C.J. Date Commandments for DBMS.

Page No.
Date

DATABASE MANAGEMENT SYSTEMS.

Credits: 07

- UML
  - General Electricals. ← James Rumbaugh.
  - Rational Software. ← Grady Booch.
  - U.K. - Objectory ← Ivan Jacobson.

- 1st system: P4004. → P type chip.

- FPS.
- DBMS.
  - → Hierarchical.
  - → Network.
  - → Relational.
- RDBMS.
- ORDBMS

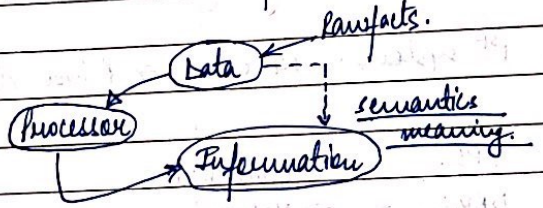
Rules.

→ E.F. Codd.

- Before database, the record keeping was done using Flat files which are managed using File Processing System (FPS).
- 1963 → Charles Bachman IMS — Information Management System.
- 1970 → EF Codd. ← Relational Model.
- DB2 → used on server & mainframes today.
- 1980 → Distributed DB.

UNIT – 1.

- **Data :–**
  These are raw facts which are processed to translate into information.



Ex: Data:

10 20 30 40.

Information:

	CP_1	CP_2	CP_3	CP_4.
S_1	10	20	30	40

- **Database:**
  Repository of data and metadata.
  data defining data (accounting data).
  for storage and retrieval in the most efficient way.

★★★ **Definitions:**

★ • Data represents raw facts that are recorded.

★ • Information represents processed data and may involve assigning meaning to data and establishing domain specification.

★ • **Database:**
  An integrated, self describing collection of data about the related set of thing (entities, objects) and the relationship / association among them.
  It concerns with information relevant to an enterprise. It is a repository of data and metadata.

- **DBMS: (Codd).**

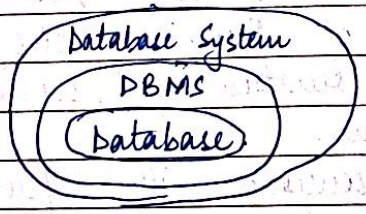
1. A DBMS is a set of inter-related data and a set of tools and programs to access and process these data. The primary goal of DBMS is to facilitate convenient and efficient mechanisms for storage and retrieval of database information.

2. Elmasri:
   Database management systems are packages used to develop, implement, manage and maintain databases.

RAM
64KB = 2^{16}

8085	8 bit data bus 16 bit Addr. bus.	Bus interleaving.
8086	16 bit data bus. 20 bit Addr. bus.	

3. It is a general purpose software system that facilitates defining, constructing, manipulating and sharing databases among various users and applications. It may include protecting the database and maintaining over a long period of time.



- ORDBMS — not fully Object Oriented neither fully Relational.
  Ex: NewSQL, NoSQL, etc.

- Types of Users:

1. Novice Users:
   Normal users who have no idea about internal functioning of the ~~users~~ System.

2. Parameterized Users:
   They have idea about the internal functionality and work with only some part of the database entities.

Ex: Cashiers / Tellers.

3. System Designers / Developers:
   These people are the ones who develop the database by considering the performance of the system to be offered with constraints.

4. DB Administrators:
   These people manage the database and adjust the constraints and maintain the system.

- Note:
  Data is stored once, never changes, Data can be represented in multiple ways using various indexing mechanisms.

- BST is a indexing structure where the indexes are sorted for efficient access time.

** FILE-PROCESSING SYSTEMS: (Data Processing (DP)).

1. Flat File:
   Flat files are general textual files storing data whose structure is known to the users in advance for storage and access.

OLTP – Online Transaction Processing uses DB and data is transient (vulnerable).

OLAP – Online Application Processing Data Warehouse & Mining uses stale, historical data.

- Different set of files required for every transaction / process.

- Problems:

1. Data Redundancy and inconsistency:
   - ↳ Diff. programmers create files & application program. (Isolation).
   - ↳ higher storage and access cost due to Data Redundancy. (DR) to access the right/correct copy.
   - ↳ Disagreement between copies of data (Data Inconsistency (DI)).

   Ex: Format conflicts like date causing wrong data interpretation.

2. Difficulty of Accessing Data:
   - ↳ Mechanism to retrieve data from file are time consuming and inefficient when different report on same data is required.

* FILE ← property of secondary storage
  - ↓ set of records (persistent / voluminous data)
    - → Variable length Records (VLRs)
    - → Fixed length Records (FLRs)
  - ↓ set of fields.
  - ↓ set of symbols.

Client / Slave — Server / Master model

2PC :- 2 Phase Commitment.

2PL :- 2 Phase locking.

3. Data Relation:

Data in different files with different formats necessitates writing new application programs.

4. Integrity Problems:
   - ↳ Data in our Database must satisfy some certain consistency constraints.
   - ↳ Developers embed constraints in various application programs.
   - ↳ Addition of new constraints requires changes to applications.

5. Atomicity Problems:
   - ↳ In many applications, if a failure occurs, the data should be restored to consistent state that existed prior to failure.
   - ↳ It is difficult to ensure atomicity in a conventional file processing system.

* Database
  - ↳ OLTP Application.

Query → Process information → transaction // read write cycle.
  - ↳ data is in transient state.

stable state ← → Consistent state

consistent

Abstraction: The level at which we are looking at the system (perspective). — views

ACID

6. Concurrent Access Anomalies:
↳ To guarantee overall performance and faster response, many systems allow multiple users to update the data simultaneously. (2PL).
↳ Supervision is difficult because data may be accessed by many different application programs that have not been co-ordinated previously.
• Dirty Read Problem:
↳ Evident where multiple copies of data exist.
↳ Occurs when Transaction has not locked and transaction are done based on previous state.

7. Security Problems:
↳ Not every user of database system be able to access all the data.
↳ File Processing systems through global/common data areas bypass this requirement.

★★★ CHARACTERISTICS OF DATABASE APPROACH:

1. Self-Describing Nature of Database System:
↳ A database system contains the database along with complete definition or description of its structure and constraints.
(data is not online but database is online).
↳ The data & catalog / dictionary contains information — structure of each file, the type and storage format of each data item, and various constraints on the data (the meta-data).

2. Insulation between Programs and Data, and Data Abstraction: focuses on the internal essential characteristics of data object.
↳ The structure of data files is stored in the DBMS catalog separately from the access programs — Program-Data Independance
↳ The implementation of the operation is specified separately and can be changed without affecting the interface.
↳ Guarantees Program-Operation Independance
(i) Program-Data Independance
(ii) Program-Operation Independance.

3. Support of Multiple views of Data:
↳ The characteristic allowing program-data independance and program-operation independence is called Data Abstraction.
↳ DBMS provides users with a conceptual representation of data without storage specification and implementation of operations.

↳ A data model is an abstraction providing conceptual representation. It uses logical concepts such as objects, their properties and their interrelationships, that may be easier for most users to understand than computer storage concepts.

↳ A data model hides storage and implementation details that are not of interest to most database users.

↳ A view may be a subset of database or it may contain virtual data that is derived from the database files but is not explicitly stored.

* Conceptual Model

ER Model ← Entity → Relational Model ← Class Diagram UML

Relational Model

snapshot → OLAP — view ← logical concept.
OLTP — materialized view ← physical concept.
stored DB.

DB Snapshots — State of the database at a particular time is stored.
— Data warehousing.
— Saves time for repeated queries.

A — Atomicity.
C — Consistency.
I — Isolation
D — Durability.

4. Sharing of Data and Multiuser Transaction Processing

↳ DBMS must include concurrency control software to allow several users to update (same) data in a controlled manner so that the result of the updates is correct (OLTP applications).

↳ Multiuser DBMS must guarantee concurrent transactions to operate correctly and efficiently.

↳ A transaction is an executing program or process that includes one or more database accesses. [read / write].

↳ Each transaction is supposed to execute a logically correct database access when executed in its entirety without interference from other transactions (ACID Properties).

** Instance:

↳ The collection of information stored in the database at a particular moment is called as instance of the database.

↳ The overall design of the database is called the database schema.

↳ Schemas are changed infrequently.

Ex: A(x, y, z) — schema — like a class.
a — instance — like an object.

RAID – Redundant Array of Independent Disks.
MTTF – Maximum / Minimum time to fail.

★ DATA MODELS:

1. Relational Model: (Conceptual Model:)

↳ It uses a collection of tables to represent both data and relationships among those data. Each table (relation) has multiple columns and each column has a unique name.
↳ It is a record-based model (fixed-format records).
↳ Most widely used data model, and a vast majority of current database systems are based on the relational model.

RAW ← text / image.
CLOB ← Character large object } for storing
BLOB ← Binary large object. } image and video.
VARCHAR2 ← 180 char max.

2. Entity – Relationship [ER] Model: (Logical Model)

↳ E-R data model uses a collection of basic objects (entities) and relationships among those objects.
↳ Widely used in database design.

	Ideal (Theory)	Practice.
OLTP →	1NF	
	2NF	
	3NF	3NF ✓
	BCNF	BCNF
	4NF (DKNF)	
	PJNF 5NF.	

OLAP	1NF	STAR
	2NF	SNOWFLAKE — 2NF.
	3NF	GALAXY.
	BCNF	

3. Object Based Data Model:

↳ Object Oriented programming (Java, C++, C#) has become the dominant software-development methodology.
↳ It extends the E-R model with notions of encapsulation, methods (functions), and object identity.
↳ ORDM combines the features of Object Oriented DBMS and RDBMS.

4. Semi-Structured Data Model:

↳ This model permits the specification of data where individual data items of the same type may have different sets of attributes.
↳ XML language adds markup information to text document and also has applications in data exchange.
↳ XML provides a way to represent data with nested structure and allows much flexibility in structuring of data which is important for certain kind of non-traditional data.

- **Optimizer:**
  It generates executable code that performs the necessary operations for the query and makes calls on the runtime processor.

- **Runtime Database Processor:**
  ↳ It executes (1). The privileged commands (2). The executable query plans, and (3) the canned transactions with runtime parameters. It works with the system catalog and may update it with statistics.
  ↳ It handles other aspects of data transfer, such as management of buffers in main memory.
  ↳ It integrates concurrency control & backup & recovery systems for purposes of transaction management.

Database Users:

1. **Database Administrators (DBA):**
   ↳ In a database environment the primary resource is the database itself, and the secondary resource is the DBMS and related software.
   ↳ DBA administers the resources.

2. **Database Designers:**
   ↳ They are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.
   ↳ They interact with each potential group of users and develop.

3. **End Users:**

   1. **Casual End Users:**
      They occasionally access the database, but they may need different information each time.

   2. **Naive or Parametric End Users:**
      Same kind of data accessed frequently. Ex: Teller / cashier of bank.

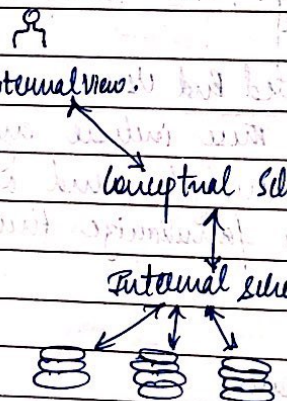
   3. **Sophisticated End Users:**
      These include engineers, scientists, business analysts and others who thoroughly familiarize themselves with the facilities.

   4. **Standalone:**
      Personal database maintenance by using ready made program packages.

4. System Analysts and Application Programmers:
↳ System analysts determine the requirements of end users, (naive & parametric end users) and develop specifications for standard canned transactions that meet these requirements.

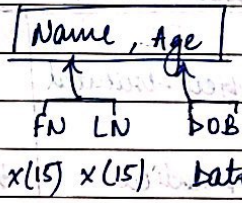
** ANSI/SPARC Architecture (3-Schema Architecture)
Standards Planning and Requirements Committee defined a framework for data modeling based on degrees of data abstraction.

Three-Schema Architecture:
↳ External
↳ Conceptual
↳ Internal.



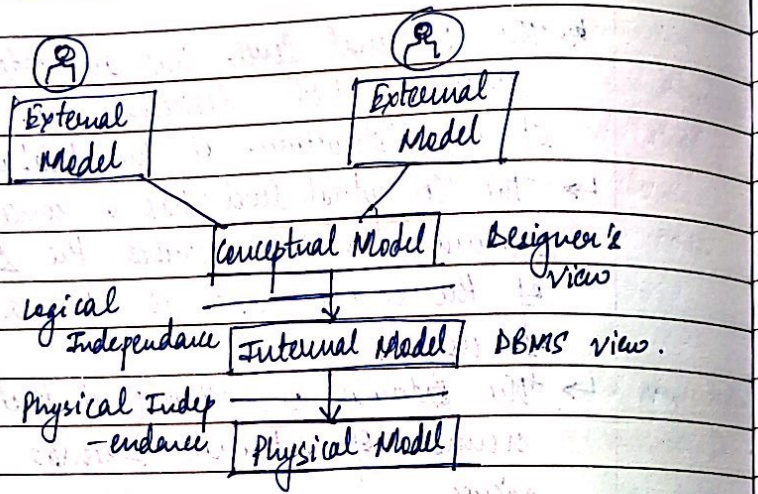
↳ The Internal level has an internal schema, which describes the physical storage structure of the database.
↳ The Conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users.
↳ The External or View level includes a number of external schemas or user views.

Ex:



- Three-Schema Architecture separates the users' external level, the database's conceptual level, and the internal storage level for designing a database. These schemas are only descriptions of data; the stored data that actually exists in the physical level only.

P.T.O.



Degree of Abstraction.		Characteristics.
High	ER.	Hardware-Independant
	Object-Oriented	Software-Independant
Medium	Relation.	Hardware-Independant
		Software-Dependant
Low	Network	Hardware-Dependant
	Hierarchical	Software-Dependant

External Model:

↳ The External Model is the end user's view of the data environment, who use the application programs to manipulate the data and generate information.

↳ Advantages:

1. Allows for identification of data required to support each business unit's operations.
2. It makes the designer's job easy by providing feedback about the model's adequacy.
3. It helps to ensure security constraints in the database design.

CONCEPTUAL MODEL:

↳ The Conceptual Model represents a global view of the entire database as viewed by the entire organization. It integrates all external views into a single global view of the data in the enterprise.

↳ The most widely used conceptual model is the ER model. The ERD is used to graphically represent the conceptual schema and is the basic database input.

↳ It is independent of both Hardware & Software.

INTERNAL MODEL:

↳ Once a specific DBMS has been selected, the internal model maps the conceptual model to the DBMS. The internal model requires the designer to match the conceptual model's characteristics and constraints to those of the selected implement

-ation model.

** ↳ The development of a detailed internal model is especially important to database designers who work with hierarchical or network models because these models require very precise specification of data storage locations and data access paths.

↳ software dependant but hardware Independant.

** PHYSICAL MODEL:

↳ The Physical Model operates at the lowest level of abstraction, describing the way data are saved on storage media such as disks or tapes.

↳ It requires the definition of both the physical storage devices and the access methods required to reach the data within those storage devices making it both software & Hardware dependant.

• Comparison of Different Models:

* Data Independance:

1. Logical Data Independance:

↳ It is the capacity to change the conceptual schema without having to change external schemas or application programs.

↳ Changes to constraints can be applied to the ↳ conceptual schema without affecting the external

2. ~~Physical Data Indep~~ Physical schemas or application programs

2. Physical Data Independance:

↳ It is the capacity to change the internal schema without having to change the conceptual schema.

↳

** DBMS INTERFACES:

1. Menu-Based Interfaces for Web Clients or Browsing.
2. Forms-Based Interfaces.
3. Graphical User Interfaces.
4. Natural Language Interfaces.
5. Speech Input and Output.
6. Interfaces for Parameteric Users. — Minimizing the number of keystrokes
7. Interfaces for the DBA. — Priviledged Commands.
   → Creating Accounts, Setting System Parameters

*** QUERY LANGUAGE:

It is a high-level programming language in which a user requests information from the database. They are categorized as:

I) Procedural language:

The user instructs the system

4GL – Generation Language.

to perform a sequence of operations on the database to compute the desired result.

Embedded Scripts ← Host 3GLs | 2GLs || (What / How)

II) Non-Procedural Languages:

It allows user to describe the desired information without giving specific procedure for obtaining the information.

What ← 4GL.

Ex: SEQUEL – (SQL)
└ Non-Procedural Language.

Query language → Tuple Relational Calculus (TRC)
- Domain Relation Calculus (DRC)
→ Relational Algebra (RA).
(Set Theory)

SQL is based on Relational Algebra.

Relational Algebra:

Basic set of operations for the relational model enables a user to specify basic retrieval requests as relational algebra expressions.

The operations may take one or more relations as operands and produce a result

Notequal to → (!=) × (<> or ≠) ✓

Division → (÷) or (/)

which is always a new relation.

- Set Operations:
  Union, Intersection, Set difference, Cartesian Product.
- Specialized Relational Operations.
  Select, Project, Join, Rename.
- Extended Relational Algebra Operations:
  Groupby, Aggregate Functions (SUM, AVG, MIN, MA
- Relational Expressions include:
  Comparison operators: >, <, =, ≠, ≥ and ≤.
  Logical: ∧ (And), ∨, ¬
  Arithmetic: +, −, ×, ÷.

