DATABASE MANAGEMENT SYSTEM

Minimum and maximum height of a B-tree (m children)

. Combining the above results from minimal and maximal trees, we obtain the following bounds for h:

```
ceil(log (n+1)) <= h <= 1 + floor(log          (n+1)/2)
```

a. In such a tree, the root would have only 2 children (1 key), since this is the minimum allowed for a root. All other nodes would have ceil(m/2) children, and ceil(m/2) - 1 keys.

b. For convenience, let c = ceil(m/2).

```
1 node                1 key                    at level 1
2 nodes               2 * (c-1) keys           at level 2
2*c nodes             2 * c * (c-1) keys       at level 3
2*c**2 nodes          2 * c**2 * (c-1) keys    at level 4
...
2*c**(h-2) nodes      2 * c**(h-2)*(c-1) keys at level h
                      -----------
                      2 * [c**(h-1)-1] + 1 =

                      2 * c**(h-1) - 1 keys total
```

the root) is ceil(2m/3)

Maximum and Minimum number of keys in a B-Tree

```
1 node              m-1 keys                  at level 1
m nodes             m * (m-1) keys            at level 2
m**2 nodes          m * m * (m-1) keys        at level 3
...
m**(h-1) nodes      m**(h-1) * (m-1) keys     at level h
                    --------------------
                    m**h - 1 keys total
```

B+ Tree: The minimum number of block accesses ceil(No of blocks / (No of keys-1)) in the first time, then whatever comes as the output, just divide it with the number of keys and carry on.

Whatever is not aggregated in SELECT must be present in GROUP BY

B$^+$-Trees: Internal nodes contain the indices to the records corresponding to the key values $k_1, k_2,..., k_m$ stored at the internal node. This obviates the need for repeating these key values and associated indices at the leaf level. More efficient than B-Trees in the case of successful searches.

B*-Trees: The minimum number of children at each internal node (except

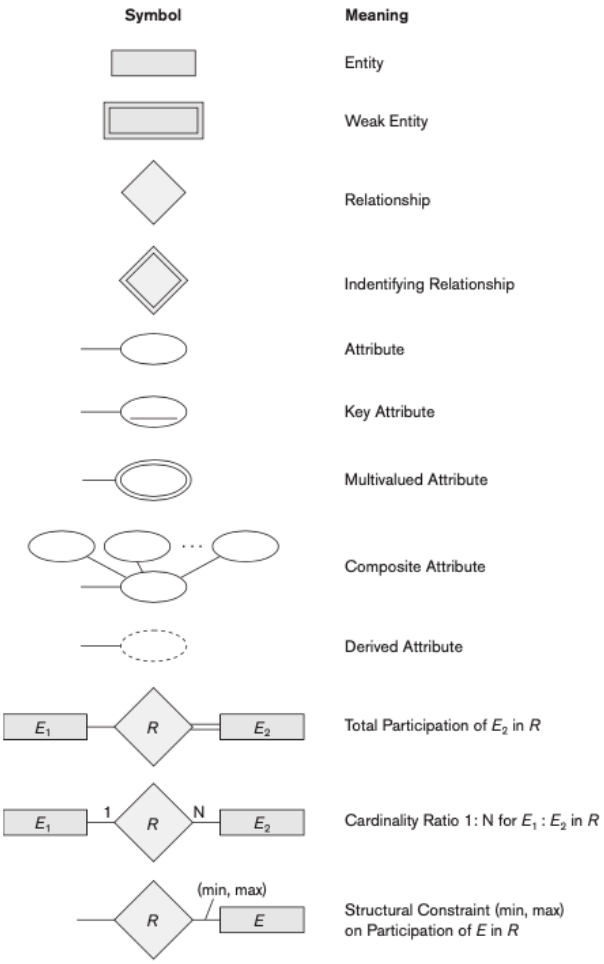


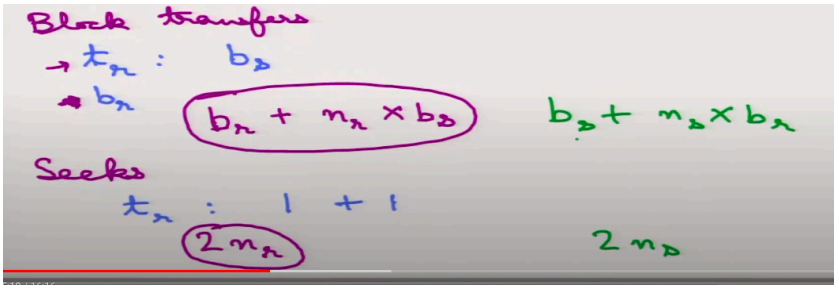
Table 17.1 Types of Indexes Based on the Properties of the Indexing Field

