# Week1

- Book-keeping, from a data management perspective, is the process of physical data or records management using physical ledgers and journals.
- Spreadsheet files are better than paper-based book-keeping in terms of: Durability, Scalability, Security and Consistency.
- Following are some disadvantages to using files are used as data storage mechanism
    - There exists an upper limit on the number of rows.
    - Unchecked constraint violations
    - File handling via programming takes more time
    - CRUD operations are very complex when handled through programming
    - NOTE: All problems are solved when databases are used to store data.
- Following are some disadvantages with databases.
    - The effort to install and configure a database is expensive & time consuming.
    - It is not easy to do any set of arithmetic operations using DBMS
    - Databases are inherently complex and need specialized people like database administrators to configure and maintain it
- DBMS are versatile for the following reasons
    - In-built mechanisms to ensure the consistency and sanity of data during operations.
    - No redundant data
    - Atomicity of updates
    - Concurrent access by multiple users
    - Effective data security
    - High efficiency of operation
    - Easy Data Recovery
    - Easy to handle large datasets
- Hi-level architecture of database systems
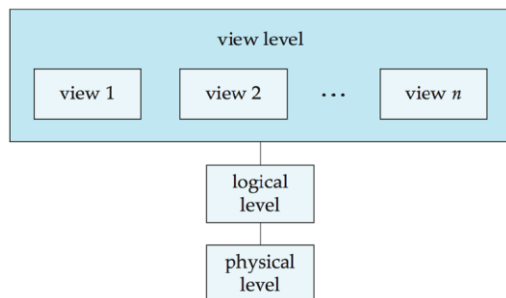


- Physical level describes how the records are actually stored in a database.
- Logical level describes the relationship between the data in a database.
- At the view level, application programs may hide certain information for security purposes.
- A change in Physical Level of DBMS should not affect either the View Level or the Logical Level. This is due to the physical level independence.
- A change in Logical Level of DBMS should not affect the View Level. This is due to the logical level independence.

- A schema describes how the data is organized in a database.  It is similar to the type information of a variable in programming languages.
- DBMS schema, along with its tables, constraints and indexes, can be defined and manipulated using the Data Definition Language (DDL).
- In order to create new records and manipulate existing records in a database, Data Manipulation Language (DML) is used.
- Types of database managements systems include relational model, network model and hierarchical model.
- Data dictionary contains the Database schema, Integrity constrains and Authorizations
- Databases use two types of languages – Pure and Commercial.  Pure languages are used for purposes like query optimization.  Examples of such languages are Relational algebra and Tuple relational calculus.  Commercial languages are used in commercial/industry systems.  Example is SQL.
- ER Model is used to define a high level view of the data entities and the relationships between them. It is used mainly for designing and planning the database structure.
- Object Relational Data Model provides upward compatibility with existing relational languages.  It also allows attributes of tuples to have complex data types.
- Concurrency control manager controls the interaction among concurrent transactions in order to ensure the consistency of the database.
- XML can specify new tags and create nested tag structures, which makes it very suitable to exchange data in general, and not just documents.
- The following diagram represents query processing in a database system.



-
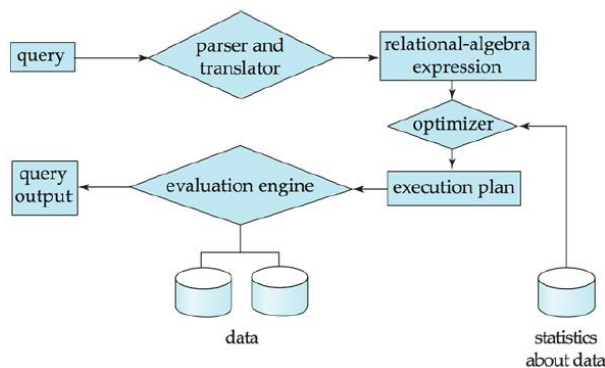
# Week2

- Set of allowed values for a given attribute is called the domain of the attribute.
- *null* is used to represent unknown values, and is a special value of every domain.
- In a table, order of tuples is irrelevant and no two tuples can be identical.
- Super-key(s) in a relation is any subset of the entire set of attributes of the relation, as far as its values identifies a unique tuple in the relation.
- A super-key is a candidate key if it is minimal.
- All candidate keys are super keys, but the converse is not true.  One of the candidate keys is selected as the primary key.

- One major difference between primary key and candidate key is that while the former doesn't allow nulls, the latter allows them.
- All candidate keys that have not been selected as the primary key are called secondary keys
- A surrogate key (or synthetic key) in a database is a unique identifier for either an entity in the modeled world or an object in the database. It is not derived from application data, unlike a natural (or business) key which is derived from application data
- Foreign key is one which is referred to from another table, where it's a primary key. A table can have multiple foreign keys. They can be added at the time of table definition, or subsequently using ALTER TABLE query.
- Foreign key can be one of the super-keys in the table, including its primary key.
- In procedural programming style, the programmer specifies how to achieve the output. In declarative programming style, the programmer specifies what is required, in which case, the programmer must know what relationships hold between various entities.
- Relational algebra supports the following operations
  - Select ($\sigma$) – chooses attributes based on specified condition.
  - Project ($\pi$) – selects attributes in the output relation.
  - Union (U) – "concatenates" two relations, provided both have same number of attributes.
  - Difference (-) – output relation has all tuples in the first relation, but not in the second.
  - Intersection ($\cap$) – output relation has all tuples that are common between first and second relations.
  - Cartesian Product (x) – cross-product of both relations.
  - Natural Join ($\bowtie$) – cartesian product with a filter on the common attribute between both relations.
- Intersection between two relations r and s can also be represented as r – (r – s)
- *Difference* is implemented using the *except* keyword in SQL.
- While taking a cartesian product, attributes in one relation can be renamed using operator *ρ (rho)*
- Most of the SQL implementations adhere to the SQL-92 standard.
- Following are some of the SQL implementations available commercially. There are minor variations in the features of these implementations.
  - SchemeQL and CLSQL (Lisp-based)
  - LINQ (.NET based)
  - ScalaQL and ScalaQuery (based on Scala)
  - SqlStatement, ActiveRecord (Ruby-based)
  - HaskellDB
- SPARQL (SPARQL Protocol and RDF Query Language) is the W3C standard used in semantic web, and many NoSQL systems. It's derived from the SQL protocol.
- The syntax of create table is as follows:

  An SQL relation is defined using the **create table** command:

  **create table** $r$ $(A_1 D_1, A_2 D_2, \ldots, A_n D_n)$,
  $(integrity\text{-}constraint_1)$,
  . . .
  $(integrity\text{-}constraint_k)$);

  - $r$ is the name of the relation
  - each $A_i$ is an attribute name in the schema of relation $r$
  - $D_i$ is the data type of values in the domain of attribute $A_i$

  NOTE: The integrity constraints can be *not null, unique, primary key, foreign keys* and *check*.
- Primary key declaration on an attribute automatically ensures not null.

- Unlike set operations, SELECT query allows duplicates in the resultant relation. If duplicates should be eliminated, use DISTINCT keyword.
- By default, all the set operations supported by SQL namely *union, intersect* and *except* eliminates duplicates from the resultant relation. If the duplicates need to be retained, use keyword *all* after each of these keywords – *union all, intersect all, except all.*
- Suppose a tuple occurs m times in r and n times in s, then, it occurs:
  - m + n times in r union all s
  - min(m, n) times in r intersect all s
  - max(0, m - n) times in r except all s
- It is not possible to test for *null* values with comparison operators, such as =, <, or <>. We need to use the *is null* and *is not null* operators instead.
- Truth-table for operations using OR
  - null or true = true,
  - null or false = null
  - null or null = null
- Truth-table for operations using AND
  - true and null = null,
  - false and null = false,
  - null and null = null
  - not null = null
- *P is null* evaluates to true if predicate P evaluates to null
- *where* clause predicate is treated as *False*, if it evaluates to *null*
- Attributes in *select* clause outside of aggregate functions must appear in group by list. This is true for all SQL standards-compliant databases, including PostgreSQL. However, MySQL/SQLite returns the first tuple, when *group by* clause is used.
- *avg, min, max, sum, count* are aggregate functions, all of which may be used with a having clause filter on a *group-by* output.
- All aggregate operations except *count(\*)* ignore tuples with *null* values on the aggregated attributes. If all tuples are *null,* then *count* returns 0.

