GATE exam 2025.

last date to apply: August-10
exam date to CS: February

Let's start preparing
and aim high.

JANUARY - 2024

Wk	M	T	W	T	F	S	S
01	1	2	3	4	5	6	7
02	8	9	10	11	12	13	14
03	15	16	17	18	19	20	21
04	22	23	24	25	26	27	28
05	29	30	31				

Success is stumbling from failure to failure with no loss of enthusiasm. - Winston Churchill

Syllabus:

Section 1: Engineering Mathematics

- Discrete Mathematics: Propositional and First Order Logic, Sets, Relations, Functions, Partial orders and Lattices. Monoids, Groups, Graph: connectivity, matching, coloring. combinatorics: counting, recurrence relations, generating functions.

- Linear Algebra: Matrices, Determinants, System of linear equations, eigenvalues and eigenvectors, LU decomposition.

- calculus: Limits, continuity & Differentiability, Maxima and Minima, Mean value theorem. Integration.

- Probability & statistics: Random variables, Uniform, Normal, Exponential, Poisson & Binomial, Mean, Median, mode & standard deviation. conditional Probability & Bayes theorem.

Computer Science:

Section 2: Digital Logic: Boolean Algebra, combinational and sequential circuits. Minimization. Number representations and computer arithmetic (fixed & floating point).

Section 3: computer Organization & Architecture: Machine Instructions and Addressing modes, ALU, Datapath & control unit. Instruction pipelining pipeline hazards. Memory Hierarchy: cache, main memory & secondary stage; I/O Interface (Interrupt & DMA mode).

FEBRUARY - 2024

M	T	W	T	F	S	S	Wk
			1	2	3	4	05
5	6	7	8	9	10	11	06
12	13	14	15	16	17	18	07
19	20	21	22	23	24	25	08
26	27	28	29				09

Have no fear of perfection - you'll never reach it. - Salvador Dali

Section 4: Programming & Data Structures:
Programming in C, Recursion, Arrays, Stacks, Queues, Linked lists, trees, BST, Binary Heaps, Graphs

Section 5: Algorithms: Searching, Sorting, Hashing, Asymptotic, worst case time & space complexity, Design techniques; greedy, dynamic programming & Divide & conquer. Graph Traversals, Minimum Spanning Trees, Shortest paths

Section 6: Theory of Computation: Regular Expression, Finite Automata, CFG and Push Down Automata, Regular & context Free Languages, Pumping Lemma. Turing Machines and Undecidability.

Section 7: Compiler Design: Lexical Analysis, Parsing, Syntax Directed Translation. Runtime Environments, Intermediate Code Generation. Local optimization, Data flow Analysis, constant propagation, Liveness Analysis, common Subexpression Elimination.

Section 8: OS: System calls, processes, threads, IPC, concurrency & synchronisation, Deadlock, CPU and I/O Scheduling, Memory Management & Virtual Memory, File Systems

X Section 9: DBMS: ER, Relational Model: Relational Algebra, tuple calculus, SQL, Integrity constraints, Normal forms, File organisations, Indexing (B, B+) Transaction & concurrency control.

JANUARY - 2024

Wk	M	T	W	T	F	S	S
01	1	2	3	4	5	6	7
02	8	9	10	11	12	13	14
03	15	16	17	18	19	20	21
04	22	23	24	25	26	27	28
05	29	30	31				

Kites rise highest against the wind not with it. - Winston Churchill

Computer Networks : concept of layering : OSI , TCP/IP stack ; Packet, circuit , virtual circuit switching ; Data link layer: Framing, Error Detection, MAC, Ethernet Bridging, Routing protocols : shortest path, flooding, distance vector & link state routing; Fragmentation & IP addressing , IPv4 CIDR notation , Basics of IP protocols (ARP, DHCP, ICMP) , Network Address Translation (NAT) ; Transport Layer : Flow control & congestion control ; UDP, TCP , sockets : Application Layer : DNS, SMTP, HTTP, FTP , Email.

Books for each subject :

Algorithms : Introduction to Algorithms (Ed. 3) Thomas, Charles, Ronald, Clifford
Computer Networks : Computer Networking : A Top Down Approach ; Kurose, Ross
Operating Systems : OS concepts , Silberschatz
Databases : Database concepts , Silberschatz
Theory of computation : Introduction to Formal Languages , Peter Linz
Computer Organization & Architecture : Computer Org & Arch. , Hamacher
Digital Logic : Digital logic & Computer Design , Morris Mano.
Compiler Logic : Compilers , Aho, Ulman.
Data Structures : Debasis Samanta , Classic Data Structures
Engineering Mathematics : Higher Engineering Mathematics , B.S. Grewal
Aptitude : Verbal & Non Verbal Reasoning , R.S. Agarwal
Quantitative Aptitude , R.S. Agarwal.

Friday 05 January 2024

Wk 01 (005-361)

Plan:

May (10 Days): Database Management Systems.

June

July

August

September

October

November

December

January : Revise & Mock Exams.

February : Revise & Mock Exams

JANUARY - 2024

Wk	M	T	W	T	F	S	S
01	1	2	3	4	5		

Database Management Systems:

A Database System: collection of interrelated data & set of programs to access those data. Primary goal: Provide store and retrieve database information that is both convenient & efficient, & manage large, complex collection of data.

- Manage Large Bodies of Information - Management :- Defining structures for storage and manipulation. Also provide storage safety regardless of system crashes & attempts to Hack - Avoid Anamalous Results
- First Database system in 1960s, central Aspect: Data Stored.

Store: *) Highly valuable Information
*) Relatively Large "
*) Shared Information.

Modes in Database to be used:
*) Online Transaction Processing (Execute Queries).
*) Data Analytics (Processing of data to Draw conclusions).

Purpose of DBMS: *) Data Redundancy & Inconsistency
*) Difficulty in Accessing Data. (More Responsive Data Retrieval System required).
*) Data Isolation
*) Integrity problem (Satisfy certain consistency constraints)
*) Atomicity Problem (operations must be atomic)
*) concurrent Access Anomalies
*) Security problems.
*) Provide Abstract view of data.

Data Model: *) Relational Model (Tables)
Represent both data & relationship among data.
Record Based Model, fixed format, most widely used.

FEBRUARY - 2024

M	T	W	T	F	S	S	Wk
			1	2	3	4	05
5	6	7	8	9	10	11	06
12	13	14	15	16	17	18	07
19	20	21	22	23	24	25	08
26	27	28	29				09

Entity Relationship Model: collection of basic objects (entities and relationships between objects, widely used in database design

Semi Structured Data Model: JSON & XML, same type can have different sets of attributes.

Object Based Data Model: Store objects & procedures to be executed by the database (Encapsulation, Methods, object Identity).

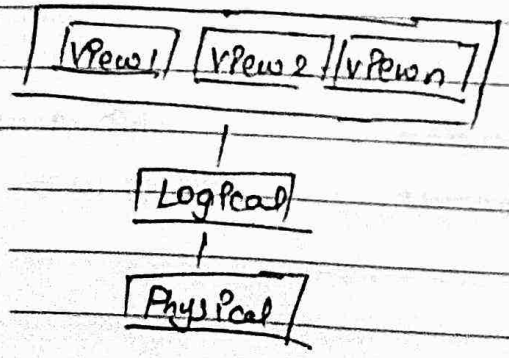
Relational Model: Multiple Tables - Multiple columns - ~~multi~~ unique name for each column.

Data Abstractions: Physical level, Logical level, View Level

Physical - How Data is actually stored
- Describes low level data structures

Logical - What data is stored
- What relationships exist among data
- Also known as Physical Data Independence
- Data Base Admin use this layer

View Level: Highest level of abstraction
- Only part of system accessible to end users.
- Simplifies the existence & interaction b/w systems.



JANUARY - 2024

Wk	M	T	W	T	F	S	S
01	1	2	3	4	5	6	7
02	8	9	10	11	12	13	14
03	15	16	17	18	19	20	21
04	22	23	24	25	26	27	28
05	29	30	31				

Index: Data Structure for efficient retrieval of data from records.
- In physical level.

Users access some or all the views of an data. Provides complex operation and protection against access ~~from~~ to Database.

Instances: collection of information stored in the database at a particular moment

Schema: Overall design of the database is called the schema.
↳ Physical Schema: Database Design at 'Physical level'
↳ Logical Schema: Database Design at 'Logical Level'
↳ Subschemas: can have several schemas at 'View Level' to describe different views.

Logical Schema: More Important

Database Languages:
* Data Definition language: To specify schema of DB
↳ Specify additional properties of data.
↳ Specify storage structure & access methods using Data storage & Definition language.
↳ Provides facilities to specify constraints (arbitrary predicates)
↳ Must have minimal overhead.
* Domain constraints: Specify common data type, Easily Tested Most Elementary form of constraint.
* Referential Integrity: To Ensure that a value appears in both relations.
↳ can cause violations in Referential Integrity due to data modification.
↳ if violation occurs, reject the action that causes violation.

FEBRUARY - 2024

M	T	W	T	F	S	S	Wk
			1	2	3	4	05
5	6	7	8	9	10	11	06
12	13	14	15	16	17	18	07
19	20	21	22	23	24	25	08
26	27	28	29				09

Authorization: * Restrict users' access to data in database
* Read Auth., Insert Auth., Update Auth., Delete Auth.
* can assign a combination, all or none of the above.

Output of DDL stored in "Data Dictionary" - contains metadata about Relations, can only be accessed by database system.
DB uses / consults data dictionary before reading / modifying

Data Manipulation language: A language that enables users to access or manipulate data as organized by the data model.
* Types: Retrieval of Information (Access)
Insertion of "
Deletion of "
Modification of ".
* Types of DML:
↳ Procedural: what data is needed and how to get the data
↳ Declarative: only specify what data is needed.

Query: Statement to retrieve information, query language = DML

SQL: Not Turing complete and therefore we use applications that query database and interact with it.
* using an API to send DML & DDL queries
* Standardized by ODBC (open database connectivity), for C
* " JDBC (Java " " ") for Java

DataBase Engine : Different Modules : Storage Manager, Query processor, Transaction Manager.

Storage Manager : Databases are large, specify how data is stored in disks.

Query Processor : Helps to simplify and facilitate access to data.

*) allows users to obtain good perf. without understanding or changing anything in the physical level.

*) converts non procedural queries in logical level to procedural efficient sequence of operations in physical level.

Transaction Manager : Enables data to be atomic

*) Abstracts over concurrent access and system failures

*) P

Parallel Processing & Storage is implemented in modern DB.

Storage Manager : Component that provides interface between database and the file system

*) Translates DML statements to low-level file system commands.

*) Responsible for storing, retrieving & updating.

Components :

↳ Authorization & Integrity manager

↳ Transaction Manager

↳ File Manager : Allocation of space on disk and Data structures

↳ Buffer Manager : Fetch data to main memory for cache.

Data Structures of File Manager : *) Data Files,

*) Data Dictionary, Metadata

*) Indices, Fast access

Query Processor components: *) DDL Interpreter: Interprets DDL statements and records definitions in Data Dictionary.
*) DML compiler: Translate DML to low level instructions that can be understood by the query evaluation engine.
*) Query optimization is also done.
*) Query Evaluation Engine: Executes low level instructions from DML compiler

Transaction Manager: It guarntees: Atomicity
Consistency
Durability (Existence of values after query, system failure).

Transaction: Collection of operations to perform a single logical function in a DB.
*) Unit of Atomicity & consistency.
*) Database must be consistent before/after a transaction is completed.
*) Sometimes necessary to go to a state of inconsistency. And if failure happens in inconsistency, it will lead to difficulty.

Recovery manager ensures atomicity & durability is ensured.
*) To ensure atomicity, a failed transaction must not have any effect on the database and must be restored to state before the transaction. (called as Failure Recovery) by detecting an error / system failure.

*) when transactions are done concurrently, consistency is not preserved even if transactions are correct, concurrency control manager controls the interaction among concurrent transactions to preserve consistency.

Database Administrator: central control over the system is called a Database Administrator.

- Schema Definition:
- Storage Structure & Access method definition (Indices)
- Schema & Physical Org. Modification.
- Granting of Authorization for data access
- Routine Maintenance:
  - Backup
  - Free Disk space.

Relational Algebra / Relational Model.

Relational Model - Tables (unique name) - represents a relationship among a set of values. Tuple - Row & Attribute - Column. Each Attribute has a set of permitted values called the Domain. Domains of all attributes of relation r be atomic. A Domain is atomic if it is made up of indivisible units.

- We also need to treat it as indivisible even if it has a single value.
- Null - special value, indicates non-existence of a value / unknown value.
- Not suitable for storing data that changes over time

Relation Schema: Consists of a list of attributes and their corresponding domains. Relation Instance: values of all rows in the given relation R.

Keys: Values of the attribute of a tuple such that they can uniquely identify tuples in the relation.

- No two tuples are allowed to have exactly same values for all attributes.

Superkey: Set of one or more attributes allow us to uniquely identify a tuple in a relation.

Don't let what you cannot do interfere with what you can do. - John R. Wooden

FEBRUARY 2024

M	T	W	T	F	S	S	Wk
			1	2	3	4	05
5	6	7	8	9	10	11	06
12	13	14	15	16	17	18	07
19	20	21	22	23	24	25	08
26	27	28	29				09

If K is a Superkey, so is any super set of K.

Superkeys for which no proper subset is superkey is called as a candidate key / minimal superkey.

Primary key : A candidate key chosen by the database admin for identifying tuples in a relation.

Represents a constraint in the realworld being modeled and also referred to as primary key constraints.

Should be chosen carefully, values should never or rarely change.

Foreign key constraint: An Attribute from relation r_1 is related to a primary key from relation r_2,

r_1 - Referencing Relation

r_2 - Referenced Relation.

This requirement for attribute in R_2 to be ~~a primary key~~ in Relation R_1 is called as Referential Integrity constraint.

Schema Diagrams : Name of Relation at top

Attributes in separate box.

Student
ID
Name
dep_name
total_cred

Query Language: Language for requesting information to Database.
- Higher level language.
*) Categories: Query language -
Imperative
functional (side effect free).
declarative.

"Pure" Query Languages:
*) Relational Algebra (Functional)
*) Tuple Relational calculus & Domain Relational calculus (Declarative).

Relational Algebra: Forms theoretical basis of SQL
*) Operations that take in one or two relations as input and produce a new relation for the result.
*) Eg: Unary - select, Project, Rename
Binary - Union, cartesian Product & set difference
*) Relational Algebra doesnot allow duplicate tuples, but practically db allows them.
*) can be extended to support multiset relations.

Select Operation: selects tuples satisfies a predicate.
using σ to denote selection
arguement; ~~Attributes to display~~ Relation to operate on

Eg: $\sigma_{dept = "physics"}$ (college_prof).

$\sigma_{dept = "physics" \wedge salary > 90000}$ (college_prof).

$\sigma_{deptname = building}$ (department).

FEBRUARY - 2024

M	T	W	T	F	S	S	Wk
			1	2	3	4	05
5	6	7	8	9	10	11	06
12	13	14	15	16	17	18	07
19	20	21	22	23	24	25	08
26	27	28	29				09

Project operation: ~~Su~~ To list all instructors' ID, name, salary but not deptname,

*) we select only the attributes we want to display.

*) Denoted by Π (upper pi) (Duplicate Rows are eliminated).