	Index Field Used for Physical Ordering of the File	Index Field Not Used for Physical Ordering of the File
Indexing field is key	Primary index	Secondary index (Key)
Indexing field is nonkey	Clustering index	Secondary index (NonKey)

Table 17.2 Properties of Index Types

Type of Index	Number of (First-Level) Index Entries	Dense or Nondense (Sparse)	Block Anchoring on the Data File
Primary	Number of blocks in data file	Nondense	Yes
Clustering	Number of distinct index field values	Nondense	Yes/no[a]
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records[b] or number of distinct index field values[c]	Dense or Nondense	No

Path Lengths are smaller for obvious reasons. Insertions and deletions are more complex.



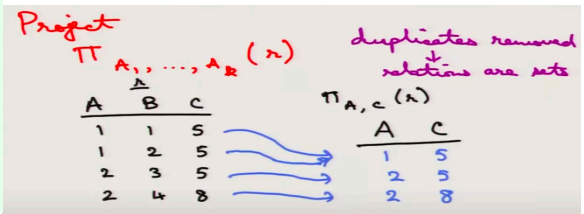
Nested loop Join : Br+NrBs block transfers, Seeks: 2Nr
Block Nested Loop Join: Br + Br*Bs Block transfers, Seeks: Nr+Ns
You can only access elements by their primary key in a hashtable. This is faster than with a tree algorithm (*O(1) instead of log(n)*), but you cannot select ranges (*everything in between x and y*). Tree algorithms support this in Log(n) whereas hash indexes can result in a full table scan O(n). Also the constant overhead of hash indexes is usually bigger (*which is no factor in theta notation, but it still exists*). Also tree algorithms are usually easier to maintain, grow with data, scale, etc. B+ Tree allows duplicate keys

Retrieve all project numbers in descending order in which more than 3 *employees are working*?

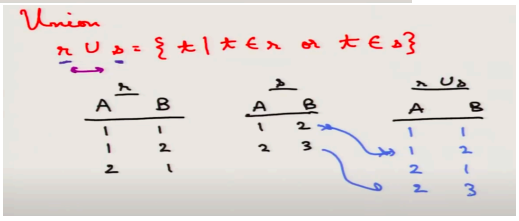
SELECT W.Pno, count(*) as No_employees_working FROM
Project P, Works_on W ON W.pno= P. Pnumber

GROUP BY Pno HAVING **Count(*) >3** ORDER BY Pno DESC;

*ZERO rows selected "SELECT * FROM STUDENT WHERE NULL = NULL";*
Relational Calculus is weaker than relational algebra
Null is a part of every attribute



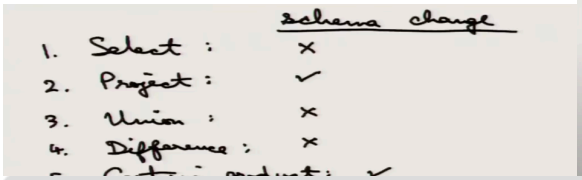
Projection *changes the schema, while* **Selection** *does not in the relation.* **Duplicates** are removed when we **project.** Only Project, Cartesian product and Rename operator changes the SCHEMA. Null == null returns TRUE, otherwise it returns NULL. **4> NULL** evaluates to NULL. Three valued tables for the unknown. Unknown and false is false. Unknown or True is True. Unknown and unknown is unknown. Indexing in Databases can be both done by indexing and B+ Trees

