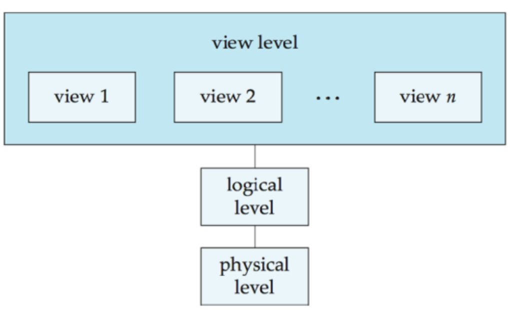
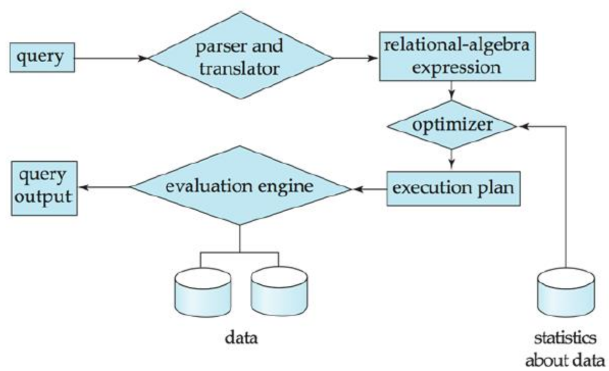
Week1

- 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 this case, the programmer must know what relationships hold between various entities.
- Relational algebra supports the following operations
  - Select (σ) – chooses attributes based on specified condition.
  - Project (π) – selects attributes in the output relation.
  - Union (U) – "concatenates" two relations, provides both have same number of attributes.
  - Difference (-) – output relation has all tuples in the first relation, but not in the second.
  - Intersection (∩) – output relation has all tuples that are common between first and second relations.
  - Cartesian Product (x) – cross-product of both relations.
  - Natural Join (⋈) – 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 the operator ρ
- 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
- *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> **some** $r \Leftrightarrow \exists t \in r$ such that (F <comp> t)
  where <comp> 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');
  ```
  IMPORTANT: (= **some**) ≡ **in**
  However, (≠ **some**) ≢ **not in**
- Definition of *all* clause

  F <comp> **all** $r \Leftrightarrow \forall t \in r$ such that (F <comp> t)
  Where <comp> can be: $<, \leq, >, \geq, =, \neq$

  IMPORTANT: (≠ **all**) ≡ **not in**
  However, (= **all**) ≢ **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.
- $\sigma_p(\boldsymbol{r}) = \{t | t \in r \textbf{ and } p(t)\}$

  where p is a formula in propositional calculus consisting of terms connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
  Each terms is one of:

  $< attribute > op < attribute > or < constant >$

  where op is one of: $=, \neq, >, \geq, <, \leq$
- $r \cup s = \{t | t \in r \text{ or } t \in s\}$
- $r - s = \{t | t \in r \text{ and } t \notin s\}$
- $r \cap s = \{t | t \in r \textbf{ and } t \in s\}$
- $r \times s = \{t\ q | t \in r \textbf{ and } q \in s\}$

- $\rho_{X(A_1,A_2,\cdots,A_n)}(E)$ 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 ∩ 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, D, E) can be obtained by any of the following expressions.
    - $R \bowtie P$
    - $\{< a, b, c, x, y > \mid < a, b, c > \in R \wedge < b, x, y > \in P\}$
    - $\{t \mid \exists m \in R \, \exists n \in P(m.B = n.B \wedge t.A = m.A \wedge t.B = m.B \wedge t.C = m.C \wedge t.X = n.X \wedge t.Y = n.Y)\}$
- X ÷ Y, where $\mathbf{X} = (A, B)$ and $\mathbf{Y} = (B)$ is equivalent to the following expressions.
