CMPE 138/180B Database Systems

Midterm review

Dr. Gheorghi Guzun

SAN JOSÉ STATE UNIVERSITY

Midterm Exam: 3/27

- During class time
- Closed book. One letter sized page of hand written notes allowed
- Calculator allowed
- Duration: 1hr 10min
- Combination of multiple choice, short answer, SQL queries, and problem solving exercises.

Typical DBMS Functionality

- *Define* a particular database in terms of its data types, structures, and constraints
- *Construct* or Load the initial database contents on a secondary storage medium
- *Manipulating* the database:
  - Retrieval: Querying, generating reports
  - Modification: Insertions, deletions and updates to its content
  - Accessing the database
- *Processing* and *Sharing* by a set of concurrent users and application programs – yet, keeping all data valid and consistent

Application Activities Against a Database

- Applications interact with a database by generating

- Queries: that access different parts of data and formulate the result of a request

- Transactions: that may read some data and "update" certain values or generate new data and store that in the database

- Applications must not allow unauthorized users to access data
- Applications must keep up with changing user requirements against the database

Example of a simple database

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

Main Characteristics of the Database Approach

- **Self-describing nature of a database system:**
  - A DBMS **catalog** stores the description of a particular database (e.g. data structures, types, and constraints)
  - The description is called **meta-data***.
  - This allows the DBMS software to work with different database applications.
- **Insulation between programs and data:**
  - Called **program-data independence**.
  - Allows changing data structures and storage organization without having to change the DBMS access programs.

----

* Some newer systems such as a few NOSQL systems need no meta-data: they store the data definition within its structure making it self describing

Database Schema vs. Database State (continued)

- Distinction
  - The ***database schema*** changes very infrequently.
  - The ***database state*** changes every time the database is updated.

- **Schema** is also called **intension**.
- **State** is also called **extension**.

Example of a Database Schema

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Figure 2.1
Schema diagram for the database in Figure 1.2.

Example of a database state

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

Example of a Relation

Relation Name

Attributes

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Tuples

Figure 5.1
The attributes and tuples of a relation STUDENT.

Formal Definitions - Schema

- The **Schema** (or description) of a Relation:
  - Denoted by R(A1, A2, ...An)
  - R is the **name** of the relation
  - The **attributes** of the relation are A1, A2, ..., An
- Example:

  CUSTOMER (Cust-id, Cust-name, Address, Phone#)
  - CUSTOMER is the relation name
  - Defined over the four attributes: Cust-id, Cust-name, Address, Phone#
- Each attribute has a **domain** or a set of valid values.
  - For example, the domain of Cust-id is 6 digit numbers.

Formal Definitions - Tuple

- A **tuple** is an ordered set of values (enclosed in angled brackets ‘< … >’)
- Each value is derived from an appropriate *domain*.
- A row in the CUSTOMER relation is a 4-tuple and would consist of four values, for example:
  - <632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">
  - This is called a 4-tuple as it has 4 values
  - A tuple (row) in the CUSTOMER relation.
- A relation is a **set** of such tuples (rows)

Formal Definitions - Domain

- A **domain** has a logical definition:
  - Example: “USA_phone_numbers” are the set of 10 digit phone numbers valid in the U.S.
- A domain also has a data-type or a format defined for it.
  - The USA_phone_numbers may have a format: (ddd)ddd-dddd where each d is a decimal digit.
  - Dates have various formats such as year, month, date formatted as yyyy-mm-dd, or as dd mm,yyyy etc.

- The attribute name designates the role played by a domain in a relation:
  - Used to interpret the meaning of the data elements corresponding to that attribute
  - Example: The domain Date may be used to define two attributes named “Invoice-date” and “Payment-date” with different meanings

Formal Definitions - State

- The **relation state** is a subset of the Cartesian product of the domains of its attributes
  - each domain contains the set of all possible values the attribute can take.
- Example: attribute Cust-name is defined over the domain of character strings of maximum length 25
  - dom(Cust-name) is varchar(25)
- The role these strings play in the CUSTOMER relation is that of the *name of a customer*.

Formal Definitions - Summary

- Formally,
  - Given R(A1, A2, ..., An)
  - r(R) $\subset$ dom (A1) X dom (A2) X ....X dom(An)
- R(A1, A2, …, An) is the **schema** of the relation
- R is the **name** of the relation
- A1, A2, …, An are the **attributes** of the relation
- r(R): a specific **state** (or "value" or “population”) of relation R – this is a *set of tuples* (rows)
  - r(R) = {t1, t2, …, tn} where each ti is an n-tuple
  - ti = <v1, v2, …, vn> where each vj *element-of* dom(Aj)

Definition Summary

<u>Informal Terms</u>		<u>Formal Terms</u>
Table		Relation
Column Header		Attribute
All possible Column Values		Domain
Row		Tuple
Table Definition		Schema of a Relation
Populated Table		State of the Relation

Relational Integrity Constraints

- Constraints are **conditions** that must hold on **all** valid relation states.
- There are three *main types* of (explicit schema-based) constraints that can be expressed in the relational model:
  - **Key** constraints
  - **Entity integrity** constraints
  - **Referential integrity** constraints
- Another schema-based constraint is the **domain** constraint
  - Every value in a tuple must be from the *domain of its attribute* (or it could be **null**, if allowed for that attribute)

COMPANY Database Schema

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure 5.5
Schema diagram for the COMPANY relational database schema.

Populated database state for COMPANY

Figure 5.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

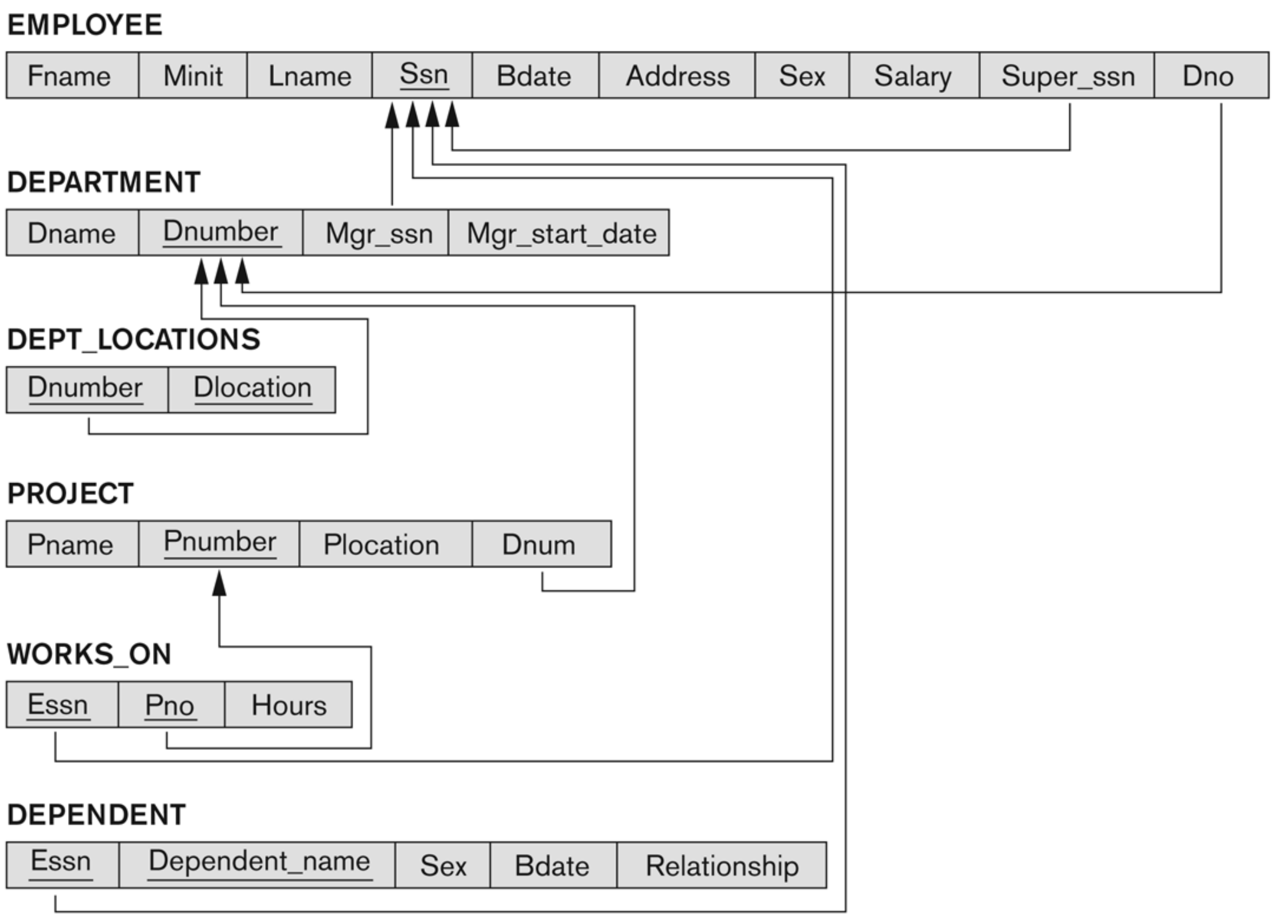
Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Referential Integrity Constraints for COMPANY database

