COMP2411 REVIEW

* Final focus
  - File Organization and Indexing
  - Query Optimization
  - Transaction and Concurrency Control

Lecture 01 Introduction to Database System

Motivation

1. information about particular enterprise
2. file-processing system
3. disadvantages of traditional file-processing systems

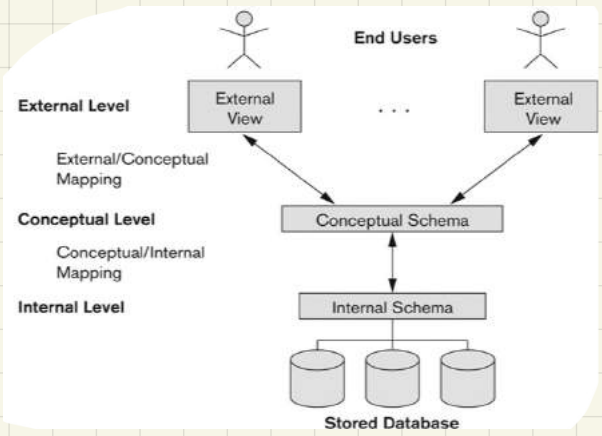
DB non-redundant, persistent collection of logically related records/files that are structured to support various processing and retrieval needs.

DBMS set of software program for creating, storing, updating and accessing data of DB.

* different b/t DBMS & other programming system
  ability to manage persist data
  DBMS: use conveniently, efficiently, robustly in retrieving & storing data

Data Abstraction: abstract view of data – simplify interaction with system.
– hide detail how data is stored and manipulated

* Tutorial
level of abstraction:
physical/internal level: data/structure (table), how data are actually stored
conceptual level: schema, what data are actually stored
view/external level: partial schema



1. The **internal level** has an **internal schema**, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. The **conceptual level** has a **conceptual schema**, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This *implementation conceptual schema* is often based on a *conceptual schema design* in a high-level data model.
3. The **external** or **view level** includes a number of **external schemas** or **user views**. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.

details of data storage & access path

describe entities/data type/relationships /user operation/constraints

describe parts of database that a particular user group is interested in and hides rest of the database from that user group

- **Physical level.** The lowest level of abstraction describes *how* the data are actually stored. The physical level describes complex low-level data structures in detail.
- **Logical level.** The next-higher level of abstraction describes *what* data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as **physical data independence**. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.
- **View level.** The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

Data Model

conceptual tools for describing data
ER model (the entity-relationship model)
relational model

Data Independence

ability to modify a schema definition in one level without affecting schema in next higher level.
physical data independence: modify physical schema — X alter conceptual schema — X rewrite program
logical data independence: modify conceptual schema — x rewrite program

Data Definition Language (DDL)

compile to data dictionary (data about data).

Data Manipulation Language (DML)

Query Language

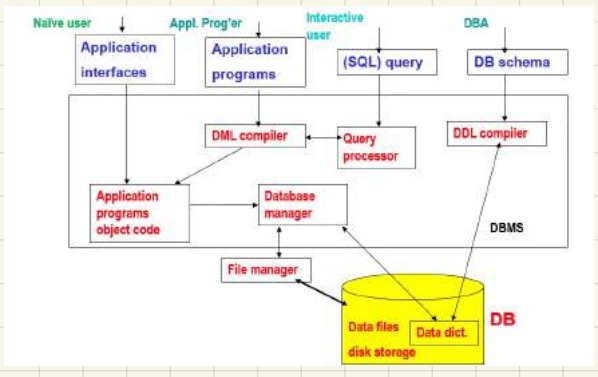
Database user

programmer, interactive users, naive users
DDL — query language — run program

Database Administrator (DBA)

central control over DB

DB System Architechture



Course Objectives

- ER model: characterize relationships among entities
- Relational model: transform from ER diagram to tables
- SQL: language for writing queries
- Relational Algebra: logical way to represent queries
- Normal Forms: how to design good tables
- File Organization: provide file level structure to speed up query
- Query Optimization: transform queries into more efficient ones
- Transactions and Concurrency Control: handle concurrent operations and guarantee correctness of the database

The Entity-Relationship Model (ER Model)

Entity distinguishable object with independent existence.

Entity Set set of entities of same type.

Attribute (property) information describing an entity.

* relationship can carry attributes

Relationship an association among several entities.

Relationship Set set of relationships of the same type.

Integrity Constraints

mapping cardinalities 1:1 1:M / N:1 M:N

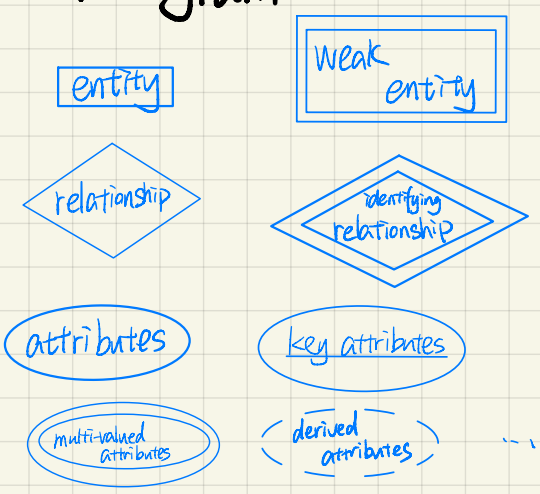
keys: distinguish individual entities or relationships

superkey set of attributes which identify uniquely an entity

candidate key minimal set of attributes which can identify unique an entity.

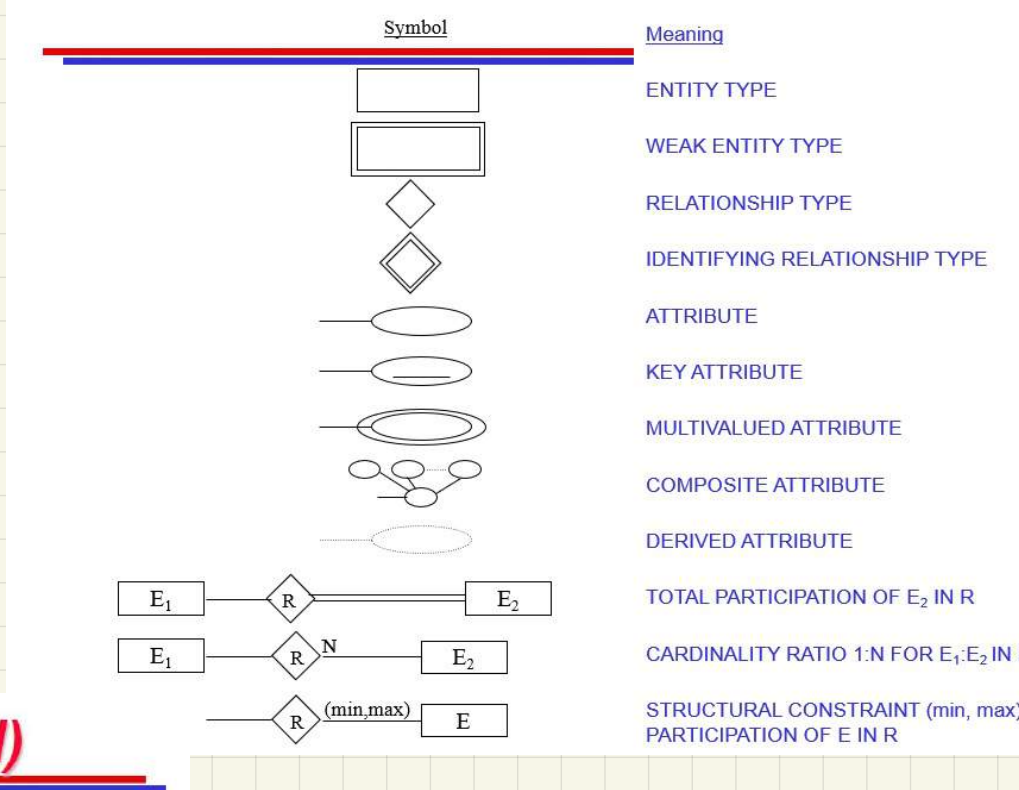
primary key Candidate key chosen by DB designer to identify an entity.

