

ICL

TRAINING

IDMS DESIGN

N O265- 6

All rights reserved. No part of this document may be reproduced or utilised, in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without permission of ICL, Consultancy and Training Services Division.

This publication is for training purposes only. It should not be regarded as a full specification of any ICL products or services. ICL makes every endeavour to ensure the accuracy of this document but does not accept liability for any errors or omissions.

Training Publications
Issued by ICL, Consultancy and Training Services Division
Beaumont, Old Windsor, Berks

© International Computers Limited, 1987

COURSE TIMETABLE

DAY 1	am	* Database Principles and IDMS
	pm	* IDMS Concepts
	pm	* Logical Data Structures
DAY 2	am	* Physical storage structures
	pm	* (continued) Using the Data
DAY 3	am	* (continued)
	am	* The Run-Time Environment
	pm	* Security and TP
DAY 4	am	* (continued)
	am	* Stages in Database Design
	pm	(continued)
DAY 5	am	* Changing the Database

COURSE SPECIFICATION

Course Code VIDMD

Start day Monday

Finish day Friday

Description

An intensive course for system designers and database specialists from VME installations on how to design, from a given data model, an IDMS or IDMSX database that will implement the required access paths efficiently.

Objectives

To enable course members to:

1. Convert the entities and relationships of a data model into the records and sets of an IDMS(X) database.
2. Tune the database to provide efficient access paths for retrieval and update.
3. Recommend security procedures for application systems using the database.

Please note

1. This course assumes knowledge of systems design.
2. Programmers who will not be involved in design decisions should attend the VME Integrated Database Management Programming course (VIDMP) instead.
3. Previous attendance of the Data Analysis and Data Dictionary course (DADD), though not a prerequisite, is strongly advised.

Contents

	Page
CHAPTER 1 : DATABASE PRINCIPLES, IDMS AND ASSOCIATED DATA MANAGEMENT PRODUCTS	
THE DATABASE APPROACH	1-1
What is IDMS	
What is a Database	
BUILDING AN IDMS SYSTEM	1-4
Identifying requirements	
Getting data under control	1-5
Designing the database	
Implementing the database system	1-6
The job of the designer	
DATA ANALYSIS AND DATA MODELS	1-6
The business level	1-7
The data model	1-9
Pathways through the data model	1-12
INPUT TO GOOD DATABASE DESIGN	1-13
Overall systems plan	
Data models	
Data Administration	1-15
THE DESIGN STAGE - TRANSLATING THE DATA MODEL	1-16
Data Structuring facilities	
Records and sets	
TECHNIQUES SPECIAL TO DATABASE ENVIRONMENTS	1-21
Multiple access paths	
Security	1-22
Insulation from change	1-23
SUMMARY OF FEATURES AND BENEFITS OF IDMS	1-23
DATA MANAGEMENT PRODUCTS	1-27
DDS	1-28
IDMS	
TPMS	
APPLICATION MASTER	1-29
REPORTMASTER	
QUERYMASTER	
QUICKBUILD PATHWAY	1-30
QUICKBUILD WORKBENCH	1-31
PROGRAMMERS WORKBENCH	
PROGRAM MASTER	
CAFS-ISP	1-32
QUICKBUILD AND DATABASE DESIGN	1-32
IDMS tramline recommendations	1-33
CAFS AND DATABASE DESIGN	1-34

CHAPTER 2 : IDMS CONCEPTS

DATA INDEPENDENCE	2-1
Subset views	2-2
Variation of the physical implementation	2-3
COMPONENTS OF AN IDMS SYSTEM	2-4
Logical Detail	
Schema	
Subschema	
Physical Detail	2-5
Storage-schema	
Service-description	
Application programs	
THE RELATIONSHIP BETWEEN IDMS COMPONENTS	2-6

CHAPTER 3 : LOGICAL DATA STRUCTURES

CHAPTER 4 : PHYSICAL STORAGE STRUCTURES

IDMS PHYSICAL STRUCTURES	4-1
Pages	4-2
Definition	
Lines	
Format of a pointer	4-4
Page formats	4-6
Pages in use	4-8
Areas	
Definition	
What are areas for?	4-11
Areas and realms	
Alternative areas	4-12
Areas and files	
Virtual store areas	
Records	4-13
Basic structure	
Fragmentable format	
Re-located records	4-15
Record indexes	
Sets	4-16
Chained sets	
Indexed sets	4-17
RECORD PLACEMENT	4-19
CALC placement	4-20
PAGE DIRECT placement	4-22
DIRECT placement	4-23
VIA placement	4-24
SEQUENTIAL placement	4-28
SYSTEM DEFAULT placement	
Overflow	4-29
STORAGE SCHEMA DEFAULTS	4-30
File defaults	4-31
Area defaults	
Record defaults	
Record identifier	
Placement	4-32
Within area clause	
Minimum root	
Minimum fragment	
SET DEFAULTS	4-33
Mode default	
Pointers default	
Index defaults	
DOCUMENTATION	4-34
Storage diagrams	
The Data Dictionary System	4-37
APPENDIX	4-38
DSDL example	
DDCL example	4-39

CHAPTER 5 : USING THE DATA

THE IDMS SUBSCHEMA	5-1
Realms	
Records	
Sets	5-2
Data manipulation language (DML) verbs	
STRUCTURE OF A COBOL IDMS PROGRAM	5-2
Data division entries	
Procedure division entries	5-3
SUCCESS UNITS	5-3
REQUESTING USE OF THE DATABASE	5-5
The READY verb	
The FINISH verb	5-6
CURRENCY	5-6
How currency indicators change	5-7
The importance of currency indicators	5-9
RETRIEVING DATA FROM THE DATABASE	5-9
Entry point access	5-10
Using logical keys	
Start of a realm	
Navigational access	5-11
Navigation within a set	
Navigational access within a realm	5-13
Retrieval by record order key	5-14
Enquiry DML - summary	5-15
Walking a set	5-16
Realm navigation	5-17
Navigation using more than one set	5-18
UPDATING THE DATABASE	5-19
The STORE verb	
The MODIFY verb	5-20
The ERASE verb	
The CONNECT verb	5-21
The DISCONNECT verb	
SUBSCHEMA CONSTRAINTS	5-22
DATABASE PROCEDURES	5-23
Uses of database procedures	5-24
IMPROVING PROGRAMMING EFFICIENCY	5-25
APPENDIX	5-27
Subschema diagram	5-28
Subschema DDL	
Subschema DDCL	5-29

CHAPTER 6 : THE RUN-TIME ENVIRONMENT

COMPONENTS OF A RUN-TIME PROGRAM	6-1
Application object code	6-2
IDMS DBMS	
Subschema tables	
Service tables	6-3
THE IDMS COMPILING SYSTEM	6-3
Using DDS	
The directory	6-4
Loading the directory	6-5
Loading the database description	6-6
CLUC	6-8
IDMS SERVICES	6-9
Structure and use of a catalogued service	
Running applications in an UNSHARED catalogued service	6-10
Running applications in a SHARED catalogued service	6-11
Uncatalogued services	6-13
UNSHARED uncatalogued services	
SHARED uncatalogued services	
THE IMPLICATIONS OF SHARING	6-15
Success units	
Locking mechanisms	6-16
Syntax of READY	6-17
AREA locks	
PAGE locks	6-18
IDMSX	
Use of locking	6-19
APPENDIX	6-19
SDL coding	
DDCL coding	6-20

CHAPTER 7 : IDMS SECURITY

SUCCESS UNIT	7-1
SECURITY	7-3
RECOVERY	7-4
Type of failure	
Mechanism for resilience	
Dump & restore utilities	7-5
IDMSX	7-6
Central journal	7-6
Rollback	
Rollforward	
Quick before locks (IDMSX)	7-8
Area journals (IDMSX)	7-9
Rollback & rollforward	
Automatic rollback	
Rollback utility	7-10
Rollforward utility	7-11
Delayed update	7-12
IDMSX	
Duplexing (IDMSX)	7-13
Controlled use of IDMS environment	
Database diary	
USING THE IDMS RECOVERY MECHANISMS	7-14
Local failure	
System failure	7-15
File failure	
IDMSX	

CHAPTER 8 : IDMS & TPMS

TP - INTRODUCTION	8-1
Control virtual machine	8-3
AVM pool	
Files used	
Recovery slot file	8-4
Sequence of events in handling a phase	8-4
Structure of a TP success unit	8-5
Components of an AVM	
TP & RECOVERY	8-6
Secure & commit	
TP/IDMS RECOVERY	8-8
Local failure	
System failure	
File failure	8-10

CHAPTER 9 : STAGES IN DATABASE DESIGN

DATABASE DESIGN OVERVIEW	9-1
FIRST STAGE - PRODUCING AN INITIAL DB DESIGN	9-3
The schema	
Mapping down from entities and relationships	
Keys	9-4
Other decisions	9-5
The storage schema	9-5
Placement	
Activity patterns	
Traversals	9-6
Bulk	
Multiplication	9-7
Access to owner	9-8
Example of traversals	9-8
Skeleton activity lists	
Detailed activity lists	9-9
Detailed activity lists in priority sequence	9-10
Further considerations	9-13
Other decisions	9-14
Areas	
Sets	9-15
Page size	
Summary	9-16
SECOND STAGE - ASSESSING PERFORMANCE	9-17
Media requirements for standard IDMS	
Record lengths	
Page size	9-18
Packing density of the area	9-20
Calculating media size	
Media requirements for IDMSX	9-23
Variable page size	
Indexes	9-24
Index size	
Index record size	9-25
Index record placement	9-27
Calculating index media requirements	
Estimating database transfers	9-29
Buffer control	
DML verbs	
Retrieval verbs	
Update verbs	9-31
The accesses for a success unit	9-35
Journalising	
Other considerations	9-36
Path lengths	
Program size	
Summary	9-38

THIRD STAGE - IMPROVING PERFORMANCE AND TUNING	9-39
Sets	
Set mapping	
Set design options	9-42
Pointers	
Set order	
Set membership	
Records	9-43
Record mapping	
Fragmentable records	9-45
Record design options	9-46
Placement	
Areas	9-48
Indexes	
Loading	9-49
Reverse loading of records	
Pre-sorting of CALC records	9-50
Summary	9-50
MONITORING PERFORMANCE	9-51
IDMS statistics	
IDMSDUMP statistics	9-52
Service statistics	9-53
Other methods	
Monitoring the live database	

APPENDIX 1 - TECHNICAL DESIGN CHECKLIST	9-54
Decisions for each record type	
Logical decisions	
Content	
Keys	
Insertion/retention class	
Physical decisions	
Placement	
Which area?	9-55
Decisions for each set type	
Logical decisions	
Is set required?	
Owner record type	
Member record type	
Set order	
Physical decisions	
Chained or indexed sets (IDMSX)	
Pointers (chained sets)	
Schema & subschema decisions	
DB procedures in schema	
What subschemas?	
Individual program decisions	
DML navigation strategy	
ERASE	
DML restrictions	9-57
Opening realms	
Size of success units	
Areas	
Mapping to files	
Area size	
Page size	
Standard IDMS	
IDMSX	
Buffers & SDL	
Program independence of storage schema decisions	
Initial database loading	
Names - rules and standards	
APPENDIX 2 - IDMS SIZING FORMS	9-59
Database record population form	9-60
Set population form	9-61
Transaction loading form	9-62
Batch workload form	9-63
Transaction sizing form	9-64

CHAPTER 10 : CHANGING THE DATABASE

WHY WILL CHANGES BE NEEDED	10-1
WHAT NEEDS CHANGING	10-2
THE PROCESS OF CHANGE	10-3
General principles	10-4
Increasing area size	10-5
Adding pages to an area	
Page expansion	10-6
Changing records in place (restructure)	10-6
Changing records by unload & reload(reorganise)	10-7
THE IMPACT OF CHANGE ON EXISTING PROGRAMS	10-8

CHAPTER 11 : TRENDS IN DATA MANAGEMENT

INTRODUCTION	11-1
REPORTMASTER	11-2
QUERYMASTER	
APPLICATION MASTER	11-3
CAFS-ISP	
CAFS AND DATABASE DESIGN	11-4

CHAPTER 1

**DATABASE PRINCIPLES, IDMS
AND ASSOCIATED DATA MANAGEMENT PRODUCTS**

THE DATABASE APPROACH

Introduction

IDMS stands for Integrated Database Management System.

This chapter discusses the concepts and benefits of a storage and retrieval system based on database principles and surveys the various stages involved in establishing an IDMS system. These stages include understanding the logical structure of the data and then the physical implementation of that structure. In short, the proper management of data.

What is IDMS?

IDMS is software that allows a database, structured according to the internationally accepted CODASYL rules, to be created, interrogated and maintained.

There are different versions of IDMS for VME and for 24 bit environments.

This course concentrates on the VME environment where besides standard IDMS there is an extended version, IDMS-X.

What is a Database

A Database system allows data to be captured once only and SHARED by all the systems and people who use it.

When data is common to several applications, collecting and storing it separately for each - as happens when systems are designed in isolation - is obviously wasteful. Moreover the various "application files" are likely to differ in format and more seriously, be inconsistent in value through being updated at different times. Inconsistencies can also arise from each department concentrating on their own narrow viewpoint which may not coincide with the correct view of the organisation - eg using different reference codes for the same item.

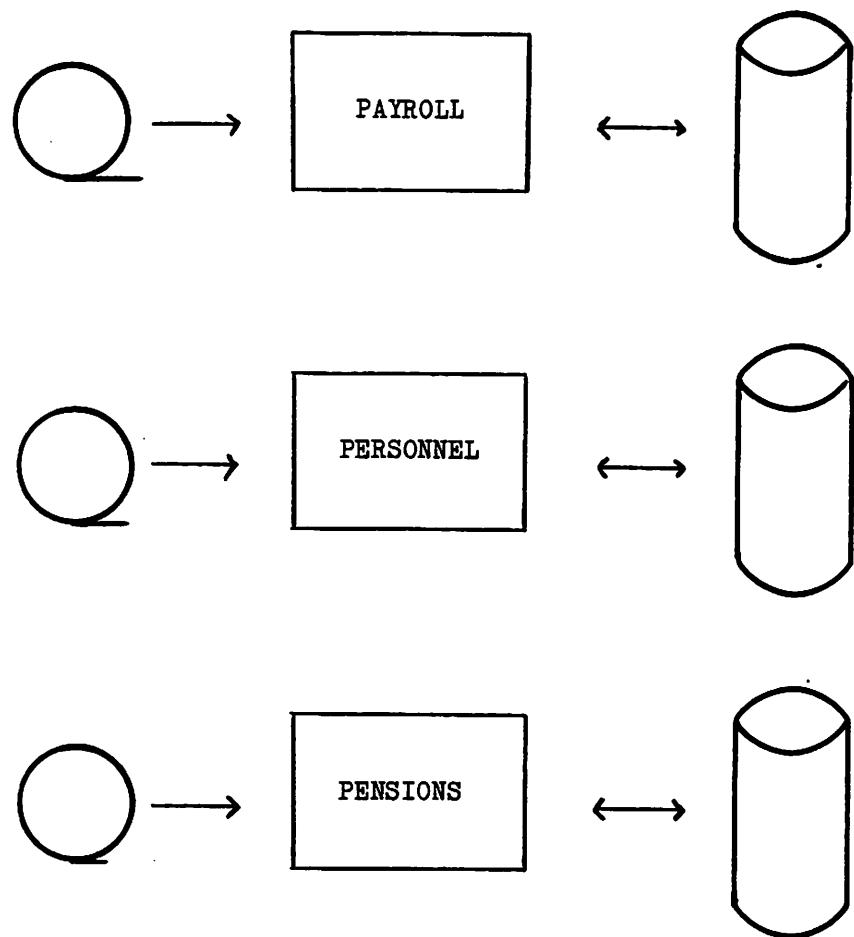


Figure 1 Application designed in isolation

Collecting data just once and letting applications share it is far more sensible. That is what database management systems, such as IDMS, enable you to do.

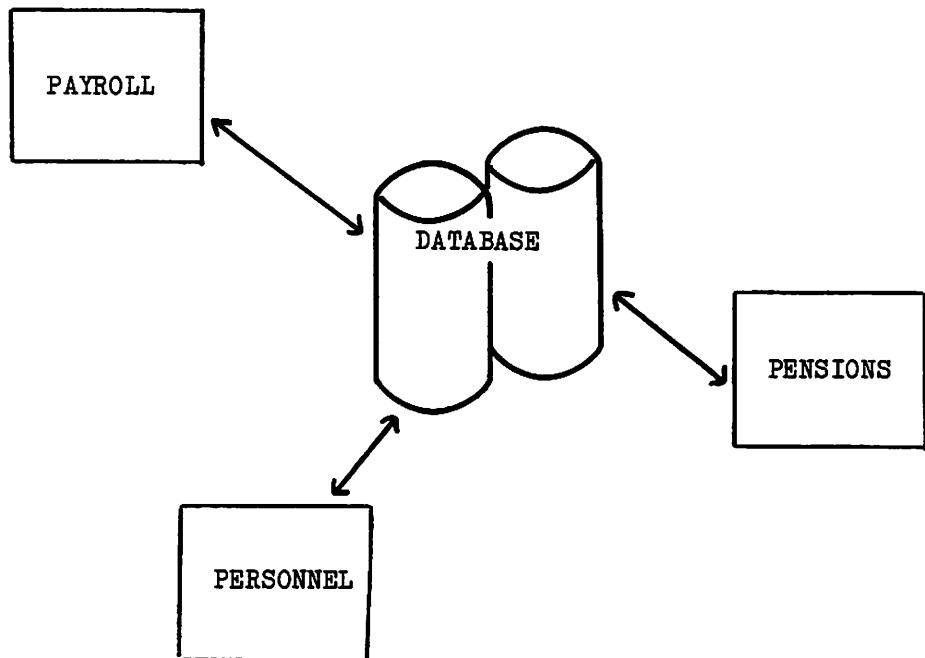


Figure 2 Applications sharing data

A database should reflect the view that data is a corporate asset. There are two aspects to this. Firstly, data belongs to the organisation as a whole, not to individual departments. Secondly, to be of value data must be correct, unambiguous and accessible. A single coherent source of accurate data provides a sound basis for decision making.

Although the contents of the database should represent the organisation's viewpoint this does not imply that all aspects of the business will or should be included in the database. In many cases a corporate database is not practical. It is more realistic to consider the systems relating to an individual business area viewed, of course, in the context of the whole organisation.

Building an IDMS System

The stages in setting up and using a database are:

- identify requirements
- get data under control
- design the database
- implement the database system.

Before concentrating on the design stage we shall look briefly at the whole process.

Identifying Requirements

Ideally this stage begins with an information study, the object of which is to gain an overall picture of what data the organisation possesses and how the data is used. In practice, the step is often omitted for political or financial reasons, or lack of time.

The second step in identifying requirements is to produce an overall system plan specifying the sequence in which database applications are to be implemented. For phased implementation of a database - and few organisations opt for the all-at-once approach - this step is essential.

Getting Data Under Control

There are three aspects to getting and keeping the data under control - data analysis, documentation and data administration.

The overall systems plan defines a subset of the organisation's activities for which the database will be constructed or extended. We will call this the 'area of interest'. The aim of data analysis is to understand the nature and use of data in the area of interest from the organisation's point of view. It is not concerned with the computer implementation. The results of data analysis should be presented as a **data model** describing how the data is structured and used.

The second aspect of controlling data is to document all the information about the data gleaned during the analysis process. The most efficient way of doing this is to hold the information on the computer using the **Data Dictionary System** (DDS). Besides storing descriptions of data and the processes that use it, DDS provides reporting, interrogation and "COBOL library" facilities.

The third aspect of controlling data is to set up a data administration function. This is a person or group of people responsible for controlling shared data - its collection, rationalisation (when users have conflicting views), accuracy and privacy.

All three aspects of getting data under control are covered in detail on other courses.

Designing the Database

The object of this stage is to translate the results of data analysis into computer terms. That is, to represent the data as records connected together in ways that reflect the structures and accessing requirements defined in the data model.

Besides the output from data analysis, you need access to the overall systems plan so that you can bear future requirements in mind.

Again, you can record in the data dictionary, the results of the design phase - the format and structure of data in the database. If you are adding to an existing database, you can use DDS to discover whether any of the relevant data is already there and, if so, its format and which other applications use it.

Implementing the Database System

Implementing the database consists of setting up all the necessary files, collecting and validating the data and loading it into the database. The application programs which access the database will include the database accessing commands, known collectively as DML (Data Manipulation Language) verbs, incorporated in otherwise normal COBOL programs.

The Job of the Designer

Is to represent the understanding of how the business works, at the computer level, as efficiently as possible.

The designer, therefore requires:

- full documentation analysing all major aspects of how the business works, ie the data model
- the use of robust software that will allow:
 - . representation of the complex real world data structures
 - . sharing of data in an efficient and secure manner.

Considering the job of the designer in the context of building an IDMS system, his task is to take over where the analysts left off. He must then be familiar enough with the process of Data Analysis to gain a full understanding of what is contained in the Data Model.

Data Analysis and Data Models

Definition

A genuine disciplined approach to Understanding the NATURE and USE of the data required in information systems - from an organisational viewpoint.

Data Analysis is concerned with the Business level of understanding (also known as Conceptual level) whereas database design is concerned with the Implementation level on the Computer.

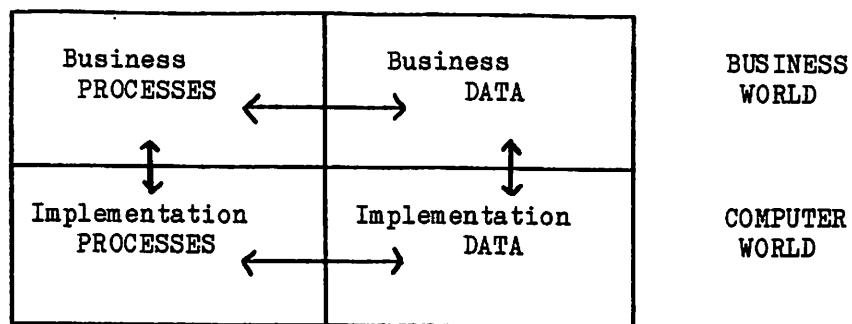


Figure 3

The Business Level

The Nature of data is an aspect that has been traditionally ignored but its understanding is fundamental to good DB systems. It can be expressed using the following notation.

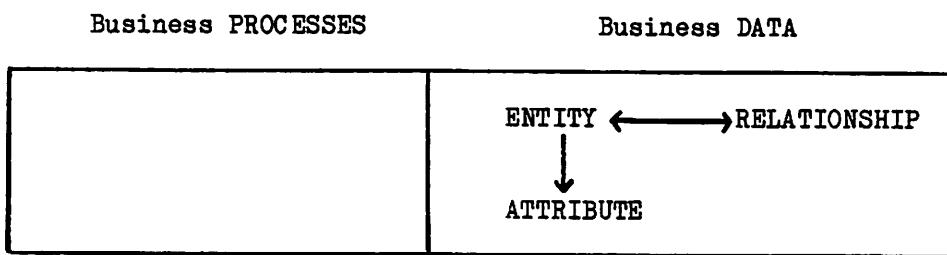


Figure 4

ENTITY - an object, person or place of interest to the enterprise
eg a man, a part, an order.
These are examples of Entity Types as opposed to Entity Occurrences, examples of which are John Smith, 3" widget, order XZ123.

RELATIONSHIP - a connection between entities. Relationships are of interest because:

- they are an essential part of the reality being modelled
- when implemented they become access paths eg from customer to all his orders or vice versa.

The nature of a relationship can be identified as:

- one to one
- one to many
- many to many.

ATTRIBUTE - a data element identifying or describing an entity eg a man's age, part number, order quantity etc.

How do we know that our data structure is accurate and complete? The Use of the Data must be investigated - a functional analysis - the results of which will be tested against the structure to ensure that the structure can support all the processing requirements.

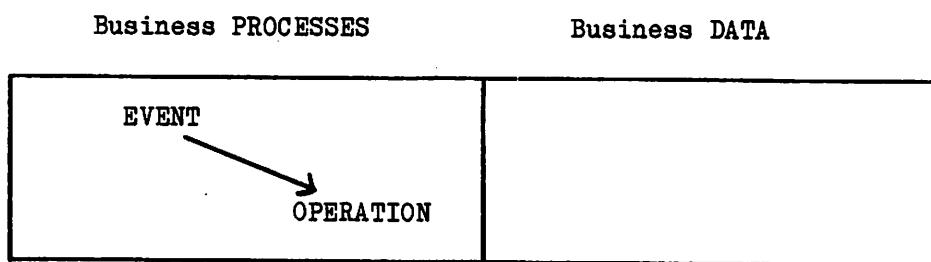


Figure 5

EVENT - an occurrence in the outside world which triggers an operation,
eg Agent sells insurance policy, Customer places an order,
end of month etc.

OPERATION - a process of interest to the enterprise,
eg Issuing a policy, recording an order, producing monthly
accounts etc.

The Data Model

All this information is used to build, validate and extend the Data Model.

Definition

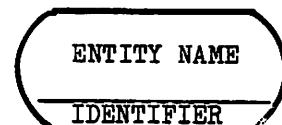
A Data Model is a representation of reality describing the Entity types with associated Attributes in an area of interest plus the Relationships that link Entities together. The Events that occur in the area of interest plus the Operations that they trigger, although not shown in any diagrammatic representation, are for all practical purposes, part of the Data Model as well.

It is a business view of the data structures necessary to meet information requirements.

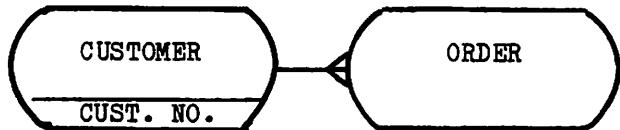
It is the end product of Data Analysis.

There is a standard pictorial representation for Data Models, known as soft-box models. It should be noted that not all the information constituting the Data Model is shown in the soft-box model.

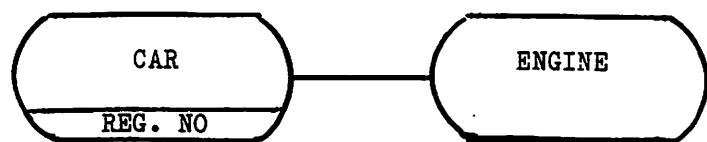
An ENTITY is represented by a 'SOFT-BOX' which optionally includes identifying attribute(s)



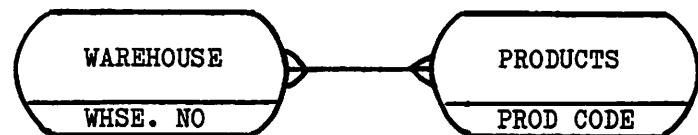
RELATIONSHIPS are represented by a line linking two entity boxes together with 'CROWS-FEET' defining the nature of the relationship.



one : many



one : one



many : many

Each relationship is given a name and according to BCS conventions this name is meaningful when read in the one to many direction. If the name is to be read in the many to one direction instead, it is enclosed in brackets.

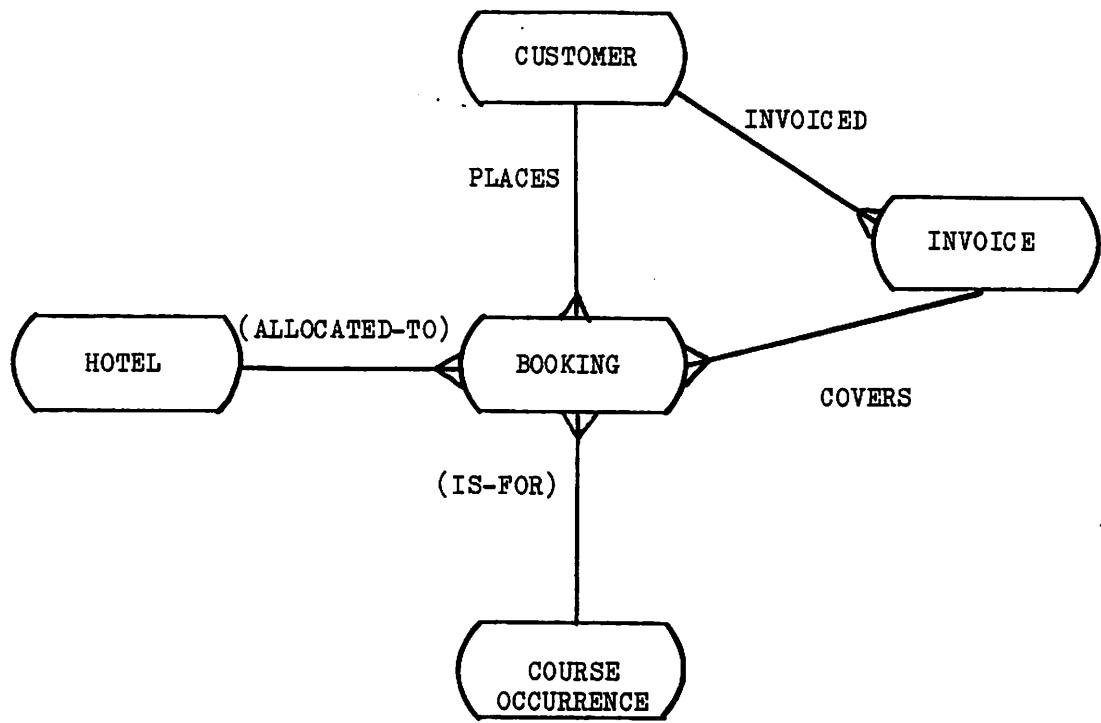


Figure 6 Data Model of a Course Reservation System

Pathways Through the Data Model

The Data Model must be able to support the operations of the business. Operations will tend to map to user transactions in the eventual DP system.

Each operation will need to enter the structure at a particular Entity type using a key, and then move on to other Entities either using Relationships as access paths or by entering the structure again from outside.

For example suppose in the ICL E&T environment there is the Event "Customer wishes to book one or more places on one or more courses". The logic of the triggered operation might be:

- enter at relevant Course Occurrence and if spare places create a BOOKING and link to relevant COURSE, CUSTOMER and HOTEL
- repeat for each course place required
- create an INVOICE and link to CUSTOMER and BOOKING.

Figure 7 - shows the pathway that could be taken through the Data Model to do this.

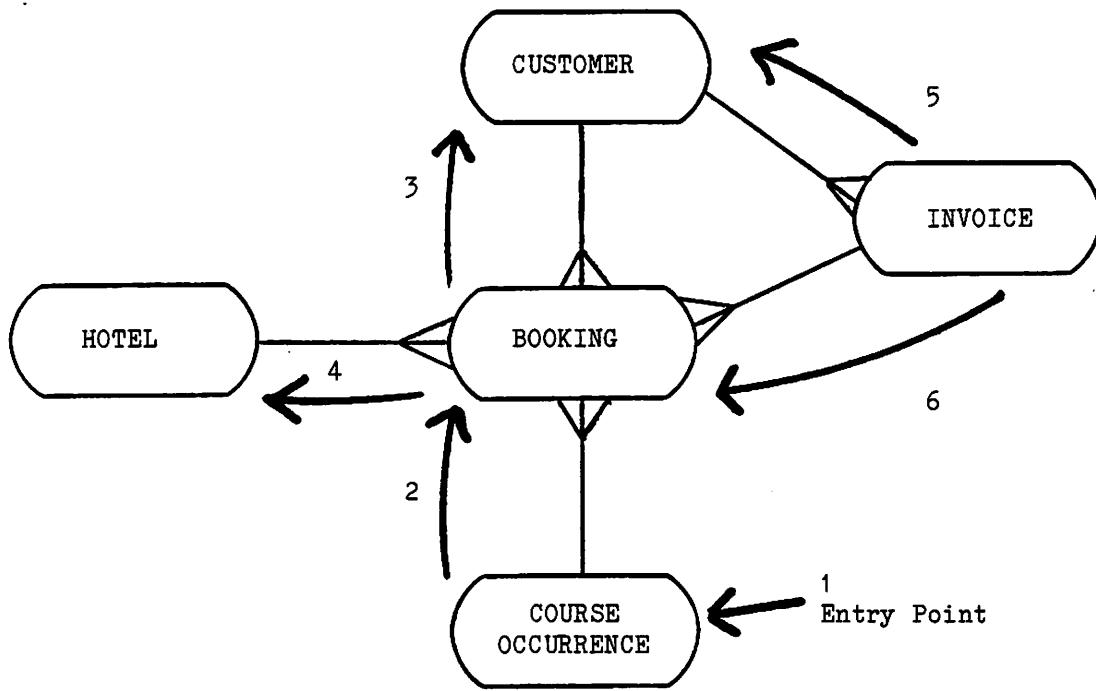


Figure 7

For each operation ask:

"Can the existing data model support the necessary steps? If not, what changes are needed to allow the processing to be done neatly and quickly?"

This so-called "Event-Driven" approach would be very tedious if used on every Event that could occur. So in practice only a limited subset comprising the most important Events is usually considered and used to check the Data Model.

Input to Good Database Design

The Data Model provides objective guidance for implementation level decisions. It guides Database design. For completeness the designer requires the following information.

Overall Systems Plan

- including details of the implementation sequence for the various applications where appropriate and the associated timescales. This will enable the designer to structure the database with awareness of future requirements, thereby reducing the impact of change.

Data Models

- an Outline data model co-ordinating a number of detailed data models relating to various application areas which are to be implemented. Some phases may already have been implemented and details of those relevant data areas should be included.

Each detailed data model should be presented as an overview in soft-box form and in detailed text form in the DDS. The minimum information in the DDS should comprise:

For each Entity:

- a unique name
- description of the entity
- the identifying attribute(s)
- the descriptive attribute(s)
- number of occurrences and growth rate
- average and maximum life of an occurrence
- any privacy rules.

For each Attribute:

- name
- description
- the format
- range of valid values, where applicable
- units (eg £ or pence).

For each Relationship:

- name
- description
- the min, average, max of membership
(eg 1 : m, how many is many?).

For each Event:

- name
- description
- frequency.

A
*!

For each Operation:

- name
- description
- event(s) initiating the operation
- average and peak frequency
- expected response time for execution of operation
- the processing logic of the operation including
 - the initial input data
 - the entry points to the structure with access keys used
 - relationships used
 - processing on each entity type accessed
- any factors which may place a priority on the operation.

Clearly it is essential to DOCUMENT the Data Model as it is developed. The DATA DICTIONARY is a software tool that can be used as an aid in the documentation, design implementation and maintenance of information systems.

Data Administration

It is of utmost importance that the data in terms of collection, rationalisation, continued integrity and access is kept under tight control so that the right users can get to the right data at the right time. The overall control is the major task of the Data Administration function.

A formal definition of DATA ADMINISTRATION states:

The function of data administration is to control the data environment of an installation. The administration must ensure that the data environment can supply the data requirements of all users as economically and as quickly as possible whilst maintaining the integrity and privacy of the data.

In practise, it is possible to implement a database without any such function. But this will only work while the systems and data remain within departmental boundaries. Experienced users believe it imperative to establish an effective data administration function at a very early stage of the database development process.

THE DESIGN STAGE - TRANSLATING THE DATA MODEL

Having clarified the understanding of the business functions of the organisation, it is important that the database software allows that understanding to be represented in an efficient way. Therefore IDMS must offer data structuring facilities allowing direct translation of the Data Model into computer terms.

Database was earlier defined as allowing the SHARING of DATA between many applications/users. If this concept is to be achieved then the database software must provide:

- multiple access paths to the data
 - many users require many pathways through the data structure
- high levels of security
 - as all the eggs are in one basket
- insulation from change
 - Data Independence.

DATA STRUCTURING FACILITIES

The Data Model in terms of ENTITIES, ATTRIBUTES and RELATIONSHIPS must be mapped down to a data structure which will retain the 'truth' of the data model.

An ENTITY maps down to a RECORD

The ATTRIBUTES of an entity map down to the FIELDS of a record

How can RELATIONSHIPS be implemented?

Consider a conventional file - eg a Personnel file organised as an indexed sequential file comprising EMPLOYEE records structured on Personnel Number.

There will be two ways of accessing the file:

- looking at all the EMPLOYEE records in turn
- entering at a particular EMPLOYEE record knowing his personnel number.

Personnel number	Name	Department	Grade
123	James	A	1
124	Brenda	B	3
125	Harry	A	2
126	Sue	B	2
127	Dave	C	6
128	Bill	A	1

Figure 8 The Personnel File

In reality we often require more flexibility in access methods, for example allowing access to all employees of a particular department or particular grade. Actually, what we are saying is that we want to access the participants in a relationship, ie the relationship becomes an access path.

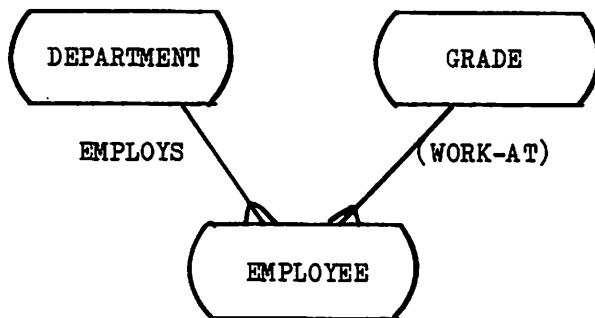
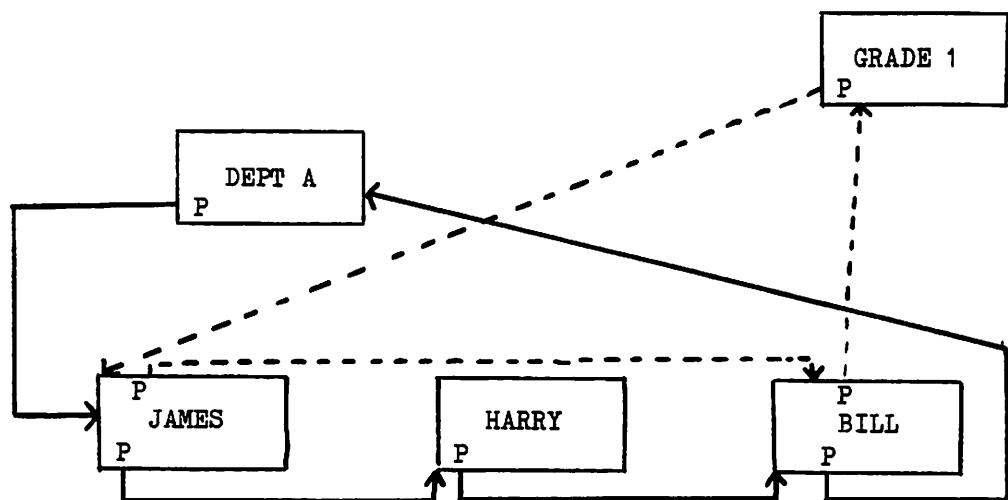


Figure 9 The Business Level Representation

This can be implemented as:

Record occurrences "connected" by pointers, forming an IDMS SET. Each SET is an implementation of a RELATIONSHIP.



Alternatively the SET can be implemented as:

REC POINTS TO REC

Figure 10 The Chained Implementation

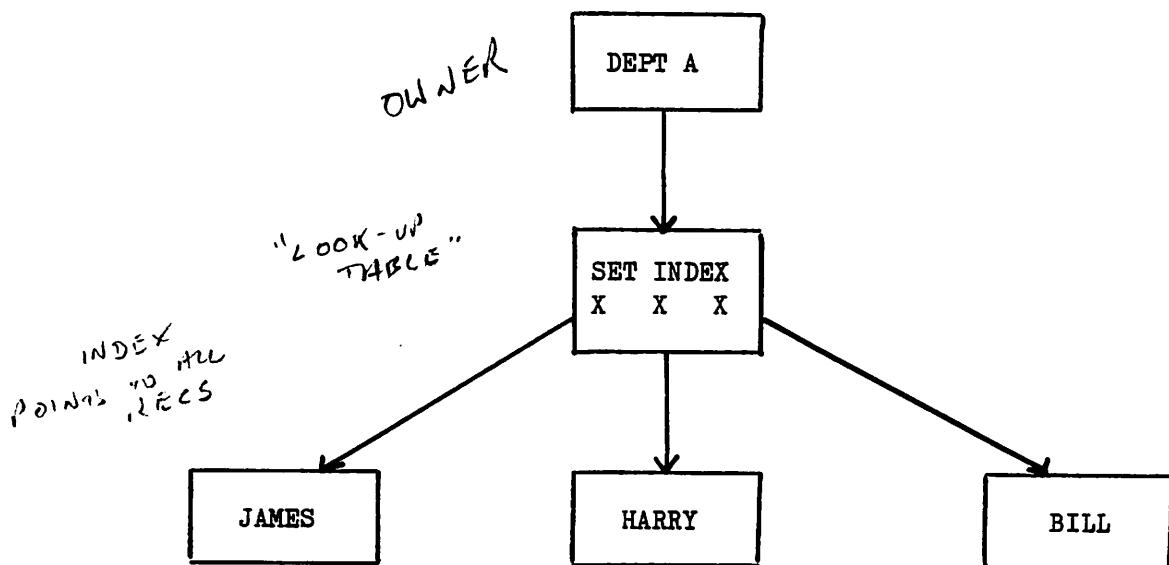


Figure 11 The Set Index

where the owning DEPARTMENT record points to an index, which contains pointers to all the member records of the SET.

Independent of the underlying implementation method, SETS and their associated records are documented as shown in the following example.

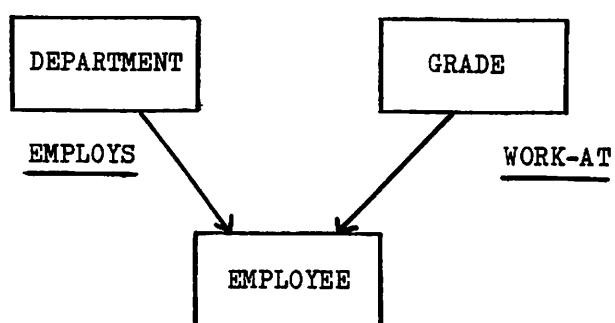


Figure 12 Example computer level representation

This can be seen as a direct translation of the example Data Model shown in Figure 9.

The Data Model in Figure 7 will map down to the following Records and Sets.

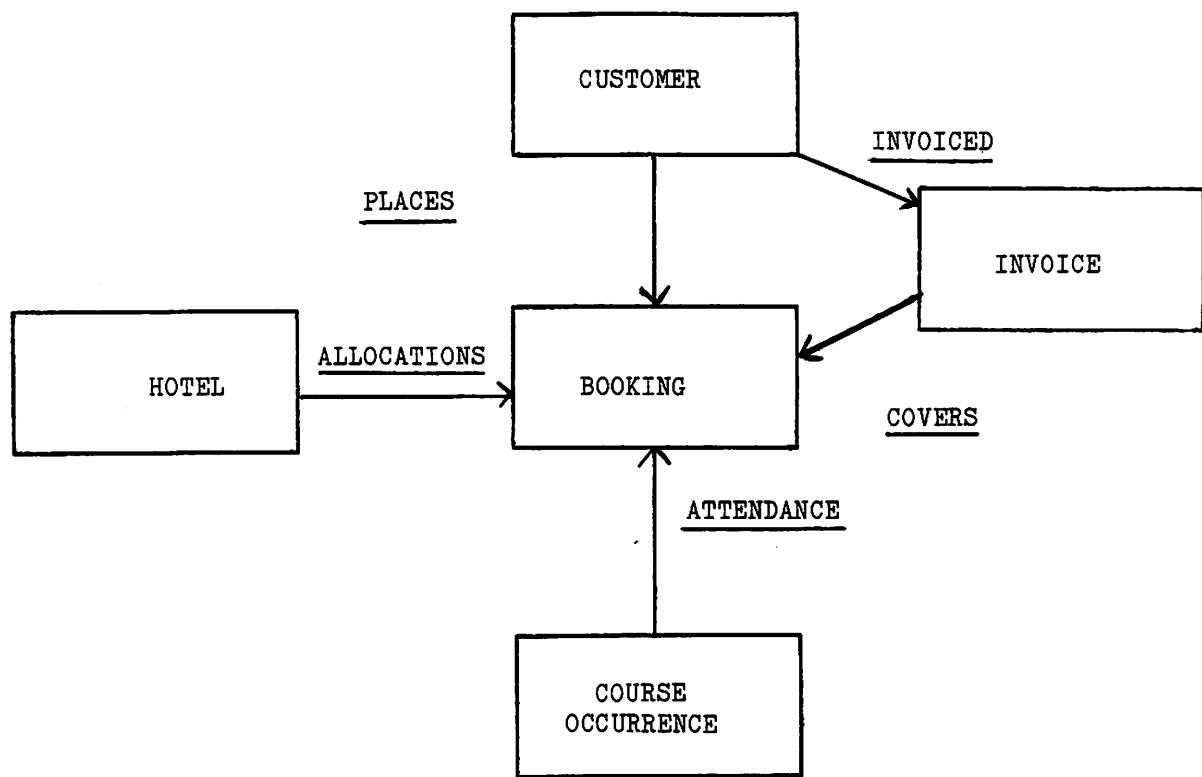


Figure 13 The computer level representation of the Data Structure

TECHNIQUES SPECIAL TO DATABASE ENVIRONMENTS

Multiple Access Paths

By being able to represent real world data structures at the computer level, the Data Structuring facilities in IDMS offer several mechanisms for accessing data including:

- The ability to access occurrences of different record types directly as **ENTRY POINTS** if the identifying key is known.
eg retrieve a CUSTOMER record on the basis of its key - CUSTOMER NUMBER. IDMS provides 2 physical mechanisms to allow such access - Address Generation (CALC records) and Record Indexes (IDMSX only)
- The ability to **NAVIGATE** from the entry point to many other records linked by the **SET** mechanism.
eg - enter at COURSE OCCURRENCE (being defined as an entry point) then by NAVIGATING the SET, access all the BOOKINGS associated with that COURSE
- The ability to serially **SCAN** all records or just records of a given type in the database.
eg - all the CUSTOMER records.

The number of possible navigation paths through IDMS data structures is enormous. As a result the data is readily available to users and is well suited to Transaction Processing systems.

The work done at the Data Analysis stage on the required pathways through the structure allows proper use to be made of these powerful IDMS Navigation mechanisms. During the translation of the Data Model down to IDMS data structures, the overall effect of these pathways can be viewed as a series of access requirements to each record type. These access requirements when put into an "Activity list" for the record type in priority sequence, provide objective guidance on how to use the IDMS data structuring facilities eg:

- should there be one or more Entry Point mechanisms for the record type and on what keys?
- should the record type occurrences be "clustered" into the same disc block as an Owner Record and if so, which Owner record?

and so on.

Security

A database system implies sharing maybe just 1 copy of data between different users, perhaps at the same time. Because of this, the problems of security are much greater than in a conventional environment.

Security can be defined as:

- resilience (the ability to recover)
- integrity (ensuring accuracy and consistency)
- privacy (restrictions on sight or use)

These divisions are remembered easily as RIP.

Let us look at each in a little more detail.

When all the installation's data eggs are in one basket, and they break, you have to be able to recover the situation as quickly as possible. This is resilience. It may mean recovering from a localised failure in one application program or at the other end of the scale, recovering from system or media failure eg disc head crash. Naturally, IDMS includes a comprehensive set of recovery procedures.

Another danger is the risk of an incorrect database due to concurrent on line updating. In other words a loss of integrity. If two users want the same unit of data, one should be forced to wait (be locked out) until the other has finished. IDMS provides locking facilities to guard against this loss of integrity. In theory, these facilities could in turn produce a deadlock situation. For example, say two programs both need records A and B to continue, and each program has locked one of the records. IDMS has in built checks which prevents the situation of total deadlock.

Although data is shared between users in a database system, you may want to restrict the details that some of these users can see. Not every one should see the salary field in a personnel record. You may want to restrict the processing that some users can do. For example, very few users are allowed to update a salary field. Both these requirements would be achieved using the in built privacy mechanisms.

Insulation from Change

Data independence implies that there is an insulating barrier between a program and its data. This means that changes to that data do not automatically require the existing program to be rewritten and recompiled.

SUMMARY OF FEATURES AND BENEFITS OF IDMS

Powerful Data Structuring

- accurate mapping of the Data Model
- better support to integrated systems and therefore development time and costs reduced
- redundant data eliminated leading to savings in store
- multiple access paths so rapid and easy access to information
- structure lends itself to on-line processing.

Security

- safe and private data
- consistent information available.

Insulation from Change

- reduced maintenance therefore increased programmer productivity
- modification of data descriptions and structures is made easier so DP can now be more responsive to business changes.

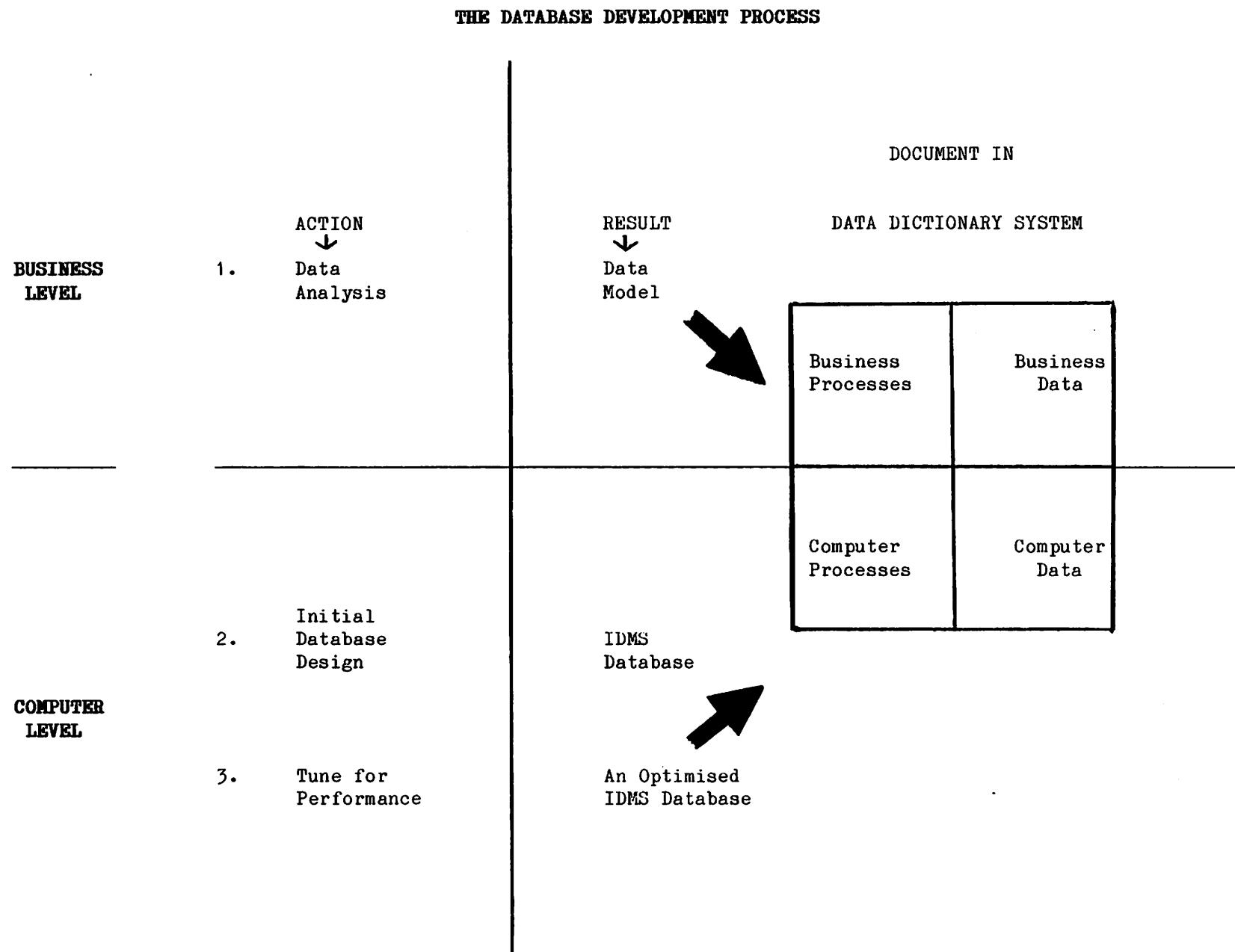
Easy Programming

- central data description allowing automatic production of data definitions.

Reliable

- a well tested product with many users.

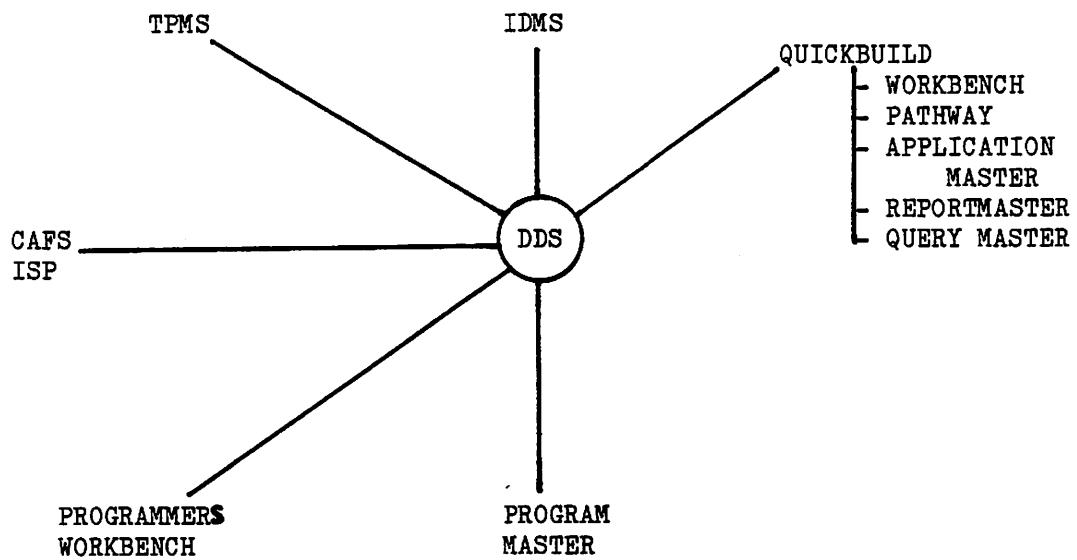
Clearly IDMS offers considerable benefits, but the maximum effects will only be felt provided the implementation and use have been properly planned. This brings us back to the overall database development process, summarised overleaf.



DATA MANAGEMENT PRODUCTS

ICL's data management products have the data dictionary system (DDS) at their heart.

Some of the products which interface with DDS are shown below:



DDS

The results of business data analysis (operations, events, entities, relationships and attributes) may be fully documented in DDS.

Computer processes (systems, modules etc.) and data (files, records, sets, schema's etc) may also be documented in DDS.

The links between the business and computer worlds may also be fully documented.

It is possible, for DDS to hold a complete, up-to-date, view of the business world and the computer world which supports it.

QuickBuild products (see below) allow applications to be fully defined within DDS and processed to produce object code.

COBOL applications may copy screen definitions and record and file descriptions from DDS.

By carefully controlling the contents of DDS it is possible to control the use of data within the computer world.