## Week3

- Definition of *some* clause

$$F <comp> \textbf{some } r \Leftrightarrow \exists t \in r \text{ such that } (F <comp> t )$$
$$\text{where } <comp> \text{ can be: } <, \leq, >, \geq, =, \neq$$

- For example,

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept name = 'Biology';
```

is equivalent to

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept_name = 'Biology');
```

$(= \textbf{some}) \equiv \textbf{in}$
However, $(\neq \textbf{some}) \not\equiv \textbf{not in}$

- Definition of *all* clause

  $F <comp> \textbf{all} \ r \Leftrightarrow \forall t \in r$ such that $(F <comp> t \ )$
  Where $<comp>$ can be: $<, \leq, >, \geq, =, \neq$
  $(\neq \textbf{all}) \equiv \textbf{not in}$
  IMPORTANT: However, $(= \textbf{all}) \not\equiv \textbf{in}$

- *Exists* clause

  The **exists** construct returns the value **true** if the argument subquery is nonempty
  - **exists** $r \Leftrightarrow r \neq \emptyset$
  - **not exists** $r \Leftrightarrow r = \emptyset$

- The *unique* construct evaluates to true if a given subquery contains no duplicates
- The *with* clause provides a way of defining a temporary relation whose definition is available only to the query in which the *with* clause occurs. For example, the following query finds all departments with the maximum budget.

  ```
  with max_budget(value) as
        (select max(budget)
        from department)
  select department.name
  from department, max_budget
  where department.budget=max_budget.value;
  ```

- A join operation is a Cartesian product which requires that tuples in the two (or more) relations match, under some condition. The result of a join operation is another relation.
- A view is a stored SQL statement that returns a specific subset of rows and/or columns from another table. It does not occupy any physical space except that occupied by the definition of the view. It provides a mechanism to hide certain data from certain users. Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a view.
- It's possible to create a physical table containing all the tuples in the result of the query defining the view. However, if relations used in the query are updated, the materialized view result becomes out of date.
- Check constraint can be used to check if the attribute value in a tuple adheres to a set rule. For example, *check (semester in ('Fall', 'Winter', 'Spring', 'Summer'))*
- With *cascading*, you can define the actions that the Database Engine takes when a user tries to delete or update a key to which existing foreign keys point. For example, the following definition for the course table, deletes all courses if the corresponding department is deleted from the database. Alternative actions to cascade are "no action", "set null" and "set default"

  ```
  create table course (
        . . .
        dept_name varchar(20),
        foreign key (dept_name) references department
              on delete cascade
              on update cascade,
        . . .
  )
  ```

- *create type* construct in SQL creates user-defined type (alias, like typedef in C). For example, define a datatype called *dollars* using the statement

  create type dollars as numeric (12,2) final

  so that, it can be used while creating the *department* table like so,

  create table department (dept name varchar (20), building varchar (15), budget dollars);

- *create domain* construct is similar to *create type.* The former allows to define additional constraints on it. For example,

```
create domain degree_level varchar(10)
constraint degree_level_test
check (value in ('Bachelors', 'Masters', 'Doctorate'));
```

- Privileges can be granted to select users/roles for the following operations - Read (select), Insert, Update, Delete, Index, Resources, Alteration, Drop. It's also possible to transfer them to select users/roles.
- grant < privilege list > on < relation name or view name > to < user list >
- revoke < privilege list> on < relation name or view name > from < user list>
- Roles are a convenient way to authorize a group of users.
- In a relational DBMS, blob data type is used for storing uninterpreted data, like videos.
- Functions return a scalar/relation, whereas procedures assign value/relation to an existing (out) variable.
- Row-level trigger fires once for every row are affected by a triggering event.
- Statement-level trigger fires at least once even if no rows are affected by a triggering event.
- Transition table stores the initial and final values of attributes for the affected rows
- Values of attributes *before* and *after* an operation can be referenced as
    - referencing old row as <variable> (for deletes and updates)
    - referencing new row as <variable> (for inserts and updates)
- Following is an example of a trigger created to maintain credits earned value

```
create trigger credits_earned after update of grade on (takes)
referencing new row as nrow
referencing old row as orow
for each row
when nrow.grade <>'F' and nrow.grade is not null
      and (orow.grade = 'F' or orow.grade is null)
begin atomic
      update student
      set tot_cred= tot_cred +
        (select  credits
        from course
        where  course.course_id=nrow.course_id)
      where student.id = nrow.id;
end;
```

- In order to design a trigger mechanism, we must specify the conditions under which the trigger is to be executed, and actions to be taken under such conditions.

# Week4

- Relational algebra uses the following symbols : σ (select), π(project), ρ (rename), U(union), ∩ (intersect), - (Except), X (cross-join) and ⋈ (natural join)
- Relational algebra is a procedural language, while tuple relational calculus is a non-procedural language. Expression written using one language can be translated to another.
- In relational algebra, duplicate tuples are dropped in the resultant relation.
- $r \cup s = \{t | t \in r \text{ or } t \in s\}$
- $r - s = \{t | t \in r \text{ and } t \notin s\}$
- $r \cap s = \{t | t \in r \text{ and } t \in s\}$
- $r \times s = \{t \, q | t \in r \text{ and } q \in s\}$

- $\rho_{X(A_1, A_2, \cdots, A_n)}(E)$ returns the result of expression E under the name X, and with the attributes renamed to $A_1, A_2, \ldots, A_n$

- When you perform $R(Z) \div S(X)$, for a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with every tuple in S

- $r \cap s = r - (r - s)$

- Given two relations R and P, R U P = $\{m \mid m \in R \vee m \in P\}$

- Given two relations R and P, R $\cap$ P = $R - (R - P)$ and $\{m \mid m \in R \wedge m \in P\}$

- Given two relations R(A ,B, C) and P(B, X, Y), the expression (A, B, C, X, Y) can be obtained by any of the following expressions.
  - $R \bowtie P$
  - $\{< a, b, c, x, y >|< a, b, c >\in R\wedge < b, x, y >\in P\}$
  - $\{t \mid \exists m \in R\, \exists n \in P(m.B = n.B \wedge t.A = m.A \wedge t.B = m.B \wedge t.C = m.C \wedge$
    $t.X = n.X \wedge t.Y = n.Y)\}$

  Note that the two relations R and P have a common attribute B.

- X $\div$ Y, where $X = (A, B)$ and $Y = (B)$ is equivalent to the following expressions.
  - $\{t \mid \exists r \in X\, \forall q \in Y (r[B] = q[B] \implies t[A] = r[A])\}$
  - $\{< a >|< a >\in X \wedge \forall < b > (< b >\in Y \implies < a, b >\in X)\}$

- An example of a division operation



- Given two relations A(P, Q, R) and B(X, Y, Z), $\Pi_{P,Z}(\sigma_{R=X}(a \times b))$ is equivalent to
  $\{t \mid \exists p \in a, \exists q \in b(t[P] = p[P] \wedge t[Z] = q[Z] \wedge p[R] = q[X])\}$

- Some examples,

Consider the relational schema
**student**(*rollNo*, *name*, *year*, *courseId*)
**course**(*courseId*, *cname*, *teacher*)

Q.2 Find out the names of all students who have taken the course name 'DBMS'.

- $\{t \mid \exists s \in student \; \exists c \in course(s.courseId = c.courseId \land c.cname = \text{'DBMS'} \land t.name = s.name)\}$

- $\{s.name \mid s \in student \land \exists c \in course(s.courseId = c.courseId \land c.cname = \text{'DBMS'})\}$

Q.3 Find out the names of all students and their rollNo who have taken the course name 'DBMS'.

- $\{s.name, s.rollNo \mid s \in student \land \exists c \in course(s.courseId = c.courseId \land c.cname = \text{'DBMS'})\}$

- $\{t \mid \exists s \in student \; \exists c \in course(s.courseId = c.courseId \land c.cname = \text{'DBMS'} \land t.name = s.name \land t.rollNo = s.rollNo)\}$

- A collection of entities is called an entity set. Attribute is a set of descriptive properties possessed by all members of an entity set.
- The relationship associating the weak entity set with the identifying entity set is called an identifying relationship.
- Every weak entity must be associated with an identifying entity.
- An entity set that is not a weak entity set is termed as a strong entity set.
- In an ER diagram, rectangles are used to represent entity sets. Attributes are listed inside entity rectangle. Underline indicates primary key attributes. Diamonds represent relationships. A directed line indicates "one" and undirected line represents "many" in a relationship.
- Thus, in the following ER diagram, two entity sets instructor and student are related through an advisor by a many-to-one relationship. Thus, an instructor can advise many students, but a student will have a single instructor who advises him/her. ID is the primary key in the instructor table; ID is the primary key in the student table also.
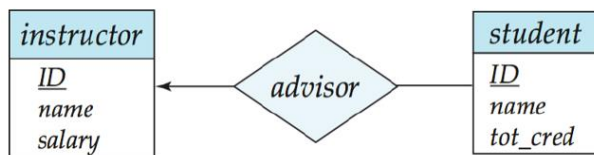


- In the case of a one-many relationship like in the above case, it's best to keep advisor as part of the student table - *advisor_id* in student table with a foreign-key relationship to *instructor_id* in instructor table.
- If there is a 1-1 relationship between two entities (with no additional attributes for the relationship), it's best to choose to have a single table with a composite key that carries the primary key from the first entity and the primary key from the second entity. Consider the following example.

10. Consider the E-R diagram in Figure 5. [ NAT: 2 points]



Figure 5: ERD

What is the minimum number of tables needed to represent this E-R diagram?