ER Diagram



line: m/n/1

Summary of ER-Diagram Notation



Learning Objectives (ER model)

1. Explaining the meaning of Entity, Attribute and Relationship; able to find them on a ER diagram
2. On ER diagram, identify and explain the meaning of mapping cardinality, participation constraint
3. On ER diagram, able to explain every symbol covered in the lectures; able to transform ER-diagrams to relational tables
4. Able to use all the symbols to design a complete and correct ER diagram based on the user description of the application scenario

D: 1; C: 1,2; B: 1,2,3; A:1,2,3,4

* ER model

Lecture 02 ER, Relational Data Model, SQL

Attributes types

Simple — single atomic value for attribute

Composite — consist of several components, e.g. Name(Firstname, Lastname)

Multivalued — multiple value for attribute, e.g. {color} → nested: example-{ PreviousDegrees (College, Year, Degree, Field) }

Relationship set/types

degree of relationship set — number of participating entity set/types — binary, ternary, n-ary

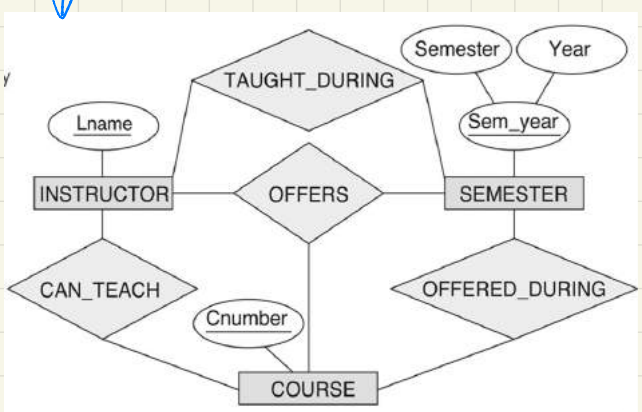
* binary: MANAGES & WORKS_ON

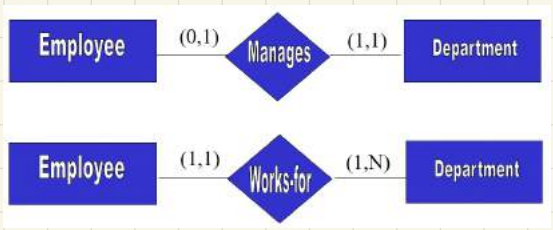
more-than one relationship set exist with same entity sets

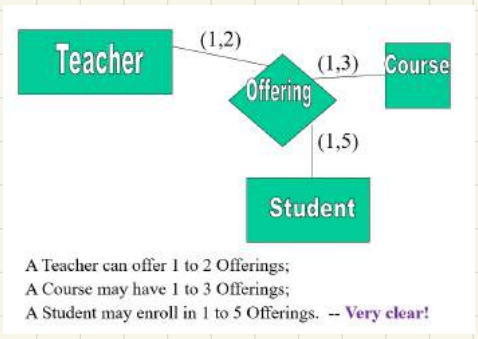
cardinality ratio: 1:1, 1:N, N:1, M:N

participation constraint: partial participation – single line; total participation – double line

(min, max) notation







Relational Data Model

– Basic Structure

relations — data stored as tables(relations) — rows & columns (tuples & attributes)

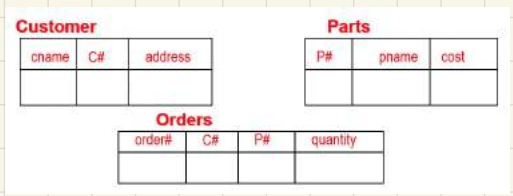
attributes — domain

records — row/tuple in a relation

attribute: particular field of a record

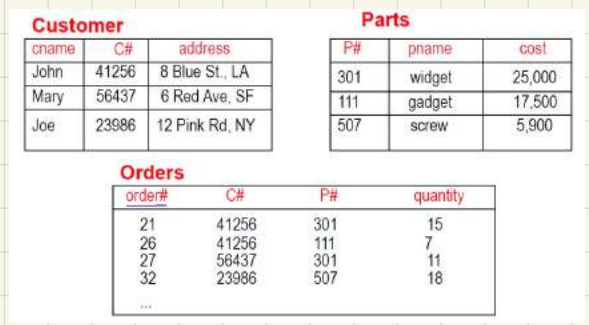
Schema

relative static



DB instance

rather dynamic



Customer

cname	C#	address
John	41256	8 Blue St., LA
Mary	56437	6 Red Ave, SF
Joe	23986	12 Pink Rd, NY

Parts

P#	pname	cost
301	widget	25,000
111	gadget	17,500
507	screw	5,900

Orders

order#	C#	P#	quantity
21	41256	301	15
26	41256	111	7
27	56437	301	11
32	23986	507	18
...			

Definition Summary

Informal Terms	Formal Terms
Table	Relation
Column	Attribute
Row	Tuple
Values in a column	Domain
Table Definition	Schema

Notes:
Although languages like SQL use the informal terms of TABLE (e.g. CREATE TABLE), COLUMN (e.g. SYSCOLUMN variable), the relational database textbooks present the model and operations on it using the formal terms.

Characteristics of Relations

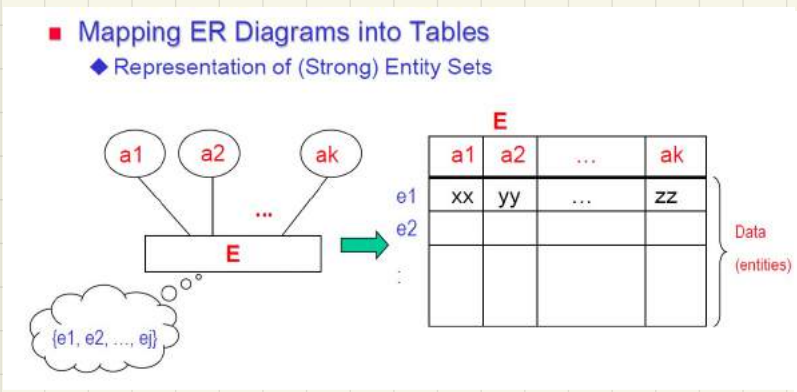
ordering of tuple in a relation r(R) — table form × ordered

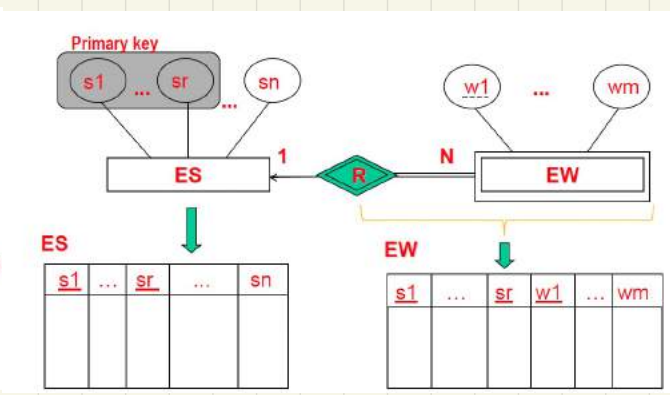
attributes in schema

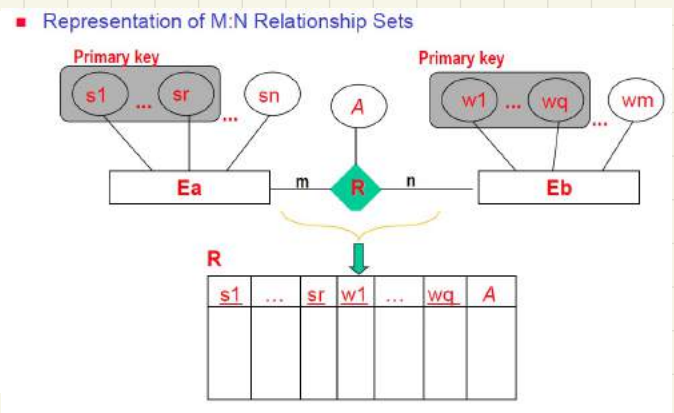
attributes } order
values }

value in a tuple — atomic/indivisible — * null

ER Model vs. Relational Data Model







Relational Query Language → relational algebra.