It is also possible to predict the impact which, say, changes in the business world will have upon the computer world by making simple enquiries of DDS.

IDMS

It is possible to fully define an IDMS database within DDS.

Service tables and subschema tables may be compiled from DDS (more information on this is given later in the handout).

Once the database has been defined in DDS, QuickBuild facilities may be used to develop applications to access the database.

TPMS

It is possible to fully define a TPMS transaction processing service within DDS.

The necessary TP service module and screen modules may be compiled from DDS.

Application Master applications may then be developed to access an IDMS database from within such a TPMS service.

APPLICATION MASTER

Allows the rapid development of on-line applications to run in a TPMS service.

The applications may read/update an IDMS database.

Processing requirements etc are defined within DDS.

Future releases of AM will include Report Master facilities and allow the production of batch applications which may update an IDMS database.

REPORT MASTER

Allows the rapid development of batch applications to extract data from or produce printed reports of data from an IDMS database or a conventional file.

Processing requirements etc are specified within DDS.

The facilities of RM will be included in future releases of Application Master.

QUERY MASTER

Allows end-users to make enquiries of an IDMS database or conventional file from a MAC terminal.

The data structures which are available to the end-user are defined within DDS.

QUICKBUILD PATHWAY

Provides a screen-driven interface to the DDS to harness the use of the QuickBuild product range.

Facilities are provided to allow:

- the specification of a simple business model (entities, attributes, relationships, operations and events) - this may be automatically converted to the records, sets, schema etc. of an IDMS database
- the specification of an IDMSX database
- the specification of a TPMS service
- the specification of AM dialogues, exchanges components and screens
- the specification of RM report-programs
- the specification of QM end-users and end-user-views
- the generation of IDMSX & TPMS services
- the setting up of the files needed by an IDMSX database
- the compilation and testing of AM and RM applications
- the compilation of QM End-user-views
- error and report viewing
- on-line HELP facilities
- the setting up and maintenance of a DDS.

Future releases of QBP will improve the product in terms of:

- supporting the next round of releases in the QB product set including WorkBench products
- better usability
- a system generation option which will provide the user with a default application for the creation, deletion and enquiry of database records.
- performance improvements in DDS accesses.

QUICKBUILD WORKBENCH

This will use the high resolution graphics capability of DRS 300 to draw and develop QB systems consisting of entity models, data flow diagrams and AM structure diagrams.

This information can be transmitted to DDS and QBP may then be used to develop a fully working system.

PROGRAMMERS WORKBENCH

This will provide the programmer with the ability to:

- create program designs using Jackson structure techniques in pictorial format
- generate source code from the diagrams
- test the resulting code
- perform interactive syntax checking with an advanced COBOL editor.

The WorkBench will interact with Program Master on the mainframe for the complete development, testing and maintenance of COBOL programs.

PROGRAM MASTER

Provides an environment in which COBOL programs may be developed and tested without any in-depth knowledge of VME.

Future releases of PM will improve usability and remove current limitations.

The Programmers WorkBench will further increase the benefits associated with the use of PM.

CAFS-ISP

CAFS hardware and software allow extremely rapid file or area scans - searching for data which meets specified selection criteria. Fuzzy matching and complex conditional statements are allowed.

The scan process is handled by disc controllers etc. Only those records which meet the selection criteria are passed on to applications. This means that processor time is only being used where "hit" records are located - giving potentially immense savings.

The speed of a CAFS scan is limited by the rotational speed of the disc.

Current scanning rates fall in the range of 1-2 megabytes per second (depending on the type of disc drive being used).

Hence, CAFS scans are likely to give enormous savings in processor time and "real" elapsed time where the 'hit rate' is low.

Since CAFS is a "retrieval engine" it cannot update either conventional files or IDMS databases.

When used on an IDMS database CAFS does not set currencies and ignores IDMS locking mechanisms - the speed of a CAFS search would be drastically reduced if this were not the case.

However, in an IDMS environment, CAFS can pass on the database keys of hit records to applications which can then retrieve the records by conventional methods.

QUICKBUILD AND DATABASE DESIGN

QUICKBUILD PATHWAY does not, of itself, make use of all of the available facilities of IDMS.

Rather, it handles those which are most useful.

The subset of IDMS facilities which are used are sometimes known as "tramline" facilities.

The tramlines fit into well established "good" methods of database design.

In **any** environment (QuickBuild or "conventional") an excellent general rule is "do not use non-tramline facilities". The use of other IDMS facilities should only be considered if they are needed in order to tune the database design when no other way is possible.

Some of the major tramline recommendations are:

1. IDMSX (rather than basic IDMS) should be used.
2. Each IDMSX area should be mapped onto one VME file.
3. For each record:
 - do not use OCCURS or OCCURS DEPENDING ON clauses
 - use a non-fragmentable format
 - use only CALC or VIA placement
 - repeat owner key fields within member records
 - keep multi-part keys contiguous
4. For each set:
 - implement sorted sets as indexed sets
 - implement all other set orders as chained sets with NEXT, PRIOR and OWNER pointers
 - do not use multi-member sets

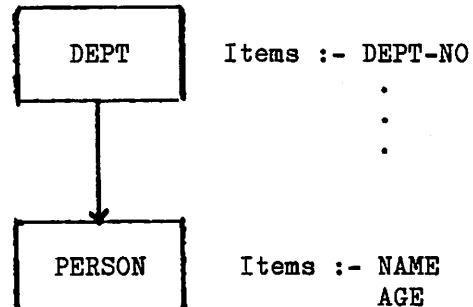
Each of these recommendations is mentioned again at appropriate positions in the handout.

CAPS AND DATABASE DESIGN

CAPS cannot follow pointers. So:

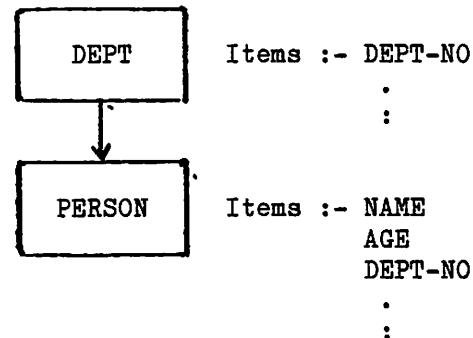
- a) Records which need to be accessed by CAPS scans must not be fragmented or relocated.
- b) CAPS cannot follow set pointers.

Consider:



A single CAPS scan could not locate, say, all of the people who are aged between 20-30 and work for department 4 or 5.

If the design were altered to:

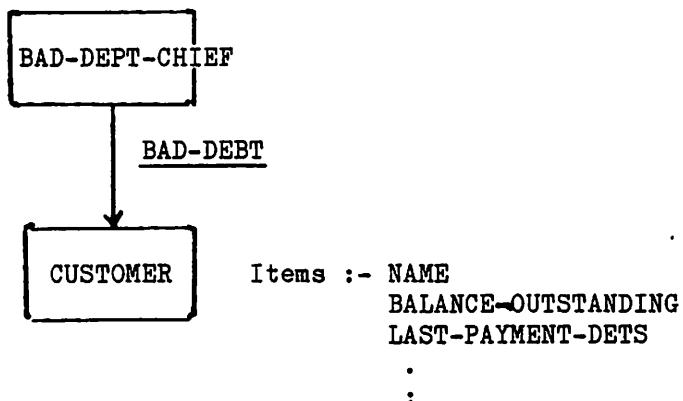


then the above enquiry would be possible in a single scan.

Hence, where CAPS access to data is important, the designer is likely to repeat some data items (especially key fields) down hierachic structures.

- c) Some records (especially 'super-chiefs') are likely to become unnecessary.

For instance:



The BAD-DEBT set could be dispensed with - a CAFS scan of CUSTOMER records could produce details of those who have owed a lot of money for a long time - or a BAD-DEBT-INDICATOR could be included in each CUSTOMER record.

The tendency is, therefore, to reduce the number of record and set types within the database - since CAFS makes the concept of AREA scans a viable proposition.

- d) Where any type of area scan is likely to be of use, it generally makes sense to keep the area as small as possible.
- e) CAFS does not "like" signed packed decimal (COMP-3) fields. They should be avoided.

These ideas are discussed more fully at appropriate points later.

CHAPTER 2

IDMS CONCEPTS

NO265-6

INTRODUCTION

This chapter considers the concepts of IDMS, the technical terms used to refer to them and the way these concepts fit together.

Setting up an Application system using IDMS involves the following tasks:

- definition of the data structures in the whole Database
- definition of the subset of data available to each individual user
- specification of the physical storage details of the data
- writing and testing the Application programs.

There are appropriate IDMS languages used at each stage. Besides defining the systems or programming decisions to the software, some of these languages provide the feature known as Data Independence.

DATA INDEPENDENCE

Introduction

This feature consists of two facilities:

- a program can be restricted to a subset view of the Database. It is then protected from any changes made to the parts of the Database it cannot see
- the logical design of the Database can be maintained across different physical implementations.

Subset Views

An IDMS Application program uses a defined subset of the Database. The structure of any part of the data not defined in that subset can be changed, without the program being affected. It is INSULATED from the effects of such changes. Were it not for this insulation, all Application programs using a Database might have to be altered whenever any structural change had to be made.

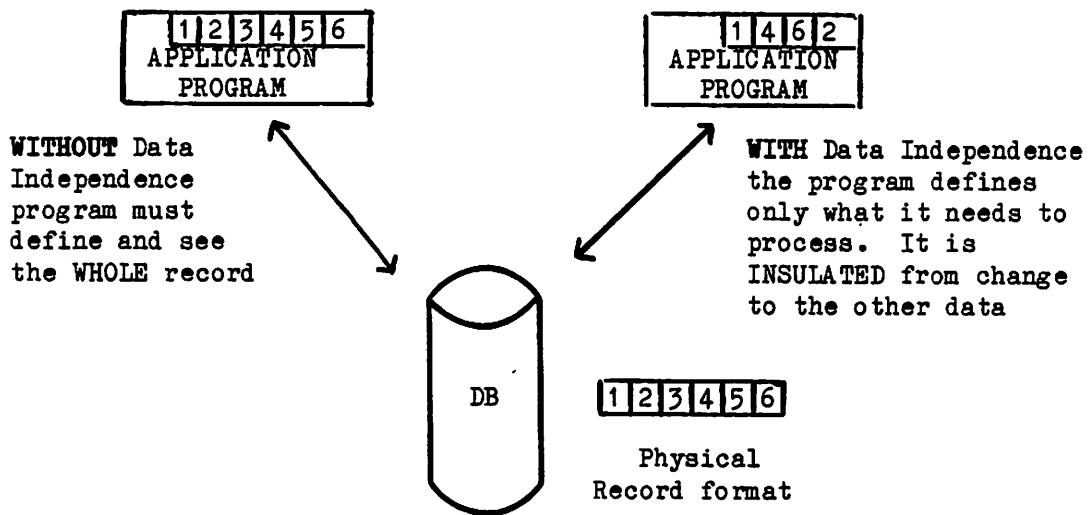


Figure 1 Subsetting Data Independence

Variation of the Physical Implementation

The Data Model can be mapped very easily to an initial LOGICAL IDMS Database design. But performance requirements, usage patterns, data volumes etc will dictate a particular optimum PHYSICAL design. IDMS allows different physical implementations to be created from the same original logical design. There are three particular cases where this can be useful:

1. To allow test Databases to be created differing in volumes from the live Database - Figure 2.

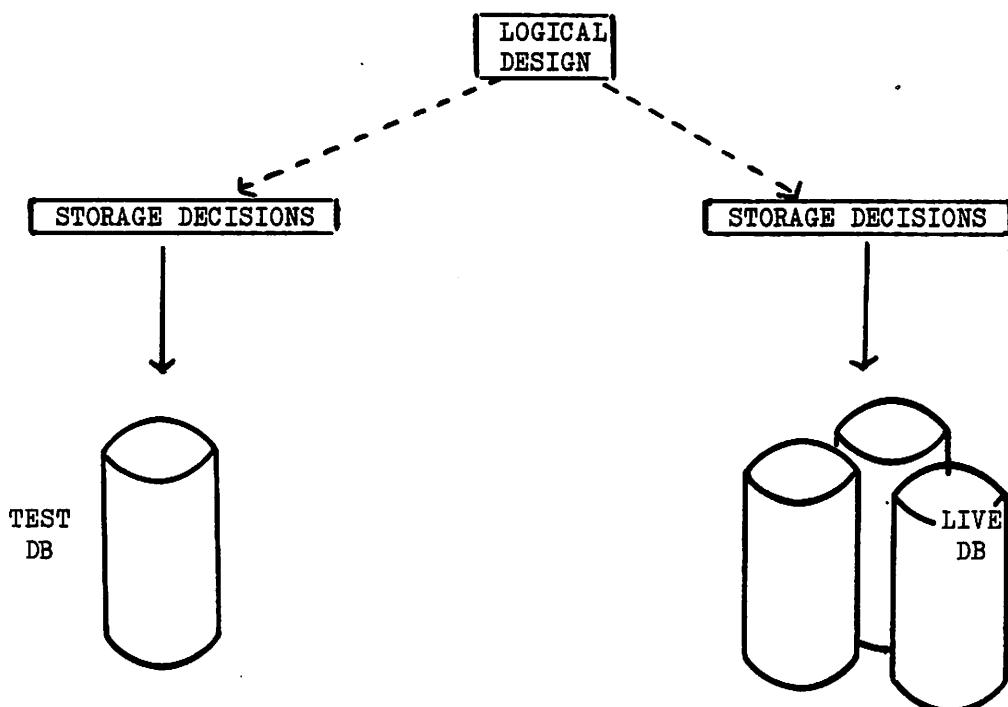


Figure 2 Varying the physical implementation decisions

2. To allow the same logical design to be implemented at different sites which vary in such factors as data volumes, usage patterns, priorities etc. A different physical design can be produced for each site, tuned to that site's particular performance requirements. This is valuable both for multi-site users and for Application packages.

Such variations in physical design have no effect on the logical design or on programs. A program's results will not be affected by physical storage decisions although its' efficiency may be. Programmers need not therefore be aware of the physical design.

3. To allow for the effects of time, bringing changes in requirements and usage. The physical storage decisions can be altered without affecting existing programs.

COMPONENTS OF AN IDMS SYSTEM

Logical Detail

Schema

This is the central LOGICAL definition of the Database. It is a complete statement of the Logical design comprising record types, data items, keys and set types - a simple mapping of the Business Data Model down to the Computer level.

Subschema

This is a named subset view of the Schema, for use by one program or a group of programs. It defines all the data available to a program at one particular time eg while processing one message pair. Subschemas provide:

1. Protection from the effects of change outside the subset view. At run-time, Subschema tables define the available subset data to the IDMS Code. The Subschema and the program will still work on an altered Database as long as the alteration is not to data defined in the Subschema tables.
2. Simple privacy by exclusion. Data not mentioned in the Subschema is "out of bounds".
3. Simpler, faster and more accurate programming. When writing the source program, the programmer merely calls for the Subschema by name and this brings into his source program all the relevant COBOL record descriptions - maybe hundreds of lines of source COBOL produced automatically.

Physical Detail

Storage Schema

This is a definition of the PHYSICAL design decisions, of how data is actually stored. It determines how the records, keys and sets are to be implemented in order to provide all the necessary access paths with appropriate efficiency.

There can be a number of different Storage Schemas to one Schema usually in order to improve performance and backing store utilisation for particular implementations.

Service Description *BATCH SERVICE OR DP SERVICE*

Application programs use the Database through IDMS Services. Each service has a Service Description defining a particular IDMS run-time environment. Services may differ from each other in such detail as the types of recovery file to be used, which parts of the Database are on-line, number of run-time buffers etc.

Changes to a Service Description do not require any changes to be made to the Schema, Subschemas or Storage Schemas.

Application Programs

Access to the Database is achieved by Application programs written in COBOL or FORTRAN. These include Data Manipulation Language (DML) statements for Database processing intermixed with normal language statements as appropriate. These programs must also call for the Subschema they are to use to view the Database.

Database access may also be achieved using the following products:

- QUERYMASTER for End-User browsing
- REPORTMASTER for DP Staff (and End-Users) reporting requirements
- APPLICATION MASTER for on-line updating applications.

THE RELATIONSHIP BETWEEN IDMS COMPONENTS

The possible relationships are shown in Figure 3.

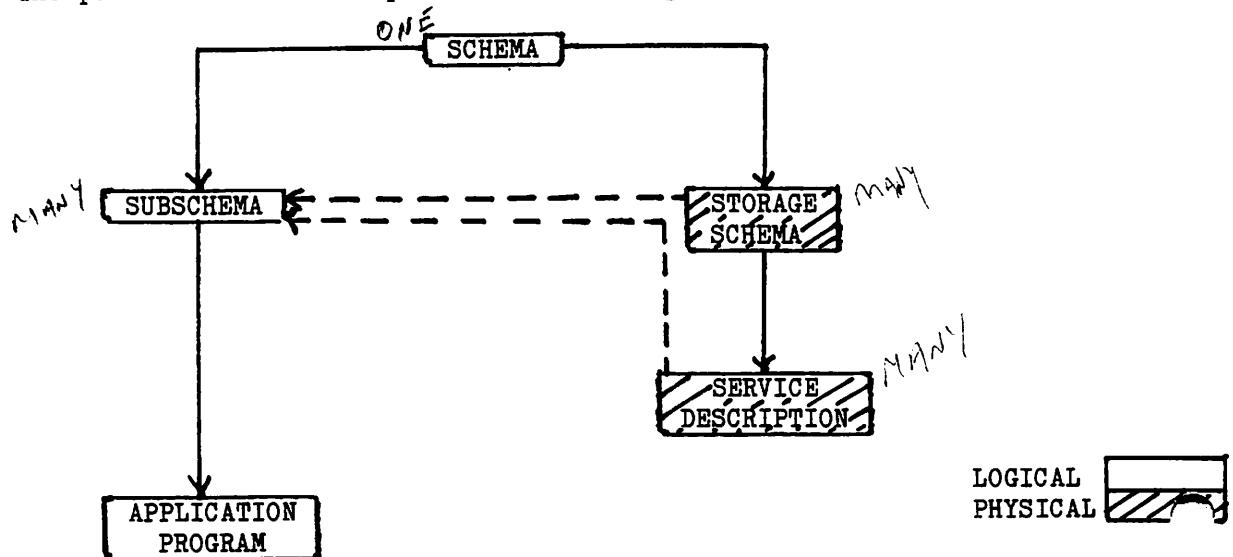


Figure 3 IDMS Logical and Physical dependencies

Optionally a Subschema can be related to a particular Service Description and Storage Schema (Dotted lines in Figure 3). This would occur when an Application program's logic depends on some physical design decision within a particular Storage Schema. Although such dependency is possible within IDMS, it is not recommended since it undermines the benefits of Data Independence.

Optionally a Subschema can be related to a particular Service Description and Storage Schema (Dotted lines in Figure 3). This would occur when an Application program's logic depends on some physical design decision within a particular Storage Schema. Although such dependency is possible within IDMS, it is not recommended since it undermines the benefits of Data Independence. A list of such decisions (together with some logical level design decisions which could have a similar effect) will be found in Appendix 2 of Chapter 9.

CHAPTER 3

LOGICAL DATA STRUCTURES

N0265-6

INTRODUCTION

This chapter is concerned with the logical data structures in IDMS.

These structures will be used by Designers and Programmers. Designers will use them during the first stage of design - the initial mapping of the Data Model to the computer level, later design work will concentrate on the physical Storage Schema level. This is considered in Chapter 4. Programmers can do all their programming work using these logical structures.

The structures to be considered are Records, Sets and Realms.

DETAIL OF LOGICAL DATA STRUCTURES

Records

Basic Concept

The Record is a familiar DP concept, the logical unit of Input/Output, comprising one or more data items which can include Groups. The data items can be of the usual types (DISPLAY, COMP, COMP-1, COMP-2, COMP-3 and COMP-6) and there can be one or more fixed length OCCURS tables and/or one variable length OCCURS table (at the end of the Record). ~~X~~

~~X~~ COMP-3 fields should be avoided where CAFS is being used. Variable length records and occurs clauses should also be avoided - they should be represented as IDMS sets. QuickBuild products cannot make use of subscripts. ~~X~~

Record Keys

Each Record type can have one selection key (in IDMS X several selection keys). Each key has a KEYNAME, which is used by the programmer, and is based on one or more data items. The items of multi-item keys need not be contiguous in the record. Records with duplicate keys values can be forbidden or allowed in first or last sequence.

It should be emphasised that these keys are for selection and do not normally allow sequential retrieval in key order. However in IDMS X, such keys can be based on a Record Index mechanism in the Storage Schema. Notice that such order keys, when multi-item, can be sequenced ASCENDING on some items and DESCENDING on others.

1. Although multi-item keys need not be contiguous in the record it is difficult to think of good reasons as to why they should not be. ~~X~~ QuickBuild products can generally only achieve keys access where multi-item keys are contiguous.

2. As will be seen later, the use of record order keys (to access records in key sequence) is "expensive". Unless they are essential they should not be used.

Database Keys

Whenever a new record occurrence is put into the Database, IDMS will give it a unique identifier called its' Database Key. A Database Key does not appear in the record it identifies but will be used by the set mechanism to link other records to that record.

A Record's Database Key never changes as long as the record lasts but may be re-used once the record is erased from the Database.

It should be emphasised that the Database Key of a record is an internal IDMS identifier. It is not a logical key like Customer Number or Order Number.

Sets

Basic Concept

A set type is a computer level device for representing a "1 to Many" Relationship from the Data Model. It links two record types together connecting ONE occurrence of the first record type to MANY occurrences of the second record type. The ONE is called the OWNER record type and the MANY are called the MEMBER records of the set.

Figure 1 shows a Data Model structure and how it would be represented by a Set Occurrence.

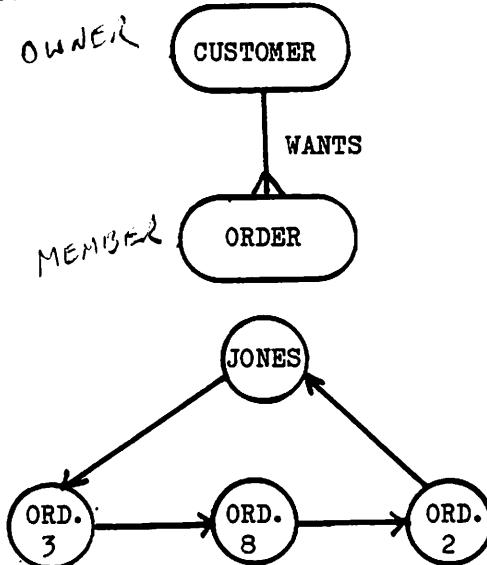


Figure 1 A set occurrence represents a 1 to Many Implementation device

ONE SET OCCURRENCE
PER OWNER REC OCCURRENCE
(CAN BE EMPTY)

The major purpose of these links is to provide access paths along which the Programmer can Navigate. As Figure 2 shows, a set can support 2 types of access.

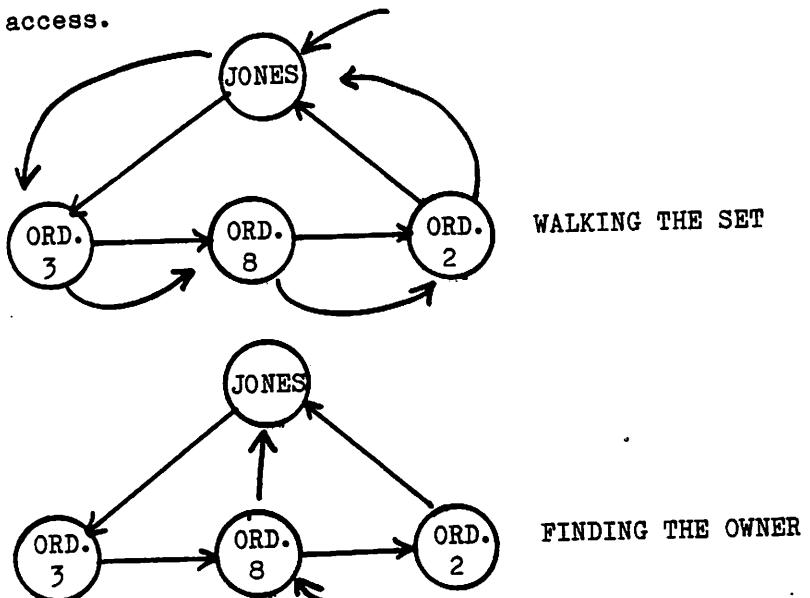


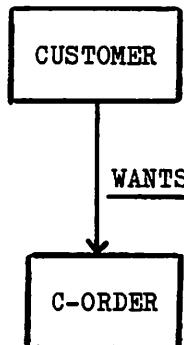
Figure 2 The two types of access provided by the Set Concept

One of the major tasks for the Designer is to decide how many Set pathways to provide. The more set types there are in his design, the more likely it is that a given Transaction can find a suitable direct route to the records it needs. On the other hand, each set type consumes a certain amount of machine resource - backing store, disc accesses, mill - to create and maintain it. As usual the Designer has to settle on a suitable compromise at the physical design level.

Some sort of documentation convention is needed to represent a set type - this is a simplified BACHMAN diagram - Figure 3.



SOFT-BOX DIAGRAM
Used by the Data Analyst



BACHMAN DIAGRAM
Used by the IDMS Designer
and Programmer

Figure 3 Bachman diagrams

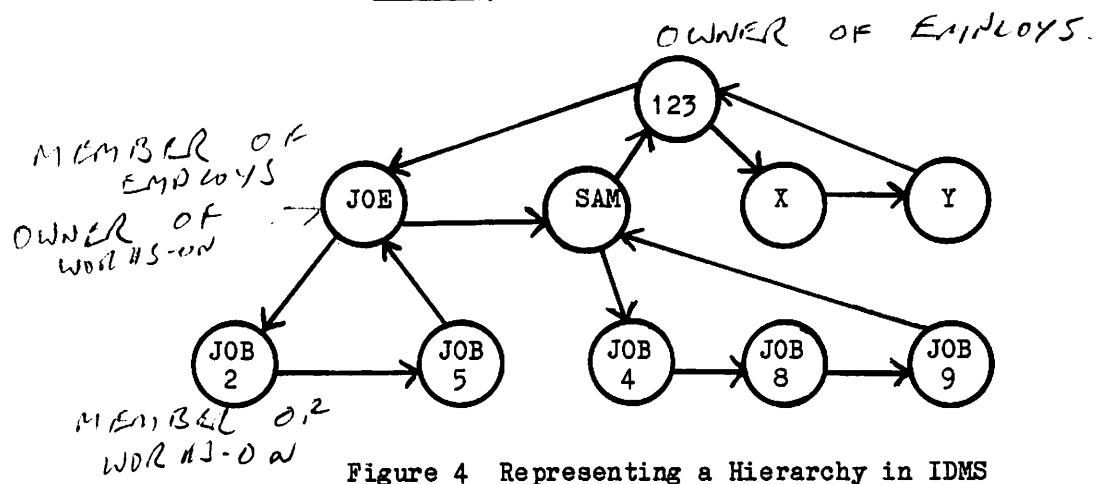
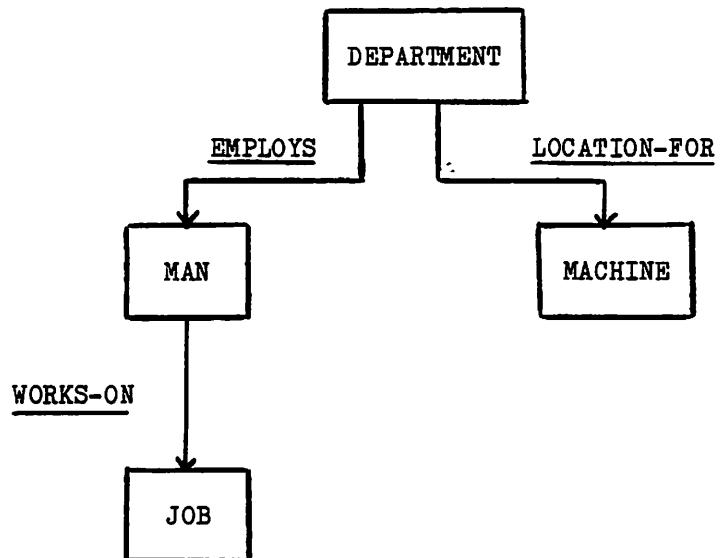
This is a diagram showing TYPES of record and set and does not give any sense of scale. That is:

- it does not indicate HOW MANY Orders there are on an individual customer. This information (Max, Min, Average) should have been collected during Data Analysis and is vital to Physical design decisions
- it does not say HOW MANY Wants sets there will be. There will in fact be one for each customer eg if you have 50,000 Customer Records in your Database, there will 50,000 Wants Set Occurrences.

Notice also that each set occurrence is completely self-contained ie there is no connection between individual Customer Owner Records. If this were required, we would have to invent a higher level set type in which customer records were members.

Building Data Structures

A record type can be an Owner in several set types and/or a Member in several other set types. By using this fact to stick set types together we can build up Hierarchies (each Record only has one Owner above it) and Networks (complete interconnection). Examples appear in Figures 4 and 5.



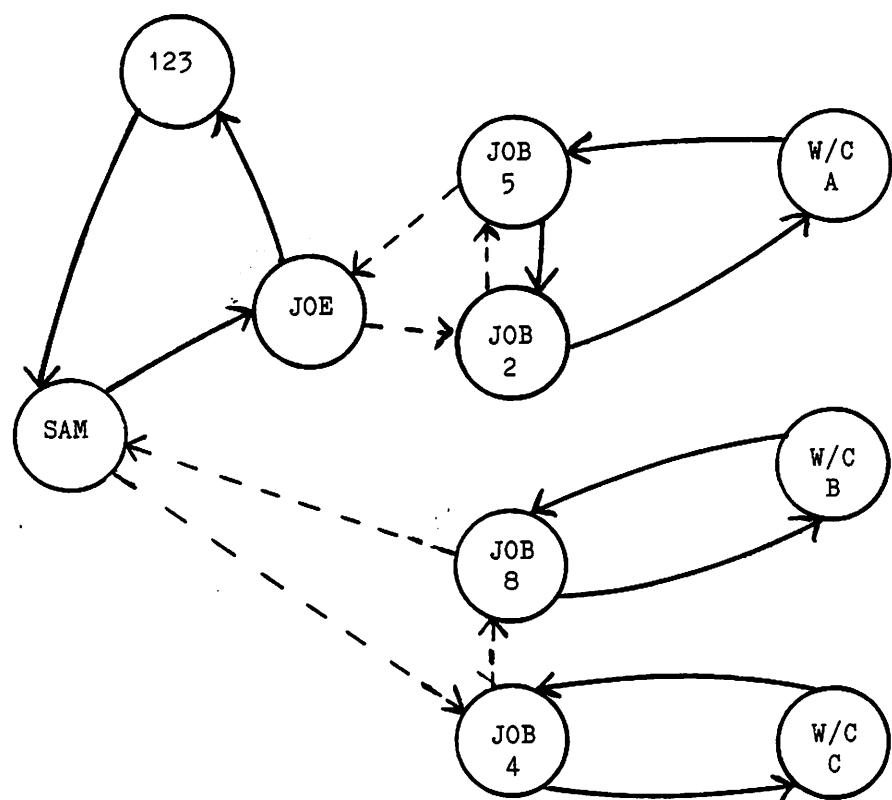
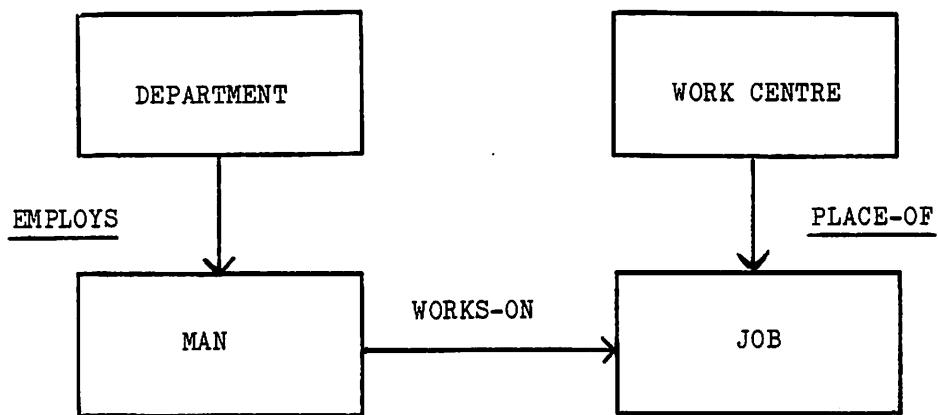


Figure 5 Representing a Network in IDMS

There are no built-in limitations to the network idea eg a record can have as many owners as required and can itself own a number of set types. However, there is one rule implicit in the set concept. This is that since the set is a "1 to Many" device, a Member Record cannot have more than one Owner from a single set type eg in Figures 4 and 5 a Job can only be linked to one Man, a Man can only be linked to one Department. If there is a requirement to link a Member to several Owner record occurrences of the same record type then a single set type cannot solve the problem. In fact, such problems are "Many to Many" structures which require two set types and an extra record type to handle them as considered in the section on "Classical translations" later in this chapter.

Two further set design decisions have to be made. These are Set Order and Insertion/Retention class.

Set Order

This is the sequence in which Member records are to be held along the linkage connecting them to the Owner. The possibilities are shown in Figure 6.

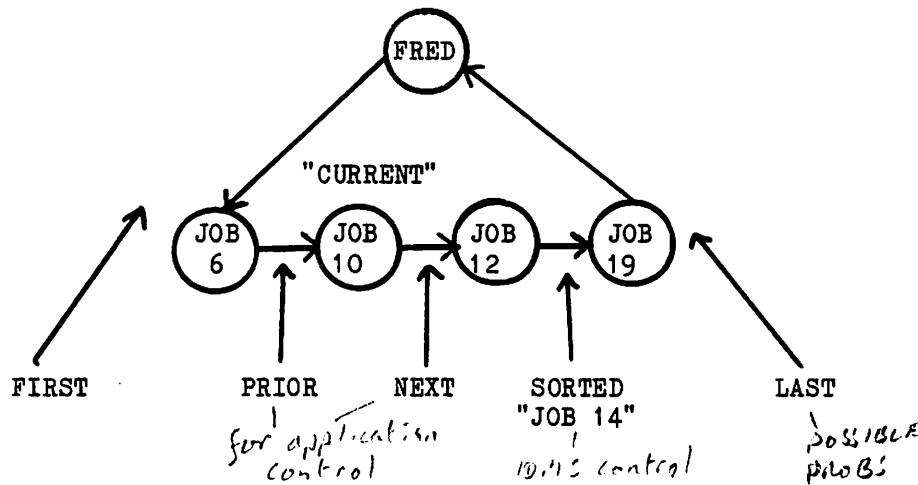


Figure 6 Options for Set Order

In effect this decides where a new Member record occurrence will be linked into an occurrence of the set type - a decision based on whatever sequence of subsequent retrieval is required when walking the set, eg LAST gives a retrieval sequence of First In, First Out.

(CAN HAVE DIFFERENT REC TYPES IN
SAME SET "MULTI-MEMBER SET"
NOT USING GOOD PRACTICE THO')

A Sorted Set sequence requires decisions on:

- the field or fields which are to be the Sort Key. The fields need not be contiguous and it is possible for some fields to be Ascending and others Descending in the same key
- what is to happen with Duplicates of the Sort Key. The new record can be placed FIRST - in front of any existing records with the same key, LAST - after existing ones or NOT ALLOWED - rejected.

One further, rarely used, option exists in IDMS X called SYSTEM-DEFAULT. This sequences the Member Records in such a way that subsequent retrieval is very efficient Physically ie with minimum head movement. In the current IDMS implementation this means that the Member Records are linked within the Set Occurrence in Database Key sequence.

Insertion/Retention Class

This decision has to be made for each Set Membership a Record type has. It controls the rules for:

- Connecting Member Records INTO an occurrence of the Set type (Insertion class)
- Disconnecting Member Records FROM an occurrence of the Set type (Retention class).

The options are as follows:

1. INSERTION Class

- AUTOMATIC = Connection is made by IDMS automatically when the Programmer stores the record into the Database for the first time
- MANUAL = Connection will only be made later (if at all) when the Programmer gives an appropriate DML CONNECT command.

2. RETENTION Class

- MANDATORY = Disconnection of the Record from the Set is forbidden. Once connected the record must stay in that Set occurrence until it is erased from the Database
- OPTIONAL = The record can be Disconnected from that set whenever the Programmer wishes to do so using a DML DISCONNECT command.

Since either Insertion option can be combined with either Retention option, there are FOUR possible Insertion/Retention classes and the Designer must decide which of the Four is most appropriate for each Member Record type in each Set type.

The four classes are:

1. AUTOMATIC MANDATORY (AM) = Insertion is automatic when the Record is first stored and Retention in the Set is then enforced throughout the Record's life. This is the usual choice.
2. MANUAL OPTIONAL (MO) = The Programmer can connect the Record into an occurrence of the Set whenever it is necessary, and remove it whenever it is necessary. This is the most flexible and allows the use of artificial CHARACTERISTIC Records to group real records into analysis categories - Figure 7.

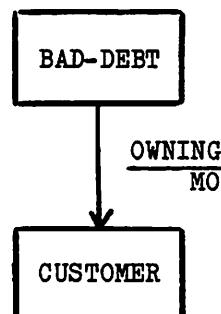


Figure 7 A Characteristic Record

3. & 4. MANUAL MANDATORY (MM) and AUTOMATIC OPTIONAL (AO) are halfway houses between the first two extreme cases.

MAY WISH TO PUT EXISTING REC INTO A
SET LATER ON e.g. BAD DEBTS
SO USE MO
IF ALWAYS WANTS CONNECTION TO OWNER
FOR LIFE OR REC USE AM
IF ALWAYS WANTS CONNECTION TO OWNER BUT
OWNER MIGHT CHANGE USE AO

Realms

The Realm is a programming level concept - it is not mentioned in Schema DDL. However, the concept is introduced in this chapter because its existence could affect logical level design decisions.

In Standard IDMS all Records of one type must be in the same Realm. This restriction is lifted in IDMS X.

The programmer can sweep or scan through all the Records, or all the Records of one type in a Realm. This will be efficient physically -it is done in Database Key order - but except in certain circumstances will not give any particular logical key sequence. However, when combined with sorting techniques batch Realm scans can perform useful work and save the Designer inventing extra Set types. The technique is especially useful for retrieving hit records meeting multiple criteria when response times are not critical. The same sequence of sweeping through a Realm is often used to load large quantities of data. In this case, however, the sequence is usually obtained by pre-sorting the input data, rather than by special programming techniques.

Note - realm scans used for retrieving records meeting multiple selection criteria are likely to be handled much more efficiently with CAFS than by conventional programming techniques.

DOCUMENTATION

Diagrams

The Schema Diagram

This is the Bachman diagram with the convention extended to cover:

- Selection Keys
- Set Order
- Insertion/Retention Class.

An example appears in Figure 8.

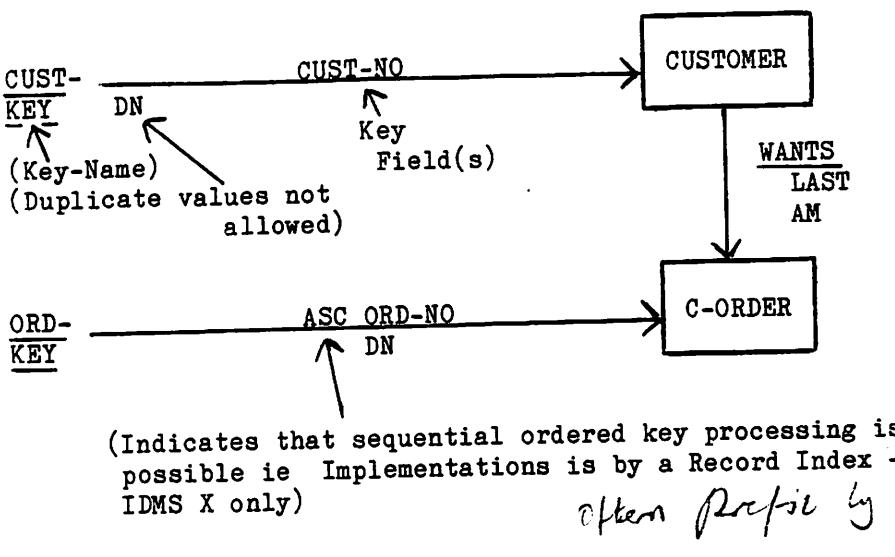


Figure 8 Standards for Schema Diagrams

naming standards

Such diagrams will be used by Designers during the logical stages of design. A slight variant used by programmers is described next.

The Subschema Diagram

This differs from the Schema diagram in two ways:

1. It describes only the Record types, Keys and Set types defined in a given Subschema.
2. The Realm Name is specified in each Record box - Figure 9.



Figure 9 Documenting Realm Names in Subschema Diagrams

REALM - GROUP TOGETHER RECORDS OF A CERTAIN TYPE

Standards for Names

Introduction

Record, Set, Item Key and Realm Names are limited to a maximum of 16 characters by the software. This section makes some tentative suggestions for naming standards within these 16 characters.

The prefixes suggested below may be useful for databases which include many record and set types.

Record Names

Should be as far as possible the name of the appropriate Entity type in the Data Model prefixed by "Rn-" where n is an integer from 1-9899 and unique for each Record type eg R1-CUSTOMER.

Item Names

Should be as far as possible the name of the appropriate Attribute from the Data Model prefixed by the "Rn-" for the appropriate Record type eg R1-CUST-NO.

Set Names

Should be as far as possible the name of the appropriate Relationship from the Data Model. Prefix this name with "Sn-" where n is the Record number for the Owner Record eg S1-WANTS.

Realm Names

Try to make these meaningful.

Note. These standards will be found of increasing use as the Schema gets larger, in helping Designers and Programmers keep control of the situation. They are less useful in small situations and have therefore have not always been used in the various examples and exercises of this course.

Key names

Try to make these applicable to what they identify. They should be suffixed by the word KEY eg CUST-KEY.

CLASSICAL TRANSLATIONS

Introduction

There are a number of standard procedures for mapping various Data Model structures down onto IDMS logical structures. This section considers these common classical procedures.

Entities

In general each Entity type becomes a Record type with each Attribute becoming a field. Variations from this simple mapping may occur during the tuning stages of design.

The Identifier of each Entity type may well become a Selection Key. In IDMS X other attributes may become selection keys instead or as well either alone or in combination with other attributes. However, at the physical level each key has to be supported by a suitable mechanism which could prove expensive in machine resources. The mechanisms will be considered later in the course, but in general only ask for a Selection Key (even on the Identifier) if a higher priority transaction needs that type of Entry point access.

In a few cases, there may be a requirement for two or more high priority selection keys on a record type when it is known that the eventual implementation is to be on Standard IDMS. One solution to this problem is to move each selection key (after the first) to an artificial Record type which can then be owned by or own the real Record in an artificial Set. In this context "artificial" means "not in the Data Model". However this device, known as a "1 to 1 access Record", allows selection of a suitable real Record by:

1. Selecting the appropriate artificial Record on Key value.
2. Navigating to the real Record using the Set linkage.

Figure 10 shows an example.

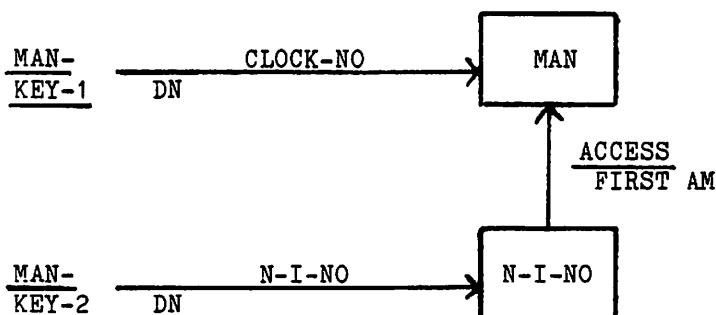


Figure 10 A 1 to 1 Access Record

Relationships

Between Two Different Entity Types

These may be "1 to Many" in which case the mapping to a Set is obvious eg Figure 1. Alternatively they may be "Many to Many" as with STUDENT and COURSE.

In most cases, the Data Analyst will have tidied up this situation in the Data Model as in Figure 11 and the IDMS logical mapping is then trivial.

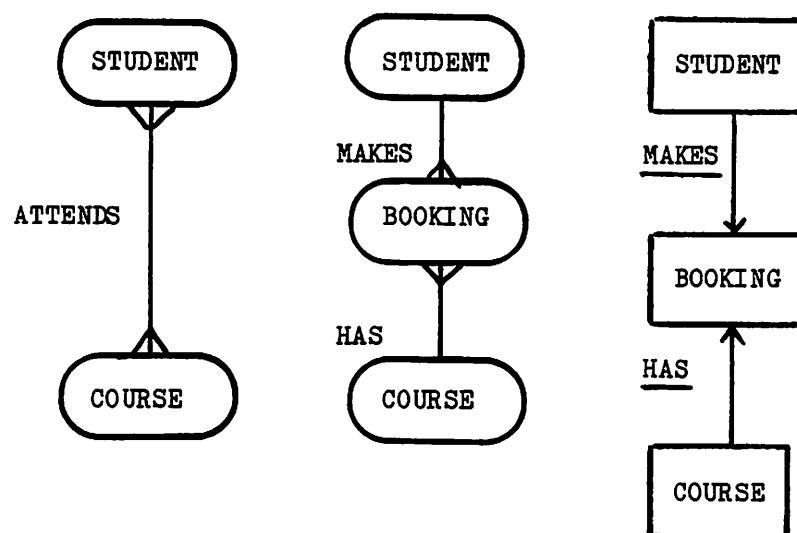
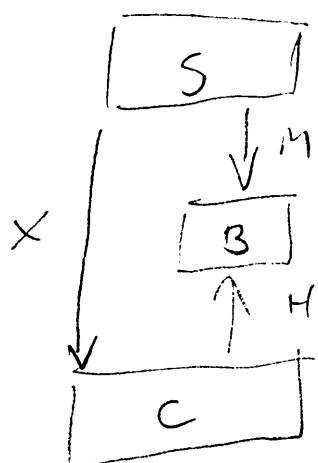


Figure 11 Evolution of a Many to Many Relationship

The BOOKING Entity/Record will contain data about that student attending that course eg when the Booking was made, whether it has been confirmed, how much money has been paid etc.



3 - 14

$X = 99\% \text{ of Time}$

$M+H = 1\% \text{ of Time}$

apply rule to determine which

NO265-3

Between Different Occurrences of the Same Entity Type

Such cases are rare (except in one classic example). Again they can be "1 to Many" or "Many to Many".

Figure 12 shows an example of a "1 to Many" case.

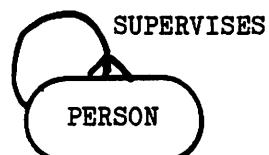


Figure 12 A 1 to Many Relationship between occurrences of the same Entity type

There is no single agreed IDMS mapping for this problem. It is handled by various tricks with the Set mechanism eg some adaption of the Bill of Materials approach shown below.

The classic structure in this area is the "Many to Many" Bill of Materials application.

Bill of Materials Structure

Figure 13 shows an example of Parts linked together in a "Many to Many" structure to form two Products.

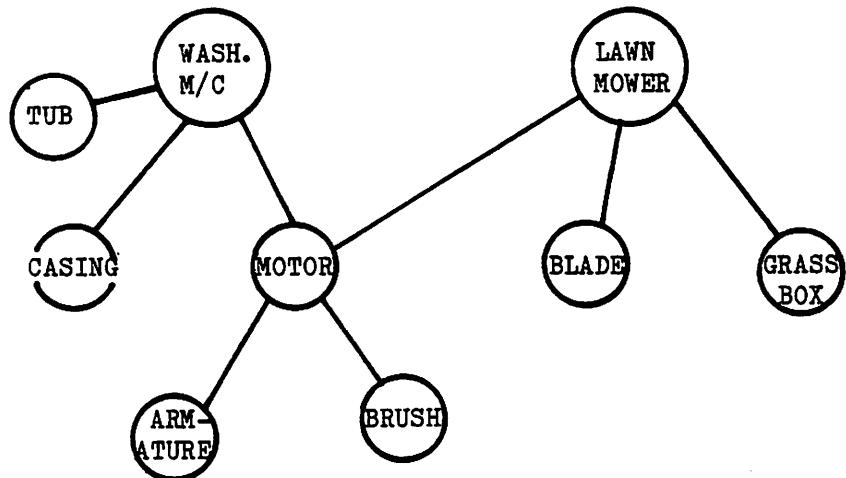


Figure 13 Part Occurrences linked to form product structures

A soft-box diagram of this shape is shown in Figure 14.



Figure 14 Underlying BOM Structure

The key to understanding how to map this structure to IDMS is to realise that TWO Entity types are involved:

- PARTS = Data about parts (whether they be Finished Products or a Nut and Bolt) like Part Number, Description etc
- STRUCTURES = Data about the connection between Parts eg the number of Carbon Brushes (Component Part) needed to make one Electric Motor (Parent Part). Each Structure Entity can then be on two Relationships, one to the Parent Part, the other to the Component Part. There will be a Structure Entity for each different use of a Part as a Component.

Once this is realised the structure can be held very easily on IDMS Set Structures - Figure 15.

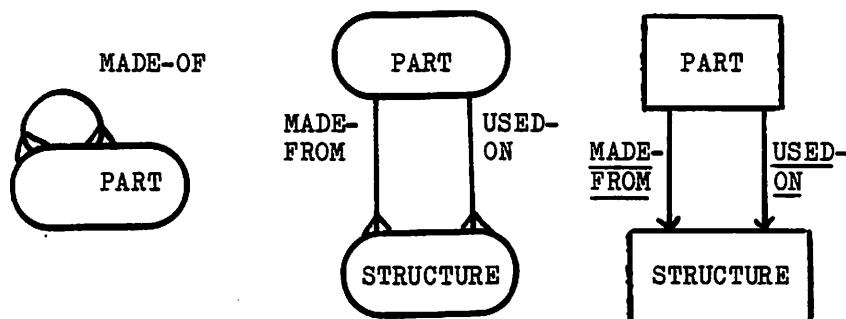


Figure 15 Handling BOM Structures

STRUCTURE RELS ARE USED
TO LINK PARTS.

Figure 16 shows some Record occurrences giving part of the structure of a Washing machine and a Lawn Mower.

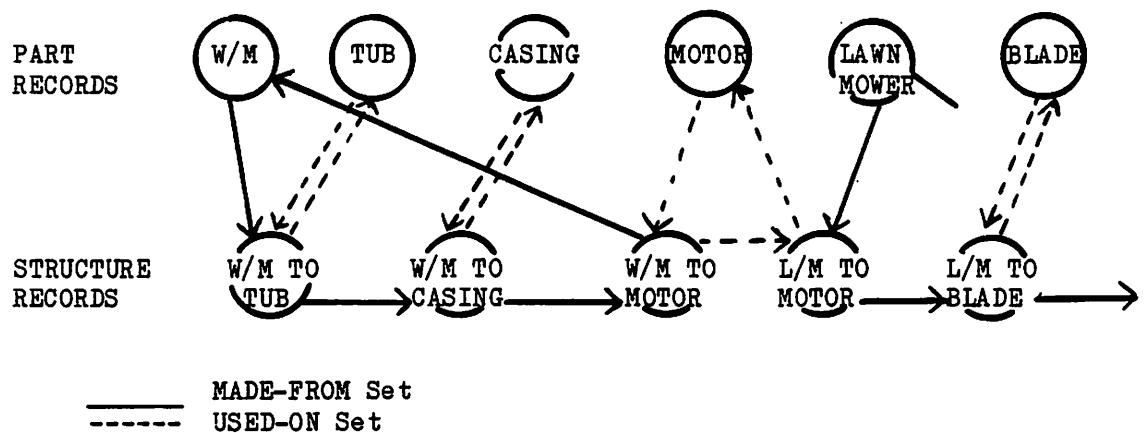


Figure 16 Part of a BOM Occurrence Structure

This sketch shows only part of the structure eg one level down. Further levels would involve the use of more structure Records and of the Made-from set on the Component Parts eg Motor could be broken down further in this way.

Multi-Member Type Set Types

It is possible to represent several Relationships with a single Set type since a single Set type may have several types of Record as Members. In Figure 17 there are really TWO Relationships (the first between Department and Man, the second between Department and Machine) represented by the single RESOURCES-ARE Set.

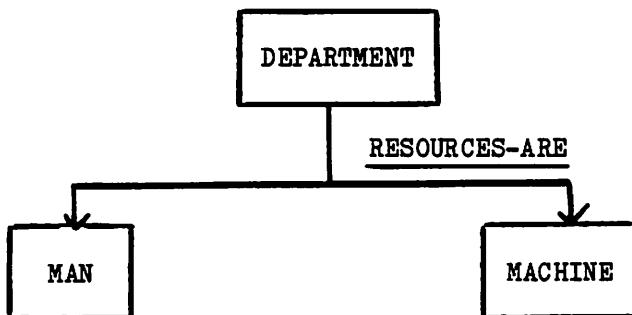


Figure 17 A Set with two types of Member Record

IF TWO THINGS ARE ALWAYS USED
TOGETHER, EFFECTIVELY 2 SETS NEED
TOGETHER

There is no limit to the number of types of Member Record in one Set type. The sequence within an occurrence of the Set depends on the Set Order and may be random as far as Record types go ie they can be completely mixed in sequence along the linkage. Sorted Set order allows multi-member type sets to be either:

- sorted on key value irrespective of Record type
- sorted on key value within Record type.

However, this multi-member type facility is essentially a distortion of the Data Model. As such this facility should not be used at the logical design level but only after physical design considerations have revealed a suitable trade-off.

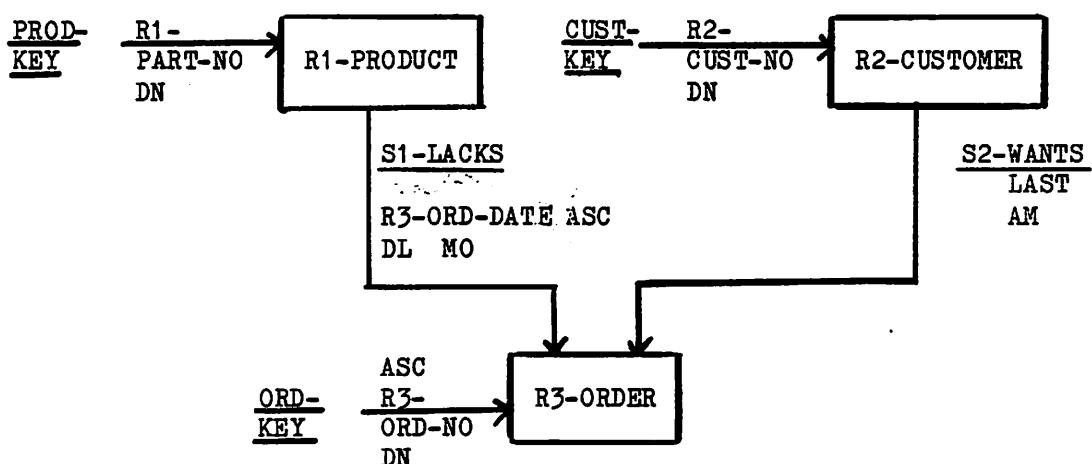
APPENDIX

This appendix describes a structure in 3 ways:

1. As a Schema Diagram.
2. As Schema DDL.
3. As DDCL for the DDS.

Once the last has been loaded into the Dictionary, the second can be generated automatically. Notice that the recommended naming standards have been used.