1. Selection Operation (σ_F): → SL.

Predicate. = Condition

R(A, B, C)

	A	B	C
t_1	10	A	tt
t_2	20	B	nn
t_3	30	A	tt
t_4	40	X	cc
t_5	50	C	xx
t_6	60	Z	yy

Relation = R.
Cardinality = 6 — No. of tuples at that instant.
Degree = 3. — No. of Column.

- Select from R where column C is 'tt'.

$\sigma_{R.C = 'tt'}(R)$

R.A	R.B	R.C
10	A	tt
30	A	tt.

Horizontal partition. (Reduces).

- A Relation [R(A,B,C)] is itself a set on which a predicate holds (a trivial default super key conformance).

- To display full relation $\sigma_{1=1}(R)$.
  a tautology, some predicate which is always true.

- $|\sigma_F(R)|$ = cardinality
  $|\sigma_F(R)| \leq |R|$. — cardinality of whole relation.

Degree := Degree (R).

Domain (R(A)) = 10, 20, 30, 40, 50, 60.

- Selection Operation: affects the cardinality and the domain values, but it doesn't affect degree of the relation.

Composite Primary Key is a combination of Primary Keys.

- The fraction of tuple selected by a selection is called as Selectivity.

- Selection Operation is Commutative:

Ex: $\sigma_{F1}(\sigma_{F2}(R))$

$\Rightarrow \sigma_{C='tt'}(\sigma_{A=10}(R)) \cong \sigma_{A=10}(\sigma_{C='tt'}(R))$.

- Find all sailors having age above 45 years.

$\sigma_{age>45}(sailors)$

- Schemata:

Sailors(sid, sname, rating, age) := dom(age) = real
Boats(bid, bname, color) := dom(color) = string
Reserve(sid, bid, day) := dom(day) = date.

- Find all sailors having age >45 and rating of 7.

$\sigma_{age>45 \wedge rating=7}(sailors)$.

2. PROJECTION OPERATOR $\Pi_A(R)$ attr(A) $\subseteq$ attr(R).
   Attribute set.

degree($\Pi_A(R)$) $\leq$ degree(R) — Vertical partition.

Output set (relation) of $\Pi_A(R)$ will eliminate duplicates.

Ex: $\Pi_x(\sigma_{y=3}(R))$.
Select tuples with 'y=3' & project only 'x' column.

Cascade of Projection is non-commutative.

Ex: 1. $\Pi_x(\Pi_{y,z}(\sigma_{y=3}(R)))$ — ✗. (not possible. ✗ doesn't exist after this.)

2. $\Pi_x(\Pi_{x,z}(\sigma_{y=3}(R)))$ — ✓

3. $\Pi_{x,y,z}(\Pi_x(\sigma_{y=3}(R)))$ — ✗.
   - after this only one column is left.
   - You cannot extract 3 columns.

(3) $A_1 \subseteq A$ $A = R$.

$A_2 \subseteq A$ $A_1 = X$.

$A_1 \subseteq A_2$ $A_2 = x, y, z$

This doesn't hold true, ∴ (3) is not valid.

Q) Find age of all sailors:

→ $\Pi_{age}(sailors)$

Ex: Sailor.

SID	Sname	Rating	Age.
10	Abc	7	24
20	Xyz	8	26
30	Tuy	6	24
40	Nut	9	25.

$\Pi_{age}(Sailors)$ O/P ⇒

Sailor.Age
24
26
25.

(Projection eliminates duplicates)

SQL: SELECT DISTINCT AGE FROM SAILORS.

Needed when you don't include a key as a part of projection. → important (ordering of o/p may change as it is a set).

Q) List for all sailors — the sid, name, age.

→ $\Pi_{sid, name, age}(sailors)$

SQL> SELECT SID, NAME, AGE FROM SAILORS;

→ SID is unique ⇒ DISTINCT Not Required.

Q). Find names of all sailors aged between 40 & 50

→ $\Pi_{name}(\sigma_{age \geq 40 \wedge age \leq 50}(Sailors))$.

SQL> SELECT DISTINCT SNAME FROM SAILORS
WHERE AGE >= 40 AND AGE <= 50;

Q) List all sailors who have reserved a boat.

→ $\Pi_{sid, bid}(Reserves)$

SQL> SELECT DISTINCT sid, bid FROM Reserves.

'day' is also primary key in reserves.

∴ sid + bid may be duplicate.

8. RENAME OPERATOR: (ρ):

- Renames either relation name, attribute name, or both.
- a unary operator.
- In SQL it is called "ALIAS".

Ex: SELECT AGE "How Old?" FROM SAILORS;
└ This only changes the Header name while displaying & not the attribute name.

Q) Create an instance Sailor named as Riders.
→ ρ_{Riders} (sailors) — renaming relation
└ Select Subquery.

SQL> CREATE TABLE Riders AS SELECT * FROM Sailors;
└ Subquery — Nested query

Q) Create an instance of Sailor as Rider with rid, rname, rating, age.
→ $\rho_{Riders(rid, rname, rating, age)}$ (Sailors).

SQL> CREATE TABLE Riders AS SELECT sid as rid, sname as rname, rating, age FROM Sailors;
└ attribute aliases.

Q) Create an instance of Sailor with attributes renamed as rid, rname, rating, age.
→ $\rho_{(rid, rname, rating, age)}$ (Sailors).

SQL> SELECT sid as rid, sname as rname, rating, age FROM SAILORS;
└ Temporary Relation.

Q) Find the highest age among all Sailors.
→ ① ④ ③
$\pi_{age}(Sailors) - \pi_{Sailors.age}(\sigma_{Sailors.age < Sx.age}$ (Sailors × ② ρ_{Sx} (Sailors)))

① age	② Sailors.age	Sx.age	③ 24,25	④ 24,25
26	24	24	24,26	└ duplicates eliminated
24	24	25	25,26	
25	24	26	+ duplicates	
	25	24		
	25	25		
	25	26		
	26	24		
	26	25		
	26	26.		

+ duplicates
(24,25,26) − (24,25) = (26) ✓

★ REDUCERS: < Distributed, Query Mgmt.
→ σ
→ π
→ ρ
} apply on R', outputs a 'R'.

card(R) ≥ card(σ_F(R)).

$\sigma_F(R)$ → selectivity over tuples } duplicates storage based on F.

$\pi_A(R)$ → a snapshot of columns over R. duplicates are eliminated.

Ex: $\pi_A(\sigma_{x=4}(R))$ → select * from R where x=4;

$\pi_{x,y}$ ↓ specific projections.

R'

x	Y	Z
4	7	8
4	8	2
4	7	3

x=4

Page No.
Date

Highly Expensive Operations ★

BINARY OPERATIONS: [UNION, INTERSECTION, MINUS]

↳ these binary operations are applied to two relations, which must have the same type of tuples — union compatibility or type compatibility.

↳ Union — compatibility.

type.

↳ arity same.

↳ domain of Ai [L→R] similar

↳ domain list of all plausible values that attribute can take.

non inclusive | inclusive

[10 , 50]

1. UNION:

↳ The union is taken between the union — compatible relations.

↳ duplicates tuples are eliminated.

R ∪ S ⇒ eliminate duplicates.

UNION ✓ eliminated duplicates.

UNION ALL ✗ doesn't eliminate duplicates.

R ∪ (S ∪ T) ≅ (R ∪ S) ∪ T ✓

2. DIFFERENCE:

↳ Taken between Union compatible relations.

Page No.
Date

↳ Returns tuple that occurs in R but not in S.

↳ Duplicates eliminated.

(R−S) − T ≠ R − (S−T).

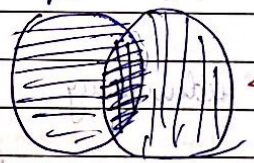
3. INTERSECTION:

↳ Taken b/w union — compatible relations.

R ∩ S = ((R ∪ S) − (R−S)) − (S−R).

⇒ R − (R−S)

⇒ S − (S−R).



database — R

database instance — r.

Ex: 1. Find All sailors:

not required as we want all tuples.

Instance 1 | $\Pi_{sname}(\sigma_{1=1}$ sailors1) ∪ $\Pi_{sname}(\sigma_{1=1}$ sailors2)

Instance 2.

O/P: Dustin ∪ Yodel ⇒ Dustin Yodel

Lubber. Lubber Lubber Gusty

Rusty. Gusty Rusty

Rusty

select * from A
where 'a' = 'a'; injection which is always true.

Joins are most expensive ☹

SELECT Sname from Sailors1;
UNION
SELECT Sname from Sailors2;

2. Find all sailors that exists in both sailors relation.

Soln: $\Pi_{Sname}(Sailors1) \cap \Pi_{Sname}(Sailors2)$.

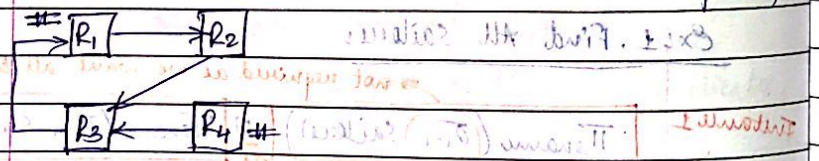
3. Find sailors who exists only in Sailors1.

Soln: $\Pi_{Sname}(Sailors1) - \Pi_{Sname}(Sailors2)$.

4. Find sailors who exists only in Sailors2.

Soln: $\Pi_{Sname}(Sailors2) - \Pi_{Sname}(Sailors1)$

~~eliminates~~ data redundancy
minimizes
↳ introduces controlled redundancy mechanism (PK, FK)
⇩
interconnected data (FPS := Islands of data).



Join → involve of Cartesian Product of operand relations.

Cartesian Product (Cross Join)

R×S ⇒ no. of tuples in R×S.
equals no. of tuples in R * no. of tuples in S
⇒ duplicates not removed. to remove them use 'Projection'.

degree (R×S) = degree(R) + degree(S).
(no. of attributes or columns of table).

Example of Cartesian Product:

R(A, B, C)

A	B	C
5	7	9
2	3	4
1	2	4

S(X, Y, Z)

X	Y	Z
2	3	3
4	3	2

R×S. ← basic operation for implementing 'Join'.

	R.A	R.B	R.C	S.X	S.Y	S.Z
✗	5	7	9	2	3	3
✗	5	7	9	4	3	2
✓	2	3	4	2	3	3
✗	2	3	4	4	3	2
✗	1	2	4	2	3	3
✗	1	2	4	4	3	2

★ Binary Join Operation: (⋈) or Natural Join:

R ⋈ S ⇒ Lists the tuples from R×S where
R.A = S.A ∧ R.B = S.B ...

★ INNER JOINS: — based on selectivity 'in' both tables.

1. Natural Join (No Predicate listed) → conditions.
2. Equi Joins. $\bowtie_F$ ⇒ Predicate can only contain '=' comparisons.
3. Theta Joins $\bowtie_\theta$ ⇒ Predicate can contain atleast one non-equality comparison.

★ Outer Join:

1. Left Outer Join — $\bowtie_F$:

From previous example,

O/P:

S	T	9	ϕ	ϕ	ϕ
2	3	4	2	3	3
1	2	4	ϕ	ϕ	ϕ

2. Right Outer Join — $\bowtie_F$:

O/P: ϕ ϕ ϕ ; 2 3 4 ; 2 3 3 ; ϕ ϕ ϕ ; ~~4 3 2~~ ; 4 3 2 ← 4 Tuples.

3. Full Outer Join — $\bowtie_F$:

$R \bowtie_F S \cong \sigma_F (R \times S)$.

↓

where Predicate is applied at the time of Join.

Omit

• Idempotance of Proj Operations:

Ex:

$\Pi_{R.A, S.Z} (\sigma_{A=A} (R \times S))$

$\Pi_{R.A, S.Z} (\sigma_{A=A} (\Pi_A(R) \times \Pi_{A,Z}(S)))$

$\Pi_{R.A, S.Z} (\sigma_{A=A} (\Pi_A R) \times \sigma_A (\Pi_{A,Z} S))$

Ex: $\sigma_{A=A} (R \times S)$ (Inner Join)

$\cong R \bowtie_{A=A} S$

Legacy Style: Select * from R,S where R.A = S.A;

ANSI SQL 1999: select * from R INNER JOIN S ON R.A = S.A; / using A;

• Left Outer Join:

$(R \bowtie_{A=A} S)$

Legacy Style: Select * from R,S where R.A = S.A (+);

ANSI SQL 1999: Select * from Sailors1 LEFT OUTER JOIN Sailors2 On Sailors1.sid = Sailors2.sid;

count (A) — domain value of A.

count (*) — All.

count (distinct A) — eliminate duplicates if any.

Sailors (sid, sname, age, rating)
Reserves (sid, bid, day)
Boat (bid, bname, colour)

— or * ⇒ Primary key.

★ Queries:

1. Find names of all sailors who have reserved boat 103.

Soln:

$\Pi_{sname}\left(\sigma_{bid=103}\left(\text{Sailors} \bowtie \text{Reserves}\right)\right)$

→ Returns 4 columns out of which only 1 is projected

Natural inner join as no predicate specified.

$\Pi_{sname}\left(\sigma_{bid=103}\left(\Pi_{sid,sname}\ \text{Sailors} \bowtie \Pi_{sid,bid}\ \text{Reserves}\right)\right)$

SELECT in Oracle doesn't implicitly eliminate duplicate values, hence used 'DISTINCT' Keyword.

```
SELECT DISTINCT SNAME, S.SID
  FROM SAILORS S JOIN RESERVES R
  ON S.SID = R.SID AND R.BID = 103;

SELECT DISTINCT SNAME, S.SID
  FROM SAILORS S JOIN (
    SELECT * FROM RESERVES R
      WHERE R.BID = 103) R.
  ON S.SID = R.SID;
```

↳ $\Pi_{sname}\left(\text{Sailors} \bowtie \left(\sigma_{bid=103}\ \text{Reserves}\right)\right)$.

5. Find the names of sailors who have reserved either a red or green boat.

Soln:

$\S(\text{Tempboats}, (\sigma_{colour='red'}\ \text{Boats}) \cup (\sigma_{colour='green'}\ \text{Boats}))$

$\Pi_{sname}(\text{TempBoats} \bowtie \text{Reserves} \bowtie \text{Sailors})$

Only works for OR, not AND because two values cannot be associated in a tuple.

$\S(\text{Tempboats}, (\sigma_{colour='red'} \vee_{colour='green'}\ \text{Boats}))$

$\Pi_{sname}(\text{Tempboats} \bowtie \text{Reserves} \bowtie \text{Sailors})$.

7. Find the names of the sailors who have reserved at least two boats.

Soln:

$\S(\text{Reservations}, \Pi_{sid,sname,bid}(\text{Sailors} \bowtie \text{Reserves}))$

$\S(\text{Reservationpairs}(1 \to sid1, 2 \to sname1, 3 \to bid1, 4 \to sid2, 5 \to sname2, 6 \to bid2), \text{Reservations} \times \text{Reservations})$

$\Pi_{sname1}\ \sigma_{(sid1=sid2) \wedge (bid1 \neq bid2)}\ \text{Reservationpairs}$.

```
SELECT DISTINCT RP1.SNAME FROM
  (SELECT S.SID, SNAME, R.BID
    FROM SAILORS S JOIN RESERVES R
      ON S.SID = R.SID) RP1
  JOIN
  (Select S.sid, SName, R.Bid
    FROM SAILORS S JOIN RESERVES R
      ON S.SID = R.SID) RP2
  ON RP1.SID = RP2.SID AND RP1.BID <> RP2.BID.
```

UNIT – 2

Conceptual Design + Logical Design.

* Design of Database Schema:

1. CONCEPT OF KEY & NORMALIZATION:

↳ Relational Model. → Relation
- arity
- cardinality
- set of rows & columns
- each tuple is exhibit EI (KEY).

EI = Entity Integrity

Relation Schema:

R (A, B, C, D, E)
- attributes, domains.
- Relation Name
- Constraints on R.
- FDs – Functional Dependancies (Special ICs)

→ determinism.

KEY ← set of attribute(s) uniquely identifying each tuple in a relation.
- ↳ singular
- ↳ composite.

Integrity Constraints.

Ex:

instance :- of a relation

R

A	B	C	D	E
10	20	A	X	T
20	20	A	Y	W
30	10	B	X	T
40	30	B	Y	W

* Primary Key is applied on the physical db & not on the instance alone. so think all possible values an attribute can have.

Define Primary Key Accordingly.

A · determines B, C, D, E · (reachability of Attribute A: discriminating attribute A in set of FDs → read as A determines on R).

A → BCDE implies

A → B	A → D
A → C	A → E

B, C, D, E.

A → A — Reflexity holds true.

⇒ A → ABCDE → R.

⇒ A → R.

A becomes a key of the relation.

(KEY = set of attribute(s) uniquely identifying each tuple in a relation).

There may be multiple keys possible in a relation

Super Key = The superset of all keys.

X → Y

X: set of discriminators → Y: set of dependants.

t1.X → t1.Y } Always ensured.
t2.X → t2.Y } t = tuple.

If a discriminator value is repeated in table, then it ensures that its dependants will be the same.

→ The NOT NULL Constraint is enforced on key to make it PRIMARY KEY → Integrity Constraints

Candidate Key may have NULL

• Example:

Student (Roll, Enroll, FName, LName, DOB)

unique unique non-unique non-unique.

Super Key: = { Roll, Enroll, Roll+Enroll, Roll+FName, Enroll+DOB, ... }.

↓ non-redundant set of SKs. (Minimal SK).