Π_{id} (instructor).

*) we can also perform operations
$\Pi_{id, name, salary/12}$ (instructor).

*) we can compose relational algebraic operations.

Cartesian Product: ~~Sp~~ Binary operation, denoted by cross (x) to combine two relations.

*) we also attach relation names for distinguishing duplicate attributes that may arise during "crossing".

*) contains $n(r_1) * n(r_2)$ tuples.

Join operation: Composition of select and cross product).

*) $\sigma_\theta (r \times s)$ or $r \bowtie_\theta s$

Union: Make them appended from one set to get another new set.

*) Denoted by '$\cup$'

Eg: $\Pi_{course_id, course_name} (\sigma_{sem = "even"} (dept)) \cup$
$\Pi_{course_id, course_name} (\sigma_{sem = "odd"} (dept))$

For union to make sense:
*) The Input Relations must have the same arity (no. of attributes).
*) The types of attributes in the same position must be same.

Relations that satisfy the conditions is called as compatible relations.

Intersection: Denoted by $\cap$
*) Relations must be compatible.

Set Difference: Denoted by -
Finds tuples present in 'r' but not in 's'.

Rename operation: Renames the result of an Expression to the name x.
*) Denoted by rho (ρ)
*) $\rho_x(E)$ or $\rho_{x(A_1, A_2, A_3 \ldots A_n)}(E)$.

Identical Queries: Same Result.
*) Query optimizers look at what an expression computes and finds an efficient way to compute the result.
*) Algebraic structure of relational algebra makes it easy to find efficient queries.

Database Design: we ensure that we avoid two major pitfalls.
*) Redundancy: Bad design may lead to repeating information. why it is bad - Redundant Information can become inconsistent / out of sync with each other.

FEBRUARY - 2024

M	T	W	T	F	S	S	Wk
			1	2	3	4	05
5	6	7	8	9	10	11	06
12	13	14	15	16	17	18	07
19	20	21	22	23	24	25	08
26	27	28	29				09

Incompleteness : A Bad Design can make certain aspects difficult or impossible to model.

Avoiding Bad Design is not Enough, we must also choose the best "good" design and apply to our schema.

ER Data Model : Facilitates Database Design by allowing specification of an schema that represents the overall logical structure of a database.

- Entity Sets
- Relationship sets
- Attributes.

Entity : An Entity is a thing / object in real world - distinguishable from all other objects

- Has a set of properties and values for each property and an uniquely identifying property.

Entity set : set of Entities that share the same properties.
Extension : set of all Entities.

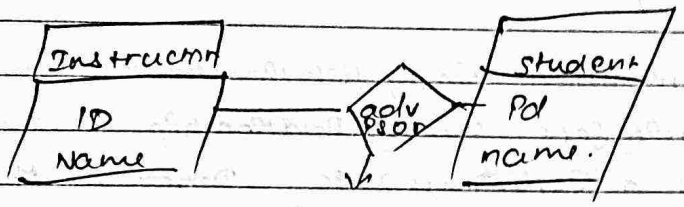
Represented by a set of attributes and diagrammatically by a rectangle.

Eg :

Student
ID
name
tot cred

→ Name of the Entity Set
→ Name of the Attributes.

Relationship : A Relationship is an association among several entities.
Relationship set is a set of relationship of the same type.
Relationship Instance represents an association between named entities.



Relationship
(Binary Relationship).

Formally, an Relationship set is a mathematical relation on $n \geq 2$ nondistinct entity sets

$$\{(e_1, e_2, e_3 \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2 \ldots e_n \in E_n\}.$$

Mapping Cardinalities : or also called as cardinality ratios, express the number of entities to which another entity can be associated
- Describe binary relationship sets

Association between entity sets is referred to as participation i.e. Entity sets $E_1, E_2 \ldots E_n$ participate in Relationship set R.

The Function that an entity plays in an relationship is called an Entity's Role. Entity sets participating in an relationship set are generally distinct roles are implicit and not specified.

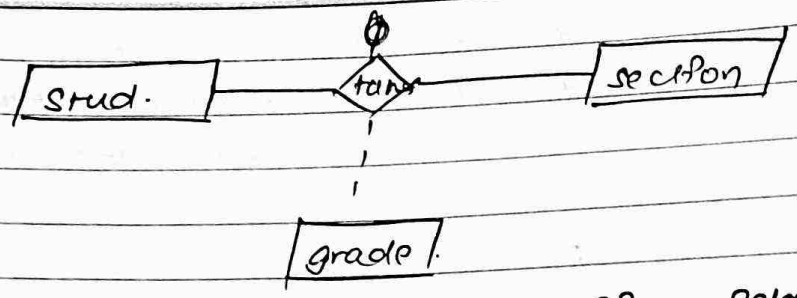
If the same entity can participate in the same relationship more than once, then it is called as an Recursive Relationship set. Explicit role names are necessary to specify how an entity participates in an instance.

A Relationship may also have attributes called descriptive attributes. An Attribute of a relationship set is represented as an undivided rectangle.

Happiness depends upon ourselves. - Aristotle

FEBRUARY - 2024

M	T	W	T	F	S	S	Wk
			1	2	3	4	05
5	6	7	8	9	10	11	06
12	13	14	15	16	17	18	07
19	20	21	22	23	24	25	08
26	27	28	29				09



Relationship set with 2 Entity Sets: Binary Relationship
" " " 3 Entity Sets: Ternary Relationship.
Number of Entity Sets of a Relationship set: Degree of the Relationship set.

Simple Attributes: Attributes that have been divided into subparts
Composite Attributes: Attributes that can be divided into subparts (other attributes). Composite Attributes helps us group together related attributes
Single Valued Attributes: Attributes with only one possible value for one entity, Eg: ID
Multi Valued Attributes: Attributes with multiple possible values for one entity, Eg: Phone_no.
Derived Attributes: Attributes that can be derived from other related attributes or entities. Eg: Neeye Yosp.

Attribute takes a NULL value if it does not have a value for it.

Mapping Cardinalities: Expresses the number of entities to which another entity can be associated via a relationship.
One to One: An Entity in A with at most one entity in B and an Entity with at most one entity in A.

JANUARY - 2024

Wk	M	T	W	T	F	S	S
01	1	2	3	4	5	6	7
02	8	9	10	11	12	13	14
03	15	16	17	18	19	20	21
04	22	23	24	25	26	27	28
05	29	30	31				

One to Many: An Entity A with (zero or more) of entity in B and Entity B with at most one entity A.

With mirth and laughter let o
wrinkles come. - William Shake

Many to One : An Entity A with at most one entity in B. An entity in B with zero or more entities in A.

Many to Many : An Entity A associated with any number (zero or more) of Entities in B and vice versa.

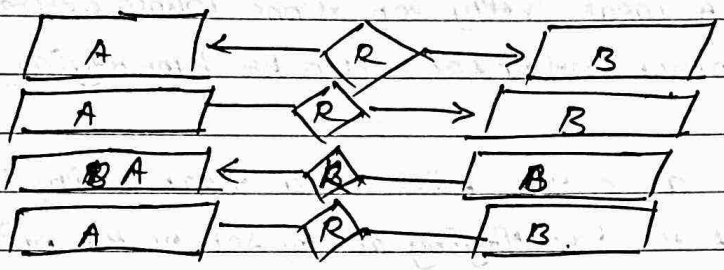
ER Representation:

One to One :

Many to One :

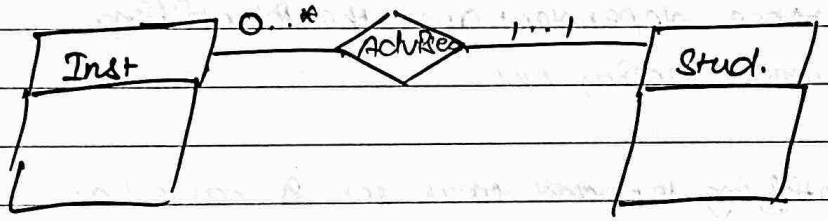
One to Many :

Many to Many :



If Every Entity in E participates in an Relationship set R, Total

If ~~Ever~~ Only some Entity only participate in an Relationship set R, Partial.



1..1 → lower limit 1, upper limit 1

0..* → lower limit 0, upper limit : No limit.

Primary Key for Relationship Sets :

If Relationship set R has entities sets $E_1, E_2 \dots E_n$ participating, then the primary key of R is represented as

$prim.(E_1) \cup prim.(E_2) \cup prim.(E_3) \cup \dots \cup prim.(E_n)$

If attr $prim.(E_1) \cup prim.(E_2) \cup prim(E_3) \cup \dots \cup prim.(E_n) \cup \{a_1, a_2 \dots a_n\}$ are present)

FEBRUARY - 2024

M	T	W	T	F	S	S	Wk
			1	2	3	4	05
5	6	7	8	9	10	11	06
12	13	14	15	16	17	18	07
19	20	21	22	23	24	25	08
26	27	28	29				09

Faithless is he that says farewell when the road darkens. - J.R.R. Tolkien

Many to Many Relation: PrimKey(E_1) ∪ PrimKey(E_2)
Many to One Relation: Prim. Key (Many Side of R)
One to Many Relation: "
One to One Relation: Prim. key(E_1) // Prim. key(E_2).

Weak Entity Set: A weak Entity set is one whose existence is dependent on another entity set called its identifying entity set.

we identify ~~the~~ an entity ~~with~~ in a weak entity set using the attributes of the identifying entity set or the primary key known as the Discriminator Attributes.

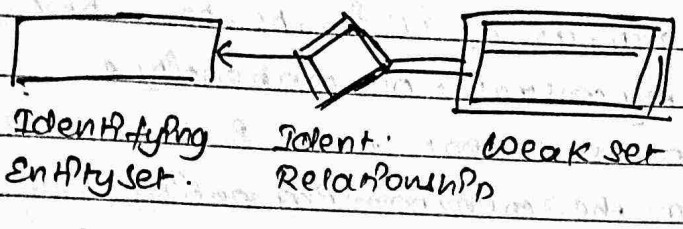
Entity set Not weak is strong entity set.
Every weak entity must have an identifying entity set.
and is said to be existence dependent and the identifying entity set owns the weak entity set.

Relationship between identifying set and weak set is called as identifying relationship. and its cardinality is Many to one

Identifying relationship should not have any ~~rela~~ attributes.

In ER Diagrams, weak Entity sets is depicted via a Double Rectangle with discriminator being underlined with a dashed line.

Identifying Relationship set with a double diamond
We also use double lines to indicate that the participation is total (every entity in weak entity set has an relationship).



Weak entity can participate in other relationship set.
A weak Entity set can own other weak entities
A weak Entity set can have more than one identifying entity sets, then the weak entity can be identified by some combination of entities one from each identifying set.

The primary key of the weak entity set then must be the union of the primary keys of the identifying sets, plus the discrimination of the weak entity set.

Strong Entity sets: We can simply use the attributes of the strong entity sets if they are simple. If they are composite, we replace them with their simple attributes
Eg: Name → First Name, Middle Name, Last Name.

Multivalued Attributes are stored as a separate schema
Derived Attributes are not explicitly stored but can be represented as stored procedures, functions or methods (if OOP model)

If M is a multivalued attribute ~~with M as an attr~~ in a relation R with a primary key K, then (K and M) is stored as an separate relation R_m.

Representation of weak Entity sets: We represent a weak Entity set A with attributes $\{A_1, A_2, A_3 .. A_n\}$ and B which owns A and has primary key $\{B_1, B_2, B_3 ... B_n\}$.

Success is getting what you want; happiness is wanting what you get. - Ingrid Bergman

FEBRUARY - 2024

M	T	W	T	F	S	S	Wk
			1	2	3	4	05
5	6	7	8	9	10	11	06
12	13	14	15	16	17	18	07
19	20	21	22	23	24	25	08
26	27	28	29				09

then the schema will be $\{a_1, a_2, a_3 \ldots a_n\} \cup \{b_1, b_2, b_3 \ldots b_n\}$
we also create a foreign key constraint on weak entity A referencing the values of B, to ensure that there is a corresponding tuple representing the corresponding entity.

Relationship sets: let the relationship set R has entity sets $\{E_1, E_2, \ldots E_n\}$ participating with primary key of the relationship set being $\{a_1, a_2, a_3 \ldots a_n\}$ and let ~~R have~~ R's descriptive attributes if any be represented as $\{b_1, b_2, \ldots b_n\}$

then the schema of R is

$$\{a_1, a_2, a_3 \ldots a_n\} \cup \{b_1, b_2, \ldots b_n\}$$

we also create a foreign key constraint that is referencing the participating entities E_i. ~~with the~~

We also specify an "ondelete cascade" to delete any entity of an entity set that references the entity to be deleted.

Redundancy of schemas: Schemas produced from the above mentioned algorithm will produce redundant schemas and practically, the redundancy is eliminated by combining the schemas together.

Combination: * If an relationship with cardinality many to one is from A to B with relationship set AB, ~~then A~~ and the entity set A is total in ~~the~~ participation, then the sets AB and A can be merged and the primary key is the primary key A.

* In one to one relationships, even if they are partial in participation they can be combined with null values in place.

* We then remove the foreign key constraint which references

the entity set to be merged.

Extended ER Features: Though, we can create a schema from normal ER features that can model the world perfectly on its own, we create ~~a schema with~~ some extensions just cause the sake of it (some features can ~~not~~ increase quality of life).

Specialization: An entity set can include subgroupings of entities and specialization can model that.

- Some attributes are not shared by all entities in the entity set.
- Eg: Person → Employee (Name, DOB, salary, Designation)
  → Student (" , " , grade, school)
- Often they can be both or neither or either one of them.
- Process of Designating subgroups within an entity set is called as specialization.
- Entity set can be specialized by more than one distinguishing feature.
- When more specializations are formed, an entity can belong to one more specialization.
- Depicted by Hollow Arrow Head Point and referred to as "Is a" relationship.
- If an entity set can have an entity to multiple specializations, then it is known as overlapping specialization.
- Else it is known as an Disjoint specialization.
- Two separate arrows are used for overlapping and a single arrow is used for an disjoint specialization.

Generalization: Bottom up approach to specialization and attributes that are conceptually the same across two entity sets can treated as the same using an general class.

- This commonality is defined as Generalization.
- Higher level entity sets : Superclass
- Lower " " " : subclass.

FEBRUARY - 2024

M	T	W	T	F	S	S	Wk
			1	2	3	4	05
5	6	7	8	9	10	11	06
12	13	14	15	16	17	18	07
19	20	21	22	23	24	25	08
26	27	28	29				09

*) Generalisation: simple inversion of specialisation and no distinguishment is done in terms of ER Diagrams.

*) Crucial property of higher-level & lower-level entities created by specialisation & generalisation is attribute inheritance.

*) Attributes are inherited by the lower level entity sets from higher level entity sets.

*) A lower level entity set also participates in all the relationship set that the higher level entity set participates in.

If a lower level entity has more than one relationship : ~~Higher level~~ Multiple inheritance

" " only one relationship with a higher level entity : single level inheritance.

In multiple levels inheritance, the resulting structure is known as a lattice.

Constraints: certain constraints are applied on particular generalisation or specialisation

*) One type is Disjoint / overlapping

*) Another type is completeness constraint, which specifies whether or not an entity in higher level set belongs to at least one of the lower entity set.