> **Solution: 1**
> The minimum and maximum cardinality is 1 (1..1).
>
> - A minimum value of 1 indicates total participation.
>
> - A maximum value of 1 indicates that the entity participates in at most one relationship.
>
> Thus, it can be represented using a single table:
> **team_captain**($\underline{team\_code}, team\_name, \underline{player\_num}, player\_name$).

- In the case of a 1-1 relationship between two entities (with added attributes for the relationship), you could choose to have two tables, with either of them carrying the added attribute and the primary key of the other entity set. Consider the following example.

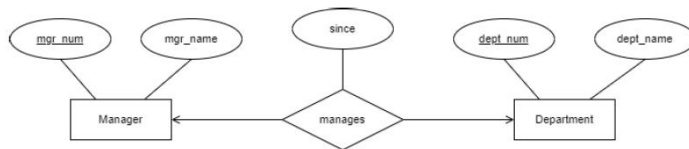5. Consider the E-R diagram given in Figure 2. [MSQ: 1 point]



Figure 2: E-R diagram

Identify the option(s) that correctly represent(s) the corresponding tables for the given E-R diagram.

- ○ **Manager**($\underline{mgr\_num}, mgr\_name$)
  **manages**($\underline{mgr\_num}, dept\_name, since$)
  **Department**($\underline{dept\_num}, dept\_name$)
- ✓ **Manager**($\underline{mgr\_num}, mgr\_name$)
  **Department**($\underline{dept\_num}, mgr\_num, dept\_name, since$)
- ✓ **Manager**($\underline{mgr\_num}, dept\_num, mgr\_name, since$)
  **Department**($\underline{dept\_num}, dept\_name$)
- ○ **Manager**($\underline{mgr\_num}, dept\_num, mgr\_name, since$)
  **Department**($\underline{dept\_num}, dept\_name$)

- Attribute types include Simple/Composite, Single-valued/multi-valued, Derived attributes.
- An entity set can be strong (has a primary key), and weak (has a partial key, called discriminator). Weak entity set cannot exist independently, but always exists in (identifying) relationship with a strong entity set. In E-R diagrams, a weak entity set is represented by double rectangle.
- Primary key of a weak entity set = Its own discriminator + Primary key of the strong entity set.
- Weak entity set has total participation in the identifying relationship, implying all its entities must feature in the relationship. In E-R diagram, total participation is represented by a double line.
- In attribute inheritance, a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set.
- Any non-binary relationship can be represented using binary relationships by creating an artificial entity set.

- In top-down design process, we designate sub-groupings within an entity set that are distinctive from other entities in the set. This defines an is-a relationship, where the lower-level entity set has some attributes that do no apply to higher-level entity set.
- There are two types of is-a relationship – overlapping and disjoint.
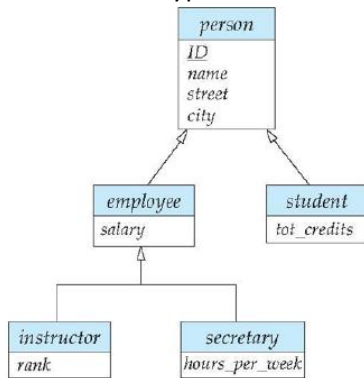


- In the above diagram, Employee and Student are overlapping (Employee can be a student and vice-versa), but Instructor and Secretary are disjoint sets.
- Bottom-up design process combines a number of entity sets that share the same features into a higher level entity set
- Depending on whether or not higher-level sets must belong to lower-level entity sets, relationships are of two types – total and partial. Consider the following example.

4. A bank consists of several **Person** entities. The **Person** entities may have two special types: **Employee** and **AccountHolder**. However, there is a possibility that some **Person** entities are neither an **Employee** nor an **AccountHolder** (like a visitor at the bank). Again, some **Person** entities can be of both **Employee** and **AccountHolder** types. [MCQ: 2 points]
Identify the constraints on specialization with respect to the above scenario.

○ Disjoint and partial

√ Overlapping and partial

○ Disjoint and total

○ Overlapping and total

Solution:

- As a **Person** can be an **Employee** or an **AccountHolder** or just a **Person** (like a visitor at the bank), it is partial specialization.
- As a **Person** can be both **Employee** and **AccountHolder**, it is overlapping specialization.

# Week5

- The schema design must promote redundant storage of the data items.
- Redundancy may result in data anomaly, and may be due to an un-normalized database.
- Redundancy can be reduced by appropriate decomposition of relations.
- Insertion, Deletion and Update poses anomalies due to improper schema design.
- Insertion anomaly is said to occur when changing a single data may require changing many records, leading to the possibility of some changes being made incorrectly.
- A deletion anomaly occurs when you delete a record that should not be deleted.

- In DBMS, we assume that all the relations are in 1NF, by default. This implies that the domains of all attributes of the relation are atomic, and the value of each attribute contains only a single value from that attribute's domain.
- A set of all functional dependencies implied by the original set of dependencies F is called a closure set of functional dependencies (F+)
- Any relation may be decomposed into smaller relations each of which is in 3NF and preserves all functional dependencies in the original relation.
- Increase in query cost of specific queries is a disadvantage of decomposition.
- For two sets of functional dependencies F1 and F2, if F1 covers F2 and F2 covers F1, then F1 and F2 are equivalent.
- Attribute closure can be used to find non-trivial functional dependencies.
- The decomposition is lossless if $R = (R_1 \bowtie R_2 \bowtie R_3 \ldots \bowtie R_n)$
- The decomposition is lossy if $R \subset (R_1 \bowtie R_2 \bowtie R_3 \ldots \bowtie R_n)$
- A lossless join decomposition doesn't ensure dependency preservation. Likewise, a decomposition that preserves dependencies doesn't ensure that its join will be lossless.
- For the decomposition to be lossless, $R1 \cap R2 \cap R3 \neq \phi$ may not always hold
- However, (R1 U R2) ∩ R3 $\neq \phi$, in the case of lossless decomposition.

## What is a candidate key?

Candidate key is a set of minimal attributes that can identify each tuple uniquely in a given relation. Alternatively, it's a set of minimal attributes from which we can determine all other attributes in a relation.

## What is a super-key?

Super set of all candidate keys is a super-key. Every candidate key is a super-key, but every super-key is not a candidate key (only the minimal super-keys are called candidate keys)

## What is a canonical cover?

Whenever a user performs an update on a relation, the DBMS must ensure that the update doesn't violate any functional dependencies. To reduce the effort spent in checking violations, we create a 'minimal' set of functional dependencies equivalent to the given set having no redundant dependencies or redundant parts of dependencies. This 'minimal' set called canonical cover/minimal cover.

## How to create a canonical cover?

1. Break multiple attributes in the RHS of functional dependencies into single attribute

A→BC ⇒ A→B, A→C        All single attributes on RHS

2. Remove extraneous attributes

3. Find out if there are redundant FDs, using transitivity, augmentation or reflexivity (Armstrong's axioms). Remove FD's from the set, as far as the closure of the whole set remains unchanged.

1) $A \to BC \Rightarrow A \to B, A \to C$     All single attributes on RHS

2) Find out if there are any extraneous attributes

3) Find out if there are any redundant FDs.

$A \to BC, A \to B, B \to C, AB \to C$

$A \to B$
$A \to C$ $\Rightarrow$ $A \to B \leftarrow$
$A \to B$     $A \to C$     $\Rightarrow A \to B, A \to C, B \to C$
$\to B \to C$     $A \to B \leftarrow$
$\to AB \to C$     $B \to C$     $\Rightarrow A \to B, B \to C, A \to C$
                $B \to C$     $\Rightarrow A \to C$
                            $\Rightarrow A \to B, B \to C$

## Canonical Cover

$A \to BC, CD \to E, B \to D, E \to A$

$\begin{cases} A \to B \\ A \to C \\ CD \to E \\ B \to D \\ E \to A \end{cases}$     $\begin{matrix} C \to E \\ D \to E \end{matrix} \}$     $A \to D$     $\begin{cases} A^+ = AB\underline{C}\underline{D}E \\ (CD)^+ = CDE\underline{A}\underline{B} = ABCDE \\ B^+ = BD \\ E^+ = \underline{E}\underline{A}\underline{B}CD = ABCDE \end{cases}$

$\boxed{A \to BC, CD \to E, B \to D, E \to A}$

## Canonical Cover

$A \to BCDE, CD \to E$

$A \to B$
$A \to C$ $\}$     $A \to CD, CD \to E \Rightarrow A \to E$
$\times \; \begin{matrix} A \to D \\ A \to E \end{matrix} \}$
$\to CD \to E$     $\boxed{A \to BCD, CD \to E}$
$\quad C \to E$
$\quad D \to E$

**Canonical Cover**

$B \to A$ , $D \to A$ , $AB \to D$

$B \to A \Rightarrow BB \to AB$, $AB \to D$

$\Rightarrow B \to D$
$AB \to D$ $\Rightarrow$ $B \to D$

$B \to A$, $D \to A$ $\quad B \to D$

$B \to D$, $D \to A \Rightarrow B \to A$

$\boxed{B \to D, D \to A}$

**NOTE**: The original set of FDs and its canonical cover are equivalent. Thus, from an exam-perspective, it would be easier to check for equivalence of the given FD set with the original FD set.