SCHEMA DIAGRAM



SCHEMA DDL

SCHEMA IS CASHFLOW.

RECORD R1-PRODUCT.

KEY PROD-KEY R1-PART-NO
DUPLICATES NOT ALLOWED.
03 R1-PART-NO PIC X(15).
03 R1-DESC-TEXT PIC X(20).
03 R1-QTY PIC 9(6).
03 R1-SAFETY-LEVEL PIC 9(3).

RECORD R2-CUSTOMER.

KEY CUST-KEY R2-CUST-NO
DUPLICATES NOT ALLOWED.
03 R2-CUST-NO PIC X(8).
03 R2-C-NAME PIC X(20).
03 R2-CREDIT-LIMIT PIC 9(8) COMP.
03 R2-ADDRESS PIC X(20) OCCURS 4 TIMES.

RECORD R3-ORDER

KEY ORD-KEY ASCENDING R3-ORD-NO
DUPLICATES NOT ALLOWED.
03 R3-ORD-NO PIC 9(6).
03 R3-ORD-DATE PIC 9(6).
03 R3-PART-NO PIC X(15).
03 R3-DESC-TEXT PIC X(20).
03 R3-QTY PIC 9(6).

SET S1-LACKS.

OWNER R1-PRODUCT.
ORDER SORTED.

MEMBER R3-ORDER.

INSERTION MANUAL RETENTION OPTIONAL.
KEY ASCENDING R3-ORD-DATE
DUPLICATES LAST.

SET S2-WANTS.

OWNER R2-CUSTOMER.
ORDER LAST.

MEMBER R3-ORDER.

INSERTION AUTOMATIC RETENTION MANDATORY.

DDCL

INSERT SCHEMA CASHFLOW.
*DBMS 2900 VME IDMSX 400
*RECORDS R1-PRODUCT
R2-CUSTOMER
R3-ORDER
*SETS S1-LACKS S2-WANTS

INSERT RECORD R1-PRODUCT
*STRUCTURE ITEM R1-PART-NO/PART-NO
ITEM R1-DESC-TEXT/DESC-TEXT
ITEM R1-QTY/QTY
ITEM R1-SAFETY-LEVEL/SAFETY-LEVEL
*KEYS KEY PROD-KEY IS
R1-PART-NO
DUPLICATES NOT ALLOWED

INSERT RECORD R2-CUSTOMER
*STRUCTURE ITEM R2-CUST-NO/CUST-NO
ITEM R2-C-NAME/C-NAME
ITEM R2-CREDIT-LIMIT/CREDIT-LIMIT
ITEM R2-ADDRESS/ADDRESS OCCURS 4
*KEYS KEY CUST-KEY IS
R2-CUST-NO
DUPLICATES NOT ALLOWED

INSERT RECORD R3-ORDER
*STRUCTURE ITEM R3-ORD-NO/ORD-NO
ITEM R3-ORD-DATE/ORD-DATE
ITEM R3-PART-NO/PART-NO
ITEM R3-DESC-TEXT/DESC-TEXT
ITEM R3-QTY/QTY
*KEYS KEY ORD-KEY IS
ASCENDING R3-ORD-NO
DUPLICATES NOT ALLOWED

INSERT SET S1-LACKS
*OWNER R1-PRODUCT
*ORDER SORTED
*MEMBERS MEMBER R3-ORDER
OM _____
KEY ASCENDING
R3-ORD-DATE
DUPLICATES LAST

NUT MO
CHARS SUITABLE

NEED TO PUT FILLER IN
FOR FUTURE FIELDS

CAN REDUCE FILLER FOR
EXTOL FIELDS AND END

INSERT SET S2-WANTS
*OWNER R2-CUSTOMER
*ORDER LAST
*MEMBERS MEMBER R3-ORDER
MA

INSERT ITEM PART-NO
*PICTURE X(15)

INSERT ITEM DESC-TEXT
*PICTURE X(20)

INSERT ITEM QTY
*PIC 9(6)

INSERT ITEM SAFETY-LEVEL
*PIC 9(3)

INSERT ITEM CUST-NO
*PIC X(8)

INSERT ITEM C-NAME
*PIC X(20)

INSERT ITEM CREDIT-LIMIT
*PIC 9(8)
*USA COMP

INSERT ITEM ADDRESS
*PIC X(20)

INSERT ITEM ORD-NO
*PIC 9(6)

INSERT ITEM ORD-DATE
*PIC 9(6)

CHAPTER 4

PHYSICAL STORAGE STRUCTURES

N0265-6

INTRODUCTION

This chapter examines the PHYSICAL structures used by IDMS ie the range of options available to the Designer as he tunes his Logical design. The decisions that he makes about a particular Physical Database design will be expressed as a Storage Schema -a particular physical implementation of the Logical design expressed in the Schema.

The chapter is divided into the following major sections:

- IDMS Physical Structures
- The physical placement of records
- Data storage defaults
- Documentation of a Storage Schema.

IDMS PHYSICAL STRUCTURES

Introduction

This section considers a number of physical concepts:

- Pages (Disc blocks)
- Areas (a group of Pages)
- Record structures
- Set Implementation mechanisms
- Index structures.

Pages

Definition

An IDMS Page is the physical unit of transfer, held on one disc block.

Each page has a page number unique across the whole Database. Pages 1 to 1000 are reserved for the directory. Some or all (100 up to 1000 pages) are used by IDMS to hold a description of the Database formats. These pages are known as the DIRECTORY.

A directory is needed to compile COBOL programs.

The remaining pages are for user records and could extend up to page 8,388,607.

Page size is specified by the Storage Schema and can be from 64 bytes to the maximum disc block size allowed by the physical device (but no more than 32 Kb at present). In fact most users have page sizes in the range 3 Kb to 6 Kb.

Page size must be constant across all user pages unless IDMS X is in use in which case Page Size can vary from one area to another. The meaning of areas in this context is described later in this chapter.

Lines

A page can contain a number of record occurrences of various types with a maximum of 255. Each record occurrence is regarded as sitting on a LINE - each line having a unique number within the page starting at 1.

Records are linked together in a Set by Pointers eg by a pointer field in one record quoting the page and line number of another - as in Figure 1 where a customer JONES is linked to his first order -ORDER 123.

PAGE 10,000

PAGE LINE
NO NO
2100 3 JONES
POINTER
FIELD

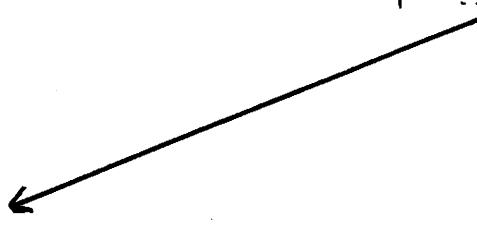
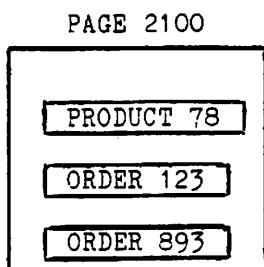


Figure 1 Linking records with pointers

The line is not a physical concept eg Line Number 3 does not necessarily indicate the third record on the page. If it did, IDMS could not shift records around within a page to optimise the use of space without having to change pointers TO the shifted records. In order to avoid this complication, a LINE INDEX is held at the bottom of each page. This is used by the software when it is following a pointer to translate the line number in the pointer into the real position of the record at present - Figure 2.

PAGE 10,000

2100 3 JONES
POINTER
FIELD

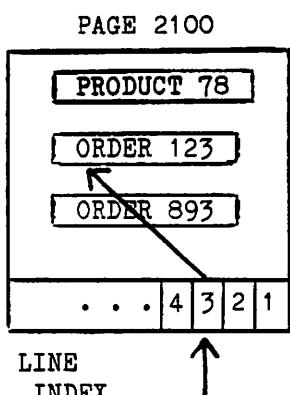


Figure 2 The line index translating the line number into the position of the record
NO265-3

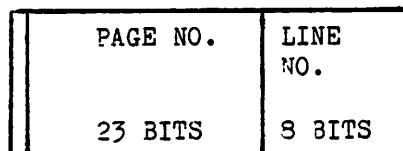
Should the Order Record have to be moved, the software merely has to update the third entry in the Line Index to point to the new position of the record for any pointer of the form 2100/3 to remain valid.

While this mechanism is completely automatic it does have a number of interesting design and programming implications:

1. As we have just seen IDMS can shift records around within a page quite freely. So IDMS Databases are self-reorganising in the sense of always making best use of available space. All records are kept together at the beginning of a page leaving the maximum free space after them.

Notice, however, that this process will never move a record out of the page it is loaded into initially.

2. The pointer field is 32 bits long (4 Bytes) laid out as in Figure 3 with values held in Binary.



1 BIT
SPARE

Figure 3 Format of a pointer

Hence the limits quoted earlier of 8,388,607 pages in an IDMS Database and of 255 records per page.

Reference was made in an earlier chapter to each record occurrence having a unique Database Key. In fact, the Database Key format is the 32 bit page number/line number shown in Figure 3.

3. The line index entry for a record occurrence not only says WHERE the record is in the page at the moment, it also says:

- what kind of record it is
- how big the record is.

This means that there is no point in having a record type field in a record. IDMS always knows what kind of record it is retrieving by looking at the line index and it will tell the programmer at that time.

Page Formats

Each page is laid out as in Figure 4.

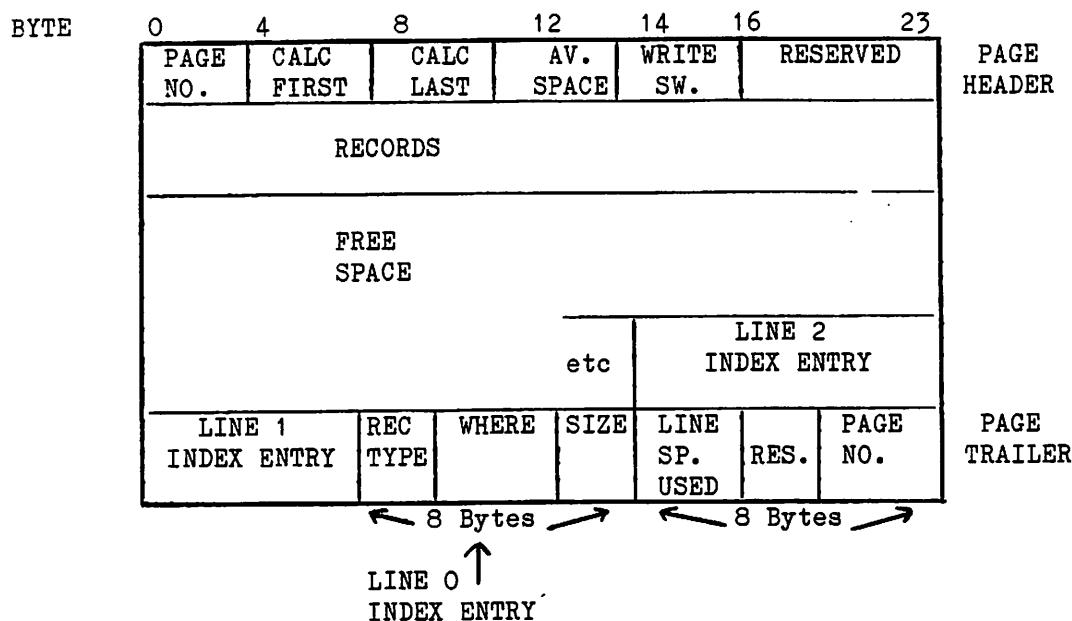


Figure 4 Page formats

1. The Page Header

This contains:

- the page number
- the first and last pointers on the CALC chain. The CALC chain will be considered in detail later in this chapter.
- a count of available space on the page
- a "Write Switch" used, when the page is in a buffer in the OCP, to decide whether it should be written back to the Database.

These entries plus a reserved section make up 24 Bytes.

2. Records

These are close packed together.

DBKEY = PAGE NO. + LINE NO.

3. Free Space

For new records and/or for existing ones to expand into.

4. The Page Trailer

This consists of a fixed 8 byte portion:

- page number
- count of line space used ie size of line index at present.

Plus a variable length portion made up of the line index entries which are 8 Bytes long each. The first of these covers the record on line 0 which is the Page Header. The rest from line 1 onwards cover real records. In all cases the format of the 8 Bytes for a line index entry is:

- LINE INDEX ENTRY
- Record type held as a Record Identifier - a binary number in the range 1-9999 unique to each record type
 - Where the record starts-in the form of the number of Bytes preceding the record on the Page (Page Displacement)
 - Size. This is actually two counts. The size of the whole line and the size of the pointers in this record.

Since the line 0 entry is always present, the Page Trailer is never less than 16 Bytes and the minimum red tape on a page is 40 Bytes.

Note. Notice that Figure 4 shows the format as the run-time IDMS code sees the page in a buffer. The structure of the physical block on the disc may be identical to this. However, if CAFS searches are to be performed on the same blocks, then an alternative SEARCHABLE format must be stored on the disc. This format has the same content as that shown in Figure 4 and takes up no more space but by shuffling the sequence of the material, it allows CAFS to get to the Line Index information before the records it covers.

PAGE No.
23 6:65

Range:
1 to 8,388,607

LINE No.
8 6:65

1 to 255

*CDFS = Slight
overlook as
changed format*

Searchable format is created by specifying FORMAT IS SEARCHABLE for the Area in question and then by a physical dump and restore process. Not all Areas need be converted. However, those that have been changed to searchable format can now be processed in two different ways in the same or different programs:

1. By CAFS high speed hardware scans to answer enquiries.
2. By other methods for enquiry and updates. In this case the searchable format is converted to the format in Figure 4 automatically by software before the main IDMS code sees the buffer and back to searchable format automatically on output.

Note - IDMS software always "sees" database pages in "normal" format rather than searchable format.

Pages in Use

Figure 5 shows a page containing 4 records.

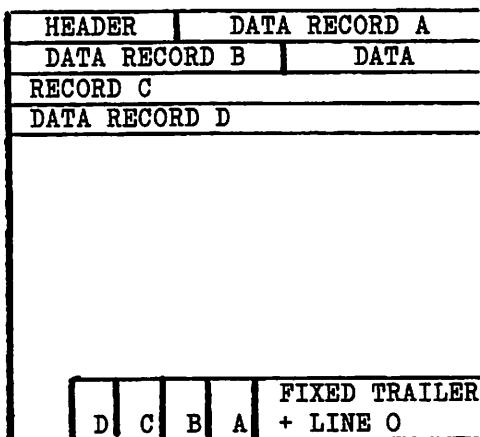


Figure 5 A page before deletion of a record

Figure 6 shows what happens when Record C is deleted:

- Record D moves up (and line index entry 4 is adjusted to point to the record's new position)
- Line index entry 3 is clear.

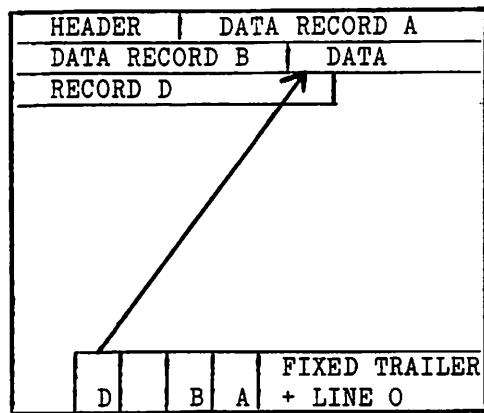


Figure 6 A page after deletion of a record

Figure 7 shows what happens if a new Record E is added.

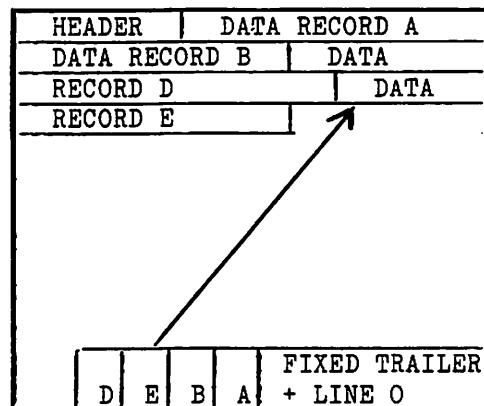


Figure 7 Result of adding a new record

If this were page x then the Database Keys of the 4 records are:

A	x/1
B	x/2
D	x/4
E	x/3

In other words if a pointer in some other record is to point to Record E, it will contain x/3 which means:

"To get to Record E read Page x, then look at the third entry in the line index which will point to where the record starts."

Areas

Definition

An Area is a group of consecutively numbered pages.

An Area should map onto one VME file although it is possible to split an Area across several files or vice versa.

Figure 8 shows a Database split into 3 Areas.

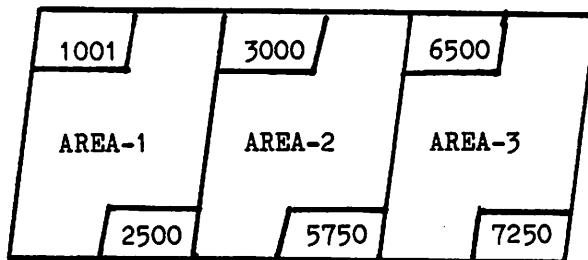


Figure 8 Three user areas in a Database

MINT THE GALT !
The gaps in the page numbering, eg from 2501 to 2999 between Areas 1 and 2, allow room for extension should an Area threaten to get full.

In standard IDMS only, all occurrences of one record type must be in the same Area - along with occurrences of other record types if required.

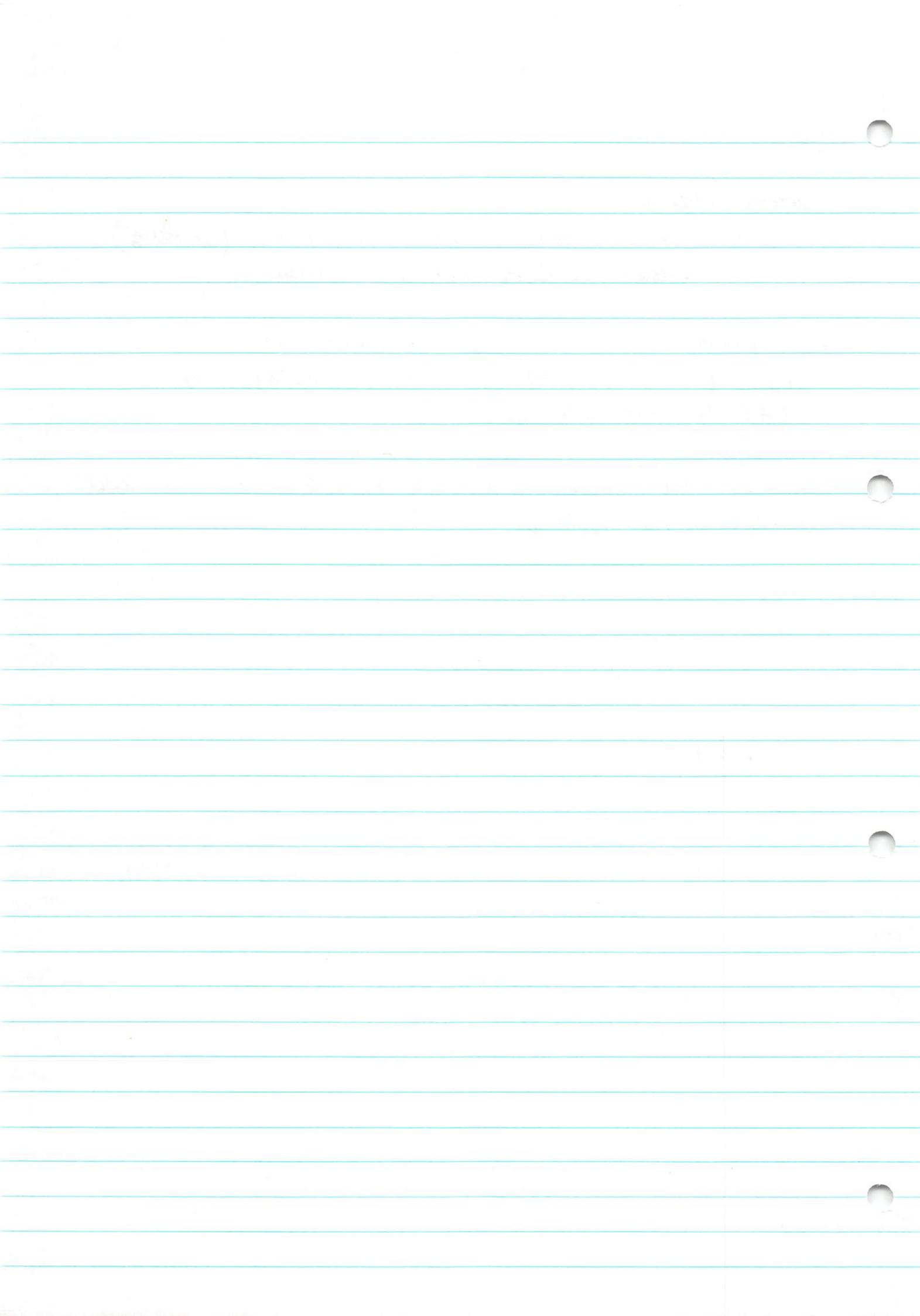
With IDMSX a record type may appear in more than one area.

SPACE MANAGEMENT RECORD

- SHOWS WHICH PAGES ARE FULL ($>70\%$)
- SHOWS WHICH ARE NOT FULL

AS PAGES FILL I/O'S INCREASE AS
MORE & MORE PAGES ARE ACCESSED IN
SEARCH OF SPACE

SIZE OF SMR ie RESPONSIBILITY FOR X PAGES
DESIGNED BY USER DESIGNER



What are Areas for?

The area concept provides 4 facilities.

1. Areas can be left off-line where possible and appropriate, in order to free on-line disc capacity.
2. Areas can be a useful unit for security dumping and for restoration after a disc failure.
3. Areas can be locked by one user against access by other users while some critical extensive updating is carried out. (In many cases, however, such a unit of locking is too large and page locking will be used instead.)
4. Programs can scan through an area page by page processing the records in each page in Database Key order ie by scanning the line index entry by entry. This can be very efficient physically and is useful for batch reports where the sequence required is random or can be achieved by a subsequent sort.

Areas and Realms

From the above points there is obviously a close parallel between the physical concept of the Area and the programmer's concept of the Realm. In fact, they are usually the same thing. When a programmer requests the use of a Realm or scans through a Realm, he is actually handling an Area. However, it is also possible for a Realm to be all occurrences of one record type irrespective of:

- whether the occurrences of the record type are in one Area or are in many Areas (IDMS X only)
- whether there are other types of record in the Area(s) as well. These will be ignored unless they are themselves specified as being in other one record type Realms.

Use of this facility gives a limited degree of program independance of any subsequent changes to the mapping of record types to Areas.

COULD MAKE ALL FLAT FILES AREAS
FOR SIMPLE RELOCVERY PROCS

USING PROB

SAME READS

TO DIFFERENT

AREAS

SAME WITH

VARIOUS JOBS.

Alternative Areas

This facility is an extension to the "Record type in more than one Area" feature in IDMS X.

It allows occurrences of the same structure to be held in a number of Areas and then for a program written specifically to process one of these Areas to be switched at run-time to process any one of the other Areas instead. For example, Customer Accounting Records for each of six Sales Regions could be placed in six separate Areas. A program could be written to process one of these Areas only eg all the Customers in the SOUTH region. Then provided the other five Areas were said in the Storage Schema to be Alternatives to the one containing the Southern Customers, that same program could process any one of the six Areas on a given run. The Area actually used on a given run is decided by an SCL procedure called IDMS_SELECT.

Areas and Files

The Storage Schema specifies which file (or files) the Area occupies. This "Logical" file is mapped to a "Real" VME file by SCL at run-time (eg the IDMSFILES command).

If required (in IDMS X) the file can be Duplexed. In this case the logical file will map to two Real files which will both be written on any output but read alternately on input.

Virtual Store Areas

It is possible for some or all of the Pages in an Area to be copied into a Virtual Store Module when an IDMS service starts (see chapter 6 for a full explanation of the concept of an IDMS service). This Virtual Store Module can be locked down in Real Store using Module Amender if required.

Any subsequent read access to that Area does not now require a Database disc access, although an update will cause a normal output to occur.

X This facility is especially valuable in speeding up access to small amounts of frequently retrieved reference data. *X*

There can be a number of Page Ranges from one or many Areas in the Virtual Store Module. The maximum is 255 Page Ranges each up to 256 Kb in size.

Records

Basic Structure

Almost all IDMS records have two major sections - POINTERS (for Set connections) followed by DATA - Figure 9.

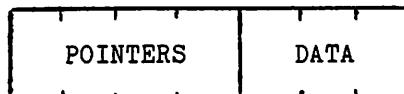


Figure 9 Structure of an IDMS record

It is possible for the physical structure of IDMS records to be exactly like this. They will be fixed in length and must be smaller than a page in size. Such a structure is simple and efficient.

Fragmentable Format MEGA YUK !!

Fragmentable format means a record can spread across a number of pages - Figure 10.

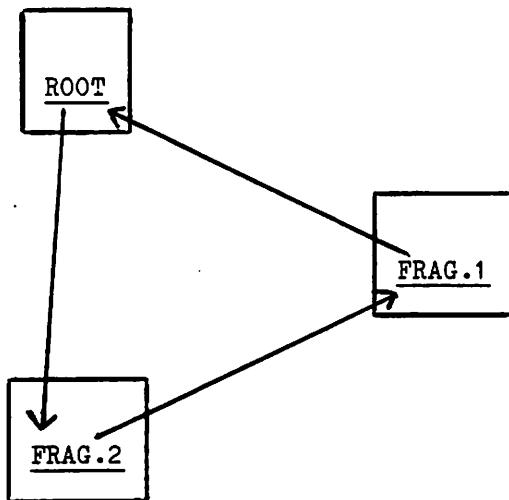


Figure 10 A record fragmented across three pages

This format is available for the following types of record:

- variable length records ie those with an OCCURS DEPENDING clause. There can be a maximum of one of these groups per record at the end of the Data section
- records defined as COMPRESSED. Such records are automatically compressed in size when stored in the Database (using an algorithm built into IDMS) and expanded again automatically when read from the Database. While this saves backing store, it consumes OCP time
- records defined as Fragmentable by specifying a MINIMUM ROOT or MINIMUM FRAGMENT length
- index records (IDMS X only) which make up the record and set indexes described later in this chapter. However, such index records are only fragmentable in as much as they have the format of the Root of a fragmentable record, they are not allowed to fragment physically.

The Root must contain (at least) all the pointers and key fields. In addition, there are two fields which do not appear in normal records - a size word and a pointer to the last fragment - Figure 11.

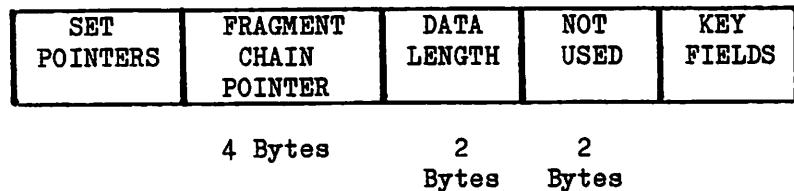


Figure 11 Minimum Root formats

Fragmentable records will not be very efficient to process if considerable fragmentation has occurred ie if the record has spread across many pages.

USE FOR TEXT STORAGE??

Fragmentation can be limited by specification of:

- a minimum Root length and a minimum Fragment length per record type
- a page reserve per Area.

RESERVE
PAGES TO
HOLD
FRAG
RECS

This would be a section on each page reserved for variable length records already on the page to expand into, so that physical fragmentation is prevented or delayed. If present the entry can be overridden by the Service Description.

Records which have fragmented cannot be accessed by CAFS. Fragmentable record formats should be avoided. CAFS + QB PROBS

Re-located Records

It is sometimes necessary for a record (whether fragmentable or not) to be moved from the page where it was physically stored at first. This could occur in two cases:

- If the Restructure Utility system has to increase the size of a record eg by adding extra fields, and then cannot get the expanded record back into the page it came from
- During index handling.

In such circumstances a TAG is left in the original page, and the record is re-located in some other page. The unchanged Database Key points at the Tag and the Tag points at the new position for the record.

Relocated records cannot be accessed by CAFS. The index phase of the restructure utility can be used to reorganise relocated records - see chapter 10.

Record Indexes

In IDMS X it is possible to have one or more indexes per record type. They are one method of implementing the Schema Keyname facility and allow both selection and ordered key access eg if there is an index on the "Order Number" field within order records then it can be used to:

- retrieve an order record with a given Order Number Value DIRECT ACCESS
- retrieve the next order record in Order Number sequence (or previous or first or last). SEQUENTIAL PROCESSING

Sets

Introduction

The Set is an implementation of a One to Many Relationship from the Data Model. Two types of Set mechanism are provided:

- Chained Sets
- Indexed Sets (IDMS X only).

In IDMS X the decision as to the mechanism to be used is made per Set type.

Chained Sets

This mechanism is the classic and simplest method for linking an Owner record to a number of Member records. As Figure 12 shows, there will be:

- a chain of NEXT pointers leading from a FIRST pointer in the Owner
- a chain of PRIOR pointers leading from a LAST pointer in the Owner
- individual OWNER pointers in each Member record.

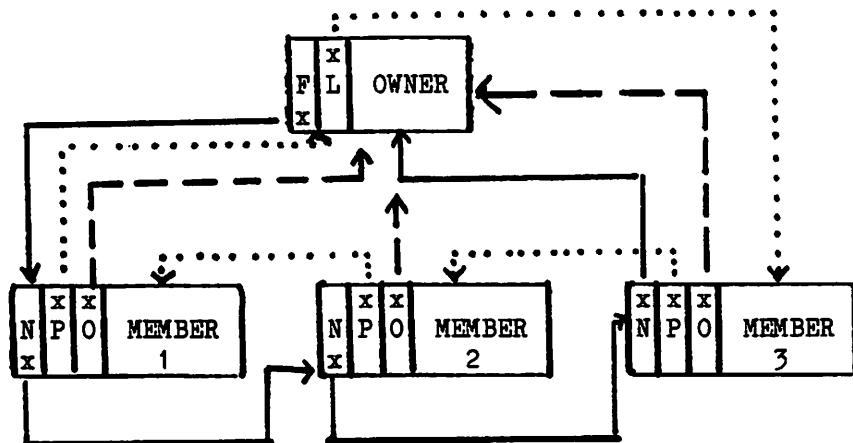


Figure 12 A chained set

OVERHEAD OF 4 BYTES PER
POINTED MEMBER
ie 12 bytes = Member
8 bytes = Owner

~~* FOR QB AUDMS PRIOR OWNER POINTERS *~~
~~AM YUN PLOC.~~

The Prior chain and the Owner pointers are optional but it is usual to have them.

Without Prior pointers:

1. FIND LAST/FIND PRIOR and a Set Order of LAST/PRIOR won't work in a Chained Set.
2. An ERASE statement may not physically remove a record but instead mark it as "Logically Deleted" in the line index. As a result, programmers will never see it again but the pointers in this dead record will keep the Chained Set mechanism working. However, the dead record wastes space and adds complexity to internal IDMS processes. It is better to avoid such situations.

Without Owner pointers, the pathway from Member to Owner can still be achieved but only by walking round the rest of the Set. This will happen automatically but could be much slower than using an Owner pointer if the Set has many Members. Since the pathway to the Owner will also be used by IDMS internally as well as when obeying a FIND OWNER, it is unusual to leave out Owner pointers.

Indexed Sets

This option of IDMS X is useful especially where:

1. A Set type has occurrences with a large number of Set Members and,
2. There is a requirement for quick selection of one particular Member record on the basis of a value for the Sort Key of the Set.

The Indexed Set mechanism is constructed by bringing the Pointers which would have formed the Next chain in a chained set together into an Index. Each entry in the index consists of:

- the Next pointer to the Member record
- the Sort Key value for the Member record.

USE OF SORTED SET

In addition, each Member record must have an Owner pointer for this Set type to allow the full Set Navigation facilities and the Owner record must have two pointers leading to the Index. These two pointers are used by the software for the two different purposes a Set Index can serve:

- Walking the Set. The so-called First Pointer is the start of a chain leading along the bottom level entries of the index giving access to each Member record in turn
- Selection of one specific Member record based on a known value for the Sort Key. The so-called Last Pointer leads to the top of the Index. From this point a search down through any levels that exist can occur, and the bottom level entry that points to the selected record can be reached rapidly.

Figure 13 shows the structure in outline.

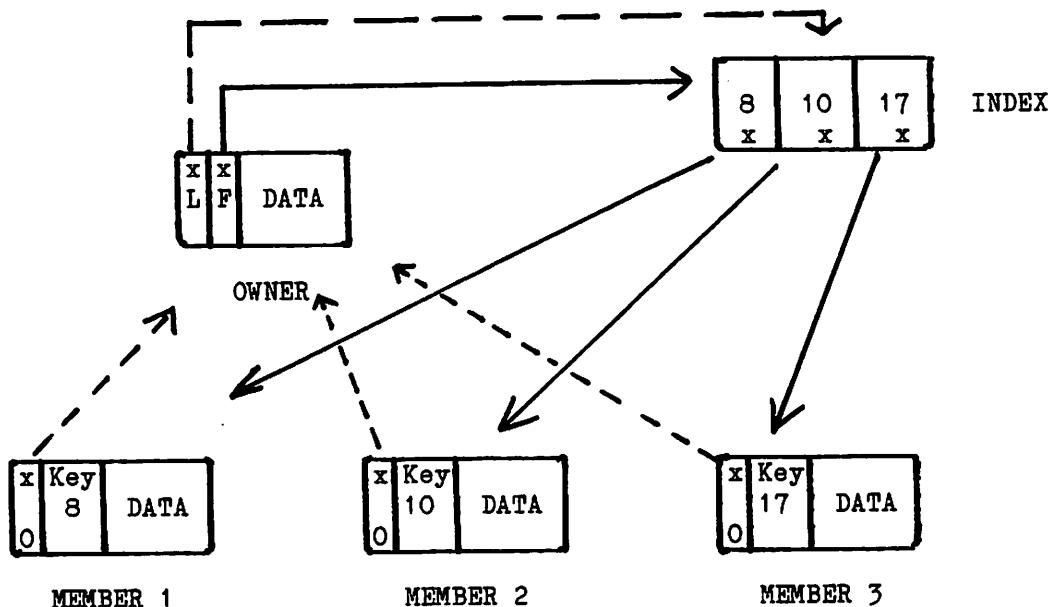
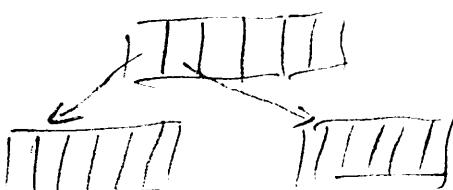


Figure 13 An Indexed Set

This is the structure when the Indexed Set has a Set order of Sorted.

WHEN INDEX CFS DO BIG A
SECOND LEVEL INDEX IS CREATED
etc.

4 - 18
DESIGNED DEFENDS 1W19.
INDEX SIZE



INDEX-TIDY sorts out the mess

NO265-5

If it has some other Set order then the Index will only contain Next pointers ie it will be a Pointer Array of Database Keys.

These other options for Set order are therefore little used when the Set type is Indexed.

RECORD PLACEMENT

Introduction

IDMS needs to be told how to place occurrences of each record type on to the backing store ie which page to try to use when a new record occurrence of the type is to be stored. The Page selected is called the TARGET page.

The Designer first selects the Area (or perhaps Areas, in IDMS X). The options are:

- anywhere in a stated Area or Areas
- within a specified Page range in a given Area. (This can be repeated for different Areas with varying Page ranges in each one, if IDMS X is used)
- anywhere within the same Area as an Owner of this record in a specified Set type.

The second stage is then to specify how IDMS is to choose the Target page within the range of pages allowed by the WITHIN clause. This is done by the Record's PLACEMENT clause in DSDL. The options are:

- *CALC - ON KEY, BY SET
- *PAGE DIRECT ON KEY, BY SET
- *DIRECT ON KEY, BY SET
- *VIA BY SET
- *SEQUENTIAL (IDMS X only) BY SET OR INDEX
- *SYSTEM DEFAULT BY SET

CALC and VIA placement are commonly used - other methods are rare and should only be used in order to achieve significant performance benefits.

The rest of this section examines how each Placement option works and then summarises the Overflow mechanisms.

CALC Placement

This is a mechanism providing Database Entry points like that used in a Random file. A built-in CALC algorithm calculates the Target Page number from the contents of some specified field or fields in the new record. The data must have been specified as a Keyname in the Schema. The process is shown in Figure 14.

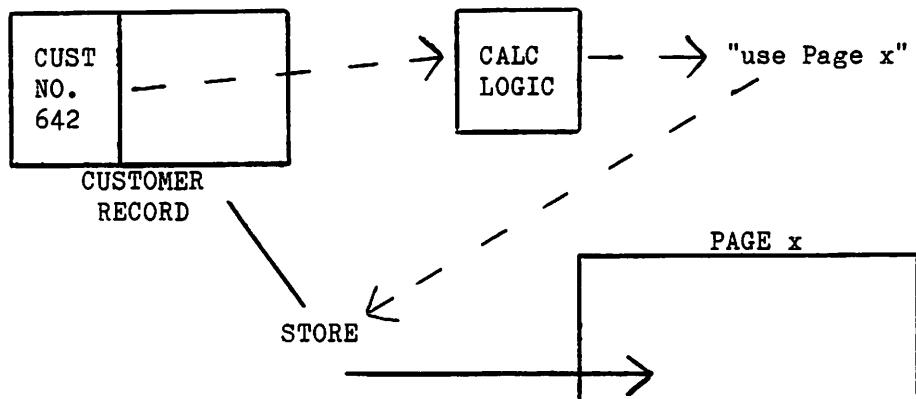


Figure 14 The CALC storage process

The CALC logic works as follows:

1. Mutilate the initial Key Value by an arithmetic process. This is to ensure that random clusters and gaps in the key population are eliminated so that the backing store is used as evenly as possible. Call the result N.
2. Divide N by the number of Pages in the Page range to be used eg if the record might go anywhere between page 2000 and 3500 divide by 1501. (If the IDMS X facility of multiple Areas for a record type is in use, the program at run-time will have already decided which Area is to be used for this record occurrence.)
3. Add the "Low Page Number" to the remainder eg in the above example add 2000 to the remainder which must be in the range 0-1500. The result (here a number in the range 2000 to 3500) is taken as the Target Page.

Although all this is automatic, there are three points to notice:

1. The process depends on the number of Pages in the Page range. If this has to be changed eg to make an Area bigger than existing CALC Records can no longer be found. They must therefore be unloaded (with all their connections to other records expressed as logical keys rather than as Database Keys) and then re-loaded using the new divisor. Re-organisation software exists to do this work.
2. The CALC algorithm may give a rather better, more even use of backing store if the divisor is an odd number.
3. The CALC algorithm is built into the IDMS run-time software. However, it is also available separately as a subroutine which can be called in COBOL. This allows data which is to be loaded or which is to be used to update existing records to be pre-sorted into the page sequence produced by CALC. This can give much less head movement and much greater efficiency during Realm scans.

Finally what happens if the Target Page is already full? The solution to this problem uses a device known as the CALC chain.

The CALC chain is an extra series of pointers on each page linking all the CALC records on that Page (whatever their type) to the Page Header. The records are sequenced along this chain so that all records of the same type are together in rising sequence of CALC Key Value. In order to achieve this, each CALC record has two extra pointers at the start which are the Next and Prior pointers on the CALC chain.

This mechanism serves three purposes:

1. It allows overflow of a new CALC record to be handled. The record is stored on some other page but linked onto the CALC chain of the correct Home Page - Figure 15.

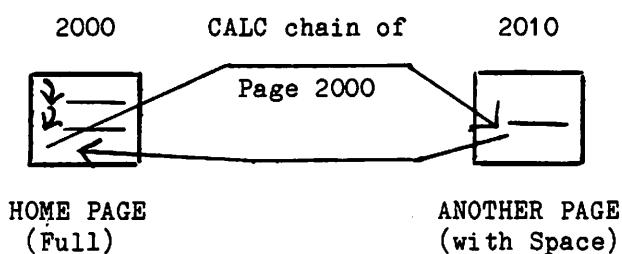


Figure 15 Overflow of CALC records

In such cases there will only be extra accesses when "CALC-ing" to the Page from outside the Database. The Database Key of the new record in Figure 15 will quote Page 2010 ie where the record actually is, so Set accesses will be as efficient as usual.

2. It speeds up the search for the record when the Page is in the Buffer. By following the CALC chain IDMS need only look at CALC records.
3. It supports the DUPLICATES option on the rare occasions when CALC records with the same Key Value are declared as legal in the Schema (Duplicates First or Last). "Duplicates First" means that a new record is placed on the CALC chain in front of any existing records with the same key. "Duplicates Last" means the new record will be placed on the CALC chain after any existing records with that key.

WHEN YOU KNOW BETTER
PAGE DIRECT Placement THAN CALC.

This is exactly like CALC but works with a User Written Algorithm instead of the built-in CALC routine. This Algorithm could be written in the program directly, but in order to allow the program to be independant of the Storage Schema detail may well be done automatically via the Database Procedure mechanism described in Chapter 5.

The Algorithm must generate a suitable Page number which is then passed to IDMS. From then on, the record is treated like a CALC record and is placed on the CALC chain of the home page.

This facility should only be used in the rare cases when the Designer has some special knowledge about the Key Population. This may then allow a better record distribution than CALC to be achieved. For example, it may be known that all or most possible Key Values are present, or present for a considerable sequence of Values. It could then be useful to organise an algorithm which stores records in a physical sequence identical to sorted Logical Key Value eg in Customer Number Order. This requires the Key to be identified to IDMS so that the records are sequenced in rising Key order along the CALC chain.

There is a programming interface which allows records in a given Page to be retrieved in CALC chain (ie Logical Key) order. This arrangement would then allow:

- quick access to a selected record
- sequential reporting in Logical Key sequence.

This would be the best of all possible worlds for record accessing requirements, although at the expense of some extra design and programming effort. In addition, the use of the CALC chain could not be concealed under any Program Independence mechanism ie the program would depend on a particular Storage Schema declaring the record type placement as Page Direct.

It is also possible to have a Page Direct Placement without specifying the Key to IDMS. In this case the program is able to control the Page a record goes to but the sequence along the CALC chain is dictated by the order of arrival as specified in the Duplicates clause (FIRST or LAST). This may or may not have some logical significance.

DIRECT Placement FORCE PLACEMENT ON INIT LOAD

This is a little used option, an ancestor of Page Direct. It requires the programmer to declare the full Database Key (Page AND Line Number) for a new record before it is stored. IDMS will then try to place the new record in that position. If the position is already in use, then the record is stored elsewhere and there is no overflow mechanism to say where it went. It can be difficult therefore to find the record again. However, the program is told what Database Key the new record was given.

This option may be used on rare occasions, eg during an initial Load to place a few important records at known positions through a currently empty Area, or to mimic a Serial file structure from an existing system.

*TRIES TO PLACE MEMBER
VIA Placement PHYSICALLY close to owner*

This option specifies that new occurrences of the record type are to be stored as close as possible to their Owner in a specified Set type. In effect the Owner and Members for a given Set occurrence of the Set type would then form a compact physical cluster within one Page or a number of adjacent Pages.

SET INDEX PAGE
In contrast if a Member record type has a Placement of CALC or Page Direct, the record occurrences are likely to be scattered all over the Page Range for the record type. As a result walking a Set will involve a disc access per record retrieved - Figures 16 and 17.

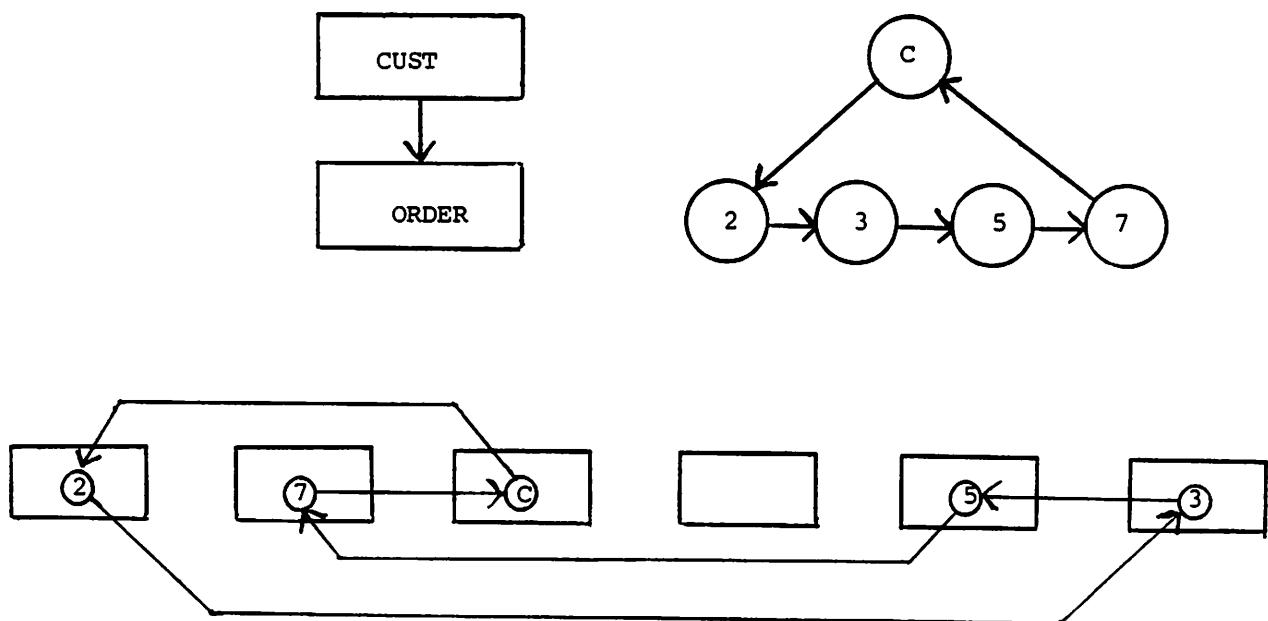


Figure 16 Record Placement with Owner and Member CALC

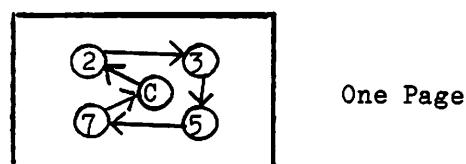


Figure 17 Record Placement with Owner CALC and Member VIA

NEED ALLOWED INDEX BY DIRECT ACCESS

The Designer has to strike a balance therefore between the number of Entry Point record types getting their Keys implemented by CALC or Page Direct Placement and the number of record types clustered through VIA Placement around an Owner in a particular Set type. He also has to decide WHICH Set type to specify if a VIA record type is a member of several Set types. Referring to Figure 18, the Designer could have chosen (among others) Placements for the ORDER record of:

- CALC
- VIA WANTS Set
- VIA LACKS Set.

The decision illustrated, (VIA WANTS) presumably matches the known patterns of access that must be supported.

CALC = RANDOM DISPLACEMENT
VIA = MEMBERS GROUPED
NO KEYED ACCESS
USE A RECORD INDEX
REFERENCE THE
OCCURRENCES OF REC.

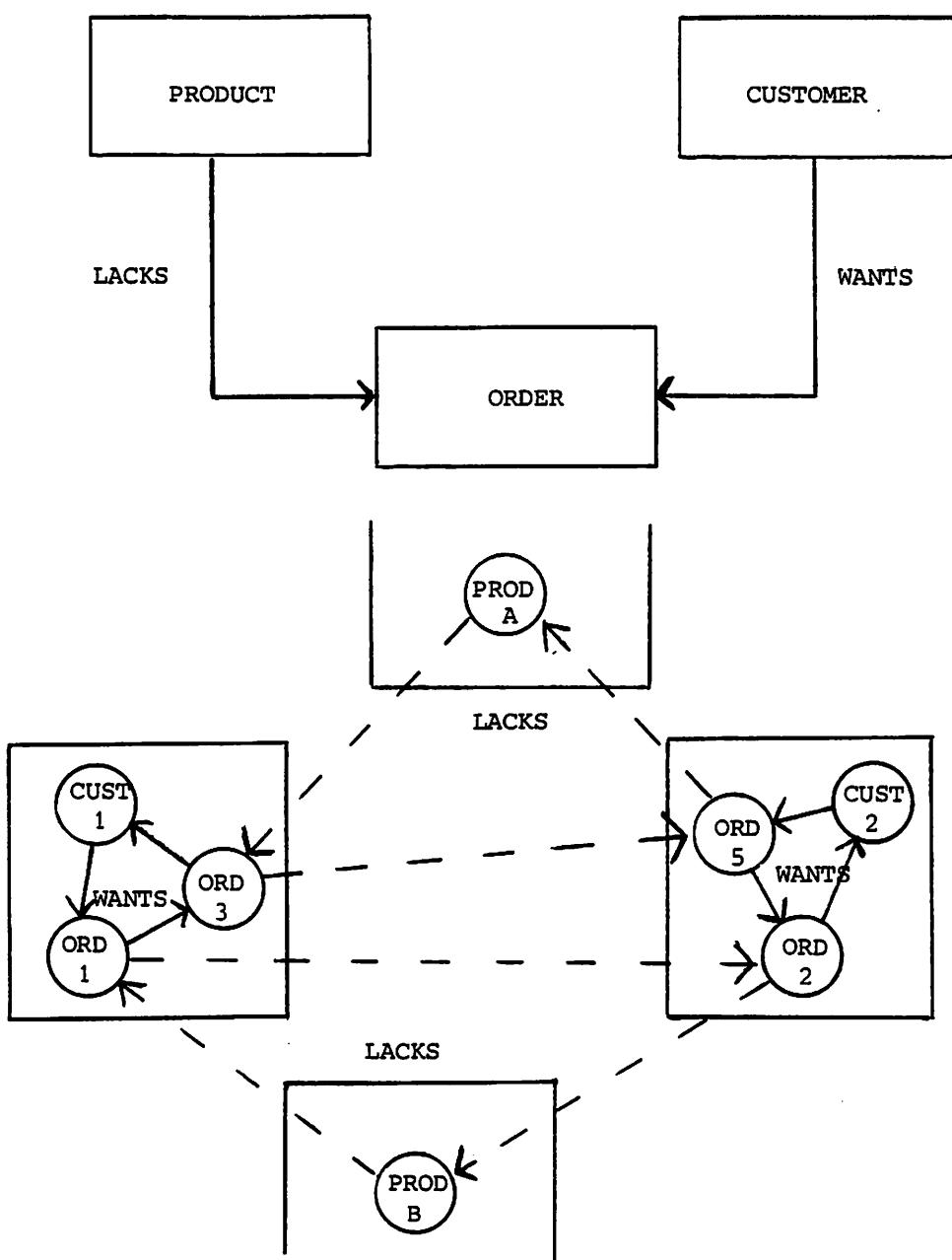


Figure 18 Effect of choice of VIA Set type on physical clustering

If one of the VIA options were used, Entry point access on one or several keys could still be provided by Record Indexes (IDMS X) or "1 to 1" Access records as described in Chapter 3. Figure 19 shows an example of the latter.

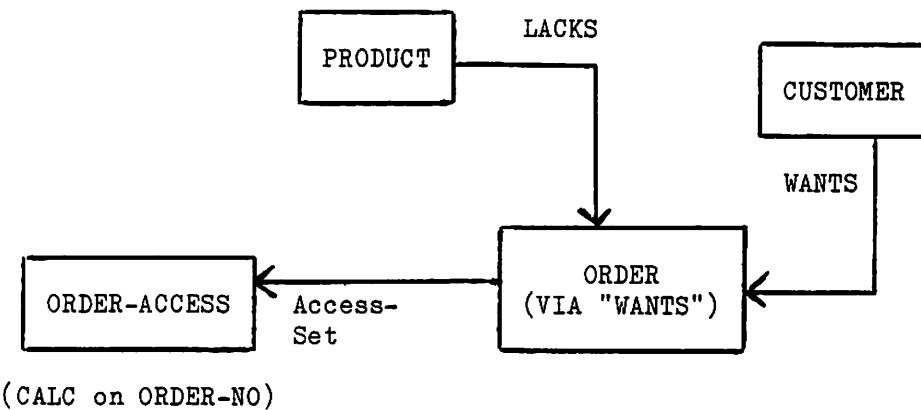


Figure 19 1 to 1 Access records

VIA records can be in a different Area from their Owner. In such cases the best clustering that can be achieved will occur ie all the Member records for the same Owner will be put in the same Page in the other Area. It seems illogical to require such a combination, but in fact when an exchangeable Medium is in use it allows the Member records to be put off-line for any application which requires to process the Owner records on their own.

Ability is sequential processing

SEQUENTIAL Placement *like an I.S. FILE*

This option is only available in IDMS X and only then if all record occurrences of the record type are to appear in a single Area.

The record occurrences will be sequenced on a nominated Order Key which will be supported by a Record Index. The actual mechanism is to place a new record as close as possible to the existing one with the next lowest Logical Key Value.

In order to get the records into a physical sequence which matches the Logical Key sequence as closely as possible, it is recommended that the following initial loading procedures are carried out:

- Physical
Index
SPREAD (AN)k*
- 1. Set the PAGE RESERVE to a high figure so that there is room for only one or two records per page.
 - 2. Load a Sorted sequence of representative records which covers the whole range of possible Key Population.
 - 3. Set PAGE RESERVE back to a normal value.

The result should be that subsequent processing, guided by the Record Index, will maintain the correct logical sequence with little loss of physical sequence.

SEQUENTIAL Placement provides facilities like those of an Indexed Sequential file combined with the possibility of using IDMS Set Navigation.

SYSTEM DEFAULT Placement

This leaves the placement of records to the system. At present such records are stored as near the beginning of their Page Range as possible.

*RECORD INDEX SHOULD USE THIS
OR "ONE OFF" RECS*

Overflow

There are two overflow situations:

1. First Level Overflow

In this case there is no room for the record on the Home Page selected by its Placement clause. In such circumstances, IDMS looks for space elsewhere in the Page Range for the record type. IDMS discovers which Pages have room by scanning SPACE MANAGEMENT PAGES which contain tables of the available space in the following Data Pages. The First Page of an Area will be a Space Management Page and there may be one or more further Space Management Pages later in the Area depending on the size of the Area and the size of the Page.

In standard IDMS where all Pages are the same size a Space Management Page will never contain data records. In IDMS X a SPACE MANAGEMENT clause can be used to specify a size for each Space Management record (= Table) in an Area. Any space left free in an IDMS X Space Management Page (ie Page size minus 40 minus Space Management record size = Spare Space) can be used for data.