*) This can be : Total (Each higher level entity belongs to one lower level entity set)

(Default) Partial (Some higher level entity may not belong to a lower level entity set)

We can specify total specialisation in an ER Diagram, by adding the keyword total in the diagram and drawing a dashed line from the keyword to the corresponding hollow arrowhead to which it applies.

JANUARY - 2024

Wk	M	T	W	T	F	S	S
01	1	2	3	4	5	6	7
02	8	9	10	11	12	13	14
03	15	16	17	18	19	20	21
04	22	23	24	25	26	27	28
05	29	30	31				

The road to success is dotted with many tempting parking spaces. - Will Rogers

Aggregation: we cannot express relationships among relationships

*) An Abstraction in which relationships are treated as higher level entities.

Representation in ER Diagrams:

*) Generalization: Two ways of creating schemas

1.) Create a schema for the higher level entity set and for each lower level entity set create all the necessary attributes plus the attributes for the higher level entity (only the primary key)

↳ we also create an foreign key constraint on the lower level entity set referencing the primary key of the higher level entity set.

2.) If generalization is disjoint and complete, we create a schema for each lower level entity set, that includes all the attributes of the higher level entity set and the attributes of all the lower level entity set.

↳ Drawback: No Foreign key ~~pro~~ constraint.

*) Redundancy when overlapping records are present.

*) If the Generalization is disjoint but not complete, then we need an extra schema for representing the additional set that belongs to neither.

Representation of Aggregation: use the same method for representing relations ~~but~~ use the relation created from representing the higher level relationship set.

Design Issues in ER Diagrams:

common mistakes in ER Diagrams:

*) using a primary key of an entity set as an attribute of another entity instead of using relationship set.

FEBRUARY - 2024

M	T	W	T	F	S	S	Wk
			1	2	3	4	05
5	6	7	8	9	10	11	06
12	13	14	15	16	17	18	07
19	20	21	22	23	24	25	08
26	27	28	29				09

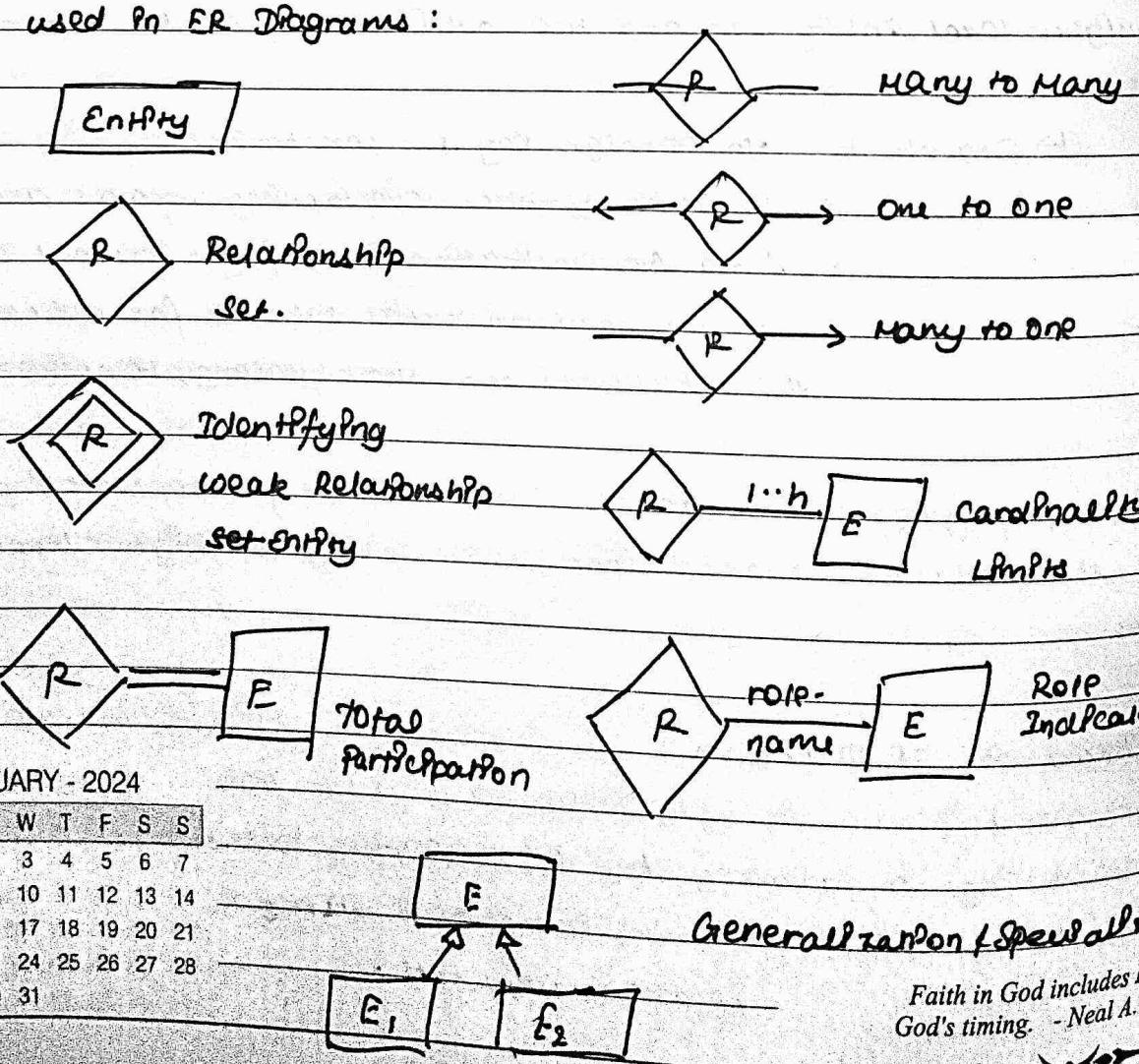
2) Another mistake - Designate the primary key attributes of the related entity sets as the attributes (descriptive) of the relationship set. (Primary key attributes are already implicit).

3.) use single valued attribute when multivalued attribute is required.

Any 'n-ary' relationship set can be turned into a sets of binary relationship sets but the resulting entity need to have a total participation.

Alternative Notations can be used such as UML Diagrams instead of using E-R Diagrams.

Symbols used in ER Diagrams :

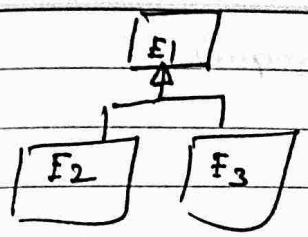


Generalization & Specialization

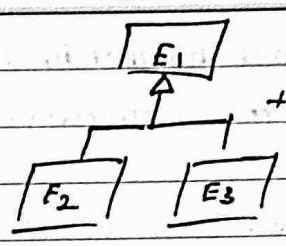
JANUARY - 2024

Wk	M	T	W	T	F	S	S
01	1	2	3	4	5	6	7
02	8	9	10	11	12	13	14
03	15	16	17	18	19	20	21
04	22	23	24	25	26	27	28
05	29	30	31				

Faith in God includes Faith in God's timing. - Neal A. Maxwell

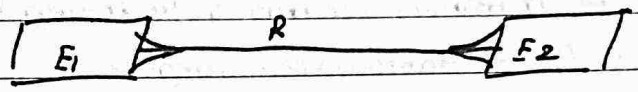


Disjoint Generalization

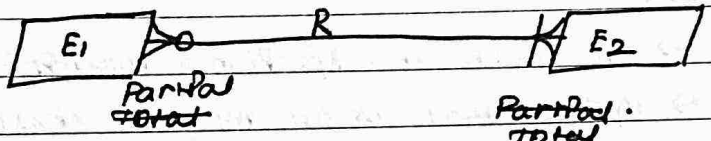


total.

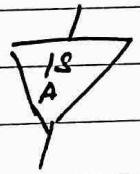
Alternative Notations:



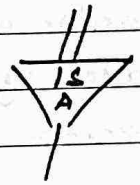
Many to Many.



Participation Participation



Generalization



Total Generalization.

UML: Unified Modeling Language.

↳ Class Diagram: Similar to an ER Diagram

↳ Use case Diagram: Shows the interaction between users and the system & the steps of tasks that users perform.

↳ Activity Diagram: Depicts the flow of tasks between various components of sy

↳ Implementation Diagram: Shows the system components and their interconnection at software & hardware level.

In UML, Relationships are referred to as Associations
We can also specify role names akin to how it is specified in ER.
Relationship sets are treated the same as an Entity in UML.
Cardinality is specified similarly to how it is specified in ER.

SQL: Developed by IBM in 1970, originally called as sequel but renamed to Structured Query Language.

Several Parts: DDL (Data Definition Language):
↳ Defining, Deleting & Modifying Relation Schemas.
DML (Data Manipulation Language):
↳ Query Information, Insert, Modify, Delete tuples.
Integrity:
↳ commands for specifying integrity constraints
↳ Updates that violate integrity constraints are disallowed.
View Definition
Transaction control:
Embedded/Dynamic SQL
Authorization.

Basic Data types:
- CHAR(N)
- VARCHAR(N)
- INT
- SMALLINT
- NUMERIC(P,D) (NUM(3,1) => 444.5, 420.6)
- REAL (FLOAT)
- DOUBLE
- FLOAT(n)

can have a special value NULL.
* NVARCHAR for multilingual varchar strings (explicit support)

Schema Definition:

Syntax: CREATE TABLE Relation (
ATTR1 DATATYPE1,
....
ATTRn DATATYPEn,
<constraint1>,
<constraint2>);

Constraints: PRIMARY KEY => primary key (ATTRk)
FOREIGN KEY => foreign key (ATTRk) references 's'
NOTNULL => not null.

Remove Schema from Database:
Syntax: DROP TABLE relation;

Remove Tuples from Database Relation:
Syntax: DELETE FROM relation;

Alter Schema Definition:
Syntax: ALTER TABLE relation
ADD ATTRP DATATYPEP;

Syntax: • ALTER TABLE relation
DROP ATTRP;

Query Database Information:
Syntax: SELECT ATTR1 ... ATTRR
FROM Relation;

Syntax: • SELECT DISTINCT ATTR FROM Relation.

FEBRUARY - 2024

M	T	W	T	F	S	S	Wk
			1	2	3	4	05
5	6	7	8	9	10	11	06
12	13	14	15	16	17	18	07
19	20	21	22	23	24	25	08
26	27	28	29				09

SELECT ALL ATTRP (for explicitly not removing duplicate tuples).
FROM RELATION;

for arithmetic expression:

SELECT <EXPR>;

CARTESIAN PRODUCT: SELECT * FROM R1, R2
⇕
R1 × R2.

RENAME: SELECT ATTR; AS <NAME>
FROM Relation;
(or)
SELECT <Rename>. ATTRP
FROM RELATION as <Rename>;

STRING OPERATIONS:

*) Equality: Case Insensitive.
*) UPPER(S)
*) LOWER(S)
*) concatenating : ||
*) TRIM(S)
*) LIKE (PATTERN MATCHING) (%, _)
*) ESCAPE
*) SIMILAR TO (UNIX Regex).

ORDERING:

```
SELECT *
FROM RELATION
WHERE PREDICATE
ORDER BY ATTRIBUTE [DESC / ASC]
```

BETWEEN:

```
SELECT *
FROM RELATION
WHERE x BETWEEN 90 AND 100;
```

We can also use NOTBETWEEN OPERATOR.

LEXICOGRAPHIC ORDERING:

```
SELECT *
FROM RELATION
WHERE (x, y) = (z, w);
```

UNION:

```
(SELECT * FROM x) UNION (SELECT * FROM y);
```

Eliminates duplicates.

```
(SELECT * FROM x) UNION all (SELECT * FROM y);
```

Doesnot eliminate duplicates.

INTERSECT:

```
(SELECT * FROM x) intersect (SELECT * FROM y).
```

Eliminates duplicates, use all keyword.

SET DIFFERENCE:

(SELECT * FROM x) EXCEPT (SELECT * FROM y);

↓

use 'all' to retain duplicates.

IS NULL / IS NOT NULL:

(SELECT * FROM x
WHERE y is NULL).

UNKNOWN:

x > NULL => UNKNOWN (result).

∴ SELECT * FROM x
WHERE y > z is NOT UNKNOWN;

AGGREGATE functions:

functions that take in a collection of values and output a single value.

Avg, count, min, max, sum.

SELECT COUNT(*) FROM COURSE;

↓

No. of tuples in course.

ILLEGAL TO use distinct in count.

GROUP BY: TO apply aggregate function to a group of sets of tuples.

Wk	M	T	W	T	F	S	S
05				1	2	3	4
06	5	6	7	8	9	10	11
07	12	13	14	15	16	17	18
08	19	20	21	22	23	24	25
09	26	27	28	29			

Eg: SELECT * FROM x group by (attribute1);

When an SQL query uses grouping, it is important to ensure that the only attributes that appear in the select statement without being aggregated are those that are present in the group by clause.
If this condition fails, the query is erroneous.

HAVING CLAUSE:
We use having clause to apply conditions after grouping

```
SELECT *
  FROM x
  WHERE P
  GROUP BY y
   HAVING P';
```

ORDER OF OPERATIONS:
*) The FROM clause is first evaluated
*) Where clause is then executed.
*) Then Group by clause is executed.
*) Having clause is executed.

SUM : ignores null values.
COUNT(*) : Ignores null values.

SOME & ALL : SELECT * FROM relation WHERE x > SOME (SELECT x FROM relation);
SELECT * FROM relation WHERE x > ALL (SELECT x FROM relation);
SOME values greater than x.
x greater than ALL values.

MARCH - 2024
M T W T F S S Wk
1 2 3 09
4 5 6 7 8 9 10 10
11 12 13 14 15 16 17 11
18 19 20 21 22 23 24 12
25 26 27 28 29 30 31 13

EXISTS: SELECT EXISTS (SELECT * FROM x);
↓
TRUE if there are some tuples.

UNIQUE: SELECT UNIQUE (SELECT * FROM x);
↓
TRUE if all tuples are unique.

SUBQUERIES IN FROM CLAUSE:

```
SELECT *
    FROM (SELECT id FROM Employees);
```

WITH CLAUSE:

The with clause provides a way of defining a temporary relation whose definition is available to query in which the with clause occurs.

Eg:
```
with · dept_total (dept_name, value) as
        (select dept_name, sum(salary)
         from Instructor
         group by dept_name),
      · dept_total_avg (value) as
        (select avg(value) from
            dept_total)
select dept_name
    from dept_total, dept_total_avg;
```

SCALAR SUBQUERIES:

SQL allows subqueries to occure wherever an expression returning a value is permitted, providing the subquery returns only one tuple containing a single attribute, such is called as scalar subqueries.

UPDATE:

```
UPDATE relation
SET salary = case
        when salary <= 100 then salary * 1.05
        else salary * 1.03
        end;
```

INTERMEDIATE SQL:

JOIN: Natural join

```
SELECT *
FROM x, y
WHERE x.id = y.id;
```

or

```
SELECT *
FROM student natural join employee;
```

JOIN conditions:

```
SELECT *
FROM x
JOIN
y on θ (predicate)
```

```
SELECT *
FROM x
join y using (id);
```

INTERMEDIATE SQL:

NATURAL JOIN:
```
SELECT x, y
FROM z, w
WHERE z.i = w.i;
```

(or)

```
SELECT x, y
FROM z NATURAL JOIN w;
```

Same result: given attributes are equal (primary key).

