What is a VSAM Data Set?

#7 IN A SERIES

VSAM is an acronym that stands for "Virtual Storage Access Method". Knowing that, unfortunately, doesn't help you understand what VSAM is; by itself, the acronym is not very descriptive. In fact, the acronym "VSAM" by itself is used to refer to: VSAM catalog management, the VSAM access method, and VSAM record management. So, you have to understand the context of the acronym usage to know which of the three are being referenced. Technically, the VSAM catalog is no longer supported and its replacement is known as the Integrated Catalog Facility (or the "ICF catalog"). "Under the covers" the ICF catalog uses VSAM data sets to store catalog information. The VSAM access method is the intermediary between an application program and the VSAM data sets. It provides an application programming interface that makes it relatively easy for an application programmer to read and write a VSAM data set without having to understand the intricacies of its data set structure. In this article, we will focus on VSAM record management.

Data Stored on Disk

Before we get into the details of record management for VSAM data sets, let's understand why having a record structure for data sets stored on disk is important. In general, the data sets businesses use to manage their clients, inventory, accounts, etc. contain thousands and possibly millions of records. To facilitate direct or random access processing, each record usually contains a unique identifier called a key. The key can be an account number, a part number, a customer number, etc. but it must uniquely identify a record. Businesses want to be able to process disk data randomly some of the time (i.e. by going directly to a specific record without having to process or look at all the records that precede it in the data set) and sequentially (i.e. in key order) at other times. And, a business wants the applications that process the data to have excellent performance, no matter how the data set is processed. "Excellent performance" implies that there should be a minimal number of read or write operations used to access disk data, and movement of the read / write heads on the disk should be avoided. Disk head movement involves mechanical motion which is very slow compared to electronic operations that can be performed by the CPU for example. Businesses also want to be able to modify a disk data set once created. So, they want to be able to insert new records in an existing key ordered data set and maintain the data set's order. While this sounds simple, it is actually quite challenging.

Besides random and sequential access with good performance, there are a number of other requirements for the storage and access of disk data. These include the ability to:

- Substantially increase the overall size of the initial data set without affecting access performance
- Provide disk device independence to allow movement of data sets and introduction of new disk technologies without having to re-write programs or re-structure stored data.
- Delete records and reuse the space they occupied
- Change the length of a variable length record when updating the record

All of these requirements for disk data storage and access argue for a data set structure that avoids a "brute force" approach that would simply re-write the entire data set whenever a change is made. VSAM data sets have an internal structure that does all of this and more. (Note: disk files on personal computers do not have the same requirements; they are usually not as large and do not have a need for frequent random access from an application program.)

Early Disk Data Storage

Prior to the introduction of VSAM record management in 1973, software developers had limited help in creating and managing direct access disk data sets. A software developer could choose from a variety of data formats and organizations that were supported, but not all of which were optimized for disk data set usage. The indexed sequential access method (ISAM) and organization was best suited to disk data, but it had shortcomings including not being able to effectively handle insertion of many records with closely related keys and not being able to reuse deleted record space. VSAM overcame these obstacles and in addition, provided a robust set of extended capabilities to effectively access disk data (e.g. random access by relative record number or relative byte address, alternate indices, generic key access, approximate key access, etc.). There are more modern implementations of ISAM on several operating systems, but VSAM remains the "gold standard" to which other disk data access implementations are compared. As we will see, VSAM has continued to evolve and enhance its functionality.

VSAM Data Set Overview

VSAM provides four data set types or organizations:

1. key sequenced data set (KSDS)
2. entry sequenced data set (ESDS)
3. relative record data set (RRDS)
4. linear data set (LDS)

All four data set types share some characteristics:

- VSAM data sets are always stored on disk
- VSAM data sets are always cataloged
- VSAM data sets are always read by an application using a GET or READ request and written using a PUT or WRITE request.
- VSAM data sets can use either fixed or variable length logical records (except for LDS)
- I/O performed by VSAM on behalf of the application always uses a fixed size record called a VSAM control interval (CI). (Note: this does not mean that the application logical records also must also be fixed length – see above). The size of the CI can be chosen when the data set is created.
- VSAM data sets always use record orientation for I/O, just like traditional sequential and partitioned data sets[1]. However, internally, VSAM treats all data as a byte stream using the offset from the beginning of the data set (called the relative byte address or RBA). Doing so makes the address of a record device independent and provides the ability to easily move VSAM data sets from one disk device to another.
- VSAM data sets all have a cluster name. The cluster name provides for "grouping" of components of a VSAM data set so that all components can be referred to using one name in the catalog. This grouping concept applies to a KSDS or ESDS that can have both a data component and an index component. The cluster name when used with the other VSAM record types simply refers to the data component.
- IDCAMS is the utility used to manage (i.e. define, delete, load, copy, print, etc.) VSAM data sets.

There are three ways to access VSAM data sets (but not all four types of VSAM data sets can be accessed using all three techniques):

- Random (or direct) access.
- Sequential access
- Skip sequential access, a combination of random and sequential access

Let's look at each VSAM data set organization in more detail. The primary difference between the four VSAM data set types is the way data is stored and accessed. As we will see, the way an application needs to process data stored in a VSAM data set will determine the type of VSAM organization chosen.

VSAM ESDS

The VSAM ESDS is very similar to a traditional sequential data set so it is a good choice when only sequential access to data will be needed; records are

[1] See ECI No. 6

entered and retrieved in the ESDS in the order written. A good use of an ESDS is to collect data that will never change. For example, a historical log of activity where a new record is entered in the ESDS each time an event occurs. Another example might be the collection of transactions in an ESDS as they occur and the later use of that transaction data set to update a master data set.

ESDS records can be either fixed or variable length. New records are always added at the end of the data set. Records cannot be physically deleted, nor space for a deleted record reused. If a record is no longer needed, it must be logically deleted. This is usually done by marking a field in the record reserved for this purpose. Although an ESDS can be accessed randomly using an RBA, this capability is rarely used.

VSAM RRDS

A VSAM RRDS organization is a good choice when only random processing will be needed to access the data. Data can be randomly accessed using the Relative Record Number (RRN) of the record in the data set. Think of an RRDS as a one dimensional array consisting of slots where records can be stored. Each slot has a number beginning at one for the first slot in the RRDS. This is the RRN for the record. New records can be (randomly) added to an RRDS using the RRN to insert them into an existing slot. Records can be deleted and the space reused. Record length cannot change when a record is updated. RRDS supports both fixed and variable length records, with random access to fixed length records being somewhat faster than random access to variable length records. Sequential access is permitted, with records being returned in RRN order.

VSAM KSDS

A VSAM KSDS is a good choice for an application that needs to randomly access data some of the time, and sequentially access the data at other times. A bank for example, needs to access account records randomly for ATM withdrawals, but will likely want to access accounts sequentially (perhaps by zip code) when printing monthly statements. The z/OS catalog uses a VSAM KSDS to randomly and quickly locate data sets. The VSAM KSDS is the most widely used of the four VSAM data set organizations.

A VSAM KSDS consists of two disk areas: an index area in addition to the data area for the data set itself. A VSAM cluster name refers to both of these areas (whereas with an RRDS and LDS, the cluster name simply refers to the data set itself since they have no index). The KSDS index includes an entry for each CI in the KSDS. Each entry consists of the disk address of the CI and the highest key value found in the CI. While the index is permanently stored on disk, it is usually small enough (even for a very large KSDS) to be brought entirely into virtual storage when a KSDS is being processed. This, and the fact that it is a multi-level index, significantly speeds up searches when doing random processing. Only a portion of the KSDS index needs to be in virtual storage when processing a KSDS sequentially. When a KSDS is processed sequentially, records are returned in key sequence.

Records in a KSDS can be fixed or variable length and are maintained in key order. Records can be updated, but the key field of a record cannot be changed using an update. An update can change the length of a variable length record. Records can be physically deleted and the space made available for reuse. Records must have a unique primary key, but they may also have secondary keys so that records can be processed based on key fields other than the primary key. KSDS records can be processed using a key or by using an RBA. While RBA is more efficient, it puts more of a burden on the application program to keep track of changing RBAs and not all programming languages support access using an RBA. Consequently, record processing using keys is much more common for a KSDS.