Another example: https://youtu.be/khXXOQhgFoA?list=PLEolU0bP1_Rg1e3z1QJZRiYPOkrV-ZR7J&t=7522

## How to find extraneous attributes?

Extraneous attributes are those attributes in a functional dependency that can be removed without changing the closure of the set of functional dependencies.

To find out if an attribute is extraneous, after removing it from the LHS of FD, check if the RHS of the same FD is present/derivable from the (remaining) set of FD's.

Example: F = {A -> B, AB -> C}

Let's check if LHS of AB -> C has any redundant attribute.

If A was extraneous in this FD, B -> C.  B+ = B.  Since this doesn't contain C, A is not extraneous.

If B was extraneous, A -> C.  A+ = ABC.  Since this contains C (RHS of the FD), B is extraneous.

## How to identify if a given set of attributes is a super-key?

For a given set of attributes, if its closure covers all attributes of the relation, then it's a super-key.

Consider the following set $F$ of functional dependencies on the relation schema $(A, B, C, D, E, G)$:

$$\{A \to BCD, BC \to DE, B \to D, D \to A\}$$

Prove that $AG$ is a superkey.

Idea credit: Silberschatz, A., Korth, H. F., & Sudarshan, S. Database system concepts. Edition 7. McGraw-Hill Education.

$(AG)^+ =$

$\text{Result} = AG$

$= AGBCD \quad (A \to BCD)$

$= AGBCDE \quad (BC \to DE)$

$(AG)^+ = AGBCDE$

$\underline{SK}$

## How to identify candidate-keys from a given relation and a set of FDs?

If the closure of a set of attributes is equal to the given relation, then this set is called a super-key. When the attribute set is minimal, it's a candidate key.

Typically, you'll first find candidate-keys of a relation, and generate super-keys from the list of candidate keys.

1. Find all attributes that are not present in the RHS of any of given set of FDs. These attributes are always part of every candidate key. This is because they cannot be determined by other attributes.
2. Now, find the closure of all attributes (grouped together) obtained from the above step.
3. If the closure contains all attributes in the relation, this group is a candidate key. If not, add each attribute from rest of the relation at a time, and repeat the above step.

Consider a relation $R(A, B, C, D, E)$ and a set of functional dependency on $R$:
$F = \{AB \to C, DE \to B, CD \to E\}$. Find out all the candidate keys and prime attributes?

$(AD)^+ = \{AD\}$

$(ADB)^+ = ADB$
$= ADBC$
$= ADBCE = R$

$①(ADC)^+ = ADC$
$= ADCE$
$= ADCEB = R$

$(AD)^+ = ADE$
$ADEB$
$= ADEBC = R$

$C \cdot IL$ $③$
$\to ADB$
$\to ADC$
$\to ADE$

Prime Attributes $\vdash A, B, C, D, E$

MORE VIDEOS

## How to identify prime attributes of a given relation?

1. Find the candidate keys of the given relation.
2. Every attribute from the candidate keys constitute prime attributes of the relation.

See same screenshot as above.

# How to identify a lossy decomposition?

In the case of a lossless decomposition, when the sub-relations are joined back, the original relation is obtained. In the case of a lossy joins, the natural join of the sub-relations always have some extraneous tuples.

**For a decomposition to be lossless, it must satisfy the following conditions :**

- $R_1 \cup R_2 = R$
- $R_1 \cap R_2 \neq \phi$ and
- $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$

Q.1 Let us consider the relation $R(A,B,C,D,E)$ with the following functional dependencies set :
$\mathcal{F} = \{AB \rightarrow C, C \rightarrow D, B \rightarrow E\}$.
Let R be decomposed into $R_1$ and $R_2$. Check whether the following decomposition is lossless or lossy join decomposition.

1. $R_1(A, B, C)$ and $R_2(D, E)$
2. $R_1(A, B, C, D)$ and $R_2(B, E)$

① $R_1 \cup R_2 = ABCDE = R$

$R_1 \cap R_2 = \phi$ ⊗

→ Lossy

② $R_1 \cup R_2 = ABCIE = R$ ✓

$R_1 \cap R_2 = B \neq \phi$ ✓

$R_1 \cap R_2 = B$

Let $B^t = BE = R_2$

$R_1 \cap R_2 \rightarrow R_2$ ✓

→ Lossless

MORE VIDEOS

Q.2 Consider the relation $R(A,B,C,D,E,G)$ and the functional dependencies set
$\mathcal{F} = \{AB \rightarrow C, AC \rightarrow B, AD \rightarrow E, B \rightarrow D, BC \rightarrow A, E \rightarrow G\}$.
Let R be decomposed into $R_1(A, B, C), R_2(A, C, D, E)$ and $R_3(A, D, G)$. Check whether the decomposition is lossless or lossy join decomposition.

① $R_1 \cup R_2 \cup R_3 = ABCDEG = R$ ☺

② $R_1 \cap R_2 = AC \neq \phi$ ☺

③ $R_1 \cap R_2 \rightarrow R_1$ or $R_2$ ✓

$AC^t = ACB = R_1$

let $R_{12} = R_1 \cup R_2 = ABCDE$

$R_{12} \cap R_3 = AD \neq \phi$ ✓

$R_{12} \cap R_3 \rightarrow R_{12}$ or $R_3$

$AD^t = ADE$

$ADE+ = ADEG$

$R_{12} \cap R_3 \rightarrow R_3$ ☺

→ lossless

- To check if the decomposition of R into (say) R1, R2, R3 is dependency preserving,
  - Let's say R has FD set F.
  - Find out the FDs from R are applicable to R1, R2 and R3 by visual inspection.
  - Call this F1, F2, F3
  - Find the closure of F1, F2 and F3 with respect to F. For this, consider each attribute individually, then in pairs, triples and so on
  - Only the attributes available in the respective relations must be considered in the closure.
    NOTE: Only non-trivial FDs should be accepted in the (F1)+

○ For example, If R1 is (A, B), consider A+, B+ and AB+. If F is {A -> B, B -> C, C -> A}, A+ is {ABC} and B+ is {BAC}. Hence F1+ = {A -> B, B ->A}

- R (A, B, C, D)
  F = {A → B, B → C, C → D, D → A}
- Decomposition: R1(A, B)   R2(B, C)   R3(C, D)

$R_1(A,B)$
$A^+ = \{ABCD\}$    $A \rightarrow B$
$B^+ = \{B, CDA\}$   $B \rightarrow A$

Reference: https://youtu.be/h-SO2pc9_0k?t=676

○ Let F` = {(F1)+, F2)+, F3)+}
○ Find the closure of each element in F based on F`. If all of F can be reproduced from F`, then dependency is preserved.
https://youtu.be/xk8e2vvLcGs?list=PLEolU0bP1_Rg1e3z1QJZRiYPOkrV-ZR7J&t=3759

NOTE: Alternatively, if options are given and asked to choose from among them, select those FD sets that are equivalent to the original FD set.

- If we've N attributes in total, each of which is a candidate key, the number of super-keys is given as 2^N-1.
- If we've N attributes in total, one candidate key formed using n attributes, the number of super-keys is given as 2^(N-n). If the candidate key is formed using 1 attribute, the number of super keys is 2^(N-1).
- If there are N attributes in total with two candidate keys, the first one formed using n1 attributes and the second one formed using n2 attributes, the number of super-keys is given as 2^(N-n1) + 2^(N-n2) – 2^(N-(n1+n2)). Note here that n1 and n2 might have common attributes, in which case they must be counted only once

https://discourse.onlinedegree.iitm.ac.in/t/formula-to-find-the-maximum-number-of-super-keys-from-the-candidate-keys/32507

An example:

Let $R(J, K, L, M, N, O)$ be a given relation with the following functional dependencies:
$$\mathcal{F} = \{K \rightarrow JL, L \rightarrow K, J \rightarrow O, M \rightarrow N\}$$

Find the total number of super keys of $R$.

Here, N = 6, n1 = 2, n2 = 2, but n1 and n2 have M as a common attribute, so will be counted only once. Thus, applying above formula, #superkeys = $2^4 + 2^4 - 2^3$ = 24

# Week6
- 1NF – if all attributes have atomic values (no multi-valued attributes), then the schema is in 1NF
- 2NF – if the schema satisfies 1NF and there's no partial dependency in the schema, then it's in 2NF
- What's partial dependency?
  ○ If a *proper subset* of any of the candidate keys decide non-prime attributes, then it's called partial dependency.

