What is indexing?

Indexing is a way of sorting a number of records on multiple fields. Creating an index on a field in a table creates another data structure which holds the field value, and pointer to the record it relates to. This index structure is then sorted, allowing Binary Searches to be performed on it.

What is hashing?

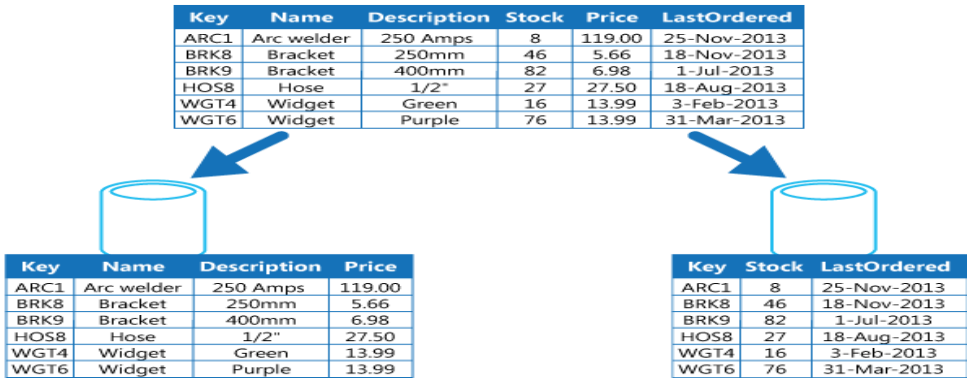
Hashing is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string. Hashing is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value.

Differentiate between partitions and views

Partitioning enables you to decompose very large tables and indexes into smaller and more manageable pieces called partitions. Each partition is an independent object with its own name and optionally its own storage characteristics.

A view is a logical representation of one or more tables. In essence, a view is a stored query. A view derives its data from the tables on which it is based, called base tables. Base tables can be tables or other views. All operations performed on a view actually affect the base tables. You can use views in most places where tables are used.

Describe Partitioning using examples



Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Price
ARC1	Arc welder	250 Amps	119.00
BRK8	Bracket	250mm	5.66
BRK9	Bracket	400mm	6.98
HOS8	Hose	1/2"	27.50
WGT4	Widget	Green	13.99
WGT6	Widget	Purple	13.99

Key	Stock	LastOrdered
ARC1	8	25-Nov-2013
BRK8	46	18-Nov-2013
BRK9	82	1-Jul-2013
HOS8	27	18-Aug-2013
WGT4	16	3-Feb-2013
WGT6	76	31-Mar-2013

When would you recommend the use of partitioning in a database design

Partitioning is used in Distributed DBs where applications are located at different sites but can also be used in centralised systems if we want to optimise access or storage for particular applications. The DBA can apply different designs for each section (Partition) to optimise the overall application.

Explain the advantages/disadvantages of using a partitioned design

ADVANTAGES: Increased availability, Easier administration of schema objects, Enhanced query performance in data warehouses

DISADVANTAGES: multiple physical objects storing data, /management is more difficult, different disk locations.

Explain the term physical database design

Physical database design translates the logical data model into a set of SQL statements that define the database. For relational database systems, it is relatively easy to translate from a logical data model into a physical database.

Explain the role of the internal level of a generalised database architecture (or Conceptual Level?)

This is where all the logical definition of all tables and data constraints are defined. Database designers work at this level.

Explain why all relational DBMSs different even though they all use the same language SQL? (Levels and separation of data structure/ processing)

The 3 levels explain why all DBMS products are different at the internal level. They all use standard SQL for the external and conceptual levels. However, the separation of the internal level makes the DBMS functions independent of the higher levels. Therefore, any DBMS can basically implement that level however they want. So for example, you might be 'expert' in SQL Server Admin, but not MySQL database system. Note OSI Network model has 7 layers. Each layer makes the higher level independent of low level details.

SSDs, what are they, why use them, what limitation?

