sim}(A, C) = 0$$

$$\text{sim}(D_1, D_2) = \sum_{i=1}^t w_{1i} * w_{2i}$$

$$\text{sim}(A, D) = 0$$

$$\text{sim}(B, C) = 0$$

$$\text{sim}(B, D) = 0$$

$$\text{sim}(C, D) = (2 * 4) + (1 * 1) = 9$$

	nova	galaxy	heat	h'wood	film	role	diet	fur
A	1	3	1					
B	5	2						
C			2	1	5			
D				4	1			

---

---

---

---

---

---

---

---



## Pair-wise Document Similarity (cosine normalization)

$$D_1 = w_{11}, w_{12}, \dots, w_{1t}$$

$$D_2 = w_{21}, w_{22}, \dots, w_{2t}$$

$$\text{sim}(D_1, D_2) = \sum_{i=1}^t w_{1i} * w_{2i} \text{ unnormalized}$$

$$\text{sim}(D_1, D_2) = \frac{\sum_{i=1}^t w_{1i} * w_{2i}}{\sqrt{\sum_{i=1}^t (w_{1i})^2 * \sum_{i=1}^t (w_{2i})^2}} \text{ cosine normalized}$$

---

---

---

---

---

---

---

---



## Vector Space "Relevance" Measure

$$D_i = w_{d_{i1}}, w_{d_{i2}}, \dots, w_{d_{it}}$$

$$Q = w_{q1}, w_{q2}, \dots, w_{qt} \quad w = 0 \text{ if a term is absent}$$

$$\text{if term weights normalized: } \text{sim}(Q, D_i) = \sum_{j=1}^t w_{qj} * w_{d_{ij}}$$

otherwise normalize in the similarity comparison :

$$\text{sim}(Q, D_i) = \frac{\sum_{j=1}^t w_{qj} * w_{d_{ij}}}{\sqrt{\sum_{j=1}^t (w_{qj})^2 * \sum_{j=1}^t (w_{d_{ij}})^2}}$$

---

---

---

---

---

---

---

---

## Computing Relevance Scores

Say we have query vector  $Q = (0.4, 0.8)$

Also, document  $D_2 = (0.2, 0.7)$

What does their similarity comparison yield?

$$\begin{aligned} sim(Q, D_2) &= \frac{(0.4 \cdot 0.2) + (0.8 \cdot 0.7)}{\sqrt{[(0.4)^2 + (0.8)^2] \cdot [(0.2)^2 + (0.7)^2]}} \\ &= \frac{0.64}{\sqrt{0.42}} = 0.98 \end{aligned}$$

---



---



---



---



---



---



---

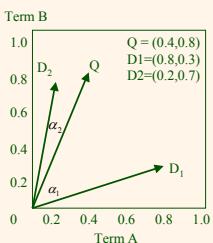


---



---

## Vector Space with Term Weights and Cosine Matching



$$\begin{aligned} D_i &= (d_{i1}, w_{di1}; d_{i2}, w_{di2}; \dots; d_{it}, w_{dit}) \\ Q &= (q_{i1}, w_{qi1}; q_{i2}, w_{qi2}; \dots; q_{it}, w_{qit}) \\ sim(Q, D_i) &= \frac{\sum_{j=1}^t w_{qj} w_{dj}}{\sqrt{\sum_{j=1}^t (w_{qj})^2 \sum_{j=1}^t (w_{dj})^2}} \\ sim(Q, D2) &= \frac{(0.4 \cdot 0.2) + (0.8 \cdot 0.7)}{\sqrt{[(0.4)^2 + (0.8)^2] \cdot [(0.2)^2 + (0.7)^2]}} \\ &= \frac{0.64}{\sqrt{0.42}} = 0.98 \\ sim(Q, D1) &= \frac{.56}{\sqrt{0.58}} = 0.74 \end{aligned}$$

---



---



---



---



---



---



---



---



---



---

## Text Clustering

- ❖ Finds overall similarities among groups of documents
- ❖ Finds overall similarities among groups of tokens
- ❖ Picks out some themes, ignores others

---



---



---



---



---



---



---



---



---



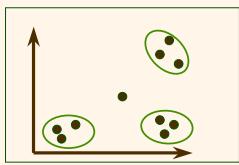
---

## *Text Clustering*

### Clustering is

"The **art** of finding groups in data."  
-- Kaufmann and Rousseeu

Term 1



Term  
2

---

---

---

---

---

---

---

## *Problems with Vector Space*

- ❖ There is no real theoretical basis for the assumption of a term space
  - It is more for visualization than having any real basis
  - Most similarity measures work about the same
- ❖ Terms are not really orthogonal dimensions
  - Terms are not independent of all other terms; remember our discussion of correlated terms in text

---

---

---

---

---

---

---

## *Probabilistic Models*

- ❖ Rigorous formal model attempts to predict the probability that a given document will be relevant to a given query
- ❖ Ranks retrieved documents according to this probability of relevance (**Probability Ranking Principle**)
- ❖ Relies on accurate estimates of probabilities

---

---

---

---

---

---

---

## Probability Ranking Principle

- ❖ If a reference retrieval system's response to each request is a ranking of the documents in the collections in the order of decreasing probability of usefulness to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data has been made available to the system for this purpose, then the overall effectiveness of the system to its users will be the best that is obtainable on the basis of that data.

Stephen E. Robertson, *J. Documentation* 1977

---

---

---

---

---

---

---

## Iterative Query Refinement

---

---

---

---

---

---

---

## Query Modification

- ❖ **Problem:** How can we reformulate the query to help a user who is trying several searches to get at the same information?
  - Thesaurus expansion:
    - Suggest terms similar to query terms
  - Relevance feedback:
    - Suggest terms (and documents) similar to retrieved documents that have been judged to be relevant

---

---

---

---

---

---

---

## Relevance Feedback

### ❖ Main Idea:

- Modify existing query based on relevance judgements
  - Extract terms from relevant documents and add them to the query
  - AND/OR re-weight the terms already in the query

### ❖ There are many variations:

- Usually positive weights for terms from relevant docs
- Sometimes negative weights for terms from non-relevant docs

### ❖ Users, or the system, guide this process by

---

---

---

---

---

---

---

## Rocchio Method

### ❖ Rocchio automatically

- Re-weights terms
- Adds in new terms (from relevant docs)
  - have to be careful when using negative terms
- Rocchio is *not* a machine learning algorithm

---

---

---

---

---

---

---

## Rocchio Method

$$Q_1 = \alpha Q_0 + \frac{\beta}{n_1} \sum_{i=1}^{n_1} R_i - \frac{\gamma}{n_2} \sum_{i=1}^{n_2} S_i$$

where

$Q_0$  = the vector for the initial query

$R_i$  = the vector for the relevant document  $i$

$S_i$  = the vector for the non-relevant document  $i$

$n_1$  = the number of relevant documents chosen

$n_2$  = the number of non-relevant documents chosen

$\alpha, \beta$  and  $\gamma$  tune the importance of relevant and nonrelevant terms  
(in some studies best to set  $\beta$  to 0.75 and  $\gamma$  to 0.25)

---

---

---

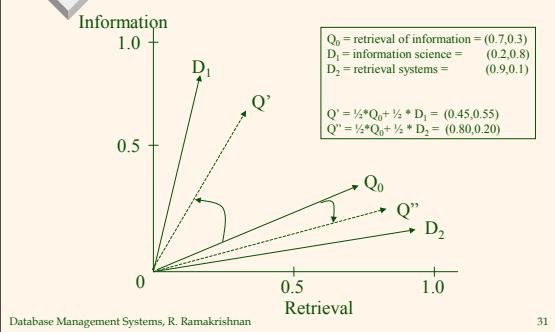
---

---

---

---

## Rocchio/Vector Illustration



## Alternative Notions of Relevance Feedback

- ❖ Find people whose taste is “similar” to yours.
  - Will you like what they like?
- ❖ Follow a user’s actions in the background.
  - Can this be used to predict what the user will want to see next?
- ❖ Track what lots of people are doing.
  - Does this implicitly indicate what they think is good and not good?

## Collaborative Filtering (Social Filtering)

- ❖ If Pam liked the paper, I'll like the paper
- ❖ If you liked Star Wars, you'll like Independence Day
- ❖ Rating based on ratings of similar people
  - Ignores text, so also works on sound, pictures etc.
  - But: Initial users can bias ratings of future users

	Sally	Bob	Chris	Lynn	Karen
Star Wars	7	7	3	4	7
Jurassic Park	6	4	7	4	4
Terminator II	3	4	7	6	3
Independence Day	7	7	2	2	?