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



Basic SQL Queries,

- SQL Data Definition and Data Types
- Specifying Constraints in SQL
- Basic Retrieval Queries in SQL
- `INSERT`, `DELETE`, and `UPDATE` Statements in SQL
- Additional Features of SQL

The CREATE TABLE Command in SQL

- Specifying a new relation
  - Provide name of table
  - Specify attributes, their types and initial constraints
- Can optionally specify schema:
  - `CREATE TABLE COMPANY.EMPLOYEE ...`

    or
  - `CREATE TABLE EMPLOYEE ...`

Attribute Data Types and Domains in SQL

- Basic **data types**
  - **Numeric** data types
    - Integer numbers: `INTEGER`, `INT`, and `SMALLINT`
    - Floating-point (real) numbers: `FLOAT` or `REAL`, and `DOUBLE PRECISION`
  - **Character-string** data types
    - Fixed length: `CHAR(`*n*`)`, `CHARACTER(`*n*`)`
    - Varying length: `VARCHAR(`*n*`)`, `CHAR VARYING(`*n*`)`, `CHARACTER VARYING(`*n*`)`

Attribute Data Types and Domains in SQL (cont'd.)

- **Bit-string** data types
  - Fixed length: `BIT(`*n*`)`
  - Varying length: `BIT VARYING(`*n*`)`
- **Boolean** data type
  - Values of `TRUE` or `FALSE` or `NULL`
- **DATE** data type
  - Ten positions
  - Components are `YEAR`, `MONTH`, and `DAY` in the form YYYY-MM-DD
  - Multiple mapping functions available in RDBMSs to change date formats

Attribute Data Types and Domains in SQL (cont'd.)

- Additional data types
  - **Timestamp** data type

    Includes the `DATE` and `TIME` fields
    - Plus a minimum of six positions for decimal fractions of seconds
    - Optional `WITH TIME ZONE` qualifier
  - **INTERVAL** data type
    - Specifies a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp
  - **DATE, TIME, Timestamp, INTERVAL** data types can be **cast** or converted to string formats for comparison.

Attribute Data Types and Domains in SQL (cont'd.)

- **Domain**
  - Name used with the attribute specification
  - Makes it easier to change the data type for a domain that is used by numerous attributes
  - Improves schema readability
  - Example:
    - `CREATE DOMAIN SSN_TYPE AS CHAR(9);`
- **TYPE**
  - User Defined Types (UDTs) are supported for object-oriented applications. (See Ch.12) Uses the command: `CREATE TYPE`

Specifying Constraints in SQL

Basic constraints:

- Relational Model has 3 basic constraint types that are supported in SQL:
  - **Key** constraint: A primary key value cannot be duplicated
  - **Entity Integrity** Constraint: A primary key value cannot be null
  - **Referential integrity** constraints : The “foreign key “ must have a value that is already present as a primary key, or may be null.

Specifying Attribute Constraints

Other Restrictions on attribute domains:

- Default value of an attribute
  - **DEFAULT** `<value>`
  - NULL is not permitted for a particular attribute (NOT NULL)
- **CHECK** clause
  - `Dnumber INT NOT NULL CHECK (Dnumber > 0 AND Dnumber < 21);`

Specifying Key and Referential Integrity Constraints

- **`PRIMARY KEY`** clause
  - Specifies one or more attributes that make up the primary key of a relation
  - `Dnumber INT PRIMARY KEY;`
- **`UNIQUE`** clause
  - Specifies alternate (secondary) keys (called CANDIDATE keys in the relational model).
  - `Dname VARCHAR(15) UNIQUE;`

Specifying Key and Referential Integrity Constraints (cont'd.)

- **`FOREIGN KEY`** clause
  - Default operation: reject update on violation
  - Attach **referential triggered action** clause
    - Options include `SET NULL`, `CASCADE`, and `SET DEFAULT`
    - Action taken by the DBMS for `SET NULL` or `SET DEFAULT` is the same for both `ON DELETE` and `ON UPDATE`
    - `CASCADE` option suitable for “relationship” relations

Basic SQL Retrieval Query Block

```
SELECT      <attribute list>
FROM        <table list>
[ WHERE     <condition> ]
[ ORDER BY <attribute list> ];
```

INSERT, DELETE, and UPDATE Statements in SQL

- Three commands used to modify the database:
  - `INSERT`, `DELETE`, and `UPDATE`
- `INSERT` typically inserts a tuple (row) in a relation (table)
- `UPDATE` may update a number of tuples (rows) in a relation (table) that satisfy the condition
- `DELETE` may also update a number of tuples (rows) in a relation (table) that satisfy the condition

Nested Queries (cont'd.)

```
Q4A:  SELECT   DISTINCT Pnumber
      FROM     PROJECT
      WHERE    Pnumber IN
               ( SELECT    Pnumber
                 FROM      PROJECT, DEPARTMENT, EMPLOYEE
                 WHERE     Dnum=Dnumber AND
                           Mgr_ssn=Ssn AND Lname='Smith' )
               OR
               Pnumber IN
               ( SELECT    Pno
                 FROM      WORKS_ON, EMPLOYEE
                 WHERE     Essn=Ssn AND Lname='Smith' );
```

USE of EXISTS

Q7:

```
SELECT Fname, Lname
FROM Employee
WHERE EXISTS (SELECT *
                    FROM DEPENDENT
                    WHERE Ssn= Essn)

          AND EXISTS (SELECT   *
                          FROM Department
                          WHERE Ssn= Mgr_Ssn)
```

USE of EXISTS

Q7:

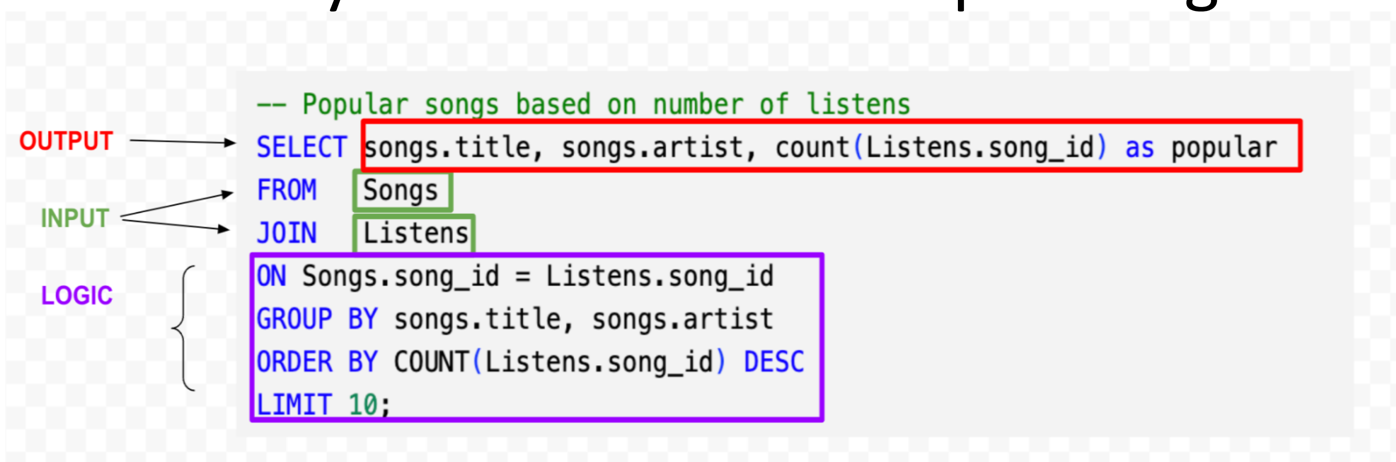
```
SELECT Fname, Lname
FROM Employee
WHERE EXISTS (SELECT *
                  FROM DEPENDENT
                  WHERE Ssn= Essn)

         AND EXISTS (SELECT   *
                         FROM Department
                         WHERE Ssn= Mgr_Ssn)
```