SQL : Structed Query Language

Learning Objectives (SQL)

1. Able to explain the meaning of a basic SQL query (without aggregate functions and nested queries)
2. Given a requirement written in English, able to write a SQL query to achieve the requirement (without aggregate functions and nested queries)
3. Able to handle nested queries and views for 1 and 2
4. Able to handle aggregate functions for 1 and 2

D: 1; C: 1,2; B: 1,2,3; A:1,2,3,4

- Basic syntax (structure) of SQL:

  SELECT [**DISTINCT**] attribute1, …
  FROM table1 [name1], tabel2 [name2], …
  [**WHERE** requirement]
  [**GROUP BY** (attribute) **HAVING** requirement];

1. requirement:

   [requirement / logical-relation] [AND / OR] requirement

2. logical-relation:

   attribute [> / = / < / <> / <= / >=] [constant / attribute]

   attribute [in / not in] [SQL-expression set SET(attribute)]

   [SQL-expression set SET(attribute)] [CONTAINS / NOT-CONTAINS / = / !=] [SQL-expression set SET(attribute)]

   [EXISTS / NOT EXISTS] [SQL-expression]

example

- Assuming the following relational schema:

  Customer (cname, street, city)
  Branch (bname, assets, b-city)
  Borrow (bname, loan#, cname, amount)
  Deposit (bname, acct#, cname, balance)

◆ *Find all customers of the Sai Kong branch*

```
(SELECT cname
 FROM Deposit
 WHERE bname = "Sai Kong")
        UNION
(SELECT cname
 FROM Borrow
 WHERE bname = "Sai Kong");
```

◆ *Find the name and city of all customers having a loan at the Sai Kong branch*

```
SELECT DISTINCT Customer.cname, city
 FROM Borrow, Customer
 WHERE Borrow.cname = Customer.cname
    AND bname = "Sai Kong";
```

◆ *Find the names of all customers whose street has the substring 'Main' included*

```
SELECT cname
 FROM Customer
 WHERE street LIKE "%Main%";
```

(Note: if we use "_ _ _ Main%", then it becomes a special case.)

◆ *Find all customers who have an account at some branch in which Jones has an account*

Both show the use of tuple variables

solution 1:

```
SELECT DISTINCT T.cname
 FROM Deposit T
 WHERE T.cname != "Jones"
    AND T.bname IN (SELECT S.bname
                    FROM Deposit S
                    WHERE S.cname = "Jones");
```

nested query

solution 2:

```
SELECT DISTINCT T.cname
 FROM Deposit S, Deposit T
 WHERE S.cname = "Jones" AND S.bname = T.bname
    AND T.cname != S.cname;
```

rename

◆ *Find branches having greater assets than **all** branches in N.T.*

solution 1:

```
SELECT X.bname
 FROM Branch X
 WHERE NOT EXISTS (SELECT *
                   FROM Branch Y
                   WHERE Y.b-city = "New Territory"
                      AND Y.assets >= X.assets);
```

empty

solution 2:

```
SELECT bname
FROM Branch
WHERE assets > ALL (SELECT assets
                    FROM Branch
                    WHERE b-city = "New Territory");
```

all

◆ *Find names of all branches that have greater assets than **some** branch located in Kowloon*

solution 1:

```
SELECT bname
 FROM Branch
 WHERE assets > SOME (SELECT assets
                      FROM Branch
                      WHERE b-city = "Kowloon");
```

solution 2:

```
SELECT X.bname
 FROM Branch X, Branch Y
 WHERE X. assets > Y.assets
    AND Y.b-city= "Kowloon";
```

◆ *Find all customers who have a deposit account at **all branches** located in Kowloon*

```
SELECT DISTINCT S.cname
 FROM Deposit S
 WHERE (SELECT T.bname
        FROM Deposit T
        WHERE S.cname = T.cname)
 CONTAINS
       (SELECT bname
        FROM Branch
        WHERE b-city = "Kowloon");
```

The set of branches he/she has a deposit account

The set of branches in Kowloon

("CONTAINS" is dropped in most commercial languages)

(Note: the last 3 examples show SQL supporting set comparisons)

◆ *Find all customers of Central branch who have an account there but no loan there*

```
SELECT C.cname
FROM Customer C
WHERE EXISTS (SELECT *
              FROM Deposit D
              WHERE D.cname = C.cname
                AND D.bname = "Central")
AND NOT EXISTS
              (SELECT *
               FROM Borrow B
               WHERE B.cname = C.cname
                 AND B.bname = "Central");
```

◆ *Find all customers who have a deposit account at ALL branches located in Kowloon*

```
SELECT DISTINCT S.cname
FROM Deposit S
WHERE NOT EXISTS ((SELECT bname        S1
                   FROM Branch
                   WHERE b-city = "Kowloon")
                  MINUS
                  (SELECT T.bname      S2
                   FROM Deposit T
                   WHERE S.cname = T.cname));
```

The set of branches located in KLN

The set of branches customer S holds some deposit accts

$S_1 - S_2 = \emptyset$

(Note: the last 2 examples show SQL can test whether a subquery has any tuples in its result, i.e., to test for empty tables.)

■ *List in alphabetic order all customers having a loan at branches in Kowloon*

```
SELECT DISTINCT cname
FROM Borrow
WHERE b-city = "Kowloon"
ORDER BY cname;
```

By default, in ascending order.

ascending order

■ *List the entire borrow table in descending order of amount, and if several loans have the same amount, order them in ascending order by loan#*

```
SELECT *
FROM Borrow
ORDER BY amount DESC, loan# ASC;
```

descending order

SQL: Data Definition and Manipulation

■ *Table creation*

```
CREATE TABLE table-name
        (A1, D1,
         A2, D2, ...
         Ak, Dk);
```

■ *Table deletion*

```
DROP TABLE table-name;
```

■ *Row/Tuple deletion*

```
DELETE table-name;
```

■ *Table update*

```
ALTER TABLE  table-name
        ADD Aj, Dj
```

(to add new attribute Aj with domain Dj to an existing table.)

■ Views

- ◆ A view is a logical window of one or more tables
- ◆ in SQL, a view is defined using the CREATE VIEW command:

```
CREATE VIEW  v-name  AS <SQL-expression>;
```

E.g., suppose we define a view:

```
CREATE VIEW branch-city  AS
        SELECT bname, city
        FROM Borrow, Customer
        WHERE Borrow.cname = Customer.cname;
```

Now if we want to insert a tuple (Brighton, Shatin) through this view, it will cause:

(Brighton, null, null, null) => Borrow, and
(null, null, Shatin) => Customer

Then what happens to the following query?

```
SELECT * FROM branch-city;
```

(would the tuple (Brighton, Shatin) be part of the answer?)

null value comparison → false

■ *Deletion*

◆ Syntax:

```
DELETE r
WHERE P
```

Meaning: those tuples t in relation r for which P(t) is true are deleted from r.

◆ Example1:

```
DELETE Deposit
WHERE b-name IN (SELECT b-name FROM Branch
                 WHERE b-city = "Laguna")
```

◆ Example2:

```
DELETE Deposit
WHERE balance < (SELECT AVG(balance) FROM Deposit)
```

(Any possible anomalies/problems?)

■ *Insertion*

◆ Syntax:

```
INSERT INTO r VALUES (v1, v2, …, vn);        or

INSERT INTO r
  SELECT clmn_name1, clmn_name2, …, clmn_namek
  FROM s1, s2, …
  WHERE ...
```

◆ Example1:

```
INSERT INTO Deposit
     SELECT b-name, loan#, c-name, 0
     FROM Borrow WHERE b-name = "ClearWaterBay"
```

(Meaning?)

■ *Update*

◆ Syntax:

```
UPDATE r
  SET clmn1 = expression
  WHERE ...
```

Meaning: to change a value in one tuple (without changing all values in the tuple)

◆ Example1:

```
UPDATE Deposit
  SET balance = balance * 1.05
```

(to increase the payment by 5% to all accounts; it is applied to each tuple exactly once.)

◆ Example2:

```
UPDATE Deposit
  SET balance = balance * 1.06 WHERE balance > 10000
UPDATE Deposit
  SET balance = balance * 1.05 WHERE balance <= 10000
```

(to increase the payment by 6% to all accounts with balance over $10000; all others receive 5% increase.)