## Ringo Collaborative Filtering

- ❖ Users rate items from like to dislike
  - 7 = like; 4 = ambivalent; 1 = dislike
  - A normal distribution; the extremes are what matter
- ❖ Nearest Neighbors Strategy: Find similar users and predicted (weighted) average of user ratings
- ❖ Pearson Algorithm: Weight by degree of correlation between user U and user J
  - 1 means similar, 0 means no correlation, -1 dissimilar
  - Works better to compare against the ambivalent rating (4), rather than the individual's average score

$$r_{UJ} = \frac{\sum(U - \bar{U})(J - \bar{J})}{\sqrt{\sum(U - \bar{U})^2} \cdot \sqrt{\sum(J - \bar{J})^2}}$$



## Web Search Engines

*Chapter 27, Part C*

*Based on Larson and Hearst's slides at  
UC-Berkeley*

<http://www.sims.berkeley.edu/courses/is202/f00/>

---

---

---

---

---

---

---



## Search Engine Characteristics

- ❖ Unedited – anyone can enter content
  - Quality issues; Spam
- ❖ Varied information types
  - Phone book, brochures, catalogs, dissertations, news reports, weather, all in one place!
- ❖ Different kinds of users
  - Lexis-Nexis: Paying, professional searchers
  - Online catalogs: Scholars searching scholarly literature
  - Web: Every type of person with every type of goal
- ❖ Scale
  - Hundreds of millions of searches/day; billions of docs

---

---

---

---

---

---

---



## Web Search Queries

- ❖ Web search queries are **short**:
  - ~2.4 words on average (Aug 2000)
  - Has increased, was 1.7 (~1997)
- ❖ User Expectations:
  - Many say “The first item shown should be what I want to see!”
  - This works if the user has the most popular/common notion in mind, not otherwise.

---

---

---

---

---

---

---

## Directories vs. Search Engines

### ❖ Directories

- Hand-selected sites
- Search over the contents of the *descriptions* of the pages
- Organized in advance into categories

### ❖ Search Engines

- All pages in all sites
- Search over the contents of the *pages themselves*
- Organized in response to a query by relevance rankings or other scores

---

---

---

---

---

---

---

## What about Ranking?

### ❖ Lots of variation here

- Often messy; details proprietary and fluctuating

### ❖ Combining subsets of:

- IR-style relevance: Based on term frequencies, proximities, position (e.g., in title), font, etc.
- Popularity information
- Link analysis information

### ❖ Most use a variant of vector space ranking to combine these. Here's how it might work:

- Make a vector of weights for each feature
- Multiply this by the counts for each feature

---

---

---

---

---

---

---

## Relevance: Going Beyond IR

### ❖ Page “popularity” (e.g., DirectHit)

- Frequently visited pages (in general)
- Frequently visited pages as a result of a query

### ❖ Link “co-citation” (e.g., Google)

- Which sites are linked to by other sites?
- Draws upon sociology research on bibliographic citations to identify “authoritative sources”
- Discussed further in Google case study

---

---

---

---

---

---

---

## *Web Search Architecture*

---

---

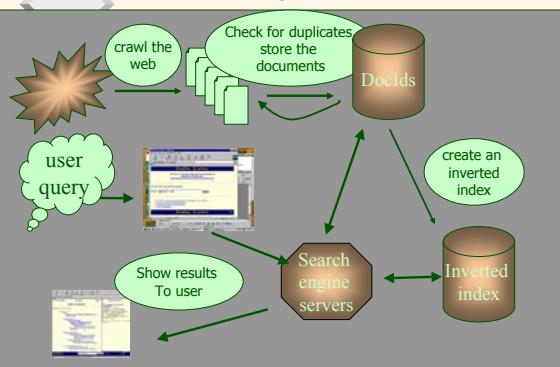
---

---

---

---

*Standard Web Search Engine Architecture*



---

---

---

---

---

---

## *Inverted Indexes the IR Way*

---

---

---

---

---

---

## How Inverted Files Are Created

- Periodically rebuilt, static otherwise.
- Documents are parsed to extract tokens. These are saved with the Document ID.

Doc 1

Now is the time  
for all good men  
to come to the aid  
of their country

Doc 2

It was a dark and  
stormy night in  
the country  
manor. The time  
was past midnight

Term	Doc #
now	1
is	1
the	1
time	1
for	1
all	1
good	1
men	1
to	1
come	1
to	1
the	1
aid	1
of	1
their	1
country	1
it	2
was	2
a	2
dark	2
and	2
stormy	2
night	2
in	2
the	2
country	2
manor	2
the	2
time	2
was	2
past	2
midnight	2

Database Management Systems, R. Ramakrishnan

10

## How Inverted Files are Created

- After all documents have been parsed the inverted file is sorted alphabetically.

Term	Doc #	Term	Doc #
now	1	a	2
is	1	aid	1
the	1	all	1
time	1	and	2
for	1	come	1
all	1	country	1
good	1	country	2
men	1	dark	2
to	1	for	1
come	1	good	1
to	1	in	2
the	1	is	1
aid	1	it	2
of	1	manor	2
their	1	men	1
country	1	midnight	2
it	2	night	2
was	2	now	1
a	2	of	1
dark	2	past	2
and	2	stormy	2
stormy	2	the	1
night	2	the	2
in	2	the	2
the	2	their	1
country	2	time	1
manor	2	time	2
the	2	to	1
time	2	to	1
was	2	was	1
past	2	was	2
midnight	2		

Database Management Systems, R. Ramakrishnan

11

## How Inverted Files are Created

- Multiple term entries for a single document are merged.
- Within-document term frequency information is compiled.

Term	Doc #	Term	Doc #	Freq
a	2	a	2	1
all	1	aid	1	1
all	2	all	1	1
come	1	and	2	1
country	1	come	1	1
country	2	country	1	1
for	1	country	2	1
good	1	dark	2	1
in	2	dark	1	1
is	1	for	1	1
it	2	good	1	1
manor	2	in	2	1
men	1	is	1	1
midnight	2	it	2	1
night	2	manor	2	1
now	1	men	1	1
of	1	midnight	2	1
past	2	night	2	1
stormy	2	now	1	1
the	1	of	1	1
the	1	past	2	1
the	2	stormy	2	1
the	2	the	1	2
their	1	the	2	2
time	1	time	1	1
time	2	time	1	1
to	1	time	2	1
to	1	to	1	1
was	2	was	2	1
was	2			

Database Management Systems, R. Ramakrishnan

12

## How Inverted Files are Created

- ❖ Finally, the file can be split into
  - A Dictionary or Lexicon file
  - and
  - A Postings file



## Inverted indexes

- ❖ Permit fast search for individual terms
- ❖ For each term, you get a list consisting of:
  - document ID
  - frequency of term in doc (optional)
  - position of term in doc (optional)
- ❖ These lists can be used to solve Boolean queries:
  - country -> d1, d2
  - manor -> d2
  - country AND manor -> d2
- ❖ Also used for statistical ranking algorithms



## Inverted Indexes for Web Search Engines

- ❖ Inverted indexes are still used, even though the web is so huge.
- ❖ Some systems partition the indexes across different machines. Each machine handles different parts of the data.
- ❖ Other systems duplicate the data across many machines; queries are distributed among the machines.
- ❖ Most do a combination of these.

---

---

---

---

---

---

---

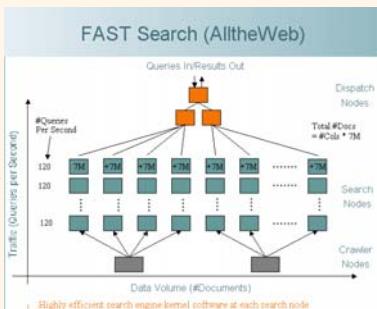


In this example, the data for the pages is partitioned across machines. Additionally, each partition is allocated multiple machines to handle the queries.

Each row can handle 120 queries per second

Each column can handle 7M pages

To handle more queries, add another row.  
From description of the FAST search engine, by Knut Rissvik  
[http://www.infonortics.com/searchengines/sh00/rissvik\\_files/frame.htm](http://www.infonortics.com/searchengines/sh00/rissvik_files/frame.htm)



---

---

---

---

---

---

---



## Cascading Allocation of CPUs

- ❖ A variation on this that produces a cost-savings:
  - Put high-quality/common pages on many machines
  - Put lower quality/less common pages on fewer machines
  - Query goes to high quality machines first
  - If no hits found there, go to other machines

---

---

---

---

---

---

---



## *Web Crawling*

---

---

---

---

---

---



## *Web Crawlers*

- ❖ How do the web search engines get all of the items they index?
- ❖ Main idea:
  - Start with known sites
  - Record information for these sites
  - Follow the links from each site
  - Record information found at new sites
  - Repeat

---

---

---

---

---

---



## *Web Crawling Algorithm*

- ❖ More precisely:
  - Put a set of known sites on a queue
  - Repeat the following until the queue is empty:
    - Take the first page off of the queue
    - If this page has not yet been processed:
      - Record the information found on this page
        - Positions of words, links going out, etc
      - Add each link on the current page to the queue
      - Record that this page has been processed
  - ❖ Rule-of-thumb: 1 doc per minute per crawling server

---

---

---

---

---

---

## Web Crawling Issues

### ❖ Keep out signs

- A file called **norobots.txt** lists “off-limits” directories
- Freshness: Figure out which pages change often, and recrawl these often.

### ❖ Duplicates, virtual hosts, etc.

- Convert page contents with a hash function
- Compare new pages to the hash table

### ❖ Lots of problems

- Server unavailable; incorrect html; missing links; attempts to “fool” search engine by giving crawler a version of the page with lots of spurious terms added ...