How to think about SELECT versus logic

1. Output versus logic
    a. SELECT specifies the output (like “return(a, b)” in a function)
    b. GROUP BY, ON, ORDER … control the logic

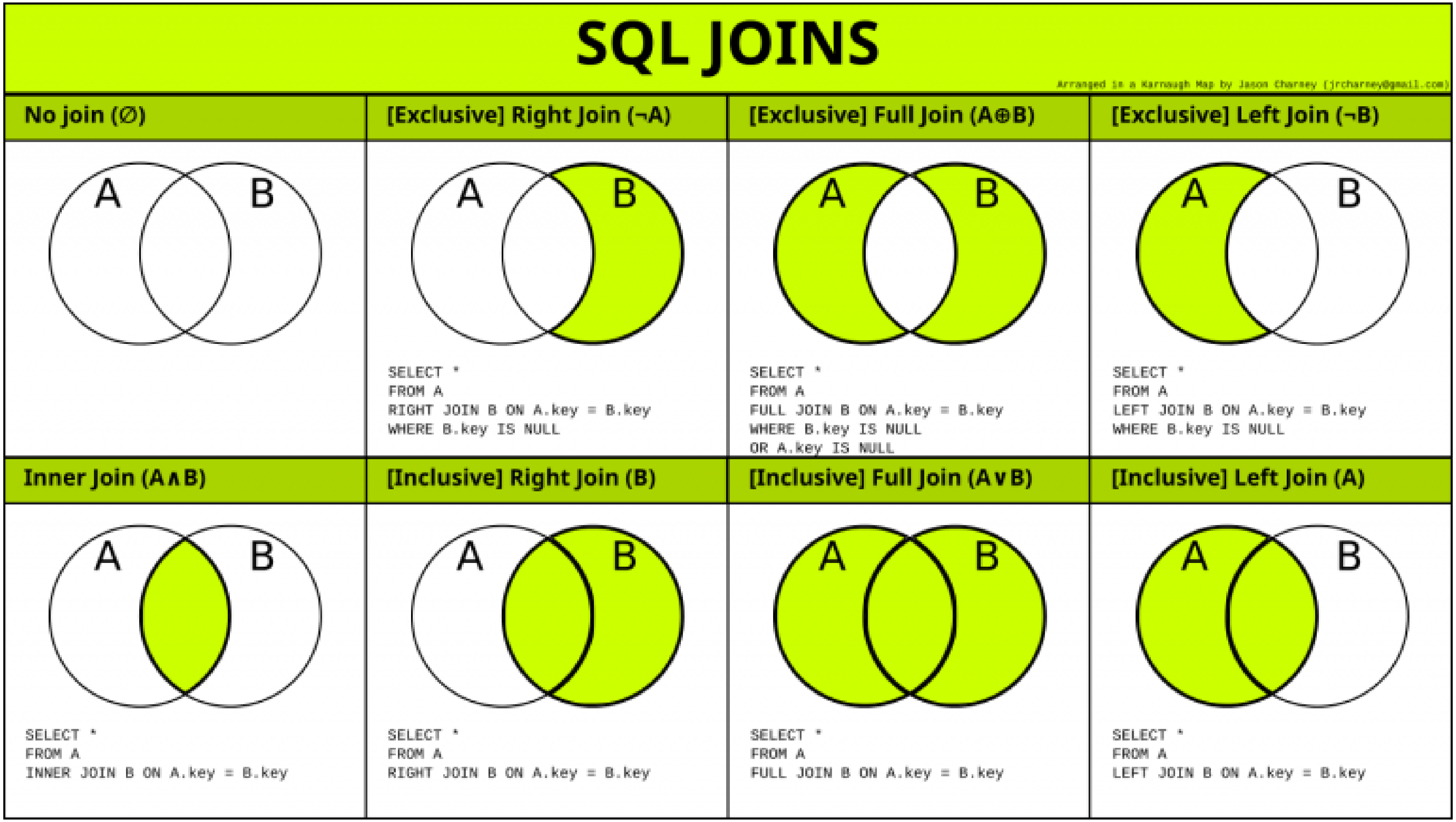
2. After you GROUP BY a set of columns (e.g., title, artist), the extra columns (e.g., genre) are not available anymore for the subsequent logic.



Different Types of JOINed Tables in SQL

- Specify different types of join
  - NATURAL JOIN
  - Various types of OUTER JOIN (LEFT, RIGHT, FULL)
- NATURAL JOIN on two relations R and S
  - No join condition specified
  - Is equivalent to an implicit EQUIJOIN condition for each pair of attributes with same name from R and S

Selecting from multiple tables: SQL JOINS



Aggregate Functions in SQL

- Used to summarize information from multiple tuples into a single-tuple summary
- Built-in aggregate functions
  - **`COUNT`**, **`SUM`**, **`MAX`**, **`MIN`**, and **`AVG`**
- **Grouping**
  - Create subgroups of tuples before summarizing
- To select entire groups, `HAVING` clause is used
- Aggregate functions can be used in the `SELECT` clause or in a `HAVING` clause

Grouping: The GROUP BY Clause

- **Partition** relation into subsets of tuples
  - Based on **grouping attribute(s)**
  - Apply function to each such group independently
- **`GROUP BY`** clause
  - Specifies grouping attributes
- COUNT (*) counts the number of rows in the group

EXPANDED Block Structure of SQL Queries

```
SELECT <attribute and function list>
FROM <table list>
[ WHERE <condition> ]
[ GROUP BY <grouping attribute(s)> ]
[ HAVING <group condition> ]
[ ORDER BY <attribute list> ];
```

Table 7.2 Summary of SQL Syntax

Table 7.2 Summary of SQL Syntax

```
CREATE TABLE <table name> ( <column name> <column type> [ <attribute constraint> ]
                           { , <column name> <column type> [ <attribute constraint> ] }
                           [ <table constraint> { , <table constraint> } ] )
---
DROP TABLE <table name>
ALTER TABLE <table name> ADD <column name> <column type>
---
SELECT [ DISTINCT ] <attribute list>
FROM ( <table name> { <alias> } | <joined table> ) { , ( <table name> { <alias> } | <joined table> ) }
[ WHERE <condition> ]
[ GROUP BY <grouping attributes> [ HAVING <group selection condition> ] ]
[ ORDER BY <column name> [ <order> ] { , <column name> [ <order> ] } ]
---
<attribute list> ::= ( * | ( <column name> | <function> ( ( [ DISTINCT ] <column name> | * ) ) )
                     { , ( <column name> | <function> ( ( [ DISTINCT] <column name> | * ) ) } ) )
---
<grouping attributes> ::= <column name> { , <column name> }
---
<order> ::= ( ASC | DESC )
---
INSERT INTO <table name> [ ( <column name> { , <column name> } ) ]
( VALUES ( <constant value> , { <constant value> } ) { , ( <constant value> { , <constant value> } ) }
| <select statement> )
```

continued on next slide

Table 7.2 (continued) Summary of SQL Syntax

Table 7.2 Summary of SQL Syntax

DELETE FROM <table name> [WHERE <selection condition>]
UPDATE <table name> SET <column name> = <value expression> { , <column name> = <value expression> } [WHERE <selection condition>]
CREATE [UNIQUE] INDEX <index name> ON <table name> (<column name> [<order>] { , <column name> [<order>] }) [CLUSTER]
DROP INDEX <index name>
CREATE VIEW <view name> [(<column name> { , <column name> })] AS <select statement>
DROP VIEW <view name>