- o For example, if R={A, B, C} and AB is candidate key, AB-> C is NOT partial dependency, even though AB is a subset of candidate key and C is non-prime.  This is because, AB is **NOT** a proper subset of AB.  However, A -> C is partial dependency.
- 3NF – if the schema satisfies 2NF, and none of the non-prime attributes determines another non-prime attribute (no transitive dependency), then it's in 3NF.  Now, if a non-prime determines a prime, it's still in 3NF.
- Transitive dependency occurs only in a relation that has three or more attributes.
- BCNF – In the case of 3NF, a non-prime attribute can determines a prime.  However, in BCNF, only super-keys (including the primary key, since primary key is one of the super-keys) can determine other attributes.
- BCNF guarantees lossless decomposition, but not dependency preservation.
- ETNF = Essential Tuple Normal Form
- PJNF = 5NF = Project-Join Normal Form
- The main purpose of normalization is to eliminate redundancy, and to reduce anomalies in the database.
- The main purpose for a relational database design is to achieve lossless decomposition and dependency preservation.
- Temporal databases provide a uniform and systematic way of dealing with historical data.
- Temporal data have an associated time interval during which the data are valid.
- Temporal relation is one where each tuple has associated time; either valid time or transaction time or both associated with it.
- Valid time in a temporal relation is considered as historical information.
- Transaction time in a temporal relation is considered as rollback information.

# Week7
- Presentation layer is responsible for providing the graphical user interface (GUI).  This layer deals with browsers.
- Business Logic Layer and Application Layer are responsible for supporting functionality based on frontend interface of the application.  These layers link the frontend and backend of an application.
- Data Access Layer is responsible for storing and accessing persistent data.
- Business Logic Layer hides the feature of data storage schema
- DBMS could have centralized as well as decentralized architectures.
- URIs can be classified as URLs or URNs or both.
- ODBC and Embedded SQL are ways to work with databases from within C/C++.  JDBC is used while working with Java.
- *psycopg, pg8000, ocpgdp, PyGreSQL* are various drivers used to work with PostgreSQL in Python.
- WSGI = Web Server Gateway Interface
- Google App Engine and Microsoft Azure are both RAD platforms use in web development.  JSP, Ruby on Rails, JSF are some of the RAD frameworks used in web development.
- A server saves information about cookies it issued, and can use it when serving a request.
- Cookies are created and shared between a server and a browser with the help of an HTTP header.
- The duration for which a cookie is stored may be permanent or temporary.
- Sessions are server-side files whereas, cookies are client-side files.

- `EXEC SQL' statement is used to identify embedded SQL request to the pre-processor.
- Embedded SQL combines high-level language statements, like those from Java, with SQL.
- The three components of three-layer web architecture are: web server, application server, and database server.  Single-tier architecture keeps all the elements of an application in one place

## Week8

- Big-O, Big-Omega and Big-Theta are all used to denote the time complexity of algorithms.
- O(1), O(log n), O(n), O(n log n) are the preferred time complexities in the same order.  $O(n^2)$, $O(2^n)$ are the least preferred.
- Arrays, Linked lists, Stacks and Queues are linear data structures.
- Trees, Graphs, Hash tables, Skip lists are non-linear data structures.
- Linked lists, Stacks and Queues use referential mechanism of storage, while Arrays permit random-access using the concept of indexing.
- Arrays, Stacks and Queues uses contiguous memory locations for storage, while Linked lists store them in non-contiguous locations.
- Queue = FIFO, Stack = LIFO
- Internal nodes of a tree includes all nodes other than leaf nodes and includes the root node.  All internal nodes have at least one child.
- Root node is considered to be at level-0 of the tree.
- *-arity* of a tree is the maximum number of child nodes among all tree nodes.
- The maximum number of nodes at level *h* of a binary tree is $2^h$
- Maximum total number of nodes in a binary tree, given height *h* is $2^{h+1} - 1$
- If there are *n* nodes in a binary tree, the maximum height of the binary tree is *n−1*, where all nodes are in sorted order.
- In a binary search tree (BST), each node in the left sub-tree (LST) is less than the value of its root, and each node is the right sub-tree (RST) is greater than the value of its root.
- No duplicates can exist in BST
- Time/space complexities for known data structures

## Common Data Structure Operations

| Data Structure | Time Complexity | | | | | | | | Space Complexity |
|---|---|---|---|---|---|---|---|---|---|
| | Average | | | | Worst | | | | Worst |
| | Access | Search | Insertion | Deletion | Access | Search | Insertion | Deletion | |
| Array | Θ(1) | Θ(n) | Θ(n) | Θ(n) | O(1) | O(n) | O(n) | O(n) | O(n) |
| Stack | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| Queue | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| Singly-Linked List | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| Doubly-Linked List | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| Skip List | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(n) | O(n) | O(n) | O(n) | O(n log(n)) |
| Hash Table | N/A | Θ(1) | Θ(1) | Θ(1) | N/A | O(n) | O(n) | O(n) | O(n) |
| Binary Search Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(n) | O(n) | O(n) | O(n) | O(n) |
| Cartesian Tree | N/A | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | N/A | O(n) | O(n) | O(n) | O(n) |
| B-Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| Red-Black Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| Splay Tree | N/A | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | N/A | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| AVL Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| KD Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(n) | O(n) | O(n) | O(n) | O(n) |

- Cache, Registers and RAM are volatile, while HDD, SSD, magnetic tapes and USB are non-volatile memory storages.
- NAND flash is cheaper than NOR flash.
- Disc Controller provides an interface between the computer system and the disk drive hardware.  It accepts command to read/write a sector, and initiates action by moving the disk arm to the right track/sector.
- Slotted page header contains
    - The number of record entries in the header,
    - An array containing the information on the location and size of the records
    - End of free Space in the block
- Magnetic tapes allows sequential access, while most other forms of storage allows random access.
- In the case of a magnetic disk,
    - Access time is defined as the time gap between read/write issue request and data transfer begins.  It equals Seek time + Rotational latency time
    - Average seek time = ½ * worst seek time.
    - R = ½ * T
    - Data transfer rate = S/T =  S/(2 * R)

      NOTE: T = Time taken for one complete rotation, S = Track size, R = Average rotational latency

    - MTTF for each disk in a disk system = MTTF for the entire system * #disks
    - #cylinders = #tracks
- Problems
    1. Consider a file with 1000KB.  Seek time of the hard disk is 3ms, rotational speed is 30000 rpm.  The disk has 200 sectors/track and sector size is 512 bytes.  Find the transfer rate.

Transfer rate = Amount of Data/Time taken for one rotation
Amount of data = 200 * 512 bytes = 102400 bytes
Time taken for one rotation = 60 / 30000 = 2 ms

So, transfer rate = 1024000/2 = 50 KB/ms

2. In the above problem, if the file is not in consecutive sectors, how much will it take to read the entire file.

   Transfer time for 1000KB at 50 KB/ms is 1000/50 = 20 ms.

   However, the seek time increases drastically in this case.
   Rotational latency = ½ * time taken for one rotation = 1 ms.
   Seek time = 3 ms
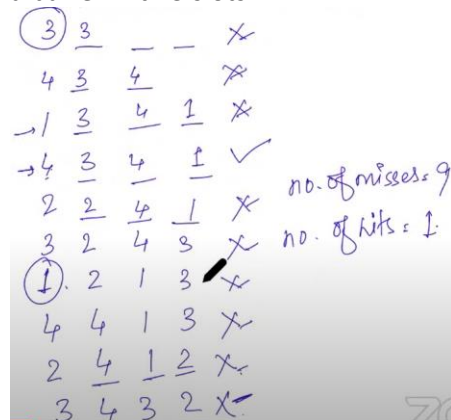
   Access time (per sector) = 3 ms + 1 ms = 4 ms

   Now, number of sectors in a 1000KB file is 1000 * 1024 bytes/512 bytes = 2000.
   Thus total access time = 2000 * 4 ms = 8000 ms = 8 s.
   Hence, the total read time for 1000 KB file = 8 s + 20 ms = 8.02 s.

3. How many misses/hits will occur while reading [3,4,1,4,2,3,1,4,2,3], when using LRU strategy on a buffer with 3 slots.

   

4. You are the quality control engineer of a magnetic disk manufacturing firm. While testing a set of 30 disks for 24 hours, you observed that 15 disks failed after 18 hours, 8 other disks failed after 22 hours and the remaining disks never failed. What is the MTTF for 15 disks randomly selected from the set ?

   Total number of hours experiment is run = 15 x 18 + 22 x 8 + 7 x 24 = 614 hrs
   Number of hard-disk failures in 614 hrs = 23.
   Hence, MTTF for one disk = 614/23 = 26.73 hrs

   MTTF for 15 disks = 26.73 / 15 = 1.782 hrs

   Reference: https://discourse.onlinedegree.iitm.ac.in/t/quiz-3-mttf-questioni/22481/2

- DNA data storage is the process of encoding and decoding binary data to and from synthesized strands of DNA.
- A DNA synthesizer machine builds synthetic DNA strands matching the sequence of digital code
- Both DNA Digital Storage and Quantum Memory can store enormous data which is not possible in file based storage system.
- In heap file organization records can be stored in any available free space and no ordering is done in this file organization. Hence a selection operation has to go through all the possible records to find the required record. In worst case the required record can be the last one.