### ❖ Web crawling is **difficult** to do robustly!

---

---

---

---

---

---

---

---



## Google: A Case Study

---

---

---

---

---

---

---

---



## Google's Indexing

### ❖ The *Indexer* converts each doc into a collection of “hit lists” and puts these into “barrels”, sorted by docID. It also creates a database of “links”.

- **Hit:** <wordID, position in doc, font info, hit type>
- **Hit type:** Plain or fancy.
- **Fancy hit:** Occurs in URL, title, anchor text, metatag.
- Optimized representation of hits (2 bytes each).

### ❖ Sorter sorts each barrel by wordID to create the **inverted index**. It also creates a lexicon file.

- **Lexicon:** <wordID, offset into inverted index>
- Lexicon is mostly cached in-memory

---

---

---

---

---

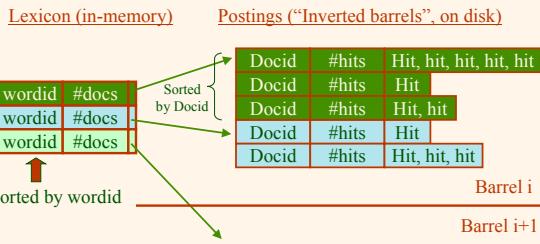
---

---

---

## Google's Inverted Index

Each “barrel” contains postings for a range of wordids.

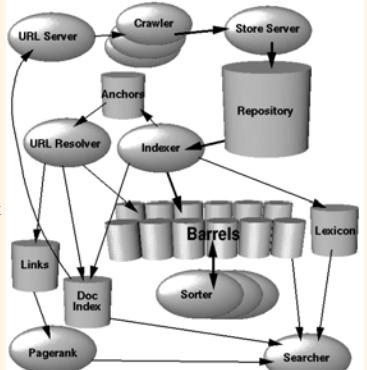


Database Management Systems, R. Ramakrishnan

25

## Google

- Sorted barrels = inverted index
- Pagerank computed from link structure; combined with IR rank
- IR rank depends on TF, type of “hit”, hit proximity, etc.
- Billion documents
- Hundred million queries a day
- AND queries



Database Management Systems, R. Ramakrishnan

26

## Link Analysis for Ranking Pages

- ❖ **Assumption:** If the pages pointing to this page are good, then this is also a good page.
  - References: Kleinberg 98, Page et al. 98
- ❖ Draws upon earlier research in sociology and bibliometrics.
  - Kleinberg's model includes “authorities” (highly referenced pages) and “hubs” (pages containing good reference lists).
  - Google model is a version with no hubs, and is closely related to work on influence weights by Pinski-Narin (1976).

Database Management Systems, R. Ramakrishnan

27

## Link Analysis for Ranking Pages

### ❖ Why does this work?

- The official Toyota site will be linked to by lots of other official (or high-quality) sites
- The best Toyota fan-club site probably also has many links pointing to it
- Less high-quality sites do not have as many high-quality sites linking to them

---

---

---

---

---

---

---

## PageRank

❖ Let A<sub>1</sub>, A<sub>2</sub>, ..., A<sub>n</sub> be the pages that point to page A. Let C(P) be the # links out of page P. The PageRank (PR) of page A is defined as:

$$\text{PR}(A) = (1-d) + d \left( \frac{\text{PR}(A_1)}{C(A_1)} + \dots + \frac{\text{PR}(A_n)}{C(A_n)} \right)$$

- ❖ PageRank is principal eigenvector of the link matrix of the web.
- ❖ Can be computed as the fixpoint of the above equation.

---

---

---

---

---

---

---

## PageRank: User Model

- ❖ PageRanks form a probability distribution over web pages: sum of all pages' ranks is one.
- ❖ **User model:** "Random surfer" selects a page, keeps clicking links (never "back"), until "bored": then randomly selects another page and continues.
  - PageRank(A) is the probability that such a user visits A
  - d is the probability of getting bored at a page
- ❖ Google computes relevance of a page for a given search by first computing an IR relevance and then modifying that by taking into account PageRank for the top pages.

---

---

---

---

---

---

---

## Web Search Statistics

---

---

---

---

---

---

---

---

---

---

### Searches per Day

Service	Searches Per Day	As Of/Notes
AltaVista	50 million	9/00 (as reported to me by AltaVista, for its site and queries through partners)
Inktomi	47 million	4/00 (site reflects queries from Yahoo, which will be handled by Google from July 2000)
Google	40 million	8/00 (14 million of these are at Google.com, 15 million are probably generated through Google's partnership with Yahoo, and the remainder come through Google partner sites, such as Netscape Search)
GoTo	5 million	4/00 (as reported by GoTo to a reader, who forwarded the information to me. Includes queries through affiliates and partners)
Ask Jeeves	4 million	3/00
Voda	1.5 million	1/00 (as reported to me by Voda, for its entire network of sites)

---

---

---

---

---

---

---

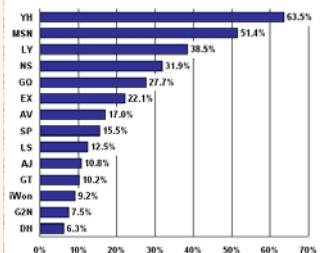
---

---

---

### Web Search Engine Visits

Below is a look at the latest Media Metrix's ratings. They show audience reach, which is the percentage of web surfers estimated to have visited each search engine during the month. Because a web surfer may visit more than one service, the combined totals exceed percent.



KEY: YH=Yahoo, MSN=MSN, LY=Lycos, NS=Netscape, GO=Go (Infoseek), EX=Excite, AV=AltaVista, SP=Snap, LS=LookSmart, AJ=Ask Jeeves, GT=GoTo, Wn=Web, GZ=Go2Net, DH=Direct Hit. Also use this key for charts below. See the [Major Search Engines page](#) for links to these services.

---

---

---

---

---

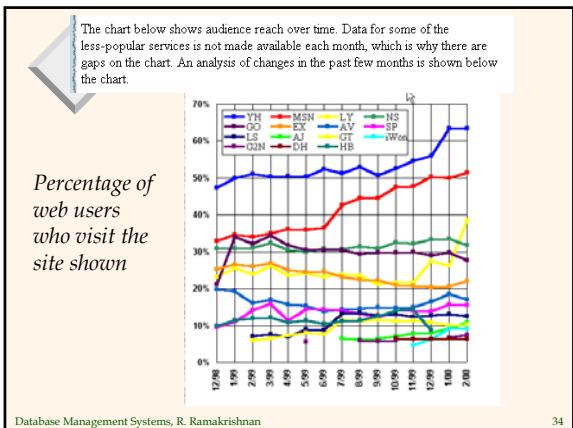
---

---

---

---

---



**Obscure Terms**

The first test checked on how well each search engine did in finding four obscure terms. By obscure, I mean that these were words unusual enough that no search engines found more than 100 matches. A separate page listed at the end of this article shows exactly what terms were used and the scoring methodology. The chart below summarizes the test. Search engines are listed in order of performance, with the best at the top of the list.

Search Engine	Reported Size	Expected Score	Actual Score	Rank
Google	560	1.0	1.0	1
FAST	340	2.0	1.8	2
Northern Light	265	3.0	2.3	3
HotBot	110	4.0	2.3	3
iWen	110	4.0	2.3	3
AltaVista	350	2.0	2.5	4
Yahoo-Google	560	1.0	3.0	5
Excite	250	3.0	3.0	5
Yahoo-Infoseek	110	4.0	4.3	6

The first column shows you how many millions of pages each search engine claims to have indexed. The "Expected Score" column is based on this. You

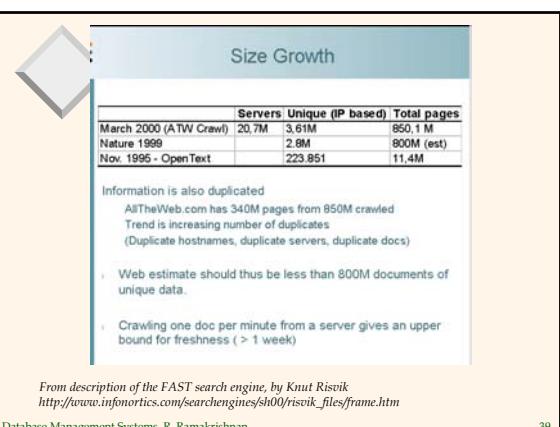
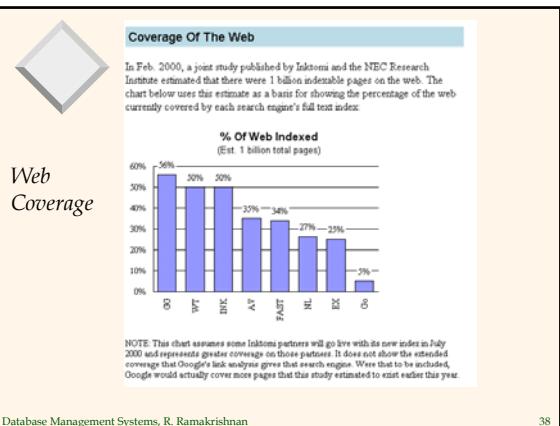
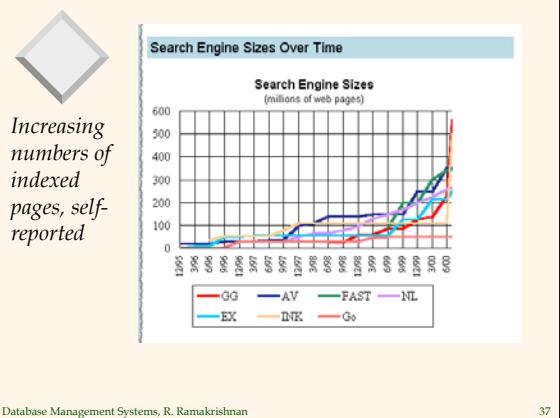
Database Management Systems, R. Ramakrishnan 35

**Does size matter? You can't access many hits anyhow.**

For the curious, here's a list of the total number of results you can possibly recover from the search engines tested.