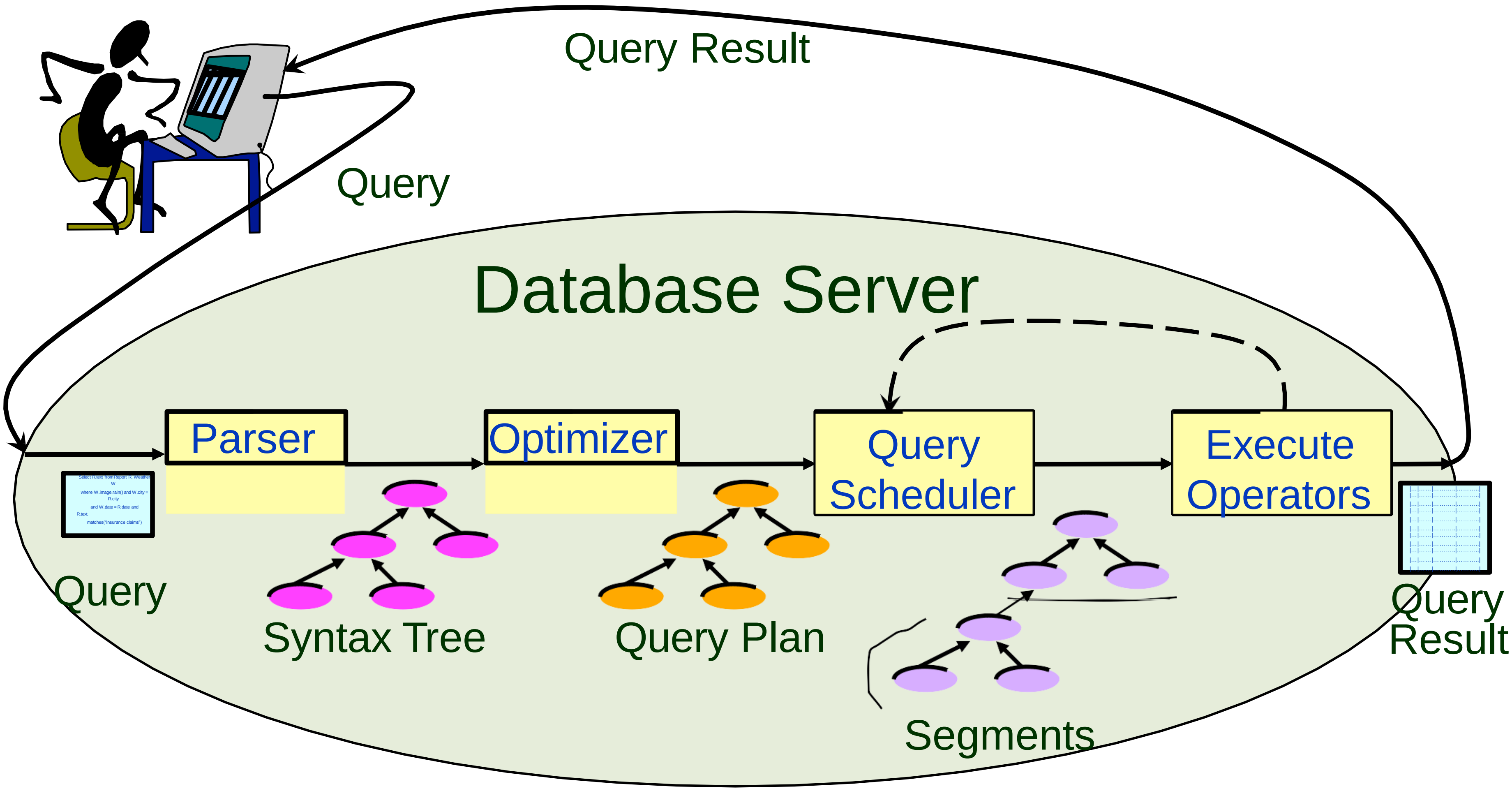
NOTE: The commands for creating and dropping indexes are not part of standard SQL.

Database Management System (DBMS)

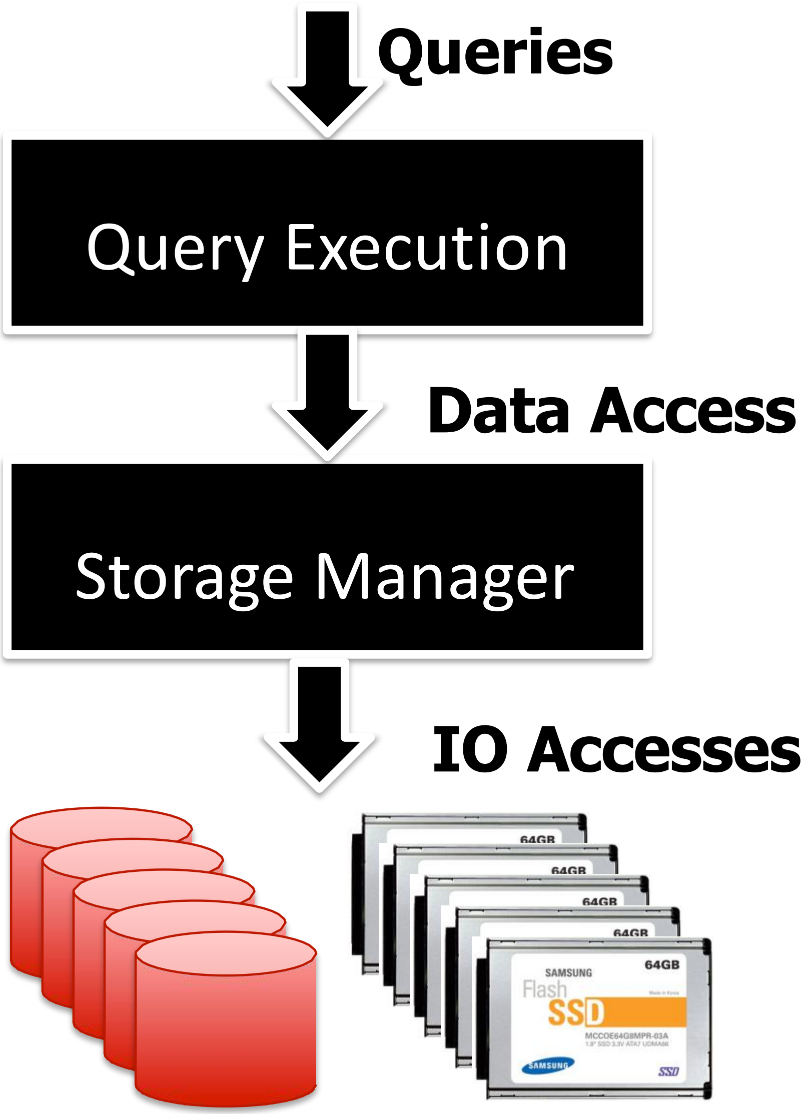
- A DBMS ***manages*** a ***database***.
- A database is a collection of data, usually with some description of the structure of the data.