Once IDMS finds a page with space, it stores the overflowed record there. It is linked back to its Home Page with the Home Page's CALC chain (CALC and Page Direct record types) or with the Set mechanism (VIA record types) or with the Record Index entries (Sequential record types). There is no connection to the Home Page for Direct or System Default records.

2. Second Level Overflow

This case means that there is no room anywhere in the record type's Page Range.

It is possible to declare a certain range of Pages at the end of an Area as Second Level Overflow Pages.

If specified, such Pages are used to store Second Level Overflow records. These will be connected back to their Home Page by the same mechanisms considered under "First Level Overflow".

Note: Space management entries are normally only updated if a page is more than 70% full. Generally, live databases should not be run with areas much more than 70% full since updates are likely to cause excessive usage of space management pages.

PAGE EXPAND or AREA EXPAND
WILL BUY TIME

If there is no range of Overflow Pages or if they get full, then an error is reported to the program when Second Level Overflow occurs. Recovery of the Area must then be considered. Such AREA FULL cases should not occur since:

- generous calculation of Area size is advised
- various Statistics available from IDMS should indicate that an Area full condition is likely, months before it would actually happen.

If Area Full has occurred, or is likely to occur, the most commonly used recovery and/or avoidance mechanisms are:

- the re-organisation software
- a Page expand facility making Pages in an Area larger (IDMS X only).

These solutions plus the Area sizing procedures which may prevent the problem altogether are considered in more detail in later chapters.

STORAGE SCHEMA DEFAULTS

Introduction

In theory, the Storage Schema need have no effect on the RESULTS of Application Programs - provided they are written to be independent of Storage Schema decisions. However, the Storage Schema decisions will have a considerable effect on the EFFICIENCY of Application Programs. This means that in practice the Storage Schema will be designed with care and that several may be produced for the same Schema eg to cover the requirements of different sites, to cover change, to cover Development V. Live requirements etc.

However, it is possible to default the whole Storage Schema or parts of it. Such a Storage Schema could be useful during initial testing of Storage-independent programs.

This section considers the major defaults for Files, Areas, Records, Sets and Indexes. Their main interest lies in the fact that they are also the defaults used if any particular section of a more normal Storage Schema is omitted.

File Defaults

If no File entries are specified, there will be one File allocated per Area with a Page size of 2048 bytes.

If no Page size is specified in the first File entry, a size of 2048 bytes is used. If no Page size is specified in any later File entry then the Page size used for the first File (that specified, or the default of 2048) will apply.

Area Defaults

If no Areas are specified, IDMS creates a single Area called IDMSAREA, Page Numbers 1001-2000. In this case there should be either no File entries or a maximum of one File entry.

The WITHIN clause which maps an Area to a logical file can be omitted if:

1. There is only one File entry. All Areas would then map to that File in an undefined sequence
OR
2. There are no File entries at all. Each Area is then mapped to a File of the same name as the Area - Page size 2048 bytes
OR
3. A File entry exists with the same name as the Area.

Record Defaults

Introduction

If a Schema record is not mentioned in the Storage Schema then a Record entry is created with all the following defaults. The defaults may also be used individually.

Record Identifier

IDMS allocates Record Identifiers automatically starting at 100. However, ALL Record entries must default the Record Identifier for Data and Index records if this is to happen.

*EXPLICIT NAME OTHERWISE PDS
DATA*

Placement

1. If a Record has a Key which is not an Order Key (ie no ASC or DES entry) then it will be given a Placement of CALC. If (in IDMS X), there are several non-order keys, one will be used arbitrarily for CALC Placement (and the rest implemented by Record Indexes).
2. If this does not apply and the Record is an Automatic Member of a Set, it will be given a Placement of VIA that Set. If one of several Set types could be used, one is chosen arbitrarily.
3. Otherwise Placement will be SYSTEM DEFAULT unless:
 - IDMS X is in use, and
 - the Record has an Order Key.

In such cases SEQUENTIAL will be used instead.

Within Area Clause

Occurrences of the Record type will go to the first Area defined in the Storage Schema.

Minimum Root

If the record is shown to require fragmentable format (variable length or it has a Minimum Fragment clause or may be Compressed) then Minimum Root is the control length ie that part of the record preceding the data fields plus all Record Key and Set Order Key fields.

*(NEAR TO REC)
5N 5 12E*

Minimum Fragment

~~If the record is fragmentable, a Minimum Fragment length of 4 Bytes is assumed. This is likely to be extremely inefficient.~~

Set Defaults

Introduction

If a Schema Set type is not mentioned in the DSDL, then a default entry is created using the defaults for Mode and Pointers described below. This will only be possible if the record types involved are not mentioned in other Set types with explicitly defined Pointer Positions.

Mode Default

The Set will be implemented as a Chained Set unless (in IDMS X) the Pointer clause includes the Keyword INDEX.

Pointers Default

If the Pointers required in a Set type are not specified explicitly, then FIRST and LAST Pointers will be allocated in the Owner record. The Member records will be given NEXT, PRIOR and OWNER Pointers if the Mode is Chain, but OWNER Pointers alone if the Mode is Index.

Index Defaults

Record Identifiers are allocated in descending order from 9999. System Default Placement will be used for each Record Index and VIA Placement for each Set Index. The Area used will be that containing the Owner records for a Set Index, and that containing the occurrences of the relevant Record type for a Record Index. In the latter case, the Area will be chosen arbitrarily if several Areas could contain occurrences of the Record type.

Finally, the Index Record size will be the largest that can be fitted into the relevant Pages involved.

Note: For live databases it is normally better to specify all physical properties in full rather than to make use of the defaults.

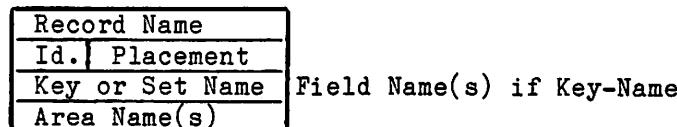
DOCUMENTATION

Storage Diagrams

These show the detailed design decisions represented by a particular Storage Schema. The conventions cover the representation of Record types, Record Indexes and Set types (Chained and Indexed).

Figure 20 shows the conventions for Record types.

1. CONVENTIONS



2. EXAMPLE

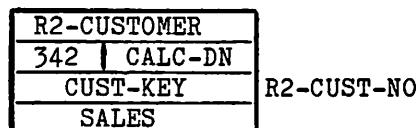
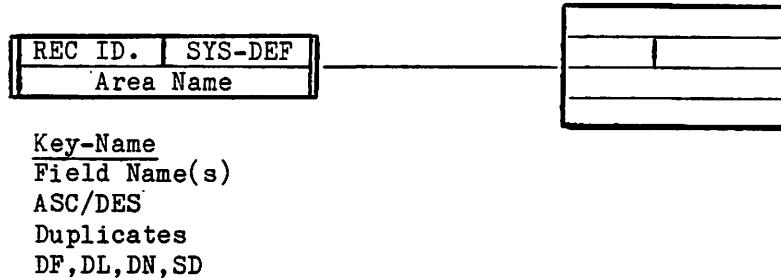


Figure 20 convention for Record types

Record Indexes can be shown by a Box as in Figure 21.

1. CONVENTIONS



2. EXAMPLE

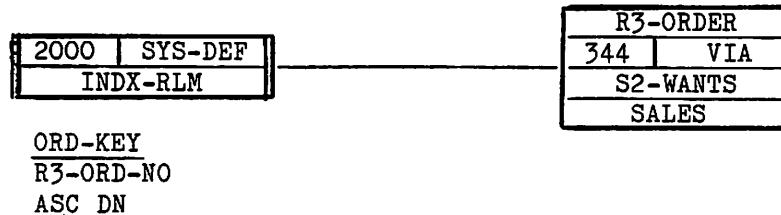
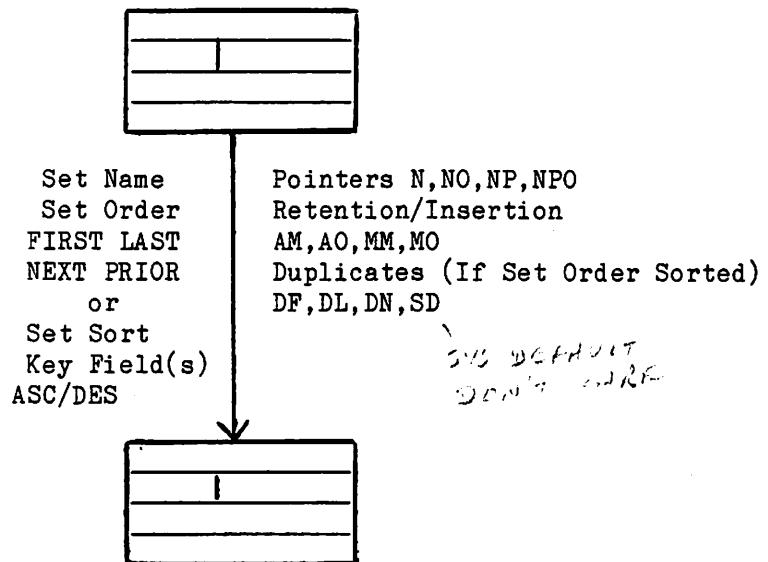


Figure 21 Convention for Record Indexes

The Set decisions can be shown beside the line connecting the Owner and Member Boxes. No further complications are involved if the Set is a Chained Set - Figure 22.

1. CONVENTIONS



2. EXAMPLE

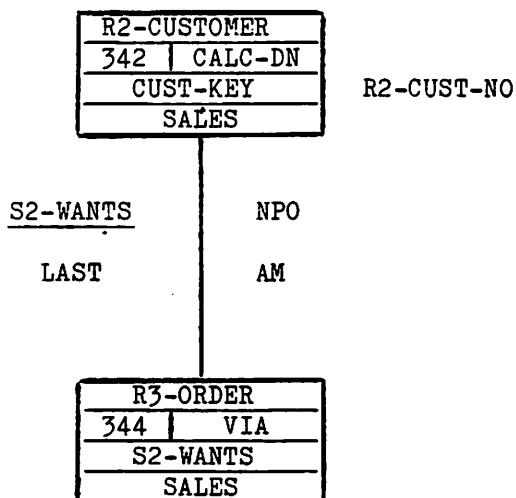
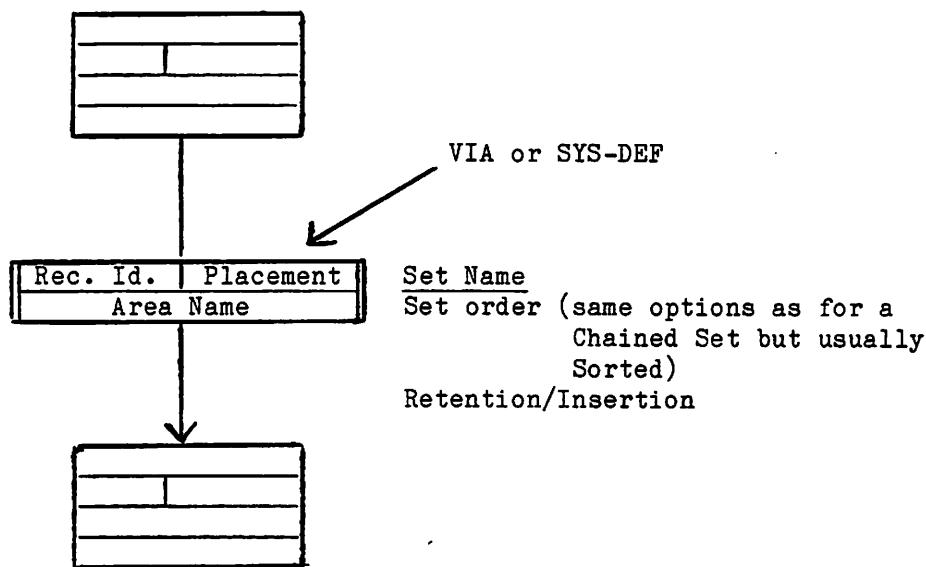


Figure 22 conventions for Chained Sets

If the Set is an Indexed Set an Index box can be shown on the line between the Owner and Member boxes - Figure 23.

1. CONVENTIONS



2. EXAMPLE

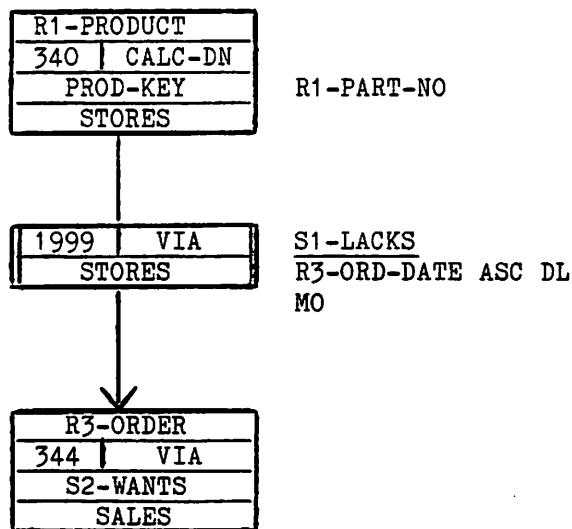


Figure 23 Conventions for Indexed Sets

A complete example is shown in Figure 24.

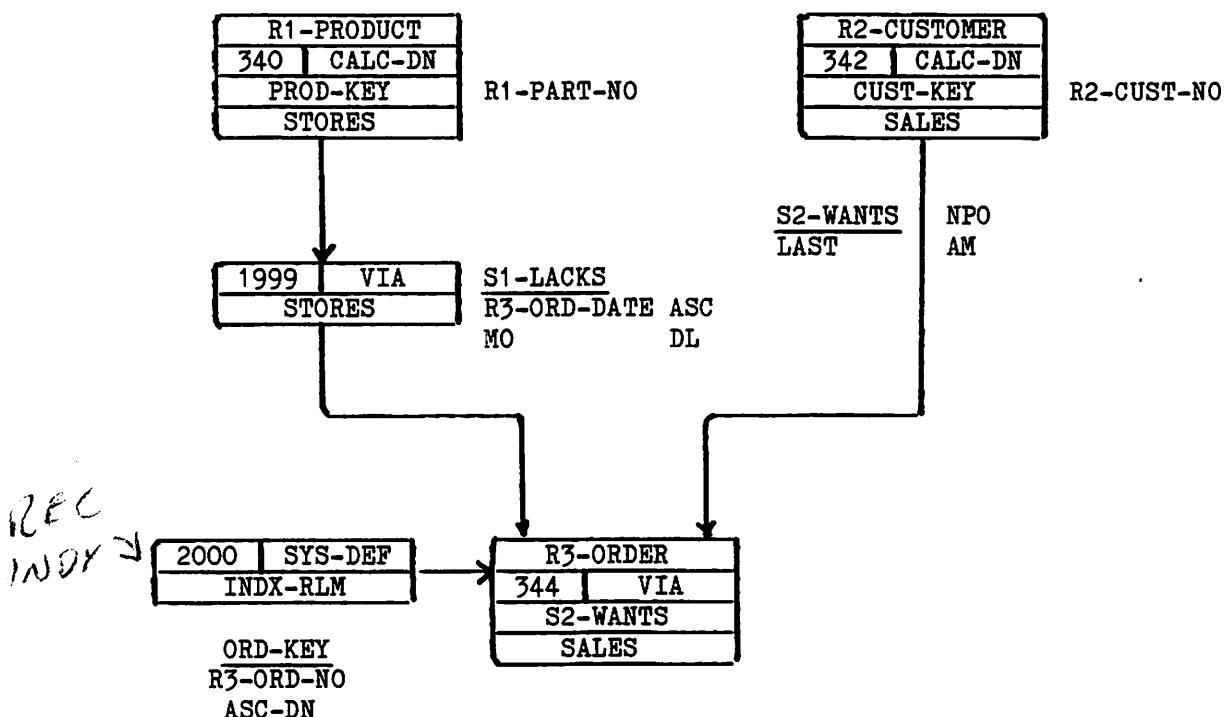


Figure 24 An example Storage Diagram

The Data Dictionary System

Full documentation of a Storage Schema can be held in the DDS.

DDS documentation requires a Storage Schema Element to be inserted and linked to the relevant Schema, File, Area, Index Set and Record Elements. The same subsidiary Elements can be used as were linked to the appropriate Schema Element but extra properties are needed eg *RECORD-ID on Records. Where several Storage Schemas are to be documented for one Schema, Separate Set and Record Elements for each one will be advisable.

APPENDIX

1. DSDL Example

STORAGE SCHEMA CASHSS
FOR CASHFLOW.

FILE SALES-FILE
PAGE 4096.

FILE STORES-FILE
PAGE 4096.

FILE INDX-FILE
PAGE 2048.

AREA SALES
RANGE 1001 1750
WITHIN SALES-FILE
FROM 1 750.

AREA STORES
RANGE 2001 2999
WITHIN STORES-FILE
FROM 1 999.

AREA INDX-RLM
RANGE 4000 4499
WITHIN INDX-FILE
FROM 1 500.

RECORD R1-PRODUCT
RECORD ID 340
PLACEMENT CALC USING PROD-KEY
WITHIN STORES.

RECORD R2-CUSTOMER
RECORD ID 342
PLACEMENT CALC USING CUST-KEY
WITHIN SALES.

RECORD R3-ORDER
RECORD-ID 344
PLACEMENT VIA S2-WANTS
WITHIN SALES.

```
SET S1-LACKS
    MODE INDEX.

SET S2-WANTS
    MODE CHAIN.

INDEX KEY ORD-KEY
    INDEX ID 2000
    PLACEMENT SYSTEM DEFAULT
    WITHIN INDX-RLM
    MAXIMUM LENGTH 1986.

INDEX SET S1-LACKS
    INDEX ID 1999
    PLACEMENT VIA
    WITHIN STORES
    MAXIMUM LENGTH 816.
```

2. DDCL Example

It is assumed that the DDCL in the Appendix to Chapter 3 has already been fed to DDSPROCESS.

```
INSERT STORAGE-SCHEMA CASHSS
*SCHEMA CASHFLOW
*FILES FILE SALES-FILE
    FILE STORES-FILE
    FILE INDX-FILE
*AREAS SALES STORES INDX-RLM
*RECORDMAP R1-PRODUCT/R1-PRODUCT-S
    R2-CUSTOMER/R2-CUSTOMER-S
    R3-ORDER/R3-ORDER-S
*SETMAP S1-LACKS/S1-LACKS-S
    S2-WANTS/S2-WANTS-S
*INDEXES ORD-KEY-IND S1-LACKS-IND

INSERT FILE SALES-FILE
*PAGE-SIZE 4096

INSERT FILE STORES-FILE
*PAGE-SIZE 4096

INSERT FILE INDX-FILE
*PAGE-SIZE 2048

INSERT AREA SALES
*RANGE 1001 1750
*WITHIN FILE SALES-FILE
    FROM 1 750

INSERT AREA STORES
*RANGE 2001 2999
*WITHIN FILE STORES-FILE
    FROM 1 999

INSERT AREA INDX-RLM
*RANGE 4000 4499
*WITHIN FILE INDX-FILE
    FROM 1 500

INSERT RECORD R1-PRODUCT-S
*RECORD-ID 340
*PLACEMENT CALC USING PROD-KEY
*WITHIN STORES
```

INSERT RECORD R2-CUSTOMER-S
*RECORD-ID 342
*PLACEMENT CALC USING CUST-KEY
*WITHIN SALES

INSERT RECORD R3-ORDER-S
*RECORD-ID 344
*PLACEMENT VIA R2-WANTS
*WITHIN SALES

INSERT SET S1-LACKS-S
*MODE INDEX

INSERT SET S2-WANTS-S
*MODE CHAIN

INSERT INDEX ORD-KEY-IND
*INDEX-FOR KEY R3-ORDER.ORD-KEY
*INDEX-ID 2000
*PLACEMENT SYSTEM DEFAULT
*WITHIN INDX-RLM
*MAXIMUM-LENGTH 1986

INSERT INDEX S1-LACKS-IND
*INDEX-FOR SET R1-LACKS
*INDEX-ID 1999
*PLACEMENT VIA
*WITHIN STORES
*MAXIMUM-LENGTH 816

CHAPTER 5

USING THE DATA

INTRODUCTION

This chapter examines the logical structure of IDMS programs - the Subschema (the program's view of the database) - Success Units - and the DML verbs that may be used in them.

THE IDMS SUBSCHEMA

A subschema defines the view of the database that is seen by an application program. The view may be the same as that represented by the schema, more commonly, though, it will be a subset.

The view or restricted view of the data, provided by the subschema is defined in terms of REALMS, RECORDS and SETS.

Realms

An IDMS realm is a collection of records. This can be:

- all the records in a physical area of the database. In this case the realm corresponds to an area
- all occurrences of a record type regardless of physical areas.

All the necessary realms should be included in the subschema.

The usage modes with which programs may READY those realms may be restricted, eg to prevent updating.

Records

Not all records in the included Realms need be included in a subschema. Users of the subschema will not be able to access or update any data held in those records.

Not all data items need be included. Those included may be specified in a sequence other than that defined in the schema, ie the physical format.

In IDMS-X where a record can have more than one record key associated with it, some record keys can be excluded. However, the first key (according to the schema definition) of records included will automatically be present in the subschema. Keys excluded will prevent access to their record occurrences by those keys.

Where order-keys can be used, sequential access can be prohibited in the subschema, even though direct access is permitted.

Sets

Not all set types need be included. Users will not be able to access data via those sets nor to connect or disconnect records to or from those sets.

DML Verbs

Specific DML verbs may be prohibited for a record or set type. Thus retrieval access may be allowed but not update access.

A subschema is defined in Subschema DDL an example of which is given in an appendix to this chapter.

STRUCTURE OF A COBOL IDMS PROGRAM

There is little difference between the structure of a COBOL IDMS source program and one written for conventional files.

Data Division Entries

The Data Division is laid out as in Figure 1.

```
DATA DIVISION.  
SCHEMA SECTION.  
DB    Subschema name WITHIN Schema name.  
FILE SECTION.  
      (Non-IDMS file definitions)  
WORKING-STORAGE SECTION.  
      (intermediate work areas).
```

Figure 1 Data Division of a COBOL IDMS program

The Schema section and DB line call for the particular view of the Database provided by the named subschema. All the necessary input/output record areas and field descriptions are brought in to the end of the working storage section of the Source Program, by IDMS.

Procedure Division Entries

The differences here are that there are a number of new verbs, known as the DML verbs, available for Database handling like STORE and OBTAIN.

Also the procedure division will be structured, comprising one or more success units.

SUCCESS UNITS

A Success Unit should be a sequence of code which leaves the Database in a consistent state.

- A. Request required Realms
- B. Check Customer Credit limit
- C. Check Stock availability and reduce by amount of Order
- D. Create new Order and link to Customer and Product
- E. Finish with the Realms

Figure 2 Sequence of logic in a Success Unit handling a Customer Order

Figure 2 shows an example of such a sequence of Code. The Success Unit approach to this would be:

- EITHER the end is reached. In this case the Database is consistent and the processing will never need to be repeated on the Order in question. Even if the Database is destroyed, the results of this Success Unit can always be retrieved by the Recovery process
- OR the end is not reached because of some hardware or software failure. In this case because the Database may be inconsistent, all updates carried out in the Success Unit will be reversed, ie the Database is **always** returned to the state it was in at the start of the Success Unit. As an example of inconsistency, consider a failure just after Stage C in Figure 2. This leaves the Product stock balance reduced, but no new Order yet created to justify the change.

Figure 3 shows the two possibilities.

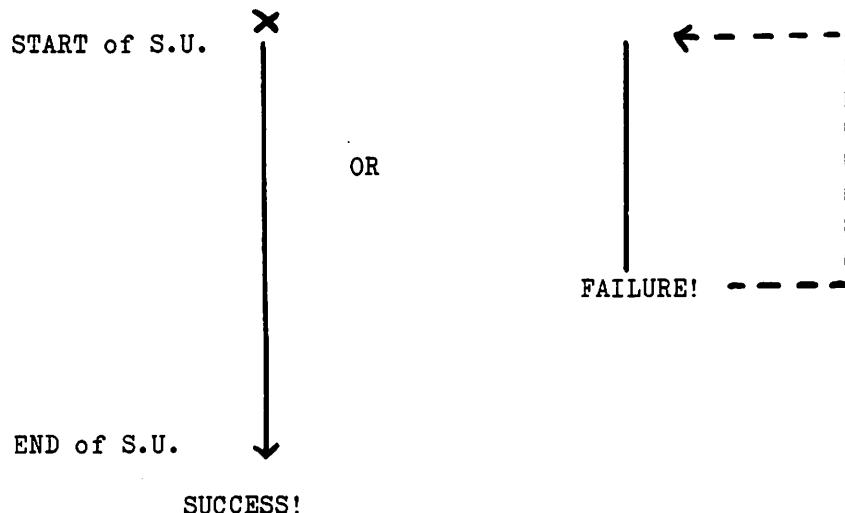


Figure 3 The two possible results of a Success Unit

At the Programming level Success Unit boundaries are marked by the two verbs which begin and complete a task - READY and FINISH.

At the Design level a Success Unit corresponds to:

- a TP message pair
- a complete Batch program
- a portion of a Batch program.

A success unit should be kept as short as possible since it is:

- the unit of recovery
- a unit across which locks are held.

REQUESTING USE OF THE DATABASE

Before a program can gain access to the database, all the required realms must be requested by the program. This is done by use of the READY verb. All the realm requirements of a success unit must be given together at or near the start of the procedural code. The rule is that no more READYS will be accepted after the first DML processing verb. Thus the READY or group of READYS denotes the start of a success unit. At the end of database processing for a success unit, a FINISH verb must be given. This completes the processing task and ends the success unit.

The Ready Verb

The syntax is shown in Figure 4.

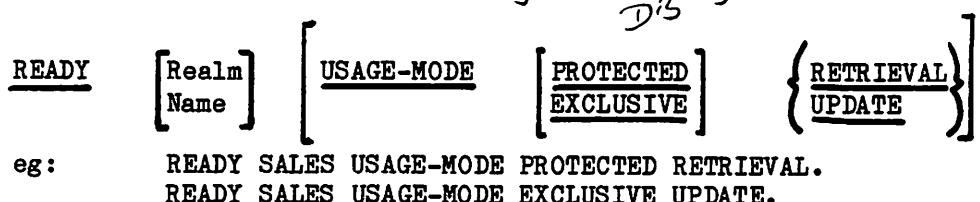


Figure 4 Syntax of READY

The PROTECTED/EXCLUSIVE clause controls the Locking mechanism in IDMS. This mechanism allows the Success Unit to control who else can access the Realms it is using. The mechanism is powerful and may lock some other users out either at the Area or Page level.

The first requests the SALES Realm for PROTECTED RETRIEVAL processing which means "I am only going to Read from this Realm, I don't mind if other Success Units read at the same time but I do not want anyone to be updating".

The second requests the SALES Realm for EXCLUSIVE UPDATE processing which means "I am going to Read and Write in this Realm, no-one else may use it for the duration of my Success Unit".

The programmer declaring his Success Unit requirements in this way can then leave the work of meeting these requirements entirely to IDMS.

The FINISH Verb

The syntax is simply the word FINISH, which will cause any locks set during the Success Unit to be freed. From that moment any changes made to the Database by the Success Unit are committed and will not be undone by the automatic Recovery mechanisms.

CURRENCY

Introduction

The logic of all the DML Verbs between READY and FINISH in a Success Unit is based on the use of CURRENCY INDICATORS. A Currency Indicator is a Field in the Subschema Tables containing the Database Key of a Record recently processed.

The following Currency Indicators exist in any Subschema Tables:

- Current of Run Unit (CRU), contains the Database Key of the Record the Success Unit is processing at the moment
- Current of Set type, one for each Set type in the Subschema. Each contains the Database Key of the most recently accessed record in that set
- Current of Record type, one for each Record type in the Subschema. Each contains the Database Key of the most recently accessed record of that type
- Current of Realm, one for each Realm in the Subschema. Each contains the Database Key of the most recently accessed record in that realm.

Example

Figure 5 shows the Records and Sets in a Subschema.

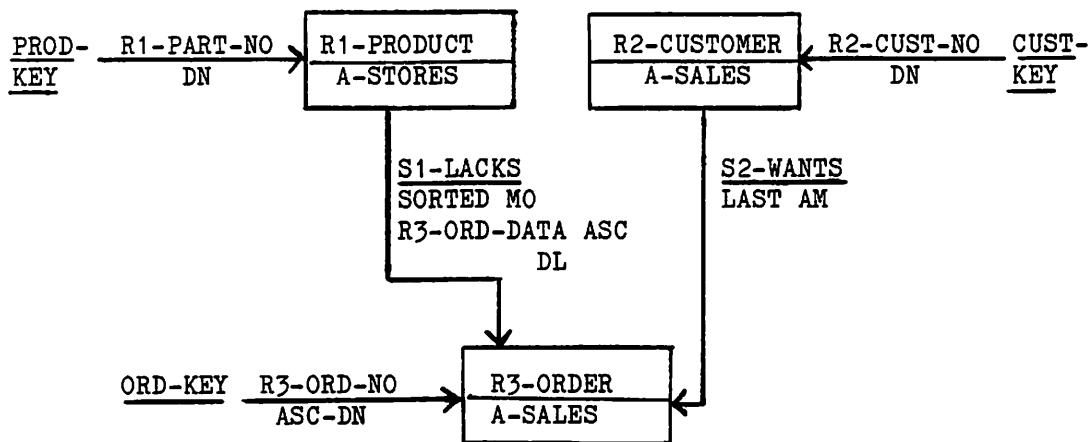


Figure 5 A simple order processing subschema

The Currency Indicators for this Subschema are listed in Figure 6 with some suitable settings (logical keys are shown for clarity rather than the Database Keys that would really be there).

How Currency Indicators Change

A Record becomes CRU when it is worked on by either of the three commands FIND (Locate a record), OBTAIN (Read a Record) or STORE (Write a New Record).

When a Record becomes CRU, it also becomes current of the following other Currency Indicators:

- all Set types it is physically in
- it's Record type
- it's Realm.

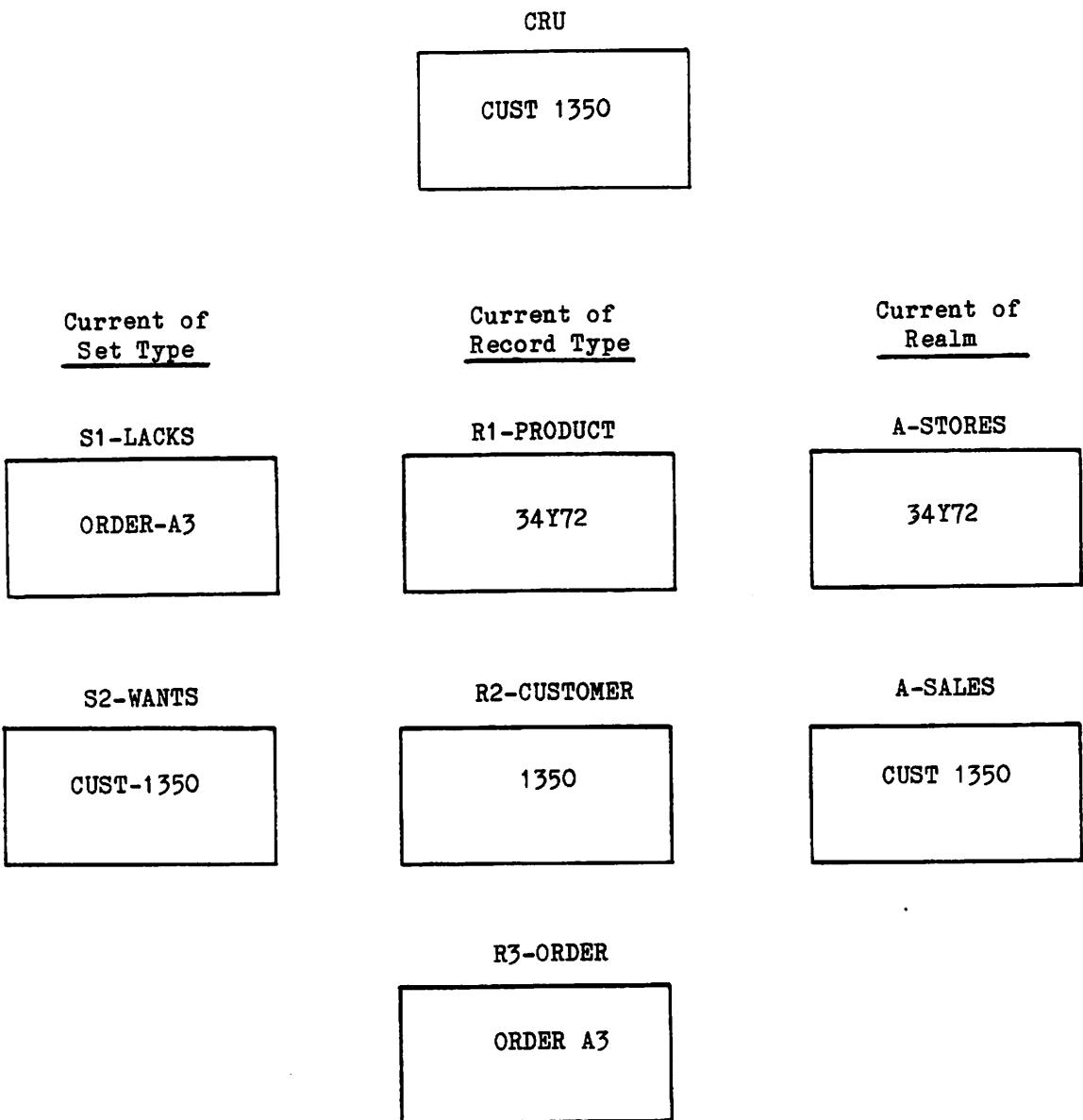


Figure 6 Currency Indicators for the Subschema

The Importance of Currency Indicators

Currency Indicators dictate the precise effect of each individual DML Verb, eg which Record Occurrence is actually ERASE-d (that indicated by the current of record type), which Customer a new Order is automatically connected to by STORE (that indicated by the Current of S1-NEEDS Set Indicator) etc. So the programmer must understand:

- how Currency Indicators are set and changed
- which Currency Indicators particular DML Verbs look at.

If he understands these points DML programming is easy. If he does not, he will make serious mistakes like deleting the wrong Record or connecting an Order to the wrong Customer.

Even when the general principle is understood, ie that OBTAIN and STORE set the Indicators while the other DML Verbs use them, the programmer should still check his Currency logic with great care in each new Success Unit he writes. In particular watch the effect of ERASE and DISCONNECT in nullifying Currency settings.

These points are considered further in the following Sections which deal with Retrieving data and Updating data in an IDMS Database.

RETRIEVING DATA FROM THE DATABASE

Introduction

Records in the database are located using the FIND verb. This has the general format of:

FIND Record-selection-expression

The record selected is brought into the program's buffer area, but its contents are not available to the programmer. In order to see the record contents a GET verb is used. This will bring the located record into the programs work area for subsequent processing by normal COBOL statements. The format is:

GET record-name

Alternatively, the OBTAIN verb may be used, which has the combined effects of a FIND followed by a GET. Thus OBTAIN records to be read from the database. It has the same format as FIND and sets currency indicators in the same way, ie the record becomes CRU, current of its record type, current of its realm and current of all the sets to which it is currently connected.

The Record selection process may be of two types

- ENTRY POINT. We may wish to locate or read a specific record, knowing its identifying key.
- NAVIGATION. We want to move on to the next record on a given access path from our current position in the Database.

Entry Point Access

Using Logical Keys

When the logical key(s) of a record is known, then this format applies:

{FIND } ANY recordname [USING keyname]
{OBTAIN }

Before using this verb, the logical key value(s) must be loaded into the appropriate field(s) in the program's record area. If the USING option is omitted, then IDMS assumes the key to be the primary key, eg:

```
MOVE XIN-CUSTNO TO R2-CUSTNO.  
OBTAIN ANY R2-CUSTOMER USING CUST-KEY  
ON DB-REC-NOT-FOUND GO TO L-ERR.
```

Note. IDMS provides comprehensive error checking and trapping facilities which should be employed after every DML call to the database. The programmer will then know if IDMS has or has not been able to carry out the required task.

Start of a Realm

A program can gain access to an area by finding the first record in a realm. IDMS will select the record with the lowest database key in that realm. Having gained entry, it will then be possible to scan all the records or all the records of that type, in the realm. The syntax is:

{FIND } {FIRST} recordname WITHIN realm-name.
{OBTAIN } {LAST }

If LAST is used, then the record with the highest database key in the realm is found

MAY NOT BE ON FIRST PITCH
COULD BE ANYWHERE.
CAN USE DIRECT PLACEMENT TO
FORGET IT.

Navigational Access

This type of access moves on from a starting point WITHIN the Database to the next record required. The navigation may be within a Set or a Realm. The appropriate Current of Set or Current of Realm indicators are used by IDMS to establish what the starting point is to be.

Navigation Within a Set

Navigation will either involve moving around the set, accessing member records or accessing the Owner from any member record. The syntax for accessing member records is:

{ FIND OBTAIN }	{ FIRST NEXT PRIOR LAST }	recordname WITHIN setname.
--------------------	------------------------------------	----------------------------

Eg:

OBTAIN NEXT R3-ORDER WITHIN S2-WANTS.

This will use the Current of Set type indicator for S2-WANTS to decide:

1. Which occurrence of the S2-WANTS set is to be processed.
2. Where are we in that set at the moment.

It will then follow the next pointer on the current record, for chained sets, reading in the next R3-ORDER record. For indexed sets, the next database key entry in the index will be used to retrieve the R3-ORDER record it points to.

The sequence of DML shown in Figure 7 will process all the Orders for Customer 82, assuming the Database shown in Figure 5. Such a processing pattern is very common in IDMS and is called WALKING the SET.

```
MOVE 82 TO R2-CUST-NO.  
FIND ANY R2-CUSTOMER USING CUST-KEY  
    ON DB-REC-NOT-FOUND GO TO L-ERR.  
L-NAVIGATE.  
    OBTAIN NEXT R3-ORDER WITHIN S2-WANTS  
        ON DB-END-OF-SET GO TO L-NEXT.  
        - process order record -  
    GO TO L-NAVIGATE.
```

Figure 7 DML to walk a set

As processing continues, the currency indicators will be changed by IDMS. Figure 8 shows the updated currency settings after the first R3-ORDER record has been read. Note how the currency settings have changed from their earlier state in Figure 6.

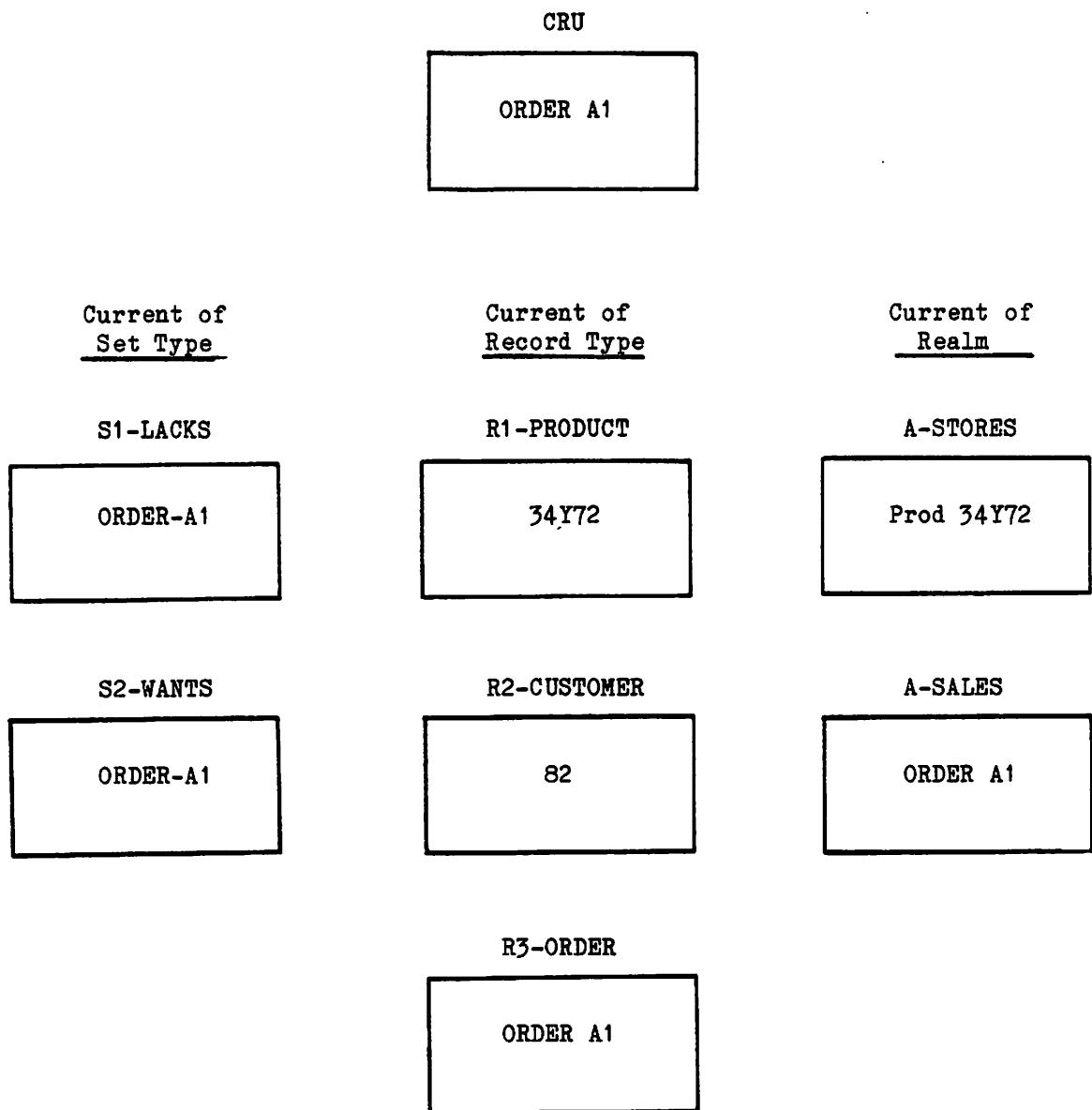


Figure 8 How Currency Indicators change

The second type of set Navigation is moving from a member record to the Owner. The syntax is

{ FIND }
OBTAIN }

OWNER WITHIN setname.

COUNT
BTS OVERHEAD
IF NO OWNER
POINTERS.
WANTS WHICH SET.

eg:

OBTAIN OWNER WITHIN S2-WANTS.

The current of S2-WANTS set will be used as before to determine which WANTS set occurrence is involved and the OWNER, once found becomes CRU etc.

Navigational Access Within a Realm

This approach usually involves processing a Realm like a Serial file, reading all the Records or all of a type, in the physical order in which they are stored, ie page by page.

This might be done if a Report is required on all Customers or if there is a large batch of Updates to be applied to the Realm at one time. Loading is an example of this second case.

The syntax for this type of access is:

OBTAIN {
NEXT
PRIOR
FIRST
LAST }

Record-Name WITHIN Realm-Name

COULD BE LOADS OF I/Os
SERIAL SEARCH THRU AREAS

eg:

OBTAIN NEXT R2-CUSTOMER WITHIN A-SALES.

A sequence to look at every Customer Record in the Sales Realm is shown in Figure 9.

```

    OBTAIN FIRST R2-CUSTOMER WITHIN A-SALES
    ON DB-END-OF-REALM GO TO ERR-1.
        Process first
        Customer Record's data
    NAVIGATE-REALM
        OBTAIN NEXT R2-CUSTOMER WITHIN A-SALES
        ON DB-END-OF-REALM GO TO END-IT
            Process next
            Customer Record's data
        GO TO NAVIGATE-REALM.
    END-IT

```

ERR-1

Figure 9 Navigating a Realm

Retrieval by Record Order

In IDMS-X, where the designer has the choice of implementing record indexes, retrieval of records is allowed in record key order. Providing that the subschema used by a program allows record order processing for particular record types, the program can issue the following syntax:

$\{$ FIND OBTAIN $\}$	$\{$ FIRST NEXT PRIOR LAST $\}$	<i>SELECTIVE INDEX</i> recordname USING keyname. <i>USE RECORD INDEX</i>
--------------------------	--	--

eg:

FIND NEXT R3-ORDER USING ORD-KEY.

The contents of the indicator for current of record type R3-ORDER will determine the starting point in the record index. IDMS will then look at the next entry in the ORD-KEY record index which in turn points to the required R3-ORDER record occurrence

Figure 10 shows the DML coding to retrieve ten R3-ORDER records in key sequence starting at the one with a key of "A100".

```
MOVE "A100" to R3-ORD-NO.  
OBTAIN ANY R3-ORDER USING ORD-KEY  
    ON DB-REC-NOT-FOUND GO TO L-ERR.  
MOVE 1 TO W-COUNT  
L-NEXT.  
    OBTAIN NEXT R3-ORDER USING ORD-KEY  
        ON DB-END-OF-KEY GO TO L-PARA.  
    ADD 1 TO W-COUNT  
    IF W-COUNT < 10  
        GO TO L-NEXT.
```

Figure 10 DML to access order records via a record index

Enquiry DML - Summary

Introduction

This section considers three simplified examples of DML navigation summarising the material considered so far.

Walking a Set

```

MOVE "SMITH" TO R2-CUST-NO.
FIND ANY R2-CUSTOMER USING CUST-KEY.
→ OBTAIN NEXT R3-ORDER WITHIN S2-WANTS
    ON DB-END-OF-SET ——————
    ELSE —————— ↓

```

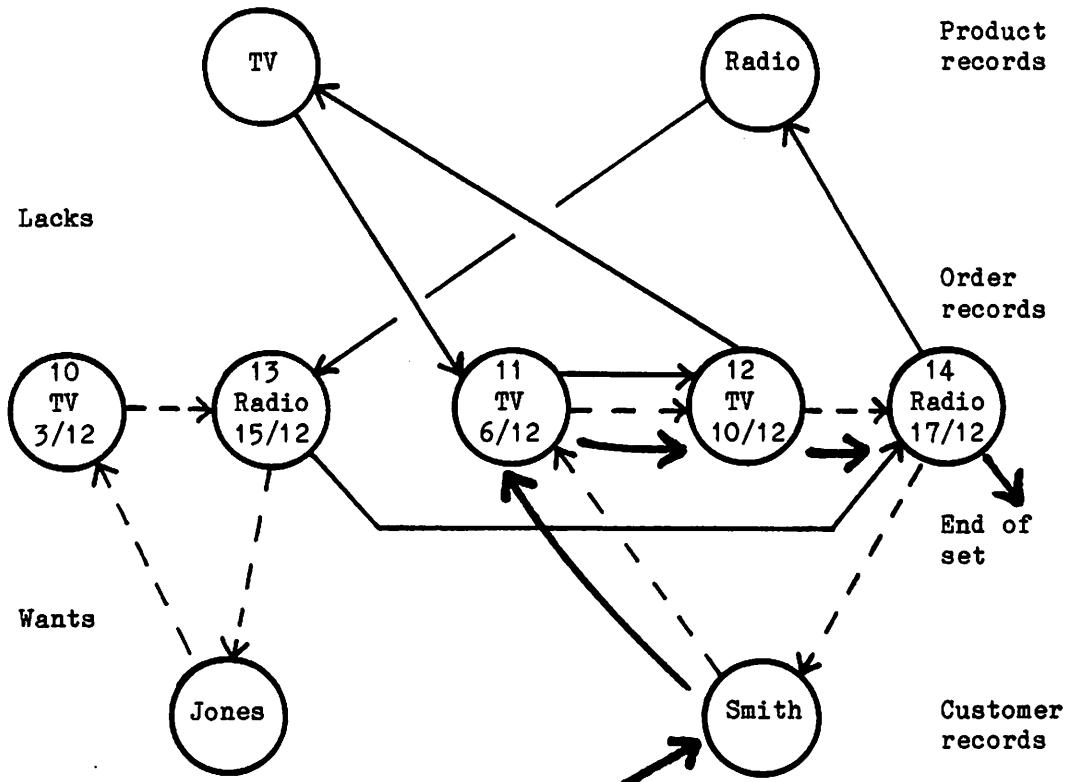


Figure 11 DML sequence 1 - all orders from Smith

Realm Navigation

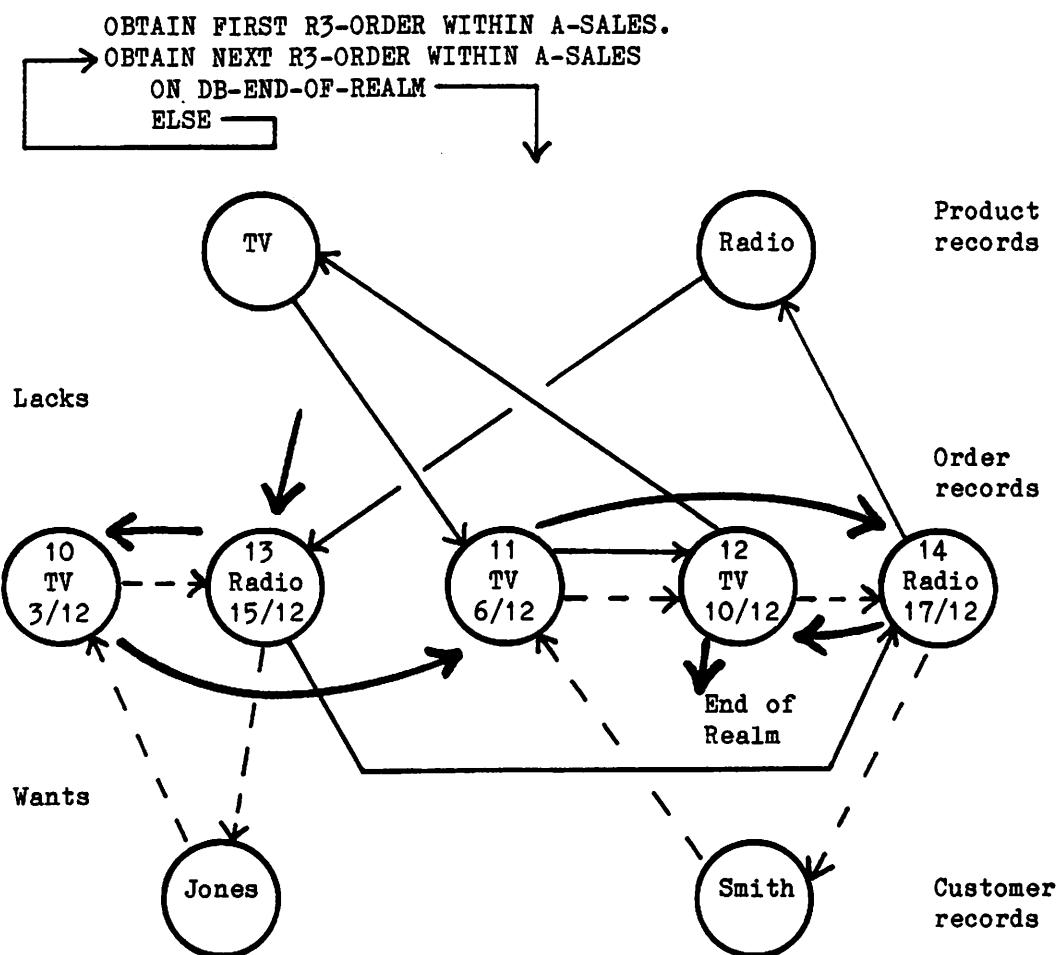


Figure 12 DML sequence 2 - all the order records

Navigation using more than one Set

MOVE "TV" TO R1-PART-NO.
FIND ANY R1-PRODUCT USING PROD-KEY.
FIND FIRST R3-ORDER WITHIN S1-LACKS.
OBTAIN OWNER WITHIN S2-WANTS.

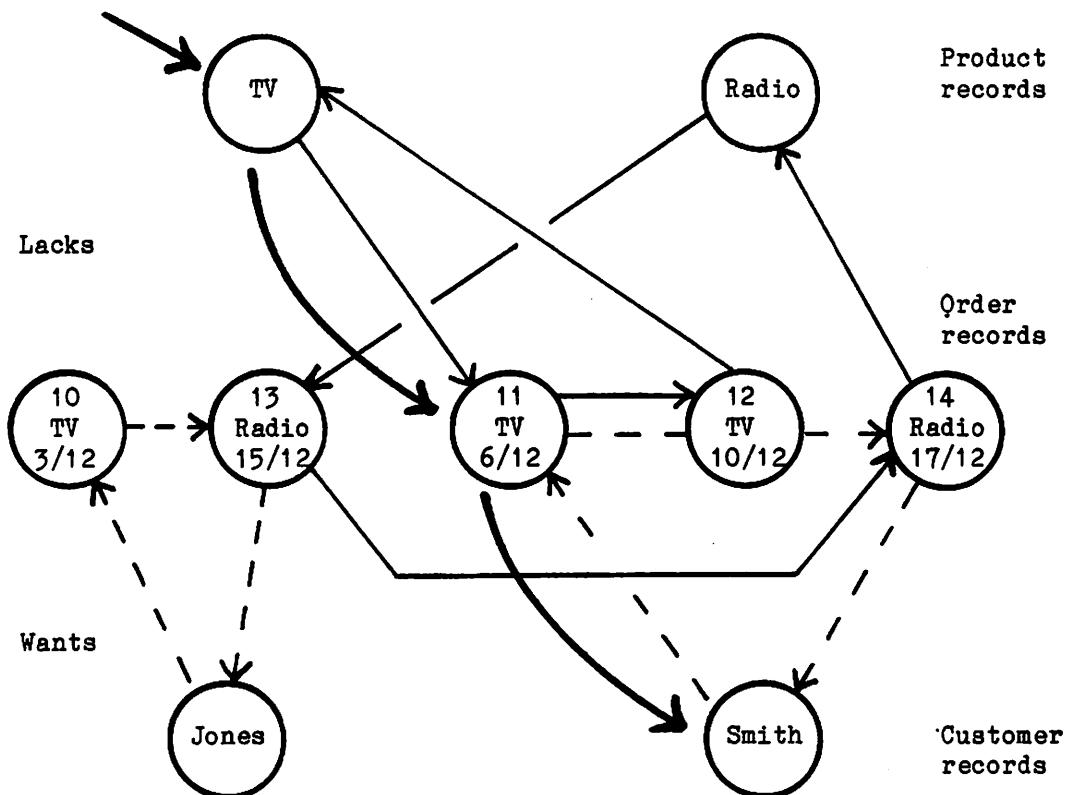


Figure 13 DML sequence 3 - to find the customer waiting longest for a TV

UPDATING THE DATABASE

Introduction

This section deals with the following Verbs:

- STORE Puts a new Record into the Database
- MODIFY Writes an altered Record back to the Database
- ERASE Deletes a Record from the Database
- CONNECT Connects a Record into a Set
- DISCONNECT Removes a Record from a Set connection

The STORE Verb

This Verb places a new Record into the Database, giving it a Database Key according to its placement and connecting it to an occurrence of each Set type where it is defined as an Automatic Member.