JOIN: We can use normal joins with arbitrary 'join condition'.

```
SELECT *
FROM x JOIN y
ON x.i = y.j;
```

OUTER JOIN: Preserves tuples that would have been eliminated by the join operation.

Forms of outer join:

- left outer join: preserves tuples in relation left to the join
- right outer join: preserves tuples in relation right to the join.
- full outer join: preserves both tuples.

LEFT OUTER JOIN:

Select * FROM x Natural left outer join y;

RIGHT OUTER JOIN:

Select * FROM x Natural right outer join y;

FULL OUTER JOIN:

SELECT * FROM x Natural full outer join y;

If we have to specify a join condition, do so with "on" clause;

Important: On & where behave differently, OUTER JOIN adds null padded tuples which does not contribute to the result of the corresponding inner join.

Join Types: To distinguish various join types, we explicitly use the keyword outer ~~wherever~~ since inner joins are the default join operation.

Note: we can also use the keyword using as such.

Select *
FROM x ~~natural~~ join y
using (ID)

Views: Virtual Relations that only exposes certain data to users for security purposes.

*) It is not synchronized, therefore when we update the relation, the view is not updated. (Don't Believe this!)

*) SQL views are recomputed whenever the query is executed.

View Definition:

create view "v" as
"select . . . ";

Materialized Views:

Certain Databases allow views to be precomputed but when actual relations change, the view is kept upto date called as materialized views

*) The Process of keeping the views upto date is called as Materialized view maintenance / view maintenance (when view is accessed).

*) The Maintenance can be done lazily / periodically. and some systems allow the admin to pick the method to be used.

*) Applications that use a view frequently greatly benefit from this.

*) Benefits to queries from materialization of a view must be weighed against storage costs and added overhead for maintenance.

View updates:

operations such as updates, inserts, deletes are not permitted on views, since there are endless possibilities to do with data obtained from / updated in the view w.r.t to the relation. But if some conditions are satisfied, we are allowed to perform calculations/~~on~~ updates on views.

*) Conditions:

1) The From clause has only one database

2.) The Select clause only has attributes / names of the relation and not expression, aggregates / distinct specification.

3.) Any Attribute not listed in the select clause can be set to NULL i.e it has no "not null" and "primary key" constraint.

4) The Query has no group by / having clause.

*) SQL provides an option where at the end of the view definition, we can add 'with check' option that checks if a tuple inserted into a view satisfies the view definition's where clause condition, updates are also rejected.

*) We can also use the 'trigger mechanism' by mentioning the keyword 'instead of'

Transactions:

*) A Transaction consists of sequence of query and or update statements.

*) SQL standard specifies that an transaction begins implicitly when an SQL statement is executed.

commit work: commits the transaction, updates performed by the queries are made permanent.

*) After a commit, a new transaction is started automatically.

Rollback work: *) causes the changes done by the queries to be rolled back (undoes all the updates)

*) Database state is restored to the state of last commit.

*) Once a transaction has been committed, it can no longer be rolled back

*) used when a power outage, system crash, a transaction's effects are rolled back to the last commit and is done automatically when an system restarts.

*) In modern databases, an SQL statement is a standalone transaction and any changes done by the statement is automatically committed after execution is done.

*) This is known as automatic commit.

MARCH - 2024

M	T	W	T	F	S	S	Wk
				1	2	3	09
4	5	6	7	8	9	10	10
11	12	13	14	15	16	17	11
18	19	20	21	22	23	24	12
25	26	27	28	29	30	31	13

*) we can turn off autocommit by:
'set autocommit off'

*) As of SQL: 1999 std., allows multiple SQL statements to be executed by enclosing between begin [atomic] ... end block. which then forms a single transaction, which is committed when the execution reaches the end statement.

*) MySQL & PostgreSQL only allows begin statement and disallows end keyword. Instead the transaction (started by begin) must be ended by either a commit / rollback statement.

Integrity constraints:
we can add an constraint to a relation by:
"ALTER TABLE " "
ADD CONSTRAINT " x not null ";
*) We first check, if the relation already satisfies the constraint, if not we reject the constraint.

Constraints:
*) Not null / null
*) unique
*) check (<predicate>)

Not Null: The constraint prohibits the insertion of a null value for the attribute and is an example of a domain constraint.

FEBRUARY - 2024

Wk	M	T	W	T	F	S	S
05				1	2	3	4
06	5	6	7	8	9	10	11
07	12	13	14	15	16	17	18
08	19	20	21	22	23	24	25
09	26	27	28	29			

The pure and simple truth is rarely pure and never simple. - Oscar Wilde

Unique constraint: The unique key specification states that attributes A_1, ~~A_2~~ A_2 ... form a super key.

*) Unique attributes are permitted to be null unless explicitly constrained to be not null.

Check clause: the check clause specifies a predicate P that must be ~~applied~~ satisfied by every ~~relation~~ tuple in the ~~tuple~~ relation.

we can simulate enumerations by:

~~Same~~

check (x in ("Summer", "Winter", "Spring", "Autumn"));

*) clauses that evaluate to unknown are permitted (use not null to restrict null values)

Referential Integrity:

constraints that enforce referential integrity ~~constraints~~ are called as Referential Integrity constraints. and foreign keys ~~are~~ is a way to enforce referential integrity in databases.

"foreign key (attribute) references table"

or

"foreign key (attribute) references table(attribute)"

Note: The specified list of attributes must be declared as a super key. (primary key / unique constraint).

Note: the attributes must be compatible (same no. of attributes, with same data types)

we can also use the shorthand syntax:
attribute datatype references relation;

when referential integrity constraint is violated,
the normal procedure is to reject the action that violated the constraint.

We can also specify if the action (delete / update) on the relation violates the constraint, instead of rejecting, the system can take necessary preconfigured steps to restore the constraint.

Eg: Create table course (
foreign key (x) references y
on delete cascade
on update cascade,

);

If the delete operation on y violates the constraint, then ~~we~~ instead of rejecting the delete, we also delete the tuple in the course attribute.

If the update operation on y violates the constraint then instead of rejecting the update, we update the tuple in the course attribute to the newest value.

Other actions such as setting to default ~~also (NULL) can be~~ or setting to NULL
eg: ~~set~~ using 'set default'
'set null'

*) If a chain of cascading is present then the operations ~~are~~ gets expanded, if any problems occur in deleting ~~rows~~ (circular dependency) and the system cannot handle it, the system aborts the transaction and all changes are undone.

Assigning Names to Constraints:

salary numeric (8,2), constraint minsalary check (salary > 29000);

alter table instructor drop constraint minsalary.

Defer Constraint Checking:

We can force databases to check if constraints are satisfied after a transaction is completed (default: immediately after a statement) by using deffered keyword.

" set constraints minsalary deferred;

Assertions:

An Assertion is a predicate expressing a condition that is required for a database to always satisfy.

" create assertion <assertion-name> ~~ch~~ check <predicate>;

SQL Types: *) Date: A Calendar Date: YYYY-MM-DD.

*) Time: The Time of Day in Hours, Minutes, Seconds.
We can also specify the number of fractional digits in seconds, using Time (p)
- We can also store Timezone Info using 'time with timezone'

*) Timestamp: combination of Date & Time
- timestamp(p)
- with timezone.

MARCH - 2024

M	T	W	T	F	S	S	Wk
				1	2	3	09
4	5	6	7	8	9	10	10
11	12	13	14	15	16	17	11
18	19	20	21	22	23	24	12
25	26	27	28	29	30	31	13

The unexamined life is not worth living.
- Socrates

*) To Extract field from date / time
extract (field from d)
*) fields: year, month, day, hour, minute.
timezone information by using:
timezone-hour, timezone-minute.

Functions for Date/Time:
*) current_date
*) current_time
*) localtime (current local time w/o timezone)
*) current_timestamp (with timezone)
*) local timestamp (without timezone).

Datatype: "Interval" analogous to deltatime.

Type conversion:
cast (expr as type)
↓
cast (pd as numeric(2,4));

Format:
coalesce (salary, 0)
↓
display 0 if null appears.
Note: salary and the coalesced value must be of same type.

System specific functions are available for formatting values,
Eg: decode ();

Default value: We can assign default value to a attribute by
" create table a (y int default 0);

Large Object Types:
Many Database applications need to store attributes, whose domain consists of large data items such as photo, video etc. Therefore SQL provides "Large Object Datatypes" for character data (clob) and binary data (blob)

book_review	clob(10KB)
image	blob (1MB).
movie	blob (2GB)

for large data objects, we just return an "locator" which can be used by programming languages to read data at arbitrarily any point instead of loading it into memory which is impractical.

This locator can then be used to read the file/data using "read" system call.

User Defined Types:
SQL supports two forms of user defined data types:
↳ Distinct Types (Typedefs).
↳ Structured Data Types. (complex Data types)

Distinct Types: Create type Rupees as Numeric(12,2) final;
Here final is required to be specified as per SQL1999 but some implementations allow the final keyword to be omitted.

We cannot make direct operations (+, -) on distinct types, we must first cast it to normal given data types, perform the operation and then cast it back to the distinct data type

"drop type ..." : Delete
"alter type ..." : Modify.

Before datatypes were added in SQL 1999, we used "Domains" from SQL 92 which had integrity constraints

Eg: create domain x as Numeric (12,2) not null;

Note: Not strongly typed, values can be added as long as the underlying types are the same.

Some Database Systems do not implement ~~both~~ domain or types or both

Eg Domain Enum:

```
create domain Direction as Varchar(12)
    constraint direction-constraint
        check (value in ('North',
                         'South',
                         'East',
                         'West'));
```

Automatic ID Generation:

Sometimes we require that id (numeric types) to be automatically generated.

Syntax: id number(5) generated always as identity

always: If specified, insert statement must avoid specifying a value for the attribute.

identity: with this specification, the database, can choose the starting value, increment value and minimum, maximum value.

by default: (different from always) we can set the ID or else let the database generate one.

We can also use create sequence construct, which is a sequence object that allows SQL queries to obtain the next value from the sequence.

- This can be used for generating identifiers that are unique across multiple relations.

Create Table Extensions:

Create a table with the same schema as an existing table.

```
create table x like y;
```

Create table from a query:

```
create table x as (<query>) with data;
```

If we omit "with data" clause, the schema from the output of the query is used and the relation is not populated.

Schemas, Catalogs & Environments: Modern Database Systems provide a three level Hierarchy,

- At the top, catalogs are present (database)
- At the middle, schemas are used.
- SQL objects (Relation, views).

I lie to myself all the time. But I never believe me. - S.E. Hinton, The Outsiders

MARCH - 2024

M	T	W	T	F	S	S	Wk
				1	2	3	09
4	5	6	7	8	9	10	10
11	12	13	14	15	16	17	11
18	19	20	21	22	23	24	12
25	26	27	28	29	30	31	13

In order to perform any actions on a DBMSystem,
*) a user/program must connect to the database, providing the username, password.
*) Each user has a default catalog & schema and the combination is unique to the user.
*) To uniquely identify an relation, we use the following three part name:

<catalog>.<schema>.<relation>

*) we can skip specifying the catalog, which results in using the default catalog name by default.
*) we can also skip the schema name, resulting in default schema name for the user.

The Default catalog, schema are a part of an SQL environment that is setup for each connection.
*) The Environment also contains the user identifier (authorization identifier)
*) Schemas are created automatically with the user name as the default schema name. (create schema, drop schema)
*) If a schema is newly created, it becomes the default schema for the user.
*) Creation & Deleting of catalogs is implementation dependent & not specified in SQL standard.

FEBRUARY - 2024

Wk	M	T	W	T	F	S	S
05				1	2	3	4
06	5	6	7	8	9	10	11
07	12	13	14	15	16	17	18
08	19	20	21	22	23	24	25
09	26	27	28				

Indices / Index :

An Index on an attribute for an relation, is a data structure that allows the database system to find the tuples in the relation with a specified value for that attribute efficiently

*) They are redundant datastructures and are not required for correctness.

*) Very important for efficient processing of transactions and enforcing primary key / foreign key constraints.

*) Part of the Physical Schema and not the logical schema and because of that it is not standardized by the SQL.

Syntax :

create index <name> on <relation> (<attribute-list>);

If we wish to declare that the ~~se~~ attribute-list is a candidate key, we add the attribute <unique> to the index definition

Eg: create unique index . . .

This statement assumes that the attribute-list is a candidate key and failing to do so will result in a index creation error.

Index name required to drop index :

drop index <name>

Most indices also support the specification of the type of index datastructure to be used, such as B^+-Trees / Hash Indices.

MARCH - 2024

M	T	W	T	F	S	S	Wk
				1	2	3	09
4	5	6	7	8	9	10	10
11	12	13	14	15	16	17	11
18	19	20	21	22	23	24	12
25	26	27	28	29	30	31	13

Monday

Wk 08 (050-316)

Authorization:

Several forms of authorization:

↳ Read

↳ Insert

↳ Update

↳ Delete. (Each of them called an privilege)

The superuser (Database Admin) can grant / revoke privileges to other users. We can also grant privileges on creating, deleting, updating schemas.

When an user submits a query, the database first checks if the user is authorized or not.

Syntax: grant <privilege-list>
on <relation/view ~~list~~ name>
to <user/role list>

Eg: grant select
on course
to dinesh;

Update authorization can be granted for all relations or to specific relations by:

grant update
on course (instructor)
to dinesh;

Similarly insert authorization on a relation allows a user to insert tuples in a relation or specific attributes

with. The System either gives them the unauthorized attributes to the default value or null

Username: public, refers to all present & future users.
We can also specify, if the privilege recipient can also grant the same privilege to others.

We can revoke a privilege by:

```
revoke <privilege-list>
on <relation/view list>
from <user-list>
```

Roles: Class of users to which privileges can be granted / revoked at the same time.

- Disadvantage: We cannot track which user from a role perform a specific transaction.
  - If a user is removed from a role, then a new password must be generated.
- Any authorization granted to a user can be granted to a role.
- Roles can be granted to users as well as roles.

```
Create role dean;
grant instructor to dean;
grant dean to dhesh;
```

The Dean Role inherits all the privileges of the instructor role.

*) An user who creates a view does not necessarily receive all privileges on the view.
*) If a user creates a view with no authorization privileges to the underlying relation, then the creation of the view is ignored.
*) Execute privilege can be given to any function/procedure and is treated as an user.
*) We can add the clause "sql security invoker", which uses the privileges of user who invokes the function, rather than the definer.

Authorization on Schema:

We can grant the privilege "references" with the relation and attribute name to authorize the creation of ~~quer~~ foreign key constraints.

*) This is restricted to insert/select & not delete/update.

Transfer of Privileges:

We can allow the recipient of the privilege to pass on the privilege to other users by.

grant select on dept to dinesh with grant option.

We can revoke grant option by

revoke grant option for select on dept from dinesh;

we can revoke privileges from users which further revokes all the privileges down the dependency chain. This is called as cascading revocation.

*) we can restrict this by specifying 'restrict' clause at the end of the revoke statement.

*) Cascading revocation is inappropriate and therefore rec fr is specified by using the keyword cascade.

Functions / Procedures & SQL API:

*) Not all queries can be expressed in SQL (Not turing complete) and therefore we express queries in languages such as C, Java, Python

*) Two Approaches to accessing SQL
- ↳ Dynamic SQL
- ↳ Embedded SQL

Dynamic SQL: A General Purpose program connects to and communicates with a Database server

*) Allows the program to construct queries at runtime submit the query and retrieve the result.