# Week9

- Indexing mechanisms are used to speed up access to desired data.
- A search key is an attribute among a set of attributes that is used to look up records in a file.
- In ordered indices, search keys are stored in sorted order.
- Search key that orders the data records in the sequential order of data is called primary index; others are called secondary indices.
- Primary index is a type of clustered index.
- An ordered sequential file with a primary index is known as Index Sequential File; the method of access is known as ISAM.
- When all data records have indices associated with them, it's called as dense index; if only some data records have indices associated with them (typically only blocks), it's called a sparse index.
- Sparse index consumes lesser space and requires lesser maintenance overhead for insertions and deletions than dense index.
- Dense index is generally faster than sparse index for locating records.
- In hash indices, search keys are distributed uniformly across buckets using a hash function.
- When there are n elements in the BST having height h, the tree is said to be balanced when h ~ O(log n)
- BST is considered good, if it's balanced; skewed trees are considered bad.
- When child trees of BST differ in height by at most one, such BST is called AVL tree.

## 2-3-4 trees

- A 2-3-4 tree can have 2, 3 or 4 children for each node. A 2-node tree must contain a single data item and two child pointers. A 3-node tree must contain two data items and three child pointers. A 4-node tree must contain three data items and four child pointers. It generalizes easily to larger nodes.
- Height of a 2-3-4 tree is given as h ~ O(log n)
- The maximum number of data items that a leaf node of a 2-3-4 tree can contain is 3, in case of a 4-node tree.
- A 2-3-4 tree can be implemented as an external data structure.

## B, B+ trees.

- https://www.youtube.com/watch?v=aZjYr87r1b8 (Video by Abdul Bari)
- B-trees have lesser #nodes compared to B+ trees. It's sometimes possible to find the keys without reaching till the leaf level.

- In B-Trees, leaf nodes are not linked together. In B+ trees, leaf nodes are linked using a linked list.
- B+ trees typically have higher depth than B trees, and hence search could be slower.
- However, In B-trees, insertion and deletion are more complicated and time-consuming than in B+ trees.
- B+ trees are preferred for all above reasons.
- Order of a tree is the #child-pointers supported by a node. The number of keys supported by each node is one less than the #child-pointers.
- In a B+ tree with order n, minimum number of pointers supported by root node is 2 (because there should be at least one search key at the root node), and maximum number of pointers supported by root node is n.
- Problems
  1.



Number of records = 2^30
block size = 1024 bytes
record size = 32 bytes
primary key field of the record is 20 bytes and the block pointer size is 12 bytes.
We create the Sparse indexing file using the primary key.
What is the minimum number of blocks required for an index file?

Blocking factor
or
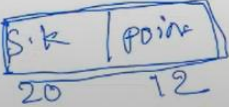Number of Records accommodate in block $= \left\lceil \dfrac{1024}{32} \right\rceil = \dfrac{2^{10}}{2^5} = 2^5$
main file

Number of block require in main file $= \left\lceil \dfrac{2^{30}}{2^5} \right\rceil = 2^{25}$

Blocking factor indexing $= \left\lceil \dfrac{2^{10}}{20+12} \right\rceil = 2^5$

no of block require indexing $= \left\lceil \dfrac{2^{25}}{2^5} \right\rceil = 2^{20}$

NOTE: Sparse index must be created one per block.

2. Consider a multilevel index with four levels as L1, L2, L3 and L4. Let L1 be the innermost and L4 be the outermost levels. Let the index blocking factor or the maximum number of entries held by a block be 50. If the number of blocks in level L1 is 1,00,000, then how many blocks are required at L2, L3 and L4?

Number of blocks in L2 is ⌈100000/50 ⌉ = 2000.
Number of blocks in L3 is ⌈2000/50 ⌉ = 40.
Number of blocks in L4 is ⌈40/50⌉ = 1.

3. Number of block accesses required to read a record in the above scenario is 5. This is due to the 4 levels that need to be accessed (to read the index) and 1 for getting to the record.

## Hash indexing.

- A hash function maps a data of arbitrary size to a value of fixed size.
- In static hashing, more than one records may be stored in a *bucket*.
- Hash collision occurs when two (or more) different keys produce same hash values, when subject to the same hash function.
- An ideal hash function maps the same number of search-key values to each bucket from the set of all possible values, irrespective of the actual distribution of search-key values in the file.
- While constructing a bit-map index, as many bits are used in the index as there are records in the table.  For each attribute value, there's a bit-map index.  Thus, if there are n records, each having m attributes, size of the bit-map index is *m x n* bits

## Week10

- ACID = Atomicity, Consistency, Isolation, Durability.  Note that, consistency follows, if the transaction is atomic and isolated.
- A transaction that successfully completes its execution will have a commit instruction as the last statement.
- A transaction that fails to successfully complete its execution will have an abort instruction as the last statement.
- A transaction, when starting to execute, must start with a consistent database.
- When a transaction completes successfully, the database must be consistent.
- Rules to identify conflicts between two or more transactions:
  - Works on same data, across two or more transactions, and where at least one operation is write.
  - If both transactions work on different data, it doesn't result in conflict.  In this case, the two operations can be swapped.
  - If both operations read, it doesn't result in conflict.  In this case, the two operations can be swapped.
- How to check if a schedule is conflict serializable?
  - Construct precedence graph for the schedule.
  - Only if it's acyclic, the schedule is conflict serializable.
  - [How to find topological ordering](#) (Use this to find the number of conflict-equivalent schedules)
  - [Alternative to drawing precedence graph](#)
- Every view serializable schedule that isn't conflict serializable have blind writes.  This implies, some of the transactions write data without reading them.
- For a schedule to be view-serializable,
  - Initial read of all data items must be done by same transactions in both schedules
  - Final write of all data items must be done by same transactions in both schedules
  - If in schedule S1, the transaction T1 is reading a data item updated by T2 and then, in schedule S2. T1 should read the value after the write operation of T2 on same data item
- All conflict serializable schedules are also view serializable.
- How to check if a schedule is view serializable?
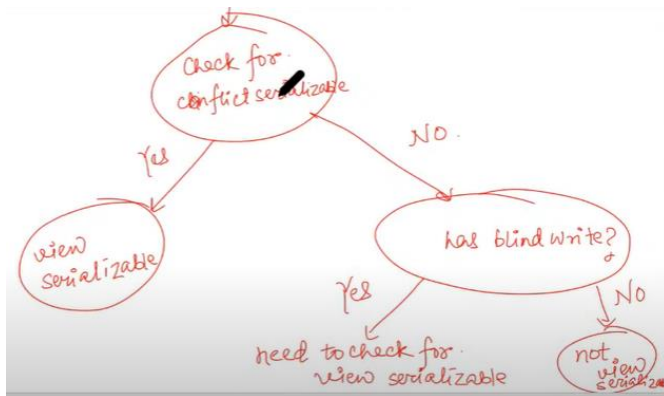
- Problem: Find view serializable schedules?



For the given schedule **S**, answer questions 4 and 5

$S : r5(Z), w1(Y), r2(Y), w3(Y), r4(Y), w2(P), r5(P), w4(X), r1(Q), r5(X), w5(Y)$

4) The schedule **S** is serializable to which of the following serial schedules?

- $T3 \rightarrow T4 \rightarrow T1 \rightarrow T2 \rightarrow T5$
- $T1 \rightarrow T5 \rightarrow T3 \rightarrow T2 \rightarrow T4$
- $T5 \rightarrow T4 \rightarrow T3 \rightarrow T2 \rightarrow T1$
- $T3 \rightarrow T2 \rightarrow T5 \rightarrow T1 \rightarrow T4$

- Database must provide a mechanism that's both conflict serializable, and recoverable (preferably cascade-less)
- Recoverability is not possible if a transaction T2 reads the updated item of another transaction T1 and performs the commit operation before T1.
- Cascade-less Schedules are preferred over Cascading Rollbacks.
- For achieving a cascade-less recovery, a transaction T1 must be committed before any other transaction reads the data item written by T1.
- Under shared lock, data items can only be read. Under exclusive lock, data items can be read and written to.
- A lock manager:
    - Maintains a data-structure called a lock-table to record granted locks and pending requests.
    - Receives lock/unlock requests from all transactions
    - Replies to a lock request by sending messages asking the transaction to roll back, in case of a deadlock.
- When a deadlock occurs, there is a possibility of cascading roll-back.
- Dead-lock Prevention:
    - Non-preemptive scheme: Oldies get preference!
        - Suppose that transaction T5, T10, T15 have time-stamps 5, 10 and 15 respectively
        - If T5 requests a data item held by T10 then T5 will "wait"
        - If T15 requests a data item held by T10, then T15 will be killed ("die")
    - Preemptive scheme: Oldies win! Youngsters wait.