Null Value

Not	
T	F
U	U
F	T

AND	T	U	F
T	T	U	F
U	U	U	F
F	F	F	F

OR	T	U	F
T	T	T	T
U	T	U	U
F	T	U	F

Aggregated Function

operator: AVG, SUM, MIN, MAX, COUNT

◆ *Find the average account balance at each branch*:

```
SELECT b-name, AVG(balance)
FROM Deposit
GROUP BY b-name;
```

◆ *If only interested in branches where average balance is > $12000*:

```
SELECT b-name, AVG(balance)
FROM Deposit
GROUP BY b-name
HAVING AVG(balance) > 12000;
```

■ *Find those branches with the highest average balance*:

```
SELECT b-name
FROM Deposit
GROUP BY b-name
HAVING AVG(balance) = SELECT MAX(AVG(balance))
                      FROM Deposit
                      GROUP BY b-name);
```

■ *Find the average balance of all depositers who live in Laguna city and have at least 3 accounts*:

```
SELECT AVG(balance)
FROM Deposit, Customer
WHERE Deposit.c-name = Customer.c-name AND city = "Laguna"
GROUP BY Deposit.c-name HAVING COUNT(DISTINCT acct#) >= 3;
```

Lecture 04 Relational Algebra

Learning Objectives

1. Fully understand SELECT σ and PROJECT π operations.
2. Fully understand Set operations: UNION ∪, INTERSECTION ∩, DIFFERENCE -, CARTESIAN PRODUCT X.
3. Fully understand JOIN operations.
4. Given a requirement written in English or SQL, you should be able to transform it into an expression in relational algebra; Given a relational algebra expression, you should be able to transform it into SQL and also explain its meaning

D: 1 C: 1,2 B: 1,2,3 A:1,2,3,4

SELECT σ

select tuples(rows) satisfy selection condition C

$\sigma_C(R)$

selection condition C → Boolean expression

Examples:

$\sigma_{DNO=4}(EMPLOYEE)$

$\sigma_{SALARY>30000}(EMPLOYEE)$

$\sigma_{(DNO=4\ AND\ SALARY>25000)\ OR\ DNO=5}(EMPLOYEE)$

PROJECT π

keep certain attributes (columns) specified in an attribute list L

$\pi_L(R)$

Example: $\pi_{FNAME,LNAME,SALARY}(EMPLOYEE)$

Example: $\pi_{SEX,SALARY}(EMPLOYEE)$

Combine

Example: Retrieve the names and salaries of employees who work in department 4:

$\pi_{FNAME,LNAME,SALARY}(\sigma_{DNO=4}(EMPLOYEE))$

- Alternatively, we can specify explicit intermediate relations for each step:

$DEPT4\_EMPS \leftarrow \sigma_{DNO=4}(EMPLOYEE)$

$R \leftarrow \pi_{FNAME,LNAME,SALARY}(DEPT4\_EMPS)$

SET Operation

UNION $R1 \cup R2$

INTERSECTION $R1 \cap R2$

DIFFERENCE $R1 - R2$

CARTESIAN PRODUCT $R1 \times R2$

(a)

STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b) union

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c) intersect

Fn	Ln
Susan	Yao
Ramesh	Shah

(d) stu - instructor

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e) instructor - stu

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

CARTESIAN PRODUCT

↳ followed by SELECT operation.

Example: Combine each DEPARTMENT tuple with the EMPLOYEE tuple to get the manager.

$DEP\_EMP \leftarrow DEPARTMENT \times EMPLOYEE$

$DEPT\_MANAGER \leftarrow \sigma_{MGRSSN=SSN}(DEP\_EMP)$

As another example: Combine each female EMPLOYEE tuple with the DEPENDENTS tuple to get her dependent(s).

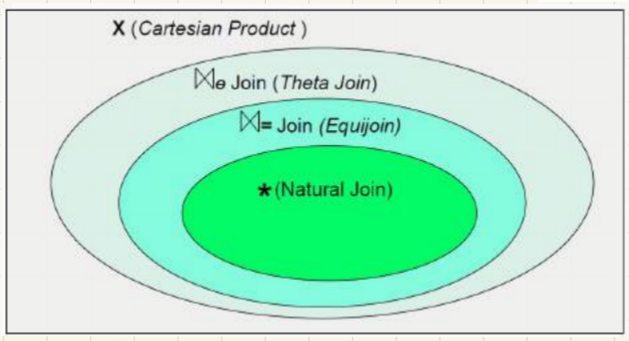
$FemEMPNAMES \leftarrow \pi_{FNAME,LNAME,SSN}(\sigma_{Sex=F}(EMPLOYEE))$

$EMPDEPENDENTS \leftarrow FemEMPNAMES \times DEPENDENTS$

$ActualDEPENDENTS \leftarrow \sigma_{SSN=ESSN}(EMPDEPENDENTS)$

$Result \leftarrow \pi_{FNAME,LNAME,DEPENDENT\_NAME}(ActualDEPENDENTS)$

JOIN Operation



Theta join cartesian product followed by a SELECT condition C/θ → join condition.

$R(A1, A2, ..., Am, B1, B2, ..., Bn) \leftarrow R1(A1, A2, ..., Am) \bowtie_c R2(B1, B2, ..., Bn)$

Equijoin join condition C (one or more equality comparision) $C = (Ai=Bj)\ AND\ ...\ AND\ (Ah=Bk);\ 1 \le i,h \le m,\ 1 \le j,k \le n$

Ai - An ⇒ join attribute of R1

Bj - Bk ⇒ join attribute of R2

Example of using EQUIJOIN:
Retrieve each DEPARTMENT's name and its manager's name:

$T \leftarrow DEPARTMENT \bowtie_{MGRSSN=SSN} EMPLOYEE$

$RESULT \leftarrow \pi_{DNAME,FNAME,LNAME}(T)$

Natural Join redundant join attributes of R2 is eliminated.

$R \leftarrow R1 *_{(join\ attributes\ of\ R1),(join\ attributes\ of\ R2)} R2$

- Example: Retrieve each EMPLOYEE's name and the name of the DEPARTMENT he/she works for:

$T \leftarrow EMPLOYEE *_{(DNO),(DNUMBER)} DEPARTMENT$

$RESULT \leftarrow \pi_{FNAME,LNAME,DNAME}(T)$

LEC05 Integrity Constraints (ICs) & Normal Forms

Learning Objectives

1. Understand all Integrity Constraints covered in the lecture
2. Understand Functional Dependency, able to judge which set of attributes is a candidate key by looking at the given FDs
3. Understand Normalization, able to judge whether a table with a given set of FDs is 1NF, 2NF, 3NF or BCNF
4. Given an initial table together with a set of FDs, able to turn it step by step into a set of tables which is in BCNF

D: 1 C:1,2 B: 1,2,3 A: 1,2,3,4

From Lecture 1

keys: distinguish individual entities or relationships
- superkey: set of attributes which identify uniquely an entity
- candidate key: minimal set of attributes which can identify unique an entity.
- primary key: candidate key chosen by DB designer to identify an entity.

Key Constraints

- Super key: no two tuple have same value for superkey.
- Candidate key: minimal superkey (delete any one)
- Primary key: choice by DB designer to identify an entity
  - underlined

Domain/Entity Integrity Constraints

specify set of possible value (may be associate with an attribute)

primary key ×null * other: may not allow null

E.g.,

C-Name: string of char (30)

Balance: Number (6,2)

Referential Integrity Constraints (foreign key) involving 2 relations

specify a relationship among tuples in 2 relations
ensure a value appear in one relation also appear in another
* subset dependency

```
CREATE TABLE Customer
    (C-name  Char(20) NOT NULL,
     Street  Char(30),
     City    Char(25),
     PRIMARY KEY (C-name));

CREATE TABLE Branch
    (B-name  Char(20) NOT NULL,
     Asset   Integer,
     B-City  Char(20),
     PRIMARY KEY (B-name));

CREATE TABLE Deposit
    (B-name  Char(20),
     Acct#   Char(10) NOT NULL,
     C-name  Char(20) NOT NULL,
     Balance Integer,
     PRIMARY KEY (Acct#, C-name),
     FOREIGN KEY (B-name) REFERENCES Branch,
     FOREIGN KEY (C-name) REFERENCES Customer);
```

Functional Dependency (FD) relation

Let R be a relation scheme, and $\alpha \subseteq R$, $\beta \subseteq R$ (ie, α and β are sets of R's attribuets). We say:

$$\alpha \rightarrow \beta$$

if in any legal relation instance r(R), for all pairs of tuples t1 and t2 in r, we have:

$$(t1[\alpha] = t2[\alpha]) \Rightarrow (t1[\beta] = t2[\beta])$$

Equivalence of FDs

Two sets of FDs F and G are **equivalent** if:

- Every FD in F can be inferred from G, and
- Every FD in G can be inferred from F
- Hence, F and G are equivalent if $F^+ = G^+$