Embedded SQL: the SQL statements are identified at compile time using a preprocessor, translating the requests in embedded sql into function calls

We can also run an Embedded Database like sqlite.

Functions & Procedures: SQL allows the definition of functions, procedures, methods. These can be defined either by the procedural component of SQL (PSQL)

Most Databases implement things defined by the standard some define & implement syntax that is non standard.

Performance problems are observed since the function is invoked on a large number to tuples.

Eg:

```
create function dept_count (dept_name varchar(20))
        returns int
begin
    declare d_count int;
    select count(*) into d_count
    from dept_instructor as d
    where d.name = dept_name;
    return d_count;
end
```

SQL standard supports function that can return tables as results which are called as table functions.

*) In general table-valued functions can be thought as parameterized views.

```
create function instructor_of (dept_name varchar(20))
        returns table (
            id varchar(20),
            :
        )
return table (
    select id ... from inst
    where inst.name = name);
```

SQL also supports procedures:

```
create procedure dept_count (in dept_name varchar(20),
                             out d_count integer)
begin
    select count(*) into d_count
    from instructor
    where instructor.dept = dept_name;
end
```

The keywords in, out indicate the parameter return type.
can be invoked using:

```
declare x integer;
call dept_count("Phy", x);
```

Procedures & functions can have same name as long as they have different number of arguements.

SQL supports constructs and is implemented by Persistent Storage Modules (PSM)

* Variables are created using declare statement
* Assigned using set statement
* Compound statements using begin and end. and also can be atomic using begin atomic end
* while loop:

```
while boolean_expr do
    sequence...
end while
```


MARCH - 2024

M	T	W	T	F	S	S	Wk
				1	2	3	09
4	5	6	7	8	9	10	10
11	12	13	14	15	16	17	11
18	19	20	21	22	23	24	12
25	26	27	28	29	30	31	13

Repeat statements:

```
repeat
    sequence
until <expression>
end repeat.
```

For Loop:

```
declare n int default 0;
for r as select budget from dept do
    set n = n - r.budget
end for
```

break : leave
continue : iterate

conditional:

```
if <boolean expression> then
    <statement>
elseif <boolean expression> then
    <statement>
else <statement>
end if
```

declare out_of_seats condition.
declare exit handler for out_of_seats begin
...
end.

This statement says that if there's a exception named "out_of_seats" in the compound statement, the action is to exit.

we can also specify continue.

we also have predefined conditions:
sqlwarning
sqlexception
not found.

triggers: An action that gets executed automatically as a side effect of a modification to the database.

To Define a trigger we must: 1.) specify when a trigger is to be executed by specifying the action and a condition
2.) specify the actions to taken.

CREATE TRIGGER x BEFORE/AFTER UPDATE/INSERT/DELETE
ON RELATION
[REFRENCING NEW ROW AS ...]
[REFRENCING OLD ROW AS ...]
FOR EACH ROW
WHEN (COND)
BEGIN .... END.

Triggers can be used to ensure referential integrity by activating on a relation after insertion and checking if the tol is present in the referencing relation and if it is not present rollback.

* we can insert multiple rows and update rows, by explicitly telling for each row, we can exptl iterate over each inserted row.

* The "referencing new row as" creates a variable (a transition variable) that stores the value of the row being inserted.

* The when statement specifies a condition and the system executes the trigger body only for tuples that satisfy the condition.

* we can specify update triggers to activate on specific attributes by:
after update of ~~re~~ relation on attribute.

* referencing old row can be used to store the value of an updated / deleted row. referencing new row can be used with updates / inserts.

* we disable / enable triggers by alter trigger (name) disable/enable
* if triggers fail, then the action that set off the trigger will also be failed.

* A Trigger chain is avoided by limiting the length of the chain. to 16/32

A thing is not necessarily true because a man dies for it. - Oscar Wilde

Relational Database Design:

Goal of Database Design is to generate a set of relation schemas that allow us to store relational schemas without unnecessary redundancy but allows us to retrieve information efficiently.

We accomplish this by starting it in the appropriate normal form.

Features of Good Relational Design:

The Goodness or Badness of the schemas generated from the E-R ~~De~~ Model depends on how good the model is.

Decomposition: If we feel that certain schemas are useless / redundant we can decompose the schemas as follows

- Lossy Decomposition (Loss of Information after decomposing and joining).
- Lossless Decomposition.

(All Decompositions must be lossless).

Lossless Decomposition: let R be a schema and decomposed into R_1 & R_2

$R = R_1 \cup R_2$ (as sets of attributes).

We say that this decomposition is lossless since we can replace R with R_1 & R_2.

Normalization Theory: A General Methodology for deriving a set of schemas that does not suffer from the repetition-of-information problem.

Approach: We decide if a relation is in good form or not, if not in good form we decompose it into a number of smaller relation schemas that are in appropriate "normal form" and decomposition must be lossless.

Decomposition using Functional Dependencies:

There are some real world constraints that are implicit in a database.

Eg: There is only one budget for a department.

An instance of a relation that satisfies all such real world constraints is called a legal instance of the given relation.

Notational conventions:

- Greek letters for set of attributes.
- K = superkey.

Keys & Functional Dependencies:

Whereas a superkey is a set of attributes that uniquely identifies an entire tuple,

A functional dependency allows us to express constraints that uniquely identifies the values of certain attributes.

- Value of the determinant decides the value of another attribute
- Specifies integrity constraints on candidate keys of relation
-

$$\underbrace{A}_{\text{Determinant}} \rightarrow B$$

B is functionally dependent on A.

If you know value of A, then we know the value of B.

Eg: cust_id → name ✓
name → cust_id ✗

Given an instance of $r(R)$, $\alpha \subseteq R$ $\beta \subseteq R$
we say that the instance satisfies the functional dependency
$\alpha \rightarrow \beta$ if for all pair of tuples t_1, t_2 such that
$t_1[\alpha] = t_2[\alpha]$
$t_1[\beta] = t_2[\beta]$

$\alpha \rightarrow \beta$ holds on $r(R)$, if every legal instance of $r(R)$ satisfies the functional dependency.

In general a functional dependency is said to be trivial if they are of the form:

$\alpha \rightarrow \beta$, if $\beta \subseteq \alpha$

F^+ denotes the closure of the set F, set of all functional dependencies that can be inferred from the set F.

we can decompose using functional dependencies ~~by~~ lossleasly if atleast one of the following dependencies is in F^+/F

$R_1 \cap R_2 \rightarrow R_1$
$R_1 \cap R_2 \rightarrow R_2$

If $R_1 \cap R_2$ forms a superkey for either R_1 or R_2, the decomposition is lossless.

MARCH - 2024

Wk	M	T	W	T	F	S	S
09					1	2	3
10	4	5	6	7	8	9	10
11	11	12	13	14	15	16	17
12	18	19	20	21	22	23	24
13	25	26	27	28	29	30	31

Every strike brings me clo
home run. -Babe

March:

2024 March 02 Saturday

Wk 09 (062-304)

If we decompose a relation $r(R)$ to $r_1(R_1)$ and $r_2(R_2)$ where

$$R_1 \cap R_2 \rightarrow R_1$$

then the following constraints must be satisfied:

- $R_1 \cap R_2$ is the primary key of r_1.
- $R_1 \cap R_2$ is the foreign key from r_2 referencing r_1.

Normalisation:

First Normal Form Rules:

- Using row order to convey information is not permitted.
- Mixing Datatypes within the same column is not permitted.
- Having a table without a primary key
- Repeating groups are not permitted.

Second Normal Form:

Each Non-Key attribute must be dependent on the entire primary key.

Third Normal Form:

Each Non-Key Attribute should only depend on the key. (No Transitive Dependency), the whole key and nothing but the key.

BCNF (Boyce Codd Normal Form):

Every Attribute in a table should depend on the key, the whole key and nothing but the key

Fourth Normal Form: Multivalued dependencies in a table must be multivalued dependencies on the key.

APRIL - 2024

M	T	W	T	F	S	S	Wk
1	2	3	4	5	6	7	14
8	9	10	11	12	13	14	15
15	16	17	18	19	20	21	16
22	23	24	25	26	27	28	17
29	30						18

You miss 100% of the shots you don't take. –Wayne Gretzky

Closure of a set of Functional Dependencies:

Given a set of Functional Dependencies on a schema R we can prove that other functional dependencies also hold, called logically implied functional dependencies.

If F is the set of functional dependencies and closure of F, is denoted by F^+, is set of all functional dependencies logically implied by F.

Armstrong's Axioms:

we can find logically implied dependencies by applying these rules repeatedly:

Reflexivity Rule: If α is a set of attributes & $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$

Augmentation Rule: If $\alpha \rightarrow \beta$ holds & γ is a set of attributes, then

$\gamma\alpha \rightarrow \gamma\beta$

holds.

Transitive Rule: If $\alpha \rightarrow \beta$ holds,

$\beta \rightarrow \gamma$ holds,

then $\beta \rightarrow \gamma$ holds.

These axioms are sound & complete.

There are some additional rules.

Union rule: if $\alpha \rightarrow \beta$

$\alpha \rightarrow \gamma$

then $\alpha \rightarrow \beta\gamma$

Decomposition Rule: if $\alpha \rightarrow \beta\gamma$

then $\alpha \rightarrow \beta$

$\alpha \rightarrow \gamma$.

MARCH - 2024

Wk	M	T	W	T	F	S	S
09					1	2	3
10	4	5	6	7	8	9	10
11	11	12	13	14	15	16	17
12	18	19	20	21	22	23	24
13	25	26	27	28	29	30	31

We all need people who will give us feedback. That's how we improve. - Bill Gates

Pseudotransitivity Rule: If $\alpha \to \beta$
$\gamma\beta \to \delta$
then $\alpha\gamma \to \delta$

Procedure to compute F^+

$F^+ = F$
repeat until F^+ doesn't change
for each functional dependency in F^+
apply the augmentation rule on f
add the resulting functional dependencies to F^+
for each pair f_1, f_2 in F^+
If f_1, f_2 can be combined
add the resultin.

Closure for attribute sets:
an attribute β is functionally determined by α if
$\alpha \to \beta$
To Determine if α is a superkey.
we generate all the dependencies in F^+
and take the dependencies with
α in the left hand side.
and union them
If the attributes are equal to every attributes in R, then
α is a super key.

File Organizations / Indexing / Transaction Control.

Storage Interface:

Magnetic Disks (DVD) as well as flash based solid state drive are connected to a computer, through a hispeed interconnection. Disks support SATA (Serial ATA) or Serial Attached SCSI (SAS) (used in servers).

SATA-3 : 6 gigabits / per second.
SAS-3 : 12 gigabits / per second.
NVMe (Non Volatile Memory Express) logical interface developed to support SSD used with PCIe Interface.
SAN : Storage Area Network
FC: Fiber Channel.
ISCSI : IP SCSI
SCSI : Small computer System Interface
PCIe : Peripheral component Interface Express.
NAS: Network Attached Storage.

Measures of Disks' Performance:
* Access Time: A Read / Write Request Issued and data transfer begins
* Data Transfer Rate: Rate at which data can be retrieved or stored to disk.
* Reliability: Mean Time to Failure (MTTF) measure for reliability
* Capacity.

Flash Memory.
↳ NOR flash
↳ NAND flash. (used for data storage).

NAND Flash: 4096 bytes => one page
similar to sector.
Much faster random access than hard disks.
SSD (20-100 μs)
HDD (5-10 ms).
Power consumption is lower
Higher Data Transfer Rate (3 Gigabit per second).

- writes to flash memory is slower, A write to a page of flash (100 ms)
- Once written cannot be directly overwritten.
- It has to be erased and rewritten subsequently.
- Flash Memory Systems limit the impact of slow erase speed by having a logical mapping / physical mapping of pages. During erasing, we can then write to a empty page and erase the logical page.
- Logical to Physical page mapping is replicated in an in-memory translation table for quick access.

The principle of ~~wear~~ evenly distributing erase operations across physical blocks = wear levelling.

If a physical page is damaged ~~or~~ due to excessive updates, we can remove it from usage.

This is all done by using a flash translation layer and above this layer, the flash drive looks just like an hard drive.

APRIL - 2024

M	T	W	T	F	S	S	Wk
1	2	3	4	5	6	7	14
8	9	10	11	12	13	14	15
15	16	17	18	19	20	21	16
22	23	24	25	26	27	28	17
29	30						18

SSD Perf: Number of Random Block Reads.
Data Transfer Rate.
No. of Random Block writes.

RAID: A Variety of Disk organization techniques called Redundant Array of Independent Disks proposed to achieve performance and reliability.

Reliability by Redundancy, extra information to get back the contents.

Simplest & Most Expensive: Mirroring / Shadowing

Database Storage Architecture:
Persistent Storage Data => Storage Non-Volatile.
Databases: Records
HDD / SSD: Blocks

Most Databases use operating systems files as an layer of abstraction to avoid dealing with underlying blocks.
To ensure efficient access & recovery from failure, the database must be aware of block structure.

For a CPU to access data, it must be in main memory and ~~then~~ persistent data must be loaded into memory first and then used. If values are updated, then it must be saved back.

~~Use~~ Databases store data by storing all values of a column together rather than a row together, this has found to be very good for analytical query processing, known as column oriented storage.

Databases that can be stored in the main memory entirely is known as main memory databases, which exploit fast data access time of main memory ~~into account~~ and use algorithm that manipulate this.

File Organization: A Database is mapped into number of files maintained by the operating systems, residing on disks.

* A File => Logical sequence of Records.

*) Each file is logically partitioned into fixed length storage units called blocks, units of storage allocation & data transfer.

*) Block size: 4 - 8 KB by default.
can be bigger if we want.

*) A Block can contain several records, we assume that a block is larger than a record.

*) Each record is stored entirely in a single block.

Tuples of distinct relations are generally of different sizes, one approach is to use several files and to store records of only one fixed length in a given file.

Fixed Length Records:

In this method, we continuously allocate the space for each member in the record.

We store the records contiguously and any record that can't be stored entirely is stored in another block.

When a record is deleted we could do two things, we co

1) move the previous records up and use the free space.

2.) use a header that points to the next free space which then forms a linked list ~~to~~. (Deletions are rare compared to insertions, therefore we do not use the free space immediately after deletion.)

d) This is known as a free list and when we change insert a new record, we use the pointer to the first space and insert the record there.

Variable Length Records:

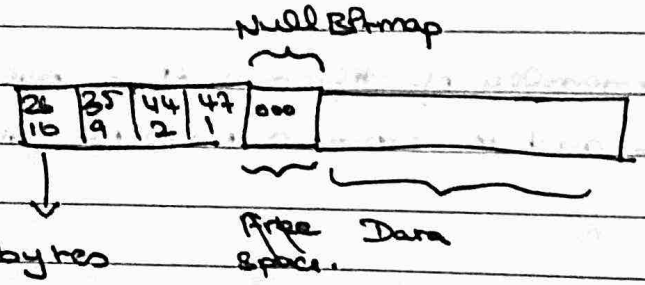
Motivation: Strings, Arrays, Multisets.

Questions: * How to represent a single record such that individual attributes can be easily accessed?

* How to store variable length records within a block, such that the records in a block can be extracted easily.