    - $\{t \mid \exists r \in X \, \forall q \in Y(r[B] = q[B] \implies t[A] = r[A])\}$
    - $\{< a > \mid < a > \in X \wedge \forall < b > (< b > \in Y \implies < a, b > \in X)\}$
- An example of a division operation

A

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

B2

pno
p2
p4

B3

pno
p1
p2
p4

A/B1

sno
s1
s2
s3
s4

A/B2

sno
s1
s4

A/B3

sno
s1

- 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(<u>*rollNo*</u>, *name*, *year*, *courseId*)
course(<u>*courseId*</u>, *cname*, *teacher*)

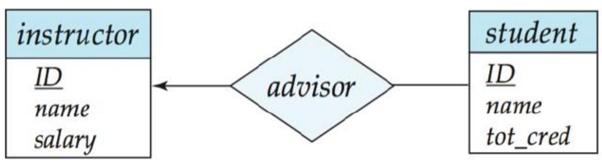
Q.2 Find out the names of all students who have taken the course name 'DBMS'.

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

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

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

- 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]

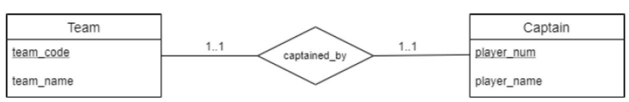


Figure 5: ERD

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:
$\textbf{team\_captain}(\underline{\textit{team\_code}}, \textit{team\_name}, \underline{\textit{player\_num}}, \textit{player\_name})$.

- 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. Consider the following example.

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

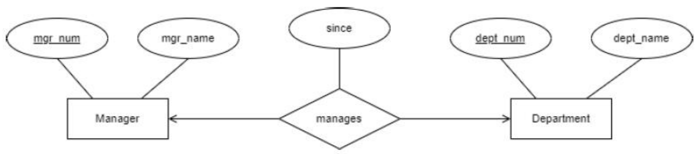
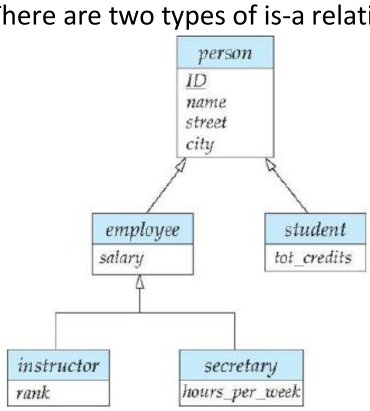


Figure 2: E-R diagram

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

- ○ **Manager**(<u>*mgr_num*</u>, *mgr_name*)
  manages(*mgr_num*, *dept_name*, *since*)
  Department(<u>*dept_num*</u>, *dept_name*)
- √ **Manager**(<u>*mgr_num*</u>, *mgr_name*)
  Department(<u>*dept_num*</u>, *mgr_num*, *dept_name*, *since*)
- √ **Manager**(<u>*mgr_num*</u>, *dept_num*, *mgr_name*, *since*)
  Department(<u>*dept_num*</u>, *dept_name*)
- ○ **Manager**(*mgr_num*, *dept_num*, *mgr_name*, *since*)
  Department(*dept_num*, *dept_name*)

- Attribute types include Simple/Composite, Single-value/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.

Finding 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 it's present/derivable from the (remaining) set of FD's.

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

If A was extraneous, B -> C. B+ = B. Since this cannot derive any dependency, A is not extraneous.

If B was extraneous, A -> C. A+ = ABC. Since this is equal to (AB)+, B is extraneous.

How to create a canonical cover?