Before using the Verb the programmer must:

1. Set up the appropriate values in the relevant Record Area.
2. Ensure that the Current of Set type indicator for each of the Sets into which automatic connection is to occur, is indicating the appropriate Set occurrences.

For example, in the Database in Figure 5 before STORE-ing an R3-ORDER Record, the Current of S2-WANTS Set must indicate the right Customer. In practice, this will usually be so anyway, because of the preceding logic. However the programmer when desk-checking such code should always ensure that the relevant Current of Set indicators are correct and give FIND Verbs to set any that are not right. If this checking is not done, there is a risk that an Order might be connected to the wrong Customer

The syntax of STORE is simply:

STORE Record-Name

and the new Record becomes CRU.

IF AUTOMATIC SET CONNECTION
ENSURE THAT REQUIRED OWNER IS CURRENT

The MODIFY Verb

This is a re-write Verb. It puts back on to the Database a Record which has previously been made current, and has since been altered by normal COBOL processing on the Record Area. The Record must be current of record type.

The syntax is:

MODIFY Record-Name

It is permissible to alter any field values including Record Keys and Sort Keys.

~~X~~ Note: If the CALC key of a record is modified, the record is left on its original page but is connected to the CALC chain of its new target page - effectively we have created overflow.
NOT A GOOD IDEA!

~~X~~

The ERASE Verb

This wipes a Record out of the Database. It is removed from any Sets in which it is a Member. There are several formats of ERASE.

ERASE Record-Name

the Record must be current of record type. For example:

ERASE CUSTOMER

~~S A F E~~
will have no effect (see Figure 5) if the Customer Record that is current of its type owns any ORDER Records. The ERASE will be successful if the CUSTOMER does not own any records. This is known as the unqualified ERASE.

*obj appli
erse specif
members*

There are three more powerful formats of ERASE:

ERASE recordname { PERMANENT
SELECTIVE
ALL }

The PERMANENT option will cause the object record to be erased along with all its permanent members (ie: those members with a retention class of Mandatory). Any optional members will be disconnected.

The SELECTIVE option goes one step further. So as well as erasing the object record and all mandatory members, it also erases optional members not owned by any other record.

The ALL option is extremely powerful and will erase the object record, all mandatory members and optional members whether or not they are owned by any other record.

* The importance of error checking after the use of ERASE cannot be emphasised too strongly.

* Note: Logically deleted records are produced if a member record is erased from a chained set which has no PRIOR pointers.

The CONNECT Verb

This connects a record into a set. The verb makes no sense and will not work if the record is an AM member of that set. The syntax is:

CONNECT recordname TO setname.

This verb connects the record which is current of record type into the appropriate set occurrence usually denoted by the current of set indicator.

The DISCONNECT Verb

This removes a Record from one Set occurrence but not from the Database or any other Sets it may be in. The verb makes no sense and will not work if the Record is an AM or MM Member of that Set. The syntax is:

DISCONNECT Record-Name FROM Set-Name

eg:

DISCONNECT R3-ORDER FROM S1-LACKS.

Like CONNECT, This verb uses Current of Record type (ie Customer in this example) to decide which Record to Disconnect.

* Note: If records are disconnected from chained sets which have no PRIOR pointer, IDMS has to walk the whole set in order to disconnect the record from the set.

SUBSCHEMA CONSTRAINTS

So far we have described the Subschema as an application program's view of the database, in terms of the Realms, Sets and Records that it wishes to access. However, when considering updating applications, the Subschema must define not only the applications Database requirements, but also any extra Realms, Sets and Records that IDMS requires in order to effect the update. For example an application program may simply issue a 'STORE recordname' command, but IDMS in addition to storing the record, may also have to update the Pointer Areas on the Owner records of Sets where the object record is an automatic Member.

If any of the necessary information is omitted from the Subschema, then the updating DML verb will cause a 'Subschema Violation' error at the pre-processing stage and would fail at run-time.

The DML verb to be used will determine the extra Records and Sets that will have to be included in the Subschema. These rules, known as Subschema Constraints, are as follows:

- If STORE is used, the Subschema must contain the object record and all the Sets in which it is a Member, plus the Owner record of those Sets. If the Record has a Record Key, then that Key must also be included
- If MODIFY is used on a Record which is a Member record, then the Record, the Set and the Set Owner must be defined in the Subschema, before MODIFY will work. If the Record to be modified is associated with an Index, be it Record or Set, then the Index and the Realm in which it resides must be present in the Subschema.
- If ERASE is used then the object record, all the Sets in which it is a Member and their Owners, and all the Sets in which the object record is the Owner, and their Members Records must be included. Otherwise the ERASE will not be executed. Note, however, that the actual number of Records affected by ERASE depends on the format of ERASE to be used. The more powerful formats eg ERASE ALL will require the Subschema to include the Records and Sets described above plus any subordinate Records and Sets which will be affected
- Also, if a multi-member Set which is sorted by defined keys is used, then all the Member Records of that Set must be present in the Subschema.

RETRIEVAL ONLY SUBSCHEMAS

From a design point of view, Subschema Constraints mean that more Record and Set types have to be included in the Subschema than are required for processing purposes. This reduces the degree of privacy that the Subschema mechanism provides. However, a proper use of Privacy Locks and dummy fields can re-establish this level of Privacy. For example we can lock FIND on a Set type included just to avoid a Subschema Constraint. Similarly for otherwise unwanted Record types we need only define one unimportant or artificial field.

There is one case where ignorance does not prevent action. If a Record is STORED and not all its fields are included in the Subschema, then the Record is stored with the unknown fields Set to COBOL low values (Binary ZERO).

Proper communication between the designers, database administrators and the programmers is essential to ensure the efficiency of their job tasks. Any design decisions which may affect the programmer must be communicated. Suggestions in this area are discussed later in this chapter.

DATABASE PROCEDURES

Database procedures are a feature of IDMS which can be used to get a low level of control and/or to help application programmers.

A database procedure is a database administrator's subroutine which will be called by a program when a certain condition occurs. The condition is that a specific DML function has been issued by a program and directed at a specific Record type. The routine which may be invisible to the program, can be called before or after the DML instruction has been obeyed, or only if it goes wrong. The logic of the process is shown in Figure 14.

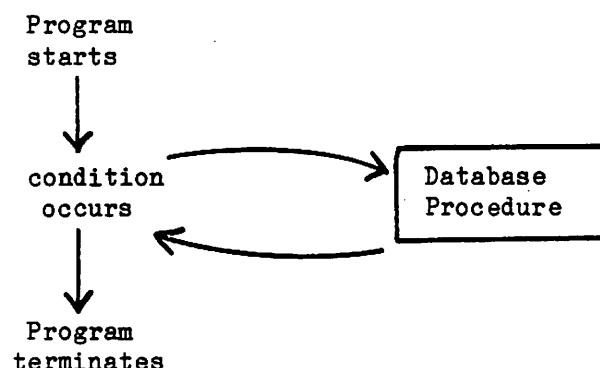


Figure 14 Database Procedures

Notice that the programmer is powerless to stop this. The procedure is specified by the schema in the appropriate record description using the syntax shown in Figure 15.

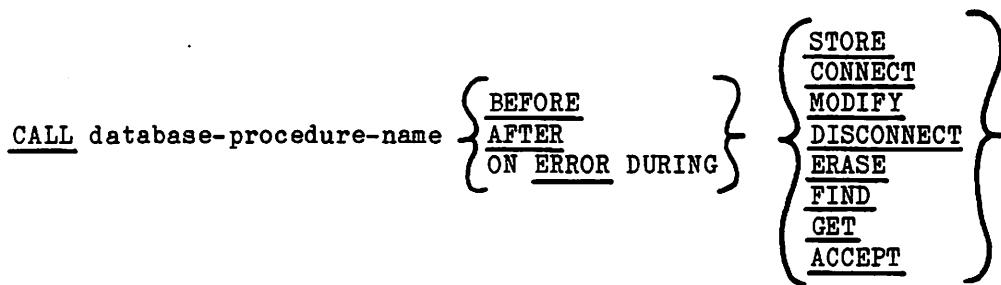


Figure 15 Database procedure syntax in the Schema DDL

Uses of Database Procedures

The subroutine gives the database administrator a lot of power. He can see:

- the Schema record ie the record as it is on the database (not necessarily the Subschema view) assuming the DML function has occurred). Therefore he can alter the data before it reaches the program. He can also modify a record on its way into the database
- the error indicator and thus standardise error handling procedures for a DML function going wrong
- an optional communication field within the user program.

He can discover a lot about the user program - its name for example. He can forbid the DML function if it has not occurred. But the uses of database procedures are not restricted to control. They can also be used to help programmers. Below are outlined some of the more obvious uses for database procedures:

- validation
- to provide standard error handling procedures
- encoding and decoding data for security reasons
- checking privacy at the record occurrence level. This gives a third level of control after those provided in the Subschema. It can for example prevent access to record occurrences that have particular data item values. The two earlier levels blocked access at the type level eg to particular record types.
- gathering of statistics and an aid to auditing

- a principle use of database procedures is to contain Storage Schema dependent code, so that application programs can remain independent. For example, the code that deals with the storage of records with PAGE-DIRECT placement could be written as a database procedure.

The use of database procedures should be considered very carefully as there is significant performance overheads on each call.

Improving Programming Efficiency

In order to improve efficiency at the programming end, any decisions made by the designer that may affect the programmer should be communicated to him.

A number of procedures are beneficial to the smooth running of any system.

1. set up adequate communication and documentation using as much automatic IDMS reporting as possible. The REPORTS utility, in particular the User REPORTS version, can generate an accurate view of what is needed in the system from the IDMS Directory contents. The User Reports provides:
 - . a data item listing detailing the items that make up each record type in a Subschema
 - . a record description listing detailing all the record types included in a Subschema and documents the allowable DML functions on each
 - . a set description listing detailing all the Sets in a Subschema and their allowable DML functionsThese are useful documents which can be included as part of a program specification.
2. arrange comprehensive training of programmers and systems personnel to enable them to carry out their tasks
3. ensure the database administrator function serves as a consultancy, not only for database design but also for DML strategy
4. vet programs meticulously

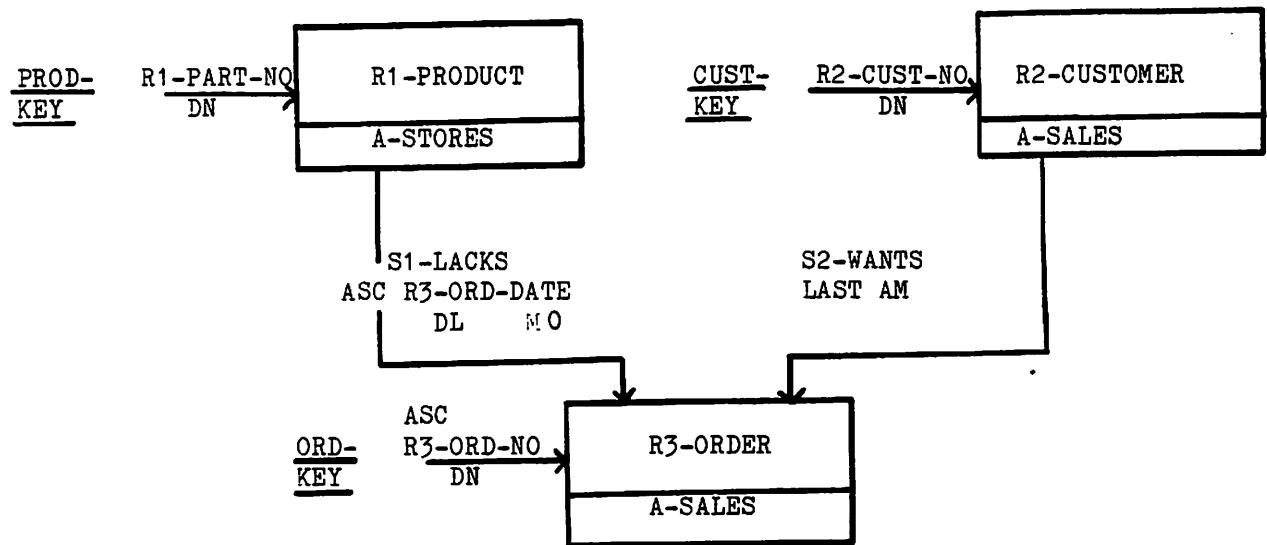
5. test programs on a separate test database
6. set up database procedures to allow certain DML functions on given record types to be detected at run-time and if necessary disallowed. Consider also using database procedures to help the programmer eg to provide some commonly used but tricky validation as an automatic procedure.

APPENDIX

This appendix describes a Subschema in 3 ways:

1. As a Subschema Diagram.
2. As Subschema DDL.
3. As DDCL for the DDS.

SUBSCHEMA DIAGRAM



SUBSCHEMA DDL

Example 1

```

SUBSCHEMA IDENTIFICATION DIVISION.
SUBSCHEMA NAME TOTDATA OF SCHEMA CASHFLOW.
SERVICE SDL1.
STORAGE SCHEMA CASHSS.
SUBSCHEMA DATA DIVISION.
REALM SECTION.
  REALM A-STORES CONTAINS R1-PRODUCT RECORDS.
  REALM A-SALES CONTAINS ALL RECORDS WITHIN SALES.
RECORD SECTION.
  COPY ALL RECORDS.
SET SECTION.
  COPY ALL SETS.
  
```

This Subschema allows access to all parts of the database as it is the same as the Schema.

Example 2

```
SUBSCHEMA IDENTIFICATION DIVISION.  
SUBSCHEMA NAME ORDDATA OF SCHEMA CASHFLOW.  
SERVICE SDL1.  
STORAGE SCHEMA CASHSS.  
SUBSCHEMA DATA DIVISION.  
REALM SECTION.  
    REALM A-SALES CONTAINS ALL RECORDS WITHIN SALES.  
RECORD SECTION.  
01 R2-CUSTOMER  
    PRIVACY LOCK FOR STORE IS "NO"  
    PRIVACY LOCK FOR ERASE IS "NO".  
03 R2-CUST-NO.  
03 R2-C-NAME.  
  
COPY R3-ORDER RECORD  
    PRIVACY LOCK FOR ERASE IS "NO".  
SET SECTION.  
COPY S2-WANTS SET.
```

The DDCL to load the first Subschema into the DDS would be:

```
INSERT SUBSCHEMA TOTDATA  
*SCHEMA CASHFLOW  
*DB-SERVICE SDL1  
*STORAGE-SCHEMA CASHSS  
*REALMS A-STORES A-SALES  
*RECORDMAP ALL RECORDS  
*SETS ALL SETS  
  
INSERT REALM A-STORES  
*CONTAINS R1-PRODUCT RECORDS  
  
INSERT REALM A-SALES  
*CONTAINS ALL RECORDS WITHIN AREA SALES
```

To load the second Subschema subsequently, the DDCL would be:

```
INSERT SUBSCHEMA ORDDATA
*SCHEMA CASHFLOW
*DB-SERVICE SDL1
*STORAGE-SCHEMA CASHSS
*REALMS A-SALES
*RECORDMAP
    R2-CUSTOMER/R2-SS-CUSTOMER
        PRIVACY LOCK FOR STORE IS "NO"
        PRIVACY LOCK FOR ERASE IS "NO"
    R3-ORDER
        PRIVACY LOCK FOR ERASE IS "NO"
*SETS S2-WANTS

INSERT RECORD R2-SS-CUSTOMER
*STRUCTURE
    ITEM CUST-NO
    ITEM C-NAME
```

Note. The need to have two Record Elements where a subset of a Record is required in the Subschema. R2-SS-CUSTOMER is an artificial Record Element placed in the DDS to define the restricted CUSTOMER Record viewpoint that users of this Subschema are to have.

CHAPTER 6

THE RUN-TIME ENVIRONMENT

INTRODUCTION

This chapter considers the IDMS Run-time Environment:

- The components of a run-time program
- The compiling system
- The implications of sharing, in particular the locking mechanisms and their interaction with the Success Unit Concept
- IDMS services and the Service Description Language (SDL) which defines some of their characteristics.

COMPONENTS OF A RUN-TIME PROGRAM

Introduction

Logically an IDMS run-time program has the 4 major components shown in Figure 1.

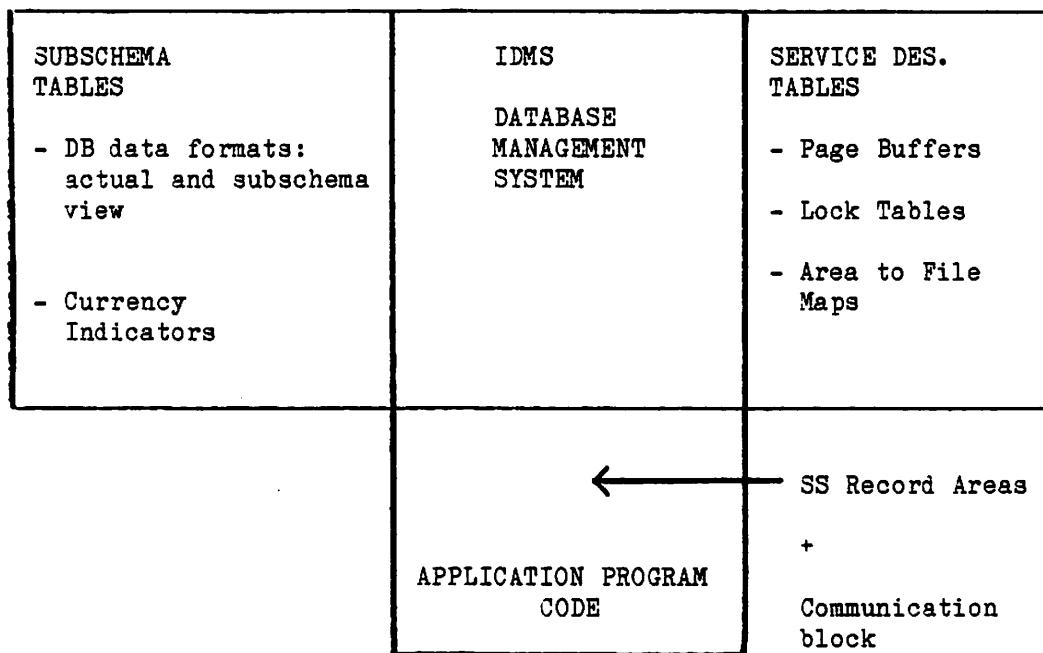


Figure 1 LOGICAL COMPONENTS OF AN IDMS RUN-TIME PROGRAM

Application Program Code

This was written by the programmer but now includes additional material placed there during compilation. The most notable divisions of this additional material are:

- the Subschema record areas which are used as Input/Output areas by the DML Record handling verbs. There will be a Record Area for each Record type defined in the Subschema, containing just the fields defined in the Subschema for that Record type.
- the communication block containing fields used for communication between the IDMS DBMS and the run-time Program. The most important of these fields is ERROR-STATUS which after a DML verb has been obeyed contains a code indicating the result ie Correct, Exception (eg End of Set) or Error (eg no currency set up).

IDMS DBMS

This is the Record level Input/Output code. It receives Disc blocks from the Operating System's Data Management which will then be put into Buffers in the SDL Tables.

The DBMS then processes these Buffers regarding their contents as Pages which in turn contain Records linked by Setsm Indexes etc. The formats of this data are derived from the Subschema Tables which define the data as it is, and as this program is to see it. When a buffer is required for a new Page or at the end of a Success unit, the contents are passed back to the Operating System, (if they have been altered), for writing out as a Disc Block.

Subschema Tables

As well as defining the data formats for the DBMS and specifying any Database Procedure requirements, the Subschema Tables also contain the Currency Indicators. These are set by DML verbs like FIND and STORE, and then guide later DML verbs in precisely which Record occurrences they process. They are, therefore, intimately associated with the logic of a particular Success Unit and cannot be shared between different programs.

Service Tables

Besides the Page Buffers, these contain the Lock Tables and Area to File Maps.

The Lock Tables control any simultaneous access requirements between different Success Units as described later in this chapter. Unlike the SST Currency Indicators, the Lock Tables must be shared between all simultaneous Success Units where any updating is involved if Database corruption is to be avoided and/or logical results achieved.

The Area to File Maps specify which real file or files each Area occupies during the current run.

THE IDMS COMPILING SYSTEM

Using DDS

Normally the database is fully defined within DDS.

DDS elements which allow this include:

SCHEMA
STORAGE-SCHEMA
SUBSCHEMA
DB-SERVICE
DATABASE.

The schema and storage-schema(s) are processed by use of the DDCL commands:

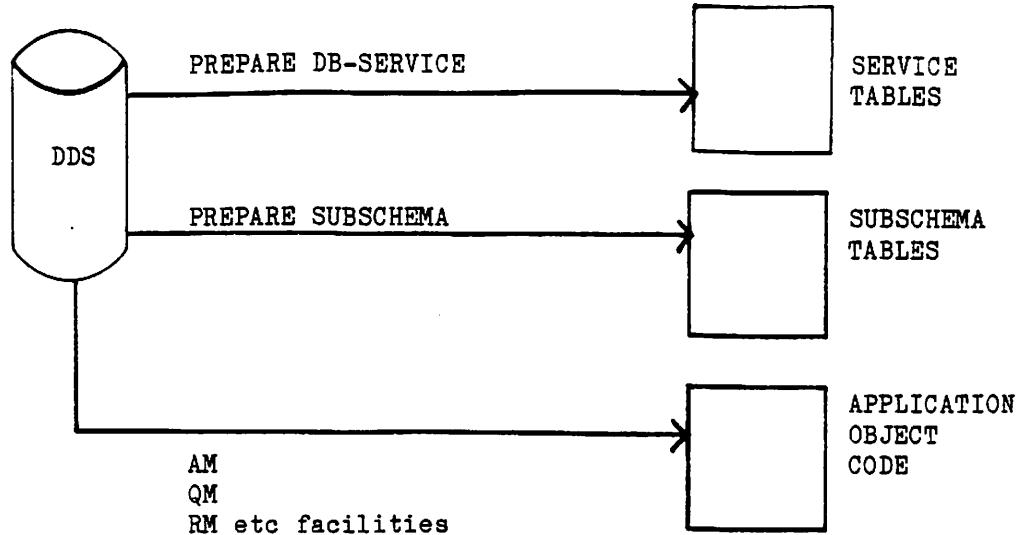
PREPARE SCHEMA ...
PREPARE STORAGE-SCHEMA ...

The subschema(s) and database-service(s) are processed by using:

PREPARE SUBSCHEMA ...
PREPARE DB-SERVICE ...

The latter two commands also produce object subschema tables and service tables.

Application Master, Report Master and Query Master applications may also be defined within DDS and compiled.



If COBOL programs or DATA DISPLAY require access to the database then a database directory is needed.

THE DIRECTORY

Pages 1-1000 of the database are reserved for use by the database directory.

It contains complete logical and physical definitions of the database together with other information.

A directory is needed in order to:

1. compile COBOL programs which contain DML verbs
2. access the database using DATA DISPLAY.

Once the directory has been created and initialised there are several methods of loading the appropriate data into it.

Loading the Directory

In order to allow COBOL programs to be compiled the directory must contain things such as:

- a) details of the logical and physical structure of the database - schema, storage-schema and subschema definitions.
 - b) information on how DML verbs should be "translated" into COBOL
 - c) details of a communications block which allows IDMS to communicate with COBOL applications
- a) is loaded by use of the various IDMS processors.
- b) and c) are loaded by use of the CLUC utility.

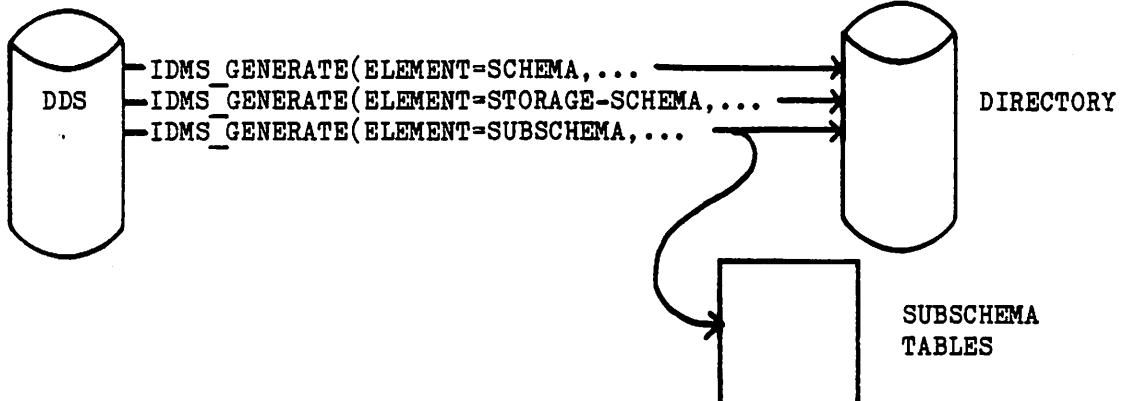
Loading the Database Description

The directory must be loaded with:

- a schema
- one or more storage schemas
- one or more subschemas.

The SCL command IDMS GENERATE may be used to process a DDS definition of a database and load the directory with the appropriate information. Subschema tables may be generated at the same time.

If IDMS GENERATE is used to load a schema then IDMS CLUC is called automatically by the system.



Note: when IDMS GENERATE is used the system produces a file of schema DDL or data storage DDL or subschema DDL (see below). This is then loaded onto the directory by the appropriate processor and - in the case of a subschema - the subschema tables may be compiled into an object module. See below for additional information.

Alternatively, the database may be defined outside of DDS. The schema is then coded in schema data description language; storage schema(s) in data storage DL; subschema(s) in subschema DDL and the service description(s) in service description language.

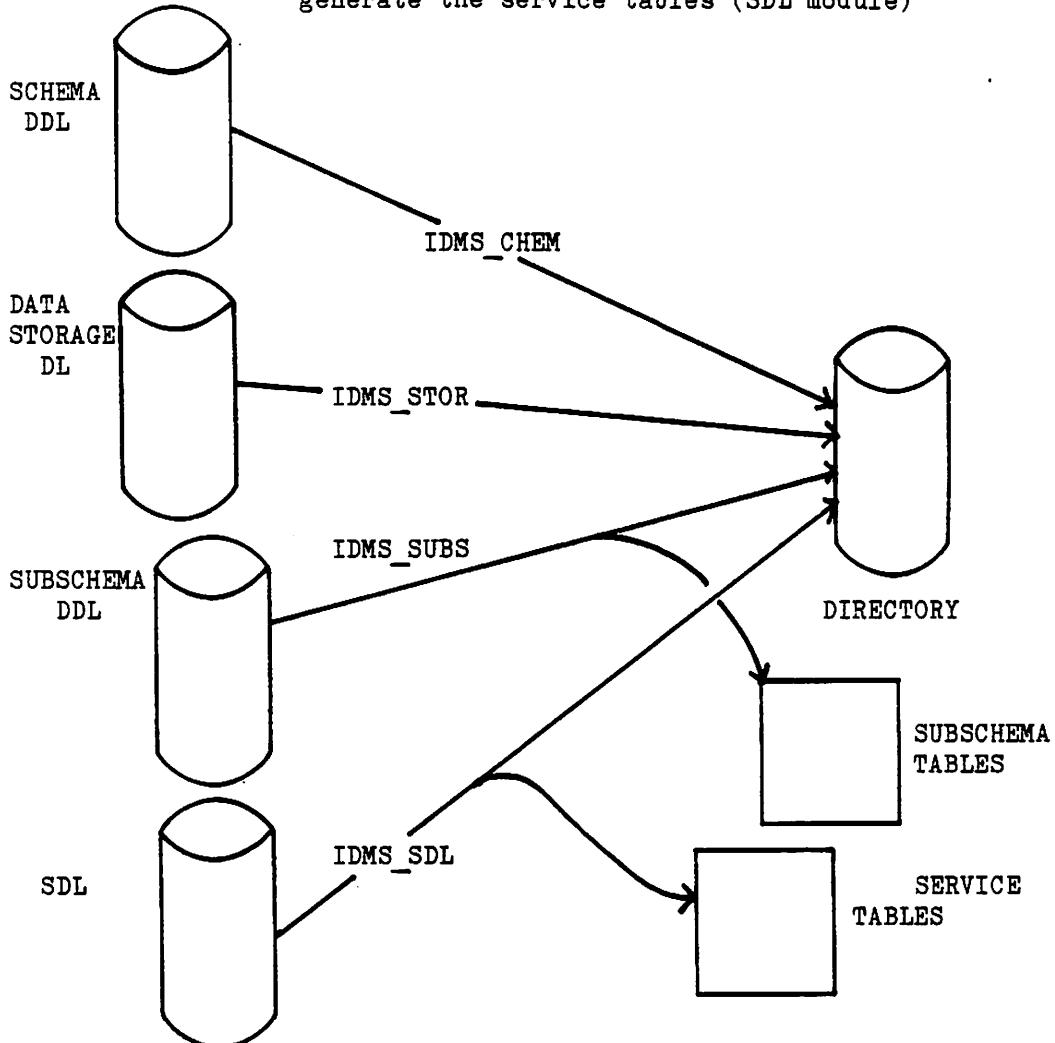
The following SCL commands may then be used:

IDMS_CHEM - load the schema into the directory

IDMS_STOR - load a storage schema into the directory

IDMS-SUBS - load a subschema into the directory and generate
subschema tables

IDMS SDL - load a service description into the directory and
generate the service tables (SDL module)

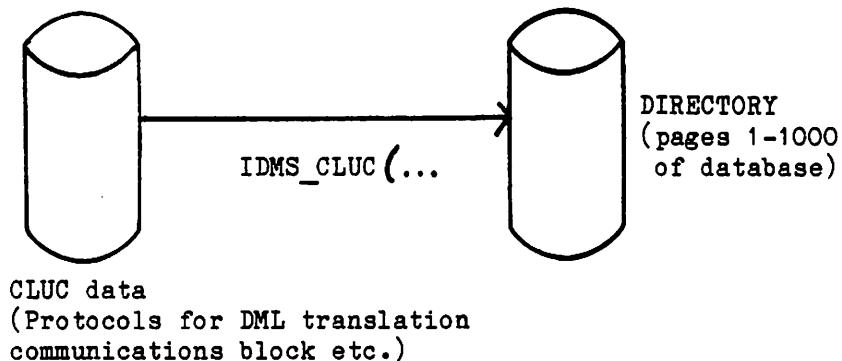


Other possibilities also exist - for example IDMS_GENERATE may be used to produce a file of schema DDL (or subschema DDL etc). This can then be loaded into the directory by use of IDMS_CHEM (or IDMS_SUBS etc).

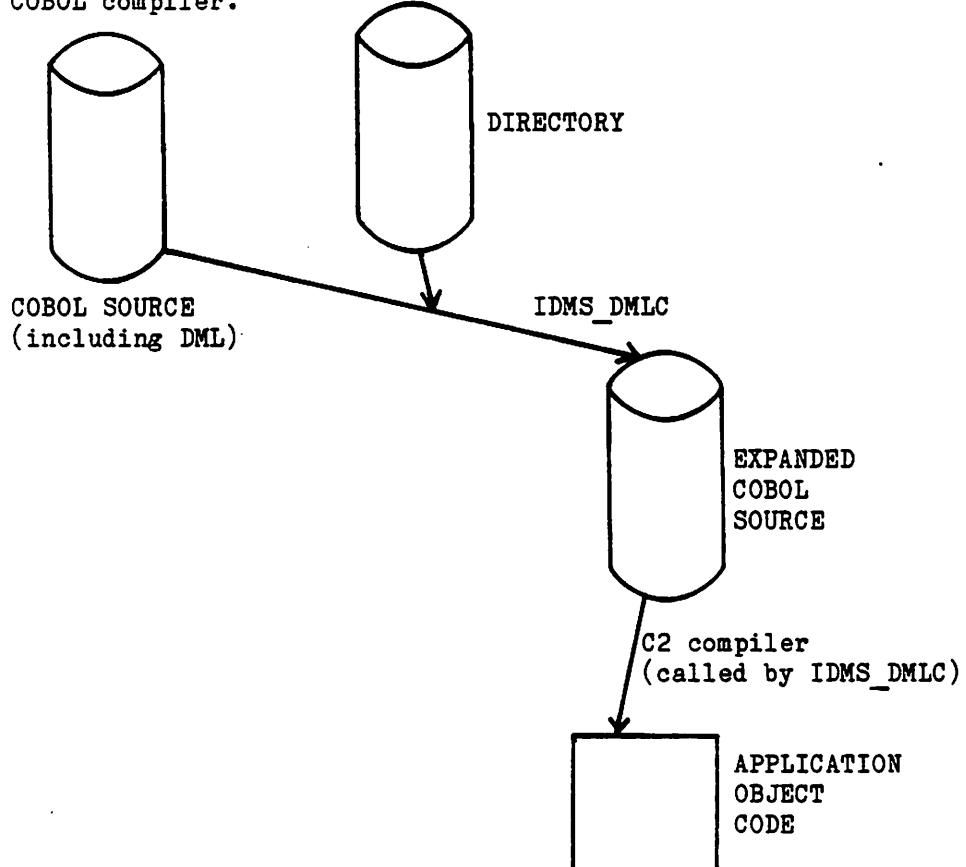
Note: it is not possible to use IDMS_GENERATE to produce either a file of SDL or to compile a set of service tables (ie an SDL module).

CLUC

The SCL command IDMS_CLUC is used to run the CLUC utility. Details of the communication block and DML translation are loaded onto the directory. It is also possible for CLUC to register subschemas so that they may only be used by specified programs and/or by specified VME users.



Once the appropriate information has been stored in the directory; COBOL DML programs are compiled by using the command IDMS_DMLC. The directory is accessed in order to translate DML verbs etc. and the resulting 'expanded source program' is passed through the standard C2 COBOL compiler.



IDMS SERVICES

There are two basic types of IDMS service:

1. A catalogued service *part of VME*
2. An uncatalogued service.

Both catalogued and uncatalogued services may be used as:

- a) An unshared service (in a batch environment). This allows only one application to access the database.
- b) A shared service (in a batch or TP or mixed environment). This allows many applications to access the database concurrently.

Catalogued services offer much better recovery and resilience facilities and are used for system testing and live running.

Uncatalogued services are easier to set up and are generally used for testing applications.

Structure and Use of a Catalogued Service

A catalogued service provides a consistent means of accessing the 'extended database'.

The extended database consists of the database itself together with recovery files such as - security dumps, journals and the database diary.

The service is permanently defined in the VME catalogue.

In order to catalogue a database service the following steps must be taken:

1. The system manager must set up the standard service description by using the SCL command

INTRODUCE_DB_SVD

2. The system manager must set up a SERVICELIBRARY for the user who is to own the catalogued service. This is achieved by use of the SCL command

INTRODUCE_SERVICE_ENVIRONMENT

The SERVICELIBRARY is used to hold the compiled service tables.

3. The service tables must be stored in the SERVICELIBRARY.
4. A database diary must be set up and initialised.
5. The database service is then catalogued by use of the SCL command

INTRODUCE_DB_SERVICE.

Running Applications in an Unshared Catalogued Service

If the catalogued database service is not currently running then it may be made available for unshared access by using the following SCL commands:

IDMS_FILES - to assign the database journal and other database files

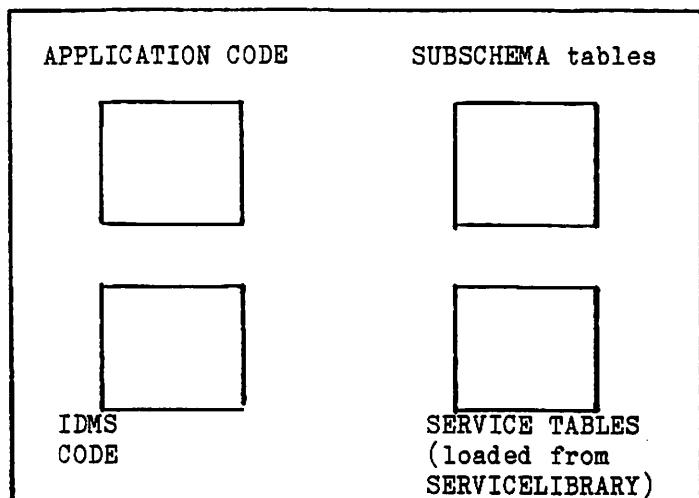
IDMS_UNSHARED_RUN - to start the unshared service running

SCL to run the application

IDMS_ENDRUN - to close the service down tidily.

The use of automatic file assignment facilities make it possible to omit the IDMS_FILES command provided that the names of the required files have been recorded in the database diary.

The virtual machine in which the unshared service is running loads the appropriate subschema and service table modules.



Unshared Catalogued Service (1)

It is also possible to request exclusive access to a shared catalogued service - see below for details.

Running Applications in a Shared Catalogued Service

First, the shared service must be started.

The SCL command `START DB SERVICE` is used. This sets up a service virtual machine and starts it running.

The service virtual machine assigns the database files and journal(s) etc and makes the database available to its authorised users.

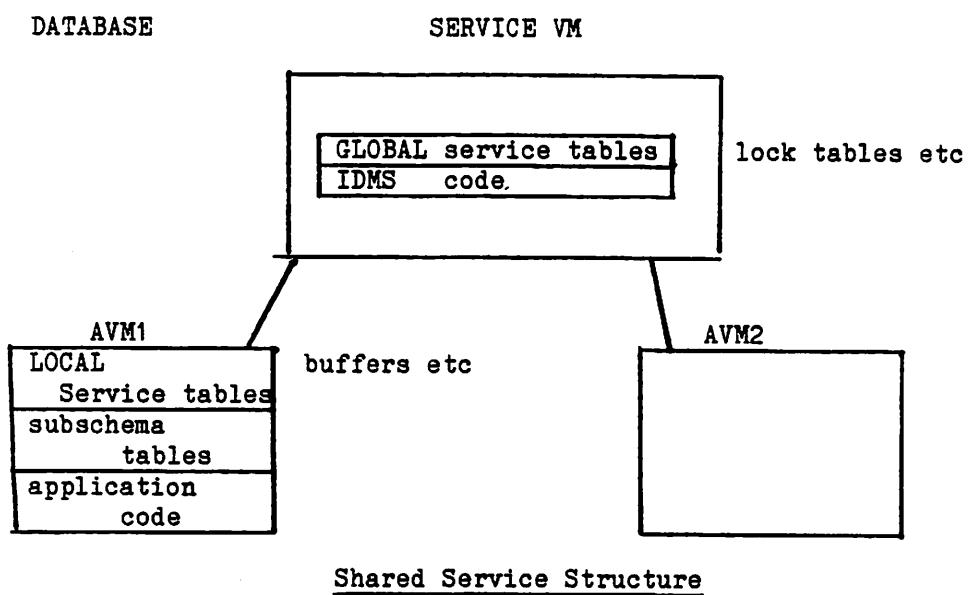
In order to access the database; other virtual machines use the following SCL commands:

IDMS_RUN - informs the shared service that access to the database is required and updates the database diary etc.

SCL to run the application

IDMS_ENDRUN - disconnects the application virtual machine from the database service and updates the diary.

The service virtual machine may be closed down by using CLOSE_DB SERVICE.



Note: it is possible to use the command `IDMS RUN` to request unshared access to the database service. If this is done then accesses to the database are still made via the service virtual machine.

UNCATALOGUED SERVICES

An uncatalogued service allows access to a "private" database in a simpler way than a catalogued service.

Unshared Uncatalogued Services

In order to access such a service the following SCL commands are used:

IDMS_FILES (if automatic file assignment is not being used)

IDMS_UNSHARED_RUN

SCL to run the application

IDMS_ENDRUN

It is not necessary for the system manager to take any specific action.

Shared Uncatalogued Services

In order to set up and run such a service the following steps must be taken:

1. The system manager must use **INTRODUCE SERVICE ENVIRONMENT** to set up a **SERVICELIBRARY** for the user who is to own the database service.
2. The appropriate service tables must be stored in the **SERVICELIBRARY**.
3. A database diary should be set up.
4. The SCL command **IDMS_SV_GEN** should be used to set up event nodes in VME catalogue.

In order to start the shared service running the following SCL commands are used:

IDMS_FILES - if not using automatic file assignment

IDMS_START_UP

The virtual machine which issues these commands is then used as the service virtual machine.

In order to access the shared service; other virtual machines use:

IDMS_RUN

SCL to run the application

IDMS_ENDRUN

To close down the shared service, the command

IDMS_CLOSEDOWN is used.

THE IMPLICATIONS OF SHARING

Introduction

If several Applications are being run together on the same Database files, the reasons may be:

- to provide a service to different users at the same time
- to improve throughput.

Whatever the reason, each Application must produce correct results ie as if it were run alone, and must leave the Database in a consistent state. The mechanisms for this are the Success Unit and the IDMS Service which together provide appropriate Locking and Recovery procedures.

The Success Unit concept was introduced in Chapter 5 and all IDMS Applications must be written to this standard. The general principle is to keep Success Units short since:

- they are the Unit of Recovery ie a failure during a Success Unit will cause the Recovery System to return the Database to the state it was in at the Success Unit's start - whether that was 2 seconds ago or 2 hours
- they are the Unit of Locking. Any Success Unit waiting for locked data must wait until the locking Success Unit finishes successfully or is recovered successfully. A Success Unit should not be longer than an acceptable waiting time for other users.

Hence a Success Unit will be a TP message Pair, a complete Batch Program or a section of a Batch Program split into suitable short logical sections.

Recovery is described in Chapter 8. The rest of this section considers the locking mechanisms.

Locking Mechanisms

Introduction

IDMS provides locking at two levels - Areas and Pages. The effect of each is shown in simplified form in Figure 6.

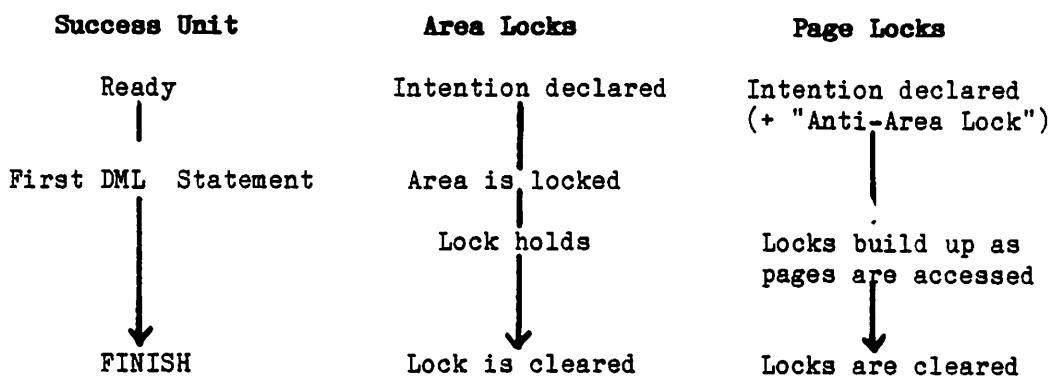


Figure 6 Effect of Area and Page Locks in a Success Unit

An area lock applies to the whole area for the duration of the success unit; page locks come into force as required. The 'Anti-Area Lock' prevents other success units putting area locks on that area.

When choosing between area and page locks, you need to consider whether the extra throughput gained by letting several success units access an area simultaneously may be dissipated in other ways. Page locking uses more mill time and more store.

In general use page locks for TP message pairs and for Batch programs that are likely to share IDMS with on-line programs. Such Batch programs should be artificially segmented to keep success units short.

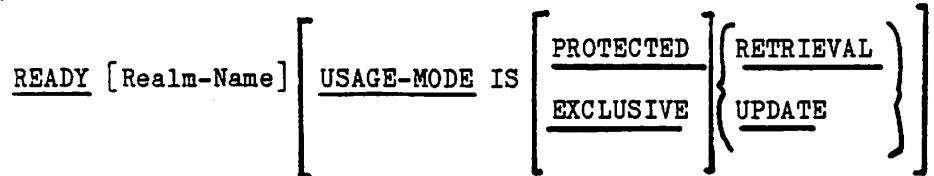
If one success unit locks many pages, this increases the chances of contention - ie time wasted while a success unit is suspended waiting for locks to be released - and deadlock. Success Units which are complete Batch programs are likely to use area locks, often holding large sections of the database for their own exclusive updating use for substantial periods of time. Such jobs may well need to run alone eg overnight.

The programmer controls the level of locking via the READY statement for an area. IDMS then takes over the locking process.

Locks are recorded in lock tables. IDMS checks any READYs (for area locking) and page accesses (for page locking) from other success units against the current lock table entries.

Syntax of READY

The syntax for READY as we saw in Chapter 5 is:



Area Locks

IDMS applies area locks for

- PROTECTED RETRIEVAL
- EXCLUSIVE RETRIEVAL
- EXCLUSIVE UPDATE

PROTECTED means that any other success unit can gain access to the Realm for reading but not for updating (ie the current success unit is protecting itself from updates).

EXCLUSIVE means that other success units cannot gain access to the Realm at all (ie the current success unit has exclusive use of the Realm).

Note. The rest of this section refers to Area locks since in locking Realms, the Programmer is actually locking the underlying Areas. Should one success unit lock the same Area via, say, two different Realm viewpoints then the requirements are amalgamated with the strongest taking precedence eg if one Ready specifies EXCLUSIVE RETRIEVAL and the other PROTECTED RETRIEVAL then the underlying Area is opened for EXCLUSIVE RETRIEVAL.

If a success unit needs to issue several READYs, they must occur together at one point in the code. IDMS collects those declaring area locking together and attempts to open all the areas at once. If it fails because the Lock Tables already have entries for one or more of these areas, the success unit is suspended waiting for those locks to be released. Consequently deadlock can not occur with area locking.

Page Locks

Areas opened with READY for:

- RETRIEVAL
- UPDATE BIT TOO FREE??

are subject to page locks. This means that the success unit in question locks only the pages it accesses in the area. If the success unit is a retriever in the area, its locks will only lock out success units that are updaters. If the success unit is an updaters, its locks will prevent all other success units from accessing its pages.

PROTECTED UPDATE is a special case. It prevents other success units from updating the entire area but allows them to read pages that are not being updated. In other words, it applies an area lock against updating but page locks against retrieval.

If the page lock system is about to produce deadlock, IDMS steps in and returns an error status of XX93 to the success unit whose DML function would have caused deadlock. That success unit must then use one of the resilience mechanisms to cause any updating it has done to be reversed and its locks to be released. The success unit may then be repeated.

IDMSX

In IDMSX, an alternative Delayed page locking mechanism is available on request.

The basic locking system described above could delay other success units unnecessarily by putting on too many locks. For instance, if an updaters is FINDing a specific record in a sorted chained set, the basic system applies update page locks to all pages accessed in the search - not just to the page containing the record with the quoted sort key.

The alternative delayed locking system puts read locks on the pages searched. It converts the lock on last page searched to an update lock if the required record is subsequently modified. This method reduces the impact of the locking system on other success units.

Also available in IDMSX is a DML Verb FREE which given within a success unit will free any retrieval page locks perhaps allowing some other success unit to proceed.

Use of Locking

Your locking strategy should cause the minimum of disruption to other users, so:

- READY just the areas you need and specify the minimum access requirements eg PROTECTED UPDATE rather than EXCLUSIVE UPDATE
- Operations such as loading a magnetic tape or changing line-printer stationery should be done outside the success unit, so that the areas/pages are available to other success units while such operations are going on
- If a program's requirements change during a run, divide the run into several success units so that area/page locks are not applied for longer than necessary.

When Batch programs are likely to share IDMS with on-line programs, keep success units short to reduce area/page conflict.

Appendix 1 SDL EXAMPLES

SDL coding looks like this:

SERVICE NAME IS SDL1
OF STORAGE SCHEMA CASHSS
OF SCHEMA CASHFLOW.

BUFFER SECTION
BUFFER CONTAINS 5 PAGES.

MORE NAVIGATION = MORE BUFFERS

SUGGEST 10 for MP

3 for Serial Banks.

AREA SECTION

COPY SALES AREA
LOCKS ARE DELAYED.
COPY STORES AREA.
COPY INDX-RIM AREA.

LOCK SECTION.

LOCK TABLE CONTAINS 500 PAGES.

NOT ALLOWED

ACCESS CONTROL SECTION.

DATABASE DIARY IS MANDATORY.

JOURNAL SECTION.

JOURNAL BEFORE-IMAGES.
JOURNAL AREA-JNL-1
CONTAINS AFTER RECORDS FOR ALL AREAS.

AREA / CENTRAL

An equivalent DB-SERVICE Element can be loaded into the DDS by the following DDCL:

```
INSERT DB-SERVICE SDL1
*STORAGE-SCHEMA CASHSS
*SCHEMA CASHFLOW
*BUFFER CONTAINS 5 PAGES
*AREAS SALES
    LOCKS ARE DELAYED
    STORES
    INDX-RLM
*LOCKS LOCK TABLE CONTAINS 500 PAGES
*ACCESS-CONTROL DATABASE DIARY IS MANDATORY
*JOURNALS
    JOURNAL BEFORE-IMAGES
    JOURNAL AREA-JNL-1
        CONTAINS AFTER RECORDS FOR
            ALL AREAS
```

CHAPTER 7

IDMS SECURITY

NO265-6

INTRODUCTION

The essence of database is the sharing of data between different users. This means that an IDMS database will be used by a number of programs in a number of systems.

Each program accessing the database may run alone. This situation differs little from conventional systems except that security is more important.

To speed up throughput, a number of programs may access the database at the same time. The most complex cases could involve concurrent on-line updates in the same area. Such processing requires that:

- each program produces the same results as if it had run alone in the machine
- the formidable security problems are solved satisfactorily.

This handout examines the mechanisms for security that IDMS provides and the ways in which they may be used.

SUCCESS UNIT

The concept of a SUCCESS UNIT must be understood if the Locking and Recovery mechanisms are to make sense. This was introduced in Chapter 5.

Remember a better term might be a **potentially successful unit of coding** since there are two possible situations:

- The success unit reaches its end successfully. We need not repeat the processing even if the database is destroyed at some future date; we can use the recovery mechanisms to retrieve the results of the processing from a dump
- A failure occurs before the end of the coding is reached. In this case, we need to reverse any updating the success unit has done and, if the processing is still appropriate, we need to repeat it from the start.

This concept is vital to understanding the security mechanisms. Any locks applied will hold until the end of the success unit that applied them and any recovery action will recover in units of one (or more) success units.

To systems designers, a success unit is either a TP Message Pair (phase) or a Batch program. If a success unit corresponds to a complete Batch program, it could build up an enormous number of locks and require a long time to recover. For this reason, Batch programs are usually artificially divided into many success units each processing just a few transactions.

As a general rule, keep success units as short as possible. This will

- minimise delay to other success units that are waiting for locks to be released
- make any recovery process as rapid as possible
- make lock tables smaller and more efficient.

Figure 1 shows a series of success units for three simultaneous users.

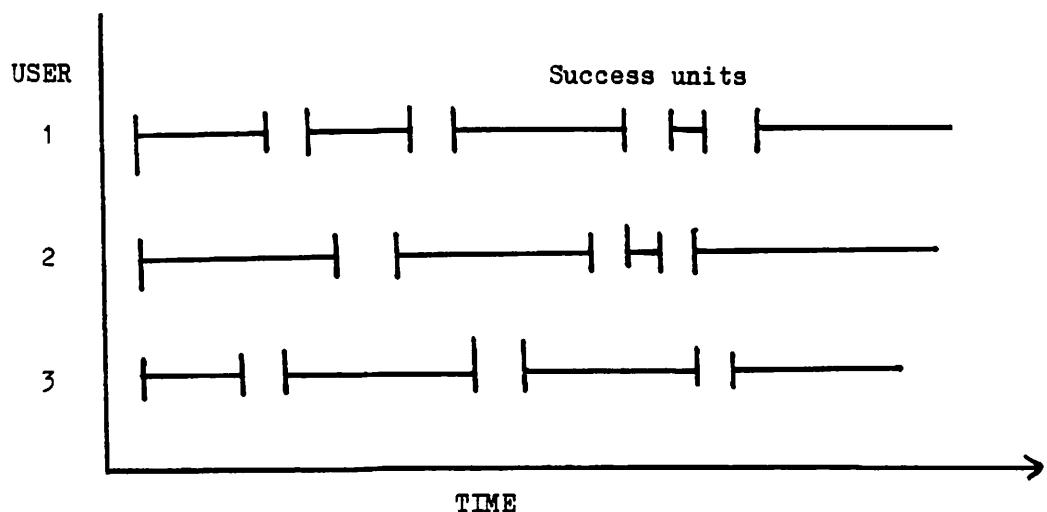


Figure 1 Three streams of success units

SECURITY

Security was defined in an earlier chapter as RIP:

- **Resilience** ie the ability to recover from failure.
- **Integrity** ie ensuring, via a locking mechanism, that multiple simultaneous update access does not corrupt data.
- **Privacy** ie imposing appropriate restrictions on access and use.

Privacy and the three levels of control that IDMS provides have already been considered:

- leaving things out of the Subschema
- privacy locks in the Subschema
- database procedures.

These can be supplemented by:

- restricting access to READ ONLY
- privacy mechanisms for files provided by the operating system
- privacy checks in the application code
- privacy checks in the TP software and hardware eg Badges
- limiting access to the run-time software through the ACR level mechanism and to the libraries containing commands and utilities. (Available only in IDMSX and currently only with VME/B. Needed only in a few highly sensitive sites.)

Integrity and Resilience are the major security problems in a concurrent on-line update environment. Integrity (or locking) was again considered in an earlier chapter. This section concentrates therefore on Recovery.

RECOVERY

Type of Failure

There are three types of failure:

- local failure, which affects just one success unit. In this case there may be other success units still running
- system failure, which stops all current success units
- file failure, which may cause a local failure or a system failure or none at all.

We shall look at the various recovery mechanisms available with IDMS, then at their application to each type of failure.

Mechanisms for Resilience

The mechanisms for recovering an incorrect database are:

- Dump and Restore Utilities
- Central Journal
- Quick-Before-Looks
- Area Journals
- Rollback and Rollforward
- Delayed Update
- Duplexing
- Database Diary and Consistency Checking.

Dump and Restore Utilities

The Dump Utility can dump either the entire database or specified areas to a dump file. It can also produce the static database statistics (described in Chapter 9) either as well as or instead of the dump.

The Restore Utility takes the result of a dump and uses it to re-create either the whole database or selected areas.

These two utilities provide a recovery mechanism that may be acceptable in simple environments or during the early days of a complex environment. Provided the database is small and is updated only in Batch mode, the following procedures may provide adequate resilience:

- Dump the database at frequent intervals (eg after each update run)
- Keep all transactions since the last dump
- If there is a failure, restore the last dump and re-run the relevant application programs, re-applying the transactions.

As the database gets bigger and/or on-line updating is required, this simple approach becomes uneconomic and/or impracticable. For example, dumping a larger database takes longer so that it may have to be done less often. This means that recovery takes longer because more transactions have to be re-processed. But this is at the very moment when faster recovery is required as the use of the database expands.