Inference Rules for FD

IR1 Reflexive	$Y \subseteq X \Rightarrow X \rightarrow Y$	Decomposition	$X \rightarrow YZ \Rightarrow \{X \rightarrow Y, X \rightarrow Z\}$
IR2 Augmentation	$X \rightarrow Y \Rightarrow XZ \rightarrow YZ$ (XZ: $X \cup Z$)	Union	$\{X \rightarrow Y, X \rightarrow Z\} \Rightarrow X \rightarrow YZ$
IR3 Transitive	$\{X \rightarrow Y, Y \rightarrow Z\} \Rightarrow X \rightarrow Z$	Pseudotransitivity	$\{X \rightarrow Y, WY \rightarrow Z\} \Rightarrow WX \rightarrow Z$

Relative DB Design normalization theory *lossless join

1NF if all underlying domain contains atomic value only e.g. FIRST(S#, Status, City, P#, Qty)

problem: insert/delete/update anomalies.
- ×represent certain info
- destroy all info
- inconsistency

2NF in 1NF and every non-key attribute is fully depend on any candidate key. e.g. SECOND(S#, Status, City)

SP(S#, P#, Qty)

problem: insert/delete/update anomalies
- ×insert if exist some info
- destroy all info
- potential inconsistency

solution: SECOND table → 2 sub table SC(S#, City), CS(City, Status)

3NF in 2NF and every non-key value is non-transitively depends on any candidate key.

problem: redundancy

BCNF multiple candidate keys (composite one), overlaps common attributes

NFs Defined Informally:

- 1st normal form
  - All attributes are of single atomic values which are dependent on **the key**
- 2nd normal form
  - All non-key attributes depend on **the entire key**
- 3rd normal form
  - All non-key attributes **only** depend on **the key**

Lecture 07 File Organization & Indexing

Learning Objectives

- File organization
  - Unordered file 1
  - Ordered file 2
  - Hash file 3
  - Dynamic and extendible hash 4
- Indexes
  - Primary Indexes 1
  - Clustering Indexes 2
  - Secondary Indexes 3
  - Multilevel Indexes 4
  - Definitions of and operations on B-Trees and B+-Trees 3

D: 1 C: 1,2 B: 1,2,3 A:1,2,3,4 61

File Operation

* File

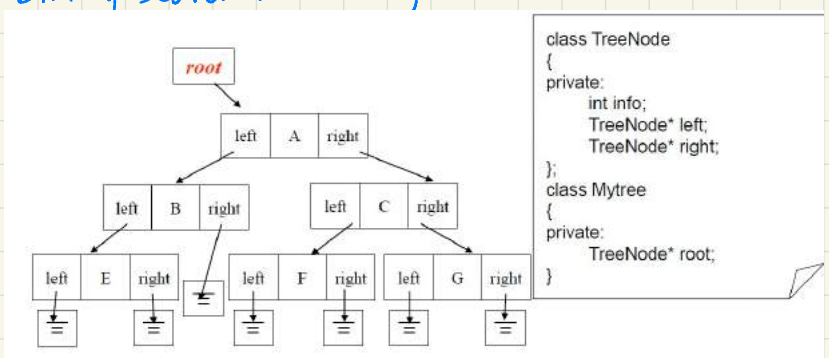
Unordered File pile file

insert new record → end, efficient
search: linear search – read half block require sorting

Ordered File sequential file (sorted by value)

insertion: expensive – to correct position
search: binary search $\log_2 n$ efficient

* binary search with binary tree



Hashed File external hashing

hash key ← one of file field
buckets ← file blocks
$i = h(K)$ h ← hashing function

search efficient.
collision occurs when bucket is full
* overflow

Solving collision problem
- Linear Probing try Bucket_id+1, Bucket_id+2, ...
- Quadratic Probing try Bucket_id+1, Bucket_id+4, ...

disadvantage: fixed bucket when records grows/shrinks

* assume $h(x) = x \% 10$

Homework

4) If a hash structure is used on a search key for which range queries are likely, what property should the hash functions have? **[10 marks]**

Answer:

The hash function should preserve the order of the data.
That is, if h is the hash function, and $x < y$, then it implies that $h(x) < h(y)$.

Dynamic and extendible hash

binary representation of hash value
dynamic hashing → binary tree
extendible hashing → array in 2^d (d → global depth)

* Type of Single-Level Indexes

Primary Indexes on ordered data (key field)

fixed length size with two field | primary key | specific data block |
dense index → search faster
→ more space for store index record

sparse index → search slower
→ need less space

Homework when to use dense index?

2) When is it preferable to use a dense index rather than a sparse index? **[8 marks]**

Answer:

It is preferable to use a dense index rather than a sparse index when
- the file is not ordered on the indexed field (such as when the index is a secondary index) (4 marks)
- or, when the index file is small enough to fit in the main memory (4 marks)

Clustering Indexes on ordered data (non-key field)

record stored in index (not pointer)
not unique for each record

Secondary Indexes ordered data (candidate key & non-key field)

non-ordering index
reduce mapping size

Homework

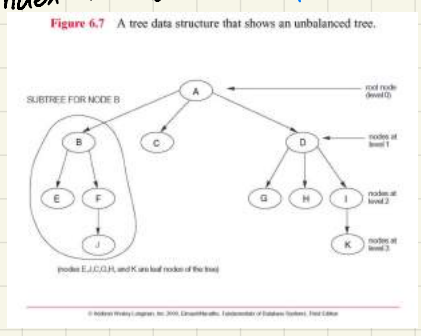
3) What is the difference between primary index and a secondary index? **[7 marks]**

Answer:

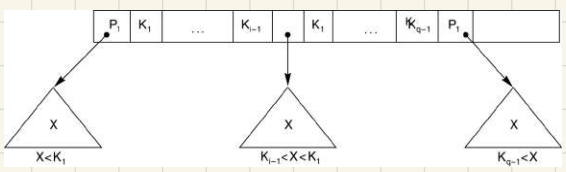
The difference between primary index and a secondary index lies in the facts that:
- The primary index is on the field which specifies the sequential order of the data file (4 marks)
- There can only be one primary index while many secondary indices (3 marks)

Multilevel Indexes

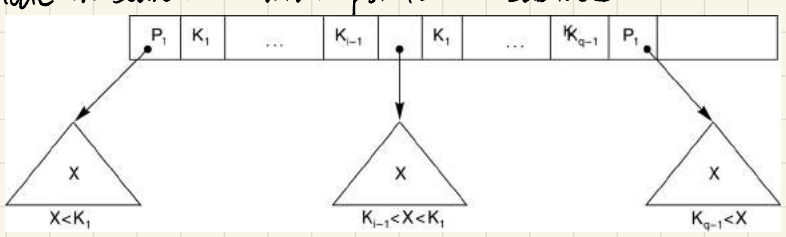
index to index → second-level index



search tree of order p=3 (3 pointer)



node in search tree with pointer to subtree



B Tree and B+ Tree

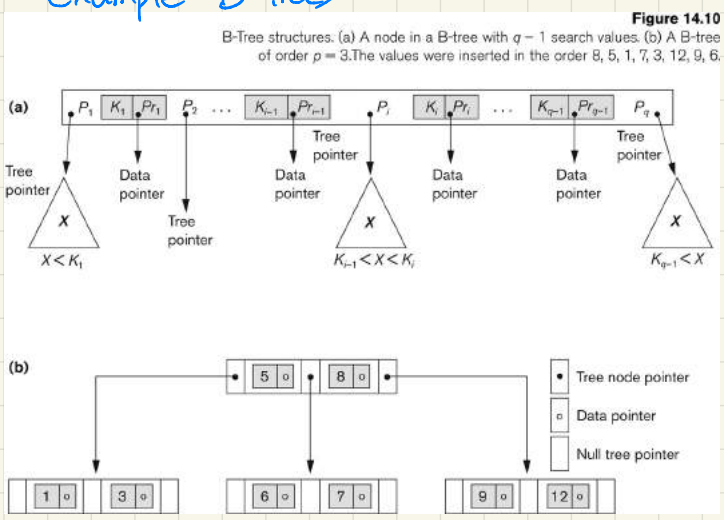
→ leave space for each tree node
→ each node ⇔ a disk block
→ node : half full & completely full

Insertion: if node not full, add
if node full, split to two nodes → other tree level