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

$A \to BC \Rightarrow A \to B, A \to C$ 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 → BC ⇒ A → B, A → 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 → BC, A → B, B → C, AB → C

A → B
A → C
A → B
→ B → C
→ ~~A~~B → C

⇒ A → B ←
A → C
~~A → B~~ ←
B → C
~~B → C~~

⇒ A → B, A → C, B → C

⇒ A → B, B → C, ~~A → C~~

⇒ A → C

⇒ A → B, B → C

Canonical Cover

A → BC, CD → E, B → D, E → A

A → B
A → C
→ CD → E
B → D
E → A

C → E
D → E

A → D

$A^+ = ABCDE$

$(CD)^+ = CDEAB = ABCDE$

$B^+ = BD$

$E^+ = EABCD = ABCDE$

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

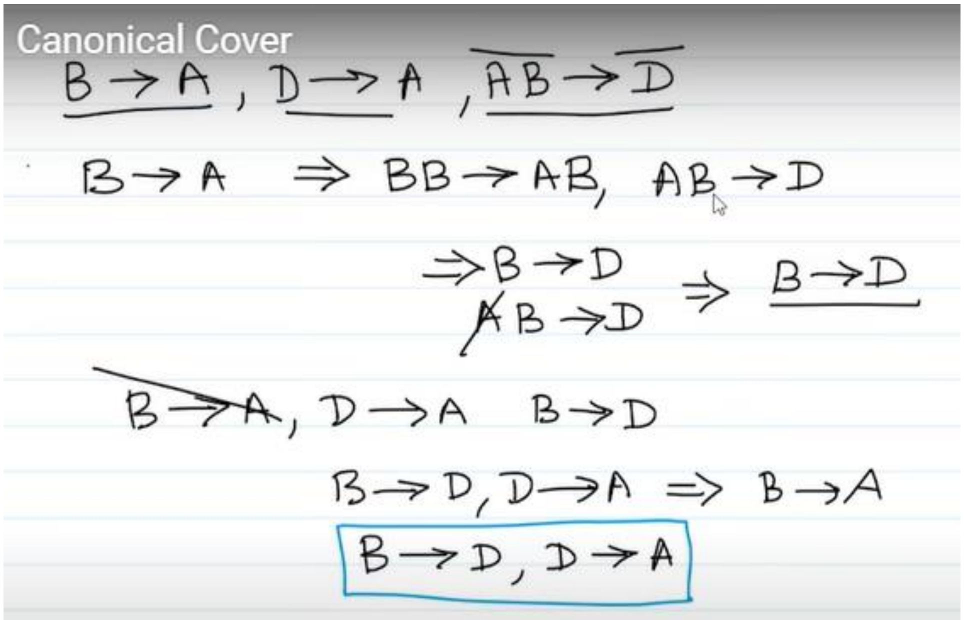
Canonical Cover

A → BCDE, CD → E

A → B
A → C
A → D
X ~~A → E~~
→ CD → E
~~C → E~~
~~D → E~~

A → CD, CD → E ⇒ A → E

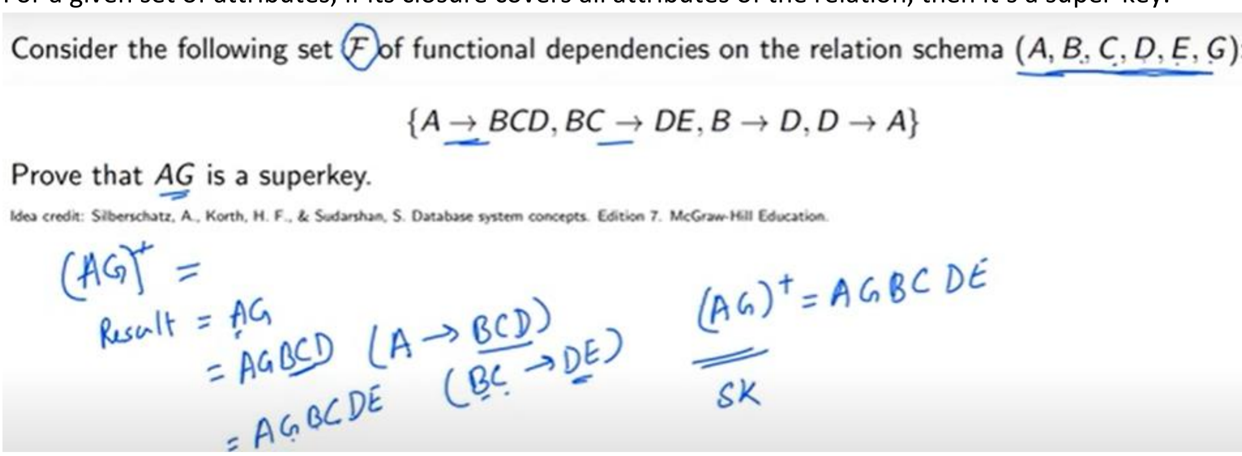
$\boxed{A \to BCD,\ CD \to E}$



NOTE: The original set of FDs and its canonical cover are equivalent.