The solution to this dilemma is to use a journalising technique. The journal is a magnetic media file containing copies of changes to the database. Recovery then involves a simple merge of this file with the database, or with a restored dump. This is far faster than re-processing the transactions. It means less frequent dumps will still allow rapid recovery.

While the journal can be on tape, a disc journal is recommended.

IDMSX

IDMSX allows one or more areas to be dumped while they are being processed - even if they are open for exclusive update.

In-process dumps can be restored using the Restore Utility and rolling forward the journal in use at the time of the dump. The database can be in use during such a restore, but the areas being restored must not be accessed by success units during that process.

Central Journal

All the success units using IDMS can share one main journal, often called the central journal. This journal is optional. When one is used, IDMS produces its contents automatically.

The journal's contents correspond to the READY - Process - FINISH sequence for each updating success unit:

- READY causes a beginning of job marker to be recorded on the journal. This begin marker contains such details as program-id, date and time, areas in use and locking details
- Changes to the database produced by DML verbs like STORE and MODIFY cause entries in the journal. These are before images showing the records as they were and/or after images showing the results of the changes
- FINISH causes an end of job marker to appear on the journal. The end of run statistics (defined in Chapter 9) are written at this time.

The journal can then be used for:

- **Rollback** ie applying before images to the existing database to eliminate incorrect updates (see Figure 2)
- **Rollforward** ie applying after images to a database that has been restored from a dump file, bringing the contents up to a recent state (see Figure 3). This is necessary if the existing database becomes corrupt and unusable eg through failure of a disc pack.

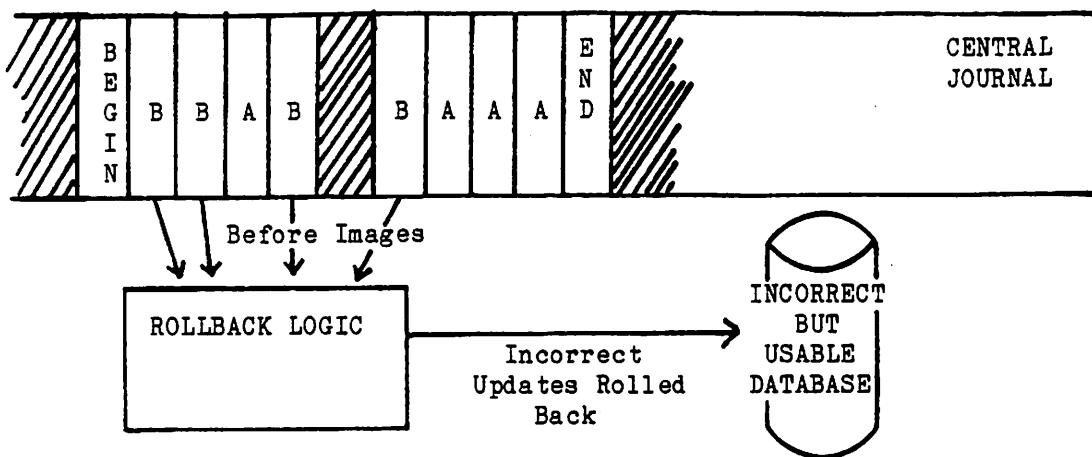


Figure 2 The logical process of rollback

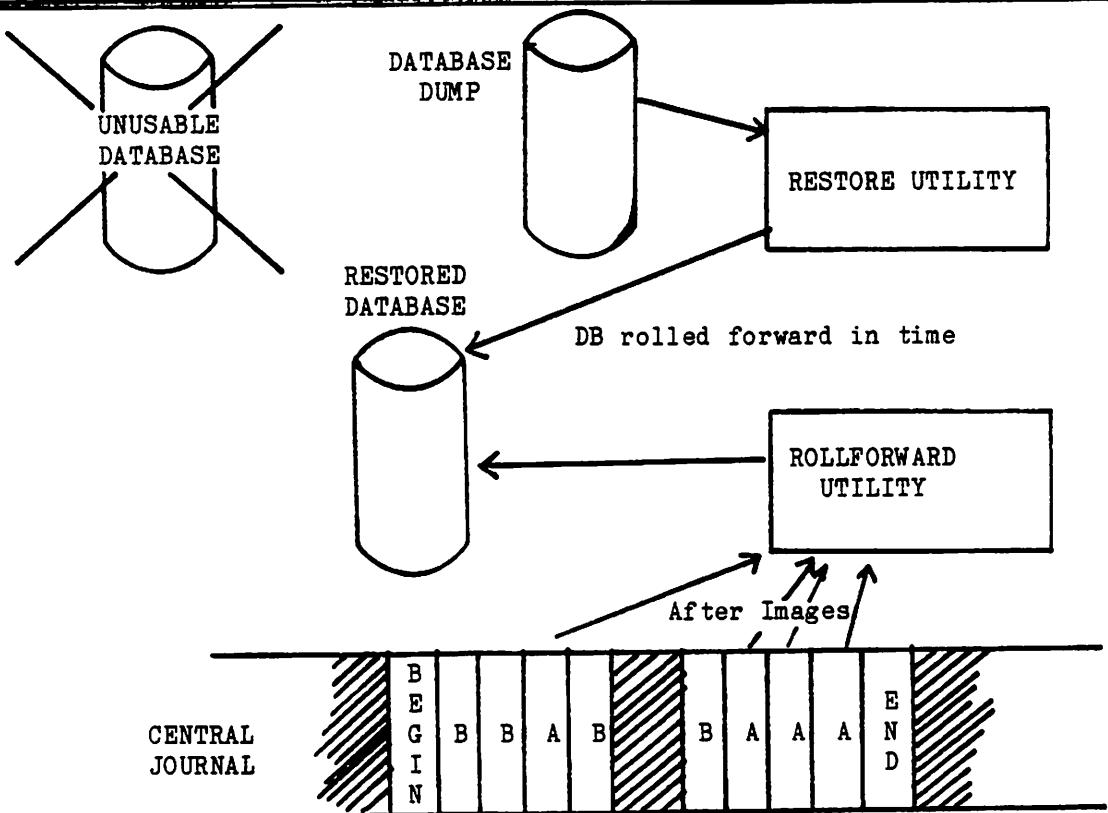


Figure 3 The logical process of rollforward

The unit of journalising is a database line - ie an unfragmented record, or a root or fragment of a fragmented record.

When a record is updated, before and after images are stored in the journal buffer. Each before/after image includes the database key and line index entry. In a shared service, the before and after looks of concurrent success units are interleaved on the journal.

~~H~~ The contents of the buffer are written to the journal when

- the buffer is full
- a page copied back to the database has related images in the journal buffer
- a success unit FINISHes.

~~X~~ The block written to the journal includes all images in the journal buffer plus IDMS control information, such as changes to CALC chain pointers.

Success units that do not update database records, regardless of their nominal usage modes, cause no entries in the journal.

~~ONLY GOT BEFORE LOOK~~
Quick Before Looks ONE PER AVU

~~NEEDS TO BE
BIG ENOUGH TO
HOLD ONE
SUCCESS UNIT
ROLLBACK
SUCCESS UNIT~~
If an updating success unit FINISHes successfully, its before looks are unlikely to be of further use. The principle of quick before looks is to give each updating program its own reusable file space, holding before images for the current success unit only. When one success unit completes successfully, its before looks are overwritten by the next success unit. After looks for all success units are held on one or more separate journals.

The benefits of quick before looks are:

- journal volume is greatly reduced
- mill usage is reduced
- rollback of one failed success unit does not delay any others, which can continue unaffected using the after-look journal and their own quick before look file.

IF SNL FULL (USED
SECTIONED
JOURNAL)
EXF
EXSNL
RBCN (UNFIN)

Quick before looks are available on IDMSX only. The software allocates a QBL File to each virtual machine.

You cannot use QBL Files with a central journal. Instead, you must supply one or more area journals to hold the after looks.

Area Journals (IDMSX) ONLY GOT AFTER LOOKS CAN ONLY HOLD FOR WHOLE

On a very active database, journalising can become a bottleneck. The idea of area journals is to prevent this by having several journals, each associated with a group of areas in the database. An area journal holds after looks for its associated areas, before looks going to QBL Files.

You may find, however, that a single area journal holding after looks for the whole database is sufficient.

Rollback and Rollforward

Automatic Rollback

Automatic Rollback is logic embedded in the normal IDMS run-time code. It will roll back one success unit that has failed while other success units are still running.

It can be called by IDMS itself, by the TP software or by the application code via the DML statement FINISH AFTER ROLLBACK.

Figure 4 shows Automatic Rollback working in the three user environment from Figure 1.

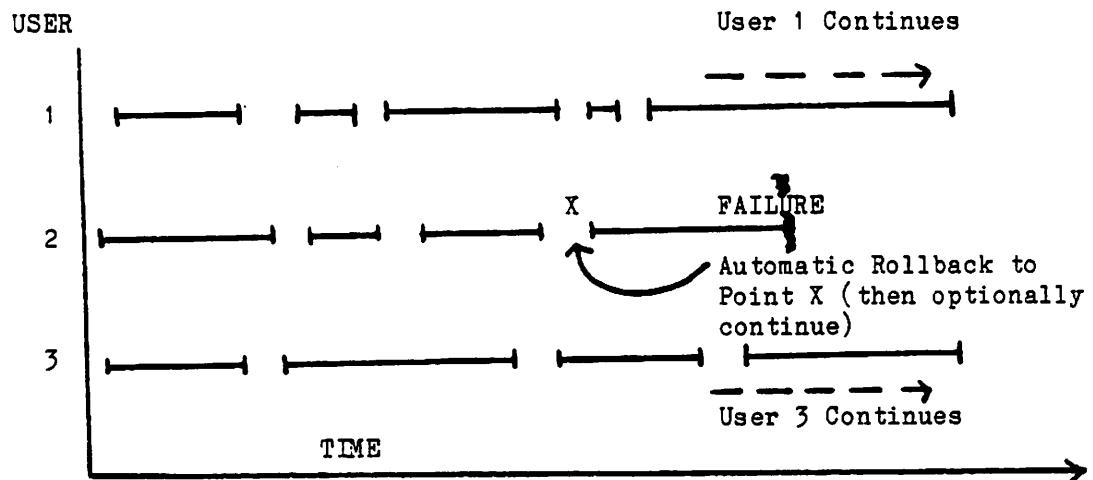


Figure 4 Automatic Rollback on one success unit

NO265-3

IF BUFFERS BIG ENOUGH MAY
NEVER NEED TO WRITE QBL'S ^{7 - 9}
IF USE DELAYED UPDATING (See 7-12)
CAN'T ROLL OUT GOOD SU'S WITH
AREA JOURNALS

Rollback Utility

The Rollback Utility is a separate program which may be run manually or called automatically. Runtime parameters decide just what the utility does. It can:

- rollback a number of success units that have failed simultaneously eg a systems crash leaving the database incorrect but physically usable
- print the contents of the journal.

The main use of the utility is to rollback all success units on a given journal that did not reach end of job (PROCESS = UNFINISHED). The utility writes abort markers on the journal for each success unit that it rolls back. Figure 5 shows the Rollback Utility in use.

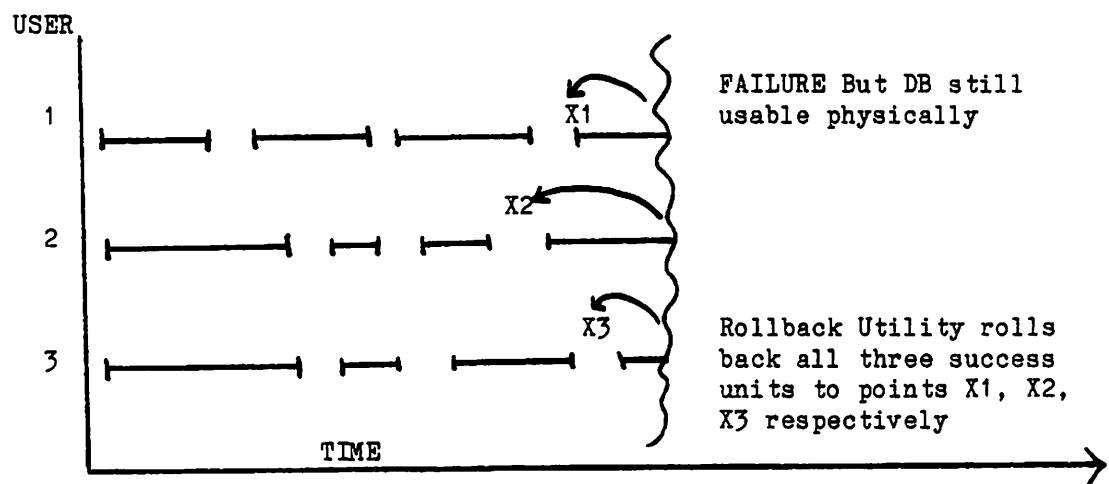
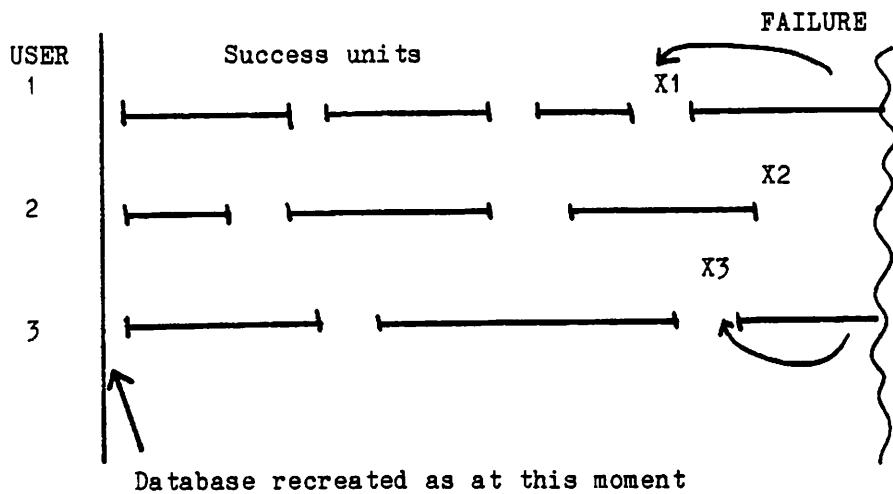


Figure 5 Rollback Utility rolling back all current success units

Rollforward Utility

The Rollforward Utility is also a separate program. It is usually used manually in the process of recovering from a serious failure - ie one that requires the database to be restored from a dump. It brings the database forward in time by writing after images to the restored dump. Parameters allow the process to be limited to specific areas or to be applied to the whole database. Also the process can stop at a selected quiesce point - ie one where no update units were running so that the database was consistent at that moment - or at other selected points. Figure 6 shows an example.

Like the Rollback Utility, the Rollforward Utility is able to print the journal file.



- A) Restore a dump to a newly initialised disc
- B) Rollforward applying after images to point X2
- C) Rollback users 1 and 3 to points X1 and X3
- D) Continue from points X1, X2 and X3

Figure 6 Example of recovery using Rollforward Utility

Delayed Update

A logical alternative to the Rollback process (which undoes incorrect updates) is not to do the updates at all until we are sure they are correct - ie until the success unit has FINISHed successfully. Any failure occurring before FINISH has no effect on the database since the success unit has not changed it at all. (FINISH AFTER ROLLBACK has a null effect, except for an abort marker on the journal.)

Standard IDMS delays updating until the end of the success unit provided:

- the journal buffer is large enough to hold all before images for the success unit
- there are enough database page buffers to hold all the pages the success unit accesses. (The number of buffers is set in the service description but may be varied via a parameter to IDMS_RUN.)

In this case, however, you are taking advantage of the way the software works. IDMS does not know you are doing this. So it still writes before looks to the journal when the success unit finishes.

IDMSX

IDMSX offers a fuller implementation of delayed updating. Before images are not journalised.

To obtain this version of delayed update you must use a QBL File, with one or more Area Journals for the after looks - so the before and after looks are in different journal buffers.

Of course IDMSX cannot delay updating to the end of the success unit if the amount of buffer space is inadequate. If an updated page has to be written back, the before looks are written to the QBL File first - whether the journal buffer is full or not. (After looks are only written to the Area Journal when their buffer is full, or on FINISH.)

Provided the number of success units that switch to QBL in this way is a small proportion of the total, IDMSX delayed updating provides significant savings in OCP time and number of journal transfers.

Duplexing (IDMSX)

With IDMSX you can have two copies of:

- the Database
- the various Journals ← *VERY important*
- the Database Diary

in any required combination.

The IDMS software automatically writes to both copies and reads from one copy (alternating between the two copies in the case of the database).

The purpose of duplexing is to provide greatly improved resilience. If one copy of, say, a journal becomes unusable, the service can still continue in simplex mode with the surviving copy.

Facilities exist for reconstituting the failed copy and bringing it back into use.

Controlled Use of IDMS Environment

Database Diary

The Database Diary is a file upon which IDMS can keep a record of database usage. Database Diary records describe noteworthy events, such as:

- Start and end of IDMS service
- Start and end of each application
- Files and/or areas in or out of use
- Utilities used
- Recovery facilities involved
- Journals used (IDMSX only).

* In addition, VME allows users to write records to the Diary. *

The Database Diary helps the database administrator to plan roll-forward recoveries and to keep control in general - eg by printing out the Diary to see what is happening during an IDMS service.

In theory the Diary is optional but, as it is essential for database consistency checking, running without it is inadvisable.

*Diary cycles BUT WON'T OVERWRITE current
SS ENTRY.
MAYBE ISN'T BIG ENOUGH*

Database consistency information is held in the Database Diary. When an IDMS service closes, a record in the Diary indicates whether or not all updating success units completed successfully. This consistency checking mechanism is the reason why we should run even a single program within an IDMS service.

When a service is started it tests this record. If the record shows that the last service to use the database left it in an inconsistent state, the Rollback Utility is invoked. Once the database is consistent, it records this fact in the Diary.

IDMSX

In IDMSX, the Rollback Utility consults the Diary to find out which Quick-Before-Look files to use.

USING THE IDMS RECOVERY MECHANISMS

Note. This section concentrates on Recovery in Batch environments. Recovery in TP environments is considered later in this chapter.

Local Failure

Here a single success unit has failed, as in Figure 4, and our concern is to reverse any updating it may have done.

Some errors are known to IDMS but not to the application code. These include contingencies, which are hardware failures or events caused by the operator. In such cases, any updates are reversed automatically and the success unit aborted.

Other errors are known to the application code. These include:

- logic errors detected by the program
- non-zero error codes returned to the program which, in the view of the application logic, invalidate previous processing. An example of this is deadlock.

To recover from such errors, a Batch success unit can use either:

- PERFORM IDMS-STATUS which aborts the success unit and reverses any updating
- FINISH AFTER ROLLBACK which reverses any updating but allows the application code to continue eg with new READYs.

In all cases, any locks are held on until recovery is complete and the error is logged on the Database Diary.

System Failure

In this case, a number of success units may have been cut off in mid-flight since the whole service has come to a halt.

Errors that can create this situation include a system crash and some file failures.

For Batch Services, we can recover the database using the Rollback Utility with the parameter PROCESS = UNFINISHED. This will use the Journal to rollback all success units without an end marker, as in Figure 5.

File Failure

With standard IDMS, duplexing is not available so the service is bound to be affected if any of the database files, a journal or the database diary fails.

If a database file fails (after the usual automatic retries) the operating system informs IDMS, which will:

- take the file and the areas it supports out of the service. Any programs attempting to access those areas will fail
- Rollback the success unit that detected the error (on any other files it may have updated) so that the rest of the database is consistent.

We then recover the failed file by restoring a dump and rolling forward (see Figure 6). If the service has been continuing in limited mode mean-while, we can introduce the new file via the command IDMS_FILE_IN and a full service can then resume. The command automatically re-introduces the area(s) into the service as if IDMS AREAS_IN had been invoked for each one.

If a journal file fails on a write, there are procedures for replacing it. If it fails on a read, then the failure has occurred during a recovery process and the service must close down.

If the Database Diary fails, the service must close down.

IDMSX

Duplexing provides much greater resilience. Any or all of the above files may be duplexed. In this case IDMSX maintains the two copies automatically, writing to both but reading from them alternately.

If a failure occurs on one copy of the database, the service continues quite happily using the other in simplex mode. Meanwhile a replacement file should be placed, initialised and passed to IDMSX via the command IDMSX_FILE_IN. IDMSX then reverts to duplex mode, copying the data to the new file automatically.

If one copy of a duplexed journal fails, the IDMS_NEW_JOURNAL_FILE command can be used to force a quiesce point, release the remaining copy and continue with two new duplex files. The surviving journal should be copied as soon as possible.

Similar facilities are available if a database diary fails.

Duplexing is likely to be most popular on the Diary followed by duplexing the Journal(s).

CHAPTER 8

ITEMS AND TPIES

N0265-6

TP

Introduction

A success unit in TP terms is a **phase** ie a message pair consisting of one input message from the terminal to the mainframe and of (usually) one output reply.

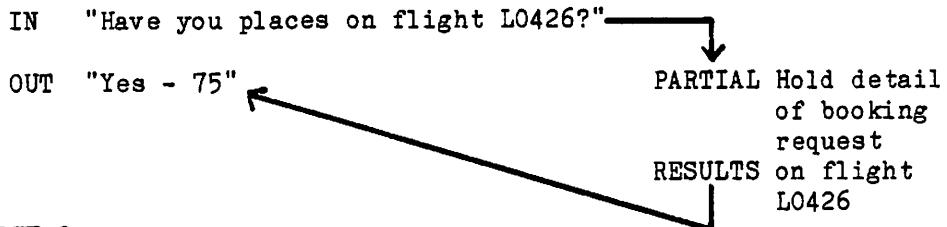
One or more phases make up a single user transaction.

Single phase transactions are often simple enquiries, while multi-phase transactions often consist of initial enquiry phases followed by a final update phase. Indeed, multi-phase transactions should be organised in this way, with any updating confined to the last phase. This approach simplifies recovery.

In multi-phase transactions, you often need to save data from one phase to the next. Such data is known as partial results. TPMS has a mechanism for this purpose using a special RECOVERY SLOT FILE.

The system provides locks on data within a phase, not across phases. Figure 7 shows a 2-phase transaction

PHASE 1



PHASE 2

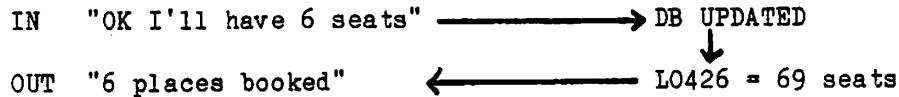
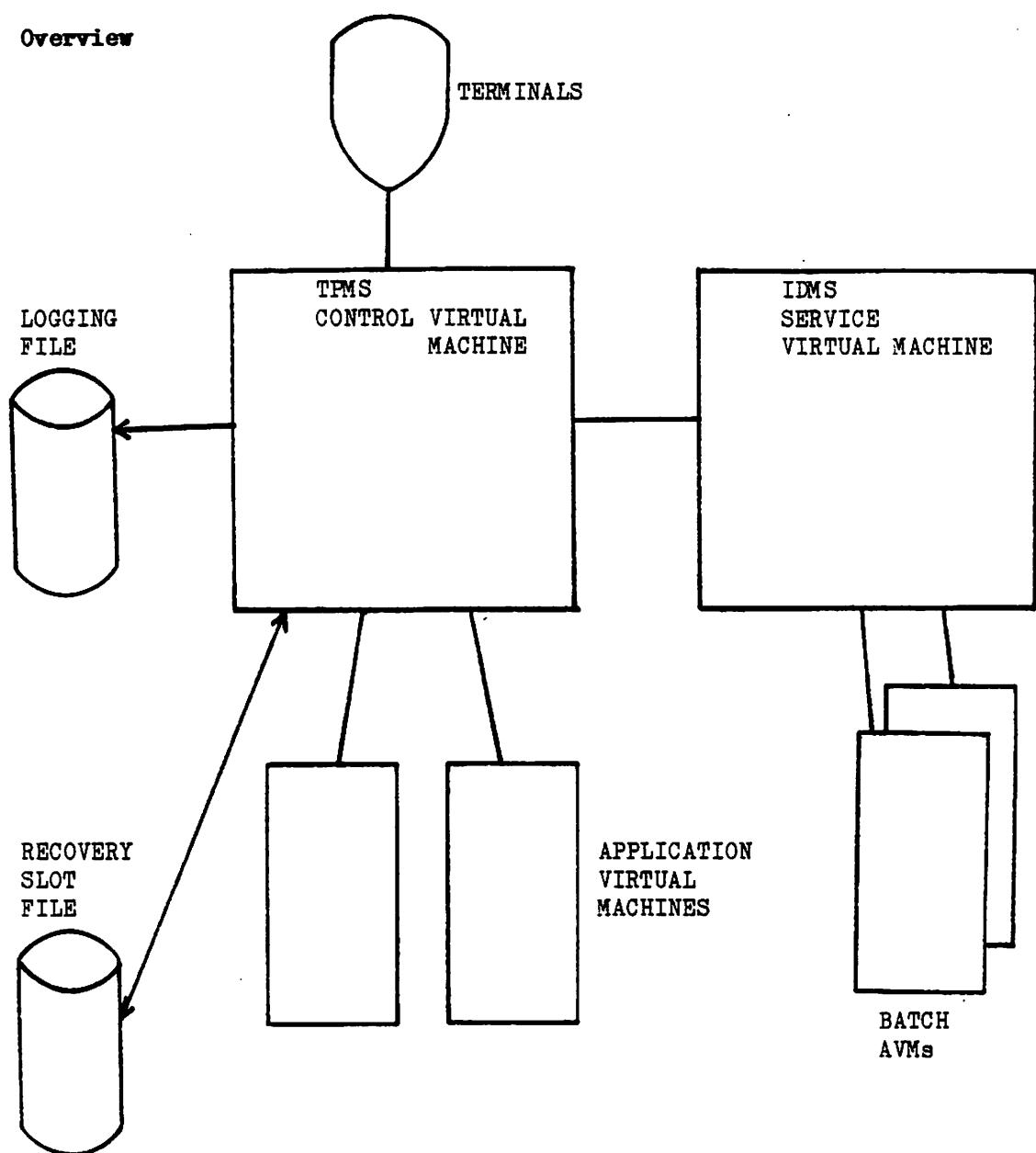


Figure 7 Multi-phase transaction using partial results

Several multi-phase transactions can run concurrently at the same terminal. Although these interleaved multi-phase transactions are less typical, the TP software can handle them.

STRUCTURE OF TPMS

Overview



Control Virtual Machine

The CVM is a system virtual machine set up when a TP service is run. Its major functions are:

- To receive each input message, establish its type and pass the message to a suitable user-written AVM
- To optionally log input messages, replies and statistics to the **logging** file
- To control the pool of AVMs, increasing or decreasing their number within specified limits according to the load on the system
- To pass replies back to terminals
- To handle initialisation of the TP service and recovery after breaks
- To communicate with the IDMS service VM.

AVM Pool

Each user AVM can handle one phase at a time. If we wish to handle six phases at once, we need to have at least six AVMs in being simultaneously. In a TP Service, many AVMs of many types can exist at the same time. Each AVM type has one or more message types associated with it. The system can deal with several occurrences of a message type at the same time by replicating the relevant AVM type to provide the required simultaneity. The total number of AVMs in being at one time can also be varied dynamically by the system according to the message load.

Files Used

Apart from the data files, which may be RECMAN (ie conventional) and/or IDMS, a TP Service uses a number of files for other purposes.

The Recovery Slot File forms the basis of recovery within TP. It is a direct access file holding different types of slot including:

- a) A single **status slot** containing information about currently connected terminals, whether recovery is required and information to control duplexing, if applicable (the Recovery Slot File can be duplexed).
- b) A **terminal slot** for each terminal that is, or might be, connected to the TP Service. It contains, among other things, the last input message and reply for that terminal and any current partial results.

Sequence of Events in Handling a Phase

Let us now follow one phase through the system. Figure 9 summarises the sequence of events, which is described below:

1. The input message is passed to the CVM, which allocates a unique 'Phase-Id' to the message. (Later, if the message causes an IDMS database to be updated, any relevant journal blocks will include the Phase-Id automatically.)
2. The CVM establishes the message type. The message is now ready to be passed to an AVM - together with a 64-byte header containing system information like terminal of origin, date/time received and message type name.
3. The CVM schedules the message into a queue for processing by an AVM of the correct type.
4. The AVM is activated. System code at the beginning of the AVM examines the header. If the message is part of a multi-phase transaction, any partial results from the previous phase are retrieved from the relevant Terminal Slot on the Recovery Slot File.
5. The application code in the AVM for this message type then processes the message.
6. System code at the end of the AVM is then entered. Its main purpose is to send the message(s) to the CVM for onward transmission to the terminal(s) and to perform appropriate recovery and update procedures. These procedures, known as SECURE and COMMIT, are considered in detail in the next section. Any partial results are saved in the Recovery Slot File.
7. Control returns to the CVM, which sends the reply or replies after handling any formatting and statistics that may be required.

Structure of a TP Success Unit

Writing a TP program in COBOL is very simple. The sequence of major verbs is:

```
ACCEPT input message
READY(s) causing begin marker on Journal (if update)
DML Verbs accessing the database
DISPLAY reply
FINISH
STOP RUN
```

Figure 10 Outline of a COBOL TP Program

The sequence from ACCEPT to FINISH is performed for each phase. These verbs get the input message (ACCEPT), declare the locking requirements (READY), process and then output the reply (DISPLAY). Unlike a Batch Success Unit FINISH neither releases locks nor writes an end marker to the journal. These are delayed until the system code at the end of the AVM - FINISH merely prevents further DML verbs being obeyed.

Components of an AVM

In a TP environment, there are effectively four parts to an AVM:

- TP_MSG_BEGIN, the system code at the start
- TP program, ie applications code, including the ACCEPT and DISPLAY verbs as in Figure 10
- TP_MSG_END, the system code at the end which handles the secure and commit procedures (see below)
- TP_APPLICATION_ERROR_HANDLER, system code called by the applications programmer or by IDMS to inform the TP software that there is an error in the success unit. As a result, TP will arrange for an Automatic Rollback on the success unit. This is considered in more detail in the section on using the IDMS Recovery Mechanisms with TP.

TP AND RECOVERY

Secure and Commit

The concepts of Secure and Commit are used in the design of TP_MSG_END code.

Commit implies a decision:

"This processing is complete now and I do not propose to repeat it."

In fact the DML verbs COMMIT and FINISH mean this, marking the end of a success unit in the program logic. However we are using the term 'commit' in a general sense here, not limiting our attention to the DML verb of that name.

In a TP success unit we have three types of result:

- the reply sent to the terminal
- updates to the IDMS database
- updates to any conventional files.

All of these need the decision about commitment.

Before we can take this decision, we must be sure that, should something go wrong after the point of commitment, we have the means to recover - to bring back into being what we decided to commit to.

Secure means ensuring that such a recovery is possible for all three types of result.

For IDMS/conventional files this implies ensuring that the after-looks have been recorded successfully on the IDMS Journal/non-IDMS Recovery File. For the reply, it means ensuring that the reply has been written successfully to the TP Recovery Slot File.

Once we are sure we are secure, we can commit. The logical sequence is:

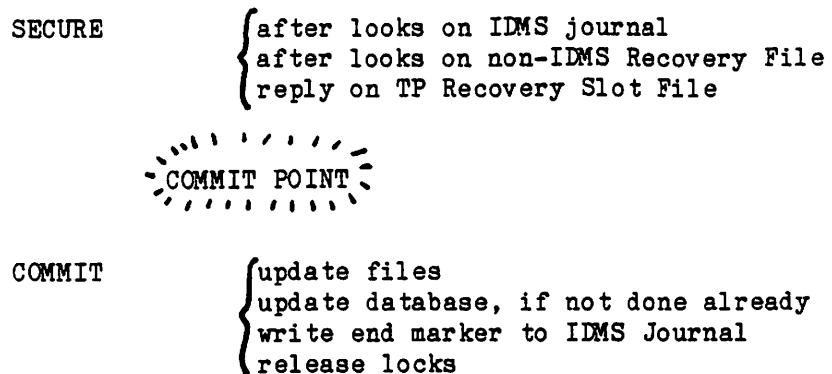


Figure 11 Sequence of Secure/Commit

What you need to understand is the concept of the **Commit Point**. If the success unit reaches and achieves the work at that point - it **does not need to be repeated**. If the success unit fails to reach this point, it must start again from the beginning. Figure 12 illustrates the concept.

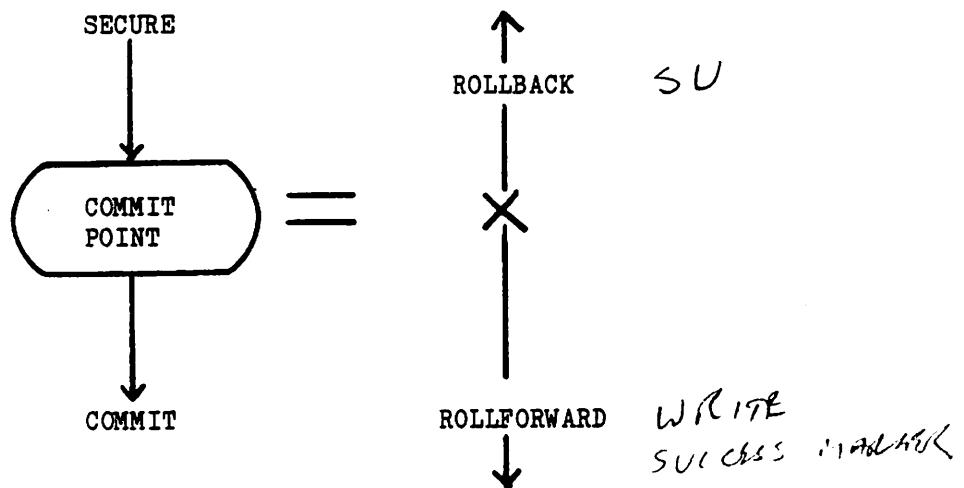


Figure 12 The Commit Point

This is, of course, the basic idea of a success unit, but restated in a very precise form.

For each TP success unit in being when a failure occurs, the recovery process decides what to do on the basis of whether the success unit had reached the Commit Point.

If the success unit had reached the Commit Point, its logic so far is accepted - ie some Rollforward process is involved. If it had not reached the Commit Point its logic must be repeated - a Rollback process. Notice that the terms Rollforward and Rollback are used here in a general way. Depending on the actual point of failure and the recovery options chosen, Rollback and Rollforward may be null events.

With Batch success units the situation is much simpler. The FINISH or COMMIT verb is the Commit Point, at which the end marker is written to the journal, locks are released etc.

TP/IDMS RECOVERY

Local Failure

When IDMS detects an error, such as a contingency or hardware failure, it calls TP_APPLICATION_ERROR_HANDLER (TAEH) to reverse any updates and abort the success unit.

If a TP success unit detects an error, it must use PERFORM IDMS-STATUS to report an error to IDMS and TP. Again any updating is reversed, the IDMS-STATUS code causing a call to TP_APPLICATION_ERROR_HANDLER. If the error was due to deadlock the message will then be reprocessed.

System Failure

The first thing to do is to bring the TP service back up. The TP software then passes to IDMS a list of committed messages - ie those which reached their commit point - giving the Phase-Id of each. The TP service retrieves this information from the Recovery Slot File.

IDMS then examines its journal(s), in which each block produced by a TP success unit includes the relevant Phase-Id.

Taking a simple view, there are three possible situations:

1. A success unit has an end marker on the journal. Whether TP or Batch, it has been committed and its updates are not reversed.

2. A success unit has no end marker and is not in the TP list of committed messages. This is either a Batch success unit, or a TP success unit that did not reach its commit point. Any updates are reversed ('ROLLBACK' in terms of Figure 13).
3. A success unit has no end marker but is in TP's list of committed messages. This is a TP success unit that reached the TP commit point but did not reach the subsequent IDMS commit point, where the end marker is written. This success unit has its after-looks applied to the database in order to complete the processing ('ROLLFORWARD' in terms of Figure 13).

This is a simplified description of the procedures followed. For instance, the journal may need tidying to ensure that the current recovery process does not leave loose ends that could affect subsequent use of the journal for recovery.

Some success units in a TP Service may use Recman files either instead of or as well as IDMS. If so, TP ensures that such Recman success units are rolled back or forward as appropriate, using similar procedures to those described for IDMS.

Although the Recovery procedures will handle the situation, life will be simpler and more efficient if a success unit needs recovery of IDMS or of RECMAN files, but not both.

IDMSX

The journal needs tidying if delayed update is used with IDMSX. In this case there are no before-looks on the journal. So any after-looks recorded by failed success units would be applied in any later rollforward - producing an inconsistent database. To avoid this problem, IDMSX retrieves the current correct state of the relevant data from the database and writes it as more after-looks after the incorrect ones. Hence any later rollforward will have a null effect as far as the failed success units are concerned.

File Failure

With standard IDMS, duplexing is not available so the system is bound to be affected if any of the database files, the journal or the Database Diary fails.

A TP service can only continue while a file is missing if file sub-setting is used. This entails specifying on the TP Parameter File the precise file requirements for each AVM type. If this is not done, TP assumes that all IDMS AVMs need all the IDMS files it knows of - and the TP Service cannot continue, or even start, unless all the files are present.

Another optional feature of TP inhibits message types automatically while any file they require is missing. Again this requires entries on the TP Parameter File.

If a journal file fails on a write, there are procedures for replacing it. If it fails on a read, then the failure has occurred during a recovery process and the service must close down.

If the Database Diary fails, the service must close down.

IDMSX

Duplexing provides much greater resilience. Any or all of the above files may be duplexed. In this case IDMSX maintains the two copies automatically, writing to both but reading from them alternately.

If a failure occurs on one copy of the database, the service continues quite happily using the other in simplex mode. Meanwhile a replacement file should be placed, initialised and passed to IDMSX via the SCL command IDMSX_FILE_IN. IDMSX then reverts to duplex mode, copying the data to the new file automatically.

If one copy of a duplexed journal fails, the IDMS_NEW_JOURNAL_FILE command can be used to force a quiesce point, release the remaining copy and continue with two new duplex files. The surviving journal should be copied as soon as possible.

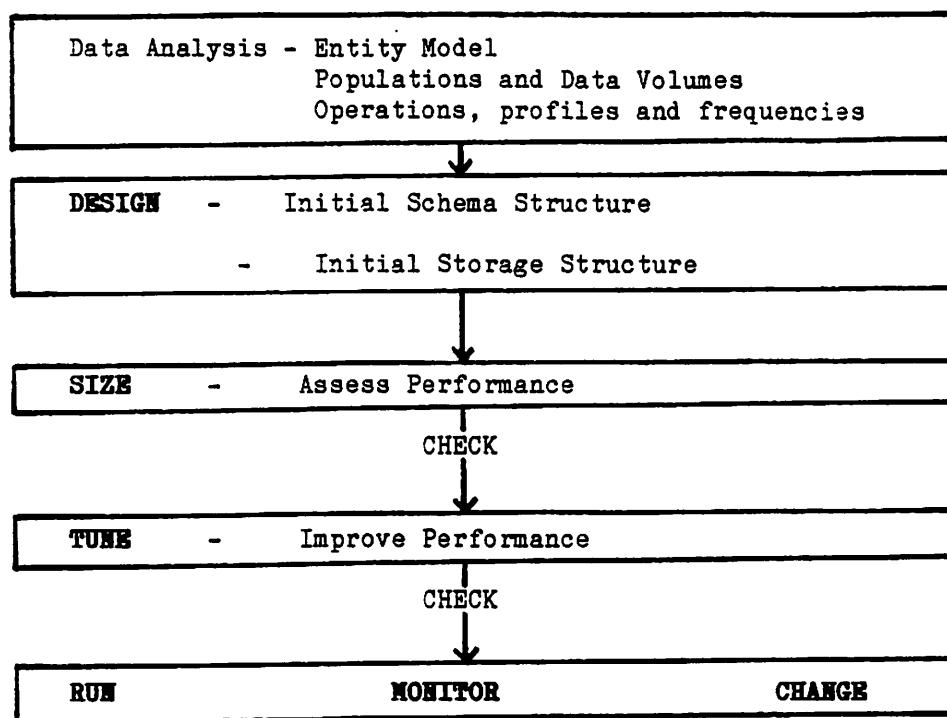
Similar facilities are available if a database diary fails.

CHAPTER 9

STAGES IN DATABASE DESIGN

DATABASE DESIGN OVERVIEW

There are several stages to database design:



The starting point is the information produced during the pre-design phase of data analysis.

The first stage in the Database design process is to produce an initial design whose performance can then be assessed and improved in the later stages. This first stage involves the following procedures:

1. Map the Entities and Relationships of the Data Model to the Record types and Set types.
2. Make decisions on Keys, on Set Orders and on Insertion/Retention Class to complete the Schema.
3. Make physical decisions on Placement, Set implementation mechanisms, Areas and Page size(s) so that an initial Storage Schema can be produced.

The second stage is to assess performance, under such headings as Media Size required, Number of Disc transfers etc. This stage is likely to show up inefficiencies in the initial design.

The third stage is to improve performance and tune the design. This may involve changing design options and using techniques that will improve the performance of key applications.

It is impossible to design an IDMS database in such a way that it will be best for every application. At the tuning stage you may well have to optimise the design for high priority applications at the expense of lower priority ones. Therefore priorities must be established in as objective a way as possible.

When the database design appears to be acceptable, the final stage is to check it by loading and running a test database and examining the run-time statistics.

The entire design process is an iterative one, with each stage being examined, checked objectively and improved until the design criteria are met.

The rest of this chapter examines each stage in the design process. It assumes that the Data Analysis phase has produced a Data Model containing as a minimum requirement:

Nature of Data:

Entity	- description identifying and data attributes volumes
Attribute	- description format
Relationship	- description nature (1:1, 1:M etc) volume (how many is many?)

Use of Data:

Event	- description frequency
Operation	- description frequency pathway through the data expected response time any other priority factors

There should also be an overall Systems Plan giving the timescale and sequence for implementing the various applications. This enables you to keep future requirements in mind at each stage.

FIRST STAGE - PRODUCING AN INITIAL DB DESIGN

The Schema

Mapping down from entities and relationships

Relationships:

- map each 'one to many' relationship to a set type
- explode any remaining 'many to many' relationships into two set types and a relationship record
- explode any remaining relationships between entities of the same type eg for Bill of Materials.

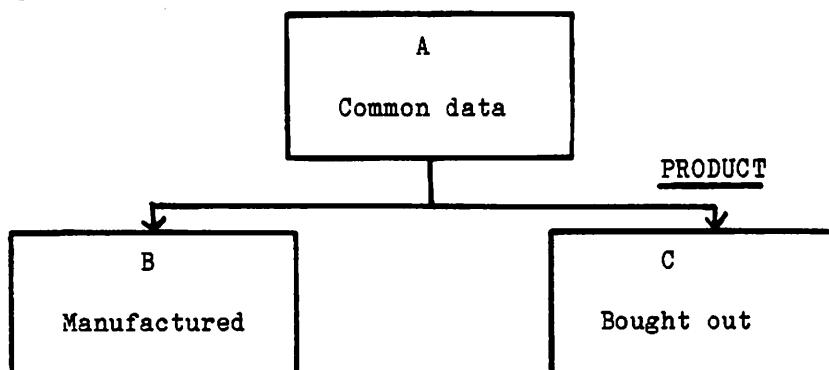
Entity types:

- map each entity type to one record type.

Alternative mappings

The inexperienced IDMS designer would probably be wise to leave the mapping process at that for the time being. Alternatives will then be brought to his attention by the later Assessment of performance and Tuning stages. However, an experienced designer may well try to save time by taking short cuts at this point. Among the things he might try are:

1. Mapping one entity to several record types if groups of fields can be missing on occasion. For example, in a part record some fields may be present all the time, others when the part is bought out, others when the part is manufactured. A simple and efficient way to handle this is to segregate the fields in different record types and structure them like this:

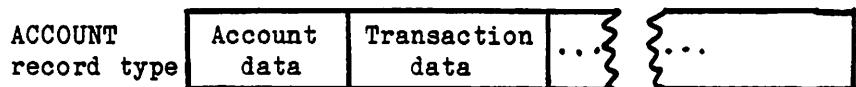


Each A record owns either one B record or one C record. A multi-member set could be used, as above, to handle this.

2. Mapping several entities to one record type if the volume of entity occurrences is large. In this case, mapping 1 to 1 would use up a lot of space in pointers and line index entries. Suppose we have more than 1 million ACCOUNT entities each owning 20 TRANSACTION entities:



This may be better implemented as a single ACCOUNT record type with an OCCURS clause for the transactions, thus reducing the amount of backing store required in suitable cases:



(The record may be fixed or variable length.)

* Note that QuickBuild products do not have subscripting facilities. *

3. Add extra Record/Set types to the Design to hold reference tables, re-start information, eg last good input transaction number etc.

Keys

Every Record type can have one Selection Key (several in IDMS X, where the possibility of an Ordered ASCENDING/DESCENDING Key also exists). However, it is certain that a Key on every Record type would be too inefficient to support. In standard IDMS, it would mean that every Record type would have to have a Placement of CALC or PAGE DIRECT. The Records would then be so scattered that Set Walking would be a disc access per Record retrieved. In IDMS X some keys could be supported by Record Indexes but again these could be very bulky to hold and time-consuming to update.

The number of Records with Keys should be limited therefore. Consider the important Record types (Customer, Product etc) and make the field corresponding to the relevant Entity's identifier a key. Put in any further keys that vital transaction types demand. But otherwise leave the matter of entry points for detailed examination later (see under PLACEMENT below).

*IF > Sort Nine Sets
INDEXED*

Other decisions

Set Orders should be something neutral like NEXT or FIRST at this stage. During Tuning other alternatives can be examined. In the same way, Insertion/Retention class should be AM unless obviously inappropriate, eg as in BAD-DEBT to a CUSTOMER.

THE STORAGE SCHEMA

Introduction

The decisions made so far are LOGICAL although we may still go back and alter them later in order to improve performance, eg Set Order, Data Model mappings etc. However, for the moment the Schema is complete and Storage Schema decisions must now be made.

Placement

The choice of placement for a record requires careful consideration. The principle given here is that the placement should depend on the activity pattern of a record type. You can extract the information you need from the event and operation characteristics determined during the Data Analysis phase.

Activity Patterns

Drawing up an activity list, which reflects the activity pattern of a record type, is a three stage process:

Stage	Data required
1. Skeleton activity list	Ways in which record is accessed
2. Detailed activity list	Volume of each activity
3. Detailed activity list arranged in priority sequence	Any priorities given to activities

You can then decide on the placement.

Methods of Accessing Records

First of all, check the operations' pathways against the data structure, now in simple Schema diagram form. This has two benefits:

- it ensures that all the records and sets are present in the structure and able to support the operations
- it shows the effect of the operations on each record type in the individual pathways.

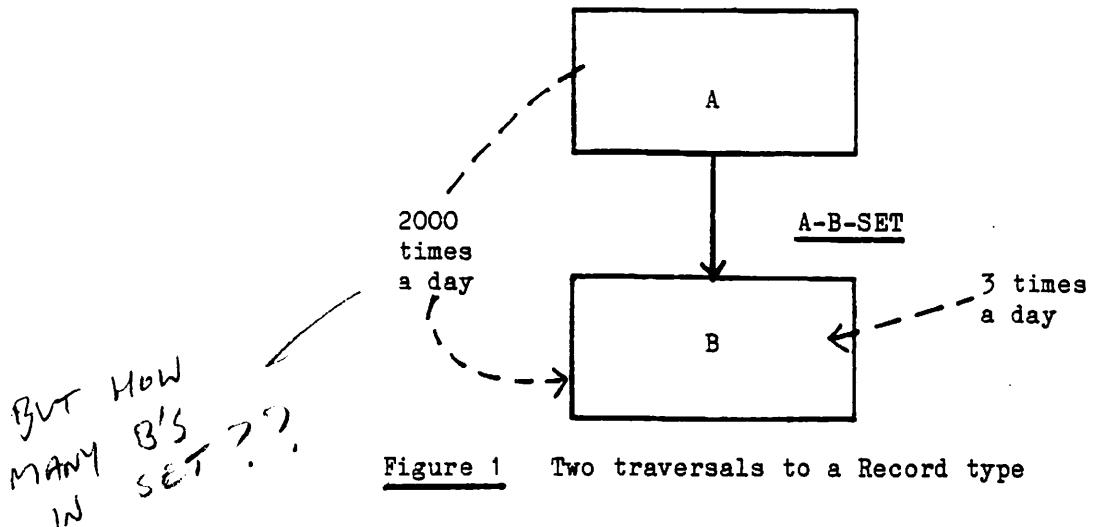
Now you can examine the various ways each record type is accessed. Traversals are a useful concept when establishing types of access.

A **traversal** is an access to one record.

There are 4 basic types of traversal. The first two listed here relate to transaction driven systems and are by far the most common. The other types of traversal parallel the access methods within conventional file systems ie serial or sequential:

- Direct to the record from outside the structure
- Using a set relationship from an occurrence of a different record type. The record so accessed may be the owner, or a member, of the set
- Serially accessing records (an area scan). This usually has lower priority than other methods of access and is not greatly affected by different placements. Hence we shall ignore this type
- Sequentially accessing records. This can be achieved in several ways, the first two of which are usually covered by other types of traversal:
 - by extracting data and sorting
 - by implementing a sorted set
 - by the use of SEQUENTIAL placement in IDMSX.

The concept of establishing a Record type's PLACEMENT and additional Entry Point requirements by examining an activity list (ie a list of Traversals) for that Record type is very simple. In Figure 1, it is obvious that Record B must be VIA the A-B set and that the Entry Point requirement must be met by some other mechanism than Placement, eg an Extract Realm scan, a Record Index etc. (This is assuming that frequency is the correct measure of relative importance here.)



However, this simple concept is made complex by three factors:

1. Bulk

There tend to be a lot of different pathways through a shared database. Establishing a shared database Activity lists for every Record type can be a very large job. In practice it may be necessary to concentrate on the most important pathways, ie those for high activity, for on-line update.

2. Multiplication

Where frequency of access is being computed, it is necessary to allow for a multiplication effect. For example in Figure 2, the pathway shown starts by representing 3 accesses to A a day, but represents 12 Accesses to B because 4Bs are obtained each time, and 120 accesses to C because 10Cs are obtained from each B.

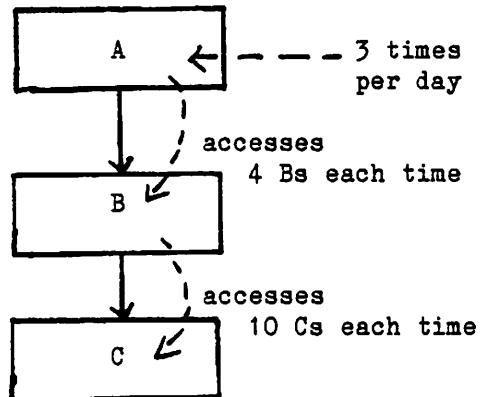


Figure 2 Multiplication of accesses

3. Access to Owner

A Traversal is an access to a Record. However, if the access is to the Record as an Owner of a Set, then the Traversal will not affect the Placement decision on the Owner. But it could affect the decision on the Member Record. In figure 3 the Traversal to A can be achieved WHATEVER placement A is given, as long as the A-B-SET exists. But the Traversal might affect the decision on B, ie B VIA the A-B-SET would give great efficiency for this Traversal.

So, Accesses to the Owner using the Set mechanism should be transferred from the Activity list of the Owner to the Activity list of the Member Record.

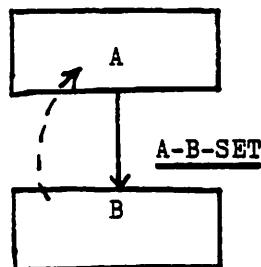


Figure 3 Shifting Traversals

Example of traversals

1. Skeleton activity lists

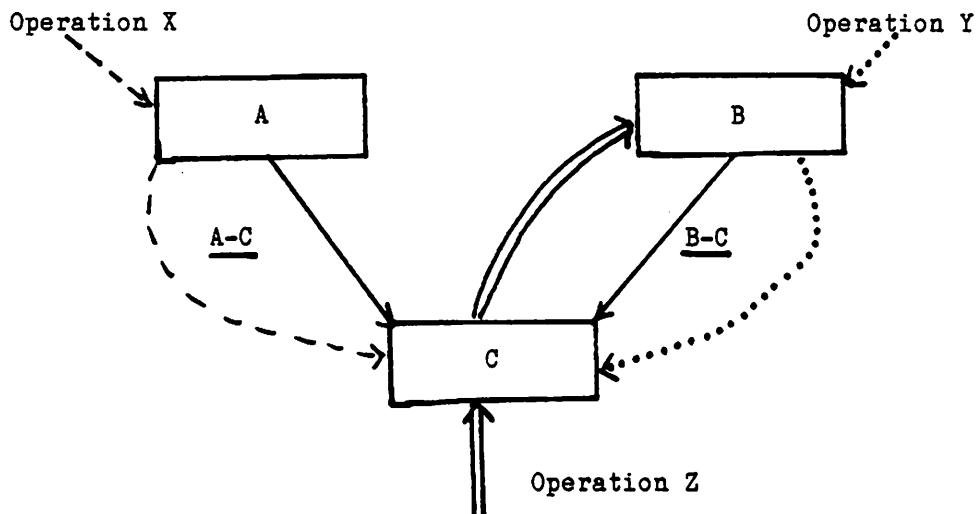


Figure 4 Pathways through a structure

Using the traversal concept, we can get a feel for the activity on each record type. Consider records C and B:

Traversals Record C	Direct to C (Op Z)
	Via A-C (Op X)
	Via B-C (Op Y)

Traversals Record B Direct to B (Op Y)
 Via B-C (Op Z)

The second traversal on B's list will not affect the placement of B, as B is the owner of the B-C set. It may well affect the placement of the member record C though. In this case, the traversal should be removed from the owner's list and included on the member's list. The amended lists are:

Activity list Record B Direct to B

2. Detailed activity lists

The next step is to calculate the volumes associated with each traversal from information provided by the Data Analysis phase, eg the frequency of an operation and the number of entity occurrences processed.

Suppose that, for our example, the operation details are:

Operation X - enters at A, moves to 50 Cs, 3 times a day

Operation Y - enters at B, moves to 10 Cs, 4 times a day

Operation Z - enters at C, moves to B, 160 times a day.

Applying these volumes to each traversal on the skeleton activity lists, we can produce detailed activity lists:

Activity list C Direct to C 160
 Via A-C 150
 Via B-C 40 } 200 combined to show
 Via B-C 160 } total activity
 (traversing on the set
 from C to B)

Activity list B Direct to B 4

Activity list A Direct to A 3

3. Detailed activity lists in priority sequence

Finally, each activity list should be organised into priority sequence. Priorities may depend on:

- the frequency of a traversal
- type of processing ie update or retrieval
- any other factor which gives one traversal priority over others eg:
 - . critical run time response
 - . an operation for which the response is critical to running the business (end of month, end of financial period etc).