– Structure description, if present, is described using a ***schema***. e.g. the CREATE TABLE command in SQL

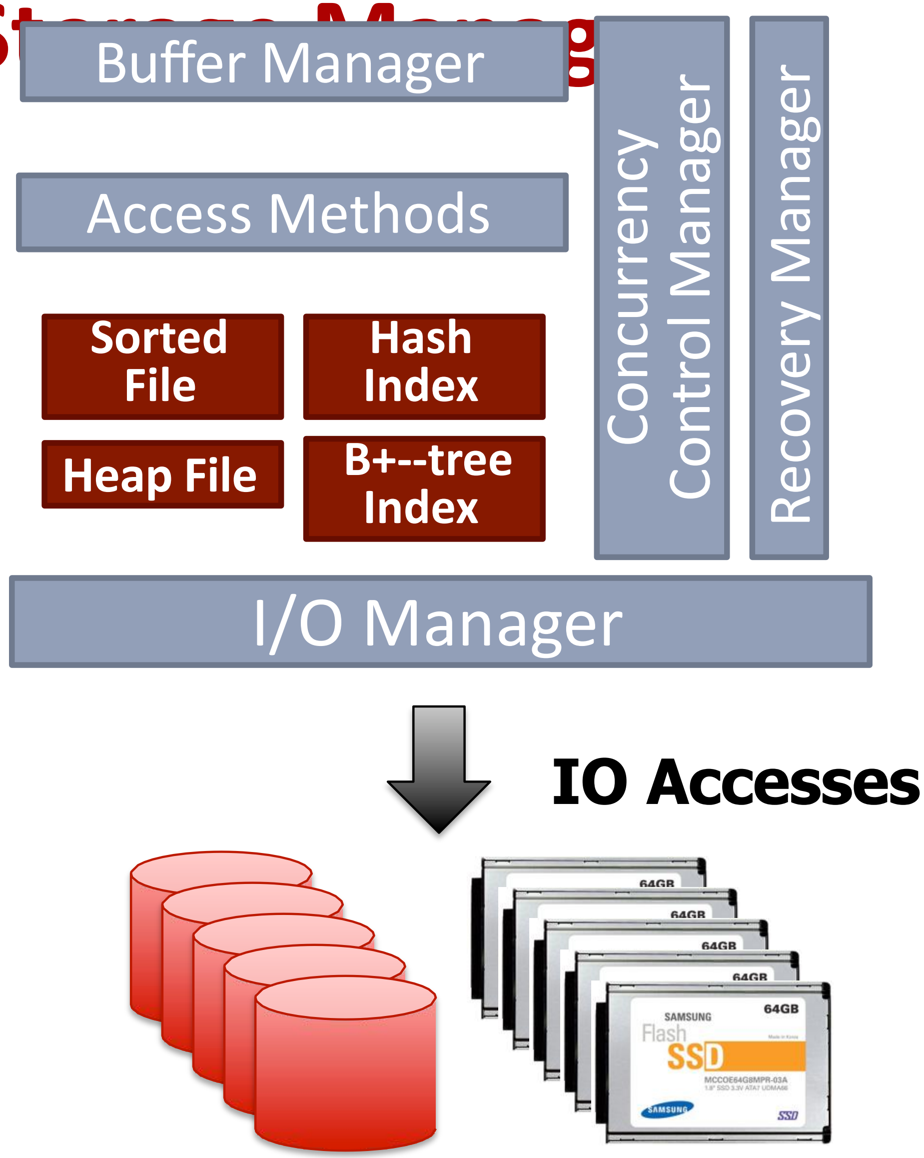
Life Cycle of a Query

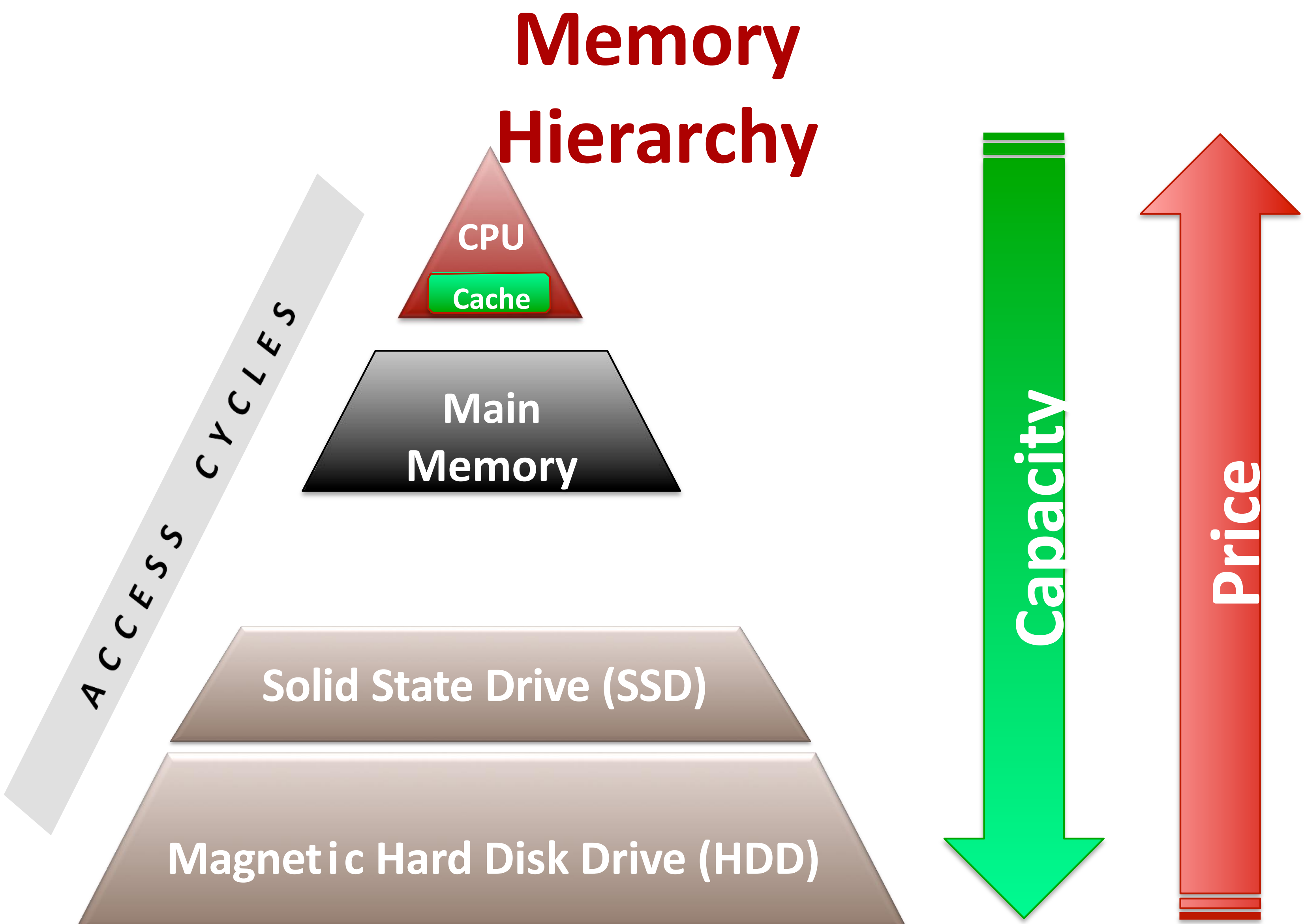


Internal Architecture of a Data Processing System



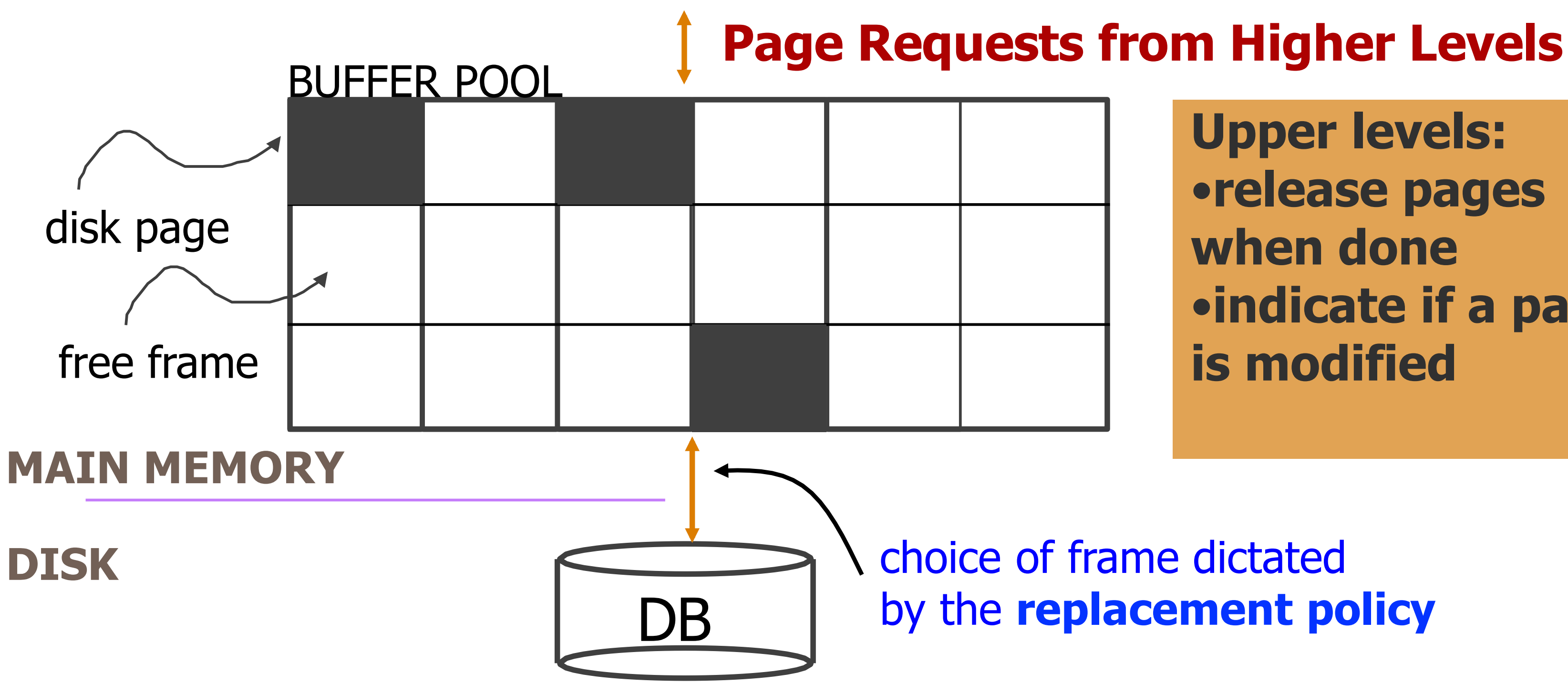
Architecture of a Storage Manag




Memory
Hierarchy
CPU
Cache
Main
Memory
Solid State Drive (SSD)
Magnetic Hard Disk Drive (HDD)
ACCESS CYCLES
Capacity
Price

Buffer Management in a DBMS

- Data must be in RAM for DBMS to operate on it!