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.

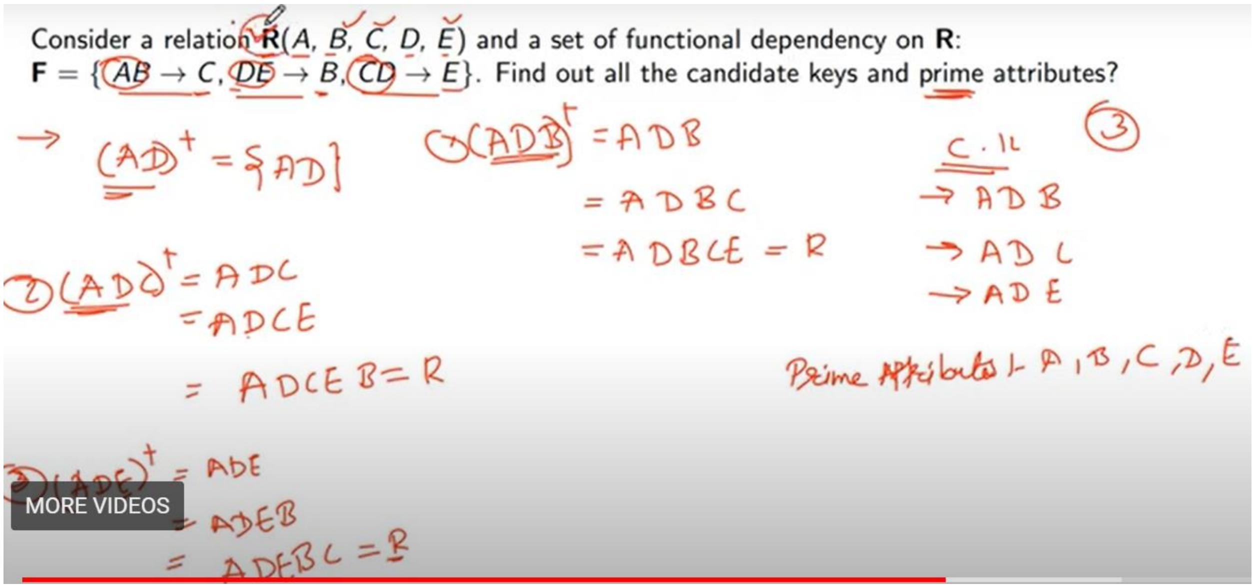
How to identify candidate-keys from a given relation and a set of FDs?

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.