~~Roll+Enroll~~

~~Enroll+DOB~~

✓ Enroll

✓ Roll

~~Roll+FName~~

Candidate Key: Non Redundant set of attributes that uniquely identify a tuple is Candidate Key.

They can be composite too but should be minimal.

CK = { Enroll, Roll }.

DBA's Key → PK = { Roll } — Primary Key.

The Candidate Key chosen by the DBA is called Primary Key. This key is used to implement the database.

ROLL := PK — Automatically Not Null.

Enroll := UQ + NN.

Problems of DB:

Spurious tuple
Data Anomaly ← insert update delete.
Not Null

Now Roll & Enroll are equally powerful.

UQ = Unique Constraint.

★ Goodness Measure for Schema:

↳ Relation Schema attributes + domains are defined

↳ The dependancies.

↳ Trivial integration of attribute to schema.

• Motives: ↓ Conditions to be guaranteed while decomposition.

1. Minimize Redundancy.
2. Information Preservation (Controlled Redundancy).

↳ same no. of tuples, same set of tuples, w/o extra tuples (SPURIOUS tuples solved)

Decomposition of Schema is the underlying activity.

SReg (Roll, FN, LN, Dept, DCode, YOS).

decomposition SReg.

Required to guarantee Information Preservation

Student ⋈ Dept.

↳ Roll, FN, LN, YOS, DCode. ↳ Dept, Dept Code.

redundant but necessary.

• Is Decomposition Essential?

Don't do composition - if the information required

Refer Elmasri for Normalization based on Primary Key & Candidate Key.

WAL — Write Ahead Log.

is very often required / queried which requires joining decomposed relations.

★ DB Design:
↳ Conceptual / Logical DB design. (applies to Base Table + Views).
↳ attribute + domains.
↳ Physical / Implementation DB design (Base tables in relational schema)
↳ storage of data / attribute.

• Approaches of DB design:
1. Bottom-Up (Design-by-Synthesis) — Integrative,
↳ The basic relationship between attributes.

2. Top-Down (Design-by-Analysis) — Divisive,
↳ The basic grouping of attribute in relation.

★★ NORMAL FORMS:
→ All set of FDs guaranteed & conforms to Integrity constraints.

• Integrity Constraints:
Primary Key, Foreign Key, Check, FDs.
Assertions, Triggers, stored programs procedures,
FD — Functional Dependency.

• Data Warehouses are built on used on OLAP (historical / stale) data.

Transient data — data changeable in time — it is volatile.
Transient data = in time. OLAP — Pre-Computed Queries.
OLTP — Dirty, Hit & go / fly Queries.

★ Decompose Relations: — Reduce Redundancy
— Information Preservation

• Problems with decomposition of database Relation schema:

DB Relation.
Decomposition of Schema ← Normalization.
↳ eliminates certain anamolies.
modifications ↳ update } groups of
↳ insert } related info.
↳ delete
↳ dependencies leading to redundancy.

discriminating attribute (KEY) — multivalued attribute — atomic domain — restricted / managed.

Multivalued dependency. | Projection dependencies | Partial dependencies | Transitive dependencies | Rev-Transitive dependencies.

RR. DATA ANAMOLIES:

Ex: EMP_PROJ(EMP#, PROJ#, PName, EName, No_Hrs).

CK = {(Emp#, Proj#)}. = Primary Key.

Cardinality (Emp) = 1000 ∴ 1000/10 = 100
Domain (Proj) = 10. projects for an Employee.
Proj#: P1 → "Billing"
update to, "Cust-Accounting"

Scanned with CamScanner

■ Update Anamoly:
↳ Changing the name of project P1 from 'Apple' to 'Orange' will cause to update all 100 employees with this. If even one is left, it will create anamoly.

■ Insert Anamoly:
↳ Cannot insert a project unless an employee is assigned to it.
∵ PK = (Emp#, Proj#) ← Unique, Not Null

Conversely:
↳ Cannot insert an employee unless he/she is assigned to a project.

■ Delete Anamoly:
In Oracle — ONDELETE
✓①. CASCADE :- Parent killed, all children killed.
✓② NO ACTION
✓③. SET NULL
→ X ④. SET DEFAULT.

↳ When a project is deleted, it will result in deleting all the employees who work on the project.
↳ Alternatively, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project

RR ★ NULL Value in Tuples:
↳ Relations should be designed such that their tuples will have as few NULL values as possible.
↳ Attributes that are NULL frequently could be in separate relation (with the Primary Key).

IP ★ SPURIOUS TUPLES:
↳ Bad designs for a relational database may result in erroneous results for certain JOIN operations.
↳ The "lossless join" property is used to guarantee meaningful results for join operations.

★ Two important properties of decompositions:
a) Non-additive or lossiness of the corresponding join
b). Preservation of the functional dependencies.

(a) – Cannot be sacrificed; extremely important.
(b) – less stringent and may be sacrificed.

★ **Normalization:**

↳ The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations.

- **Normal Forms & FDs:**
  - ∈ Repetitive group of info.
    - ↳ updation Anomalies.
  - ∈ NULL Value (OLTP)
  - ∈ Spurious (Phantom) Tuples ∴ Bad Decomposition

  ∴ Normalization.

★ **Normal Form:**

↳ Condition using Keys and FDs of a relation to certify whether a relation schema is in a particular normal form.

- **Denormalized Relation:** Student 20 → 4 Proj

  Student Project

  (SName SRoll SDOB | PName PID | NoHoursWorked)

  repetitive group of attributes.

SPhone
M | L | A
✓ ϕ ϕ

  Multivalued

  Multivalued Attributes.

  ← Problems.

- **Normal Forms & FDs: (Keys & FDs).** (PK, CK)
  - ↳ 1NF
  - ↳ 2NF
  - ↳ 3NF
  - ↳ BCNF

  } Key based — PK | CK.

  - ↳ 4NF
  - ↳ 5NF

  } FDs. Multi-Valued Dependancies (MVDs)
  Project-Join Dependancy (PJDs)

RDBMS

↳ Relation Schema ← Relation

↳ always ensure non-duplicacy of tuple.

★ **1NF:**

↳ A Relation Schema 'R' is in First Normal form (1NF) if the values in of Domain(A) are atomic for every attribute A in R.

1NF:

↳ no multivalued attribute.

↳ includes a KEY (PK | CK).

↳ Disallows:

- Composite Attributes.
- Set-Valued Attributes.
- Nested Relations: a cell of an individual tuple is a complex relation.

P.T.O.

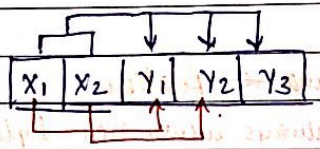
★ 2NF:

↳ Uses the concept of FDs, primary key.
↳ Must be in 1NF; (PK|CK).
↳ Must eliminate Partial Dependancies.

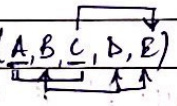
Ex:

$\{X\} \rightarrow \{Y\}$.

$\{X_1, X_2\} \rightarrow \{Y_1, Y_2, Y_3\}$ where $X_1 \xrightarrow{\text{uniquely}} Y_1$, $X_2 \rightarrow Y_3$



Ex: $R(A, B, C, D, E)$



A	B	C	D	E
1	T	A	X	X
1	W	B	Z	Y
2	W	B	T	Y
2	P	A	K	X

PK $\{A, C\}$

$\{A, C\} \rightarrow \{B, D, E\} \rightarrow R$.

$\{C\} \rightarrow \{E\}$.

$\{C\} \subseteq$ KEY $\Rightarrow$ Partial Dependency

Decompose

R_1.

A	C	B	D
1	A	T	X
1	B	W	Z
2	B	W	T
2	A	P	K

R_2

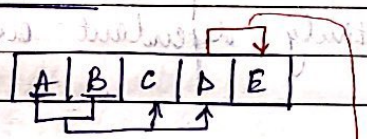
C	E
A	X
B	Y

★ 2NF Definition:

↳ A relation schema R is in second normal form (2NF) with respect to a set of FDs F if it is in 1NF and every nonprime attribute is fully dependent on every key of R.

★ 3NF:

R



1NF ✓

2NF ✓ No. PD.

★ Transitive dependency:

$X \rightarrow Y$. in R.

where both ~~to~~ X & Y are non-prime attributes.

• Definition:

Given a relation schema R, a subset X of R, an attribute A in R and a set of FDs F, A is transitively dependent upon X in R if there is a subset Y of R with:

$X \rightarrow Y$, $Y \not\rightarrow X$, and $Y \rightarrow A$ under F and $A \notin XY$.

R_1 | A | B | C | D |

R_2 | D | E | (3NF)

- **Definition:**
  A Relation schema R is in 3NF if it is in 2NF and no-non prime attribute A in R is transitively dependent on the primary key.

Ex:1 Given a Relation 'R' = {A, B, C, D, E, H} and having the following FDs:

$G = \{A \to BC,\ CD \to E,\ E \to C,\ D \to AEH,\ ABH \to BD,\ DH \to BC\}$.

Find the Key for Relation R with FDs in 'G'.

Ex: 2. $R = \{A, B, C, D, E, F\}$

FDs $\Rightarrow X := \{AB \to C,\ DC \to AE,\ E \to F\}$.

Find the Key of R.

$\Rightarrow$ **Attribute closure Algo: (A^+):**

1. First left FD:

$\{AB\}^+ = \{A, B\}$ → Reflexive property

$= \{A, B, C\}$ by FD1

$\neq R$.

$\{DC\}^+ = \{D, C\}$.

$= \{A, C, D, E\}$ — FD2.

$= \{A, C, D, E, F\}$ — FD3 $\neq R$.

$\{E\}^+ = \{E\}$

$= \{E, F\}$ — FD3 $\neq R$.

None of them is a primary key, therefore adding attribute,

$\{ABD\}^+ = \{A, B, D\}$

$= \{A, B, C, D\}$ — FD1.

$= \{A, B, C, D, E\}$ — FD2.

$= \{A, B, C, D, E, F\}$ — FD3.

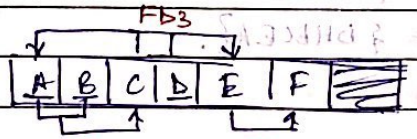
$\Rightarrow R$.

$\{BCD\}^+ = \{B, C, D\}$.

$= \{B, C, D, E, A\}$ — FD2.

$= \{B, C, D, E, F, A\}$ — FD3.

$\Rightarrow R$.



1 NF ✓

2NF ✗

3NF ✗

Ex:- $\{A\}^+ \rightarrow \{A\}$

$\rightarrow \{A,B,C\}$ — FD1.

$\neq R$.

$\{CD\}^+ \rightarrow \{CD\}$

$\rightarrow \{CDE\}$ — FD2.

$\rightarrow \{CDEAH\}$ — FD4

$\rightarrow \{CDEAHB\}$ — FD6.

$= R$

$\{E\}^+ \rightarrow \{E\}$

$\neq R$.

$\{D\}^+ \rightarrow \{D\}$

$\rightarrow \{AEHD\}$.

$\rightarrow \{AEHDBC\}$.

$= R$.

$\{ABH\}^+ = \{ABH\}$

$= \{ABCHD\}$ — FD1 FD5

$= \{ABCDEH\}$. — FD4.

$= R$.

$\{DH\}^+ = \{DH\}$

$= \{DHBC\}$

$= \{DHBCEA\}$.

- Primary key is one of the candidate keys.
- Prime attributes are inclusive in candidate keys.

- ELMASRI BOOK → Primary key.
  → General Forms (candidate key)

Superkey: A superkey of a relation schema $R = \{A1, A2, \ldots, An\}$ is an set of attributes S subset of R with the property that no two two tuples t1 and t2 in any legal relation state r of R will have $t1[S] = t2[S]$.

R(A, B, C, D, E)

S (A, B)

S → (R−S)

AB → CDE

t1 → 11 → 456

t2 → 12 → 465

21 → 456.

domain super key.

Superkey → key

S_{AB}. non-redundant Ai

if we remove B from key,

A → R doesn't hold.

DNum		Dloc
11	— n —	loc1
11	— n —	loc2
11	— n —	loc3.

Dloc1	Dloc2	Dloc3.
✓	ϕ	ϕ
✓	✓	✓
✓	✓	ϕ

Whole Key = Full Functional Dependency

PROJECT composite entity.

- EMP_PROJ (SSN, EName, PNum, Hours)

SSN	PNum	Hours

SSN	ENAme

After decomposing.

- Trivial Dependencies:
  Simply Reflexive non-needed dependencies
  $A \to A$, $B \to B$, $AB \to B$ - - - and so on.

- Prime Attribute:
  An attribute that is member of the primary key.

- Left Reduced or Full Functional Dependancy:

$$\frac{A \to R}{A_1 A_2}$$

A_2 is redundant.

$\therefore A_1 \to R$.

★ • INF. → Key should be determining the tuples &
  No multivalued or complex value.

Ex: Activity → Fee,
Sid Activity → Instructor

× Partial dependency.

R (Sid, Activity, Fee, Instructor)

Key ← (Sid, Activity).

Decompose Relation:

R1:

Sid	Activity	Instructor

R2:

Activity	Fee.

Ex:

	SSN	PNumber	Hours	EName	PName	Plocation
FD1						
FD2						
FD3						

FD1. Key (SSN, PNumber) where the whole key determines Hours ✓ (SSN, PNum) → Hours

FD2. SSN → EName. × Partial Dependancy.

FD3 PNumber → PName, Plocation. × Partial Dependency

SSN	PNumber	Hours

SSN	EName

PNumber	PName	Plocation.

$S \rightarrow E$ 1NF, 2NF, 3NF.

$P \rightarrow N, L$ 1NF, 2NF, 3NF

$SP \rightarrow H$ 1NF, 2NF

Left Reduced as we cannot minimize further.

- **3NF:**

$R(A, B, C, D, E)$ — FD1: $AB \rightarrow CDE$, FD2: $D \rightarrow E$

1NF ✓

2NF ✓ No PD.

3NF ✗ FD2 = TD.

$R1(A, B, C, D)$ $\quad R2(D, E)$

$\{A, B\} \rightarrow \{C, D\}$ $\quad D \rightarrow E$

OR

$X \rightarrow A$

X: SK, A: Prime ✗

$R(A, B, C) \leftarrow (A, B)$ — All Key Relation.

(B, C)

★ **BCNF:**

A Stronger form of 3NF where a Non-Prime attribute determines Prime Attribute.

AND

$X \rightarrow A$

X: SK, A: Prime ✗

Slide 10-57

- **Relation:**

$R(A, B, C) \Rightarrow (ABC)^+ \cong R.$

↓

minimal?

R — Candidate key.

| A | B | C | D |

$K_R = \{A, CD\}$.

$A_{Prime} = \{A, C, D\}$.

$A_{NonPrime} = \{B\}$.

$A \rightarrow B$ ✓

$CD \rightarrow B$

∴ No Partial Dependency.

$R(A, B, C)$

$AB \rightarrow C \cong R.$ ✓

$BC \rightarrow A \cong R.$ ✓

} 2NF, No PD.

Ex: $S(sname, saddress, item, iname, price)$ and FDs.

FDs:

{ sname → saddress,

item → iname,

sname item ⇒ price.

}.

Abbreviate the above schema

$SUP(S, A, I, N, P)$

$S \rightarrow A$

$I \rightarrow N$

$SI \rightarrow P.$

} Keys of R.

$\{S\}^+ = \{S, A\}$.

$\{I\}^+ = \{I, N\}$.

$\{SI\}^+ = \{S, A, I, N, P\} \cong R$.

SI is the only key.

SI becomes primary key.

$A_P = \{S, I\}$.

$A_{NP} = \{A, N, P\}$.

$S \to A$. } Partial dependencies.
$I \to N$ }

- Decomposing the Relation:

SUPPLIER ITEMS (SName, ~~Item#~~, Price) } in 2NF
ITEM (Item#, IName)
SUPPLIER (SName, Addr).

- 3NF:

A Relation schema R is in third Normal Form (3NF) if whenever a FD $X \to A$ holds in R, then either:
(a). X is a superkey of R or
(b) A is a prime attribute of R. (All key Relation, no non-prime attribute)

Ex: A B C D — TDX ∴ 3NF ✓
PDX BCNF X.

** Boyce Codd Normal Form (BCNF):

- A relation schema R is in Boyce-Codd Normal form (BCNF) if whenever an FD $X \to A$ holds in R, then X is a Superkey of R.

Ex: R — FD2.
[SID | Major | FName]
FD1
FD3.

$F \to M$ ✓
$F^+ \not\cong R$ ← Superkey (if at all) ∴ X 3NF.

R_1 [S | F] R_2 [F | M]

It is lossless decomposition but we lost FD SM. (dependency non-preserving decomposition)

- OLTP generally are in 3NF or BCNF.

Ex: R [A | B | C] (All Key Relation).

No violation as Prime – Prime dependency.

• ABV Test: To check whether or not the decomposition is lossless.
(Tabular tests of FDs).

★ ARMSTRONG'S AXIOMS: Sound + Complete:

Database Schema
↳ Relation Schema
↳ Integrity Constraint → NULL ← Null Value Problem.
→ RI } RDBMS
→ EI } Key → PK/CK, Alternate Keys.
↳ enforced at DB Level
Special constraints.
↳ FDs
Domain ← Atomic Value
↳ Business Logic
↳ Assertions. ↑ all possible values.
↳ Triggers. PL/SQL
↳ Procedures & Functions.

• R(A, B, C)
X := { A → B, B → C } What all FDs are satisfied in R given FDs X.

Exhaustive Approach.