Search Engine	Max. Results
FAST	4,010
AltaVista	1,000
Excite	1,000
Google	1,000
HotBot	1,000
iWen	1,000
Yahoo *Web Pages*	199
Northern Light	couldn't determine

Database Management Systems, R. Ramakrishnan 36





## Directory sizes

### Directory Sizes

Directories are usually human-complied guides to the web, where sites are organized by category. The chart below compares the size of directories at various services, along with other key data. A ? symbol indicates where information is not known or hasn't been released.

Service	Type	Editors	Cats	Links...	As Of
<a href="#">Open Directory</a>	D	25,000	304,000	2 million	9/00
<a href="#">LookSmart</a>	D	200	200,000	2 million	8/00
<a href="#">Yahoo</a>	D	100+	?	1.5 to 1.8 million	8/00
<a href="#">NDS (Snoopy)</a>	D	30-50	70,000	1 million+	8/00
<a href="#">On DirectoRy</a>	SE	10,000	50,000	500,000+	1/00
<a href="#">AskJeeves</a>	AS	30	n/a	7 million	11/98
<a href="#">Altavista</a>	SE	See LookSmart			
<a href="#">Excite</a>	SE	See LookSmart			
<a href="#">HotBot</a>	SE	See Open Directory			
<a href="#">Lycos</a>	D	See Open Directory			
<a href="#">Metacrawler</a>	SE	See LookSmart			
<a href="#">WebCrawler</a>	SE	See Open Directory			



## *Introduction to Semistructured Data and XML*

Chapter 27, Part D  
Based on slides by Dan Suciu  
University of Washington

---

---

---

---

---

---



## *How the Web is Today*

- ❖ HTML documents
  - often generated by applications
  - consumed by humans only
  - easy access: across platforms, across organizations
- ❖ No application interoperability:
  - HTML not understood by applications
    - screen scraping brittle
  - Database technology: client-server
    - still vendor specific

---

---

---

---

---

---



## *New Universal Data Exchange Format: XML*

- A recommendation from the W3C
- ❖ XML = data
  - ❖ XML generated by applications
  - ❖ XML consumed by applications
  - ❖ Easy access: across platforms, organizations

---

---

---

---

---

---

## Paradigm Shift on the Web

- ❖ From documents (HTML) to data (XML)
- ❖ From information retrieval to data management
- ❖ For databases, also a paradigm shift:
  - from relational model to semistructured data
  - from data processing to data/query translation
  - from storage to transport

---

---

---

---

---

---

## Semistructured Data

### Origins:

- ❖ Integration of heterogeneous sources
- ❖ Data sources with non-rigid structure
  - Biological data
  - Web data

---

---

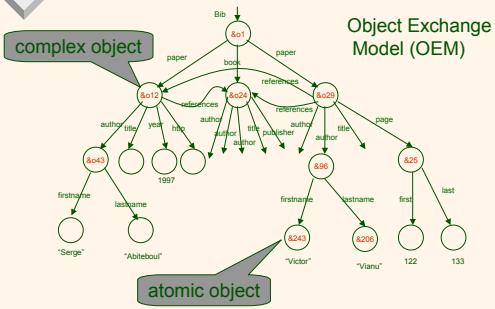
---

---

---

---

## The Semistructured Data Model



---

---

---

---

---

---

## *Syntax for Semistructured Data*

```
Bib: &eol { paper: &eol2 { ... },
 book: &eol2 { ... },
 paper: &eol2
 { author: &eol2 "Abiteboul",
 author: &eol2 { firstname: &eol3 "Victor",
 lastname: &eol26 "Vianu" },
 title: &eol3 "Regular path queries with constraints",
 references: &eol2,
 references: &eol24,
 pages: &eol25 [first: &eol64 122, last: &eol92 133]
 }
}
```

Observe: Nested tuples, set-values, oids!

Database Management Systems, R. Ramakrishnan

7

## *Syntax for Semistructured Data*

May omit oids:

```
{ paper: { author: "Abiteboul",
 author: { firstname: "Victor",
 lastname: "Vianu"},
 title: "Regular path queries ...",
 page: { first: 122, last: 133 }
 }
 }
```

Database Management Systems, R. Ramakrishnan

8

## *Characteristics of Semistructured Data*

- ❖ Missing or additional attributes
  - ❖ Multiple attributes
  - ❖ Different types in different objects
  - ❖ Heterogeneous collections

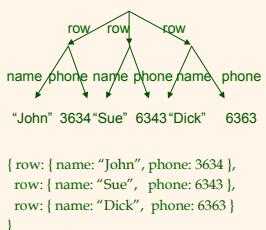
Self-describing, irregular data, no a priori structure

Database Management Systems, R. Ramakrishnan

9

## Comparison with Relational Data

name	phone
John	3634
Sue	6343
Dick	6363



Database Management Systems, R. Ramakrishnan

10

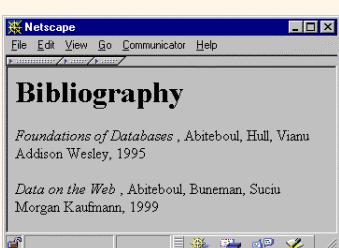
## XML

- ❖ A W3C standard to complement HTML
- ❖ Origins: Structured text SGML
  - Large-scale electronic publishing
  - Data exchange on the web
- ❖ Motivation:
  - HTML describes presentation
  - XML describes content
- ❖  $\text{HTML 4.0} \in \text{XML} \subset \text{SGML}$   
<http://www.w3.org/TR/2000/REC-xml-20001006> (version 2, 10/2000)

Database Management Systems, R. Ramakrishnan

11

## From HTML to XML



HTML describes the presentation

Database Management Systems, R. Ramakrishnan

12

## HTML

```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
 Abiteboul, Hull, Vianu

 Addison Wesley, 1995
<p> <i> Data on the Web </i>
 Abiteboul, Buneman, Suciu

 Morgan Kaufmann, 1999
```

Database Management Systems, R. Ramakrishnan

13

---

---

---

---

---

---

---

---

## XML

```
<bibliography>
 <book> <title> Foundations... </title>
 <author> Abiteboul </author>
 <author> Hull </author>
 <author> Vianu </author>
 <publisher> Addison Wesley </publisher>
 <year> 1995 </year>
 </book>
 ...
</bibliography>
```

**XML describes the content**

Database Management Systems, R. Ramakrishnan

14

---

---

---

---

---

---

---

---

## Why are we DB'ers interested?

- ❖ It's data, stupid. That's us.
- ❖ Proof by Google:
  - database+XML - 1,940,000 pages.
- ❖ Database issues:
  - How are we going to model XML? ([graphs](#)).
  - How are we going to query XML? ([XQuery](#))
  - How are we going to store XML (in a relational database? [object-oriented](#)? [native](#)?)
  - How are we going to process XML efficiently?  
(many interesting research questions!)

Database Management Systems, R. Ramakrishnan

15

---

---

---

---

---

---

---

---

## Document Type Descriptors

- ❖ Sort of like a schema but not really.

```
<!ELEMENT Book (title, author*)>
<!ELEMENT title #PCDATA>
<!ELEMENT author (name, address, age?)>
<!ATTLIST Book id ID #REQUIRED>
<!ATTLIST Book pub IDREF #IMPLIED>
```

- ❖ Inherited from SGML DTD standard
- ❖ BNF grammar establishing constraints on element structure and content
- ❖ Definitions of entities

---

---

---

---

---

---

---

---

## Shortcomings of DTDs

Useful for documents, but not so good for data:

- ❖ Element name and type are associated globally
- ❖ No support for structural re-use
  - Object-oriented-like structures aren't supported
- ❖ No support for data types
  - Can't do data validation
- ❖ Can have a *single* key item (ID), but:
  - No support for multi-attribute keys
  - No support for foreign keys (references to other keys)
  - No constraints on IDREFs (reference *only* a Section)

---

---

---

---

---

---

---

---

## XML Schema

- ❖ In XML format
- ❖ Element names and types associated locally
- ❖ Includes primitive data types (integers, strings, dates, etc.)
- ❖ Supports value-based constraints (integers > 100)
- ❖ User-definable structured types
- ❖ Inheritance (extension or restriction)
- ❖ Foreign keys
- ❖ Element-type reference constraints