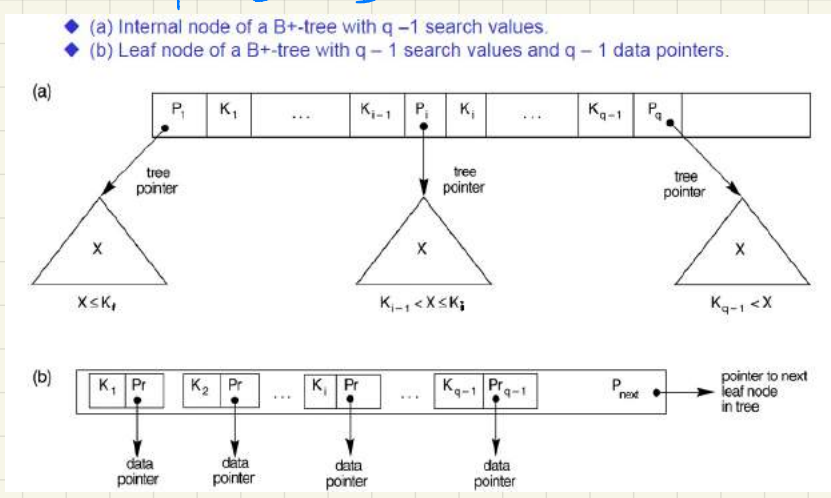
deletion: if node x become less than half full, delete
else → merge neighboring nodes

difference: B tree pointer to data records exist at all level of tree
B+ tree pointer to data records exist at leaf level

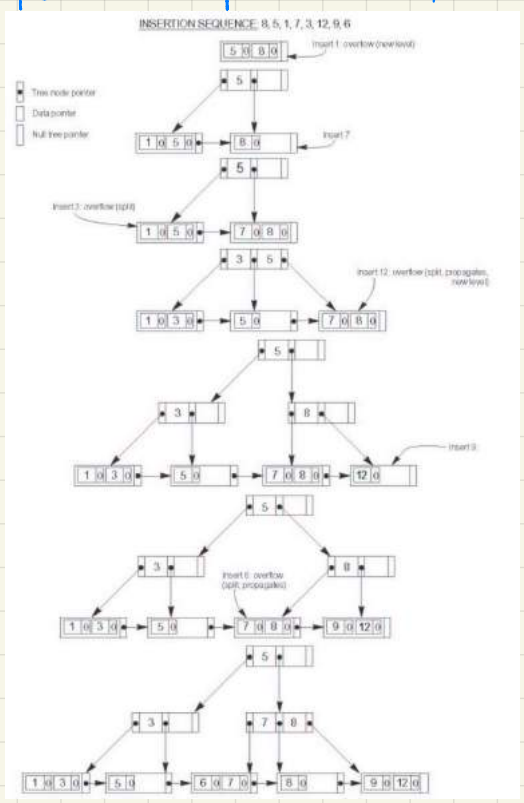
Example B trees



Example B+ trees



Example Insertion of B+ tree p=3 Pleaf = 2 $\lceil \frac{3}{2} \rceil = 2$



Example Deletion of B+ tree

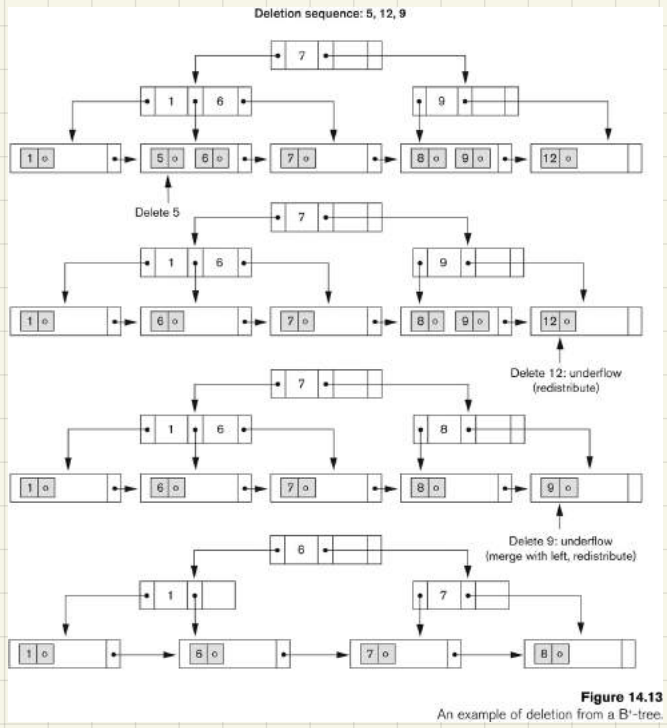


Figure 14.13
An example of deletion from a B+-tree.

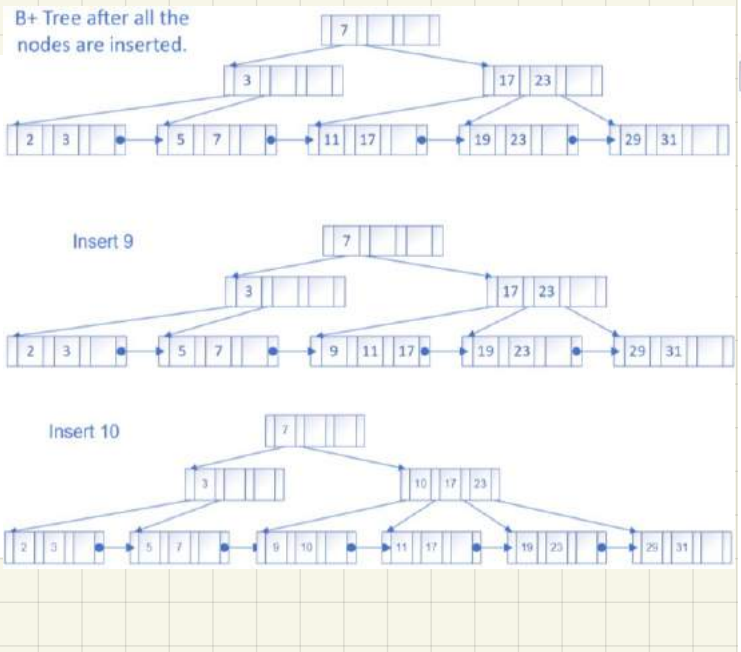
* Tutorial

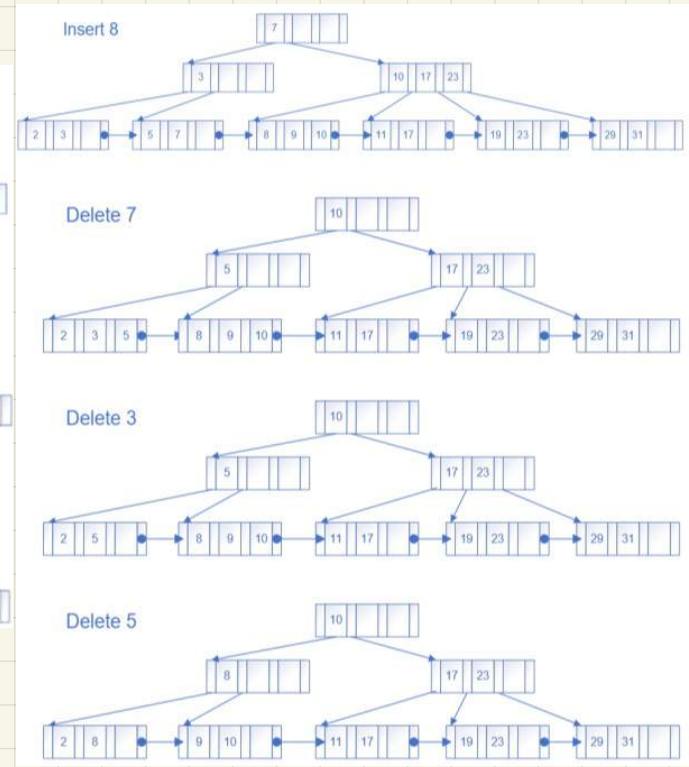
2. Construct a B+ tree for the following set of key values:
(2, 3, 5, 7, 11, 17, 19, 23, 29, 31)

Assuming that the tree is initially empty, values are added in ascending order, and the order of the tree is 4 and the leaf order is 3.