The management of free space and the way records are inserted into a KSDS is the one thing that clearly distinguishes a KSDS data set from the other VSAM data sets. When a KSDS is created, it is usually created with free space distributed throughout the

CIs of the KSDS. Each CI contains: data records followed by free space (as specified by the user that creates the KSDS) followed by VSAM control information. When a new record is to be added to the KSDS, it is inserted in key order. To do so, VSAM consults the KSDS index to find out which CI should hold the new record and the record is added in key order (using the free space in the CI) to the other records already in the CI. This may require that some of the existing records be moved to make room in the sequence for the new record as records are always kept in key order sequence in a CI. Occasionally, a CI does not have free space available to contain a new record. In that case, VSAM performs a "CI split". That is, it looks for an "empty" CI within the control area (CA) of the CI that is full, and moves half of the records from the full CI to an "empty" CI to make room for the new record. This of course results in updates to the index and changes the RBA for some records. The CA referred to earlier is simply a collection of contiguous CIs (for example, all the CIs that will fit in a cylinder on disk). The CA is at least a track and at most a cylinder of disk space. The CA is defined by VSAM when the KSDS is created, not by the user that creates the KSDS. If an "empty" CI does not exist within the CA where the record is to be added, VSAM performs a "CA split" and moves some CI's to a new CA to create free space. If there are no remaining CAs in the KSDS, VSAM dynamically acquires more disk storage to create a new CA for the KSDS.

VSAM LDS

A VSAM LDS is a good choice when the application that will use the data manages logical records directly itself. That is, VSAM will treat the LDS data as a stream of bytes and unlike the other VSAM organizations, does not store any control information regarding the logical records (since VSAM does not understand what constitutes a record for the data stored in an LDS). An LDS is processed like an ESDS, with certain restrictions. When an LDS CI is brought into virtual storage for an application, it is up to the application to assign meaning to the CI data. For example, the DB2 database manager uses VSAM LDS data sets to store its relational data. DB2 knows what constitutes a row, column, and table in the relational data within a CI. The VSAM LDS organization is also of benefit when exceptional performance is an application requirement.

VSAM has been enhanced with new features and functions many times over its long history. Two significant recent enhancements are VSAM Record Level Sharing (RLS) and Transactional VSAM (TVS).

VSAM Record Level Sharing (RLS)

VSAM RLS was introduced in the mid-1990s and is designed to be used by Customer Information Control System (CICS) applications accessing VSAM data in a Parallel Sysplex. It allows VSAM data sets to be shared (at the record level), with full update capability and data integrity, between CICS applications running in multiple CICS regions. VSAM RLS was conceived to address some of the short comings of CICS access to VSAM data using a CICS file owing region (FOR) which, in some circumstances, could be a performance bottleneck and/or a single point of failure. When a batch[2] job shares a VSAM data set with a CICS on line region, only read access from the batch job is supported. Transaction support for VSAM RLS is a shared responsibility with CICS providing logging and recovery while VSAM RLS provides record level serialization (locking) and sysplex wide data sharing using the coupling facility (CF).

Transactional VSAM (TVS)

Transactional VSAM builds on and enhances VSAM RLS. While VSAM RLS represents a good first step in terms of enhancing VSAM for Parallel Sysplex data sharing, it does not go far enough in terms of the breadth of shared access – it only applies to CICS applications and even then it only supports read access when on line CICS applications share a VSAM data set with a batch job. Further, it does not support data sharing with non-CICS applications. To

[2] See ECI No. 4

remedy these shortcomings, TVS was developed and delivered in the mid-2000s. TVS builds on the VSAM RLS support by using the RLS locking and data sharing mechanisms. It adds logging using the z/OS logger and transaction recovery by using the z/OS Resource Recovery Services (RRS). TVS thus provides VSAM data set sharing for applications in a Parallel Sysplex with full read/write access and data integrity. Unfortunately, programs run as batch jobs require modifications to be used effectively with TVS. Consequently, TVS is not widely used.

Summary

So there you have it – VSAM provides a variety of organizations for disk data and the services needed to access them which, when taken all together, delivers industry leading capabilities for data stored on disk.

We will explore other aspects of Enterprise Computing in subsequent articles.