Variable length Attributes such as the varchar are represented by a pair of ints (offset, length). The values for the variable length attributes are stored consecutively

Eg:



4 bytes per attribute.

Null Bitmap: Indicates which attributes of the record have a null value.

: 0 → Not Null

1 → Null., sometimes it is found in the beginning

Slotted Page Structure: A common way to organize records within a block, it contains the following information.

*) Number of record entries in the header

*) End of free space in block

*) Array containing location & size of records.

MARCH - 2024

Wk	M	T	W	T	F	S	S
09					1	2	3
10	4	5	6	7	8	9	10
11	11	12	13	14	15	16	17
12	18	19	20	21	22	23	24
13	25	26	27	28	29	30	31

Time is nature's way of keeping everything from happening at once. - Woody Allen

The actual records are contiguously allocated. The free space in the block is also contiguous. If a record is inserted, the space is allocated at the end of free space, an entry is created with its location and size.

*) If a record is to be deleted, the space occupied is freed and its entry is set to be deleted. (size is set to -1)

*) The records after the deleted record are moved, so that the free space created during deletion gets occupied.

If the record needs to be resized, then the records are moved to make space since a typical block is around 4 to 8 KB.

The slotted page structure requires that no pointers that point directly to the entry are present. Instead, pointers must point directly to the entry which contains the actual location of the record.

Storing Large objects:

Databases store large data (that are bigger than a block) in two ways.

- Stored either as files in file system area managed by the database.
- File structures stored and managed by the database.

*) Most Databases restrict the size of a record to be lesser than a block size. Audio/ Video/ Images are stored separately from other attributes.

*) A logical pointer to the object is then stored in the record containing the large object.

*) The efficiency of accessing large objects via databases is another concern, with the increase in also the buildup of size in database backups.

APRIL - 2024

M	T	W	T	F	S	S	Wk
1	2	3	4	5	6	7	14
8	9	10	11	12	13	14	15
15	16	17	18	19	20	21	16
22	23	24	25	26	27	28	17
29	30						18

Many applications therefore do not use databases to store very large objects, instead opting to choose the file system with the path of the data stored in databases.
Storing Data in files outside the database can result in filenames in being inconsistent with the file system (files can be deleted)
*) This results in referential integrity violations and database authorization controls.
*) Some databases support file system integration that ensures that referential integrity & access authorization are enforced

Organization of Records in Files:
↳ Types:
*) Heap file organization
*) Sequential file organization.
*) Multitable clustering file organization
*) B+ Tree file organization.
*) Hashing file organization.

Heap file organization:
*) In this method, a record may be stored anywhere in the file corresponding to the relation.
*) When a record is inserted, it is added to the end of the file.
*) When a record is deleted, the space is used to store the new records.
*) A Database must easily find blocks that have free space without sequentially searching all the blocks.
*) Databases use a datastructure called freespacemap that tracks all the free space to store records.

The free space map is an array with an entry for each block in the relation.

Each entry represents a fraction f, such that atleast a fraction f of the space in the block is free.

This array is stored in a file and is fetched into memory.

The array gets updated whenever the fraction changes after insertion, updation, deletion.

0	2	4	6	7	3	2	0	1	0	2	7

↓

3/8 block size is free.
(8 since 3 bits are used to store an entry).

To store the record/~~block~~ of a given size, the database scans the free space map and finds the index of the block that is large enough to store the record.

To speed up this process, we can use a second free space map that contains the maximum value of each 100 entries in the ~~f~~ first level ~~of~~ free space map, this minimizes the search space by 1/100.

~~for~~

For very large relations, we can create many levels of free space map as required.

Updating a free space map each time an entry is updated is very expensive and is therefore done periodically.

This can result in a free space map being inconsistent with the underlying data ~~used~~ (when a database starts up).

APRIL - 2024

M	T	W	T	F	S	S	Wk
1	2	3	4	5	6	7	14
8	9	10	11	12	13	14	15
15	16	17	18	19	20	21	16
22	23	24	25	26	27	28	17
29	30						18

This can result in the database claiming blocks that are marked free but is occupied. This can be ~~fixed~~ dealt/detected by examing the block that is fetched. We can then find another entry in free space map

If the block is marked used, but is actually free, then it will not result in a problem since it only results in unused free space.

As a result, the relation is scanned periodically and the free spacemap is recomputed and written to disk.

Sequential file organization:

This "Sequential file" is designed for efficient processing of results/records in sorted order based on a search key.

A search key is an attribute / set of attributes. To permit fast retrieval of records in search key order. We chain together records by pointers.

The pointer in each record points to the next record in search key order.

To minimize number of block accesses, we store records physically in search key order, or as close to search key order as possible.

This allows data to be accessed in sorted order but is difficult to maintain physical sequential order as records are inserted and deleted.

We can manage deletion by pointer chains.

For Insertion we can apply the following rules:

MARCH - 2024

Wk	M	T	W	T	F	S	S
09					1	2	3
10	4	5	6	7	8	9	10
11	11	12	13	14	15	16	17
12	18	19	20	21	22	23	24
13	25	26	27	28	29	30	31

*) Locate the record that comes before the record to be inserted.
*) If there's a free record (space left after deletion) within the same block as this record, insert it there. Otherwise, insert the new record in an overflow block. In either adjust the pointers so as to chain together the records in search key order.

*) Locate the record that comes before the record to be inserted.
*) If there's a free record (space left after deletion) within the same block as this record, insert it there. Otherwise, insert the new record in an overflow block. In either adjust the pointers so as to chain together the records in search key order.

If relatively few records need to be stored in overflow blocks, then this approach works well.
However the search key order might eventually be ~~dist~~ no longer satisfied over a period of time.
In this case sequential order will be less efficient and files should be reorganized so that is in sequential order again.
Such reorganizations are costly and must be done when the system load is low.
The frequency of the reorganizations depends on the frequency of the insertions.

Multitable clustering file organization:
Most Relational DBMS store relations in a separate file / separate set of files.
*) Thus each block has records of a single relation.
*) In some cases it is useful to store records of more than one relation in a single block.
*) A Multitable clustering file organization is a file organization that stores related records of two or more relations in each block
*) Cluster key is the attribute that defines which records are stored together.
*) Although this can speed up join queries, it slows down processing of other queries since it results in more block accesses (lesser records stored due to higher record size).

Love is composed of a single soul inhabiting two bodies. - Aristotle

APRIL - 2024

M	T	W	T	F	S	S	Wk
1	2	3	4	5	6	7	14
8	9	10	11	12	13	14	15
15	16	17	18	19	20	21	16
22	23	24	25	26	27	28	17
29	30						18

*) we can use pointers to increase efficiency but it results in the same no. of block reads.
*) It can be used when join type queries are the most frequent.

Partitioning:
Databases allow given records in a relation to be partitioned into smaller relations stored separately such partitioning is done on the basis of an attribute value.
*) Eg: year: 2019, 2020 . . .
table_2019, table_2020.
*) Query optimizers can then translate these queries on the original table to queries on the smaller relation.
Eg: SELECT * FROM x
WHERE Year = "2019";

*) Partioning helps reduce overheads such as finding space for a record, increase with relation size.
*) we can also store infrequently accessed records /partitions in Hard Disk, and frequently accessed partitions in SSD's.

Data Dictionary: Data Dictionary is a data structure that stores information / metadata about other relations
*) Also known as System Catalog

The Data Stored by the catalog:

- Name of relation
- Names of Attribute of each relation
- Domains / Lengths of Attribute.
- Names of views in DB / Definition of views.
- Integrity constraints (Key constraints).
- Name of users, Default Schemas, passwords.
- Information about authorizations for each user.
-

The Database can also store statistical and descriptive data about the relations like the number of tuples in each relation / number of distinct values of each attribute.

It also stores information about storage organization of relations and where each relation is stored.

↳ If ~~each~~ relations are stored in OS files, the dictionary would note the names of the file / files containing each relation.

↳ If Database stores all relations in a single file, the dictionary notes the blocks of all the containing records of each relation in a linked list.

↳ This metadata is a miniature database and often times, this metadata is stored in relations to simplify the structure of the database.

↳ The Metadata is not normalized for efficient access.

↳ Whenever the DB must retrieve information ~~about~~ from relations, it must consult the RelationMetadata first and then find the location and storage organization

↳ The location / storage Organization about the metadata is hardcoded / stored in database. Since it is frequently accessed, we store it in-memory when the system starts.

APRIL - 2024

M	T	W	T	F	S	S	Wk
1	2	3	4	5	6	7	14
8	9	10	11	12	13	14	15
15	16	17	18	19	20	21	16
22	23	24	25	26	27	28	17
29	30						18

Database Buffer:

The size of main memory on servers has increased greatly. But still not enough for all the database to be stored completely in the memory but only some parts of it.

A Major goal of a DB is to minimize the no. of block transfers. One way to accomplish this is to keep as many blocks as possible in main memory, the goal being to maximize the chance that when a block is accessed, it is already in main memory.

Since it is not possible to keep all the blocks in main memory we need to manage the allocation of blocks in the available main memory.

Buffer for a database keeps the copies of disk blocks in main memory. The system responsible for the allocation of buffer space is called the buffer manager

Buffer Manager: Programs in a DB make requests / calls to the Buffer Manager when they need a block from disk.
If the block is in the buffer, the manager passes the address of the main memory of to the requester.

*) If the block is not in the buffer, the buffer manager allocates space for the new block.

*) If necessary, it throws out another block to make space. The thrown out block is written to disk, if any modifications are done.

*) Then the buffer manager reads the block into the main memory and pass the address to the requester.

*) This is analogous to an virtual memory manager ~~and~~ but the buffer manager deals with blocks larger

MARCH - 2024

Wk	M	T	W	T	F	S	S
09					1	2	3
10	4	5	6	7	8	9	10
11	11	12	13	14	15	16	17
12	18	19	20	21	22	23	24
13	25	26	27	28	29	30	31

The best way to keep one's word is not to give it. - Napoleon Bonaparte

than the memory available in a machine.

Buffer Replacement Strategy:

when no room is left in buffer, a block is evicted

LRU (Least Recently used) strategy is used in databases.

Block Pinning:

To avoid unloading/evicting blocks that are being currently used, the reader "pins" the block (marks it) and starts reading the block.

while a block is pinned, it is never evicted and the reading process must unpin after it completes its task.

Multiple Processes can pin and unpin. Therefore we use a pin count for each buffer block and blocks with pin count == 0 will be evicted.

Exclusive & Shared locks in Buffer Blocks:

- Shared Locks: Many people
- Exclusive lock: only one
- A shared lock is obtained only when we want to read. and multiple people can acquire it at the same time.
- locked blocks must be unlocked.
- only one process must have one lock at a time (exclusive) and if a block has a exclusive lock no shared lock is granted.
- Exclusive locks are granted only when all previous locks are unlocked.
- shared lock is granted when no exclusive lock is present.
- Blocks must be pinned and then locked and unpinned
- for updating a block, they must have an exclusive lock.

APRIL - 2024

M	T	W	T	F	S	S	Wk
1	2	3	4	5	6	7	14
8	9	10	11	12	13	14	15
15	16	17	18	19	20	21	16
22	23	24	25	26	27	28	17
29	30						18

Output of Blocks:

We can evict a block only when necessary but it makes sense to do so even before a need arises.

- A Block that is not pinned can be evicted
- Block that is being updated must not be evicted.

Forced output of Blocks:

For some situations it is necessary to evict blocks to ensure that a hard disk is in a consistent state.

- Such a write is called Forced output.
- We uses forced outputs with a logging system that is used to recover the block since during a forced output, the blocks are in an inconsistent state.

Buffer - Replacement strategy:

- Least Recently Used (LRU)
- Toss Immediate
- Most Recently Used (MRU).

If the Buffer manager is allowed to communicate with the concurrency control system, then it can easily toss the blocks that have its requests being delayed.

Column oriented storage:

Databases traditionally store all attributes of tuple together in a record, such a storage layout is referred to as a row oriented storage

In column-oriented storage / columnar storage each attribute of a relation is stored separately, with values of the

attributes from successive tuples stored at successive positions in the file.

* Each attribute is stored in a separate file, each file is commerced to reduce its size.
* If ith row of a table, the values at the ith position in each of columns are retrieved and used to reconstruct the row.

column-oriented storage has the drawbacks that multiple attributes of a single tuple multiple I/O operations.

Reasons to use column oriented storage:

* Reduced I/O: Only required attributes are fetched.
* Improved CPU cache performance: When the query processor fetches the contents of a particular attribute, with modern CPU architectures multiple consecutive bytes, called cache line are fetched from memory to CPU cache. If these bytes are accessed later, access is much faster if they are in cache. Row oriented storage ~~is~~ suffers from cache wastage, but column oriented storage does not suffer from cache wastage.
* Improved compression: Storing values of the same types together significantly increases the effectiveness of compression, compared to compressing data stored in row format.
* Vector Processing: Many modern CPU architectures support vector processing which allows a CPU operation to be applied in parallel on a number of elements of an array. Storing data columnwise allows vector processing of operations such as comparing with a constant.

Drawbacks: cost of tuple Reconstruction: reconstructing a tuple from the individual columns can be expensive.

APRIL - 2024

Mo	T	W	T	F	S	S	Wk
1	2	3	4	5	6	7	14
8	9	10	11	12	13	14	15
15	16	17	18	19	20	21	16
22	23	24	25	26	27	28	17
29	30						18

Cost of Tuple Deletion & update: Deleting or updating a single tuple in a compressed representation is expensive. Data warehouses therefore only allow insertions and bulk deletes.

Cost of Decompression: Fetching data from a compressed representation requires decompressing the entire file.

To combat this however databases allow compression from any arbitrary point. and typically compression is done for every 10000 values.

Therefore to access i^{th} value from DB, we need to decompress $\lfloor i/10000 \rfloor^{th}$ group.

Indices:

Indices are critical for efficient processing of queries in databases.

There are two types of Indices:

↳ Ordered Indices:

↳ Hash Indices:

Each technique must be evaluated on the basis of these factors:

⇒ Access Types: The types of access that are supported efficiently. It can include a specific value or a range of values.

⇒ Access Time: The time it takes to access a value / set of values.

⇒ Insert time

⇒ Deletion time.

Space overhead: Additional space occupied by an index structure.

Note that Insertion Time / Deletion Time also includes the time to update the data structure.

An Index Relation can have many indices on different attributes and these attributes can be is called as a search key.

Ordered Indices: An ordered index stores the values of an search key in sorted order and associates with each search key ~~that~~ the records that contain it.

* If the file containing the attributes is sequentially ordered then the index on the attribute is called a clustering index
* This type of index is also called as Primary Index.
* The search key of a clustering index is often the primary key but it is not required to be the primary key.
* Indices on attributes that are not sequentially ordered are called as non-clustering indices / secondary indices
* ~~An~~ Clustering Index files are called as Index sequential files

Dense / Sparse Indices:

An Index Entry / Index Record consists of a search key value and pointers to one more records with that value.

The pointer contains the identifier of the disk block and an offset within the disk block.

Dense Index: In an dense index, an entry appears for every search key value. In a Dense clustering index, the entry contains the search key and a pointer to the first data record with the search key value. (The Rest of them would be stored sequentially)

APRIL - 2024

M	T	W	T	F	S	S	Wk
1	2	3	4	5	6	7	14
8	9	10	11	12	13	14	15
15	16	17	18	19	20	21	16
22	23	24	25	26	27	28	17
29	30						18

In a Dense Nonclustering Index, the Index must store a list of pointers to all records with the same search key value.