For the tree derived from the above question, show the form of the tree after each of the following series of operations:

a) Insert 9
b) Insert 10
c) Insert 8
d) Delete 7
e) Delete 3
f) Delete 5



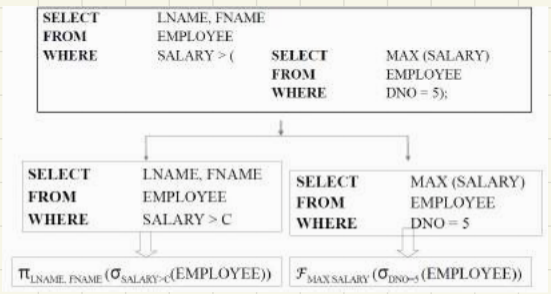


Lecture 09 Query Optimization

Learning Objectives

1. Different ways of implementing "Select". Able to select suitable ways when certain conditions are given.
2. Different ways of implementing "Join". Able to select suitable ways when certain conditions are given; Able to compare different methods (which one is better) given certain condition.
3. Be familiar with the ways to implement aggregate functions utilizing index structure.
4. Understand how to improve the execution of queries, and be able to use those 12 heuristic rules

D: 1 C:1,2 B: 1,2,3 A: 1,2,3,4



Examples:

(OP1): $\sigma_{SSN='123456789'}$ (EMPLOYEE)

(OP2): $\sigma_{DNUMBER>5}$ (DEPARTMENT)

(OP3): $\sigma_{DNO=5}$ (EMPLOYEE)

(OP4): $\sigma_{DNO=5\ AND\ SALARY>30000\ AND\ SEX=F}$ (EMPLOYEE)

(OP5): $\sigma_{ESSN=123456789\ AND\ PNO=10}$ (WORKS_ON)

Query Optimization

- SQL ⇒ Relational Algebra
- Search Method for SELECT
  1. Linear search every record
  2. Binary search equality comparision (ordered file)
  3. Use a primary index/hash key to retrieve single record
     equality comparision on key attribute with a primary index/hash key
  4. use a primary index to retrieve multiple records
     > ≥ < ≤ on key field with primary index
  5. use clustering index to retrieve multiple records
     equality comparision on non-key attribute with clustering index
  6. use secondary index / B+ tree > ≥ < ≤ (range queries)
  7. conjunctive selection
  8. conjunctive selection using complex index
  9. conjuctive selection by intersection

* disjunctive condition (OR)

e.g. for conjunctive selection (AND)

(OP4): $\sigma_{DNO=5\ AND\ SALARY>30000\ AND\ SEX=F}$ (EMPLOYEE)

⇓ optimization need mostly

- Join Operation →

TWO WAY JOIN:

$R *_{A=B} S$

MULTI-WAY JOIN:

$R *_{A=B} S *_{C=D} T$ .....

Example operations:

(OP6): EMPLOYEE * $_{DNO=DNUMBER}$ DEPARTMENT

(OP7): DEPARTMENT * $_{MGRSSN=SSN}$ EMPLOYEE

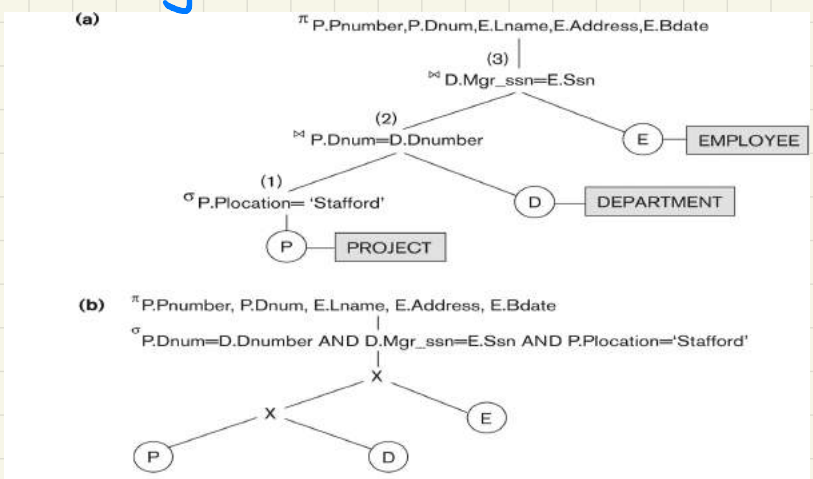
1. Nested (inner-outer) loop approach
2. access structure to retrieve matching records
3. Sort-merge
4. hash join

- Aggregated Operation { table scan / index

  max, min, sum, count, avg

Heuristic Optimization

Query Tree



Query Graph

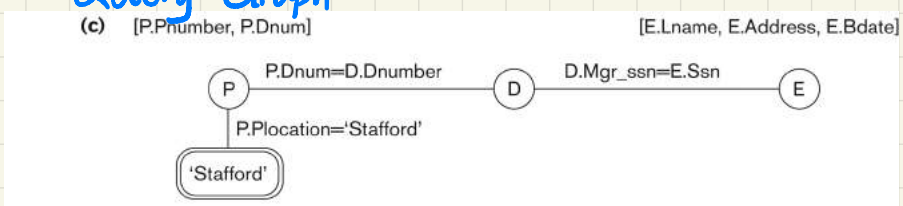


Figure 15.4
Two query trees for the query Q2. (a) Query tree corresponding to the relational algebra expression for Q2. (b) Initial (canonical) query tree for SQL query Q2. (c) Query graph for Q2.

Rules

1. cascade of σ: $\sigma_{c1\ AND\ c2\ AND\ ...\ AND\ cn}(R) = \sigma_{c1}(\sigma_{c2}(...(\sigma_{cn}(R))...))$
2. commutative of σ: $\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$
3. cascade of π: $\pi_{List1}(\pi_{List2}(...(\pi_{Listn}(R))...)) = \pi_{List1}(R)$
4. commuting with π: $\pi_{A1, A2, ..., An}(\sigma_c(R)) = \sigma_c(\pi_{A1, A2, ..., An}(R))$
5. commtativity of * (or ⋈): $R * S = S * R$
6. commuting σ with *: $\sigma_c(R * S) = (\sigma_c(R)) * S$
7. commtativity of set operation: ∩ ∪
8. commuting π with ⋈c: $\pi_L(R \bowtie_c S) = (\pi_{A1, ..., An}(R)) \bowtie_c (\pi_{B1, ..., Bm}(S))$
9. associativity *, X, ∪, ∩: $(R\ q\ S)\ q\ T = R\ q\ (S\ q\ T)$
10. commuting σ with set operation: ∪ ∩ − $\sigma_c(R\ q\ S) = (\sigma_c(R))\ q\ (\sigma_c(S))$
11. π operation commuting with ∪: $\pi_L(R \cup S) = (\pi_L(R)) \cup (\pi_L(S))$