Valid FDs:
C → C, A → B, {ϕ} → {ϕ}, A → A, B → C, A → C,
B → B, A → BC, AB → C;

a). Not Trivial FDs. — Important
b) Trivial FDs — that are default, not important to mention.

Better Approach: IS FD X → Y holds in R?

$\alpha \cup \beta \cong A$, $\alpha \to \beta$ — Then, α is CK.

• AXIOMS:
Ex. α, β, γ are set of attributes.

1. Reflexivity:
if $\beta \subseteq \alpha$, then $\alpha \to \beta$

2. Augmentation:
if $\alpha \to \beta$, then $\gamma\alpha \to \gamma\beta$

3. Transitivity:
if $\alpha \to \beta$, and $\beta \to \gamma$, then $\alpha \to \gamma$.

A = {A1, A2, A3, A4}.
(α → β)
A1 → A2A3 — Given.

Using Augmentation Axiom,
A1A4 → A2A3A4.
can be redundant but holds valid key.

• These Rules are:
Sound: (generate only functional dependencies that actually hold), and
Complete: (generate all Functional dependencies that hold).

P.T.O.

4. The Union Rule:

If $\alpha \to \beta$ and $\alpha \to \gamma$
then $\alpha \to \beta\gamma$.

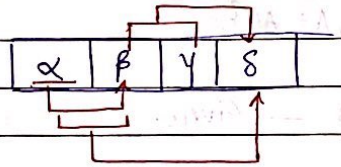
5. Decomposition Rule:

If $\alpha \to \beta\gamma$ then
$\alpha \to \beta$ and $\alpha \to \gamma$

6. Pseudo Transitivity Rule:

If $\alpha \to \beta$ and $\gamma\beta \to \delta$
then $\alpha\gamma \to \delta$.

Ex:



$\alpha \to \beta$ → FD1
$\beta\gamma \to \delta$ → FD2.
To Prove: $\alpha\gamma \to \delta$

By Augmentation rule to FD1
$\gamma\alpha \to \gamma\beta$ → FD3.

Using Transitivity Rule on FD2 & FD3.
$\gamma\alpha \to \gamma\beta$ and $\gamma\beta \to \delta$
then $\gamma\alpha \to \delta$.
Hence Proved.

Quit

Ex: $R = \{A, B, C, G, H, I\}$.
$F = \{A \to B (1), A \to C (2), CG \to H (3), CG \to I (4), B \to H (5)\}$.

Soln: Apply A3 (Transitivity Rule) to (1) & (5)
$A \to B$ and $B \to H$.
$\therefore A \to H$. → (6)

Apply A6 to (2) & (3)
$A \to C$ and $CG \to H$.
$AG \to H$ → (7).

Apply Similarly $AG \to I$. → (8).

Apply A4 to (3) & (4)
$CG \to HI$ → (9).

- Algorithm for Functional Closure / Closure set of Functional Dependency:

$F^+ = F$ Not used Practically.
repeat
  for each functional dependency f in F^+
    apply reflexivity and augmentation rules on f.
    add the resulting functional dependencies to F^+.
  for each pair of functional dependencies f_1 and f_2 in F^+.
    if f_1 and f_2 can be combined using transitivity.

WAL — Write Ahead Log.

Page No.
Date

Then add the resulting functional dependency to F^+ until F^+ does not change any further.

* F^+: Use := to check/determine whether specific FD holds good on R.

★ Algorithm to compute α^+, the closure of α (attribute closure) under F.

result := α
while (changes to result) do
for each $\beta \to \gamma$ in F do
begin
if $\beta \subseteq$ result then
result := result $\cup$ γ.
end.

Given a set of attributes α, define the closure of α under F (denoted by α^+) as the set of attributes that are functionally determined by α under F.

★ Canonical cover is used to test the goodness of instance and Normalization is used to test the goodness of the schema. (F_c)

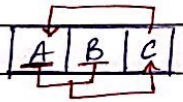
Minimal Cover / Reduced set of dependencies.

Page No.
Date

Ex 1: Consider the given relation and functional dependencies, the Relation R(A,B,C), $F = \{AB \to C, C \to A\}$. Identify the highest normal form for this relation.

Solution: Keys:

$\{AB\}^+ = AB.$
$= ABC \longrightarrow$ FD1 : $\cong R$ ∴ Key.



✓1NF → $(AB)^+$ ✓
✓2NF → P_{part} → NP ✗
✓3NF → NP → NP. ✗
via CK

∴ The relation is in 3NF.

Ex 2: Consider the given relation R(A,B,C,D) and set of functional dependencies. $F := \{AB \to C, C \to B, C \to D\}$. Determine (all keys) the keys, prime attributes, non-prime attributes.

Solu: Keys:

✓$\{AB\}^+ = AB.$
$= ABC \longrightarrow$ FD1.
$= ABCD \longrightarrow$ FD3 $\cong R \to$ CK1.

$\{C\}^+ = CBD. \not\cong R.$
✓$\{AC\}^+ = ABCD \cong R \longrightarrow$ CK2

Keys are AB & AC.

Prime Attributes = {A, B, C}.
Non - Prime Attributes = {D}.

Ex 3: Consider the given functional dependencies, $F = \{AB \to CD, AF \to D, DE \to F, C \to G, F \to E, G \to A\}$, Which of these is false ?

A) $CF^+ = ACDEFG$.
B) $BG^+ = ABCDG$.
C) $AF^+ = ACDEFG$.
D) $AB^+ = ABCDG$.

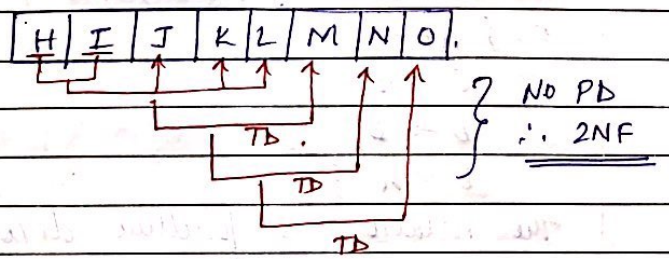
Soln: A) $CF^+ = CF$
$= CFG$.
$= CEFG$
$= ACEFG$.
$= ACEFDG$. ✓

C) is false.

Ex4: What Normal form the following relation is in where only H, I can act as a Key !
Staff (H, I, J, K, L, M, N, O).

$F = \{HI \to JKL, J \to M, K \to N, L \to O\}$.

Soln: Key:
$\{HI\}^+ = HI$
$= HIJKL$.
$= HIJKLMNO$.



Ex:5: Relation R (A, B, C, D), with AB as the primary key, Choose the FD so that R should be in 2NF but not in 3NF.
$F = \{D \to C, AB \to D, AB \to C, A \to D\}$.

Soln: $AB \to CD$
$\alpha \to \beta$

$D \to C$ $AB \to D$ When AB is
$AB \to C$ $A \to D$ Key this holds to 3NF.
↓
creates a PD, but already 2NF.

Ex6: Relation R(A, B, C, D), with AB as primary key. Choose FD such that R should be in 1NF but not in 2NF.

$\{AB \to C,\ AB \to D,\ A \to D,\ AB \to CD\}$

Soln: $A \to D$ is the PD. ∴ Not in 2NF.

Ex: Consider the given relation and its FDs.
The Relation is $R(A,B,C,D,E)$.
$F = \{ AB \to C,$
$C \to E,$
$B \to D,$
$E \to A \}$;
The relation is further decomposed into 2 ~~rep~~ relations
$R_1(B,C,D)$ $R_2(A,C,E)$.
Comment on whether the decomposition is lossless and/or dependency preserving.

★ Equivalent set of FDs:

1. A set of Functional Dependencies F ~~covers~~ is set to cover another set of functional dependency 'E', if every FD in 'E' is also in 'F^+'; i.e., if every dependency in E can be inferred from F, we can say that 'E' is covered by 'F'.

$F \supseteq E \leftarrow (E \cong F) \rightarrow E \supseteq F$

2. Two set of FDs E & F are equivalent if $E^+ = F^+$. Therefore equivalence means every FD in E can be inferred from F and every FD in F can be inferred from E. E is equivalent to F if E covers F and F covers E hold.

Ex.1 $F = \{ A \to C;\ AC \to D;\ E \to AD;\ E \to H \}$.
$G := \{ A \to CD;\ E \to AH \}$.

Soln:

F	G
$A \to C$	$A \to CD$.
$AC \to D$	$E \to AH$.
$E \to AD$	
$E \to H$.	
$A^+ = ACD$.	$A^+ = ACD$.
$E^+ = ADHEC$	$E^+ = EAHCD$.
$AC^+ = ACD$	
using G → $E^+\big\vert_G = ACDEH$.	∴ $G \supseteq F$ $A^+\big\vert_F = ACD$.
$A^+\big\vert_G = ACD$.	$E^+\big\vert_F = ADHEC$.
$\{AC\}^+\big\vert_G = ACD$.	∴ $F \supseteq G$.

using F

As $F \supseteq G$ and $G \supseteq F$,
then $F = G$

Ex 2: Given the Relation $R(A,B,C,D,E)$ and the set of FDs $F = \{ A \to B,\ AB \to C,\ D \to E \}$, P.T.O.

$G = \{A \to BC, D \to AE\}$, are these FDs equivalent

Soln:

F	G
$A \to B$	$A \to BC$
$AB \to C$	$D \to AE$.
$D \to E$	
$A^+ = ABC$	$A^+ = ABC$
$\{AB\}^+ = ABC$	$D^+ = ABED$
$D^+ = DE$	$= ABCDE$.
$A^+\|_G = ABC$	$A^+\|_F = ABC$.
$\{AB\}^+\|_G = ABC$	$D^+\|_F = DE \neq D^+\|_G$
$D^+\|_G = ABCDE$	
$\therefore G \geq F$	$\therefore F \neq G$.

★★ CANONICAL COVER:

- Irreducible set of FDs → Minimal set of FD is a canonical cover (Fc). → Standard set of FDs.
- ★ Any instance of R on which F holds, we need not check all the dependencies of F on R, rather just check 'Fc'

1. A Minimal Cover of set of Functional Dependencies 'E', is a set of FD's 'F' that satisfies the property that every dependency in 'E' is in closure 'F^+' of F.

In addition, this property is lost if any dependency from set 'F' is removed, i.e., 'F' must not have any redundancies and the dependencies in 'F' are in a standard form.

- 2. The Functional dependency 'F' is minimal if it satisfies following conditions.
  1. Every dependency in F has a single attribute for its RHS.
  2. We cannot replace any dependency of the form $X \to A$ in F with a dependency $Y \to A$, where Y is a proper subset of X ($Y \subseteq X$) and still have a set of dependencies, i.e., equivalent to F.
  3. We cannot remove any dependency from 'F' and still have a set of dependencies i.e., equivalent to 'F'.
- Refer Algorithm from Elmasri - Navathe.

Ex:1 Given a relation R(A,B,C) with following set of dependencies,
$F = \{A \xrightarrow{1} BC, B \xrightarrow{2} C, A \xrightarrow{3} B, AB \xrightarrow{4} C\}$.
Compute 'Fc'.

P.T.O

Page No.
Date

Soln: 1. RHS must have single attribute.
2. Multi-attribute on LHS in $\alpha \to \beta$
$\alpha \to \beta^r$ redundant ↑ any attribute is EXTRANEOUS.
and must $F_c \supseteq F$.

Apply ①.

$A \to B$ — ①
$A \to C$ — ②
~~$A \to B$~~
$B \to C$ — ③
$AB \to C$ — ④

from 1 & 3 using A3 (transitivity Rule),
~~① $A \to B$~~
we infer ② $A \to C \models A \to B, B \to C$.
Redundant

$F = A \to B$ — ①
~~$B \to C$~~ — ②
~~$A \to C$~~
$AB \to C$ — ④ → $(A \to C)$ — ④.

$(AB)^+ = ABC$.
$AB \to C$. $A|_{Bx} \to ABC$
without B.
∴ B is extraneous.

AC is redundant
∴ $\boxed{F_c = \{A \to B, B \to C\}}$

Ex 2: Consider a Relation R(A,B,C,D) and a set of Functional Dependencies, $F = \{A \to BCD, BC \to AD, D \to C\}$. Find the Minimal Cover of F.

$AB \to C$.
$B \to C$.

Page No.
Date

Soln: Apply ①

$A \to B$ — ①
$A \to C$ — ②
$A \to D$. — ③
$BC \to A$ — ④
$BC \to D$ — ⑤
$D \to C$. — ⑥

from ③ & ⑥ using A3 (transitivity Rule),
we infer $A \to D, D \to C \models A \to C$
but redundant.

$A \to B$ → ①
~~$A \to C$~~
$A \to D$ → ③
$BC \to A$ — ④
$BC \to D$ — ⑤
$D \to C$ — ⑥

from ④ & ③ using A3,
we infer $BC \to A, A \to D \models BC \to D$.
but redundant.

$A \to B$ — ①
$A \to D$ — ③
$BC \to A$ — ④
~~$BC \to D$~~
$D \to C$ — ⑥.

F_c.

② Check for Extraneity.
$BC \to A$. $BC^+ = ABCD$
$B^+|_{Cx} = B$

2nd Minimal Cover:

$A \to B$ ① $A \to D$ ③ $BC \to D$ ⑤

$A \to C$ ② $BC \to A$ ④ $D \to C$. ⑥.

Using Rule A3 on ③ & ⑥.

$A \to D$, $D \to C \models A \to C$.

~~now at~~

$A \to B$	$BC \to A$
~~$A \to C$~~	$BC \to D$
$A \to D$	$D \to C$.

and following steps are same as 1st Canonical Cover. ∴ Only 1 Canonical Cover exists.

Ex 2: $R(A, B, C, D)$

$A \to BCD$

$B \to CDA$

$C \to ABD$.

$A^+ = B^+ = C^+ = R$.

Soln:

$A \to B$	$B \to C$	$C \to A$
$A \to C$	$B \to D$	$C \to B$
$A \to D$	$B \to A$	$C \to D$.

Using $A \to B$, $B \to C \models A \to C$.

$A \to B$	$B \to C$	$C \to A$
~~$A \to C$~~	$B \to D$	$C \to B$
$A \to D$	$B \to A$	$C \to D$

Using $A \to B$, $B \to D \models A \to D$.

~~$A \to B$~~	$B \to C$	$C \to A$
~~$A \to D$~~	~~$B \to D$~~	~~$C \to B$~~.
	~~$B \to A$~~	$C \to D$.

Similarly eliminate ① $B \to C$, $C \to A \models B \to A$

② $C \to A$, $A \to B \models C \to B$.

③ $B \to C$, $C \to D \models B \to D$

∴ Final ~~Multiple~~ Minimal Cover is,

$\{A \to B, B \to C, C \to A, C \to D\}$.

Soln 2:

$A \to B$	$B \to C$	$C \to A$
$A \to C$	$B \to D$	$C \to B$
$A \to D$	$B \to A$	$C \to D$.

Using $A \to C$, $C \to B \models A \to B$

~~$A \to B$~~	$B \to C$	$C \to A$
$A \to C$	$B \to D$	$C \to B$
~~$A \to D$~~	$B \to A$	$C \to D$.

Using $A \to C$, $C \to D \models A \to D$.

$A \to C$	$B \to C$	$C \to A$
~~$A \to D$~~	$B \to D$	$C \to B$
	$B \to A$	$C \to D$.

Using, $B \to A$, $A \to C \not\models B \to C$.

$A \to C$ ~~$B \to C$~~ $C \to A$

$B \to D$ $C \to B$

$B \to A$ $C \to D$.

Using, $C \to B$, $B \to A$, $\not\models C \to A$

$A \to C$ $B \to D$ ~~$C \to A$~~

$B \to A$ $C \to B$

$C \to D$

Using, $C \to B$, $B \to D \not\models C \to D$.

$A \to C$ $B \to D$ $C \to B$

$B \to A$ ~~$C \to D$~~

Final cover: $\{A \to C, B \to A, B \to D, C \to B\}$.

... many more F_c's possible.

You can crosscheck by applying attribute closures on F_c, it should match with that of F.

- Normalization
  - ↳ Decomposed Relations ← lossless / lossy? Dependency Preserving.

1. Binary Decomposition:

When a relation is decomposed in two relations.

2. Multi - Relation Decomposition:

When a relation is decomposed in many relations.

~~AA~~ Binary Decomposition for non - additive join property:

A decomposition $D = \{R_1, R_2\}$ of R has the lossless (non-additive) join property with respect to a set of FDs F on R, iff either

① the FD $(R_1 \cap R_2) \to (R_1 - R_2)$ is in F^+, or the FD $(R_1 \cap R_2) \to (R_2 - R_1)$ is in F^+.

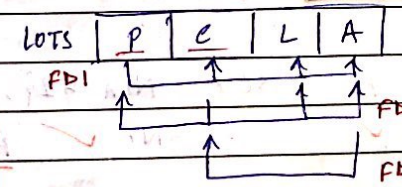
Ex: ~~TEACH (Student, Course, Instructor).~~

LOTS (Prop_ID, CountryName, Lot#, Area).

Prop_ID → {CountryName, Lot#, Area).

CountryName → {Prop_ID, Lot#, Area).

Area → CountryName.



Prime Attributes = {P, C}. NP → NP ∴ 3NF ✓

Non Prime = {L, A}. not exist

Keys = P, C. NP → P exists ∴ BCNF ✗

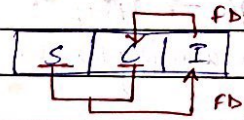
∴ Decompose the relation:
→ LOTS A (P, L, A)
→ LOTS B (A, C).