---

---

---

---

---

---

---

---

## Sample XML Schema

```
<schema version="1.0" xmlns="http://www.w3.org/1999/XMLSchema">
<element name="author" type="string" />
<element name="date" type="date" />
<element name="abstract">
</type>
</type>
<element>
<element name="paper">
<type>
<attribute name="keywords" type="string"/>
<element ref="author" minOccurs="0" maxOccurs="*" />
<element ref="date" />
<element ref="abstract" minOccurs="0" maxOccurs="1" />
<element ref="body" />
</type>
</element>
</schema>
```

Database Management Systems, R. Ramakrishnan

19

---

---

---

---

---

---

---

---

## Important XML Standards

- ❖ XSL/XSLT: presentation and transformation standards
- ❖ RDF: resource description framework (meta-info such as ratings, categorizations, etc.)
- ❖ Xpath/Xpointer/Xlink: standard for linking to documents and elements within
- ❖ Namespaces: for resolving name clashes
- ❖ DOM: Document Object Model for manipulating XML documents
- ❖ SAX: Simple API for XML parsing
- ❖ XQuery: query language

Database Management Systems, R. Ramakrishnan

20

---

---

---

---

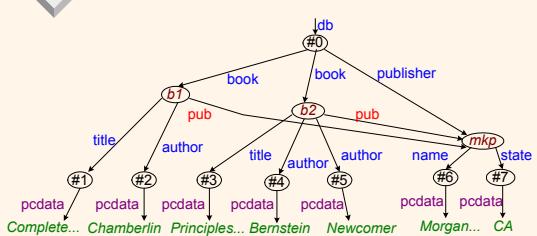
---

---

---

---

## XML Data Model (Graph)



### Issues:

- Distinguish between attributes and sub-elements?
- Should we conserve order?

Database Management Systems, R. Ramakrishnan

21

---

---

---

---

---

---

---

---

## XML Terminology

- ❖ **Tags:** book, title, author, ...
  - start tag: <book>, end tag: </book>
- ❖ **Elements:** <book>...<book>,<author>...</author>
  - elements can be nested
  - empty element: <red></red> (Can be abbrv. <red/>)
- ❖ **XML document:** Has a single root element
- ❖ **Well-formed XML document:** Has matching tags
- ❖ **Valid XML document:** conforms to a schema

---

---

---

---

---

---

---

## More XML: Attributes

```
<book price = "55" currency = "USD">
 <title> Foundations of Databases </title>
 <author> Abiteboul </author>
 ...
 <year> 1995 </year>
</book>
```

Attributes are alternative ways to represent data

---

---

---

---

---

---

---

## More XML: Oids and References

```
<person id="o555"> <name> Jane </name> </person>

<person id="o456"> <name> Mary </name>
 <children idref="o123 o555" />
</person>

<person id="o123" mother="o456"><name>John</name>
</person>
```

Oids and references in XML are just syntax

---

---

---

---

---

---

---

## XML-Query Data Model

- ❖ Describes XML data as a tree
- ❖ Node ::= DocNode |  
ElemNode |  
ValueNode |  
AttrNode |  
NSNode |  
PINode |  
CommentNode |  
InfoItemNode |  
RefNode

<http://www.w3.org/TR/query-datalogmodel/2001>

Database Management Systems, R. Ramakrishnan

25

---

---

---

---

---

---

---

---

---

## XML-Query Data Model

### Element node (simplified definition):

- ❖ elemNode : (QNameValue,  
                  {AttrNode},  
                  [ ElemNode | ValueNode])  
    → ElemNode
- ❖ QNameValue = means “a tag name”

Reads: “Give me a tag, a set of attributes, a list of elements/values, and I will return an element”

Database Management Systems, R. Ramakrishnan

26

---

---

---

---

---

---

---

---

---

## XML Query Data Model

### Example:

```
<book price = "55"
 currency = "USD">
 <title> Foundations ... </title>
 <author> Abiteboul </author>
 <author> Hull </author>
 <author> Vianu </author>
 <year> 1995 </year>
</book>
```

```
book1= elemNode(book,
 {price2, currency3},
 [title4,
 author5,
 author6,
 author7,
 year8])

price2 = attrNode(...) /* next */
currency3 = attrNode(...)
title4 = elemNode(title, string9)
...
```

Database Management Systems, R. Ramakrishnan

27

---

---

---

---

---

---

---

---

---



## XML Query Data Model

### Attribute node:

- ❖ attrNode : (QNameValue, ValueNode)  
→ AttrNode

---

---

---

---

---

---

---



## XML Query Data Model

### Example:

```
<book price = "55"
 currency = "USD">
 <title> Foundations ... </title>
 <author> Abiteboul </author>
 <author> Hull </author>
 <author> Vianu </author>
 <year> 1995 </year>
</book>
```

```
price2 = attrNode(price,string10)
string10 = valueNode(...) /* next */
currency3 = attrNode(currency,
 string11)
string11 = valueNode(...)
```

---

---

---

---

---

---

---



## XML Query Data Model

### Value node:

- ❖ ValueNode = StringValue |  
 BoolValue |  
 FloatValue ...
- ❖ stringValue : string → StringValue
- ❖ boolValue : boolean → BoolValue
- ❖ floatValue : float → FloatValue

---

---

---

---

---

---

---



## XML Query Data Model

Example:

```
<book price = "55"
 currency = "USD">
 <title> Foundations ... </title>
 <author> Abiteboul </author>
 <author> Hull </author>
 <author> Vianu </author>
 <year> 1995 </year>
</book>
```

```
price2 = attrNode(price,string10)
string10 = valueNode(stringValue("55"))
currency3 = attrNode(currency, string11)
string11 = valueNode(stringValue("USD"))

title4 = elemNode(title, string9)
string9 =
valueNode(stringValue("Foundations..."))
```

Database Management Systems, R. Ramakrishnan

31

---

---

---

---

---

---

---

---

---



## XML vs. Semistructured Data

- ❖ Both described best by a graph
- ❖ Both are schema-less, self-describing
- ❖ XML is ordered, ssd is not
- ❖ XML can mix text and elements:

```
<talk> Making Java easier to type and easier to type
 <speaker> Phil Wadler </speaker>
</talk>
```

- ❖ XML has lots of other stuff: attributes, entities, processing instructions, comments

Database Management Systems, R. Ramakrishnan

32

---

---

---

---

---

---

---

---

---



## *Introduction to Semistructured Data and XML*

Chapter 27, Part E  
Based on slides by Dan Suciu  
University of Washington

---

---

---

---

---

---



## *Management of XML and Semistructured Data*

Based upon slides by Dan Suciu

---

---

---

---

---

---



## *Path Expressions*

Examples:

- ❖ Bib.paper
- ❖ Bib.book.publisher
- ❖ Bib.paper.author.lastname

Given an OEM instance, the *value* of a path expression  $p$  is a set of objects

---

---

---

---

---

---

## Path Expressions

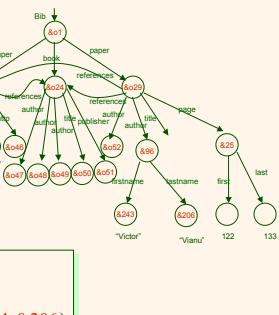
Examples:

DB =

Bib.paper={&#o12,&#o29}  
Bib.book.publisher={&#o51}  
Bib.paper.author.lastname={&#o71,&#o206}

Data

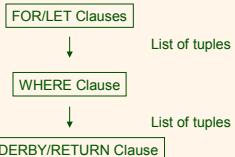
4



## XQuery

Summary:

- ❖ FOR-LET-WHERE-ORDERBY-RETURN = FLWOR



Database Management Systems, R. Ramakrishnan

Instance of Xquery data model

5

## XQuery

- ❖ FOR \$x in expr -- binds \$x to each value in the list expr
- ❖ LET \$x = expr -- binds \$x to the entire list expr
  - Useful for common subexpressions and for aggregations

Database Management Systems, R. Ramakrishnan

6

## FOR v.s. LET

```
FOR $x IN document("bib.xml")/bib/book
RETURN <result> $x </result>
```

Returns:  
<result> <book>...</book></result>  
<result> <book>...</book></result>  
<result> <book>...</book></result>  
...

```
LET $x IN document("bib.xml")/bib/book
RETURN <result> $x </result>
```

Returns:  
<result> <book>...</book>  
<book>...</book>  
<book>...</book>  
...  
</result>

---

---

---

---

---

---

---

---

## Path Expressions

### ❖ Abbreviated Syntax

- /bib/paper[2]/author[1]
- /bib//author
- paper[author/lastname="Vianu"]
- /bib/(paper | book)/title

### ❖ Unabbreviated Syntax

- child::bib/descendant::author
- child::bib/descendant-or-self::\*/child::author
- parent, self, descendant-or-self, attribute

---

---

---

---

---

---

---

---

## XQuery

Find all book titles published after 1995:

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

Result:  
<title> abc </title>  
<title> def </title>  
<title> ghi </title>

---

---

---

---

---

---

---

---

## XQuery

For each author of a book by Morgan Kaufmann, list all books she published:

```
FOR $a IN distinct(document("bib.xml")
 /bib/book[publisher="Morgan Kaufmann"]/author)
RETURN <result>
 $a,
 FOR $t IN /bib/book[author=$a]/title
 RETURN $t
</result>
```