Sparse Index: In a sparse Index, an Index entry appears for only some of the search key values and can only be used for clustering Indices.
*) To locate a record value, we first locate the largest key value that is less or equal to the search key value for that we are searching.

Multilevel Indices: Since Index entries are always sorted, we can treat it as a sequential file and construct Indices on the dense Index. and it can be sparse.
Indices with two or more levels are called as multilevel Indices. Searching for records with a multilevel Index requires significantly fewer I/O operations than does searching for records in Binary search.

Index Update: Regardless of the type of Index, we need to ~~decide~~ update our Index, if records are inserted/deleted. If records are updated, I don't know what to do.

Insertion: 1.) we first lookup using the search key value in the record to be inserted.
2.) Dense Index: If the search key does not appear in the Index, we insert the entry at the appropriate pos.
*) If it appears in a nonclustering manner, we insert the pointer inside the Index.
*) If it is ~~non~~clustering manner, we insert the record after the other records with the same search key values.

Sparse IndPces: we assume that the Pndex stores an entry for each block. If the system creates a new block, It Pnserts the fPrst search key value Pn order, appearPng Pn the new block. If It Pz Pnserted Pn a preexPstPng block, we ~~up~~ Pnsert It Pn order and update the Pndex entry to the least search key value

Deletion:

Dense IndPces: If the deleted record Pz the only record wPth the search key value, the system deletes the correspondPng Pndex entry from the Pndex

*) If there's already an entry, and Pf It Pz stored Pn non-clusterPng manner, the system deletes the pPnter to the deleted record from the Pndex.

*) If It's stored Pn a clusterPng manner, and there's an entry, and Pf the deleted record Pz the fPrst appearPng record, we delete It's entry ~~and~~ (update the Pndex entry to point to the next record).

Sparse IndPces:

*) If there's no entry, there's nothPng to be done.

*) If the deleted record was the only record wPth It's search key, the system replaces the correspondPng record wPth an Pndex record for the next search key value (Pn search key order). If the next search key value already has an Pndex entry, the entry Pz deleted.

*) If the Pndex entry for the search key value poPnts to the record bePng deleted, the system updates the Pndex entry to point to the next record wPth the same search key value.

MulPlevel IndPces' ~~ope~~ Pnserts & updates are the same as the algorithm presented ~~below~~ above. wPth the hPgher level Pndex bePng updated after the lower level Pndex updated.

APRIL - 2024

M	T	W	T	F	S	S	Wk
1	2	3	4	5	6	7	14
8	9	10	11	12	13	14	15
15	16	17	18	19	20	21	16
22	23	24	25	26	27	28	17
29	30						18

Secondary Indices : Secondary Indices (Non clustering) must be dense with an index entry for every search key value.

* A secondary index on a candidate key looks exactly like a dense clustering index except the next records are not stored sequentially.

* If the secondary index search key is not a candidate key, it is not enough to point to just the first record with each search key value, therefore we must store pointers to all the records.

* If the relation can have more than one record with the same search key value, the search key is said to nonunique search key.

* One way to implement ~~search key~~ secondary index on nonclustering search key is to point the index entry to a bucket of pointers that contains pointers to a file.

* This ~~two~~ approach has few drawbacks, first, index access takes longer due to an extra level of indirection.

* If a key has very few/no duplicates, a whole block is allocated and is completed as a waste of space.

The procedure described for deletion & insertion can also be applied to secondary indices; the actions taken are those described for dense indices ~~storth~~ (non-clustering).

Secondary indices improve the performance of queries that use keys other than the search key of the clustering index. however, they impose a significant overhead on modification of databases.

* Indices on Multiple Keys.

We can have multiple attributes for a search key, a search key containing more than one attribute is called as composite search key.

B+ Tree Index Files:

B+ - Tree Index structure is the most widely used of several index structures, that maintain their efficiency despite insertion and deletion of data.

* It takes the form of a Balanced Tree, in which every path from the root is of the same length.
* Each non-leaf node in the tree has (except root) has between $\lceil n/2 \rceil$ and n children, where n is fixed
* the Root has between 2 to n children.
* There are some space overhead (half empty nodes), performance overhead, it is acceptable since it avoids file reorganization

Structure of B+ Tree:

* The B+ tree index is a multilevel index, but it has structure that differs from that of multilevel index-sequential file.
* we assume that the search key is unique and occurs at most in one record.
* A Node contains n-1 search key values $K_1, K_2, \ldots K_{n-1}$ and n pointers $P_1, P_2, P_3 \ldots P_n$.
* the search key values are kept in sorted order.

Structure of Leaf Nodes:

* For $i = 1, 2, \ldots n-1$, pointer P_i points to a file record with search key value K_i, P_n has a special purpose.
* Each leaf node can contain upto n-1 values and we allo them to hold atleast $\lceil (n-1)/2 \rceil$ values.
* Eg: n=4, then leaf nodes contains atleast 2 nodes and at most 3 nodes.

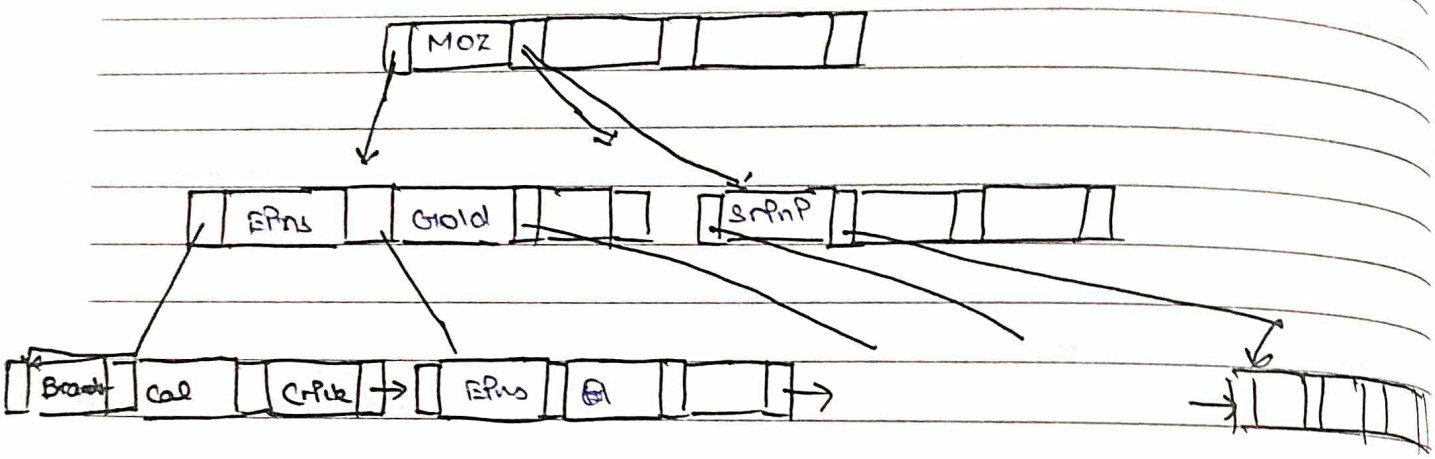
APRIL - 2024

M	T	W	T	F	S	S	Wk
1	2	3	4	5	6	7	14
8	9	10	11	12	13	14	15
15	16	17	18	19	20	21	16
22	23	24	25	26	27	28	17
29	30						18

Every search key value in Li is lesser than every search key value, given Li, Lj are leaf nodes and i < j.

*) B+ tree index is used as a dense index (usually) every search key value must appear in some leaf node.

*) In is used to chain together the leaf nodes in search key order.



*) The Non Leaf Nodes of the B+ tree form a multilevel sparse index on the leaf nodes.

*) It is the same as a leaf node, except all pointers are pointers to tree nodes.

*) Fanout of a Node: Number of pointers in a node.

*) Nonleaf nodes: Internal Nodes.

*) Root Node must hold atleast 2 node pointers unless there's only one node in the B+ tree.

*) Since B+ Trees are balanced (length of path from root to leaf is equal across all paths), it ensures that it is good for lookup, insertion, deletion.

*) If Search keys are non unique, we can deal with it in three ways:

- store each search key at a leaf node as many times it appears in records, with each copy pointing to one record.
- This approach can lead to complication of deletion and insertion operation and also make them expensive.

- Second Alternative is to store a bucket of record pointers with each search key value, this is more complicated and can result in inefficient access, especially if the duplicate count is large.

- Most Database Implementations make the nonunique search key unique by adding the primary key to ~~be~~ the search key and make it unique.

Queries in B+ Trees:

```
find (v):
    let c ← root (tree)
    while (! leaf (c)) do
        let i ← smallest number such that v ≤ c.Ki
        if i == -1 then
            let Pm ← last non-null pointer in node
            c ← c.Pm
        else if v = c.Ki then
            c ← c.Pi+1
        else
            c ← c.Pi
        end
    end
    if for some i, Ki = v
        return Pi
    else return NULL
```

We can't help everyone, but everyone can help someone. - Ronald Reagan

APRIL - 2024

M	T	W	T	F	S	S	Wk
1	2	3	4	5	6	7	14
8	9	10	11	12	13	14	15
15	16	17	18	19	20	21	16
22	23	24	25	26	27	28	17
29	30						18

The function starts at the root of the tree and traverses the tree down until it reaches a leaf node that would contain the specified value if it exists.

*) The following steps are repeated until a leaf node is reached.

1) The current node is examined looking for the smallest P such that KP is greater than or equal to v.

1.1) If value is found, i.e if KP is equal to v, the current node is set to PPH

1.2) If KP > v, the C is set to the previous pointer PP

1.3) If no such value KP is found, then v > Km-1, (greater than the last key) and current node is set to Pm (last non null pointer)

2) At the leaf node, if KP = v, pointer PP directs us to the record with search key value KP

3.) If found, PP is returned, else NULL is returned.

B+ trees can also be used to find all records with search key values in a specified range [lb, ub]

We first similarly find the node that is equal to or greater than lb and once we visit the leaf node with search key value ≥ lb, we traverse nodes until the search key value is lesser than or equal to ub.

All the nodes visited by the function will be added to a result set which is returned.

Cost of Querying B+ Tree: If there are N records the path is no longer than

$\lceil \log_{\lceil n/2 \rceil}(N) \rceil$

Eg: If $n=100$, $N = 1,000,000$
then $\log_{50}(1000000) = 4$.
Since root node is heavily accessed, it is kept inside the buffer.

Since B+ trees ~~ha~~ can contain many pointers, ~~so~~ it tends to be fat and short.

$\text{Find}(v) = \lceil \log_{\lceil n/2 \rceil}(N) \rceil + 1$
$\text{FindRange}(lb, ub) = \lceil M/(n/2) \rceil + 1$ leaf nodes need to be accessed
↓
Non-clustered Indices : M random I/O operations.
↓
clustered Indices : consecutive therefore lesser I/O operations.

Querying with Non-unique search keys: Indices:
with the composite key (a_i, A), we can find the all the occurrences of v using the range function with parameters $(v, -\infty)$ as lb and (v, ∞) as ub.

Updates on B+ Trees: Updates on B+ trees are essentially a deletion followed by a insertion.

~~Dream~~ Now we will deal with insertion:
Using the same technique as for lookup from the find() function we first find the leaf node would appear, we then insert an entry in the leaf node such that ~~the~~ the search keys are still in order. If there's no need for splitting of nodes, then insertion is done.
*) If there's no room to store the values to be inserted we first split it into two and take the new ~~Nat~~ Node and add it's entry to it's parent non-leaf node.

APRIL - 2024

M	T	W	T	F	S	S	Wk
1	2	3	4	5	6	7	14
8	9	10	11	12	13	14	15
15	16	17	18	19	20	21	16
22	23	24	25	26	27	28	17
29	30						18

*) The New Splitted Node's first search key value is updated to its parent.
*) If there's no further space in the parent then we split it ~~over~~ it and so on.
*) If the root itself is split, then, ~~we~~ the tree becomes deeper.
*) ~~Insert~~ Splitting of Non-Leaf Node is a little different from splitting of a leaf node. When the overful non leaf node is split, the child pointers are divided among the original and the newly created nodes
*) The first value in the newly splitted node is removed and an entry is inserted into its parent.
*) In general, we create entries until now

Deletions: using the same technique for lookup, we find the leaf node containing the entry to be deleted. We then remove the entry from the leaf node. All entries in the leaf node that are right to entry are shifted left by one position.

*) Now we consider the case where there are less number of nodes in the leaf (len < $\lceil (n-1)/2 \rceil$). We must merge the nodes or redistribute the entries.

*) If we can merge the entries in an underfull node to an sibling node. Once this is done we must also delete its entry in the parent node that pointed to the just deleted node

MARCH - 2024

Wk	M	T	W	T	F	S	S
09					1	2	3
10	4	5	6	7	8	9	10
11	11	12	13	14	15	16	17
12	18	19	20	21	22	23	24
13	25	26	27	28	29	30	31

*) If the parent becomes underfull we try to either coalesce or redistribute the pointers b/w the node

To Delete a value in a B+ tree, we perform a lookup on the value and delete it. If the node is too small, we delete it from its parent. This deletion results in a recursive application of the algorithm until the root is reached, a parent remains adequately full after deletion or redistribution

Complexity of B+ Tree updates:

Worst case for an insertion : $\log_{\lceil n/2 \rceil}(N)$

" " deletion : $\log_{\lceil n/2 \rceil}(N)$

with a fanout of 100, only 1 in 100 to 1 in 50 insertions would result in a node split.

B+ Tree Extensions:

We solve the degradation problem for storing the actual records by using the leaf level of B+ tree, not as an index.

In B+ tree file organization, the leaf nodes of the tree stores records instead of pointers.

When we use B+ file organization, space utilization is more important, since the space occupied by the records is much more than the space of keys and pointers.

*) We can do so by proportioning redistribution of entries to the adjacent nodes

*) If this fails, we split the node and divide its entries equally.

*) During deletion of a record, if the occupancy falls below $\lfloor 2n/3 \rfloor$, the system attempts to borrow an entry from one of the sibling nodes.

*) If the sibling nodes both contain $\lfloor 2n/3 \rfloor$ nodes, we redistribute the entry evenly between the nodes and delete the third node.

*) During initial construction, we allocate blocks contiguously but as insertions and deletions occur, the sequentiality is increasing cost. During which an index rebuild is required.

Secondary Indices and Record Relocation:

*) B+ may change the location of records even when they are updated (splitting of records).

*) When they are changed, we need to update the pointers in secondary indices.

*) To combat this, we need to store the values of primary-index ~~or~~ search key attributes.

*) First we use the secondary index to find the primary index value and then find the corresponding records.

Indexing strings:

*) for variable length strings, we use prefix compression by storing the prefix as a search key value that is sufficient to distinguish

Hashing Indices: used for building indices in main memory
Hashing can also be used for file organizations (not widely used).

*) Bucket: unit of storage that can one or more records
For In-Memory Index: A linked list of entries
For Disk Based Index: A Bucket is a linked list of disk indices.