  – Can’t keep all the DBMS pages in main memory
- Buffer Manager: Efficiently uses main memory
  – Memory divided into buffer frames: slots for holding disk pages

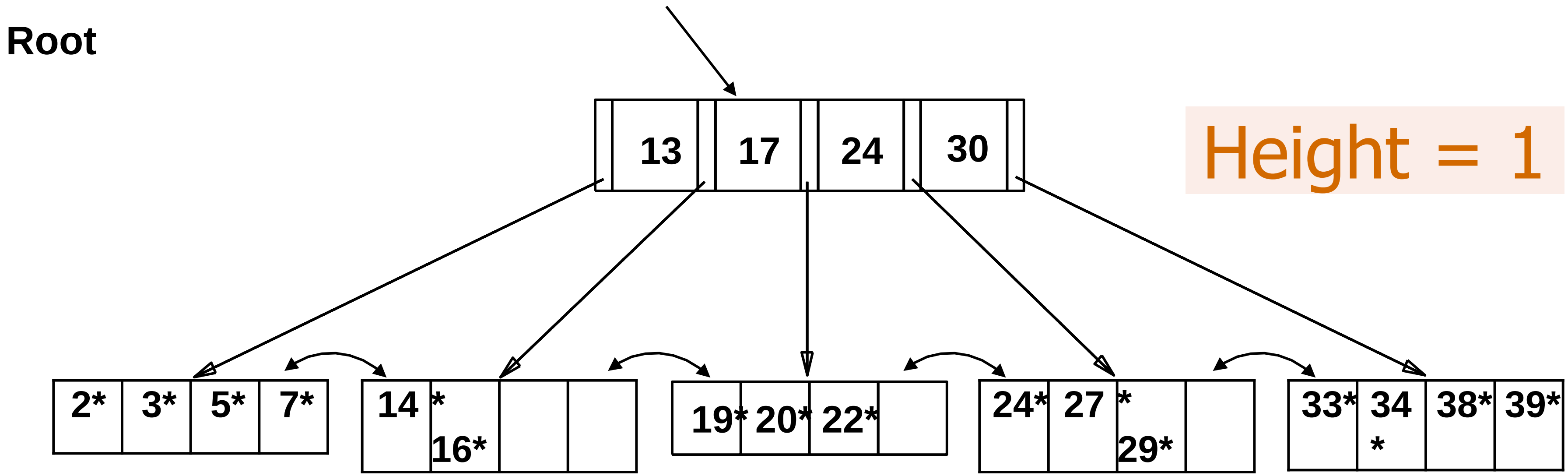


Upper levels:
•release pages when done
•indicate if a page is modified

❖ *Table of <frame#, pageid> pairs is maintained.*

Example B+ Tree

- Search: Starting from root, examine index entries in non--leaf nodes, and traverse down the tree until a leaf node is reached
  - Non-leaf nodes can be searched using a binary or a linear search.
- Search for 5*, 15*, all data entries >=24*