- - - Suppose that transaction T5, T10, T15 have time-stamps 5, 10 and 15 respectively
    - If T5 requests a data item held by T10, then it will be preempted from T10 and T10 will be suspended ("wounded")
    - If T15 requests a data item held by T10, then T15 will "wait"
- Dead-lock Detection: Using the wait-for graph
  - When Ti requests a data item currently being held by Tj , then the edge Ti -> Tj is inserted in the wait-for graph. This edge is removed only when Tj is no longer holding a data item needed by Ti
  - The system is in a deadlock state if and only if the wait-for graph has a cycle
- To release the deadlock, kill the transaction that's least costly, or partially roll-back at least until that point where the X-lock will be released.
- Schedules that follow two-phase (Growing phase + Shrinking phase) lock protocol is always conflict serializable.
- In *strict* two-phase lock protocol, exclusive lock should be released only after commit operation is performed on the transaction.
- In *rigorous* two-phase lock protocol, both exclusive/shared locks should be released only commit operation is performed on the transaction.
- Timestamp protocol:
  - To start with, the last read time (RTS) and last write time (WTS) are set to 0.
  - For the following operations, set the last read time (RTS) to current transaction time.
    - Read operations after last write time
  - For the following operations, set the last write time (WTS) to current transaction time.
    - Write operations after last read time
    - Write operations after last write time
  - Under all other cases, rollback the operation.
  **NOTE1**: RTS/WTS cannot be assigned with a value less than its current value.
  **NOTE2**: Transaction times are allotted by DBMS, and are represented in horizontal axis in the below figure.  All operations use transaction times.
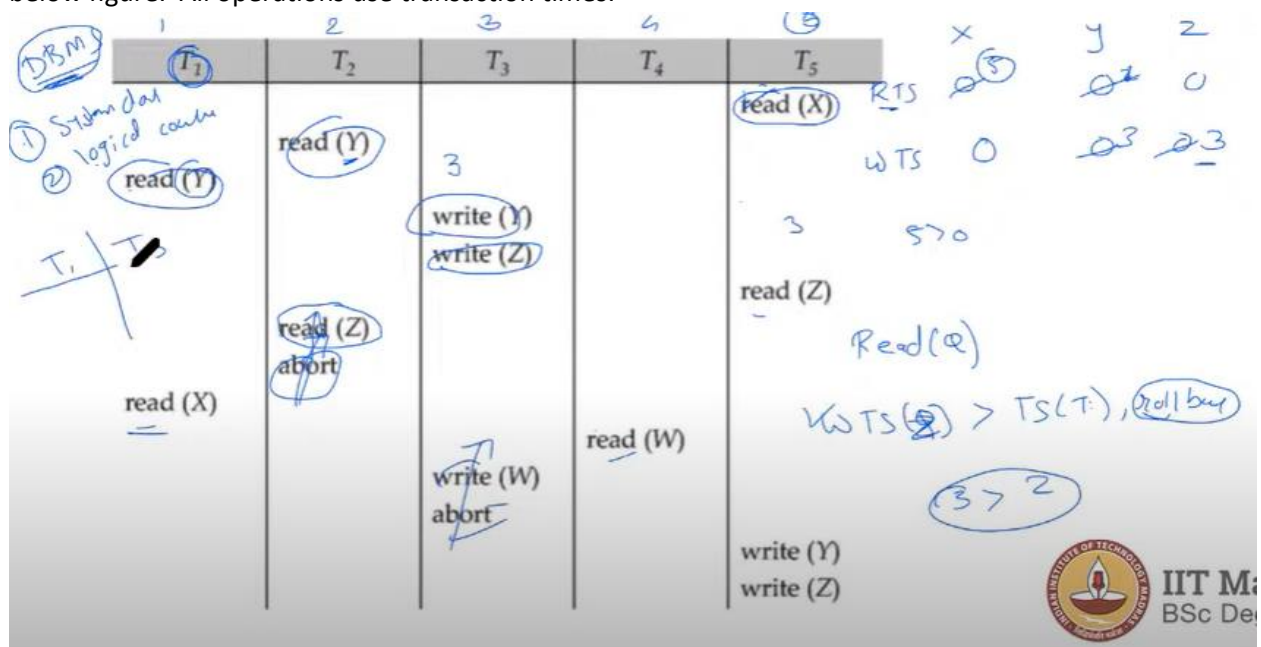
6) Tom is working as a System Designer in a reputed firm. He has 3 transactions to be analyzed as follows.
**T1:** w1(A), w1(C).
**T2:** r2(D), w2(E).
**T3:** w3(B).
His analysis involves checking each possible concurrent schedule that can be made using the transactions.
How many schedules does Tom have to check?

- 
  Answer: $(2 + 2 + 1)!/2!2!1! = 30$

# Week11

- Two types of backup:
  - Physical: A copy of physical database files such as data, control files, log files, and archived redo logs.
  - Logical: A copy of logical data that is extracted from a database consisting of tables, procedures, views, functions, etc.
- Database recovery typically happens with the help of database logs, that contain information about transactions to execute, transaction states, and modified values.
- Full backup.
  - Generally, there will not be any dependency between two consecutive backups.
  - It is relatively easy to setup, configure and maintain
  - Takes largest amount of time, and hence results in longest downtime of the system
  - Uses largest amount of storage media
- Incremental backup
  - Backups are smaller and need less media.
  - Needs shorter duration to complete the backup process. Hence lesser downtime of the system.
  - More effort/time required during recovery process.
  - Needs last full-backup, and all incremental backups since last full-backup.
- Differential backup
  - Backups are smaller and need less media, than full-backups.
  - No need to have all incremental backups since last full-backup.
- Following diagram represents all backup schemes.



| Full | Incremental | Differential | Incremental | Full |
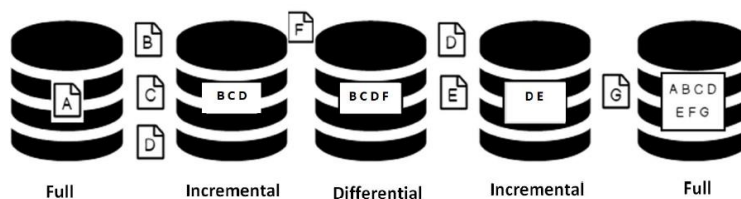
- Hot backup (Backup of transaction logs)
  - Database always up and running.
  - Efficient when dealing with dynamic and modularized data.
  - Less fault tolerant.  Error while backup can derail the entire process.
  - High maintenance and setup cost.

- Schematic representing data read/write operations on a database/system level.
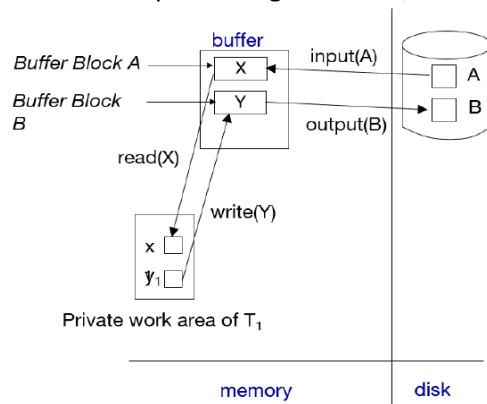


- When recovering after failure,
  - transaction will be undone, if the log contains <T start>, but not <T commit> or <T abort>
  - transaction will be redone, if the log contains <T start>, and contain either <T commit> or <T abort>
- Recovery actions in each of the following cases are:

| (a) | (b) | (c) |
|---|---|---|
| $<T_0$ start> | $<T_0$ start> | $<T_0$ start> |
| $<T_0, A, 1000, 950>$ | $<T_0, A, 1000, 950>$ | $<T_0, A, 1000, 950>$ |
| $<T_0, B, 2000, 2050>$ | $<T_0, B, 2000, 2050>$ | $<T_0, B, 2000, 2050>$ |
| | $<T_0$ commit> | $<T_0$ commit> |
| | $<T_1$ start> | $<T_1$ start> |
| | $<T_1, C, 700, 600>$ | $<T_1, C, 700, 600>$ |
| | | $<T_1$ commit> |

(a)
  - undo (T0):
    - B is restored to 2000 and A to 1000
    - Log records < T0, B, 2000 >, < T0, A, 1000 > and  T0, abort> are written out.  These are called compensation log records.

(b)
  - redo (T0):
    - A and B are set to 950 and 2050
  - undo (T1)
    - C is restored to 700.
    - (Compensation) Log records < T1, C, 700 >, < T1, abort> are written out.

(b)
  - redo (T0):
    - A and B are set to 950 and 2050
  - redo (T0):
    - C is set to 600

    If, in any of the above cases, log contains <T abort>, then the corresponding transaction is redone.
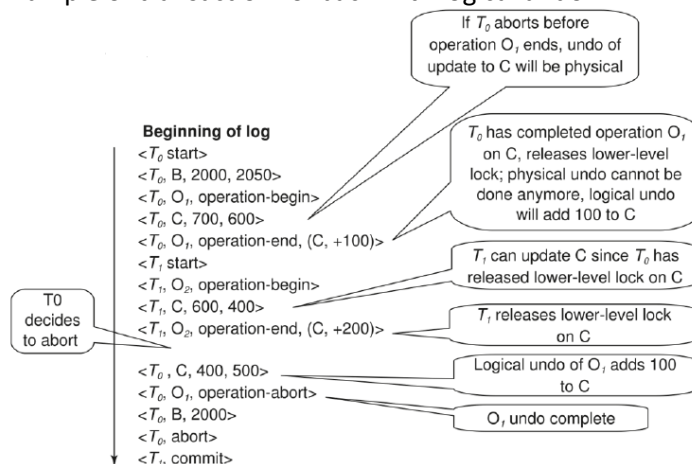- In order to avoiding redoing older transactions that are already written to the database, use checkpoints.