**distinct** = a function that eliminates duplicates

Database Management Systems, R. Ramakrishnan

10

---

---

---

---

---

---

---

---

---

## XQuery

Result:

```
<result>
 <author>Jones</author>
 <title> abc </title>
 <title> def </title>
</result>
<result>
 <author> Smith </author>
 <title> ghi </title>
</result>
```

Database Management Systems, R. Ramakrishnan

11

---

---

---

---

---

---

---

---

---

## XQuery

```
<big_publishers>
 FOR $p IN distinct(document("bib.xml")//publisher)
 LET $b := document("bib.xml")/book[publisher = $p]
 WHERE count($b) > 100
 RETURN $p
</big_publishers>
```

**count** = a (aggregate) function that returns the number of elms

Database Management Systems, R. Ramakrishnan

12

---

---

---

---

---

---

---

---

---

## XQuery

Find books whose price is larger than average:

```
LET $a=avg(document("bib.xml")/bib/book/price)
FOR $b in document("bib.xml")/bib/book
WHERE $b/price > $a
RETURN $b
```

---

---

---

---

---

---

---

## FOR v.s. LET

FOR

- ❖ Binds *node variables* → iteration

LET

- ❖ Binds *collection variables* → one value

---

---

---

---

---

---

---

## Collections in XQuery

- ❖ Ordered and unordered collections
  - /bib/book/author = an ordered collection
  - Distinct(/bib/book/author) = an unordered collection
- ❖ LET \$a = /bib/book → \$a is a collection
- ❖ \$b/author → a collection (several authors...)

```
RETURN <result> $b/author </result>
```

Returns:  
<result> <author>...</author>  
 <author>...</author>  
 <author>...</author>  
 ...  
</result>

---

---

---

---

---

---

---

## Collections in XQuery

What about collections in expressions ?

- ❖ \$b/price → list of n prices
- ❖ \$b/price \* 0.7 → list of n numbers??
- ❖ \$b/price \* \$b/quantity → list of n x m numbers ??
  - Valid only if the two sequences have at most one element
  - Atomization
- ❖ \$book1/author eq "Kennedy" - Value Comparison
- ❖ \$book1/author = "Kennedy" - General Comparison

---

---

---

---

---

---

---

## Sorting in XQuery

```
<publisher_list>
 FOR $p IN distinct(document("bib.xml")/publisher)
 ORDERBY $p
 RETURN <publisher> <name> $p/text() </name> ,
 FOR $b IN document("bib.xml")/book[publisher = $p]
 ORDERBY $b/price DESCENDING
 RETURN <book>
 $b/title ,
 $b/price
 </book>
 </publisher>
</publisher_list>
```

---

---

---

---

---

---

---

## If-Then-Else

```
FOR $h IN //holding
ORDERBY $h/title
RETURN <holding>

 $h/title,
 IF $h/@type = "Journal"
 THEN $h/editor
 ELSE $h/author
</holding>
```

---

---

---

---

---

---

---

## Existential Quantifiers

```
FOR $b IN //book
WHERE SOME $p IN $b//para SATISFIES
 contains($p, "sailing")
 AND contains($p, "windsurfing")
RETURN $b/title
```

---

---

---

---

---

---

---

## Universal Quantifiers

```
FOR $b IN //book
WHERE EVERY $p IN $b//para SATISFIES
 contains($p, "sailing")
RETURN $b/title
```

---

---

---

---

---

---

---

## Other Stuff in XQuery

- ❖ [If-then-else](#)
- ❖ [Universal and existential quantifiers](#)
- ❖ [Sorting](#)
- ❖ [Before and After](#)
  - for dealing with order in the input
- ❖ [Filter](#)
  - deletes some edges in the result tree
- ❖ [Recursive functions](#)

---

---

---

---

---

---

---

## Group-By in Xquery ??

- ❖ No GROUPBY currently in XQuery
- ❖ A recent proposal (next)
  - What do YOU think ?

---

---

---

---

---

---

## Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,
 $y IN $b/@year
WHERE $b/publisher="Morgan Kaufmann"
RETURN GROUPBY $y
 WHERE count($b) > 10
 IN <year> $y </year>
```

← with GROUPBY

Equivalent SQL →

```
SELECT year
FROM Bib
WHERE Bib.publisher="Morgan Kaufmann"
GROUPBY year
HAVING count(*) > 10
```

---

---

---

---

---

---

## Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,
 $a IN $b/author,
 $y IN $b/@year
RETURN GROUPBY $a, $y
 IN <result> $a,
 <year> $y </year>,
 <total> count($b) </total>
 </result>
```

← with GROUPBY

Without GROUPBY →

```
FOR $a IN document("http://www.bn.com")/bib/book/author,
 $y IN $a/@year
LET $b = document("http://www.bn.com")/bib/book[author=$a/@year=$y]
RETURN <result> $a,
 <year> $y </year>,
 <total> count($b) </total>
 </result>
```

Correct if the GROUPBY is node-identity based  
Not equivalent if the GROUPBY is value-based

---

---

---

---

---

---

## Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,
 $a IN $b/author,
 $y IN $b/@year
RETURN GROUPBY $a, $y
 IN <result> $a,
 <year> $y </year>,
 <total> count($b) </total>
</result>
```

← with GROUPBY

Without GROUPBY →

```
FOR $a IN distinct(document("http://www.bn.com")/bib/book/author)
 $y IN distinct(document("http://www.bn.com")/bib/book/@year)
LET $b = document("http://www.bn.com")/bib/book[author=$a/@year=$y]
RETURN
 IF count($b) > 0
 THEN
 <result> $a,
 <year> $y </year>,
 <total> count($b) </total>
</result>
```

Database Management Systems, R. Rab

---

---

---

---

---

---

## Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,
 $a IN $b/author,
 $y IN $b/@year
RETURN GROUPBY $a, $y
 IN <result> $a,
 <year> $y </year>,
 <total> count($b) </total>
</result>
```

← with GROUPBY

Without GROUPBY →

```
FOR $tup IN distinct(FOR $b IN document("http://www.bn.com")/bib,
 $a IN $b/author,
 $y IN $b/@year
 RETURN <tup><a> $a <y> $y </y></tup>),
LET $b = document("http://www.bn.com")/bib/book[author=$a/@year=$y]
RETURN <result> $a,
 <year> $y </year>,
 <total> count($b) </total>
</result>
```