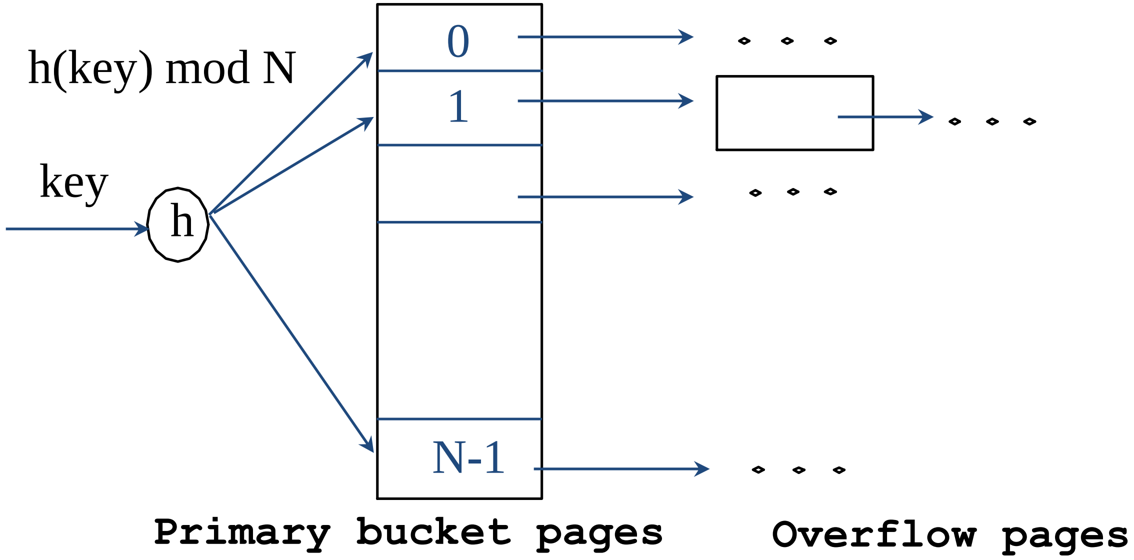


Static Hashing

Number of primary pages *N* fixed, allocated sequentially

- overflow pages may be needed when file grows
- Buckets contain data entries

h(*k*) mod N = bucket for data entry with key *k*

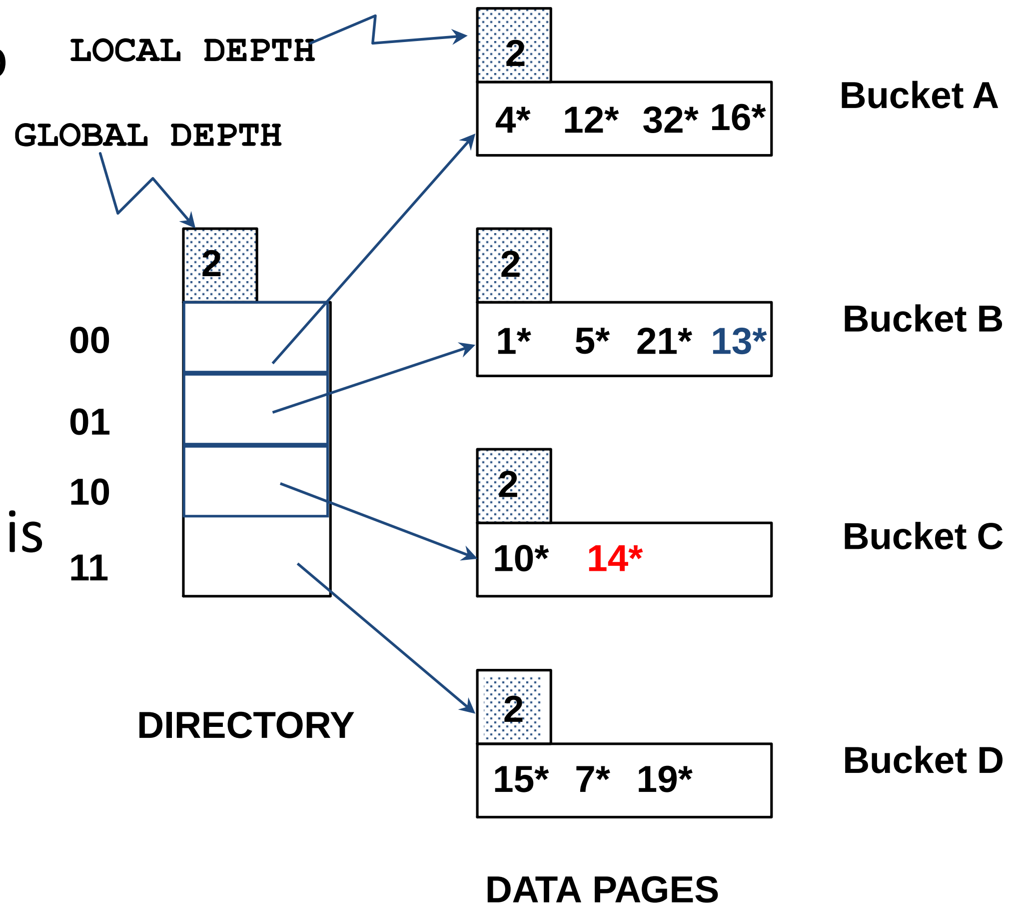


Extendible Hashing Example

- Directory is array of size 4
- Directory entry corresponds to last two bits of hash value

- If **h**(*k*) = 5 = binary 101, it is in bucket pointed to by 01

- Insertion into non-full buckets is trivial
- Insertion into full buckets requires split and **directory doubling**
- E.g., insert h(k)=20



Quad trees

- Nodes split at all dimensions at once
- Division fixed
- Cannot be balanced

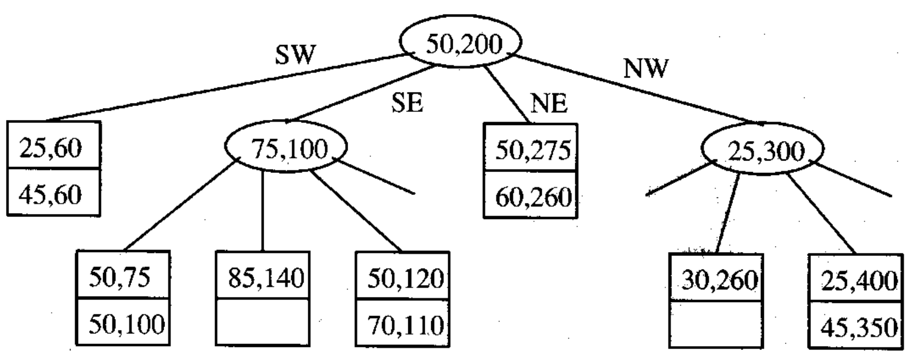


Figure 5.17: A quad tree

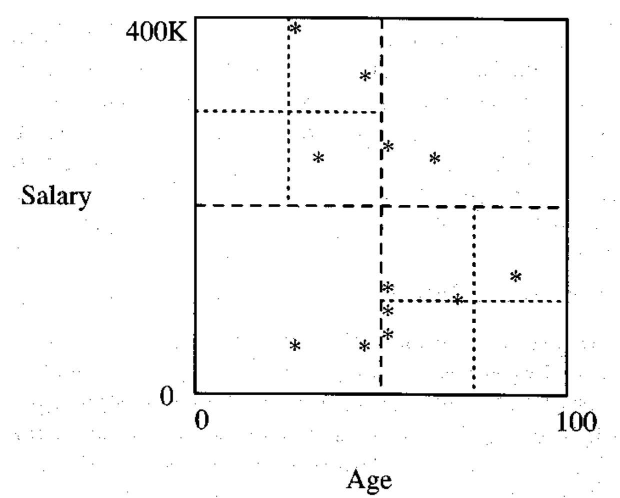
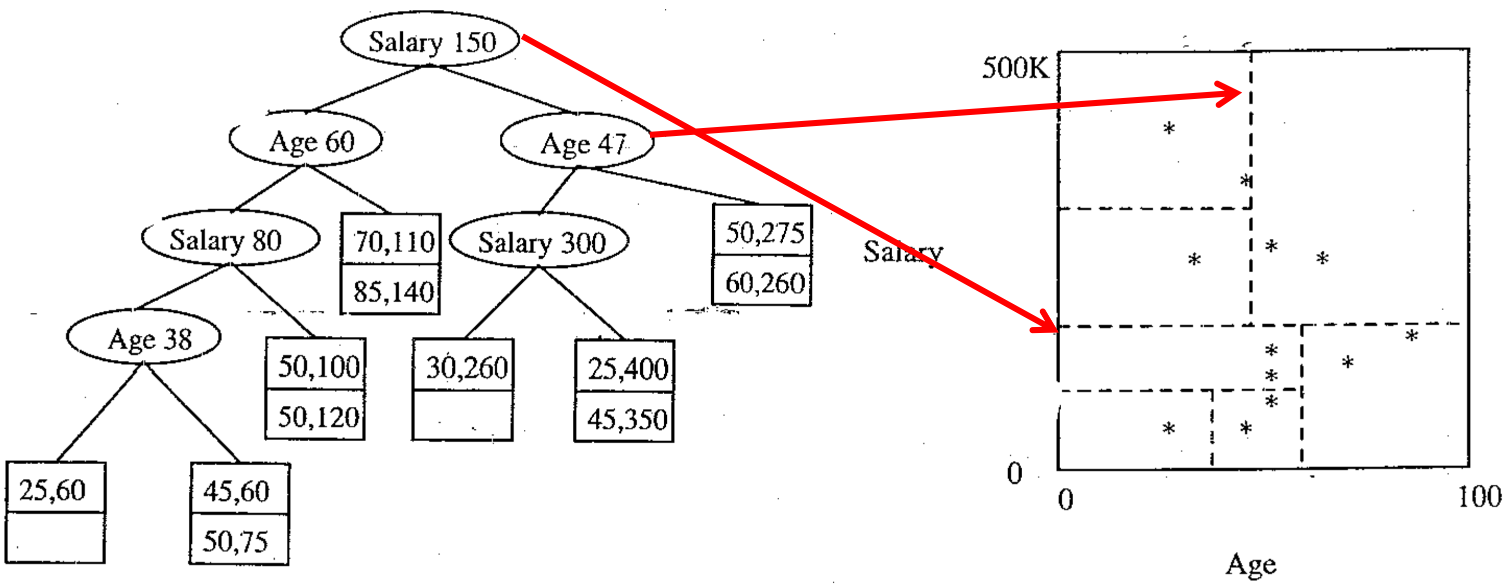


Figure 5.16: Data organized in a quad tree

Kd-trees

- Each level of a *k-d* tree partitions the space into two.
  - Choose one dimension for partitioning at the root level of the tree.
  - Choose another dimension for partitioning in nodes at the next level and so on, cycling through the dimensions.
- In each node, approximately half of the points stored in the sub-tree fall on one side and half on the other.
- Partitioning stops when a node has less than a given maximum number of points.