- Before checkpointing, output all log records and modified buffer blocks currently in main memory to stable storage. Also, write log record <checkpoint L> is written out, where L is the list of active transactions.
- During recovery, consider transactions that started before the checkpoint, and all transactions that started after the checkpoint. Here's a diagram that shows how checkpoints help to speed up the recovery process.



checkpoint                    system failure
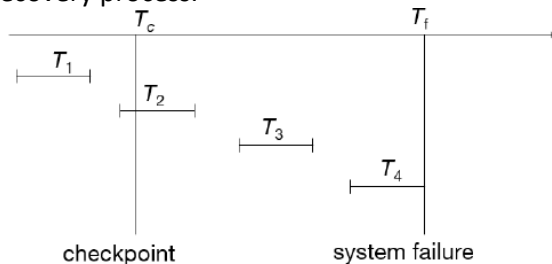
  In the above case, T1 is ignored because it was committed before checkpoint Tc. T2 (started before Tc) and T3 are redone because both are committed at the time of failure T1. Since T4 is not committed yet, it's undone.
- To support high-concurrency locking techniques in transactions, such as those used for B+-tree concurrency control (which release locks) early supports logical undo operations. This requires that the log gets created at the operation level (instead of transaction level).
- Operation log statements are embedded within *operation-begin* and *operation-end,* with the *operation-end* containing the details how to undo the operation
- Example of transaction rollback with logical undo



- Example of failure recovery with logical undo

**Beginning of log**

<$T_0$ start>
<$T_0$, B, 2000, 2050>
<$T_0$ commit>
<$T_1$ start>
<$T_1$, B, 2050, 2100>
<$T_1$, $O_4$, operation-begin>
<checkpoint {$T_1$}>
<$T_1$, C, 700, 400>
<$T_1$, $O_4$, operation-end, (C, +300)>
<$T_2$ start>
<$T_2$, $O_5$, operation-begin>
<$T_2$, C, 400, 300>

**End of log at crash!**

<$T_2$, C, 400>
<$T_2$ abort>
<$T_1$, C, 400, 700>
<$T_1$, $O_4$, operation-abort>
<$T_1$, B, 2050>
<$T_1$ abort>

Records added during recovery

Start log records found for all transactions in undo list

**Redo Pass**

Undo list: $T_1$, $T_2$   **Undo Pass**

Update of C was part of $O_5$, undone physically during recovery since $O_5$ did not complete

Logical undo of $O_4$ adds 300 to C

- Mirroring, striping and parity are methods used extensively in RAID-based systems.
- Generally, in RAID systems, if a disk experiences a failure, recovery can be made by simply XORing all the remaining data bits and the parity bit. However, in RAID-0 to RAID-5, if more than one disk fails, it cannot be recovered.
- RAID-0 uses only striping. Least costly, since no redundant information is stored. Best write performance. Poor reliability.
- RAID-1 uses mirroring. Costliest, high reliability. Space utilization is 50%.
- RAID-2 uses multiple designated parity disks and uses bit-level striping.
- RAID-3 uses a single parity disks and uses byte-level striping
- RAID-4 is similar to RAID-3, but uses block-level striping.
- RAID-5 is similar to RAID-4, but uses uniformly distributed parity across all disks.
- RAID-6 has one additional parity compared to RAID-5. Hence, it can recover from two disk failures.
- Problem:
  - RAID system codifies 0100 XXXX 0100 0001 using 0101 as the parity. Find XXXX, and the original data.
    - Divide the data into 4 parts, each carrying 4 bits.
    - XOR all bits on the 1000th place. You'll get 0-0-0-0 = 0
    - XOR all bits on the 100th place. You'll get 1-1-0-1 = 1
    - XOR all bits on the 10th place. You'll get 0-0-0-0 = 0
    - XOR all bits on the 1th place. You'll get 0-0-1-1 = 0
    - XXXX is 0100
    - Reading the entire string together, the data is 0100 0100 0100 0001 = DA

# Week11

- Cost for each algorithm :

| A# | Algorithm | Cost | Reason |
|----|-----------|------|--------|
| A1 | Linear Search | $t_S + b_r \times t_T$ | One initial seek plus $b_r$ block transfers |
| A1 | Linear Search, Eq. on Key | Average case $t_S + (b_r/2) \times t_T$ | Since at most one record satisfies condition, scan can be terminated as soon as the required record is found. $b_r$ blocks transfers in worst case |
| A2 | Prm. Index, Eq. on Key | $(h_i+1) \times (t_T+t_S)$ | Index lookup traverses the height of the tree plus one I/O to fetch the record; each of these I/O operations requires a seek and a block transfer |
| A3 | Prm. Index, Eq. on Nonkey | $h_i \times (t_T + t_S)+ b \times t_T$ | One seek for each level of the tree, one seek for the first block. Here all of $b$ are read. These blocks are leaf blocks assumed to be stored sequentially (for a primary index) and don't require additional seeks |
| A4 | Snd. Index, Eq. on Key | $(h_i+1) \times (t_T+t_S)$ | This case is similar to primary index |
| A4 | Snd. Index, Eq. on Nonkey | $(h_i+n) \times (t_T+t_S)$ | Here, cost of index traversal is the same as for $A3$, but each record may be on a different block, requiring a seek per record. Cost is potentially very high if $n$ is large |
| A5 | Prm. Index, Comparison | $h_i \times (t_T + t_S)+ b \times t_T$ | Identical to the case of $A3$, equality on nonkey |
| A6 | Snd. Index, Comparison | $(h_i+n) \times (t_T+t_S)$ | Identical to the case of $A4$, equality on nonkey |

$t_T$ is time to transfer one block. $t_S$ is time for one seek
$b_r$ denotes the number of blocks in the file
$b$ denotes the number of blocks containing records with the specified search key
$h_i$ denotes the height of the index. $n$ is the number of records fetched

- Nested-loop join
  - Number of block transfers is given by $(n_r * b_s) + b_r$

  where,
    - $n_r$ is the #records and $b_r$ is the #blocks in the outer relation,
    - $n_s$ is the #records and $b_s$ is the #blocks in the inner relation,

  This is so, because for each record in the outer relation, it must perform as many transfers as there are blocks in the inner relation, so that it can compare it with the records in each block in the inner relation. This results in $(n_r * b_s)$ transfers. In addition to this, it must transfer the outer blocks too.

  Total number of seeks = $n_r + b_r$
- Block-Nested-loop join
  - Number of block transfers in a block-nested-loop join = $(b_r * b_s) + b_r$
  - Total number of seeks = $b_r + b_r = 2b_r$
- Index-Nested-loop join
  - Number of block transfers = $n_r * \log_t(n_s) + b_r$

  Reference: https://www.youtube.com/watch?v=DyMWAgGBsJg (after 1:40:00)

  - Cost of join = $b_r$ x $(t_T + t_S) + n_r$ x c
    where,
  - c = cost of traversing index and fetching matching s tuples for 1 tuple in r,
  - $t_T$ = time to transfer one block
  - $t_S$ = time for one seek.

- Equivalence rules
  - $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
  - $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
  - $\pi_{L_1}(\pi_{L_2}(\ldots(\pi_{L_n}(E)))) = \pi_{L_1}(E)$

$$\sigma_\theta(E_1 X E_2) = E_1 \bowtie_\theta E_2$$

- $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$
- $E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$
- $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$
- $(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$
- $\sigma_{\theta_0}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_\theta E_2$ (if $\theta_0$ is applicable only to E1)
- $\prod_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \prod_{L_1}(\bar{E_1}) \bowtie_\theta \prod_{L_2}(E_2)$ (if $\theta$ is applicable only to L1 U L2)
  $E_1 \cup E_2 = E_2 \cup E_1$
  $E_1 \cap E_2 = E_2 \cap E_1$
- $(E1 \cup E2) \cup E3 = E1 \cup (E2 \cup E3)$
  $(E1 \cap E2) \cap E3 = E1 \cap (E2 \cap E3)$
- $\sigma_\theta(E_1 - E_2) = \sigma_\theta(E_1) - \sigma_\theta(E_2)$ (applicable for union and intersection also)
- $\sigma_\theta(E_1 - E_2) = \sigma_\theta(E_1) - E_2$ (applicable for union and intersection also)
- $\pi_L(E_1 \cup E_2) = (\pi_L(E_1)) \cup (\pi_L(E_2))$