Solid State drives (SSDs) eliminate the slow arm movements in Hard Disks and are increasingly used in data systems (particularly mobile/tablet). These are integrated circuits that store binary data by voltages (similar to RAM). However, cost remains prohibitive for large data applications, and lifetime for repeated writes is also an issue.

NoSQL, what are they, how different to RDBMS, what type of application are they used for?

A **NoSQL** (often interpreted as Not only SQL) database provides a mechanism for storage and retrieval of data that is modelled in means other than the tabular relations used in relational databases. Motivations for this approach include simplicity of design, horizontal scaling, and finer control over availability.

A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Motivations for this approach include simplicity of design, horizontal scaling, and finer control over availability. RDBMS focuses on the record/row as the unit of control/storage. Used for real time web applications.

Used for "**Big Data**" applications.

Explain how to calculate the storage requirements for an application.

Every database table and Index is made up of columns. Each column is defined as a particular data type e.g. integer, Char etc. Each datatype requires a number of bytes of storage.

We need to know 1. Record size 2. How many records to handle

1. Add up all the datatypes used in one record of a table to determine the space required for each record of that table.

2. The designer must then analyse the expected number of records for the table; and multiple these two values.
3.

Another factor to take into account in estimating the number of rows is the activity on the table; i.e. what type of application uses the tables; is it static (not much insert/update/delete), or highly dynamic. If it is dynamic, do the number of expected insert operations outnumber the number of delete? If so, a percentage growth should be factored into the table.

However, we have another consideration in that the unit of memory allocation is usually the 'page'. This can be 2K to 8K bytes normally in databases. So you need to divide the page size in use by the size of each record. This tells you the number of records that fit in each page.

No. of records per page divide into expected no. of records = total number of pages of storage.

Note: Extra costs of Indexes and Fill factor used in certain file organisations

Explain the difference between fixed and variable length records. & link to compression.

Note: data types **Char or Varchar**.

Using variable-length records might enable you to save disk space. When you use fixed-length records, you need to make the record length equal to the length of the longest record. If your application generates many short records with occasional long ones, using fixed-length records wastes a lot of disk space, so variable-length records would be a better choice.

Trade-off is the extra time to process the variable length, thereby slowing response to the query looking for the data, V's the extra space usage. Storage capacity can be large e.g. disks are cheap and have high capacity, so extra space may not be a serious issue for the system.

Update, also is handled differently. In a Fixed record, update is simple, just replace old with new.

However, in a variable record the length may grow, requiring relocation of the record elsewhere on the disk (re-insert) or may reduce in size. In this case the system can replace the old with the new record but this introduces **fragmentation** (unused space within the file).

Why do some file orgs need reorganisation (link to OS page table allocation, non contiguous memory allocation, and therefore seek times to overflow)

When would you need to reorganise a file? When it no longer performs efficiently. How can performance deteriorate? 1. Change of access pattern (i.e. change to the type of SQL query). The old file org may not suit the new queries/reports/ 2. Deterioration in performance due to insertions and deletions over the life time of the application. This can severely affect performance by unused or inefficiently used space.

Re-organisation entails rebuilding the file 'from scratch'. Reorganisation of a sorted/ordered file may write all records into a temporary file (heap) and then **reinsert every record in the correct logical sequence** into the (new) main file thereby eliminating any unused space e.g. space available after delete, overflow areas, etc.

What does fragmentation mean in terms of database records?

Data can be stored in different computers by fragmenting the whole database into several pieces called fragments. Each piece is stored at a different site.

Fragments are logical data units stored at various sites in a distributed database system.

Advantages of fragmentation

Before we discuss fragmentation in detail, we list four reasons for fragmenting a relation

Usage

In general, applications work with views rather than entire relations. Therefore, for data distribution, it seems appropriate to work with subsets of relation as the unit of distribution.

Efficiency

Data is stored close to where it is most frequently used. In addition, data that is ,not needed by' local applications is not stored.

Describe the basic file organisations a DBMS might implement?