In our example, if priority depends on the frequency of a traversal, the activity list for Record C becomes:

Via B-C	200
Direct to C	160
Via A-C	150

Deciding on the Record Type's Placement

When you produce your first attempt Storage diagram, the activity list provides important guidelines for:

- choosing a placement
- deciding, for a VIA record, which set to locate it via
- deciding whether a secondary access mechanism, like 1:1 CALC access or IDMSX record indexes, is necessary.

Figure 2 shows the logic for deriving placements from an activity list, for standard IDMS. For simplicity, it omits sequential accessing.

The equivalent logic for IDMSX appears in Figure 6.

Notice that if a record has no activity list, because all traversals to it are as an owner of a set, make its placement CALC.

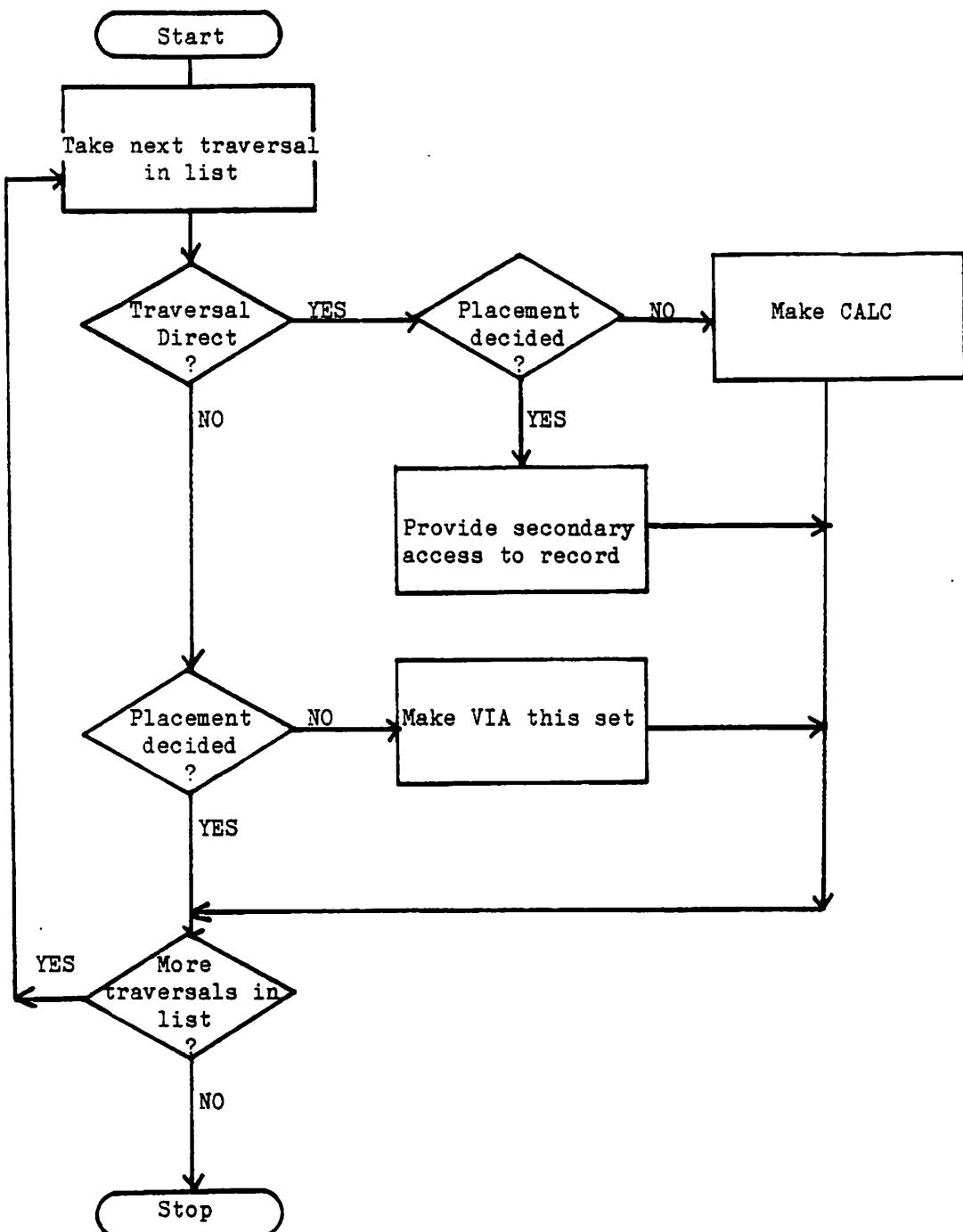


Figure 5 Logic for deciding Placement - Standard IDMS

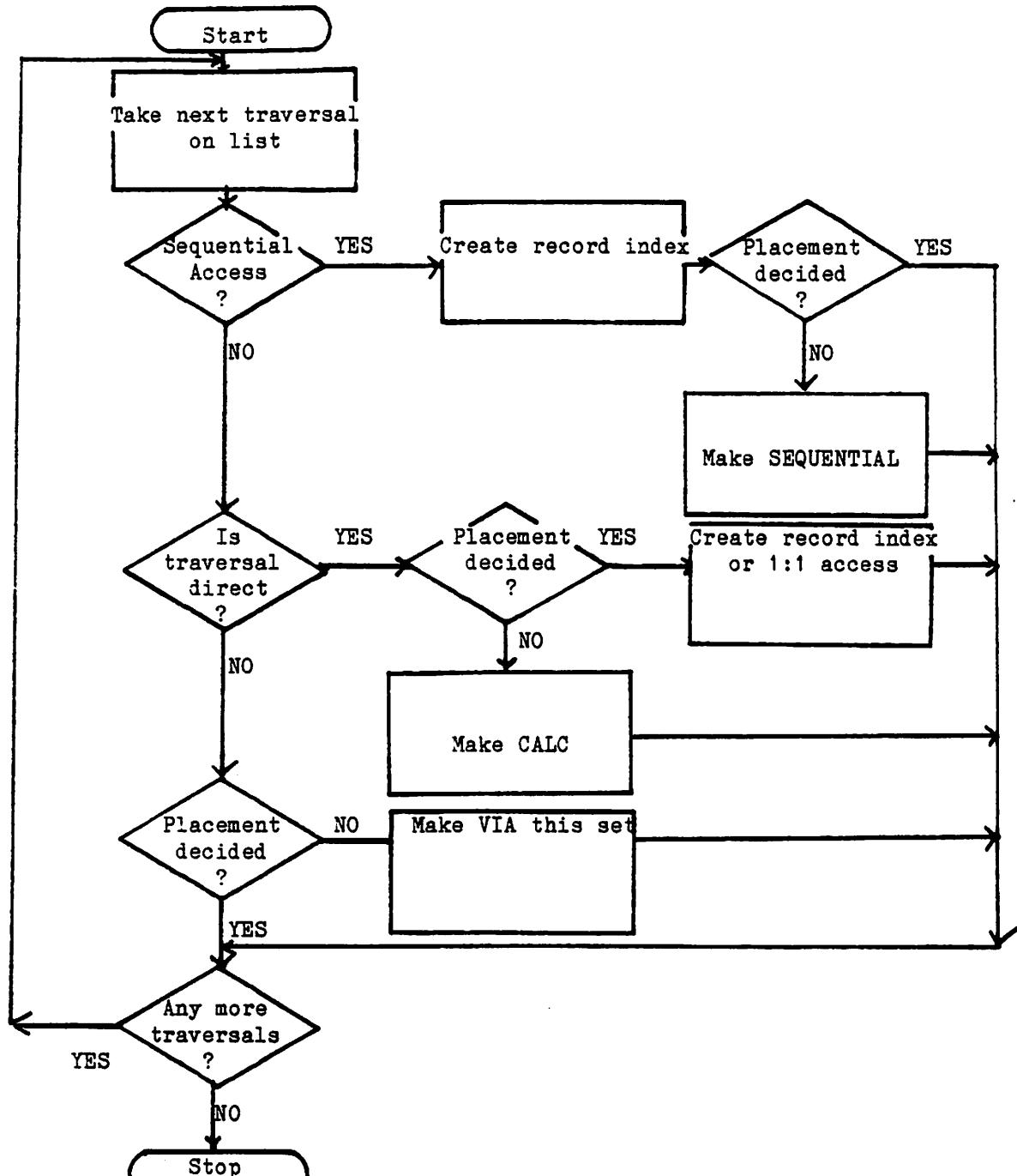


Figure 6 Logic for deciding Placement - IDMS X

The partially completed Storage Diagram in Figure 7 results from applying this logic to the activity lists in our example:

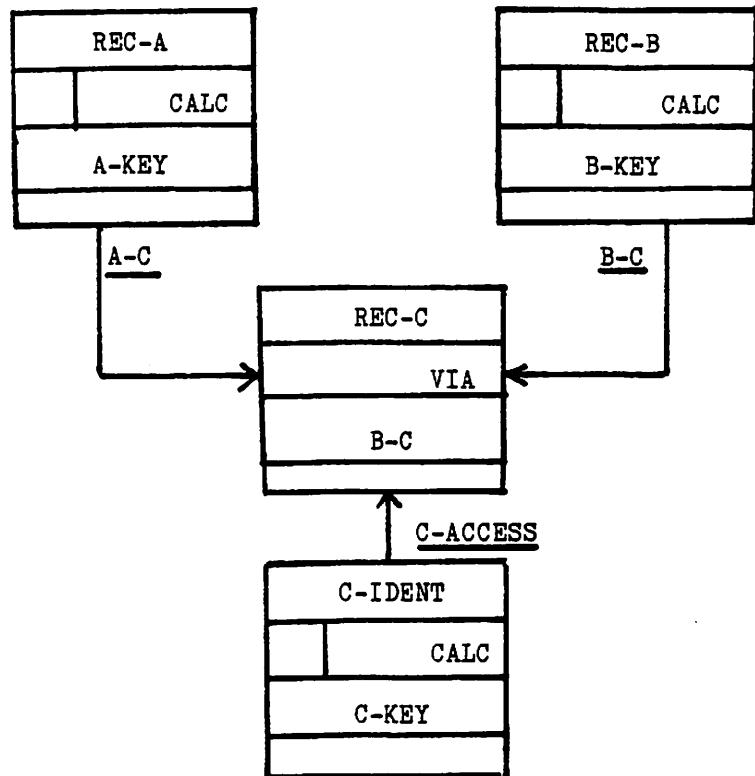


Figure 7 Placement and Entry Point decisions in the example

Further Considerations

This mechanistic approach is a useful starting point. It checks the structure and produces tentative Placement and Entry Point mechanism decisions that will work moderately well. But it tends to oversimplify. In practice you need to take account of other factors such as:

- the number of record occurrences for a given type. If the flowchart suggests creating a 1:1 CALC access record, this may increase the physical size of the database beyond available capacity, especially for the smaller user. In any case, this approach will increase the number of Direct accesses to the required record CALC

- the extent of clustering records. In some cases, this could well extend over a large number of pages, where many record types and occurrences are involved
- sequentially accessing records in an IDMS X environment. If this traversal has a high priority, then the creation of a record index would be beneficial together with allocating a SEQUENTIAL placement to the record type in question
- whether or not a traversal is an updaters (if not already accounted for when placing activity list in priority sequence). Obviously an updaters potentially causes twice as many transfers as an enquiry which just reads. Also an update run will probably write to the journal, further increasing the number of transfers
- is CALC best for direct accesses to a record? Placement DIRECT or PAGE-DIRECT may be better.

The choice of placement and its effect, combined with those of the other design options, on the data structure will be checked at a later design stage - assessing performance. At that stage you may choose to change some placements in order to improve performance.

Other decisions

Grouping Records in Areas

This is probably the most difficult decision to make because so many factors have to be taken into account. To restate the basic rules:

- with IDMS, all occurrences of a particular record type are bound to a specific area
- with IDMSX, a record type can be held in several areas unless it has a location mode of SEQUENTIAL. SEQUENTIAL records are limited to a one area existence.

There are several points to consider when deciding on the number of areas and how to group record types by area. An area is:

- a physical sub-division of the database. It enables you to group records in such a way that unwanted data can be left off-line when exchangeable media are in use. In general:
 - group obviously related records in the same area eg by application
 - start off with members in same area as owner

- a unit of locking, which can be structured to minimise contention. For instance, group records by critical transaction types when you have analysed which transactions lock out others
- a unit of recovery in terms of IDMSDUMP. You can segregate slow and fast moving data into different areas for dump cycles.

~~(X)~~ In general, more and smaller areas mean higher throughput due to:

- less contention
- smaller units for loading, restructuring and recovery.

~~(X)~~
PUT INDEXES IN SETS OF 1000
Sets

Implement sets with chains at this stage with all three Pointers unless some will obviously not be used, eg Owner and Prior Pointers in a 1:1 Access Set. If IDMS X is being used consider using a set Index mechanism instead for large set occurrences requiring selection and/or sorted sequence.

Page Size

Page size will be fixed across the whole user Database (standard IDMS) or can vary from file to file (IDMS X). A size of 4K to 6K Bytes is reasonable for a first attempt, but consider varying this if some critical cluster size (ie Owner Record plus its members) suggests an alternative. Also consider any limiting factors imposed by the device, eg Track size. Page size is considered further below under "Assessing Performance".

As a general "rule of thumb" it is sensible to aim for clusters of about 6 calc records per page.

Summary

By this stage in the proceedings you have:

1. Mapped the Entities and Relationships of the Data Model down to the Records and Sets of the simple Schema Diagram.
2. Checked that the data structure can support the necessary operations and applied the operations in the form of traversals to the structure in order to decide on the Placement of each record.
3. Allocated further design options for each record and set type.

This will have produced the Initial Storage Structure. However, that structure is by no means the end of the story. The next stage is to assess the performance and efficiency of the design.

SECOND STAGE - ASSESSING PERFORMANCE

Introduction

Once the initial design has been produced, you can estimate the number of direct access transfers per success unit as well as the physical size of the database. This information indicates where and how to optimise the design so that it provides efficient navigation paths.

The guidelines given here are sufficient for assessing performance in an IDMS environment. They also provide useful input to a full scale sizing exercise. The main areas of interest are:

- media requirements for standard IDMS and for IDMS X
- estimating database transfers
- other considerations like Program size and estimating Path Lengths.

Standard forms to aid the sizing exercise are included in the appendix.

Media requirements for standard IDMS

Introduction

With the standard IDMS product, you estimate the physical size of each area of the database in turn. IDMSX is slightly more complicated and is covered separately.

The essential steps in estimating media requirements are:

- estimating the occurrences and length of each record type
- choosing a page size
- deciding on packing density for the area.

Record Lengths

Each occurrence of a non-fragmentable record uses space in the database for:

- 2 CALC chain pointers (CALC and PAGE-DIRECT only)
- set tenancy pointers
- length of user data portion of record (minimum 4 bytes)
- line index entry (8 bytes).

Fragmentable records are records that can be split into two or more parts for storage, if necessary. Thus a fragmentable record is not necessarily fragmented.

To allow for possible future fragmentation, an unfragmented record has the format:

- 2 CALC chain pointers (CALC/PAGE-DIRECT)
- set tenancy pointers
- fragment chain pointer (4 bytes)
- data length field (4 bytes)
- data, beginning with any key fields.

A fragmented record consists of a root and one or more fragments. The root has this format:

- 2 CALC chain pointers
- set tenancy pointers mandatory root entries
- fragment chain pointer
- data length field
- data keys, if any, (CALC or SORT key fields should be in root)
- other data fields (optional).

Each fragment has the format:

- fragment chain pointer (4 bytes)
- data fields.

In addition, the root and fragments each have a line index entry (8 bytes).

The number of occurrences of a record type can be estimated from information collected during the data analysis phase. It is advisable to be on the generous side and over-estimate.

Page Size

Page size constraints are:

Database:

- 64 bytes to 32 Kbytes page size (but not greater than maximum block size for the DA device)
- up to 255 records per page
- up to 8,388,607 pages.

Directory:

- 100 to 1000 pages of user-chosen size between 272 and 13028 bytes.

Factors to consider:

Large pages

- minimise overflow for both CALC and VIA records
- can cause wasted space if the records are very small and the record limit is hit
- reduce the number of page buffers that is practical for program size. However, if clustering is good few pages are needed anyway.

Small pages

- allow more page buffers
- with a large volume of data, can hit number of pages limit
- cause more overflow, which entails frequent use of space management pages.

In general a minimum ratio of between 5 and 10 to 1 should be chosen for page size to record size.

As far as possible, page size should be matched to usage pattern.

For example, suppose the hub of the system as far as processing is concerned is a customer record with its related orders. On average a customer owns 10 orders. So the page size can be based on several occurrences of a customer record and clustered order records. The clustering may be displaced if other record types are allowed on the page. One way of preventing this is to place customer and order records in an area of their own.

A page size chosen by this method may be very practical for single application databases. Generally, though, where several applications exist, the page size is suitable for one but not so suitable for the others. In such cases, a compromise page size of 4 Kbytes is often used.

Packing Density for the area

A commonly chosen running packing density is 60%. Before automatically accepting 60% as correct, consider the nature of the data:

- **volatile data** Allow a generous margin for growth, perhaps choosing a packing density of 50%. Re-organisation of an area is a major exercise
- **static data** A packing density of 60% or less may seem to be extravagant in disc space. But beware! A higher packing density may not cater for unexpected growth and may result in performance degradation.

The packing density is included in area size calculations and is used as an expansion and spreading factor when estimating media size.

Generally it is unwise to plan to use a database which is running at more than 70% full.

Calculating Media Size

Media requirements are generally calculated area by area.

The total number of records to be held in each area and the length of each record must be known.

- L = total length of record (including pointers)
- O = number of occurrences of the record
- LO = total space occupied by all occurrences of the record.

So, if an area contains n record types;

$$L_1O_1 + L_2O_2 + \dots L_nO_n = \text{space occupied by all records}$$

However, each record on the database also requires an 8 byte line index entry.

So, for O records, the line index entries occupy $8 * O$ bytes.

For n record types;

$$8 * (O_1 + O_2 + \dots O_n) \text{ bytes are used by line index entries}$$

So, $L_1O_1 + L_2O_2 + \dots L_nO_n + 8(O_1 + O_2 + \dots O_n)$ bytes are required to hold all of the records

P = size of the database page

P-40 = space available for holding records and line index
(allowing for the page header and trailer)

N = number of pages to hold all records

$$\text{So, } N = \frac{L_1O_1 + L_2O_2 + \dots + L_nO_n + 8(O_1 + O_2 + \dots + O_n)}{P-40}$$

The above figure assumes that each database page is full.

In reality some space should be allowed for future expansion etc.

D = packing density as a percentage
(ie the proportion of each page that is to be used to store data)

and now the number of pages required is:

$$N = \frac{L_1O_1 + L_2O_2 + \dots + L_nO_n + 8(O_1 + O_2 + \dots + O_n) \times 100}{D}$$

N = number of pages

L = length of each record type (bytes - including pointers)

O = number of occurrences of each record type

P = page size (bytes)

D = packing density (as a percentage)

IDMS will also set aside space management pages.

Consider:

P = page size

N = number of pages holding data records (as calculated above)

ANTICIPATE GROWTH.

There is a space management entry for each data page. Each space management entry occupies 2 bytes.

$2N$ bytes = space occupied by space management entries

$P-40$ bytes = space available on each page for holding space management entries

$S = \frac{2N}{P-40}$ = number of space management pages required

The total number of pages required for an area is now:

data pages + space management pages

= $N + S$

The above calculations are repeated for each database area.

A convenient method of using this formula is as follows.

Consider an area which is to hold 4 record types.

The calculation is:

Record type	No. of occurrences (O)	Record size (L)	Space occupied $L*O$
R1-REC	1	10	10
R2-REC	6	100	600
R3-REC	500	20	10000
R4-REC	1000	50	50000
LINE INDEX	1507 $(O_1 + O_2 \dots)$	8	12056
Space required for records + line index			72666

If page size = 2048 bytes

Then $2048-40 = 2008$ bytes available for storing records

So $\frac{72666}{2008}$ pages are needed.

For a packing density of 70%

$$\frac{72666}{2008} \times \frac{100}{70} \text{ pages are needed}$$

So 52 pages are required.

The number of space management pages needed is

$$\frac{2 \times 52}{2048-40} \text{ (rounded up)}$$

So 1 space management page is needed - giving a total area size of 53 pages.

Media requirements for IDMS X

Introduction

Although the calculation of Backing Store requirements for an IDMS X Database are fundamentally the same as for Standard IDMS only slightly bedevilled by "Record type in more than one Area", there are two further complications:

- Variable Page Size
- Indexes.

Variable Page Size

IDMS X allows different page sizes in different Areas and even different page sizes per file. So an Area which maps to more than one file can have varying page sizes within it. In this case, the Area size calculations must consider each file separately.

The number of space management pages should be calculated using the SPACE MANAGEMENT RECORD SIZE clause specified in the AREA entry of the DSDL.

Indexes

Introduction

IDMS X Indexes have an entry for each Record covered by the Index, new entries being added as new Records are added to the Database and entries being removed as old Records are deleted. This means that Indexes may be large in size and may be complex for IDMS to use. It is important therefore (and sometimes difficult) to allow enough room for each Index and to ensure that IDMS can handle it as efficiently as possible.

The following points have to be considered.

- how big is the Index itself?
- how big should the Index Record(s) for this Index be?
- how should these Index Record(s) be placed?

An estimate of the total Media demand for that Index can then be made.

Index Size

An Index consists of Index entries. An Index entry for a Sorted Set Index with one Member Record type is shown in Figure 8.

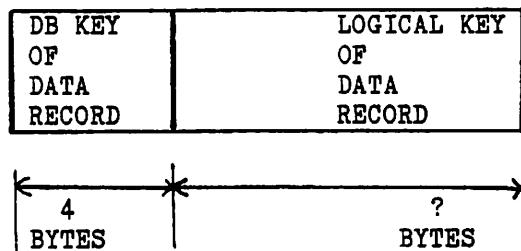


Figure 8 Format of an Index Entry

(Set Index Entries for non-sorted sets do not contain the Logical Key and whether sorted or not, Index entries for sets with more than one Member Record type include an extra 2 bytes for the Member Record-Id.)

A minimum size for an Index is therefore:

$$N(\text{Index Entry Length}) = A$$

where N is the maximum number of Records the Index is to cover.

In the case of a Record Index there will only be one Index occurrence. In the case of a Set Index there will be an occurrence for each Owner Record occurrence.

Index Record Size

The entries within the Index are one or more variable length INDEX RECORDS and a maximum Index Record size should be specified in DSDL.

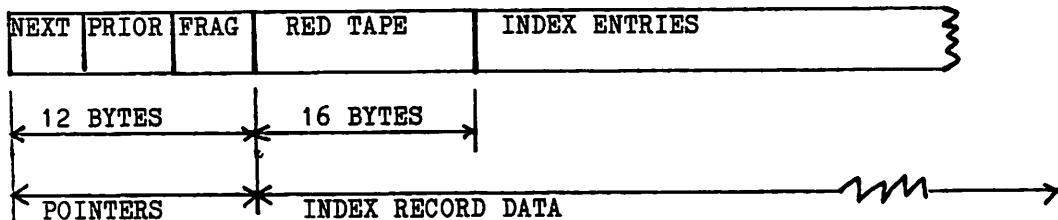
An Index Record although variable length, may not fragment so that as it expands (as new Records are added to the Database) one of two things may happen eventually:

- the quoted maximum size is exceeded. In this case the Index Record's contents are split across two new Index Records ie the Index gains an extra level in the hierarchy at that point .
- the expanded Index Record, although not yet at its maximum size, will not fit back into its Page. In this case it will be re-located.

Both these processes are expensive and should be avoided if possible. so care should be taken in specifying the Index Record size in DSDL and the Page range and Page size within which Index Records for an Index are to be placed.

The ideal would be for the Index to be held in one Index Record within one page. Notice however that there is some red tape associated with an Index Record so that it is not simply a question of whether a Page size of "A", as calculated above, is acceptable. To be precise:

The format of an index record is:



If A = the size of the index entries (bytes)

12 + 16 + A = 28 + A bytes are occupied by the index record itself

However, an 8 byte line index entry is required on the page which holds the index record giving:

$$28 + 8 + A = 36 + A \text{ bytes of space required to hold the index record on a page.}$$

Since 40 bytes of each page are used by the page header and trailer;

$$36 + 40 + A = 76 + A \text{ is the minimum page size which will hold } \underline{\text{the index record}}$$

As an example, suppose an index is to cover 100 records with a logical key that is 6 bytes long

$$\text{Each index entry} = 4 + 6 = 10 \text{ bytes}$$

$$\text{So } A = 10 \times 100 = 1000 \text{ bytes}$$

$$\text{So minimum page size} = 1000 + 76 = 1076 \text{ bytes}$$

Note the actual size of the index record is 1016 bytes (red tape + index entries)

The above calculations assume that no page reserve has been set.

However, with large Indexes there may be no chance of getting "A" into one Page ie an Index of at least two levels is involved. In this case try an Index Record size of:

$$\begin{array}{l} \text{Index} \\ \text{Entry} \quad * \sqrt{N} \\ \text{Length} \end{array} + 16 \text{ bytes}$$

where N is the number of Records to be covered by the Index. Is a Page size big enough to contain this result (plus Page Reserve + 60) acceptable?

As an example take the same case as before but assume 90,000 Records to be covered. Then an Index Record size of 3016 bytes is suggested ($10 * \sqrt{90000} + 16$) and a Page size of at least 3076 bytes.

The maximum ideally structured index would consist in this case of 300 Pages, each containing one Index Record containing 300 Index Entries each pointing at a data Record. Above this would be a single higher level Index Record containing 300 Index entries each pointing at an appropriate lower level Index Record.

In the case of really large Indexes, it may be that the Page size just calculated is still unacceptable or impossible. In this case at least 3 levels will be needed and an Index Record size should be tried of:

The Cube Root of N + 16 bytes

Index Record Placement

An Index can be put into a different Area from the one(s) containing the Records it covers. Alternatively it may be placed in the same Area.

Record Indexes are placed randomly in their Page Range. A Set Index can be placed randomly or via the Owner Record.

Placing an Index into the same Page(s) as the Records it covers requires a careful examination of the impact of this decision on the clustering patterns required for data Records. In some cases such a decision could reduce physical accesses (eg a Set with small occurrences could usefully have the Set Index VIA) but in many others it could increase them ie Large Record Indexes and Set Indexes for Set Occurrences with many Member Records.

Calculating Index Media Requirements

If an Index is put in the same Page Range as its data Records, then the Media requirements are best handled by regarding the Index Records as just another Record type in the Area (admittedly a difficult one being variable length and often volatile).

The rest of this section assumes a different decision with the Record Index (or all occurrences of a Set Index) held in an Area of its own. It is further assumed that the Index is tidy ie that there are a minimum number of levels. This can be achieved automatically (given a sensible Index Record size decision) when the Database Load utility is used. Alternatively if a User written load program is used the Index Tidy Utility can be run after the load to achieve the same result.

Note. A user written load program which loads data in key order is actually a good way of producing an untidy index. Because of the way Index Records are split and deeper hierarchies produced, an Index after a load of Sorted Records will be heavily biased (ie too many levels) on the right hand side of the Index tree. Always use the Index Tidy utility after such a load to report on the situation and then, where necessary, to correct it.

The following method can be used to calculate the total byte requirements for an index:

M = index record size (= red tape + index entries)

M-16 bytes of the index record are available for holding index entries

A = the sum total size of all index entries in all index records

$\frac{A}{M-16}$ = the number of index records needed to hold all of the index entries.

However, each index record requires a line index entry (8 bytes) and has 3 pointers ($3 \times 4 = 12$ bytes)

$M + 8 + 12 = M + 20$ bytes are "used up" by every index record.

So, $\frac{A}{M-16} * (M + 20)$ bytes are needed for all the index records

This assumes that each index record is full. Normally some free space is allowed for later expansion.

Let P = packing density as a decimal figure
(eg 70% is expressed as .7)

Then

$\frac{A (M + 20)}{P (M - 16)}$ bytes are needed

A = total size of all index entries

M = size of index record

P = index packing density

For example, given the earlier case of 90,000 Records with a 6 byte key, and assuming 30% space to be left free for later expansion:

$$\begin{array}{r} \underline{900000 * 3036} \\ .7 * 3000 \\ = 1,301,143 \text{ bytes} \end{array}$$

The byte requirement can then be divided by the (Page Size - 40) to get the number of Pages in the Area. However as with normal data it is unwise to plan to pack the Area 100%, so a packing density should be specified. One of 60% or so should sharply reduce the risk of two Index Records randomising to the same Page. In the example above, assuming a Page Size of 3076 bytes the calculation is:

$$\begin{array}{r} \underline{1301143} = 714 \text{ Pages} \\ .6 * 3036 \end{array}$$

Note. No allowance has been made for higher level Index Records or for Space Management Pages in this calculation. Their impact will be insignificant.

Estimating Database transfers

Introduction

In designing most database applications the object is to minimise the total processing time. This is usually more dependent on the number of disc transfers than on the number of machine instructions obeyed. Accordingly the disc transfers are discussed in some detail here and the path lengths are mentioned briefly later.

Before you can assess the number of disc transfers for a success unit you need an appreciation of the way Buffers are used and of the way the DML Verbs work. It is also necessary to allow for writes to Disc Journals.

Buffer Control

Each Success Unit has its own Page Buffers (usually between 3 and 7). When IDMS receives a request for a page, it first consults the buffers to see whether the page is already there. If not, the page has to be read in. If all Page Buffers are in use one has to be emptied to make room for the new page. The least recently accessed page is normally the one to be discarded. With the delayed update option, the least recently accessed read-only page is discarded.

Obviously you need enough page buffers to prevent the shunting in and out of any page accessed repeatedly in a major program loop. Also when the database is getting full, at least one page buffer is likely to be in constant use holding space management pages.

DML Verbs

Introduction

The number of disc transfers required by a given DML Verb depends on:

- the physical layouts of the data it needs to access
- what the DML Verb is trying to achieve and the logic that it uses in the process.

In trying to estimate the number of disc transfers at this stage, we are not aiming at extreme precision. Rather we are looking for design errors which at best may produce clumsy processing on critical transactions and at worst may have built Black Holes sucking in hundreds of thousands of disc accesses.

Physical layouts of data

Selection of a Record through CALC or PAGE DIRECT placement usually requires no more than one disc access. Selection through a Record Index may need no more than 4 or 5 disc accesses even on a large Index. Physical layouts become critical only where set walking is involved. The essential point here is whether the set occurrence is:

- **clustered.** Owner and its Member Records on the same page or nearby pages
- **scattered.** Owner and Member Records each one on a different page

The way DML Verbs work

1. Retrieval Verbs

FIND/OBTAIN usually requires one disc access if the page in question is not already in a Buffer. However, the following cases may require a number of disc accesses:

FIND ANY USING KEYNAME If CALC or PAGE DIRECT may be more than one access if Records have overflowed. If a Record Index may be 4 or 5 accesses

FIND within REALM	Search is in ascending or descending database key order, examining each page in the area
FIND OWNER	IDMS walks set if there are no owner pointers
FIND nth record within set	Search always starts from owner and traverses set. This is costly for scattered sets
FIND within a chained sorted set	Depending on syntax used, will begin the search at the owner or at current of set. It involves a sequence of FINDs and key comparison operations. For example in Figure 9, it would need 4 disc accesses to find the Member Record with a key of 10, if we start at the Owner.

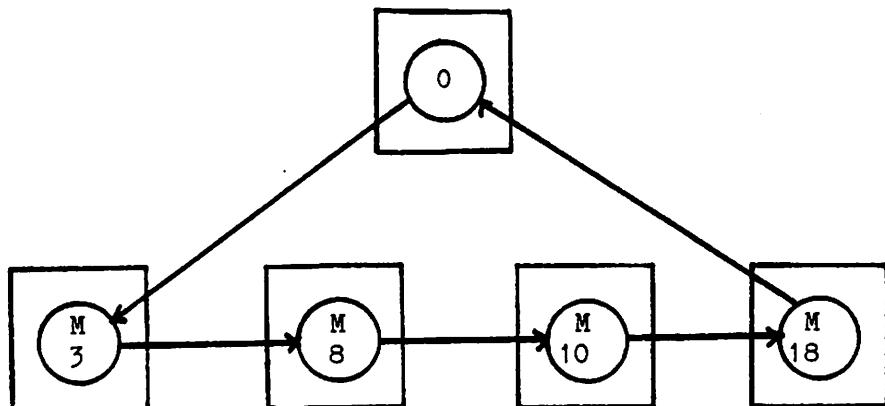


Figure 9 A Chained Sorted Set occurrence

2. Update Verbs

These DML Verbs require access to a page, then change it, then require it to be written back. The number of disc accesses involved may be more or less than two however. It will be more than two if other pages are involved perhaps only to allow access to the required page but maybe also for the alteration of set pointers. Consider the example in Figure 10, where a new Order is to be added to a Chained Customer-Order Set where the set is scattered, with Prior Pointers and ordered first. (The Pages have been given simple numbers for reference.)

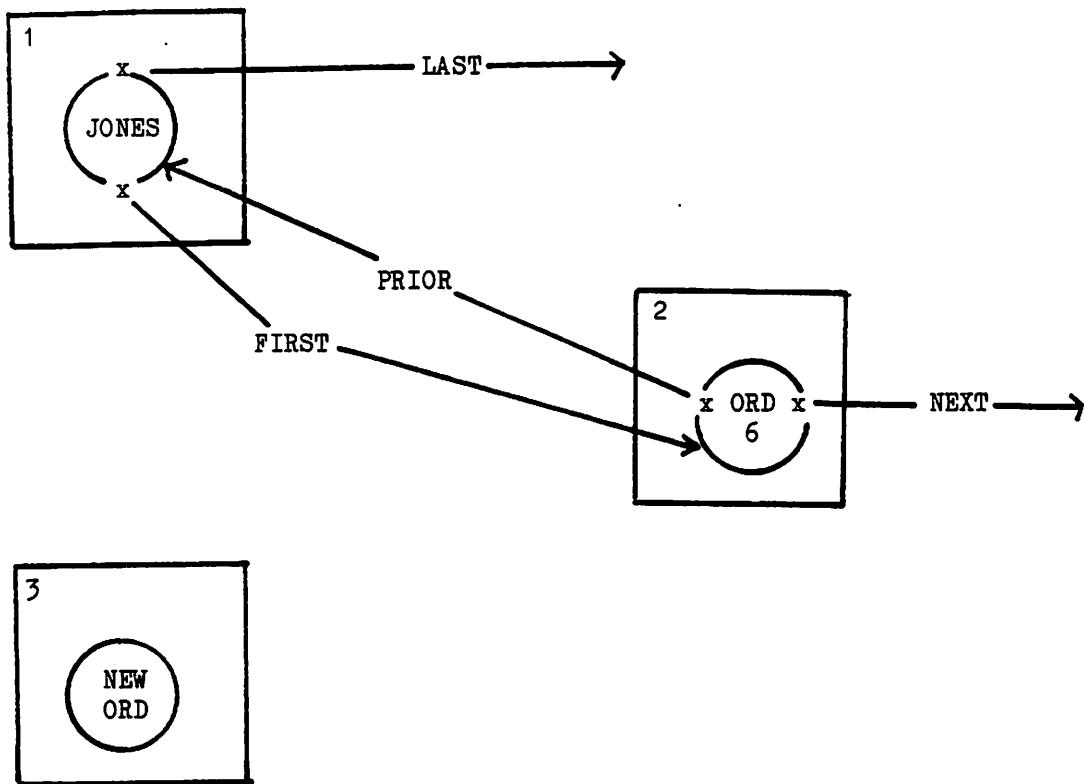


Figure 10 Connecting a new Record to a Scattered Set - Before

The following accesses will be involved in the connection:

1. Read Page 3 to get the new Order. This access is probably counted as part of an earlier DML Verb.
2. Read Page 1 to get the Owner. The FIRST pointer is changed to point to Page 3 and its previous value (Page 2) must be placed in the New Order as its NEXT Pointer. The New Order PRIOR must be set to point to the Owner in Page 1.
3. Read Page 2 to alter the PRIOR Pointer in Order 6 to point to Page 3.
4. Write back the altered Page 2.
5. Write back the altered Page 3.
6. Write back the altered Page 1.

As a result of these 5 or 6 disc accesses, the set now looks like Figure 11.

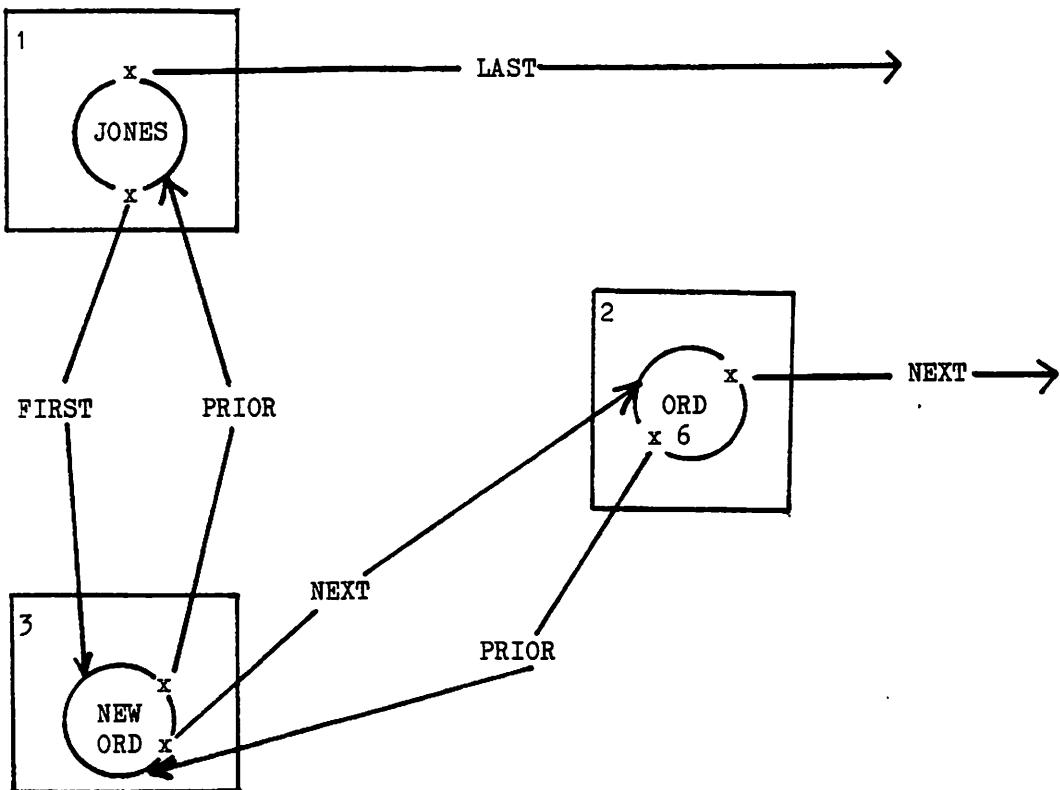


Figure 11 Connecting a new Record to a Scattered Set - After

This example contains a number of lessons as far as performance on chained sets is concerned:

1. Scattered/Clustered Sets. It is the Scattered Sets that require a disc Access per Record occurrence. The example above will have been no more than 2 disc accesses if the set had been clustered. (For the pedantic reader, it is odd but not illegal to make a member Record VIA a Set of which it is not an automatic member.)
2. Set Order. This case could have been worse. Consider a Set Order of sorted where the new Record fits in half way along on average. Or consider the classic error of sorted ASCENDING when Records are added in rising sequence of Sort Key! (The solution is to consider sorted DESCENDING or FIRST for such cases).

3. Set Occurrence size. For Set Orders which may involve a lot of set walking to find the right insertion point, the larger the scattered set, the worse things are (and the stronger the argument for an Indexed Set in IDMS X).
4. Prior Pointers. These increase the accesses slightly where set pointers have to be maintained in a scattered set.

Note. All the above was concerned with Chained Sets. Indexed sets are less efficient than Chained, if there are only a few member Records (say, 6 or so). However, in large scattered sets, a chained implementation could have to handle hundreds of disc blocks to perform a DML verb. In an Indexed set, the individual Member Records do not have to be accessed and the required work can be done by handling just a few Index Records. The IDMS X designer has to get a picture for each set type of the typical layout on the pages and of the typical patterns of DML processing for critical transactions. An objective decision can then be made as to whether the extra complexity of a Set Index is worthwhile. It will be, sometimes.

If the CONNECT case is understood, the accesses for the updating verbs can be estimated using CONNECT as a foundation:

STORE - one read/one write to put the new Record in its home page plus a CONNECT for each Scattered Set of which the Record is an Automatic Member.

DISCONNECT - the reverse of CONNECT. However, if there are no Prior Pointers on a chained set, IDMS may have to walk around the set to find out (eventually!) what the Prior Record was. It can then point that Record forward to the next Record past the disconnected one.

ERASE - Simply STORE in reverse. But no Prior Pointers can produce logically deleted Records. Could need to change Space Management Pages. Watch the Multi-level ERASEs. Calculation of disc accesses has to be done for each Record erased.

MODIFY - One read/one write only unless changing a Key field:

- CALC/PAGE DIRECT means a move from the original home Page's CALC chain to the new home Page's CALC chain
- Sort keys in sets. The Record's position in the set is altered so there could be several accesses to alter pointers if the set is chained and scattered.

Note. The way IDMS handles connection into a set can be varied by the Schema's SET SELECTION CLAUSE. Usually IDMS uses the current of Owner Record type currency Indicator to decide WHICH set occurrence to connect to if the Set Order is FIRST, LAST, SORTED or SYSTEM DEFAULT. This is efficient but can be a little awkward for the programmer. The set selection clause allows IDMS to be switched to use the Current of Set type currency indicators on that set type for those Set Orders. It uses the latter anyway for Set Orders of NEXT or PRIOR. Switching in this way will make programming a little easier but could involve extra disc accesses to reach the Owner if there were no Owner Pointers.

The Accesses for a success unit

Once the Accesses for each DML Verb can be established, it is possible to get a figure for the whole of a Success Unit, working on typical data structures. This involves laying out a DML skeleton for the Success Unit and accumulating the accesses for each verb in the skeleton as the typical data structure is processed. Since each Success Unit has exclusive use of its Buffers, the final answer will usually be less than a straight forward total. This is because one DML verb can use Buffers left by the previous verb and can pass the Buffers on to the next DML Verb in the Success Unit.

The results for each Success Unit and/or each batch program's main processing loop can then:

- give a useful guide to the number of page buffers required
- highlight any parts of the structure which need to be changed to improve performance, in particular for TP Success Units. It is unlikely that a Success Unit requiring over 50 Disc Accesses will give a satisfactory TP response time and a much smaller number (5-15) would be advisable.

Journalising

Journalising is optional, but it is the default for updating applications. You should consider the effect of journalising when estimating the number of disc transfers during the run of a success unit (unless the unusual decision has been made to have a Tape Journal).

Record level journalising is used in IDMS. Before and after images (lines) of each updated record are stored in the journal buffer and that block (one transfer) is written to the journal:

- when the buffer is full
- when a page is written to the database
- on a FINISH.

The number of blocks written, and therefore transfers to the journal, will depend on:

- choice of journals (X only)
- the journal block size(s)
- the size of the success unit
- the number of database page buffers.

In an IDMS/TP environment, extra transfers to the journal may be needed. For a detailed description of journalising see Chapter 8.

Other considerations

Path lengths

Transaction times depend predominantly on disc usage, so it is usually sufficient to estimate just the disc transfers. A more detailed sizing exercise can be carried out to estimate a transactions mill or OCP usage. This becomes extremely complex as there are many inter-related factors which influence the path lengths.

Detailed figures are not given here as the timings of machines differ widely across the range.

Program Size - IDMS Considerations

Program components

To estimate the size of application programs, think of them as conventional COBOL programs (which can be sized in the normal way) plus a number of IDMS components. The structure of the IDMS program will also affect the program size.

IDMS considerations are:

Program expansion including all data descriptions and procedural code brought into the program by the DML pre-processor

Subschema tables holding a description of the logical structure and currency information. Size varies according to the number of records, sets and areas included in the subschema

Service tables contain area to file mapping information plus buffer area. Again size varies according to the number of files and areas associated with the tables

IDMS run-time code the IDMS software. Sizes are:

50 Kbytes full code IDMS
25 Kbytes present at any one time due to overlaying

Buffer size minimum number of buffers for pages is 3. More than 12 buffers results in a large program with a marginal increase in efficiency. Too small a number of buffers can cause page thrashing ie pages frequently being written back to the database and then read in again. For details of buffer control see the section on estimating database transfers.

The size of an IDMS program also depends on whether or not it exists in a shared environment.

Unshared Mode

When IDMS is unshared, the program includes all the components listed above. Buffer size is normally 5-8 pages, slightly more for an initial load program.

Shared Mode

Each AVM contains the user code, subschema table and local SDL table including the buffer area. A normal buffer size is about 7 pages and this can be changed on loading the AVM.

The IDMS code and global lock table, which has a default size of 3K bytes, are both shared by all the AVMs using the IDMS service.

Note. Detailed information about the run-time tables and program expansion is given in the appropriate IDMS manuals.

Summary

Before you can measure the efficiency of the database structure, you must:

1. Choose a page size.
2. Calculate media requirements.
3. Determine the DML sequence per success unit and then:
 - estimate the number of disc transfers
 - optionally examine path lengths for more detailed sizing.
4. Compare the results of performance estimation with the requirements of the system.

Unless you are satisfied with the results, this leads to the next stage in the overall design process - tuning the structure to improve performance.

THIRD STAGE - IMPROVING PERFORMANCE AND TUNING

Introduction

An IDMS database can only be optimised for specific purposes; you cannot design a database that will be best for all applications. This means that priorities must be established in as objective a way as possible.

The Data Analysis phase will have established a list of applications in priority sequence, as well as facts about transaction rates and expected response times.

The Improving Performance phase is concerned with making detailed trade-offs between various aspects of the design. This is an iterative process - examine the initial design, check it objectively using the sizing procedures described earlier, and improve it until the design criteria are met.

This process involves making decisions regarding:

- sets
- records
- areas
- indexes
- loading.

Many of the changes will be to the initial storage schema decisions, some however will affect the schema.

Sets

Set mapping

Because the set concept allows the database to model the real world, there is a possibility that the initial structure may contain too many sets. The Data Analysis phase should check that only necessary relationships exist in the data model. Similarly, the exercise of examining traversals checks that sets do in fact support the required operations.

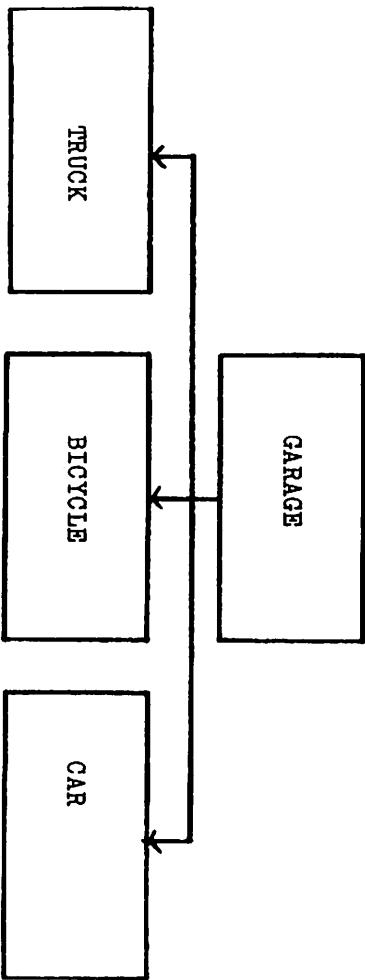
When examining the set structure, bear in mind the following points:

- Unnecessary sets involve extra space on the backing store for pointers
- Unnecessary sets also lead to extra processing time

- The more sets, the larger the subschema tables
- In general, simple structures lead to smaller faster programs
- Consider using soft pointers instead of a set type especially where only one-way access is needed eg repeat the logical key of the DEPARTMENT in each PERSON it employs. Do not use Database keys as soft pointers. They cause reorganisation problems

- In some cases, a multi-member set will be more efficient than separate sets. For example:

An enquiry to examine vehicles that have been sold through a garage irrespective of type.



Compared with three separate sets linking garage to each vehicle in turn, the multi-member set approach:

- reduces the number of pointers on GARAGE
- reduces the number of sets processed
- gives a more flexible structure; it's easy to create a new vehicle type, eg Hovercraft, since no restructure or re-organise is required.

Obviously you must consider the processing requirements before taking a decision like this.

In some cases, extra sets can be advantageous:

- an extra set can speed up an enquiry
- dummy sets are useful for later expansion.

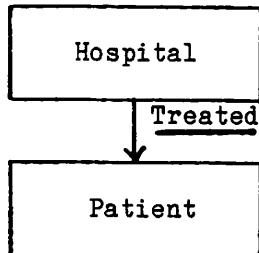
~~X~~

~~reduced space
join sets
necessary
to do~~

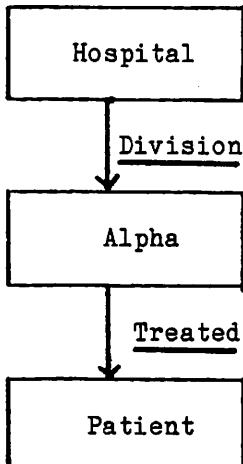
To reduce page transfers:

- keep sets small and compact, if possible
- keep scattered sets very small if frequently walking the set
- consider splitting large sets into several small sets by introducing another record type. For example:

Suppose this set is sorted and there is a large number of patients per hospital.



Finding/inserting a new patient in sequence could be very long winded if a chained set is used. An extra set and record could speed up processing.



Here there is one ALPHA record per initial alphabetic character of name. *However, in IDMS X a set index implementation for the TREATED set would probably be a better solution. *

Set Design Options

Pointers

(Chained sets only)

OWNER As well as speeding up FIND OWNER, these pointers increase the efficiency of internal processing.

PRIOR ~~*~~ Omission of prior pointers can lead to a build up of logically deleted records.

Inclusion of prior pointers means more processing and possibly more transfers in order to maintain pointers, but will improve the performance of DISCONNECT. ~~V~~

Set order

SORTED (chained only) Beware of the internal processing and maintenance involved when using sorted sets. Does your set really need to be sorted? Will a set order of FIRST or LAST be sufficient? Would employing the general techniques of extract and sort be an acceptable alternative to maintaining a sorted set?

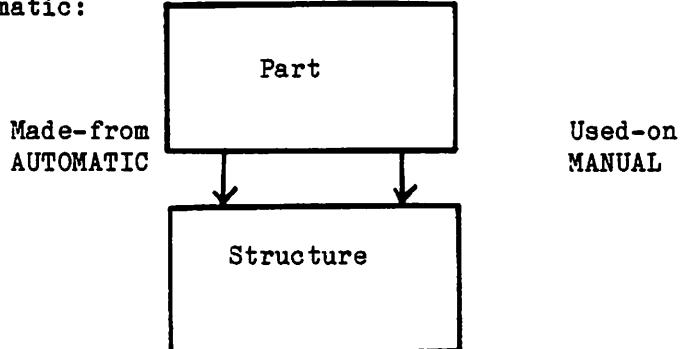
With IDMSX, the use of indexed sets makes large sorted sets a practical proposition.

PRIOR Use when the sequence is to be controlled by program logic.

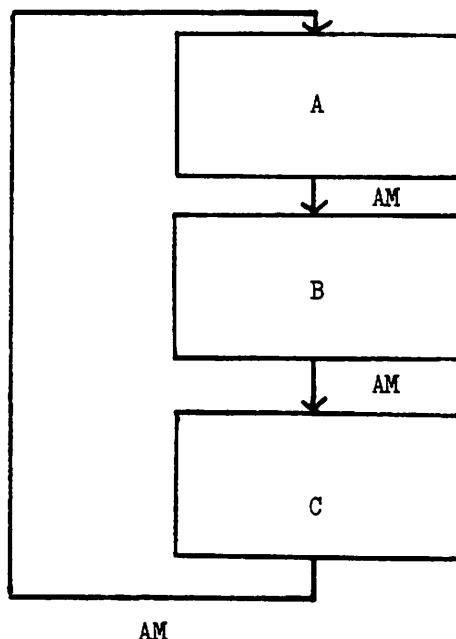
Set membership

AUTOMATIC insertion class ~~*~~ Keep to a minimum for faster TP programs. ~~*~~ An on-line STORE can be delayed by a need to connect the new record to several scattered sets. ~~*~~ Consider making some of the connection later, in a Batch run. ~~*~~

In a bill of materials structure only one set can be automatic:



Beware of the impossible cycle. In a cycle such as this at least one membership must be manual:

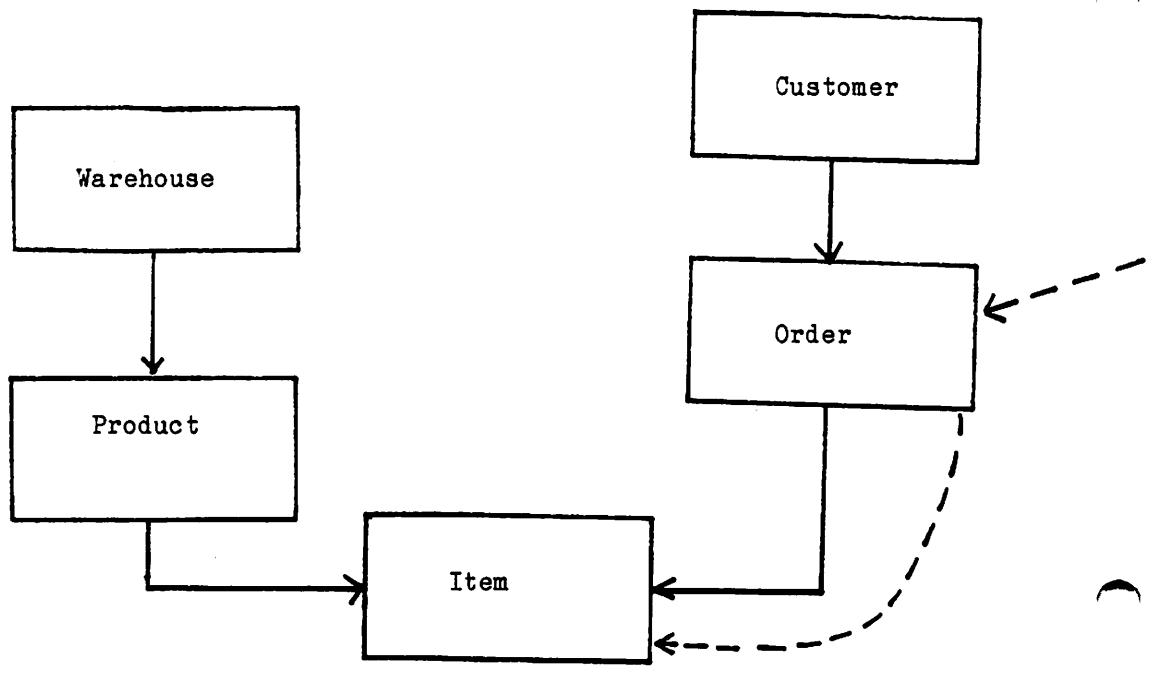


Records

Record mapping

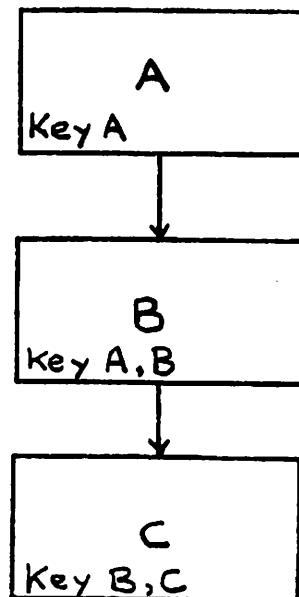
Data Analysis should have established the ideal contents of each record type. However, you may choose to modify the record contents for performance reasons. For instance:

1. Include dummy fields for later expansion.
2. Duplicate data to reduce navigation, as in the example overleaf:



Given the access path shown, including duplicated data in the ITEM record avoids accessing the PRODUCT and WAREHOUSE records. This is justifiable if the saving in disc transfers is important.