File organization: Buckets store the actual records

APRIL - 2024

Wk	M	T	W	T	F	S	S
14	1	2	3	4	5	6	7
15	8	9	10	11	12	13	14
16	15	16	17	18	19	20	21
17	22	23	24	25	26	27	28
18	29	30					

A Hash function is a function from K to B
(K: Search Key, B: set of Buckets)

Insert a record; we compute h(KP) which gives the address of that record, we add the index entry for the record to the list at the offset.

*) Deletion is straight forward, we just delete the entry

*) collisions are dealt with different strategies.

↳ Using Addressing (overflow chaining)

↳ Open Addressing, not widely used.

*) Unlike B+ trees, there's no range queries.

Static Hashing: Number of buckets is fixed when index is created.

Can result in overflow buckets

→ Therefore can become inefficient.

Dynamic Hashing: Increase buckets as ~~record~~ required.

Covering Indices: Indices that store the values of some attributes (with search key attributes) to answer queries using only the index.

Create Index:

CREATE INDEX <index-name> on <relation> (<attribute(s)>)

DROP INDEX <index-name>

CREATE UNIQUE INDEX (on candidate keys).

Bitmap Indices: Bitmap Indices are a specialized type of index for easy querying.

*) Records in a relation must be numbered sequentially.

*) Bitmap: Array of Bits.

MAY - 2024

M	T	W	T	F	S	S	Wk
		1	2	3	4	5	18
6	7	8	9	10	11	12	19
13	14	15	16	17	18	19	20
20	21	22	23	24	25	26	21
27	28	29	30	31			22

Tuple Calculus:

An Non Procedural Query Language in the form

$\{t \mid P(t)\}$

Set of tuples, satisfying predicate P.

$t[A] \Rightarrow$ value of attribute A on t.

Allowed Terms in P(t):

- Constants / Attributes
- Comparison operators
- Connectives ($\vee, \wedge, \neg$)
- Implication ($\Rightarrow$)
- Quantifiers: $\exists, \forall$

Eg: $\{t \mid t[\text{Salary}] > 80000\}$

we can generate infinite sets by accident therefore we must be careful.

Domain Relational calculus:

$\{\langle P, d, n, s\rangle \mid \langle P\rangle \in \text{Instructor} \wedge \langle s\rangle > 80000\}$

April

2024 April 04 Thursday

Wk 14 (095-271)

Transactions control:

Transaction concept: A transaction is a unit of program execution that accesses and possibly updates various data items.

*) A transaction is delimited by statements of the form:

begin transaction

:

end transaction

This collection of steps must appear to the user as a single, indivisible unit. It executes in its entirety or not at all, thus if a transaction begins to execute but fails for whatever reason, any changes to the database must be undone.

*) This all or none property is referred to as atomicity

*) Since, transaction is a single unit, it cannot appear to be separated by other database operations not part of the transaction

*) Thus, Database system must take special actions to ensure that transactions operate properly without interference from concurrent database systems' statements. This is known as Isolation.

*) If a transaction gets interrupted after finishing and the database doesn't persist its actions, ~~the~~ it's useless, therefore we persist a transaction's actions, This is known as Durability.

*) ACID Properties:

Atomicity: Either all operations of a transaction are reflected properly in the database or none.

Consistency: Executions of a transaction in isolation preserves the consistency of the database.

Isolation: Each transaction is unaware of other transactions executing concurrently in the system.

Durability: After a transaction is completed, its changes are persisted ~~to~~ even if there are system crashes.

MAY - 2024

M	T	W	T	F	S	S	Wk
		1	2	3	4	5	18
6	7	8	9	10	11	12	19
13	14	15	16	17	18	19	20
20	21	22	23	24	25	26	21
27	28	29	30	31			22

No matter what people tell you, words and ideas can change the world. - Robin Williams

Concurrency control system:
Ensures that Isolation property is enforced in the Database.

*) For a transaction to be Durable, its changes need to be written to stable storage.
*) For a transaction to be Atomic; the log records must be written to stable storage before any changes are made.

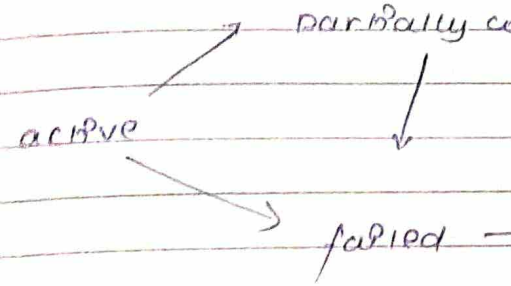
Transaction Atomicity & Durability:
A Transaction may not always complete its execution successfully, such a transaction is termed as aborted.
*) To Enforce atomicity, Aborted Transactions must have no effects on database.
*) Once changes of an aborted transaction are undone, we say that the Transaction is rolled back.
*) This done by maintaining a log, which stores each database modification by a transaction.
*) The Log contains the Id of transaction, Identifier of the data item being modified, and both the old value and the new value.
*) A Transaction that completes its execution successfully is said to be committed, which brings the database to a new consistent state and the changes must persist.
*) A committed Transaction cannot be undoed
*) Active State: Initial state, Transaction is active when it Executes the statements.
*) Partially committed: After final statement is executed, a transaction is said to be in partially committed state.

APRIL - 2024

Wk	M	T	W	T	F	S	S
14	1	2	3	4	5	6	7
15	8	9	10	11	12	13	14
16	15	16	17	18	19	20	21
17	22	23	24	25	26	27	28
18	29	30					

Happiness is not something ready made.
It comes from your own actions. - Dalai L

Abstract Model of the Transaction:



failed: After the discovery, that a transaction ~~can be~~ cannot be normally executed,

Aborted: After the transaction has been rolled back and the database has been restored to its state prior to the start.

committed: After successful completion.

Terminated: If it either has been committed or aborted.

A transaction starts out in active state, when it finishes its final statement, it enters partially committed state. At this point, the transaction can either fail (not writing out enough information to recover from failure, to stable storage) or ~~to~~ when it writes ~~the~~ enough information to ~~complete execution~~ ~~it can~~ recover the changes, the transaction enters committed state.

An Aborted Transaction can be either:
1) Restarted, only if the transaction has been aborted as a result of some hardware / software error.

MAY - 2024

M	T	W	T	F	S	S	Wk
		1	2	3	4	5	18
6	7	8	9	10	11	12	19
13	14	15	16	17	18	19	20
20	21	22	23	24	25	26	21
27	28	29	30	31			22

Things don't have to change the world to be important.- Steve Jobs

2) Kill the transaction, if it is aborted due to some internal logic error

Be careful with observable external writes (observable by the external entities).

- Reasons for concurrent transaction execution:
  - Improved throughput & resource utilization:
    Throughput: No. of transactions executed in a given amount of time.
  - Reduced waiting time: Reduces average response time.

- Schedules: Represent the chronological order in which transactions are executed on a system. (Has all the instructions a transaction executes)

- Serializable: The concurrency component of the database system ensures that any schedule that is executed will leave the database in a consistent state.
  - We can ensure this consistency by making sure that the schedule being executed concurrently produces the same effect as a schedule being serial.
  - Such schedules are called as serializable.

Conflict Serializable: for any schedule S, in which there are two consecutive instructions, I and J of transactions T_i and T_j ($i \neq j$)
If I & J refer to different data items, we can swap I, J without affecting the results of the transaction

- If I & J refer to the same data items, then the order of two steps matter.

There are four cases (since we are dealing with read(x) & write(x).)

case I: I = read (Q)
J = read (Q)
The order of I, J does not matter, since the same Q is read by T_i and T_j regardless of the order.

case II: I = read (Q)
J = write (Q)
If I comes before J, T_i does not read the value of Q written by T_j. If J comes before I, Then T_i reads the value of Q written by T_j, Thus the order of I & J matters.

case III: I = write (Q)
J = read (Q)
see case II

case IV: I = ~~rea~~ write (Q)
J = write (Q).
Since both instructions are write operations, the order of these instructions does not affect either T_i or T_j However the value obtained by the next read (Q) of S is affected, since the the latter write affects the read (Q).
If there's no other write (Q) instruction after I, J in S, then the order of I and J directly affects the final value of Q in the database state

We say that two instructions are in conflict, if they are operations by different ~~operations~~ transactions on the same data item and at least one of these is a write operation.

MAY - 2024

M	T	W	T	F	S	S	Wk
		1	2	3	4	5	18
6	7	8	9	10	11	12	19
13	14	15	16	17	18	19	20
20	21	22	23	24	25	26	21
27	28	29	30	31			22

let I and J ~~conflict~~ be consecutive instructions of a schedule S, if I & J are of different transactions and do not conflict then we can swap the order of I and J to produce a new schedule S' equivalent to S.

If a schedule S can be transformed into S' by a series of swaps of non-conflicting instructions, then S & S' are conflict equivalent.

We say that a schedule S is conflict serializable if it is conflict equivalent to a serial schedule

Determine if a schedule is conflict serializable:
we construct a precedence graph from S. The set of Edges $T_i \rightarrow T_j$ for which one of three conditions holds:
1.) T_i executes write(Q) before T_j executes read(Q)
2.) T_i executes read(Q) before T_j executes write(Q)
3.) T_i executes write(Q) before T_j executes write(Q)
If an Edge $T_i \rightarrow T_j$ exists in the precedence graph, then in any serial ~~ser~~ schedule S' equivalent to S, T_i must appear before T_j
If the precedence graph for S has a cycle, then schedule S is not conflict serializable. If the graph contains no cycles, then the schedule S is conflict serializable.

Serializability Order: A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This is topological sorting.

Topological sorting can be done by simply visiting all nodes in a particular order such that it satisfies some constraints.

Conflict Serializability: we need to construct the precedence graph and check if there are any cycles / loops.

Transaction Isolation & Atomicity: If transaction Ti fails, we need to undo the effect of this ~~concurrent~~ transaction, to ensure the atomicity property
*) If an concurrent environment, if Tj depends on Ti (i.e Tj has read data written by Ti), Tj is also aborted.

Recoverable Schedule: A Schedule is recoverable if it allows for the recovery of the database to a consistent state after a transaction failure.
*) A Recoverable Schedule has a transaction that has updated the database will commit before any other transaction reads or writes the same data.
*) If a transaction aborts / fails before committing, its updates must be rolled back and any transaction that has read it will also be rolled back.

Cascading ~~Cascaderless Schedule~~ / Rollback: A single transaction failure leads to a series of transaction rollbacks, ~~is a~~.

Cascadeless Schedule: A Schedule, where each pair of transactions Ti and Tj such that Tj reads a data item previously written by Ti. The commit operation of Ti appears before Read of Tj.

Cascadeless Schedules are also recoverable.

Isolation Levels: Serializable: This isolation level ensures that execution is serializable.

Repeatable Read: Allows only committed data to be read and between two reads of a data item, no transaction is allowed to update it.

MAY - 2024

M	T	W	T	F	S	S	Wk
		1	2	3	4	5	18
6	7	8	9	10	11	12	19
13	14	15	16	17	18	19	20
20	21	22	23	24	25	26	21
27	28	29	30	31			22

Read committed: Allows only committed data to be read. But not repeatable reads is not required.

Read uncommitted: Allows uncommitted data to be read.

Dirty writes: ~~Disallow~~ writes to a data item that is already written and not yet committed/aborted.

Automatic commit: committing as soon as a single statement is executed.

weaker Isolation Level = Higher System Performance

Implementation of Isolation Levels:

There are various concurrency control policies that we can use to ensure that, only acceptable schedules are generated

*) The Goal - High Degree of concurrency while ensuring all schedules are conflict/view serializable, recoverable and cascadeless.

Locking: A concurrency control mechanism, previously seen (shared and exclusive).

Timestamp: Transaction gets a timestamp when it begins and a database ~~has a~~ item has a read & write timestamp which contains the recent transaction's timestamp.

*) Timestamps are used to ensure that access of data is in order.

Wk	M	T	W	T	F	S	S
14	1	2	3	4	5	6	7
15	8	9	10	11	12	13	14
16	15	16	17	18	19	20	21
17	22	23	24	25	26	27	28
18	29	30					

Snapshot Isolation: Snapshots are images of the state of a database. In this policy, the transaction acts on snapshots and are then updated after its execution

is complete.

Locking: Locking can lead to an undesirable situation called deadlock where transactions can proceed with its normal execution.
*) To come out of this situation, we must rollback one of the transaction which unlocks all the data holded by the transaction.

Locking protocol: A set of rules that each transaction in the system follows. Indicating when a transaction may lock and unlock.
*) This restricts the possible schedules in the scheme.

We say that a schedule S is legal under a given locking protocol, if S is a possible schedule for a set of transactions that follows the rules of the locking protocol.

A Locking protocol ensures conflict serializability if and only if all legal schedules are conflict serializable

Granting of locks:
A Transaction that may never makes progress due to being unable to acquire locks is said to be starved.

We can avoid starvation by following that:
*) No transaction holding a lock on Q that conflicts with mode M.
*) No transaction that is waiting for a lock on Q that made its request before TP

Two-Phase Locking Protocol: Issuing of locks take place in two phases: *) Growing Phase (acquire locks)
*) Shrinking phase (release locks)

MAY - 2024

M	T	W	T	F	S	S	Wk
		1	2	3	4	5	18
6	7	8	9	10	11	12	19
13	14	15	16	17	18	19	20
20	21	22	23	24	25	26	21
27	28	29	30	31			22

Two Phase locking protocol ensures conflict serializability but does not ensure freedom from deadlock.
The point in which a schedule where the transaction has obtained its final lock is called the lock point.

Cascading Rollbacks can be avoided by a modification of the two phase protocol
* Strict Two-Phase locking protocol: Requires schedule to follow two phase locking protocol, but also all exclusive locks be held until the transaction commits.
* This ensures that other transaction ~~from~~ does not read uncommitted data
* Rigourous Two-Phase locking protocol: Requires two phase locking protocol and also all shared locks to be held.

Locking conversion can take place in schedules.
Upgrade (a_1): Shared to Exclusive, only in growing phase
Downgrade (a_1): Exclusive to Shared, only in shrinking phase.

A simple but widely used scheme:
* Obtain shared lock when reading Q
* Obtain exclusive lock when writing to Q, if there's already a shared lock, upgrade to exclusive.

Deadlocks: Deadlock prevention (used when deadlock probability is
Deadlock detection & recovery

APRIL - 2024

Wk	M	T	W	T	F	S	S
14	1	2	3	4	5	6	7
15	8	9	10	11	12	13	14
16	15	16	17	18	19	20	21
17	22	23	24	25	26	27	28
18	29	30					

Deadlock Prevention: Two approaches in deadlock prevention, one: No cyclic waits by ordering the requests for locks or requiring all locks to be acquired together.

other is to rollback instead of waiting for a lock.

Disadvantages: *) Hard to predict when data items are required to be locked
*) Data utilization can be very low.

Preemption: Granting a lock to Ti from Tj by rolling back Tj
Preemption Schemes: *) wait-die, Older transactions are allowed to wait else they are rolled back, Non preemptive
*) wound-wait: Younger Transactions are allowed to wait, Preemptive.

Lock Timeouts: A Transaction waits for atmost a specified amount of time, easy to implement but starvation is also possible.

Detection & Prevention: using a dependency graph, we can detect deadlocks and if there are cycles in the graph, there's a deadlock.
(dependency graph: wait for graph).

Recovery: *) Selection of a victim
*) Rollback
*) Avoid starvation (Don't Rollback the same transaction too many tim