Now check lossless / lossy property:
$R_1 = (P, L, A)$ $R_2 = (A, C)$.
$R_1 \cap R_2 = \{A\}$.

✗ $A \rightarrow PL$ OR $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$
✓ $A \rightarrow C$. — " — $(R_2 - R_1)$.

∴ The decomposition is lossless.

Ex 2: TEACH (Student, Course, Instructor).
$FD_1 = \{$ Student, Course → Instructor $\}$
$FD_2 = \{$ Instructor → Course $\}$.



Key = SC, $P_A = \{SC\}$, $NP_A = \{I\}$,
1NF ✓, 2NF ✓ ∴ 3NF ✓ BCNF ✗

Teach 1 = { Instructor, Course }.
Teach 2 = { Instructor, Student }.

$R_1 \cap R_2 = \{I\}$, $R_1 - R_2 = \{C\}$, $R_2 - R_1 = \{S\}$.

✓ $I \rightarrow C$ OR. ∴ The decomposition is
$I \rightarrow S$. lossless.

Ex: Consider the following decompositions for the Relation Schema 'R'. Determine whether each decomposition has the lossless join property w.r.t. 'F'.
1. $D_1 = (R_1, R_2, R_3, R_4, R_5)$
$R_1 = A, B, C$. $R_4 = F, G, H$.
$R_2 = A, D, E$. $R_5 = D, I, J$.
$R_3 = B, F$

Schema R: =
G: = $AB \rightarrow C$, $BD \rightarrow EF$, $AD \rightarrow GH$, $A \rightarrow I$, $H \rightarrow J$.

	A	B	C	D	E	F	G	H	I	J
R_1	α_1	α_2	α_3	β_{14}	β_{15}	β_{16}	β_{17}	β_{18}	β_{19}	β_{10}
R_2	α_1	β_{22}	β_{23}	α_4	α_5	β_{26}	β_{27}	β_{28}	β_{29}	β_{20}.
R_3	β_{31}	α_2	β_{33}	β_{34}	β_{35}	α_6.	β_{37}	β_{38}	β_{39}	β_{30}
R_4	β_{41}	β_{42}	β_{43}	β_{44}	β_{45}	α_6	α_7	α_8	β_{49}	β_{40}
R_5	β_{51}	β_{52}	β_{53}	α_4	β_{55}	β_{56}	β_{57}	β_{58}.	α_9	α_0

Ex:

Relation	P	L	C	A
Lots A	α_1	α_2	~~β_{13}~~	~~α_4~~
Lots B	β_{21}	β_{22}	α_3	α_4.

FD1: ~~$C \rightarrow PLA$~~
FD2: $P \rightarrow LCA$
FD3: $A \rightarrow C$.

Now, $A \to C$.

Lots A : $\alpha_4 \to \beta_{13}$

Lots B : $\alpha_4 \to \alpha_3$.

$\therefore$ we can make $\beta_{13} = \alpha_3$.

Relation	P	L	C	A
Lots A	α_1	α_2	α_3	α_4
Lots B	β_{21}	β_{22}	α_3	α_4

← the moment we see all alphas in a row we declare that the decomposition is lossless.

TESTING THE LOSSLESSNESS OF DECOMPOSITION: The CHASE Algorithm (ABU Algo) — 1979.

— Alfred Aho; C. Beeri; Jeffrey D. Ullman.

$R, F \Rightarrow F^+$

↓ Normalize

$R = R'$ $\quad R_1 \quad R_2 \quad R_3$

$\bowtie \quad \bowtie$ // Spurious / Phantom Records.

↓

R'

if $R \neq R'$, then the decomposition is lossy. either extra or some records are lost.

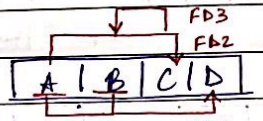
Therefore,

1. Non-Additive Join property.
2. Dependency Preservation ← may be compromized.

Ex: $R(A, B, C, D) \to A \to C,\ AB \to D,\ C \to B$.

$R_1(A, B, D)$

$R_2(A, C)$.



Check conditions for lossy or lossless relation.

1. $R_1 \cup R_2 = R$ ✓
2. $R_1 \cap R_2 \neq \phi$ $\quad \{A\}$ ✓
3. $\{R_1 \cap R_2\} \to \{R_1 - R_2\}$ $\quad A \to BD$.

   $\to \{R_2 - R_1\}$ $\quad A \to C$. ✓

$(R_1 \cap R_2) \cong CK$ of either R_1 or R_2 → here 'A' should have unique domain.

$\therefore$ The decomposition is LOSSLESS but Not Dependency Preserving.

- Definition of Non-Additive Join Property:

A decomposition 'D' $= \{R_1, R_2, \ldots R_m\}$ of 'R' has the lossless (non-additive) join property with respect to the set of dep functional dependencies F on R

instance of R.

if for every relation state 'r' of 'R' that satisfies 'F', the following holds:

$$r = \bowtie(\pi_{R_1}(r), \pi_{R_2}(r), \ldots \pi_{R_m}(r))$$

CHASE ALGORITHM:

Ex:1 Consider the schema R(S, E, P, PN, L, H).

$R_1(E, L)$.

$R_2(S, P, H, PN, L)$

$F := \{S \to E, P \to PN, L, SP \to H\}$.

So

	S	E	P	PN	H	L
R_1	β_{11}	α_2	β_{13}	β_{14}	β_{15}	α_6
R_2	α_1	β_{22}	α_3	α_4	α_5	α_6

$\bigcup_{i=1}^{m} R_i = R$ ✓

$\bigcup_{i=1}^{m} R_i = R$ if this holds, then only apply CHASE Algorithm, else it is already lossy.

CHASE: Use FDs

Check for FDs to hold, if yes replace RHS of FD by α_i.

No FD hold when we check FD from α's in same column.

∴ This is Lossy.

1. $\bigcup_{i=1}^{2} R_i = R$ ✓
2. $R_1 \cap R_2 = \{L\}$ ✓
3. L is not CK ✗✗ ∵ L → does not determine any column.

Ex2: R(S, E, P, PN, L, H)

$R_1(S, E)$	$S \to E$
$R_2(P, PN, L)$	$P \to PN, L$.
$R_3(S, P, H)$	$S, P \to H$.

So

	S	E	P	PN	L	H
R_1	α_1	α_2	β_{13}	β_{14}	β_{15}	β_{16}
R_2	β_{21}	β_{22}	α_3	α_4	α_5	β_{26}
R_3	α_1	β_{32}	α_3	β_{34}	β_{35}	α_6

is R_i is all α ? No

Iteration 1: FD 1, $S \to E$.

	S	E	P	PN	L	H.
R_1	α_1	α_2				
R_2			α_3	α_4	α_5	
R_3	α_1	α_2	α_3			α_6.

is R_i is all α ? No

FD-2, $P \to PN, L$

	S	E	P	PN	L	H.
R_1	α_1	α_2				
R_2			α_3	α_4	α_5	
R_3	α_1	α_2	α_3	α_4	α_5	α_6.

is R_i is all α ? Yes

Stop.

∴ it is lossless

Ex: Consider a relation R(A, B, C, D, E, G, H) and the set of functional dependencies $F = \{AB \to C, AC \to B, AD \to E, B \to D, BC \to A, E \to G\}$. Test whether the following decompositions following non-additive Join property.

D1 = {AB, BC, ABDE}, EG}.
D2 = {ABC, ACDE, ADG}.

$R_1 = \{A, B\}$, $R_2 = \{B, C\}$, $R_3 = \{A, B, D, E\}$, $R_4 = \{E, G\}$.

F = { AB → C FD1
AC → B FD2
AD → E FD3
B → D FD4
BC → A FD5
E → G FD6.
};

Decomposition 1:

So

	A	B	C	D	E	G
R_1	α_1	α_2	β_{13}	β_{14}	β_{15}	β_{16}
R_2	β_{21}	α_2	α_3	β_{24}	β_{25}	β_{26}
R_3	α_1	α_2	β_{33}	α_4	α_5	β_{36}
R_4	β_{41}	β_{42}	β_{43}	β_{44}	α_5	α_6

Iter -01

FD: 4 ⇒ B → D.

	A	B	C	D	E	G
R_1	α_1	α_2		α_4		
R_2		α_2	α_3	α_4		
R_3	α_1	α_2		α_4	α_5	
R_4					α_5	α_6

FD 6 : E → G

	A	B	C	D	E	G
	α_1	α_2		α_4		
		α_2	α_3	α_4		
	α_1	α_2		α_4	α_5	α_6
					α_5	α_6

Iteration 2: FD 3. AD → E.

	A	B	C	D	E	G
R_1	α_1	α_2		α_4	α_5	
R_2		α_2	α_3	α_4		
R_3	α_1	α_2		α_4	α_5	α_6
R_4					α_5	α_6

FD6 E → G

A	B	C	D	E	G
α_1	α_2		α_4	α_5	α_6
	α_2	α_3	α_4		
α_1	α_2		α_4	α_5	α_6
				α_5	α_6

Iteration : 3

No change in two consecutive states ∴ Stop.

The decomposition is lossy.

Ex: Given a relation R(A,B,C,D), test whether the following decompositions ~~and/or~~ are lossless / dependency preserving.

1. FDs = {B→C, D→A}.
   with decomposition {BC, AD}.
2. {AB→C, C→A, C→D}
   with decompositions {ACD, BC}.
3. {A→BC, C→AD}
   with decompositions {ABC, AD}.
4. {A→B, B→C, C→D}
   with decompositions {AD, ACD}.
5. {A→B, B→C, C→D}
   with decompositions {AB, AD, CD}.

	A	B	C	D
R_1	α	β	α	α
R_2	~~β~~α	α	α	~~β~~α

✓

	A	B	C	D	
R_1	β	α	α	β	X
R_2	α	β	β	α	X

	A	B	C	D	
R_1	α	β	β	α	X
R_2	α	β	α	α	X

	A	B	C	D	
R_1	α	α	α	β	
R_2	α	~~β~~α	~~β~~α	α	✓

	A	B	C	D	
R_1	α	α	β	~~β~~	X
R_2	α	~~β~~α	β	α	Y
R_3	β	β	α	α	Y

10/2/20

UNIT – 3

STORAGE STRUCTURES & INDEXING | HASHING.

Persistence of Data	Faster Access	Point Queries
	of All Time O(1)	
	Point (Exact Match) Queries.	
	Range	
RAID	B^+ Tree (B Tree)	Key-value pair.
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7	Static	O(1)
* * *		mapping
maximize MTTF (Maximum time to failure.) ← Replication (done to recover data)		↳ randomization

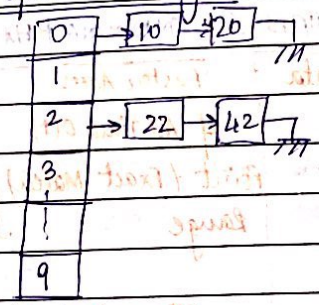
- Hashing:

UOD (logical space) → Core RAM Addresses (Physical Space.)

≡ Can be redundant ⇒ leads to key collisions
⇒ collision resolution.

- HASHING
  - Static Hashing
    - Open Addressing (Chaining)
    - Closed Addressing (Probing).
  - Dynamic Hashing
    - Extensible Hashing.
    - Linear Hashing.

Open Addressing



Closed Addressing

0	10
1	20
2	22
3	42
!	!
8	
9	

— uses directory
  └ set of pointers.
— space is open as a linked list / BST is used to point data.
— Eliminates Collision Resolution ✓
— Localize the Key (Present/Absent) Once you know the address, just need to traverse the linked list to find the data. data is found at that key or else no where.
— Faster Access near O(1)
— More Memory Required
— Directory size is fixed. Thus a static Hashing.

— dont use directory
— space is closed
— Size of Hash Table is fixed. To change it Rehashing needs to be performed.
— one key per bucket / cell.
— Memory Optimal ✓

• Extendible Hashing: Keys — binarized — Hashed.
  └ Ronald Fagin (1979, ACM Transaction on DB Systems) (ACM Trans)

① . Buckets
② . Directory.
③ . Overflow Bucket (chain is not done).
④ . Concept of Depth → Global Depth (GD) = Hash function possible address of buckets in Directory
  ↘ Local Depth (LD)

• Buckets: No more holds a single key; capacity (N) fixed at start.

• Directory: — Dynamic — Number of buckets grow and/or shrink → Insertion/Deletion

* Buckets are created on demand.

• At the time of insertion, GD = LD and overflow occurs then, Double the directory i.e., GD = GD + 1, then rehash.

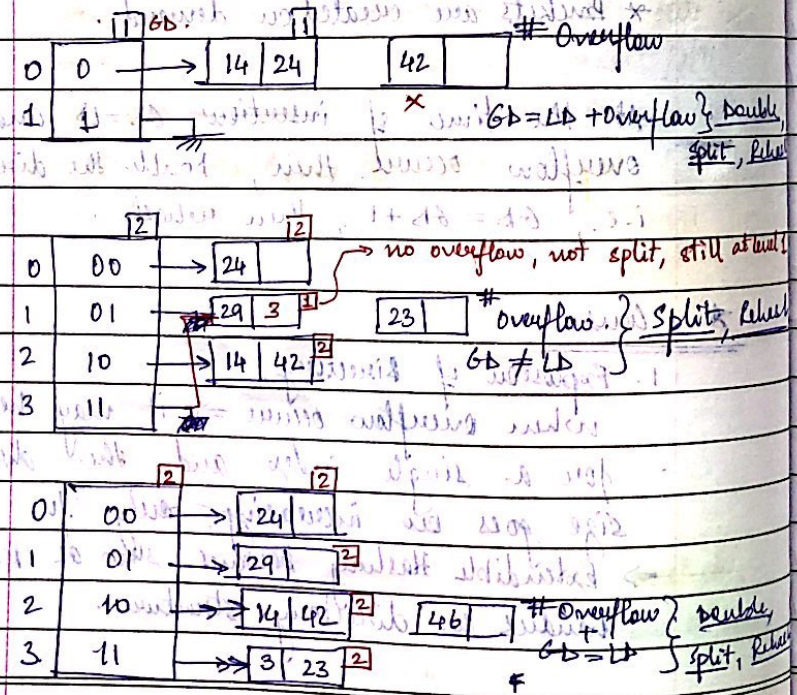
★ Cons:

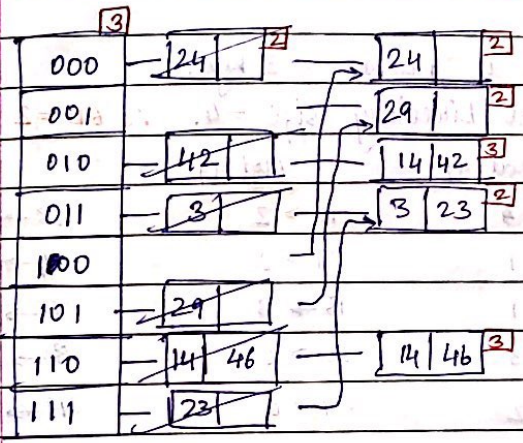
1. Explosion of Directory.
  when overflow occurs ⇒ it may be for a single index and the directory size goes on increasing. only for overflow
  ⇒ Extendible Hashing becomes slow as RAM also handles the directory structure.

Goodness of Open Addressing.

Ex: Keys = {14, 24, 42, 29, 3, 46, 23}.
s = Capacity of bucket = 2.
N = Initial buckets = 2. (GD = 1).
(If N not given, N = 2).

Keys	Binary
14	001110
24	011000
42	101010
29	011101
3	000011
46	101110
23	010111





- Disadvantages of Extendible Hashing:
(1). Needs directory, which can split when change in GD. Tedious to maintain directory structure.
(2). Buckets may split indefinitely if keys map to same bucket — Brings down the hash structure — Biased Splitting. (The same bucket keeps splitting if keys map to same bucket).

★★★ Linear Hashing:
— No directory used.
— Tries to emulate linear Probing.

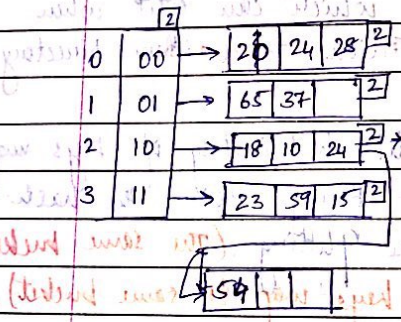
Ex: Extendable Hashing:
For the following set of keys,
23, 65, 37, 59, 18, 10, 74, 15, 20, 24, 28, 50, 75, 8, 49.
Apply Extendible Hashing with following preconditions.