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.

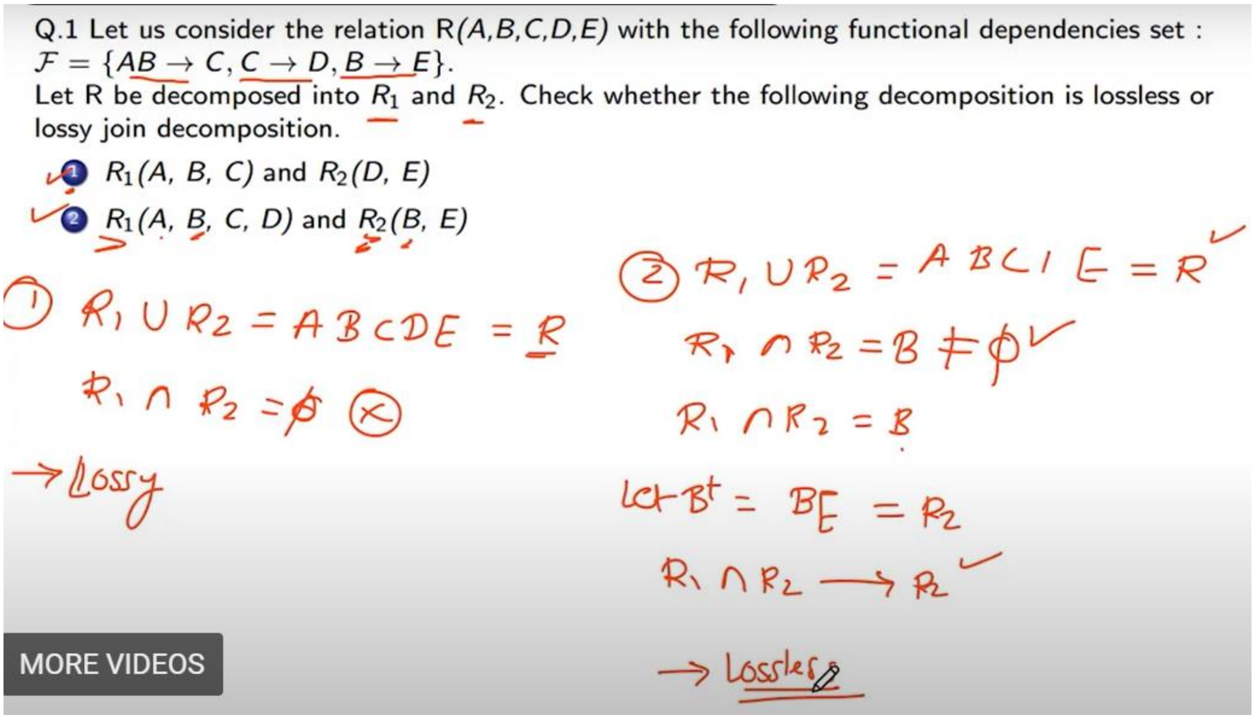
See same screenshot as above.

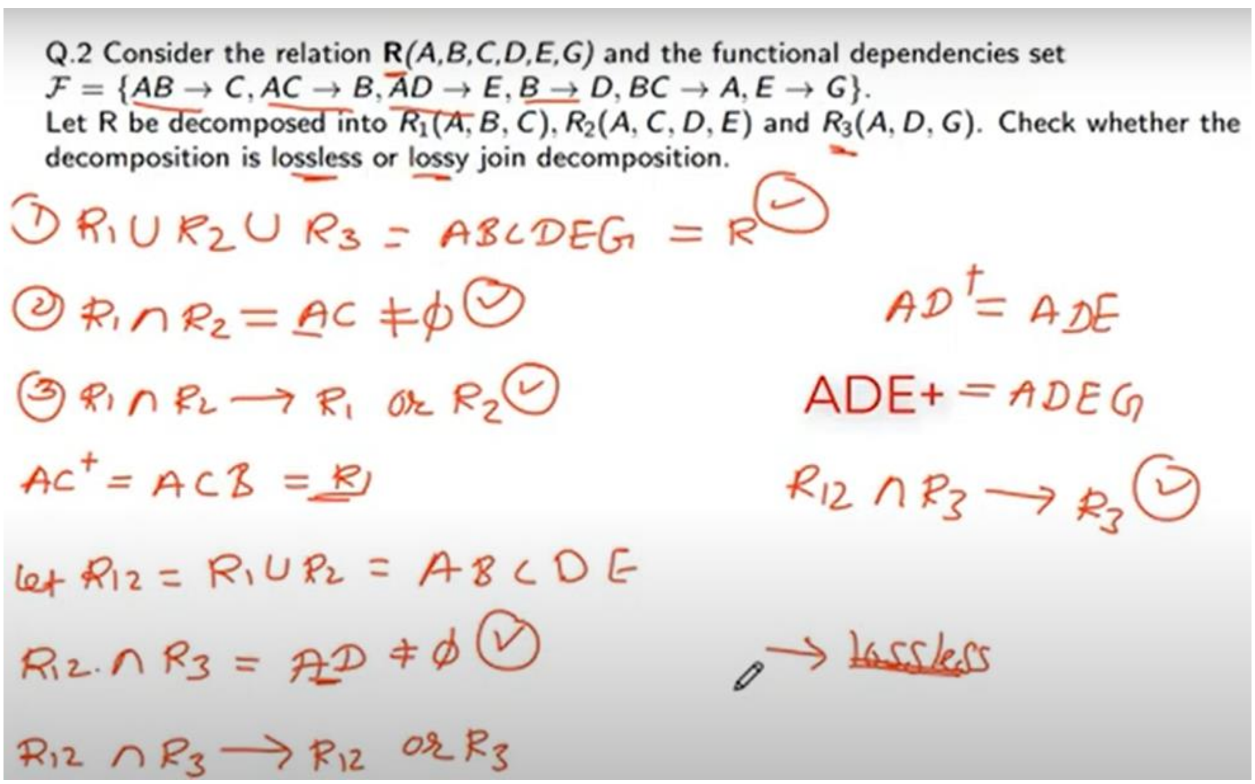
How to identify a lossy decomposition?