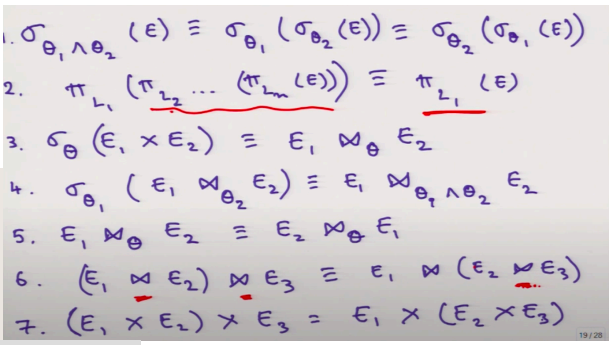


$d(R \cup S) = d(R) = d(S)$ and $\max(|R|,|S|) \le |R \cup S| \le |R| + |S|$

$d(R \cap S) = d(R) = d(S)$ and $0 \le |R \cap S| \le \min(|R|,|S|)$

$d(R-S) = d(R) = d(S)$ and $0 \le |R-S| \le (|R|)$ or $0 \le |S-R| \le (|S|)$





Relational algebra is procedural means, it has a procedure and a way of doing things. It defines the theory, but does not tell us how we should do it.

CHECK constraints enforce domain integrity

UNIQUE constraints enforce the uniqueness of the values in a set of columns

In the index allocation method, an index block stores the address of all the blocks allocated to a file. When indexes are created, the maximum number of blocks given to a file depends upon the size of the index which tells how many blocks can be there and size of each block(i.e. same as depending upon the number of blocks for storing the indexes and size of each index block). *In a **wait-die scheme**, if a transaction requests to lock a resource (data item), **which is already held with a conflicting lock by another transaction**, then **one of the two possibilities may occur − 1.** If TS(Ti) < TS(Tj) − that is **Ti, which is requesting a conflicting lock, is older than Tj − then Ti is allowed to wait until the data-item is available. 2.** If TS(Ti) > TS(tj) − that is Ti is younger than Tj − then Ti dies. Ti is restarted later with a random delay but with the same timestamp.*

```
SELECT * FROM R
WHERE x not in ( SELECT x FROM (
(SELECT x , y FROM (select y from S ) as p cross join
(select distinct x from R) as sp)
EXCEPT
(SELECT x , y FROM R) ) AS r );
```

Implementation 2 : Using correlated subquery

```
SELECT * FROM R as sx
WHERE NOT EXISTS (
(SELECT p.y FROM S as p )
EXCEPT
(SELECT sp.y FROM  R as sp WHERE sp.x = sx.x ) );
```

1-1 relationships
Depending on the optionality of the relationship, the entities are either combined or the primary key of one entity type is placed as a foreign key in the other relation.
1-m relationships
The primary key from the `one side' is placed as a foreign key in the `many side'.
m-n relationships
A new relation is created with the primary keys from each entity forming a composite key.

This scheme allows the older transaction to wait but kills the younger one.

The basic rules for converting the ER diagrams into tables are:

- **Convert all the Entities in the diagram to tables:** All the entities represented in the rectangular box in the ER diagram become independent tables in the database.
- **All single-valued attributes of an entity are converted to a column of the table:** All the attributes, whose value at any instance of time is unique, are considered as columns of that table.
- **The key attribute in the ER diagram becomes the Primary key of the table.**
- **Declare the foreign key column, if applicable:** attribute COURSE_ID in the STUDENT entity is from COURSE entity. Hence add COURSE_ID in the STUDENT table and assign it a foreign key constraint. COURSE_ID and SUBJECT_ID in the LECTURER table form the foreign key column. Hence by declaring the foreign key constraints, the mapping between the tables are established.
- **Any multi-valued attributes are converted into the new table:** A hobby in the Student table is a multivalued attribute. Any student can have any number of hobbies. So we cannot represent multiple values in a single column of the STUDENT table. We need to store it separately, so that we can store any number of hobbies, adding/ removing/deleting hobbies should not create any redundancy or anomalies in the system. Hence we create a separate table STUD_HOBBY with STUDENT_ID and HOBBY as its columns. We create a composite key using both the columns.
- **Any composite attributes are merged into the same table as different columns:** Address is a composite attribute. It has Door#, Street, City, State, and Pin. These attributes are merged into the STUDENT table as individual columns.
- **One can ignore derived attributes since it can be calculated at any time:** In the STUDENT table, Age can be derived at any point in time by calculating the difference between DateOfBirth and the current date. Hence we need not create a column for this attribute. It reduces the duplicity in the database.