Types of File Organisations:

1. Unordered (Heap)
2. Ordered (Sequential)
3. Random (Hash)
4. Indexed: B-Tree index is standard in DBMS.

Data (I/O intensive) V's Processing (CPU) Intensive Software applications:

Different types of software applications require different type of file organisation. A programming application with complex processing of data ('number crunching') may not be too interested in the disk file (I/O) as it may be able to read its data into RAM for processing any way or is read only. These are called processing intensive applications (CPU heavy).

However large data intensive applications have very large files, and the access to the data on the storage medium is the performance bottleneck (I/O heavy) due to mixed read (input) and write (output). A write includes (Insert, Update. Delete). An added problem is that these data intensive applications may process this large bank of data in different ways and therefore would like the data to be organised differently so that processing is efficient.

Explain the term fill factor and how it impacts on storage requirement (file size).

Fill factor is the value that determines the percentage of space on each leaf-level page to be filled with data.

Fill factor is the extra space you leave in data in case you want to insert more data or update data already being stored to the extent that it is going to take up more space. It wastes space as you might not use all the fill factor you put aside.

Use the SQL Create Index command

CREATE index indx1 ON suppliers (City)

What is a clustered index?

A **clustered index** is a special type of **index** that reorders the way records in the table are physically stored. Therefore table can have only one **clustered index**. The leaf nodes of a **clustered index** contain the data pages.

How many indexes can you define on a table?

Depending on the DBMS being used. In SQL server there can be 1 clustered index and up to 999 non-clustered indexes

Describe the concepts of compression and clustering

As the name suggests, compression means 'squeezing' data into a small space. As the cost of disk storage has fallen, compression is not as important as it once was. However compression itself remains widely used. Compression not only saves disk space, it can help you organize and archive (backup) older files that you don't use regularly.

It has also become a standard for allowing the easy transfer of large data.

All DBS have facilities to compress tables, e.g. remove trailing blanks. However, this is a basic compression technique. Another technique focuses on removing unnecessary data at the start of the data e.g. in CIT all local IP addresses are 157.190 so if you can eliminate that section (a 50% saving).

When deciding whether to use compression or not, the DBA must consider the trade-off of space verses performance: if you compress you save space but time is required to decompress when the data is required.

Clustering

Primary indexed sequential files are usually clustered on the primary key by default.

Data placed on secondary storage are organised into files. Normally, files are physically made up of clusters possibly spread over the disk (**non-contiguous**). A database table usually is stored in a file on a disk. But we look at a table in a different way independently of how it is physically organised. A file (or table) is usually a collection of related information organised in record structures e.g. a Employee file is made up of employee records, which are made up of data fields such as name, address, position etc. A table is a logical concept not tied to a physical implementation.

What is file reorganisation, and why/when is it required?

De fragmenting, required when fragmentation is such that performance degrades.

What is file/table partitioning, and why/when is it required?

Partitioning (fragmenting) a table means dividing it up into separate sections (smaller)

2 types of partitioning: Vertical and Horizontal

How might the designer divide up a table? 1. By vertical split of the attributes (i.e. divide down the table) or 2. By horizontal split of the rows of the table (i.e. divide across the table)

Vertical Partitioning: same number of rows in each partition, but less attributes.

Fragments a table by grouping attributes i.e. it splits the table into smaller ones by dividing up the columns.

Use SQL Join to recombine the vertical partitions therefore NB: need to replicate key column.

Horizontal Partitioning: less rows in each partition, but same attributes/data structure

Fragments a table by grouping rows i.e. it splits the table into smaller ones by dividing up the rows.

This is similar to an SQL View, except SQL views are virtual tables (the rows of the view are determined when the View is called/used). In partitioning, we go a step further and actually create physical tables for each partition and the original base table is divided up.

Discuss how a designer should choose a data type for a table column for efficient physical design? (and problems with not following this)

Char vs VarChar() wasting disk space.

Explain how different SQL queries require different processing of the underlying table/file.