Example

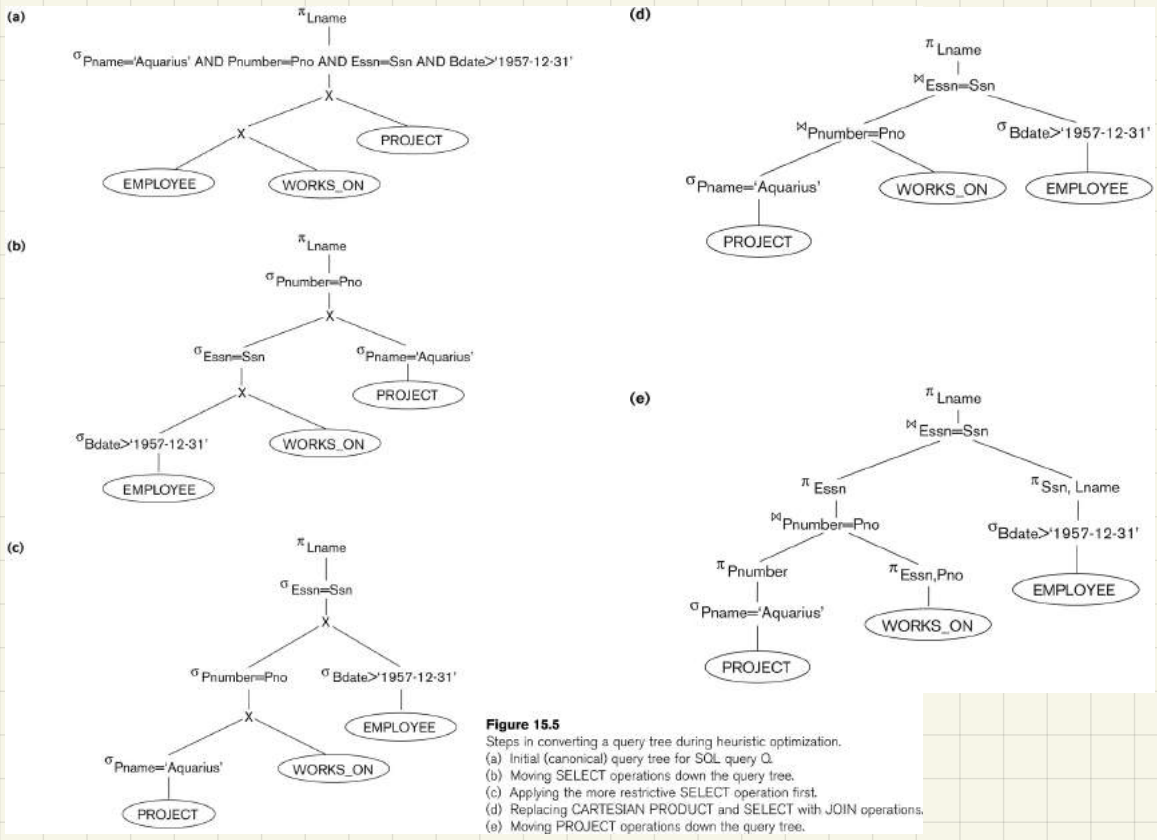
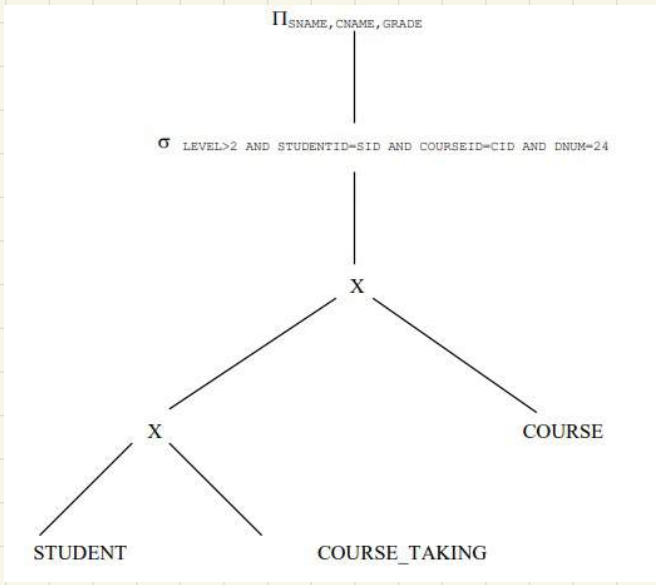
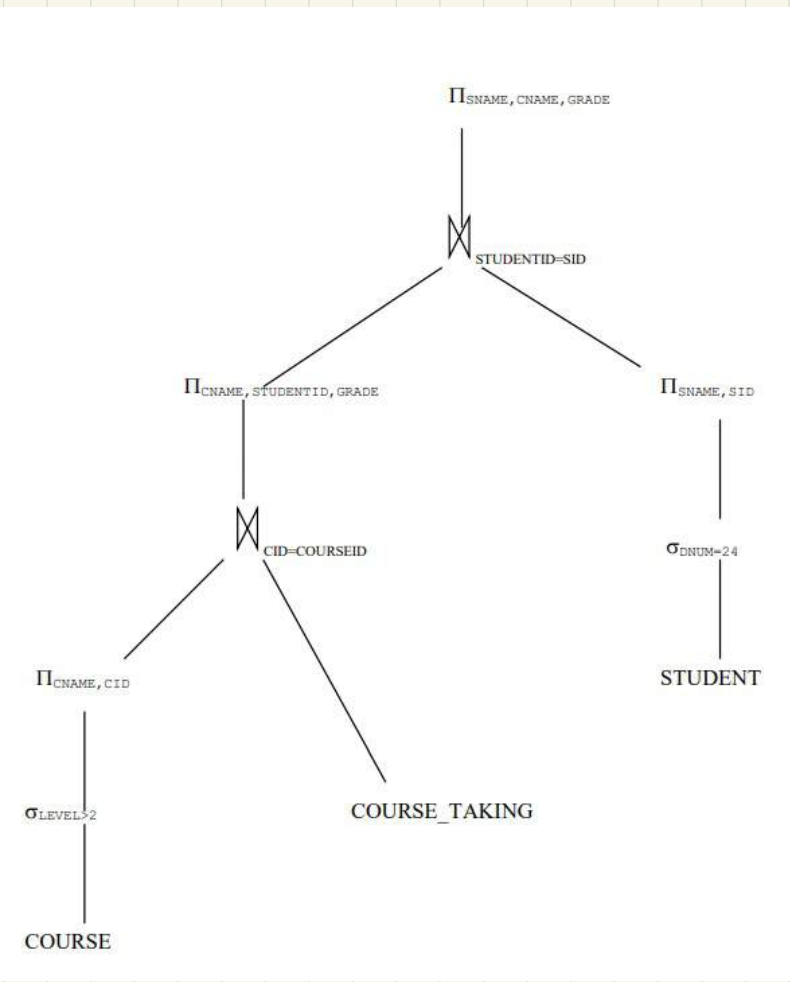


Figure 15.5
Steps in converting a query tree during heuristic optimization.
(a) Initial (canonical) query tree for SQL query Q.
(b) Moving SELECT operations down the query tree.
(c) Applying the more restrictive SELECT operation first.
(d) Replacing CARTESIAN PRODUCT and SELECT with JOIN operations.
(e) Moving PROJECT operations down the query tree.

Homework





Lecture 10-11 Transaction & Concurrency Control.

Transaction logical unit of database processing that include 1 or more access operation

boundary begin – end

simplified model

read_item(X) 1. find the address of disk block that contain X
2. copy disk block to buffer in memory
3. copy item X from buffer to program variable named X.

write_item(X) 1. same
2. same
3. copy X from program variable name X into correct location in the buffer.
4. store updated block from buffer back to disk

problem with transaction processing – concurrent execution is uncontrolled.
- lost update problem
- temporary update problem
- incorrect summary problem

Why Transaction may fail
- computer failure
- transaction or system error
- local error or exception condition detected by transaction
- concurrency control enforcement
- disk failure
- physical problem and catastrophe

Transaction States

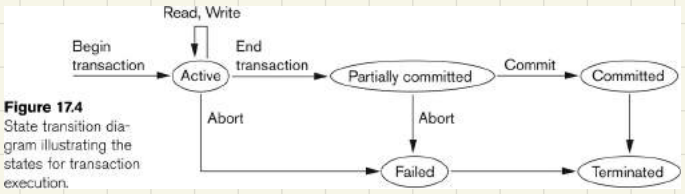


Figure 17.4
State transition diagram illustrating the states for transaction execution.

Operation
- begin transaction
- read / write
- end transaction
- commit transaction
- roll back

recovery technique

undo signal that operation end unsuccessfully operation must undone

redo certain transaction operation must redone to ensure all operations of transaction have been applied successfully to database

System log

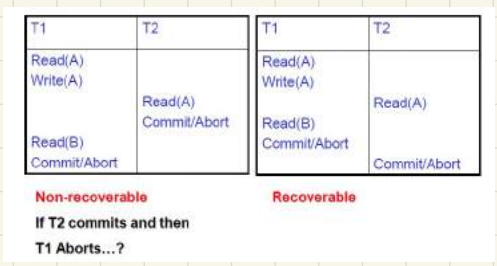
ACID properties

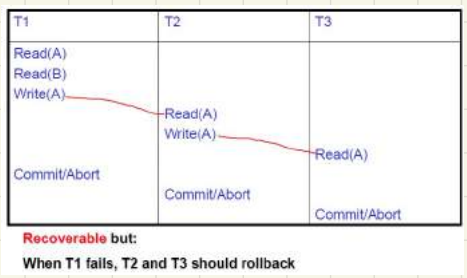
- **ACID**
  - atomicity（原子性）要么不做要么做完
  - consistency（一致性）独立执行事务可以保持数据库的一致性
  - isolation（隔离性）每个事务必须不知道其他并发执行的事务
  - durability（持久性）

transaction schedule interleaved fashion

recoverable schedule T_B write, T_A read (T_B written) $\Rightarrow$ T_B commit before T_A

cascade rollback: failure lead to series roll back





cascadeless schedule commited before being read

Strict schedule X read X write until last write operation T_n commite

Serial schedule — consecutively.

Serializable schedule — result equivalent, conflict equivalent — produce same final state, order of conflicting operation same

Isolation level

Concurrency - Two-phase locking techniques

lock(x) — secure permission to read / to write data item for transaction

unlock(x) — remove permission from data item

two lock mode
- shared lock(x) — read value ✕write
- exclusive(X) — write lock ✕ read

} no share can be applied in other T

wait for graph

two phase (mutually exclusive)
- locking (growing)
- unlocking (shrinking)

Deadlock prevention → lock all data before begin execution