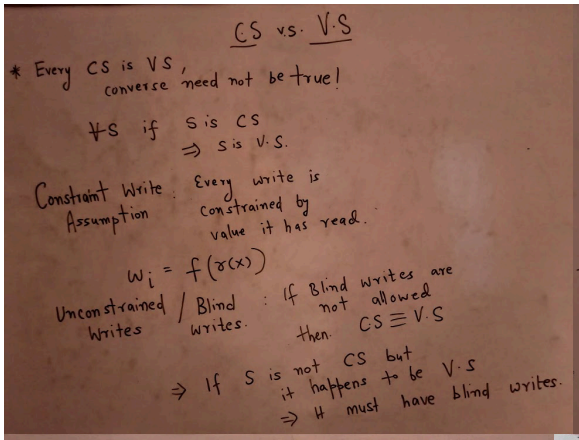
Converting Weak Entity: A weak entity is also represented as a table. All the attributes of the weak entity form the column of the table. But the key attribute represented in the diagram cannot form the primary key of this table. We have to add a foreign key column, which would be the primary key column of its strong entity. This foreign key column along with its key attribute column forms the primary key of the table.

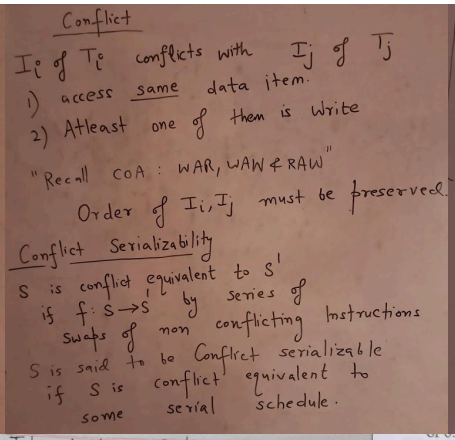
Representing 1:1 relationship We have LECTURER teaches SUBJECT relation. It is a 1:1 relation. i.e.; one lecturer teaches only one subject. We can represent this case in two ways: Create a table for both LECTURER and SUBJECT. Add the primary key of LECTURER in the SUBJECT table as a foreign key. It implies the lecturer name for that particular subject. Create a table for both LECTURER and SUBJECT. Add the primary key of SUBJECT in the LECTURER table as a foreign key. It implies the subject taught by the lecturer. In both the cases, the meaning is the same. The foreign key column can be added in either of the tables, depending on the developer's choice.

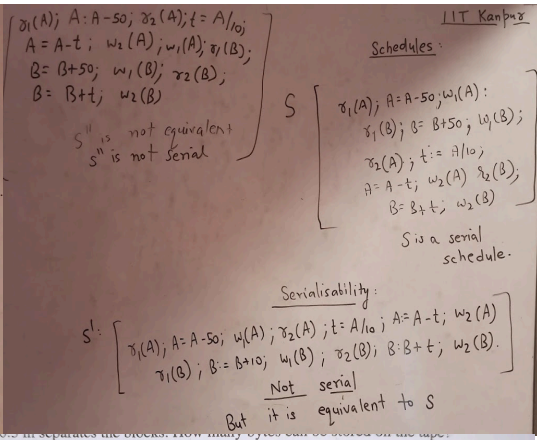
Representing 1:N relationship: Consider SUBJECT and LECTURER relation, where each Lecturer teaches multiple subjects. This is a 1: N relation. In this case, the primary key of the LECTURER table is added to the SUBJECT table. i.e.; the primary key at 1 cardinality entity is added as foreign key to N cardinality entity