They all use standard SQL for the external and conceptual levels. However, the separation of the internal level makes the DBMS functions independent of the higher levels. Therefore, any DBMS can basically implement that level however they want. So for example, you might be 'expert' in SQL Server Admin, but not MySQL database system. Note OSI Network model has 7 layers. Each layer makes the higher level independent of low level details.

Explain the effects on performance of inappropriate data types for table columns

Waste of disk space if, for example using varchar(200) for something that will only ever hold 1 char.

Differentiate between data and processing intensive applications.

Disk cost vs computation cost.

Explain the difference between a Primary and a Secondary index?

Primary index: A primary index is an index on a set of fields that includes the unique primary key for the field and is guaranteed not to contain duplicates. Also Called a **Clustered index**. eg. Employee ID can be Example of it.

Secondary index: A Secondary index is an index that is not a primary index and may have duplicates. eg. Employee name can be example of it. Because Employee name can have similar values.

The primary index contains the key fields of the table. The primary index is automatically created in the database when the table is activated. If a large table is frequently accessed such that it is not possible to apply primary index sorting, you should create secondary indexes for the table.

What data does a designer need to perform physical database design?

Physical Design: The designer needs to know specific information about the applications that'll access the database in order to evaluate what organisation to use. Explain. You should link particular SQL statements to processing requirements which in turn are best suited to particular organisations.

Explain the difference between a Dynamic application and a dynamic file organisation?

The database application will need to be off line (no users) while reorganisation takes place. Static reorganisation may be unacceptable in a busy application, so some file organisation can dynamically handle inserts and deletes to maintain the file organisation in an efficient manner i.e. reorganise after each operation rather than waiting for a complete file rebuild after a period of time. These are called **Dynamic file organisations.**

An application that uses a reference file of data for lookup requires only a static file as it has little or no inserts/deletes. This is called a Static file organisation.

When would you recommend the use of a Heap file (or an ordered file, or a hashed, Indexed, Indexed sequential)

Heap File Organisation:

Simple Org, 100% fill factor i.e. no space left for future insertions. This has the adv of maximum space utilisation compared to the other organisations.

- So, a heap is useful for large files that are to be fully read (full scan); alternatively
- they are good for small files that can fit into Main Memory for processing (i.e. no need to keep an optimised disk organisation as you will use faster RAM main memory to do the searches).
- Heaps are also used as temporary files for file re-organisation.

Since there is no order or access mechanism as part of the heap, all SQL there is no means to identify and retrieve a single record, a group, or sequence of records. Each Select is forced to read the entire file.

As a heap is space efficient, **they are used in conjunction with secondary indexes when the dominant access pattern is via non key fields.**

Note: you can only insert into the end of the file.

Ordered/Sequential File Organisation:

The file is maintained in order on a key column (or combination of columns). Fill factor is reduced to allow insertions into the file without the need for reorganisation. But this increases the overall file size compared to a heap. The benefit is that a search can use the order to implement key uniqueness and stop searching once a record key greater than the search key is found (average half the file search time). The order enables efficient range searches (SQL Between) and partial matches (SQL Like). These efficiencies only apply to the key column the file is ordered on. Any SQL selects that use other columns can make no use of the order and therefore end up reading the entire file. Note this full read is less efficient than a heap due to the reduced fill factor (larger file size).

Index structure (index file organisation)

Each DBMS implements its own index file structure(s) e.g. a basic sequential file, Btree, Hash etc. In other words, an index can be created in different file structures. This would results in a B+ Index, or a Hashed Index. Some offer more than one index structure i.e. you can decide which type of Index file you want, e.g. a hashed index, Btree index.

Indexed Sequential (B-Tree) files.

The standard DBMS Index structure is called a B-Tree. It can be used for both ordered data and index files and it allows reasonably efficient access for both ordered and random (lookup) processing. B-Tree can be used for Indexed Sequential file organisation. (Index & data in one file). Hashed Indexes should only be used for random lookups of the key value.

Indexed Sequential file organisation is a 'general purpose' organisation. It is a file that is organised as an index AND an ordered (sequential) file. Note that it may not be optimal compared to others but caters for mixed queries e.g. it isn't as good as hashed for random lookups but gives acceptably good performance

Explain why each of the operations Insert, Update and Delete may result in deterioration of the performance of a Static file organisation?