KD-trees

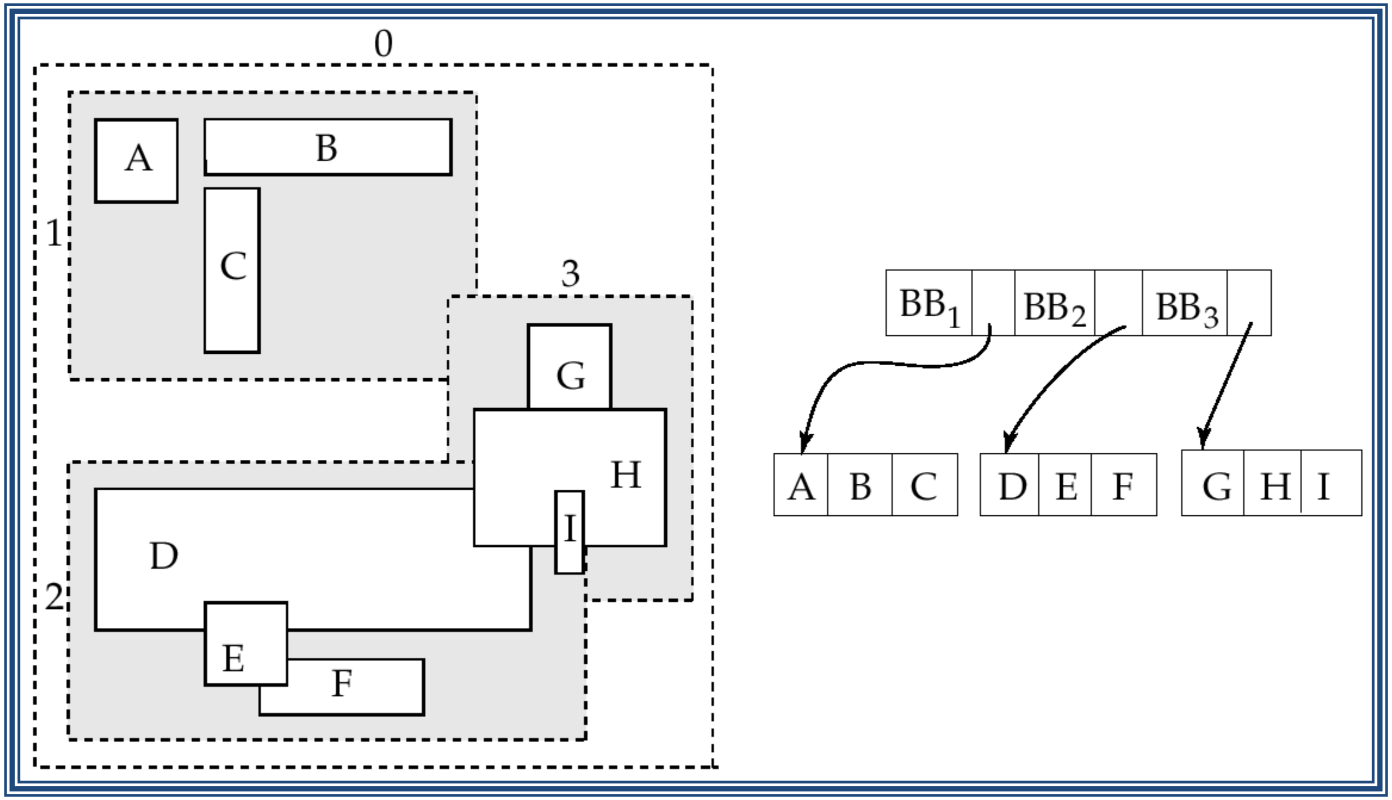

Salary 150
Age 60
Age 47
Salary 80
70,110
85,140
Salary 300
50,275
60,260
Age 38
50,100
50,120
30,260
25,400
45,350
25,60
45,60
50,75
500K
Salary
0
0
100
Age

R-Trees

- A rectangular **bounding box** is associated with each tree node.
  - Bounding box of a leaf node is a minimum sized rectangle that contains all the rectangles/polygons/regions associated with the leaf node.
  - The bounding box associated with a non-leaf node contains the bounding box associated with all its children.
  - Bounding box of a node serves as its key in its parent node (if any)
  - *Bounding boxes of children of a node are allowed to overlap*
- A polygon/region is stored only in one node, and the bounding box of the node must contain the polygon

R-Trees: Example

A set of rectangles (solid line) and the bounding boxes (dashed line) of the nodes of an R-tree for the rectangles.



FAQ: Why do we focus on IO cost?

[1] CPU Cost

AM or check tuple equality in NLJ in RAM)

Typical algorithms in RAM (e.g., quicksort in nLog n)

"Latency Numbers Every Programmer Should Know"

It is hard for humans to get the picture until you translate it to "human numbers":

1 CPU cycle	1 s
Level 1 cache access	3 s
Level 2 cache access	9 s
Level 3 cache access	43 s
Main memory access	6 min
Solid-state disk I/O	2-6 days
Rotational disk I/O	1-12 months
Internet: SF to NYC	4 years
Internet: SF to UK	8 years
Internet: SF to Australia	19 years
OS virtualization reboot	423 years
SCSI command time-out	3000 years
Hardware virtualization reboot	4000 years
Physical system reboot	32 millenia

---

from HDD/SSDs

/e read or write fromHDD/SSD)

E.g., For ExternalSort

IO Cost ~= (sort N pages) $2N \left\lceil \log_B \frac{N}{2B} \right\rceil + 2N$

data, focus on <u>IO cost</u> (i.e., it's the primary factor)

< 10 GBs, just use RAM)

Example NLJ vs. BNLJ: Steel Cage Match

Example: **P**(R) = 1000, **P**(S) = 500,
100 tuples/page ⇒ **T**(R) = 1000*100,**T**(S)=500*100]

P(R) + **T**(R)***P**(S) + OUT

P(R) + **P**(R)***P**(S)/B + OUT

	B= 100 **(+ 1 for output)**	**B= 20** **(+ 1 for output)**
NLJ	(1000 + 1000*100*500 + OUT) ⇒ IO = ~5,001,000 +OUT	(1000 + 1000*100*500 + OUT) ⇒ IO = ~ 5,001,000 +OUT
BNLJ	(1000 + 1000*500/100) ⇒ IO = ~6000 +OUT	(1000 + 1000*500/20) ⇒ IO = ~26,000 IOs +OUT

Small change in algorithm ⇒ Big speedup in JOINs **(~1000x faster)**
Also, notice if we swap R and S, we can save an extra **500 IOs** in BNLJ

Example SMJ vs. BNLJ: Steel Cage Match

Consider P(R) = 1000, P(S) = 500

IO Cost ~= (sort N pages) $2N \left\lceil \log_B \frac{N}{2B} \right\rceil + 2N$

	B = 100	B = 20
SMJ	(Sort R and S in NumPasses (np) =**1** 2*1000*(np+1) + 2*500*(np+1) = 6000 MergeJoin: 1000 + 500 = 1500 IOs) ⇒ IO = 7500 IOs + OUT	(Sort R and S in NumPasses (np)=**2** 2*1000*(np+1) + 2*500*(np+1) = 9000 MergeJoin: 1000 + 500: 1500 IOs) ⇒ IO = 10,500 IOs + OUT
BNLJ	(500 + 1000*500/100) ⇒ IO = 5500+OUT	(500 + 1000*500/20) ⇒ IO = 25500 IOs +OUT

~ Sort(**P**(R)) + Sort(**P**(S)) + **P**(R) + **P**(S) + OUT

P(R) + **P**(R)***P**(S)/B + OUT

SMJ is ~ linear vs. BNLJ is quadratic…
Redo the same with 10x? SMJ much faster.

Join Algorithms: Summary

For R $\bowtie S$ on column A

- NLJ: An example of a *non*-IO aware join algorithm
- BNLJ: Big gains just by being IO aware & reading in chunks of pages!

Quadratic in P(R), P(S)
*I.e. O(P(R)*P(S))*

- SMJ: Sort R and S, then scan over to join!
- HPJ: Partition R and S into buckets using a hash function, then join the (much smaller) matching buckets

Given sufficient buffer space, linear in P(R), P(S)

I.e. ~O(P(R)+P(S))

By only supporting equijoins & taking advantage of this structure!