If GD & LD are not given in question then take GD=1, LD=2

1. Size of bucket = 3 keys.
2. Initial directory size = 4. ∴ GD=2

	Mod(4)		Mod(4)		Mod(4)
23 →	3	10 →	2	28 →	0
65 →	1	74 →	2	54 →	2
37 →	1	15 →	3	75 →	3
59 →	3	20 →	0	8 →	0
18 →	2	24 →	0	49 →	1

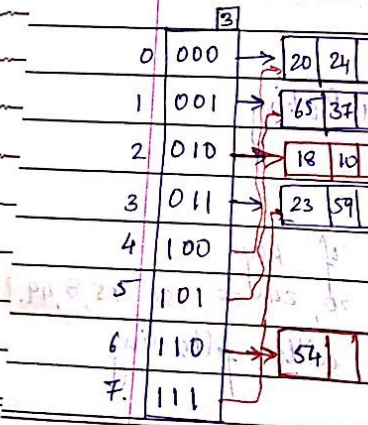
Initially:



* overflow (50)# → GD=LD ✓ overflow ✓

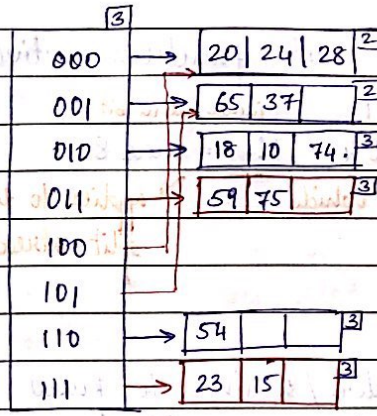
Double, split, Rehash

Only those bucket that changed. ★★★



* : overflow (75) → GD ≠ LD

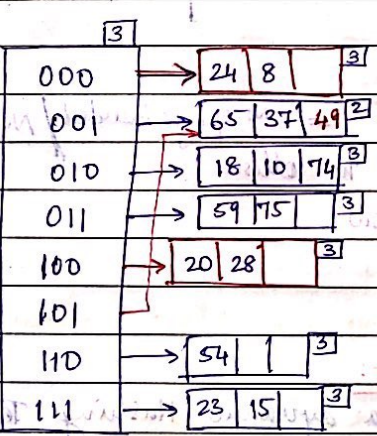
Overflow split | Rehash



* overflow (8) → GD ≠ LD.

Overflow ✓

Split | Rehash.



Insert 49 !

★ LINEAR HASHING:

Witold Litwin (1980), Paul Larson.

1. Eliminates concept of directory.
2. Introduces ~~page~~ Overflow Pages. — Chaining
3. Each bucket is equally likely to split in linear fashion. in current cycle.

→ Overflow at Bi may not cause split in Bi.

4. Here atmost 2 hash functions active at any stage h_i, h_{i+1}, increment in bit.

Ex: $h_i = \text{mod } 4$, $h_{i+1} = \text{mod } 8$

h_i applies to buckets which are not split. h_{i+1} applies to the buckets split already.

- **SPLIT Pointer:**
  - A counter / sentinel to know which bucket has been split. / has to be split.

- **Splitting Factor:**
  1. Load Factor. = No. of keys inserted / No. of buckets.
  2. Overflow due to collision
  3. LF + Overflow.

- **LINEAR HASHING:**

It is a dynamic Hashing Technique that adjusts gracefully to inserts & deletes. It does not require directory, and deals with the collisions naturally. It allows for higher flexibility with regard to timing of splitting the buckets. It includes an overflow chain of buckets for provisioning higher average space utilization.

1. When the data is heavily skewed, the linear Hashing performance degrades due to longer overflow chains. (cons)
2. Linear Hashing allows for the expansion of the hash table one slot at a time. The frequent single slot expansion can effectively control the length of the collision chain.

Ex: For the set of keys 8, 13, 10, 15, 19, 22, 29, 12, 17, 23, 11, 16.

→ Initial Buckets = 2 (N)
Current Level = 0 (L) default
Bucket Capacity = 2 (c) default
Split Pointer = 0 (S) default.

Criteria = LF # Split Policy := Load Factor LF = 75% (or 1/4 75%)

→ mod 4 mod 2 → at init number of buckets = 2.

h_1	h_0	
00	0	S → 8 | 10
01	1	13 | 15 → 19 (Overflow)
10		10
11		

LF = 4/(2×2) = 100% → Split.

h_1	h_0	
00	0	8 | 1
01	1	S → 13 | 15 → 19
10		10
11		

Keys = 5, capacity = 6 → > 75%. ∴ Split.

h1	h0	
00	0	8 | 10
01	1	13 | 15
10		10
11		15 | 19

mod8 h1	mod4 h0	
000	00	8
001	01	17 | 29 → [full]
010	10	10 | 22
011	11	15 → 19 | 11 → 23 | 31
100		12
101		13 | 29
110		22
111		15 | 23

S reinitialized

$LF_{22} = 6/8 = 75$ ✓

$LF_{29} = 7/8 > 75$ ✗ split,

$LF_{12} = 8/10 > 75$, split

$LF_{17} = 9/12 = 75$ ✓

$LF_{23} = 10/12 > 75$ ✗

$LF_{11} = 11/14 > 75$ ✗

Reinitialize S

mod16 h1	mod8 h0	
0000	000	8 | 16
0001	001	17
0010	010	10
0011	011	19 | 11
0100	100	12
	101	13 | 29
	110	22
	111	15 | 23

$LF_{16} = 12/16 = 75$ ✓

Ex: Same Example using Criteria as Overflow.

⇒ Initial Buckets = 2. ; Capacity = 2.

Split Criteria : = Overflow. Split Pointer = 0.

8, 13, 10, 15, 19, 22, 29, 12, 17, 23, 11, 16.

mod4 h1	mod2 h0	Split Ptr. / Overflow Chain.
00	0	8
01	1	13 → 19
10		10 | 22
11		15 | 19

@ insert 19 ↑ store key in overflow bucket.

- Overflow ; Split Bucket 0
- Rehash the keys in bucket 0 and 2. using Hi+1
- Increment split ptr.

@ insert 29

- overflow ; Split B_1, Rehash B_1 & B_3 with Hi+1
- Split ++ ; Complete level, Reinitiate to 0.

mod8 h1	mod4 h0	
000	00	8
001	01	13 → 29 → 17
010	10	10 | 22
011	11	15 → 19 → 23 | 11
100		12
101		13 | 29
110		22
111		

Block – Medium of transfer between Secondary Memory and main Memory.

@ insert 17:
• Overflow split B_{00} → B_{000} & B_{100}, Rehash Mod8.
@ insert 23:
Overflow split B_{01} → B_{001} & B_{101}.
@ insert 11:
Overflow, split B_{10} → B_{010} & B_{110}.

★★ B+ Trees:

B-Tree (B ↔ Balance)
N-ary Tree $N \simeq 2^8$+
Key + pointer

Integral Keys / alphanum ← ASCII EBCDIC:

• For BST: (Covering Tree / Indexes)

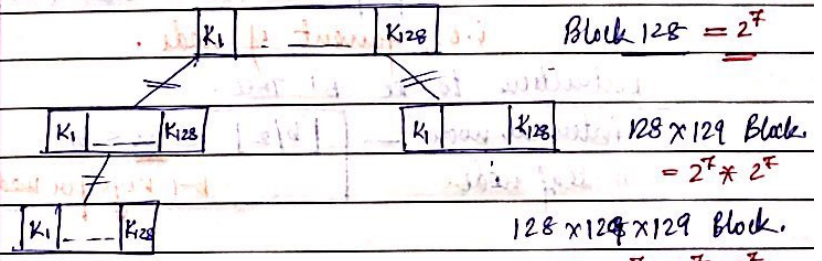
Ex: No. of Records = $2^{30} = 10^9$, consider BST.
Find no. of accesses to locate a block containing search key.
Size of Record = 1KB.
Size of Block = 8KB.
∴ No of blocks = $\frac{2^{30}}{8}$ = 2^{27} blocks.

★ BST → store key + not data!
block Access = depth BST + 1 = 26 + 1 = 27 access.
key search. actual access.

• For B+ Tree:
— Only store key and No data Records.

Size of N-ary Tree Node = Size of Block = 8KB.
Size of Key = 54 byte; pointer = 10 byte.
Keys per node = $\frac{8KB}{64 \text{ byte}} = \frac{8 \times 2^{10}}{2^6} = 8 \times 16 = 128$
Size of index entry = 64 byte.

no. of child of a node.
fanout = 128 + 1 = 129.



Block 128 = 2^7
128 × 129 Block. = $2^7 * 2^7$
128 × 128 × 129 Block. = $2^7 * 2^7 * 2^7$.
∴

2^{28} Blocks = depth ⇒ 4

∴ Access to any data will be in only = α + 1 = 4 + 1 disk access. = 5 accesses

★ B+ TREE:

→ B+ Tree is an augmented B-Tree which does not store data records in internal node. All data records are pointed by leaf node.

→ Douglas Comer (1973) Filesystem. VSAM by IBM.
Bayer & McCreight (1972) B+ Tree

B/B+ Trees are created Bottom-Up.

Page No.
Date

- Supports : 1. Point queries (Fixed value queries)
  2. Range queries.

default indexing for LINUX | Win | MSSQL | Oracle | Sybase Server, IBMDB2

★ Characteristics:

1. Balance factor on order.
   b = no. of pointer in a node (children).
   i.e., fanout of node.

Restriction to be B+ Tree.

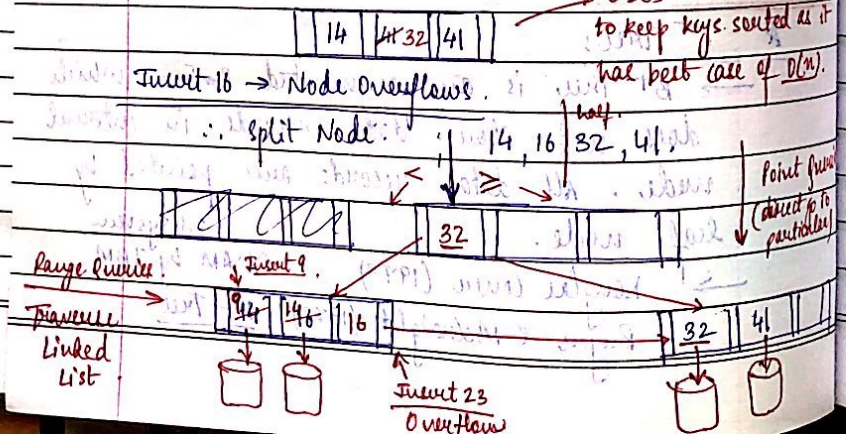
internal node & leaf node — $\lceil b/2 \rceil \le m \le b$; b−1 keys per node

Special case : ROOT $2 \le m \le b$.

Ex: Node b=4, $2 \le m \le 4$ | Keys = 3, 1 to 3.

14, 41, 32, 16, 9, 23, 42, 36, 8, 7

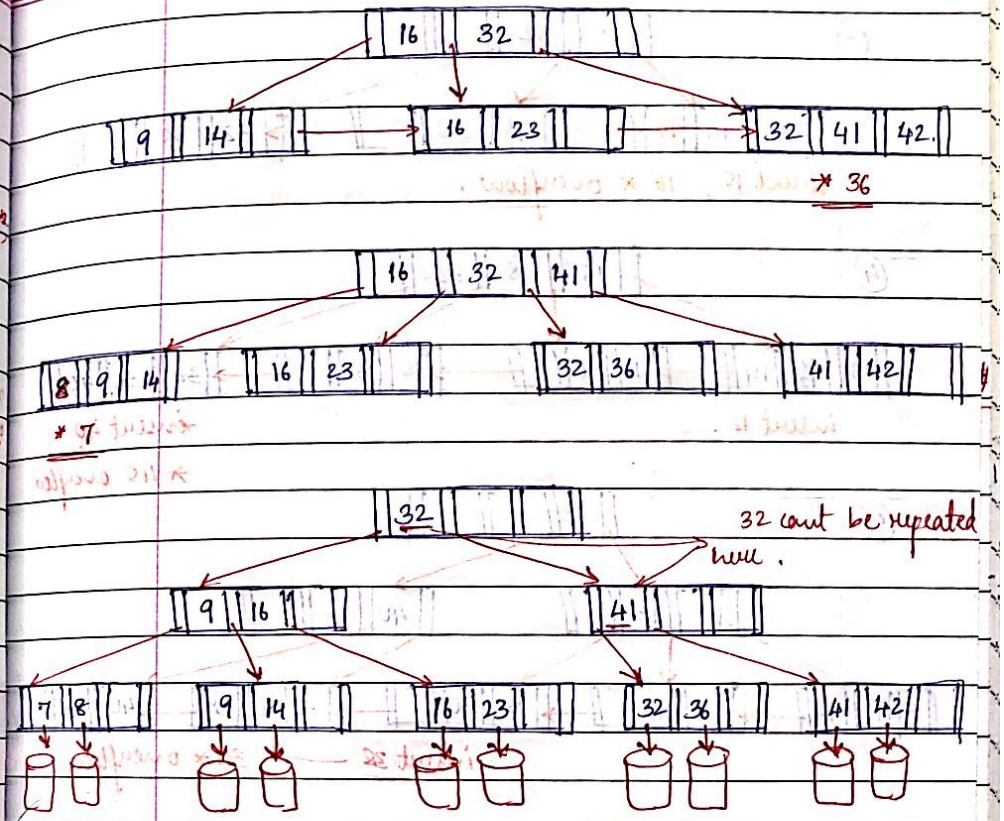
also, $m = 2-3-4$ ($b=4$) tree.



RAID from Forth

I-2 { Normalization, Covers, Storage Structure
Indexing & Hashing

Page No.
Date

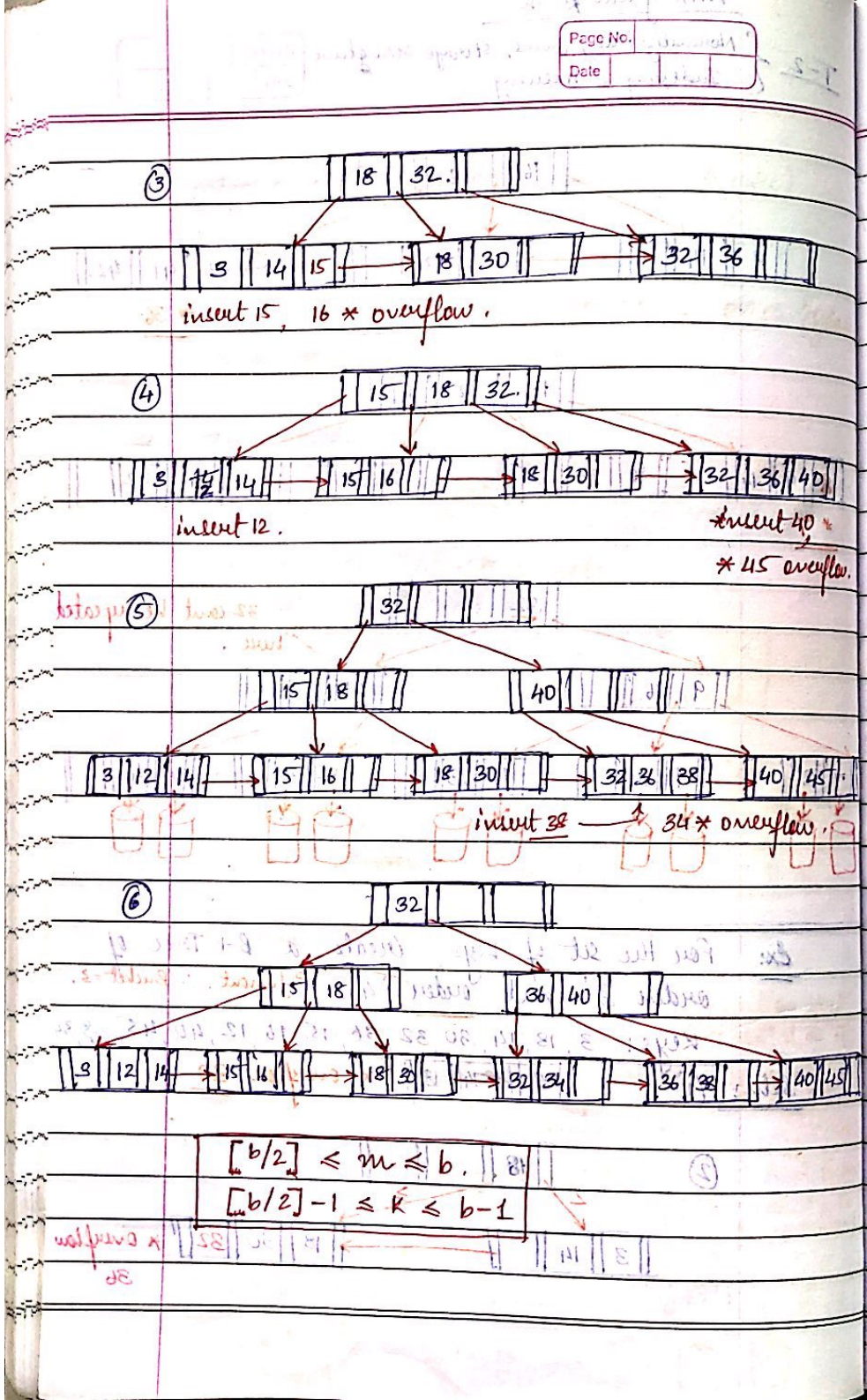


Ex: For the set of keys: Create a B+ Tree of order '3' and order '4'. → fanout. ∴ Bucket=3.