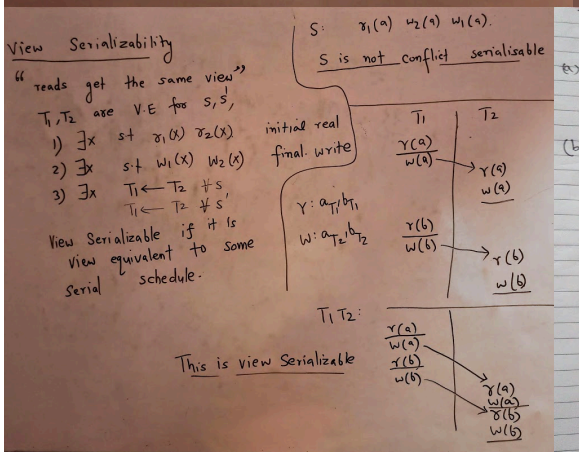
Representing M:N relationship: Consider the example, multiple students enrolled for multiple courses, which is M:N relation. In this case, we create STUDENT and COURSE tables for the entities. Create one more table for the relation 'Enrolment' and name it as STUD_COURSE. Add the primary keys of COURSE and STUDENT into it, which forms the composite primary key of the new table. Both the participating entities are converted into tables, and a new table is created for the relation between them. Primary keys of entity tables are added into the new table to form the composite primary key. We can add any additional columns if present as an attribute of the relation in ER diagram. Candidate keys are permitted to be NULL, unless it is explicitly mentioned in the question, but primary keys are always unique and not null by the definition. Cross product and Natural join are both commutative and associative For strict locks,Exclusive locks should be released after commit. No locking can be done after the first unlock and vice versa. In case of rigorous, along with being strict, shared locks also should be released after commit. Hashing works well on equal queries while ordered indeed works well better on range queries too. For example consider B+ tree, once you have searched a key in B+ Tree, you can find a range of values via the block pointers pointing to another block of values on the leaf node level. A relation is in BCNF iff every LHS is a SK. A relation is in 4NF iff for every Non trivial multivalued dependency A--->>B, A is an SK of the relation.

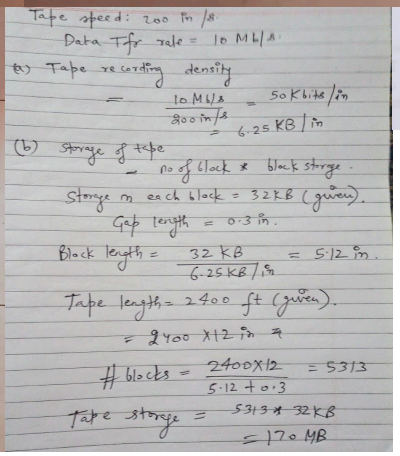
A multivalued dependency represents dependency between attributes A,B and C in a relation such that for each value of A there is a set of values of B and a set of values of B and C are independent of each other. A lossless join dependency is a property of decomposition which means that no spurious tuples are generated when relations are combined through a natural join operation. Every weak entity has to be related to a strong entity through a strong relationship. A weak entity cannot be uniquely identified by its own attributes. The participation of weak entities in its identifying relationship is total. A weak or non identifying relationship exists between two entities when PK of one of the related entities does not contain a PK component of the other related entities. A strong or identifying relationship is when the PK of the relation entity contains the PK of the parent. Index whose search key species the sequential order of the file is called a primary index. Secondary index must be dense as the data file is nor ordered by the search key and so with a sparse index (which points to a block of records than an individual record) it is impossible to retrieve all the keys. On the contrary, the primary index can be sparse. This is because the data file is ordered as per the search key for a primary index. So, using the search key we can obtain the block of records where our key could be located and then do a sequential scan to obtain the actual record. In a clustered index, order of data records is the same as order of index entries. While converting from ER diagram to relational model, we need to be certain that if we strone an identifier for one entity in a relation representing another entity then the identifier never has a null value. Also we should not introduce any redundancies. If both the entities participate fully, we need two relations as one entity related to multiple other entities. Here, we will have 2 relations for both the entities and the relation for the entity at the “one end” will have the key of the entity at the “many end” i.e. we cannot have the key of the entity at 1 end to the relation for the entity at many end as this will cause multi valued attributes since one entity in many side maps to multiple entries on the one side. If participation of many-end is mandatory we can just use the above 2 relations and we won't have any null values. If the participation of many-end is optional then the above construction fails due to nulls and we need to have 3 relations, one each for the 2 entities and another for the relationship having the keys of the 2 entities. If participation of both the entities is optional then also we need 3 relations. By definition, a prime attribute is any attribute part of **some,** not necessarily all candidate keys.











6.18. A nine-track magnetic tape has fixed block and interblock gap sizes. The gap length is 0.6 in, and the storage density is 1600 B/in. (*a*) If the space utilization *u* is 70%, what is the block size in bytes? (*b*) Let the start-stop time be 1 ms and let the measured (effective) data-transfer rate be 55 KB/s to read a single block. What is the maximum possible data-transfer rate?