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 $R_1 \cap R_2 \rightarrow R_2$





- To check if the decomposition is dependency preserving, find the closure of all attributes (taken 1, 2… at a time). Now, check if the FD in the original set of FD's can be derived from the union of closure sets of all decomposed relations.
  https://youtu.be/xk8e2vvLcGs?list=PLEolU0bP1_Rg1e3z1QJZRiYPOkrV-ZR7J&t=3759
- If we've N attributes in total and one candidate key, the number of super-keys is given as 2^(N-1).

- 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 we’ve N attributes in total, two candidate keys ,first one formed using n1 attributes and second one formed using n2 attributes, the number of super-keys is given as 2^(N-n1) + 2^(N-n2) – 2^(N-(n1+n2))

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

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.
  - 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.
- BCNF – In the case of 3NF, a non-prime attribute can determines a prime. However, in BCNF, only 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.
- A relation is said to be in the first normal form (1NF) when it does not have multivalued attributes
- A relation is in 2NF if it is in 1NF and does not contain any partial dependencies
- A relation is in 3NF if it is in 2NF and does not have transitive dependencies.
- Transitive dependency occurs when a non-key attribute determines another non-key attribute. It happens only in a relation that has three or more attributes.
- 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.

Week7

- Presentation layer is responsible for providing the graphical user interface (GUI). This layers deals with browsers.
- Business Logic Layer and Application Layer are responsible for supporting functionality based on frontend interface of the application.
- Data Access Layer is responsible for storing and accessing persistent data.
- Business Logic Layer and Application Layer links the frontend and backend of an application.
- Business Logic Layer hides the feature of data storage schema
- 1-tier architecture keeps all the elements of an application in one place
- 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.
-

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.
- 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 permits random-access.
- 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 other than leaf nodes (including the root node). Such 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, each node in the LST is less than the value of its root, and each node is the RST is greater than the value of its root.
  NOTE: LST = Left sub-tree; RST = Right sub-tree
- 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, RAM are volatile, while HDD, SSD, tapes, USB are non-volatile memory stores.
- 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

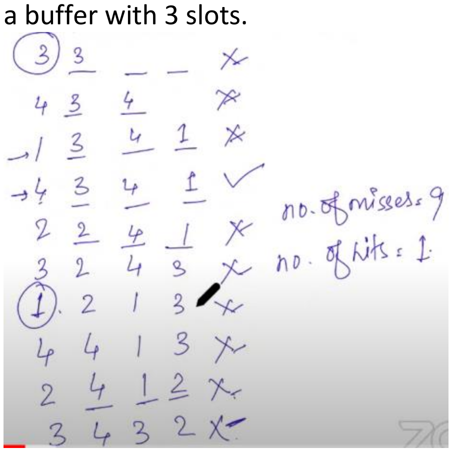
   Access time (per sector) = 3 ms + 1 ms = 4 ms

   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.



Week9

- B, B+ trees.

  https://www.youtube.com/watch?v=aZjYr87r1b8

- Problem-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?

10

4

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

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

S.k	point
20	12

= Blocking factor indexing $= \left\lfloor \frac{2^{10}}{20+12} \right\rfloor = 2^5$

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

NOTE: Sparse index must be created one per block.