Insert, update and delete operations result in file fragmentation which will degrade the performance of the application over time due to increased seek times on the disk.

Explain whether a Btree is an index or a datafile organsiation?

How can a Btree be used for both the index and the data file a the same time?

A Btree can be found in a number of forms. The form depends on what is in the nodes of the tree. We must store any pointer(s) (required for Tree structure), we won't display these, we'll just assume they are there. In a Primary Index, we must store the key of the database record. However, what else is stored in each node? i.e. besides the key and the tree pointers themselves.

Explain why Btrees are recommended for high availability applications?

A balanced Btree is a dynamic file organization. It is a self-organizing file organization in that it dynamically reorganizes itself as inserts, deletes occur to maintain the balance. The DBMS does not stop user activity to reorganize. **This is essential in high availability systems.**

Describe how Btrees are dynamic (or self organizing) & Explain what is meant by a balanced Btree?

A balanced Btree is a dynamic file organization. It is a self organizing file organization in that it dynamically reorganizes itself as inserts, deletes occur to maintain the balance. The DBMS does not stop user activity to reorganize. **This is essential in high availability systems.**

A Btree maintains balance for Inserts & Deletes by rearranging its root and branches to keep equal depth on either branch from the root. For Inserts, it does this rearrangement while adding new pages into the tree as required. A new page is only needed when the fillfactor has all been used up in a page.

Compare binary search and balanced BTrees.

Start the search at mid-point of file, compare with search key, reset search range to left or right sections of the file depending on the key comparison. This reduces the area to search by half each

time. Repeat/continue searching in this way until you locate the record searched for. Better search as less variability; more uniform response for any key. This is faster than a linear search.

A binary search is an RAM ('in memory') search; where you can have an array to search; each cell in the array can be referenced. However, how can you apply a binary search to a large disk file? See BTree later.

What is the difference between a BTree and a B+Tree?

Rather than just having two branches, B+ trees use the same principle as above but have more pointers (or records) per node. Think of having an array of records at each node rather than just one. This allows you have a fillfactor so that you can add to each node without having to reorganize the B+tree at each insert.

Explain how indexes might be managed for large batch inserts/updates?

The Index gives fast lookups for a random search e.g. on the key value; however it is not as fast as a hash (as the index must be read and processed first then the data file is accessed once you know where to look in it).

Note: for application with large batch inserts, it may be faster to drop indexes before the batch insert and then reform the index afterwards

What is a composite Index? When is it used?

The attribute list in the Create Index syntax, allows the creation of a composite index e.g. (sex, dateofbirth) would allow fast lookup of people by gender and date. Composite indexes can be critical for performance because only the smaller index file needs to be searched (scanned on disk) or even better if the small Index is stored in RAM). Avoid Data file Seeks.

Explain the term 'address space reduction' in a hashing algorithm.

Address space reduction : is a fundamental concept in hashing. In some applications the database designer doesn't have a choice in the format of the key, e.g. employee uses Prsi_no(8 digits), student uses CAO/CAS no. These integer formats are huge in comparison to the actual number of records required (e.g. company with 200 employees). This is referred to as key address space V's the actual file address space.

Discuss the factors that influence efficiency of a hashing algorithm.

Collisions occur when two or more records compete for the same address. Therefore we should try to find a hashing algorithm that distributes records fairly evenly among the available addresses
Use a bigger table • The more free slots in the table, the less likely there will be a collision. But if you are doing lots of accesses, using a bigger table will reduce the likelihood that two accesses will reference the same part of the disk

What is sharding?

Sharding is a type of **database** partitioning that separates very large **databases** the into smaller, faster, more easily managed parts called **data** shards. The word **shard** means a small part of a whole. Sharding is a common term used in modern DBMS. It is complex version of Horizontal Partitioning for large scale distributed applications. It allows some replication of data for fast read response, but has added complexity for modifications.