Database Management Systems, R. Ramakrishnan

---

---

---

---

---

---

## Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,
 $a IN $b/author,
 $y IN $b/@year,
 $t IN $b/title,
 $p IN $b/publisher
RETURN
 GROUPBY $p, $y
 IN <result> $p,
 <year> $y </year>,
 GROUPBY $a
 IN <authorEntry>
 $a,
 GROUPBY $t
 IN $t
 <authorEntry>
</result>
```

← Nested GROUPBY's

Database Management Systems, R. Ramakrishnan

27

---

---

---

---

---

---



## *Spatial Data Management*

### **Chapter 28**

---

---

---

---

---

---



## *Types of Spatial Data*

### ❖ Point Data

- Points in a multidimensional space
- E.g., *Raster data* such as satellite imagery, where each pixel stores a measured value
- E.g., Feature vectors extracted from text

### ❖ Region Data

- Objects have spatial extent with location and boundary
- DB typically uses geometric approximations constructed using line segments, polygons, etc., called *vector data*.

---

---

---

---

---

---



## *Types of Spatial Queries*

### ❖ Spatial Range Queries

- *Find all cities within 50 miles of Madison*
- Query has associated region (location, boundary)
- Answer includes overlapping or contained data regions

### ❖ Nearest-Neighbor Queries

- *Find the 10 cities nearest to Madison*
- Results must be ordered by proximity

### ❖ Spatial Join Queries

- *Find all cities near a lake*
- Expensive, join condition involves regions and proximity

---

---

---

---

---

---

## Applications of Spatial Data

- ❖ Geographic Information Systems (GIS)
  - E.g., ESRI's ArcInfo; OpenGIS Consortium
  - Geospatial information
  - All classes of spatial queries and data are common
- ❖ Computer-Aided Design/Manufacturing
  - Store spatial objects such as surface of airplane fuselage
  - Range queries and spatial join queries are common
- ❖ Multimedia Databases
  - Images, video, text, etc. stored and retrieved by content
  - First converted to *feature vector* form; high dimensionality
  - Nearest-neighbor queries are the most common

---

---

---

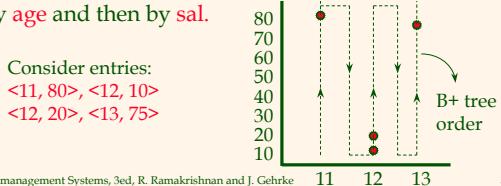
---

---

---

## Single-Dimensional Indexes

- ❖ B+ trees are fundamentally **single-dimensional** indexes.
- ❖ When we create a composite search key B+ tree, e.g., an index on `<age, sal>`, we effectively linearize the 2-dimensional space since we sort entries first by `age` and then by `sal`.



---

---

---

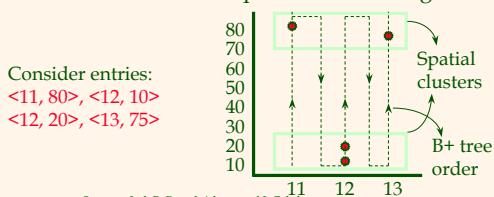
---

---

---

## Multidimensional Indexes

- ❖ A multidimensional index **clusters** entries so as to exploit “nearness” in multidimensional space.
- ❖ Keeping track of entries and maintaining a balanced index structure presents a challenge!



---

---

---

---

---

---

## Motivation for Multidimensional Indexes

### ❖ Spatial queries (GIS, CAD).

- Find all hotels within a radius of 5 miles from the conference venue.
- Find the city with population 500,000 or more that is nearest to Kalamazoo, MI.
- Find all cities that lie on the Nile in Egypt.
- Find all parts that touch the fuselage (in a plane design).

### ❖ Similarity queries (content-based retrieval).

- Given a face, find the five most similar faces.

### ❖ Multidimensional range queries.

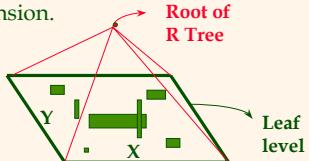
- $50 < \text{age} < 55 \text{ AND } 80K < \text{sal} < 90K$

## What's the difficulty?

- ❖ An index based on spatial location needed.
  - One-dimensional indexes don't support multidimensional searching efficiently. (Why?)
  - Hash indexes only support point queries; want to support range queries as well.
  - Must support inserts and deletes gracefully.
- ❖ Ideally, want to support non-point data as well (e.g., lines, shapes).
- ❖ The R-tree meets these requirements, and variants are widely used today.

## The R-Tree

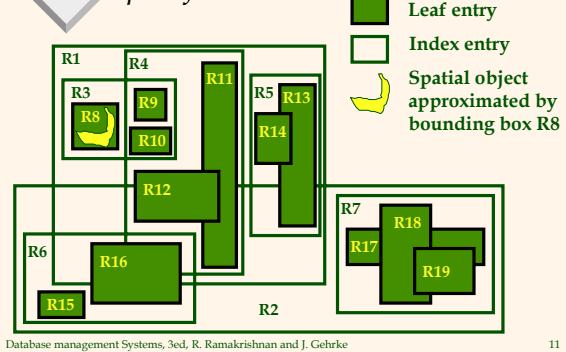
- ❖ The R-tree is a tree-structured index that remains balanced on inserts and deletes.
- ❖ Each key stored in a leaf entry is intuitively a **box**, or collection of **intervals**, with one interval per dimension.
- ❖ Example in 2-D:



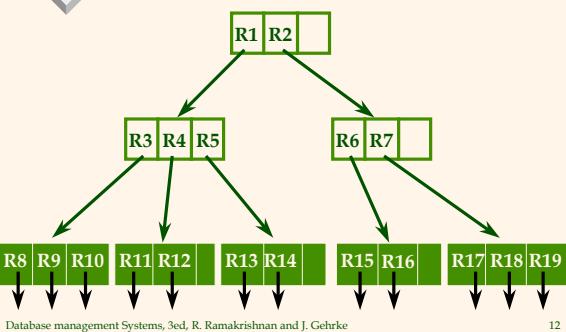
## R-Tree Properties

- ❖ Leaf entry = < n-dimensional box, rid >
  - This is Alternative (2), with *key value* being a box.
  - Box is the tightest bounding box for a data object.
- ❖ Non-leaf entry = < n-dim box, ptr to child node >
  - Box covers all boxes in child node (in fact, subtree).
- ❖ All leaves at same distance from root.
- ❖ Nodes can be kept 50% full (except root).
  - Can choose a parameter  $m$  that is  $\leq 50\%$ , and ensure that every node is at least  $m\%$  full.

## Example of an R-Tree



## Example R-Tree (Contd.)



## Search for Objects Overlapping Box Q

Start at root.

1. If current node is non-leaf, for each entry  $\langle E, \text{ptr} \rangle$ , if box  $E$  overlaps  $Q$ , search subtree identified by  $\text{ptr}$ .
2. If current node is leaf, for each entry  $\langle E, \text{rid} \rangle$ , if  $E$  overlaps  $Q$ ,  $\text{rid}$  identifies an object that might overlap  $Q$ .

Note: May have to search **several** subtrees at each node!  
(In contrast, a B-tree equality search goes to just one leaf.)

---

---

---

---

---

---

---

## Improving Search Using Constraints

- ❖ It is convenient to store boxes in the R-tree as approximations of arbitrary regions, because boxes can be represented compactly.
- ❖ But why not use convex polygons to approximate query regions more accurately?
  - Will reduce overlap with nodes in tree, and reduce the number of nodes fetched by avoiding some branches altogether.
  - Cost of overlap test is higher than bounding box intersection, but it is a main-memory cost, and can actually be done quite efficiently. Generally a win.

---

---

---

---

---

---

---

## Insert Entry $\langle B, \text{ptr} \rangle$

- ❖ Start at root and go down to “best-fit” leaf L.
  - Go to child whose box needs least enlargement to cover B; resolve ties by going to smallest area child.
- ❖ If best-fit leaf L has space, insert entry and stop. Otherwise, split L into L1 and L2.
  - Adjust entry for L in its parent so that the box now covers (only) L1.
  - Add an entry (in the parent node of L) for L2. (This could cause the parent node to recursively split.)

---

---

---

---

---

---

---

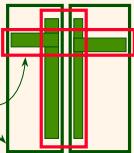
## *Splitting a Node During Insertion*

- ❖ The entries in node L plus the newly inserted entry must be distributed between L1 and L2.
- ❖ Goal is to reduce likelihood of both L1 and L2 being searched on subsequent queries.
- ❖ Idea: Redistribute so as to minimize area of L1 plus area of L2.

Exhaustive algorithm is too slow;  
quadratic and linear heuristics are  
described in the paper.

**GOOD SPLIT!**

**BAD!**



---

---

---

---

---

---

## *R-Tree Variants*

- ❖ The **R\* tree** uses the concept of **forced reinserts** to reduce overlap in tree nodes. When a node overflows, instead of splitting:
  - Remove some (say, 30% of the) entries and reinsert them into the tree.
  - Could result in all reinserted entries fitting on some existing pages, avoiding a split.
- ❖ R\* trees also use a different heuristic, minimizing **box perimeters** rather than **box areas** during insertion.
- ❖ Another variant, the **R+ tree**, avoids overlap by inserting an object into multiple leaves if necessary.
  - Searches now take a single path to a leaf, at cost of redundancy.

---

---

---

---

---

---

## *GiST*

- ❖ The Generalized Search Tree (GiST) abstracts the “tree” nature of a class of indexes including B+ trees and R-tree variants.
  - Striking similarities in insert/delete/search and even concurrency control algorithms make it possible to provide “templates” for these algorithms that can be customized to obtain the many different tree index structures.
  - B+ trees are so important (and simple enough to allow further specialization) that they are implemented specially in all DBMSs.
  - GiST provides an alternative for implementing other tree indexes in an ORDBS.

---

---

---

---

---

---

## Indexing High-Dimensional Data

- ❖ Typically, high-dimensional datasets are collections of points, not regions.
  - E.g., Feature vectors in multimedia applications.
  - Very sparse
- ❖ Nearest neighbor queries are common.
  - R-tree becomes worse than sequential scan for most datasets with more than a dozen dimensions.
- ❖ As dimensionality increases **contrast** (ratio of distances between nearest and farthest points) usually decreases; “nearest neighbor” is not meaningful.
  - In any given data set, advisable to empirically test contrast.

---

---

---

---

---

---