Keys: 3, 18, 14, 30, 32, 36, 15, 16, 12, 40, 45, 38, 34

Soln: ① 3 ~~18~~ 14 18 → overflow 30

② 18
3 14
18 30 32 → overflow 36

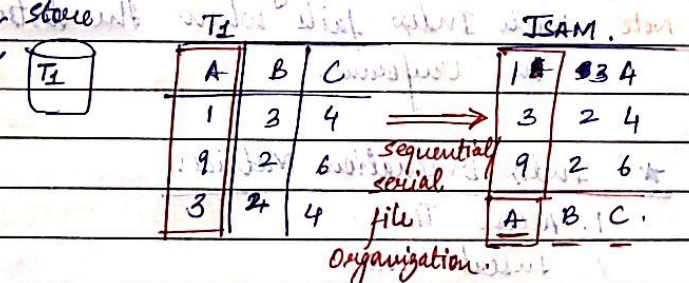


★ INDEXES:

1. Indexing Mechanisms need to speed up access to desired data.
   Ex: Author catalog in Library.

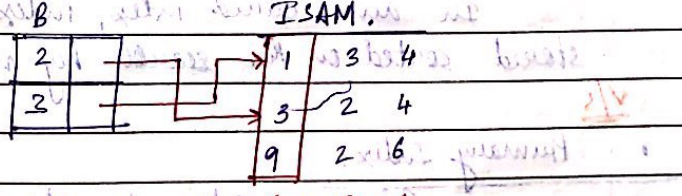
2. Search Key: Attribute to set of attributes to look up records in a file.

3. Ordered Indexes: can cause Overflow (Insert / Delete / Update)
   ↳ Sorting → Index Sequential Access Method Files (ISAM).
   ↳ Ordered Files (Data Records).



Primary Index is done on one column which reveals the actual location of the value associated, whereas in secondary Index, it does not reveal actual location.

Multivalued Indices



- Type of Ordered Indices:
  1. Primary: One Clustered Indices & one or more non-clustered.
  2. Secondary: Use one or more non-clustered / multi-value indices.

★ Hash Indices:
- ↳ Randomization
- ↳ O(1)
- ↳ Collision Handling.
- ↳ Extendible Hashing.

Note • Hash Index fails when the distribution is not Uniform.

★ Index Evaluation Metrics:
1. Access Time.
2. Insert Time
3. Delete Time
4. Space Overhead

• Ordered Index:

In an ordered index, index entries are stored sorted on the search key value...

V/S

• Primary Index:

Primary Index is stored depending upon the location of actual data.

★ It is not good to create clusters on Primary Key as it is 1-1 to associations.

• Clustering Index:

Sparse

B1	1	
B2	27	
B3	42	
B4	62	

Dense.

1	B1
!	B1
27	B2
!	!
B42	B3
!	!
62	B4
!	!

• The search key of a primary key index is usually but not necessarily the primary key.

• Indexes should be in Main Memory if it is not Multi-Staged.

★★★ Multilevel Index.

• Comparison Summary:
1. Primary Index (PK + Ordered Data Record).
2. Secondary Index (Non-Key + Unordered Data Record).
3. Clustered Index (Non Key + Ordered Data Record)

★ INDEXING:
- ↳ static ← Linear Probing, Separate Chaining.
- ↳ Dynamic ← Linear Hashing, Extendible Hashing.

} Comparisons

1. Division Remainder Method.
2. Digit Analysis Method — Sum of Number squares.

FLR - Fixed Length Record.
VLR - Variable Length Record.

$K = \{\overbrace{x}^{\text{num.}}, y, z, t\}$.

Alphanum	Char	Large No	DRM	EH / LH
Amit	A	1020926	x%k	
Ankit.	X	1926924		

19, 16, 25, 28.

mod 5 : 4 1 0 3

Extendible Hashing | Linear Hashing.

0	16	28
1	19	25.

** PK Index:

#records = 30000 — Unspanned strategy.

block = 1024 B.

record = 100 B.

Key = 6B

Value = 9B.

Unspanned Strategy — A Record is stored in a block only if required space is available.

Spanned Strategy — A record part is stored in a block if required space is not available.

- Efficiency of Indexes = No. of accesses.

- W/O Index:

Records / Block = $\frac{1024}{100} \cong 10$ records.

Block = $\frac{30000}{10} = 3000$ blocks

Case 1: sorted | Binsearch || data records are on physical order:

block Accesses = $\lceil \log_2 3000 \rceil \cong 12$.

Case 2: data records unordered | Linear Search.

block Accesses = $\left\lceil \frac{N}{2} \right\rceil$ // Average case.
= 1500+1 Blocks.

Case 3: With Index | Ordered Index Keys.

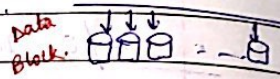
Index size = 6B + 9B = 15B.

Index Rec / Block = $\frac{1024}{15} = 68.26 \cong$ 68 Unspanned Strategy.

Leaf Blocks = # data Records = 3000 blocks.

Index Blocks = $\left\lceil \frac{3000}{68} \right\rceil \cong 45$

Block Accesses = $\lceil \log_2 45 \rceil + 1$ (Index Block)



$= 6 + 1 \cong 7$

Index Acces. physical memory access.

** Secondary Index:

↳ Does not give physical Memory locations of record.

↳ Index Records = No. of Records. = 30000

Index Block = $\lceil \frac{30000}{68} \rceil \cong 442$

Block Access = $\lceil \log_2 442 \rceil + 1 \cong 9 + 1 = 10$.

UNIT – 4.

QUERY PROCESSING & OPTIMIZATION

— Use Korth Book Mainly.

- **Query Processing:**

1 Query Tree → Query Profile → Query Transformation
↑ → DB Profile → DomainValues, Attributes
Relational Algebra → Indices.
↑ → Accesspaths
SQL.

- Root of the query Tree — Output.
  Leaves of tree — Physical Relations

2 # Estimation of cost of Query:

↳ Join
- Simple Nested Loop Join (SNLJ)
- Block Nested Loop Join (BNLJ)
- Merge Join.
- Hash Join

Merge Join = Merge Sort + Join:

1 3 4 3 9 2 → Sort (n log n)
9 3 5 4 1 → Sort (n log n)
Merge — O(n)

Heuristic for query Transformation.
Pipelining in query Optimization.

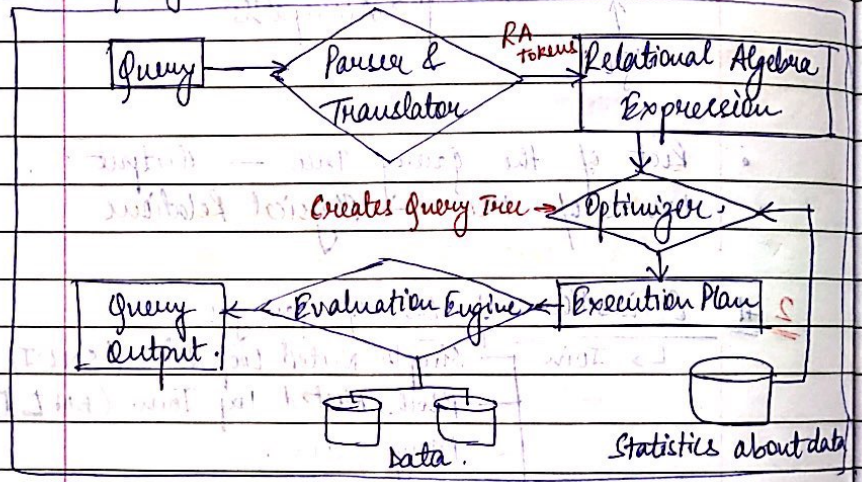
- Oracle hints

Page No.
Date

★ Basic Steps in Query Processing:

1. Parsing & translation.
2. Optimization.
3. Evaluation.

★ • Query Processing is processing the query in such a way that the cost of query execution is minimal.



1. Parsing & Translation:

↳ Translate the query into its internal form. This is then translated into relational algebra.
↳ Parser checks syntax, verifies relations.

2. Evaluation:

Omit.

3. Optimization:

Finding the cheapest evaluation plan for a query.

- Given relational algebra expression may have many equivalent expressions.

Ex: $\sigma_{balance<2500}(\Pi_{balance}(account))$ is equivalent to $\Pi_{balance}(\sigma_{balance<2500}(account))$

- Any relational-algebra expression can be evaluated in many ways. Annoted expression specifying detailed evaluation strategy is called an Evaluation Plan.

Annotation:
↳ what to do at a particular stage is given/explained, stored in data catalog.

- Amongst all equivalent expressions, try to choose the one with cheapest possible evaluation-plan. Cost estimate of a plan based on statistical information in the DBMS catalog.

★★ Catalog Information for Cost Estimation:

- n_r : number of tuples in relation r.
- b_r : number of blocks containing tuples of r.
- s_r : size of a tuple of r in bytes.
- f_r : blocking factor of r — i.e., the number of tuples of r that fit into one block.

- $V(A,r)$: number of distinct values that appear in r for attribute A; same as the size of $\Pi_A(r)$. (domain)
- $SC(A,r)$: selection cardinality of attribute A of relation r; average number of records that satisfy equality on A.
- If tuples of r are stored together physically in a file, then:

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

Ex: For $\sigma_{A=10}(X)$ & A is primary key.

Selectivity $s = \frac{1}{card(X)}$

card = cardinality = No. of records in X.

- RAID → Storage ← Reliability — Bit Parity.
  Performance — Replication.

Search ← Retrieve — File Scan — property of secondary storage
↳ locate data records — Index Scan
retrieve the data.

Tape x
Disk ✓ (DASD)

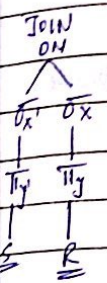
Property of Secondary Storage = I/O ← slow ← mech. devices
voluminous
monolithic ↑ usually seq. / indep seq. ✓.

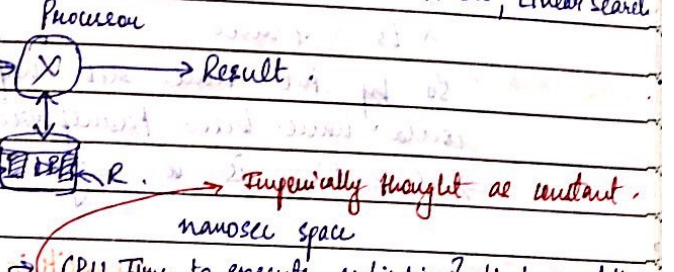
Key = contiguous ordered.
Key + Pointer = non-consecutive ordered.

Direct Access storage device (DASD)

Searching on Ordered data — Linear Search
Binary Search.

Searching on Unordered data — B+ Tree
Hash, Linear Search





(i). If OLTP is on centralized server, network is not a factor anymore.

(ii) CPU Time — taken as constant.
⇒ CPU time disregarded.

(iii) from (i) & (ii) I/O becomes main factor.
largest of all time.

I/O — ① Seek Time : $t_S \cong 4$ msec.

② Transfer time : $t_T \cong 0.1$ msec.
= time to transfer one block of data between disk & CPU.

data transfer rate for disk = 40 MB/sec.
+ Blocking factor
on Block size = 4KB.

Estimates as per 2016.

• Observation:

t_T = 0.1 msec.

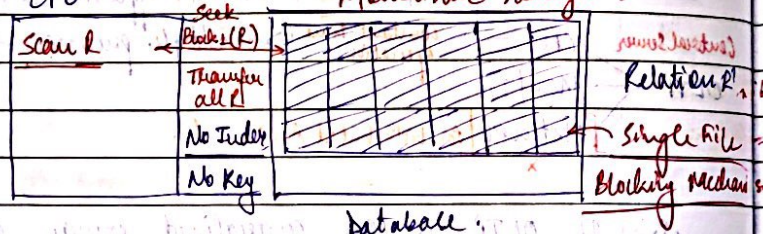
10 blocks = 1 msec.

40 blocks = 4 msec.

& t_S = 4 msec.

So by the time seek happens 40 blocks could have been transferred.

⇒ Seek time is a major factor.



Consider: No index, No key

Data is stored contiguously.

FILESCANS: A1: Linear Search (Brute Force)

Access the whole file to locate the required records satisfying selection condition σ_F in $\boxed{\sigma_F(R)}$.

$\boxed{cost_{A1} = t_S + b_R * t_T}$ — locating & searching cost

only 1 t_S as ∵ data is contiguously stored, we need to only locate / seek the block 1.

* [Write out cost is not included].

{Write out cost = "cost of writing out" the results}.

• A1*: = Linear Search with Equality on Key

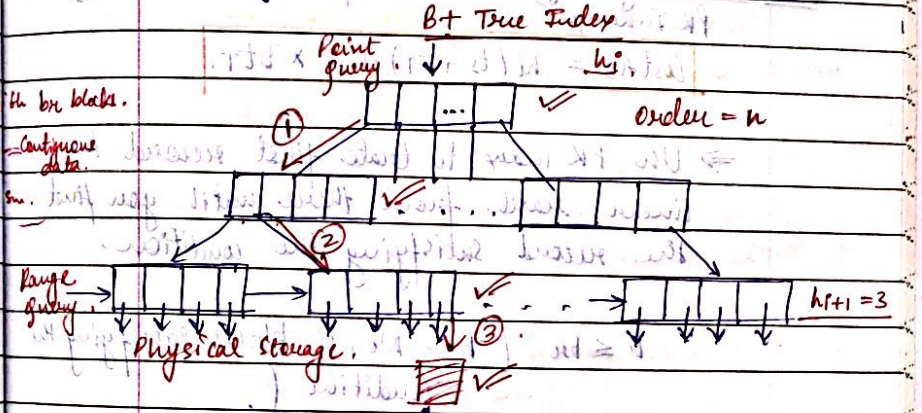
No Index, Key Exists. $\sigma_{Key=val(R)}$

Data stored on key.

[Key = Unique value].

$\boxed{Cost_{A1}* = t_S + \lceil b_R/2 \rceil * t_T}$ average estimate

Key equality record may be present on 1st position or last position, hence take average



To locate this there are 4 block accesses.

Total Seeks = 3 = $h_i + 1$.

Main memory contains the Root block atleast. Any other blocks that are needed have to be transferred from Secondary storage to Main Memory.

Index Scan • A2: = Primary Key. with Equality on Key.

↳ ordered Physically. $\sigma_{Key = 2050}$.

Types of selections — $\sigma_{A=x}$, $\sigma_{A<x}$,

$\sigma_{A=x \wedge B=c}$, ⇒ CNF

Page No. Date

clustering Index. → PK Index — B+ Tree.

Fetching ONE record only

Assumption: ROOT Node is Online. (Main memory).

$Cost_{A2} = (hi+1)(ts+tT)$

Brings in all clusters = Clustering Index (Not Clustered Index)

[In today's world 2020, the memories are huge & all indices can be online then cost will be $Cost_{A2} = ts + tT$]

A3 Primary key with Equality on Non key.

PK Index,

$Cost_{A3} = hi(ts+tT) + b * BtT.$

⇒ Use PK index to locate first record then linear search from there until you find the record satisfying the condition

$b \leq br$ [b = No. of blocks satisfying the condition].

- Index Scan: Search algorithms that use an index.
  - ↳ Select condition must be on search-key of index.

Ex: $\sigma_{B=4}(R)$ | index on A.

Index B+ Tree → PK Attr	Non key. (A3)
A	B.
10	5
20	4
30	5
40	4
50	4
60	4
70.	5

$hi(ts+tT) + ts \times tT * b.$

A4 (Secondary index, equality on nonkey).

↳ Retrieve a single record if the search key is a candidate key.

▶ Cost = $(hi+1) * (tT + ts)$.

↳ Retrieve multiple records if search-key is not a candidate key.

↳ each of n matching records may be on a different block.

▶ Cost = $(hi+n) * (tT+ts)$

↳ Most Expensive Operations.

★ Materialization (External Merge Sort):

In processing, if intermediate results are stored, it is known as Materialization

P.T.O.

CBO — Oracle 9i, 10g + | SQL Server. (Cost Based optimization)
HBO — Oracle 7i, 8i | IBM DB. (Heuristic Based optimization).

Page No. Date

★★ JOIN OPERATION:

Join operates / involves Cartesian Product as its core.

Cartesian Product is most expensive & expensive.

1. Simple Nested Loop Join
2. Block —"—
3. Indexed —"—
4. Merge - Join.
5. Hash - Join

Ex: Relation: Student ⋈ takes.

records = 5000 # records = 10000.

records = 100. # records = 400.

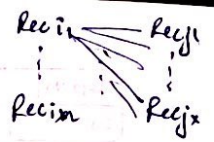
f_R = 50 records/block. f_S = 25 rec/block.

1. Simple Nested Loop Join:

File scan | No Index.

```
for each tuple tr in r do begin
    for each tuple ts in s do begin
        test pair (ts, tr) to see if they
        satisfy the join condition θ.
        if they do, add tr * ts to
            the result.
    end
end.
```

- r ⋈θ s
- θ = any condition
- Equality or non-equality
- tuple-by-tuple (not block by block)
- * fetch + processing.
- r = outer relation
  s = inner relation
- One at a time processing



For all Reci.

Page No. Date

- Requires no to Indices & works on any condition
- Worst case: If there is enough memory only to hold one block of each relation

estimated cost = $n_r * b_s + b_r$

b_r = blocks of r n_r = No. of records in 'r'.
b_s = blocks of s. n_s = No. of records in 's'.

Block Transfers = $n_r * b_s + b_r$.

Seek Time = $n_r + b_r$.

In above Eg:

~~BT = 5000 * 400 + 100 = 2000 + 100~~

Student ⋈ takes,

In above eqn,

BT = 5000 * 400 + 100 = 2000000 + 100 = 2,00,100

Seeks = 5000 + 100 = 5100 Seeks.

BT time = 2000100 * 0.1 msec

Seek time = 5100 * 4 msec

⇒ takes ⋈ Student.

BT = 10000 * 100 + 400 = 1000000 + 400
= 1000400.

Seeks = 10000 + 400 = 10400

BT Time = 1000400, 10.1 msec

Seeks Time = 10400 * 4 msec

* If the smaller relation fits entirely in memory, use that as the inner relation.

⇒ BT = $b_s + b_r$.

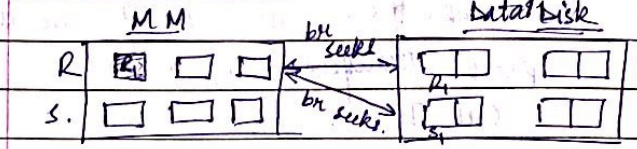
Seeks = $1 + b_r$.

Dis. — Due to 1-record-at-a-time processing.
Adv — No index needed.

This is the improvement.

- Record-at-a-time "tuple-by-tuple".
  - ↳ fetching data — physical unit of transfer = block

★ Block Nested Join:



R_1 join with S_1, S_2, S_3, S_4.. go back.

R_2 join with S_1, S_2, S_3, S_4.

i.e., S is seeked b_R times.

& R is seeked b_R times.

Seeks = $b_R + b_R = 2b_R$.

Block Transfer = $b_R + b_R * b_S$

four fors.

1. Worst case: Only one block/relation is available

Cost of $BNJ_{BT} = b_R + b_R * b_S$ blocks (R is outer relation).

$BNJ_{seeks} = 2 * b_R$ seeks. when R&S are on same platter & cylinder.

Average case

2. Enough blocks to hold the smaller relation, 's'.

$BNJ_{BT} = b_R + b_S$ blocks. (Once for S & one by one for R).

$BNJ_{seek} = 2$ seeks. when R&S are on ~~different~~ platter.

BEST

3. Enough memory to hold both relations:

$BNJ_{BT} = b_R + b_S$ transfers. (Once for R&S).

$BNJ_{seeks} = 2$.

— This is beneficial over intermediate case when in a later query you need 'R' which will already be present in the Main Memory.

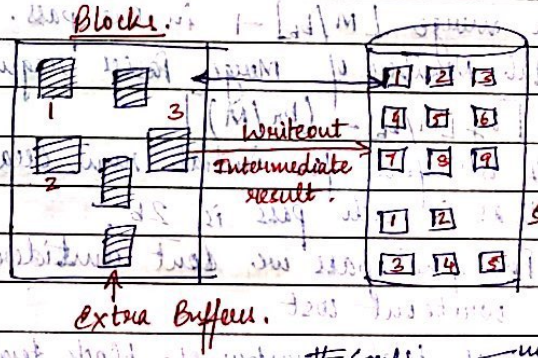
Imp — Still time of Intermediate case is greater than time of Best case as

In Best case, b_R transfers at once

In Intermediate case, b_R transfers in parts.

★ Merge Sort: (K-way Merge sort):

— It is an external sort. when K=2, it is Binary Merge sort.



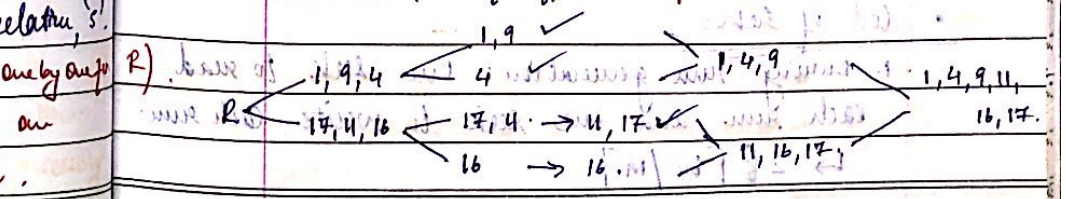
Merge sort: — # Sorting — no. of runs.

— # Merging. Capacity of MM = M

M = 3 { In/Buffers = 2 }
{ out/Buffers = 1 }

R = 1, 9, 4, 17, 11, 6.

S = 19, 3, 7, 2, 8, 5.



If $M < 3$, Merge Sort cannot be performed.

Page No. Date

— Similarly we get S → 2, 3, 5, 7, 8, 19. (S)

Merge R & S:

1, 2, 3, 4, 5, 7, 8, 9, 11, 16, 17, 19.

External Merge Sort:

1. Let M denote memory size (in pages).
   ⇒ M−1 are used as Input & 1 as output.

— Merge sort creates temporary relations on Disk. (Materialization).

1. b_b = No. of Input buffers
2. Can merge $\lfloor M/b_b \rfloor - 1$ in one pass.
3. Total Number of Merge Passes required.
   : $\lceil \log_{\lfloor M/b_b \rfloor - 1} (b_R/M) \rceil$
4. Block transfers for initial run creation as well as in each pass is $2b_r$.
   ↳ For final pass we don't consider the write-out cost.
   ↳ Thus total number of block transfers for external sorting:
   $b_R \left(2 \lceil \log_{\lfloor M/b_b \rfloor - 1} (b_R/M) \rceil + 1\right)$.

- **Cost of Seeks:**

— 1. During run generation: one seek to read each run and one seek to write each run
   ↳ $2 \lceil b_R/M \rceil$

2. During the Merge phase.
   → Need $2 \lceil b_R/b_b \rceil$ seeks for each merge pass.
   — Except for the last which is not counted.
3. Total Number of Seeks:
   $2\lceil b_R/M \rceil + \lceil b_R/b_b \rceil \left(2 \lceil \log_{\lfloor M/b_b \rfloor - 1} (b_R/M) \rceil + 1\right)$

M = No. of Blocks in Main Memory allocated for sorting & Merging.
M−1 for Input & 1 for Output.

- **Phases:**
  1. Sorting Phase ← no. of sorted runs. ← read unsorted runs / write-out sorted runs
  2. Merging Phase — read R and S; put in order, write out Output buffer & store on Disk.

- **Sorting Phase:**
  Each relation R → One for Read & One for write
  — read/write R = $2 * b_R$ Blocks.
  Total no. of runs = $\lceil \log_{M-1} (b_R/M) \rceil$
  → Both relations R & S each block of R will be read & written out.
  # total BT := $b_R \left(2 \lceil \log_{M-1} (b_R/M) \rceil + 1\right)$.
  # runs — temporarily stored on disk
  SORT.
  Cost of Reading R for first time, discard disk
  the cost of final sorted output write-out ×

seek$_{SORT}$:= $2\lceil b_R/M \rceil + \lceil b_R/b_b \rceil \left(2\lceil \log_{M-1}(b_R/M) \rceil - 1\right)$.

b_b ⇒ No. of blocks allocated for each run R.

When M=3, $b_b = 1$

1 run

I/P . O/P.

1 run

Ex: Records in R = N_R = 5000 f_R = 50. Assume M=3

Records in S = N_S = 10000 f_S = 25.

#blocks (R) = b_R = 100. $b_R = \lceil N_R / f_R \rceil$.

block (S) = b_S = 400

→ SORT-MERGE JOIN:

sort R	sort S.
# Block Transfers	# Block Transfers.
:= $b_R(2\lceil \log_{M-1} b_R/M \rceil + 1)$ + b_R (writeout).	:= $b_S(2\lceil \log_2 (400/3) + 1) + b_S$
:= $100(2\lceil \log_2 100/3 \rceil + 1) + b_R$	= $400(2\lceil \log_2 (400/3) + 1) + 400$
= 100 (2*6 +1) +400 = 1400	= 7200 blocks.
$b_b = 1$	$b_b = 1$
# seeks$_R$ = $2\lceil 100/3 \rceil + \lceil 100/1 \rceil$	# seeks$_S$ = $2\lceil 400/3 \rceil + \lceil 400/1 \rceil$
$(2(\log_2 100/3) - 1) + b_R$	$(2(\log_2 400/3) - 1) + 400$
= 2 * 34 + 100 (11) + 100 = 1268 seeks.	= 2 * 134 + 400 (8) + 400
	= 6668 seeks.

Final cost = 1400 + 7200

Verification of blocks as in NRDO. Validation.

BT = 1400 + 7200 + 400 + 100 = 9100 blocks

seeks = 1268 + 6668 + 2(b_R + b_S)

+ 2(400 + 100) = 8936 seeks

2 times — 1 for writeout / 1 for validation

Ex: Consider Relation R & S having 19200 and 54000 records resp. One block of R can store 24 records while One block of S can store 90 records. Compute the cost of Nested Loop Join, Block Nested - Loop Join and Sort Merge Join, in terms of total number of Block transfers & total number of seeks, when the number of memory buffers available are (i) 03 (ii) 20. — Kouth. M<600, M>600 smaller relation cannot fit in memory.

** HASH-JOIN (COST ANALYSIS):

- Query Optimization Phase.
  - ↳ Query Plan (Hash Join)

$R \bowtie_{A=B} S$ — Equi Join. $R \bowtie S$ — Natural Join.

↑ One CNF of equality condition

R.A = S.A and R.B = S.B.

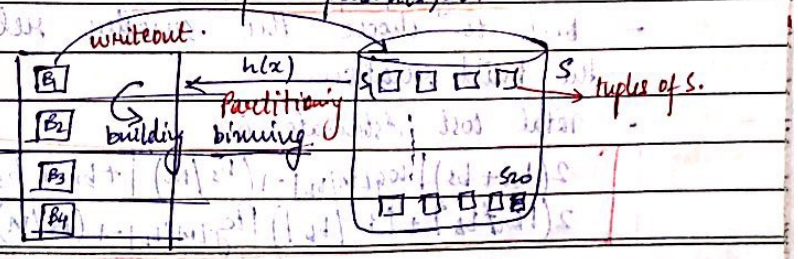
Build Relation ← Building Hash.

Probe Relation ← Probing Hash (Comparison).

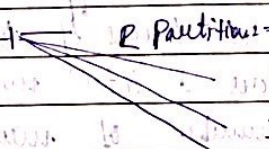
smaller relation — that can be put in main memory fully.

larger Relation. — Stays in secondary memory.

One full depend h(x)=0.



• Recursive Partitioning — If main memory can't hold any of the two relations completely Partition by Partition of each relation is brought in.



S – Partition – 2 R – Partition = 1
2
3
4

Case 1: If recursive partitioning is not required

$$3(b_R + b_S) + 4 * n_h \text{ block transfers} + 2(\lceil b_R/b_b \rceil + \lceil b_S/b_b \rceil) \text{ seeks.}$$

Case 2: If partitioning is required.

— No. of passes required for partitioning build relation. s to less than M blocks per partition is

$$\lceil \log_{\lfloor M/b_b \rfloor - 1}(b_s/M) \rceil$$

— Best to choose the smaller relation as the build relation.

— Total cost estimate is:

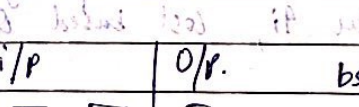
$$2(b_R + b_S)\lceil \log_{\lfloor M/b_b \rfloor - 1}(b_s/M) \rceil + b_R + b_S \text{ block transfers} + 2(\lceil b_R/b_b \rceil + \lceil b_S/b_b \rceil)\lceil \log_{\lfloor M/b_b \rfloor - 1}(b_s/M) \rceil \text{ seeks.}$$

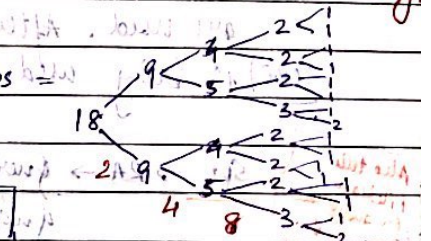
$b_b = 2$ is sufficient.
↳ 1 for I/P
↳ 1 for O/P.

Case 3: If the entire build input can be kept in main memory no partitioning is required
• Cost estimate goes down to $b_R + b_S$.

b_b ← No. of buffers / mem.
M ← Total No. of buffers in M.
b_s ← Smaller relation.
b_b = M / No. of parallel bins
$b_s > M$ ← Recursive Partitioning
$b_s \leq M$ ← Non-Recursive Partitioning

$b_b = 3$, M = 6.

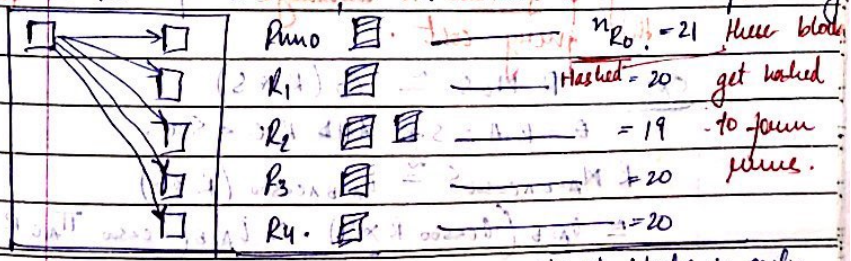




$$\lceil \log_{\lfloor M/b_b \rfloor - 1}(b_s/M) \rceil$$

Ex: M = 20 blocks. i = instructor
b_i = 100 blocks b_t = 400 blocks. t = teacher.
build (small relation) probe relation.

Step 1: Partition instructor into 5 partitions of 20 size each. 5 I/P + 1 O/P = M = 20 blocks. dont need recursive partitioning.



Runs.

No. of blocks in each read, given $b_b = 3$.

· Memoization
↳ Dynamic Prog
-ramming.

· Materialization

· Pipelining

Page No.
Date

Total cost = 3 (100 + 400) = 1500 block transfers + 2 (⌈100/3⌉ + ⌈400/3⌉) seeks. (4 * n_r)

= 1500 + 2 (34 + 134.

= 1500 + 336

= 1836.

negligible

★ Query Processing: Optimization: — minimize and delay joins & unions as far as possible.
around ROOT — at ROOT of expression.

Till Oracle 9i, Heuristic based Optimization (HBO) are used. After 9i, cost based Optimization (CBO) is being used

Also tells Fileshare or Index scan
Materialization
Pipelining

SQL → RA → Query Plan
(query Tree)

Multiple Query Plans. (CBO)

annotated Query Plan

Relation i Statistics (Ri)

Info about index

leaf Node are Relations
intermediate node = RA operators ⋈ | ∪ | ∩ | π | σ.

★ Equivalence of Transformation ⇒ Organise operand & Restructuring of QE Plan to minimize operations. benefits on Query load.
the query cost.

Ex: $R \bowtie_\theta S \cong \sigma_\theta (R \times S)$

$\theta = R.A = S.B$ AND $R.C < 500$.

$R \bowtie_{A=B \wedge C<500} S \cong \sigma_{A=B \wedge C<500} (R \times S)$

$\cong \sigma_{A=B} (\sigma_{C<500} R \times S) \cdot \cong \sigma_{A=B} (\sigma_{C<500} \pi_{A,C} R \times \pi_B S)$

$\frac{n*m-1}{2}$.

Page No.
Date

$\bowtie_\theta \cong \sigma_{A=B}$.

— JOIN

R S

σ π_B.

$\pi_{A,C}$ S.

R

$(R \bowtie S) \bowtie T \quad \cong \quad R \bowtie (S \bowtie T)$. — equivalence of Relation.

• An equivalence rule says that expressions of two forms are equivalent.
— can replace expression of first form with second and vice versa.

UNIT 5 – TRANSACTION CONTROL.

(a)

T_1
read_item(X);
X := X − N;
write_item(X);
read_item(Y);
Y := Y + N
write_item(Y);

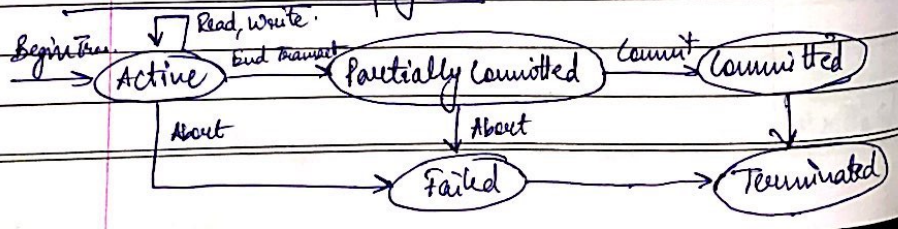
(b)

T_2
read_item(X);
X := X + M.
write_item(X);

P_1 — Lost Update Problem.
P_2 — Dirty Read Problem. (Uncommitted Problem)
P_3 — Inconsistent data.
P_4 — Unrepeatable read problem.

★ FAILURES:
1. Computer Failure.
2. Transaction or System Error.
3. Logical Errors or Exception.
4. Concurrency Control Enforcement
5. Disk failure.
6. Physical Problems & Catastrophes.

• Transaction lifecycle.



★ Transaction States:

★ Transaction Properties:
– ACID, Atomicity, Consistency, Isolation, Durability

• System Log or Journal:
↳ Used for recovery purposes & stored on Disks.

★★ SCHEDULES: (Elmasri – Navathe).
• Conflicting operations:
1. Belong to different transactions
2. They access same item X,
3. one of the operations is a write_item(X).

– Read after Write.
– Write after Read.
– Write after write.

★ Recoverability:

★ CONCURRENCY CONTROL:
– Lock: set / Unset.
↳ Binary.
↳ Shared / Exclusive. [read / write] locks.

★ Database Recovery:

- Set of mechanism for database recovery from failures.
- **Deferred Update:** (No Undo / Redo)
  Logs are forcefully written and DB is not updated until commit statement is not reached.
  - No-Undo as DB not updated until commit.
  - When crash happens, things in log files are redone to take DB back to its actual state.
  - At a checkpoint all the logs are written to the DB. So recovery can be done from that checkpoint further.
- **Immediate Update:** Force writes. DB.