## Summary

- ❖ Spatial data management has many applications, including GIS, CAD/CAM, multimedia indexing.
  - Point and region data
  - Overlap/containment and nearest-neighbor queries
- ❖ Many approaches to indexing spatial data
  - R-tree approach is widely used in GIS systems
  - Other approaches include Grid Files, Quad trees, and techniques based on “space-filling” curves.
  - For high-dimensional datasets, unless data has good “contrast”, nearest-neighbor may not be well-separated

---

---

---

---

---

---

## Comments on R-Trees

- ❖ Deletion consists of searching for the entry to be deleted, removing it, and if the node becomes under-full, deleting the node and then re-inserting the remaining entries.
- ❖ Overall, works quite well for 2 and 3 D datasets. Several variants (notably, R+ and R\* trees) have been proposed; widely used.
- ❖ Can improve search performance by using a convex polygon to approximate query shape (instead of a bounding box) and testing for polygon-box intersection.

---

---

---

---

---

---

## Query-by-Example (QBE)

### Online Chapter

Example is the school of mankind,  
and they will learn at no other.  
-- Edmund Burke (1729-1797)

---

---

---

---

---

---

## QBE: Intro

- ❖ A “GUI” for expressing queries.
  - Based on the DRC!
  - Actually invented before GUIs.
  - Very convenient for simple queries.
  - Awkward for complex queries.
- ❖ QBE an IBM trademark.
  - But has influenced many projects
  - Especially PC Databases: Paradox, Access, etc.

---

---

---

---

---

---

## ‘Example Tables’ in QBE

- ❖ Users specify a query by filling in *example tables*, or *skeletons*; we will use these skeletons in our examples.

<i>Reserves</i>	<u>sid</u>	<u>bid</u>	<u>day</u>

<i>Boats</i>	<u>bid</u>	<u>bname</u>	<u>color</u>

<i>Sailors</i>	<u>sid</u>	<u>sname</u>	<u>rating</u>	<u>age</u>

---

---

---

---

---

---

## Basics

- To print names and ages of all sailors:

Sailors	sid	sname	rating	age
		P._N		P._A

- Print all fields for sailors with *rating* > 8, in ascending order by (*rating*, *age*):

Sailors	sid	sname	rating	age
P.			AO(1).>8	AO(2).

- QBE puts unique new variables in blank columns. Above query in DRC (no ordering):  
 $\{(I, N, T, A) \mid (I, N, T, A) \in Sailors \wedge T > 8\}$

## And/Or Queries

Note: MiniQBE uses a slightly different syntax!

- Names of sailors younger than 30 or older than 20:

Sailors	sid	sname	rating	age
		P.		< 30
		P.		> 20

- Names of sailors younger than 30 and older than 20:

Sailors	sid	sname	rating	age
		_Id	P.	< 30
		_Id	P.	> 20

- Names of sailors younger than 30 and *rating* > 4:

Sailors	sid	sname	rating	age
		_Id	P.	> 4
				< 30

## Duplicates

- Single row with P: Duplicates not eliminated by default; can force elimination by using UNQ.

Sailors	sid	sname	rating	age
UNQ.		P.		< 30

- Multiple rows with P: Duplicates eliminated by default! Can avoid elimination by using ALL.

Sailors	sid	sname	rating	age
ALL.	_Id	P.		< 30
	_Id	P.		> 20

## Join Queries

- Names of sailors who've reserved a boat for 8/24/96 and are older than 25 (note that dates and strings with blanks/special chars are quoted):

Sailors	sid	sname	rating	age
	_Id  P._S			> 25
Reserves	sid	bid	day	
	_Id		'8/24/96'	

Note:  
MiniQBE  
uses  
double  
quotes

- Joins accomplished by repeating variables.

---

---

---

---

---

---

---

---

## Join Queries (Contd.)

- Colors of boats reserved by sailors who've reserved a boat for 8/24/96 and are older than 25 :

Sailors	sid	sname	rating	age
	_Id  _S			> 25
Reserves	sid	bid	day	
	_Id  _B		'8/24/96'	

Boats	bid	bname	color
	_B  'Interlake'		P.

---

---

---

---

---

---

---

---

## Join Queries (Contd.)

- Names and ages of sailors who've reserved some boat that is also reserved by the sailor with sid = 22:

Sailors	sid	sname	rating	age
	_Id  P.			P.
Reserves	sid	bid	day	
	22  _B			
	_Id  _B			

---

---

---

---

---

---

---

---

## Unnamed Columns

MiniQBE allows  
P. in multiple tables

- ❖ Useful if we want to print the result of an expression, or print fields from 2 or more relations.
  - QBE allows P. to appear in at most one table!

Sailors	sid	sname	rating	age	
	_Id  P.		_R	_A	P._D  P.(_R/_A)
Reserves	sid	bid	day		
	_Id		_D		

---

---

---

---

---

---

---

---

## "Negative Tables"

- ❖ Can place a negation marker in the relation column:

Sailors	sid	sname	rating	age	
	_Id  P._S				
Reserves	sid	bid	day		
	_Id  _B				

- ❖ Variables appearing in a negated table must also appear in a positive table!

Note:  
MiniQBE  
uses NOT  
or ~.

---

---

---

---

---

---

---

---

## Aggregates

- ❖ QBE supports AVG, COUNT, MIN, MAX, SUM
  - None of these eliminate duplicates, except COUNT
  - Also have AVG.UNQ. etc. to force duplicate elimination
- ❖ The columns with G. are the *group-by* fields; all tuples in a group have the same values in these fields.
  - The (optional) use of .AO orders the answers.
  - Every column with P. must include G. or an aggregate operator.

Sailors	sid	sname	rating	age	
	_Id G. G.PAO  _A  P.AVG._A				

---

---

---

---

---

---

---

---

### Conditions Box

- Used to express conditions involving 2 or more columns, e.g.,  $R/A > 0.2$ .
- Can express a condition that involves a group, similar to the HAVING clause in SQL:

Sailors	sid	sname	rating	age	CONDITIONS
			G.P.	A	AVG_A > 30

- Express conditions involving AND and OR:

Sailors	sid	sname	rating	age	CONDITIONS
	P.			A	20 < A AND A < 30

Database Management Systems 3ed, Online chapter, R. Ramakrishnan and J. Gehrke

13

### Find sailors who've reserved all boats

- A division query; need aggregates (or update operations, as we will see later) to do this in QBE.

Sailors	sid	sname	rating	age	
	P.G._Id				
Reserves	sid	bid	day		CONDITIONS
	_Id	_B1			COUNT_B1 = COUNT_B2

Boats	bid	bname	color	
	B2			

- How can we modify this query to print the names of sailors who've reserved all boats?

Database Management Systems 3ed, Online chapter, R. Ramakrishnan and J. Gehrke

14

### Inserting Tuples

- Single-tuple insertion:

Sailors	sid	sname	rating	age	
I.	74	Janice	7	14	

- Inserting multiple tuples (*rating* is *null* in tuples inserted below):

Sailors	sid	sname	rating	age	CONDITIONS
I.	_Id	_N		A	A > 18 OR _N LIKE 'C%'
Students	sid	name	login	age	
	_Id	_N		A	

Database Management Systems 3ed, Online chapter, R. Ramakrishnan and J. Gehrke

15

## Delete and Update

- Delete all reservations for sailors with  $rating < 4$

Sailors	<u>sid</u>	sname	rating	age
	<u>_Id</u>		< 4	

Reserves	<u>sid</u>	<u>bid</u>	day
D.	<u>_Id</u>		

- Increment the age of the sailor with  $sid = 74$

Sailors	<u>sid</u>	sname	rating	age
	<u>_Id</u>			<u>U._A+1</u>

---



---



---



---



---



---



---



---



---

## Restrictions on Update Commands

- Cannot mix I., D. and U. in a single example table, or combine them with P. or G.
- Cannot insert, update or modify tuples using values from fields of other tuples in the same table.  
Example of an update that violates this rule:

Sailors	<u>sid</u>	sname	rating	age
		john		<u>A</u>
		joe		<u>U._A+1</u>

Should we update *every* Joe's age?  
Which John's age should we use?

---



---



---



---



---



---



---



---



---

## Find sailors who've reserved all boats (Again!)

- We want to find sailors \_Id such that there is no boat \_B that is not reserved by \_Id:

Sailors	<u>sid</u>	sname	rating	age
	<u>_Id</u>	<u>P._S</u>		
Boats	<u>bid</u>	bname	color	<u>Reserves</u>
	<u>_B</u>			<u>sid</u>

- Illegal query! Variable \_B does not appear in a positive row. In what order should the two negative rows be considered? (Meaning changes!)

---



---



---



---



---



---



---



---



---

## A Solution Using Views

- ❖ Find sailors who've not reserved some boat \_B:

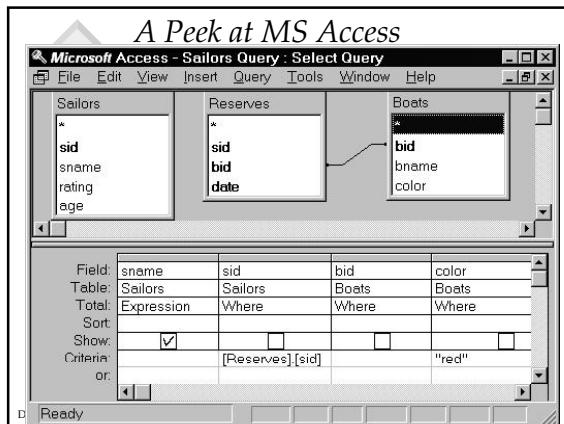
Sailors	sid	sname	rating	age		BadSids	sid
	_Id	P._S				I.	_Id
Boats	bid	bname	color		Reserves	sid	bid
	B					I.	B

- ❖ Next, find sailors not in this 'bad' set:

Sailors	sid	sname	rating	age		BadSids	sid
	_Id	P._S				I.	_Id

Database Management Systems 3ed, Online chapter, R. Ramakrishnan and J. Gehrke

19



## Summary

- ❖ QBE is an elegant, user-friendly query language based on DRC.
- ❖ It is quite expressive (relationally complete, if the update features are taken into account).
- ❖ Simple queries are especially easy to write in QBE, and there is a minimum of syntax to learn.
- ❖ Has influenced the graphical query facilities offered in many products, including Borland's Paradox and Microsoft's Access.

Database Management Systems 3ed, Online chapter, R. Ramakrishnan and J. Gehrke

21