3. Repeat owner keys in member records.



~~DON'T USE SET ROR IN PD
ON AS NAVIGATION~~

Information about an owning record is held in the member record, by including the owner keys. This is a useful technique especially when dumping and reloading the database, but is perhaps too expensive an overhead for this purpose alone. Repeating keys confers a more general benefit, however, in that data will be less application biased if sets are used only as access paths, and more useful for CAFS scans. *

Notes:

1. One of the advantages of a database approach is that it avoids duplication of information. Repeating data is beneficial in certain cases, but should be kept to a minimum in order to reduce maintenance.
2. Remember that any tuning of the design to improve one pathway will degrade others. In general, any tuning involves distorting the data model which may make current or future sharing of the Database more difficult.
3. Remember that COBOL programs can access set tenancy pointers and currency indicators and use them for navigation. Balance the improvement in performance such low level techniques allow against the risks of mistakes and of DB keys being stored away and being used for retrieval. *It is always dangerous to hold a database key for longer than the duration of a success unit without special precautions.*

Fragmentable Records

Records can be fragmented across several pages where the following definitions apply:

- Fixed length records with an associated MINIMUM ROOT and/or MINIMUM FRAGMENT clause
- Variable length records that expand on modification
- Variable length records with an associated MINIMUM ROOT and/or MINIMUM FRAGMENT clause.

* Fragmentation should be kept to a minimum because:

- it increases the number of database transfers required to obtain the record
- there may be insufficient page buffers to contain the whole record
- NO QB *

- each fragment uses a line index entry so the greater the fragmentation, the more backing store is used
- CAFS does not like it.

In order to reduce the amount of fragmentation consider the following points:

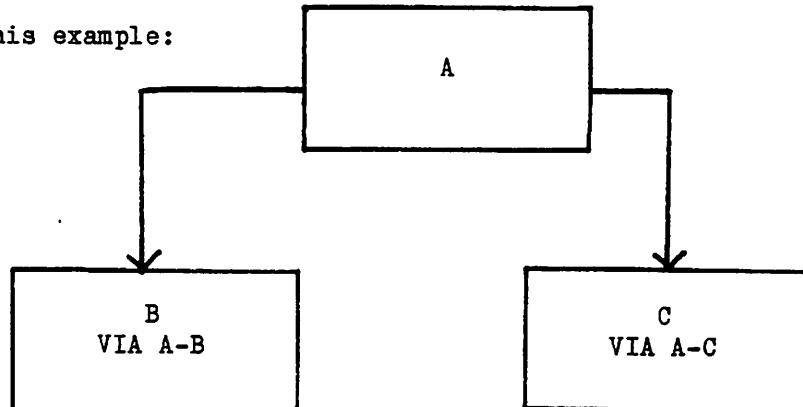
- For variable length records with a frequently changing OCCURS field, specify a MINIMUM FRAGMENT clause. Otherwise a default value of X(4) will be used, which can cause extensive fragmentation
- Set the MINIMUM FRAGMENT to the size of the fields subordinate to the OCCURS clause
- Define the record with a small number of occurrences, each of which is large in size
- If the MINIMUM ROOT or FRAGMENT is greater than 30% of the page size and a packing density of 70% has been used on that area, there may be problems storing the data!

Record Design Options

Placement:

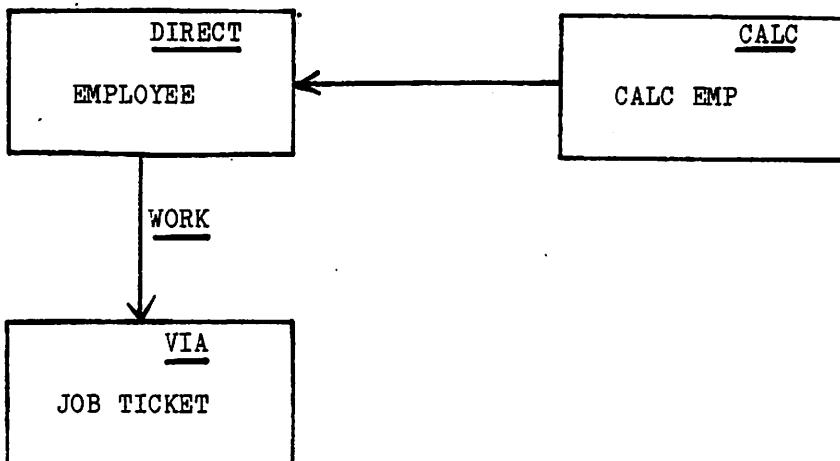
- CALC gives one of the most even spreads of any known algorithm and should only be replaced by PAGE-DIRECT when there is a strong reason for not using CALC eg all records present across a wide range of possible key values.
- VIA records may be located via a different set as a result of the sizing exercise.
- To improve clustering and to reduce accesses to the space management page, consider placing VIA records at a displacement from the owning record, or even in another area.

In this example:



both B and C records will be clustered around A. If the number of occurrences is too great to be held on one page, either place the least frequently accessed member type in a separate area or specify a displacement against the record in the DSDL.

- DIRECT should be used for low volume frequently used records such as control and superchief records, which can be loaded first. SYSTEM DEFAULT could sometimes be an alternative for such cases.
- DIRECT and PAGE-DIRECT can be used for structuring the database to user requirements. In the following example, due to the variability in size of the WORK set, we wish to store an EMPLOYEE record on every other page allowing two pages per EMPLOYEE. If DIRECT is used, an alternate access mechanism is required to gain entry to the EMPLOYEE record (users will not in general be familiar with the database key of a record).



This approach is difficult in volatile situations if, for example, a lot of employee records are going to be inserted after loading.

Areas

If an operation involves accessing all or a high proportion of records of a particular type, consider an Area Scan (or file type processing). If the record type in question is a small proportion of the total area, the record type may be placed in its own area for efficient processing.

Alternatively, restrict the record type to a particular page range within a larger area.

In order to achieve clustering of owner and member records, place them in their own area. Other record types cannot then cause displacement of member records.

As stated earlier an area, being a unit of locking, can be structured so as to minimise contention. The sizing procedures and the Transaction Loading Form and Batch Workload Form may shed new light on record to area mapping and suggest amendments to the design. For instance, it may be worthwhile to place read-only records in their own area.

With IDMS X, you can match storage pattern to usage pattern. Thus occurrences of a record type can be held in different areas. The benefits are:

- unwanted data can be off-lined
- there is less contention
- different page sizes may be used in different places to match the varying size of a typical cluster of records.

Indexes

The Indexing facilities in IDMS X are attractive and powerful but it should now be clear that they may also make considerable demands on the hardware, the software and the designer.

Review the decisions you have made for each Index:

- Do you really need it?
- Is the pathway worth the cost?
- Could the Index be replaced by an extract and print batch run, by a 1 to 1 CALC access record, by a CAFS search?
- Would a chained set be better than an indexed set?

If the Index is justified, review the decisions considered earlier on Index Record size, Placement, Page size, number of Pages and packing density.

Ensure that the Indexes remain tidy by regular reporting and, where appropriate, tidying runs of the Index Tidy utility.

Loading

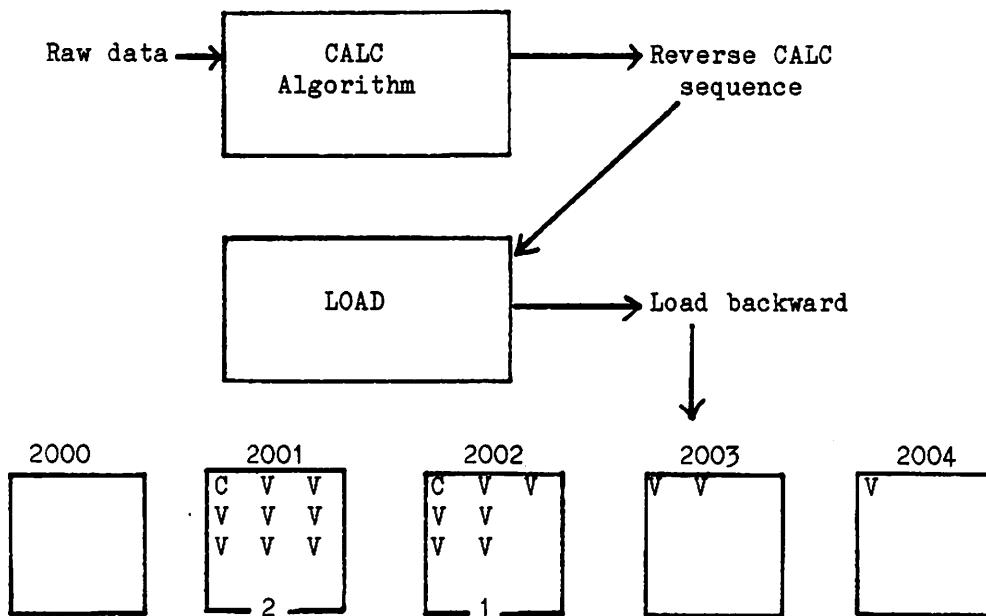
Introduction

A reduction in time taken to load data, especially CALC records and their associated VIAs, can be achieved by:

- reverse loading the records on initial load
- pre-sorting CALC records on subsequent loads.

Reverse Loading of Records

If a number of VIA records are being stored at the same time as each CALC record, it pays to pre-sort the records into reverse order of page number. This ensures that each page receives its own CALC records before being cluttered up with the overflow of the previous page's VIA records. It may, however, be worth presenting the CALC records belonging to a page in ascending order of CALC key for maximum efficiency of finding them again. For example:



Pre-Sorting CALC Records

The number of page accesses and amount of head movement can be reduced by applying the CALC algorithm to the records to be stored. Sort them into page number order; then only a serial pass of the storage area will be required.

Summary

Depending on the priorities of the system, the initial database structure can be modified either to improve efficiency of processing and response times or to reduce the physical size of the database. This is very much a trade-off situation.

The reiterative process of amending the structure, checking and tuning continues until the design is acceptable. The design should then be tried out and monitored on a test database, before being accepted for the live system.

MONITORING PERFORMANCE

Introduction

Try out the database design on a test database as the final stage of checking the design. The test database should be set up to simulate the live database, with corresponding but smaller data areas.

IDMS provides valuable information on the efficiency of the design - particularly the level of I/O activity on the database and the proportion of overflow records.

There are several types of information that may be collected during the running of an IDMS application program:

- IDMS Statistics
- IDMS DUMP Report
- Service Statistics.

IDMS Statistics

The statistics listed on the next page are maintained automatically by IDMS and are available to the program at any time. They also appear in the end of job checkpoint on the journal and may be printed by the journal utilities.

Careful examination of these statistics will prove the effectiveness of the design. The following points are of special interest:

- Compare LINES REQUESTED BY IDMS with PAGES REQUESTED BY IDMS. This shows the effectiveness of clustering
- Compare PAGES REQUESTED BY IDMS with PAGES READ BY IDMS. This measures the efficiency of buffering
- If number of I/Os is high, try increasing the buffer size
- Excessive overflow on record storage in the early days of the database tends to point to a design flaw. It can also suggest an 'area full' situation approaching, so run IDMSDUMP (see later)
- Any page range overflow indicates the population has exceeded estimates. Again run IDMSDUMP
- Excessive fragmentation suggests the need for an increase in page size or a larger minimum fragment.

Output From IDMS STATISTICS

DATE
TIME
PAGES READ FROM DATABASE
PAGES WRITTEN TO DATABASE *SHOULD BE LESS THAN*
PAGES REQUESTED FROM IDMS
LINES REQUESTED BY IDMS
RECORDS FOUND
PAGE RANGE OVERFLOWS - *2nd LEVEL, O'flow*
SERIOUS!
FRAGMENTS STORED
RECORDS RELOCATED
DML STATEMENTS EXECUTED (CALLS TO IDMS)
CALC RECORDS IN TARGET PAGE
CALC RECORDS IN OVERFLOW
PAGE-DIRECT RECORDS IN TARGET PAGE
PAGE-DIRECT IN OVERFLOW
VIA RECORDS IN TARGET PAGE
VIA RECORDS IN OVERFLOW
DIRECT RECORDS IN TARGET PAGE
DIRECT RECORDS IN OVERFLOW

SEQUENTIAL RECORDS IN TARGET PAGE
SEQUENTIAL RECORDS IN OVERFLOW
INDEX RECORDS IN TARGET PAGE Additional in IDMS X
INDEX RECORDS IN OVERFLOW
INDEX RECORDS REQUESTED BY INDEX HANDLER

IDMSDUMP Statistics

The Dump Utility, run in Report mode, summarises the database contents by area. The information includes:

By area:

- the number of characters used
- the number of characters free
- percentage file utilisation.

By record type:

- total length
- number of occurrences
- total space used by this type
- percentage of total space used in area
- the number of logically deleted records.

Service Statistics

Statistics are accumulated during an IDMS service. The statistics are output automatically by the Service VM when the service is closed. For all applications run within the service, the statistics accumulated are:

- IDMS statistics
- Contention statistics, which are collected for several resources including the page lock table and each area.

Other methods

You can set up other monitoring facilities by using the trace parameter of the IDMS (X) RUN command and by using Database Procedures

Monitoring the live Database

The statistics outlined here should also be frequently monitored when the live database has been set up and is running, in order to follow the changing patterns of use. Also they enable you to detect when an area is becoming full so that you can plan some form of reorganisation.

Appendix 1 TECHNICAL DESIGN CHECKLIST

DECISIONS FOR EACH RECORD TYPE

Logical decisions

Content

What Data Items?

Duplication to avoid Owner Access?

Spare space for Future Expansion?

Beware Variable Length - but if using ensure Root Fragment has keys.

Missing Fields situation?

Keys

Is a selection key justified? Will cost you in the scattering effect on the Records.

Is more than one selection key justified? (X) Will cost you more for the Record Index(s). Ordered?

Insertion/Retention class

AM or MM or AO or MO? Keep Automatic to a minimum for fast TP programs.

Physical decisions

Placement

CALC? Which Data Item?

What about Duplicates?

VIA? Which Set?

DIRECT? Situation static?

PAGE-DIRECT? Use own algorithm if special knowledge of situation

SEQUENTIAL? Logical key mapped to DB key sequence (IDMS X only)

If VIA or DIRECT is direct access to the record required? If so use 1:1 Calc Access Record or RECORD INDEXES (IDMS X) which can give multiple access paths to records of any placement.

Which Area?

Record types are confined to one Area in Standard IDMS (Set Relationships are not).

Cluster VIA records in different Area to Set Owner?

IDMS X - Record in several areas except for placement sequential.

DECISIONS FOR EACH SET TYPE

Logical decisions

Is Set Required?

Complex structures (many sets) can give slower update and longer records with little benefit.

Single-Member set type often better than one Multi-Member set type.

Dummy set types on records for future expansion?

Variable length records adequate? Lose data sharing.

Owner Record Type

Member Record Type(s)

Set Order

FIRST? LAST? NEXT? PRIOR? SORTED? If sorted on which field(s)?
Ascending/Descending? Duplicates?

Physical decisions

Chained or Indexed Sets? (IDMS X only)

Keep Sorted Chained Sets small - introduce extra Set and Record Type for faster processing?

IDMS X Large Sorted Sets practical.

Above 4 -6 Member Record occurrences per Set Occurrence use Indexed Set if selection within Set is common.

For each Chained Set type

Pointers If Owner F or FL
 If Member N, NO, NP, NPO

Owner Pointers speed FIND OWNER (except very small sets) and require little maintenance.

Prior Pointers avoid Logically Deleted records. Use if DISCONNECTIONS frequent. Must use if Set Order LAST or PRIOR or using FIND LAST or FIND PRIOR. Increased pointer maintenance and apply to all Records in a Set.

Schema and Subschema Decisions

DB Procedures in Schema

3rd level of Privacy?

Validation?

Editing

Etc.

What Subschemas?

Who can use?

Privacy - Ommission
- Locks on Areas/Sets/Records?

Mapping Areas/Realms? 1:1 or single record type realms?

Individual Program Decisions

DML Navigation Strategy

Is it right?

Is it efficient?

Watch Currency Indicator Settings

Erase

Ensure only correct Records deleted.

DML Restrictions

Does the Programmer know limitations imposed by Sets excluded from Subschema and by Privacy Locks.

Opening Realms

Usage-Mode? RETRIEVAL/UPDATE, PROTECTED/EXCLUSIVE?

Use minimum level of protection necessary.
Area or Page Locks?

Size of Success Units

Should Batch Programs be broken into smaller units to reduce contention?

TP - update in one phase only, preferably the last one.

Areas

Mapping to Files

Usually 1 to 1, may be N to 1 or 1 to N. Decide on basis of Operational Convenience and Efficiency (eg Head Movements could overlap if data on different Transports).

Area Size

Calculate generously. Remember to allow for growth, Calc chain pointers and Logically Deleted records. Leave some spare page numbers between Areas. Smaller Area Size for quicker dumping.

Page size

Standard IDMS

Fixed for whole DB except Directory. Usually around 4 Kbytes.

Keep pages large compared with Records to lessen possibility of overflow (10:1).

IDMS X

Adjust Page Size to suit Record and Set Occurrence sizes in file.

Buffers and SDL

Each Buffer slightly larger than a Page. Minimum of 3 buffers, usually 5-10. Test in Pilot Scheme.

Make use of Page Reserve facility? (V L Records?)

Program Independence of Storage Schema Decisions

Remember that number of Logical and Physical Design decisions have implications on whether programs can be unaffected by changes at the Storage Schema Level and (where such independence is possible) on how the program must be coded:

- Set Order PRIOR/LAST or a need to FIND PRIOR/LAST in a Set. Need either an Indexed Set or a Chained Set with Prior Pointers
- more than One Record Key on a Record type or an Ordered Record Key. Demands IDMS X Record Indexes
- placement DIRECT. Demands special DML syntax
- placement PAGE DIRECT. Demands concealment via a DB procedure
- Record/Set Indexes in a different Area from that containing the Records they cover. Program must READY the Index Realm.

Initial Database Loading

Sequence of loading - all occurrences of a Record Type together or all Record occurrences in a Set occurrence together (Owner plus Via members)?

Any pre-sorting of Records?

Effects of desired Insertion/Retention class?

Names - Rules and Standards

Records

16 characters maximum (also field/group names). Make name meaningful by tying to Entity Name. Start with "Rn-".

Sets

16 characters maximum.

Tie to Soft-box Model Relationship Name.

Start with "Sn-" where "n" = Record Number used in Owner Record's Name.

Appendix 2 IDMS SIZING FORMS

Here is a selection of recommended forms, which will prove useful when sizing within an IDMS environment.

Database Record Population Form completed per area and used for media sizing

Set Population Form volumes per set type guidelines for extent of navigation

Transaction Loading Form receives results of calculations from the next two forms

Batch Workload Form useful for checking on possible bottlenecks eg:

- area/page locking
- journal
- long transactions
- disc controller

Transaction Sizing Form DML for each success unit

Also useful for calculating path lengths.

DATABASE RECORD POPULATION FORM

Database

Page.... of....

Completed by.....

Date.....

Record Type	No. of Records	Average Size	Maximum Size	Calc Key Length	Size of Prefix

TRANSACTION LOADING FORM

System Completed by Date Page of

Transaction Name	Per Day ... Hours		Peak 1 ...to...		Peak 2 ...to...		Area Name	Blocks Read	Blocks Written	Blocks to Journal	μ Secs Mill Time
	Vol	%	Vol	%	Vol	%					

BATCH WORKLOAD FORM

System Completed by Date Page of

Program	Freq of Running	SCHEDULE		COMPUTED					
		Earliest Start Time	Latest Finish Time	No. Blocks Record	No. Blocks Written	No. Blocks to Journal	μ Secs Mill Time	Min Elapsed Time	
									.

TRANSACTION SIZING FORM

System..... Completed by..... Date..... Page... of...

TRANSACTION TYPE

DML Verb	Record	Set or Area	Av. No. recs accessed	Av. No. Blocks read	Av. No. Blocks Written	Area Name	Av. No. Blocks to jnl	μ secs mill- time	Notes

CHAPTER 10

CHANGING THE DATABASE

INTRODUCTION

This chapter considers changing the Database:

- why changes may be needed and the sort of things that need to be changed?
- how to make changes?
- the impact of changes on existing programs.

WHY WILL CHANGES BE NEEDED?

The process of Data Analysis and the general philosophy of sharing data are attempts to produce stable data structures which can be used by a number of Applications. Yet it is certain that changes to these structures will be needed.

This may be because the original work was rushed with management putting more emphasis on getting results out by an arbitrary due date, than on the quality of those results. Hopefully though, it is more likely that change to the Database is needed because the Application problems have changed:

- the Real World changes, usually slowly sometimes catastrophically (a take-over, a complete re-organisation, new legislation). In such cases existing Applications will change in their logic and in the volumes they have to handle
- the patterns of user access may alter. Indeed it is likely that the very presence of a useful Information system (correct data accessible through Decision Support as well as conventional Application interfaces) will change the way users operate. Hence the relative priorities of different Transaction types could alter and quite new Transaction types be required. As a result the original tuning design decisions may become invalid
- complete new Applications may be needed requiring the existing structures to be extended.

Two change requirements are involved here:

1. Inefficiency in the existing structures may need correcting. Overflow may be excessive, Areas may be getting too full, there can be logically deleted Records cluttering-up the Sets and the Pages. The development of this sort of situation can be detected at an early stage by regular examination of the IDMS statistics. It will be remembered that these may be:
 - DYNAMIC produced by the run of an Application within a Service. These show, for example, the amount of overflow that occurred during this run
 - STATIC produced by the DUMP utility. These show, for example, the amount of space currently in use within an Area for occurrences of each Record type.

It is absolutely essential that some mechanism is established on an installation for reviewing these statistics at regular intervals, so that deviations from the expected situation are detected at the earliest possible stage. The old saying "when an Area gets full, sack the DBA" is appropriate here!

2. Additions and/or deletions of structures may be required. For example, New Item types, Set types and Record types might have to be added to the existing structures to cater for a new Application.

WHAT NEEDS CHANGING?

There are three places where change may be required:

1. The Physical Database.
2. The DDS and directory definitions of the database (together with recompilation of subschemas and service descriptions).
3. User Programs.

It is possible that a particular change could affect only one of these eg a change to the physical Database via the Index Tidy utility to improve Index Layouts or via the Re-structure Utility to remove Logically Deleted Records will not affect the Directory or User programs (except to reduce the run-times of the latter).

But these three do interact and many types of change will affect at least two of them and sometimes all three eg removing a data item from a Record type will involve:

- re-writing and re-compiling the Schema, and some Subschemas as well as re-compiling the Storage Schema
- changing the Database to remove the item from each Record Occurrence of the type
- re-compiling and perhaps changing user programs that refer to the Record type in question.

The rest of this chapter considers the process of change including the instruments available to make changes and the sort of changes each one can make. At the end the effect of change on User Programs is considered ie the degree of Data Independence that is available.

THE PROCESS OF CHANGE

Introduction

This section considers some general principles which should be followed whenever the Database is to be changed and then looks at three types of change:

- increasing Area sizes
- changing Records in place
- changing Records by unloading and re-loading them.

GENERAL PRINCIPLES

1. Always plan the whole process with great care. Database changes, however trivial, should never be embarked upon without a meticulous examination of how to make the change and of the impact of the change, especially on User Programs. Useful impact analysis reports can be extracted from the DDS and with the RPTS utility from the Directory.
2. Consider batching a number of changes together and making them to the Database all at one time. It is much less effort to make three changes together than to make them separately at three different times.
3. Use the Integrity Check Utility. This utility checks all or part of a Database for consistency. Among the checks it can be told to perform are:
 - page formats are correct eg Page Number is one greater than previous Page, there are no more than 255 Records per page etc
 - records are in their correct Page Range and are in an occurrence of any AM Set they are to be a Member of
 - chains are a closed loop (Set Chains, CALC Chains, Fragment Chains)
 - for Chained Sets, that each occurrence has only one Owner
 - for Sorted Sets, that the correct Set Order has been maintained and the Duplicates requirement (DF, DL or DN) met
 - that Record and Set Indexes are consistent internally and with the data Records they cover.This utility should always be run on a Database after a major successful Recovery and after it has been changed (together with careful Regression testing ie ensuring that the altered Database produces the same processing results as before).
4. Dump the Old Database before changing it (and restore the Dump to make sure that the dumping process has been successful). If possible also run Integrity check on the Old Database before any change is attempted.
5. Carry out the planned changes and the Integrity Check/Regression tests on a test Database first (say 10% of the real Database). This is not only a check on the change process, it will also give some realistic run-time figures so that the cost and time involved in making the real changes can be established. These may be greater than you expected.

Increasing Area Size

Introduction

In Chapter 9, the process for calculating Area Sizes was examined and it was emphasised that this calculation should be done generously.

If it becomes clear later, during live running, that the generosity was insufficient then some measures must be taken to avoid an Area Full Condition. There are a number of approaches that can be taken but the three most popular are:

- to add Pages on to the end of the Area (AREA EXTENSION)
- to increase the size of each Page (PAGE EXPANSION - only available with IDMS X)
- to unload the Area's contents and re-load into a larger Area. This can be done with the Re-organise Utility and in a steady growth rate situation is the only permanent cure for the problem ie the first two methods are often only a temporary solution.

The Re-organise Utility is described later in this section. A little more detail of the first two methods follows.

Adding Pages to an Area

This Area Extension process is effectively adding second level overflow pages to the end of the Area. This certainly would be the view of CALC Records because their Page Range must remain unchanged if existing CALC Records are still to be retrievable by the CALC algorithm.

Non-CALC Records could use the new Pages as Home Pages if required.

The process in outline is:

1. Dump the Area.
2. Change the Storage Schema, the Service Description and any Subschemas that refer to the Area. (Page Ranges are held in the Subschema Tables.)
3. Restore the dump into the larger Area with the EXTEND parameter set to YES.

Page Expansion

This facility, only available in IDMS X, provides more space in the Area by making each Page bigger. There are no problems with the CALC Algorithm since the divisor "Number of Pages in the Area" will be unchanged.

An ultimate limit to this process will be the maximum of 255 Records per page and any existing overflow will not be retrieved. However, it may delay the need for a use of the Re-organisation Utility for a considerable time or even indefinitely so it can be very valuable.

The process in outline is:

1. Dump the Area.
2. Change the Storage Schema and the Service Description.
3. Restore the dump into the larger Area with the EXPAND parameter set to YES. As a result each Data Page is re-loaded with the Line Index at the end and the Page Header and existing Records at the beginning. The space between them is filled with zeroes. Space Management Pages are updated but their position does not change.

Changing Records in Place

This work is done with the RESTRUCTURE Utility which can perform a number of types of changes that do not involve a need to alter the position of a Record ie do not need to change its' Database Key.

The changes that can be made include:

1. Adding/Removing data items. The contents of data items can be initialised or changed through a Database Procedure interface.
2. Adding/Removing Prior and/or Owner Pointers for existing Chained Sets. When added the Pointers will be initialised to correct values.
3. Adding/Removing Set types. When added, the new Pointer Positions will be allocated but all Set Occurrences will be empty.
4. Data Compression can be introduced or removed for given Record types.
5. Record Indexes can be introduced or deleted. ← *easy to add*

DUMMY FIELDS
DUMMY SETS ↗
RESERVE SPACE
RESTRUCTURE
REORGANISE
NOT RECORDWISE

6. The Set implementation mechanism can be changed from Chained to Indexed or vice versa.
7. The Key on which a Set Index or Record Index is based can be altered.
8. Placement can be changed as long as it is not currently CALC or PAGE-DIRECT. For example, the Set type a Record is VIA could be altered. However, this would only change the clustering of subsequent new Record Occurrences. Since Re-structure never changes Database Keys, existing Records cannot be moved.
9. Removing logically deleted records.

The Utility works from a set of run-time parameters and TWO Subschemas, one describing the Old Database and the other the new One. Logically it works by going through each affected Area, pulling out each affected Record Occurrence, changing it and writing it back. In the process any logically deleted Records are physically removed and Sets containing them repaired.

* If a Record is expanded and will not fit back into the Page it came from, it will be RELOCATED. This means a Tag is left in the Home Page Pointing to the new position of the changed Record in some other Page. In effect, Re-structure has its' own overflow mechanism. *

* Relocated records cannot be located by CAFS scans. *

The INDEX phase of the reorganise utility can be used to convert relocated records into normal records.

The ALTER and INDEX phases of restructure may also be used to "unfragment" fragmented records provided that the MINIMUM ROOT for the record clause is set to RECORD LENGTH.

Changing Records by an Unload and a Re-load

For some types of change, the Database Keys of affected Records must be altered eg increasing the Page Range for a CALC Record type or moving a Record type to a different Area.

The REORGANISE Utility will do this kind of work, unloading the affected Records and re-loading them with a different Database Key.

*OR
WRITE
OWN
RE-PAGE*

Among the changes that Re-organise can make are:

- alteration of Area sizes, Page sizes and Record to Area mappings
- returning overflow Records to their Home Page
- changing Index Placement and Index Record size decisions. The resulting new Indexes are tidy
- changing any previous Record Placement decisions including those for CALC Records
- removing logically Deleted Records.

Re-organise is now available with a simple two command interface IDMSRGUNLOAD and IDMSRGRELOAD. However, this is only suitable for small Databases and in many cases the more powerful interface must be used. This comprises up to seven phases separated by Sorts, each phase triggered by the IDMSRG command. This is a time-consuming process and should not be undertaken lightly. *On a large Database, Re-organisation run-times could well be measured in days rather than hours. Every effort should be made therefore to avoid having to Re-organise. *

This involves as much generosity as possible on such decisions as Packing Density during initial design and a creative use of techniques like Area Extension, Page Expansion and Re-structure, when change is needed.

NEEDS 3 DB F/3*

THE IMPACT OF CHANGE ON EXISTING PROGRAMS

IDMS Programs can be insulated from change by two mechanisms:

1. The use of Storage Schema independent programming techniques. These allow a program to be indifferent to such Storage Schema decisions as Record Placement and Set implementation decisions. For example, a program which needs to select a Record can be quite independent of the selection mechanism decision in the Storage Schema ie whether it is CALC, PAGE-DIRECT or a Record Index.

Such independence requires the use of the right verb eg "FIND ANY Record Name USING Keyname" and if Page Direct is to be used, a Database Procedure must perform the Algorithm. But in return for this extra effort, the program code does not need altering if the Storage Schema decision is changed.

Notice that if the Program logic depends on the use of Prior Pointers or Direct Location Mode then it is not possible to make the program independent of those Storage Schema decisions.

2. Subschemas

As we know, the Subschema used by a Program limits the record types and Set types the Program can see, while within a Record the Subschema can further limit the data Items that can be seen. In addition the Realm concept provides some insulation from the full detail of Record to Area Mappings.

In theory this is a Data Independence mechanism eg only changes to Record and Set types included in the Subschema View will affect the program. The compiled program with its existing Subschema Tables should continue to work correctly on an altered Database provided the alterations are to Records and Set types completely outside the Subschema View.

This is the theory. In practice it may be necessary to re-process Subschemas after a change even if the existing compiled Subschema Tables will still work on the altered Database. This is because, as we saw in Chapter 6, any change to the Schema requires a new Directory to be established. Unless an existing Subschema is re-compiled, so that it appears in the new Directory, it could not be used by a new COBOL Program. Nonetheless Subschemas that still work on the altered Database obviously have a much lower priority for re-compilation than Subschemas that will no longer operate.

Changes starting at the Storage Schema level are less drastic than Schema changes since a new Directory is not required. However, whenever there are any changes to a Storage Schema it is advisable to at least re-generate the Subschema Tables for all associated Subschemas. This is usually enough. But some Storage Schema changes will require Subschemas to be completely re-compiled ie the Code analysed, details of the Subschema written to the Directory and then Subschema Tables re-generated. One example is changing a Record type's Page Range. The Subschema Processor can be told to do a complete compilation for a Subschema or merely a much faster re-generation of the Subschema Tables.

Schema changes are usually adequately concealed from existing programs by a Subschema re-compilation. But in a few cases, it may also be necessary to make changes to the original Subschema DDL. For example, suppose a new Set type is introduced between existing Record types. Subschemas which include the Owner and/or Member Records should not at first sight require a change. However, unless the Set is mentioned in the Subschema DDL, any update program requiring to change occurrences of the Member Record type could be affected by Subschema constraints.

REHUR DMS II

The theory of Data Independence is bedevilled therefore by various practical considerations. The situation can be summarised as follows:

1. The original Program source is always protected from re-writing unless the actual logic is affected eg it needs to process a new Record type. But Data Independence is irrelevant in such cases anyway since the concept is intended to insulate Programs from Database changes that do NOT affect their logic.
2. The original Program source is also protected from re-compilation although there are some gaps in this protection especially if Record Formats are changed. The protection here depends on how Record Formats were defined in the original Subschema Code. This can be done in one of two ways for a Record type:
 - a) The particular Items required can be specified explicitly (Explicit Mapping). In this case the Program does not need re-compiling if say a new Item type is added to the Record type, since the Record Area in question need only contain the Items the Program processes. However, this is less efficient at run-time since the individual pieces of the Record type have to be moved between the Page Buffer and the Record Area one by one.
 - b) The whole Record may be copied into the Subschema Code. In this case, the Record Area in the Program will specify the detail of the whole Record. This means the Program usually needs re-compiling if the Record Layout is altered so that the Record Area format can be updated appropriately. On the other hand, the copying technique gives greater run-time efficiency since the whole Record is moved as one unit between the Page Buffer and the Record Area.

Data Independence always requires this choice between run-time efficiency and protection from change. In this particular example it is normal to go for run-time efficiency and copy each Record type into the Subschema unless Privacy considerations intrude.

3. Changes to the Schema require a re-compilation sooner or later of all Storage Schemas, Subschemas and Service Descriptions.
4. Changes below the Schema level are less drastic since a new Directory is not involved. The interaction of the Various Components of the compiling system is complex however, and careful study is needed when any change is contemplated in order to establish what needs to be done.

The reader is referred to Part 5 of the Reference Manuals for further information. Chapter 8 of that manual contains some useful Tables giving the actions required for various types of change.

Finally as an example, consider one particular kind of change. Suppose a new Record type is to be added to a Database design which is to be linked to some existing Record types by some new Set types. The following changes must be made:

1. The Schema must be changed to define the new Record and Set types. So alter the DDS, re-generate the Schema DDL and run the Schema compiler to create a new Directory.
2. Change the Storage Schema(s). Decisions on the new Record (eg Placement) and Set types (eg Set implementation mechanisms) must be made and passed to the software. So change the DDS, re-generate the DSDL and run the DSDL Processor.
3. Subschemas associated with the Storage Schemas must be re-generated. This is best done by complete re-compilation since it will put the detail of the Subschemas on to the new Directory.
4. Existing Application Programs will not be affected nor will the Service Descriptions. However, at some point, the Service Descriptions should be re-compiled to place their details on the new Directory.
5. Generate a new Subschema for the Re-structure Utility and for a load process to use.
6. Run Re-structure to stretch existing Records where necessary to leave room for the new Set Pointers.
7. Run a load process (Database Load or a User Program) to load in and connect the new Record Occurrences into the existing structures.
8. Use Integrity Check to confirm that the altered Database is correct.

Appendix 1 CONVERTING FROM IDMS 200 TO IDMS 320

The major change required is to alter the 200 Schema DDL into 320 Schema DDL plus Storage Schema DSDL. This can be done with a supplied SCL command IDMSCHEMCONVERT.

This command will read a file containing 200 Schema DDL and write out two files one of 320 Schema DDL and the other of DSDL. If no change is required to this output, the command can then be told to go on to the equivalent of IDMSDIR. This creates the new Directory by:

- initialising it
- compiling the 320 Schema DDL
- compiling the DSDL under a Storage Schema name of STANDARD
- running the CLUC Utility.

Subsequently existing 200 Subschemas must be re-compiled.

Further details of these changes can be found in the Reference Manual Part 2 - DATABASE ESTABLISHMENT - Appendix 9.

ICL

TRAINING

VIDMD WORKBOOK

N0267

All rights reserved. No part of this document may be reproduced or utilised, in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without permission of ICL, Consultancy and Training Services Division.

This publication is for training purposes only. It should not be regarded as a full specification of any ICL products or services. ICL makes every endeavour to ensure the accuracy of this document but does not accept liability for any errors or omissions.

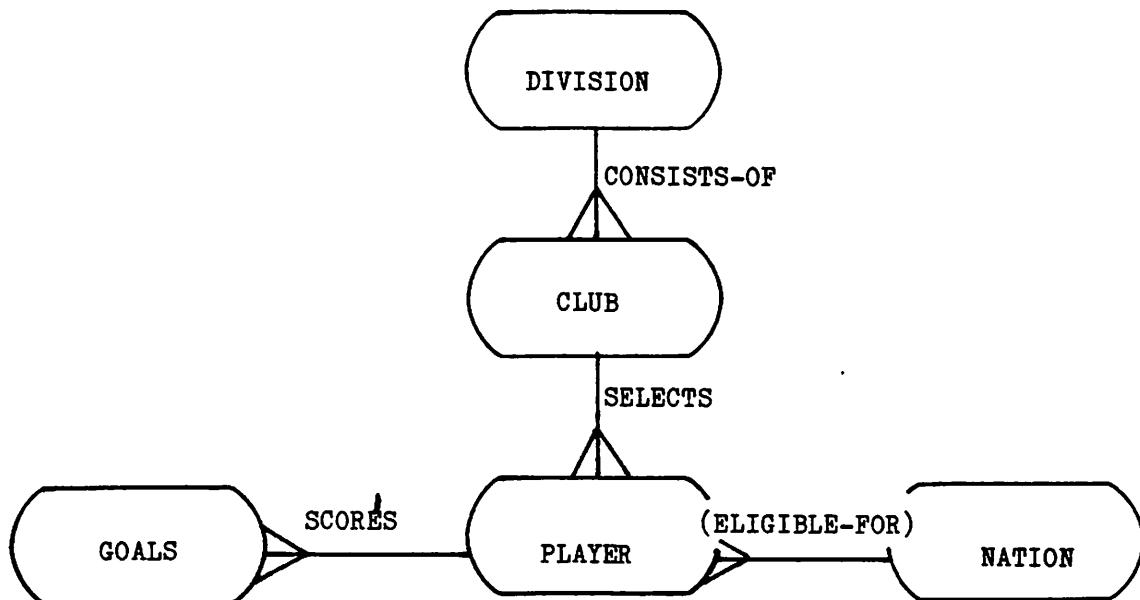
Training Publications
Issued by ICL, Consultancy and Training Services Division
Beaumont, Old Windsor, Berks

© International Computers Limited, 1987

IDMS SETS EXERCISE 10

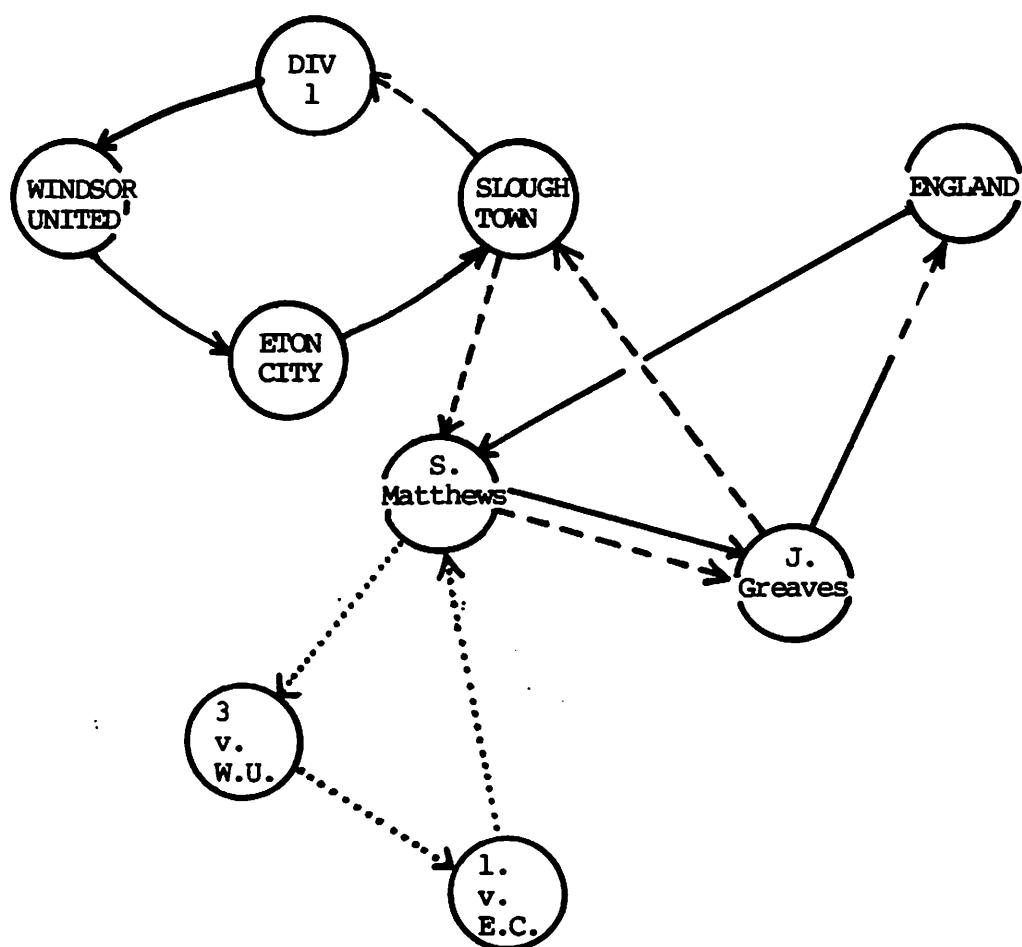
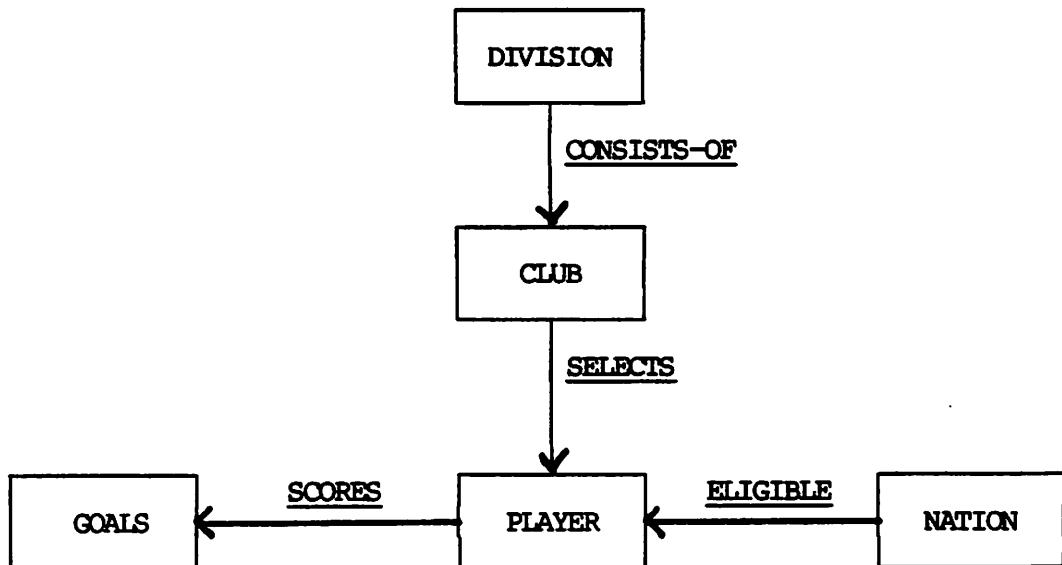
A database is to be set up for a much simplified version of the English Football League. The league consists of 4 Divisions each of which contains on average 20 Clubs. Players are selected by Clubs and at any one time up to 20 Players are eligible to play for their Club. During a season, goal information will be recorded against Players scoring for their Clubs. One Goals entity will record all goals scored by a player in one match. Details of which Nation a Player is eligible to play for will also be held.

A data analysis of the football league has produced the 'soft-box' model shown below.



1. Convert this model into a database drawn as a simple Bachman Diagram.
2. Draw the record and set occurrences of your database design, showing that player Stanley Matthews scored three goals for his club Slough Town when playing a first division match against Windsor United, and that he is eligible to play for England.

SOLUTION 10



LOGICAL DATABASE DESIGN EXERCISE 20

Windshire County Council has adopted a database approach to their information requirements and intend to implement an IDMS system to support it.

The organisation has produced an outline data model and from this the personnel project team are in the process of deriving a detailed data model. For the purpose of this exercise, the model has not been fully completed. Figure 1 shows the soft-box model of the personnel area of the organisation. The following informal description may help you interpret it.

The council's 20,000 employees are organised into several major Departments, each comprising 6 or 7 sections. Every employee is appointed a job of a defined type (eg Secretary, Section Head) which carries one of 12 defined Salary Grades. The grade of a job type rarely changes, should it do so the new grade is recorded, but no record of the previous grade need be kept. Most employees hold several different jobs during their career with the council and details of employees' careers must be preserved.

The personnel function includes training and thus will also form part of the information system. A separate training scheme is planned and recorded for each employee. No two schemes are the same and each one consists of one or more events. An event may be attendance on a course (either internal or external), or may be job experience. Obviously, the details describing course attendance are different from those describing job experience. Many types of course are run and the course schedules are recorded in the system by means of course occurrence details.

As well as recording and maintaining the council's information, the system must be able to support different types of queries. Examples of these along with their expected volumes are given below:

1. List all current Section Heads. 30 week
2. Which employees are currently on Grade 6? 10 week
3. Who is booked on the next "Introduction to Management" course and how many places are still available? 75 week
4. What sick leave has A. Grey, personnel number M6251, had in the last 3 years? 50 week
5. Check the address recorded for B. White. If incorrect, amend it. 20 week

6. Give details of E. Thick's training scheme, including details of all courses booked. 100 week
7. Which Departments has F. Bloggs worked in since joining the council? 40 week
8. How many Junior Secretaries have not yet attended an Induction Course? 30 per week
9. Which Sections have vacancies for Receptionists? 45 per week

Using this information, map the data model down to a Logical Data Structure. Determine the entry points and required selection keys. Produce a Schema Diagram of the logical structures.

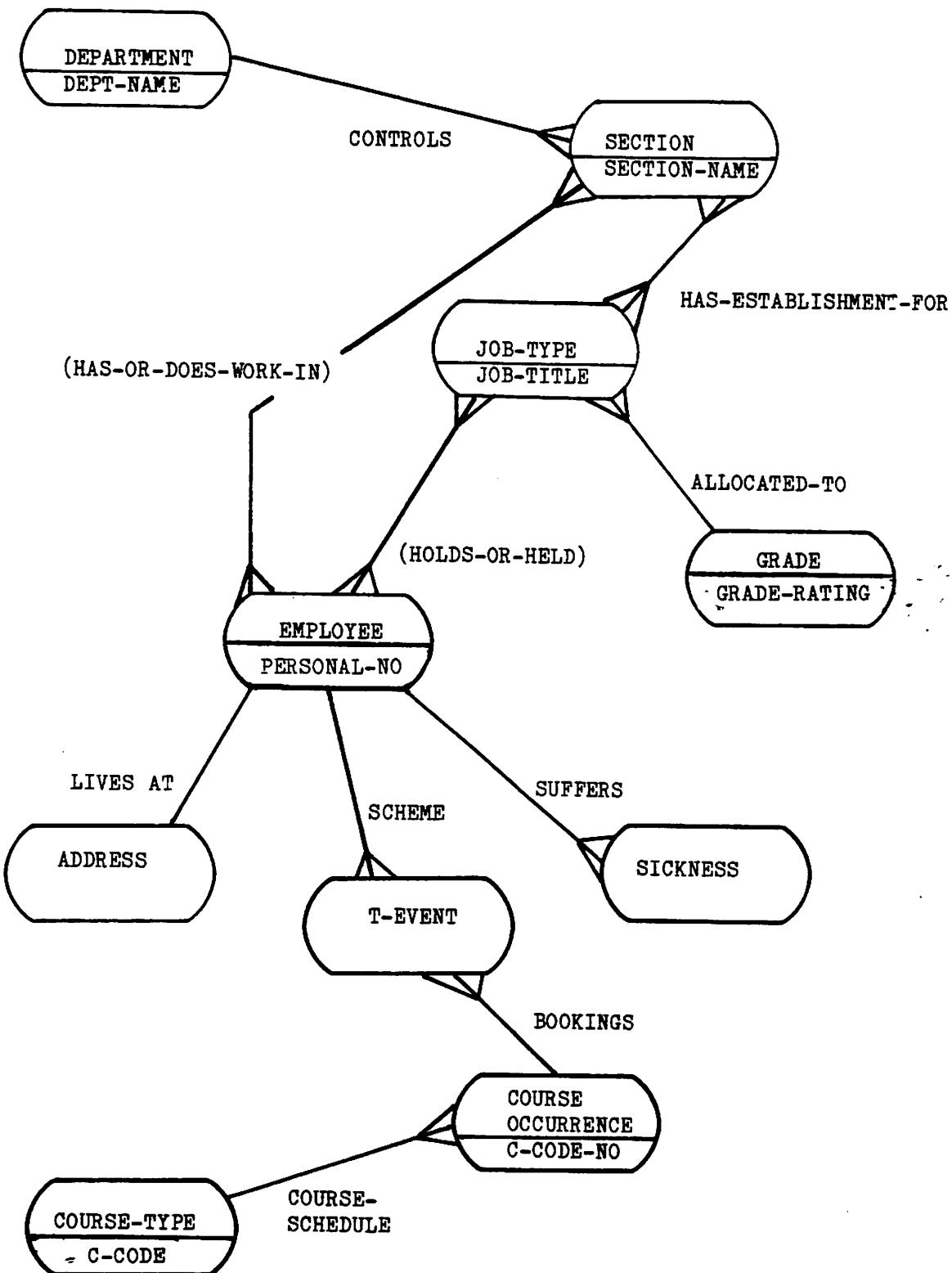
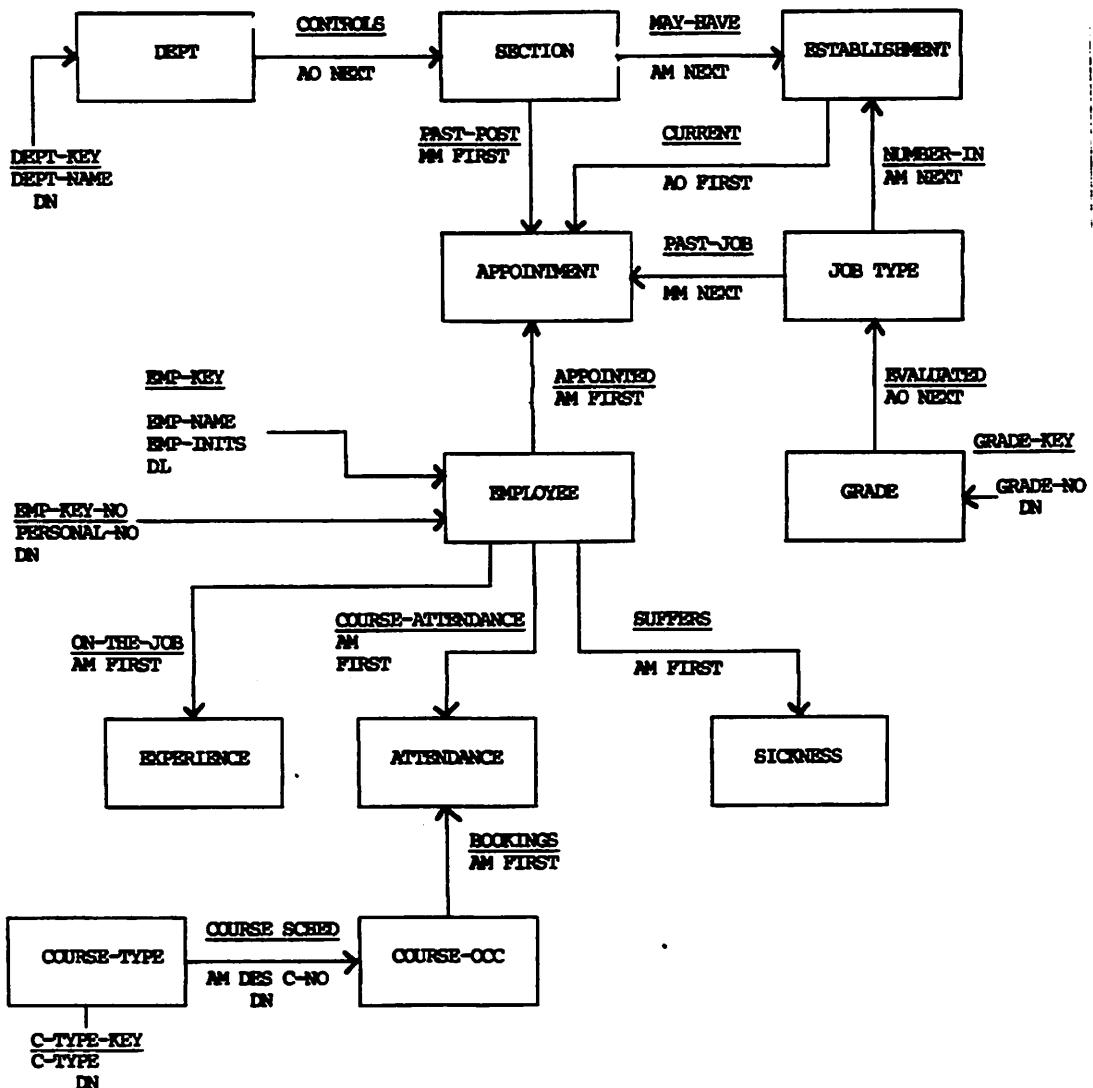


Figure 1 Soft-box model of Personnel area

N0267

SOLUTION 20



S - 2
NO266-5
© International Computers Limited 1986

S-2
NO266-5

POINTER EXERCISE 30

The storage diagram has been produced for the logical schema described in Exercise 20.

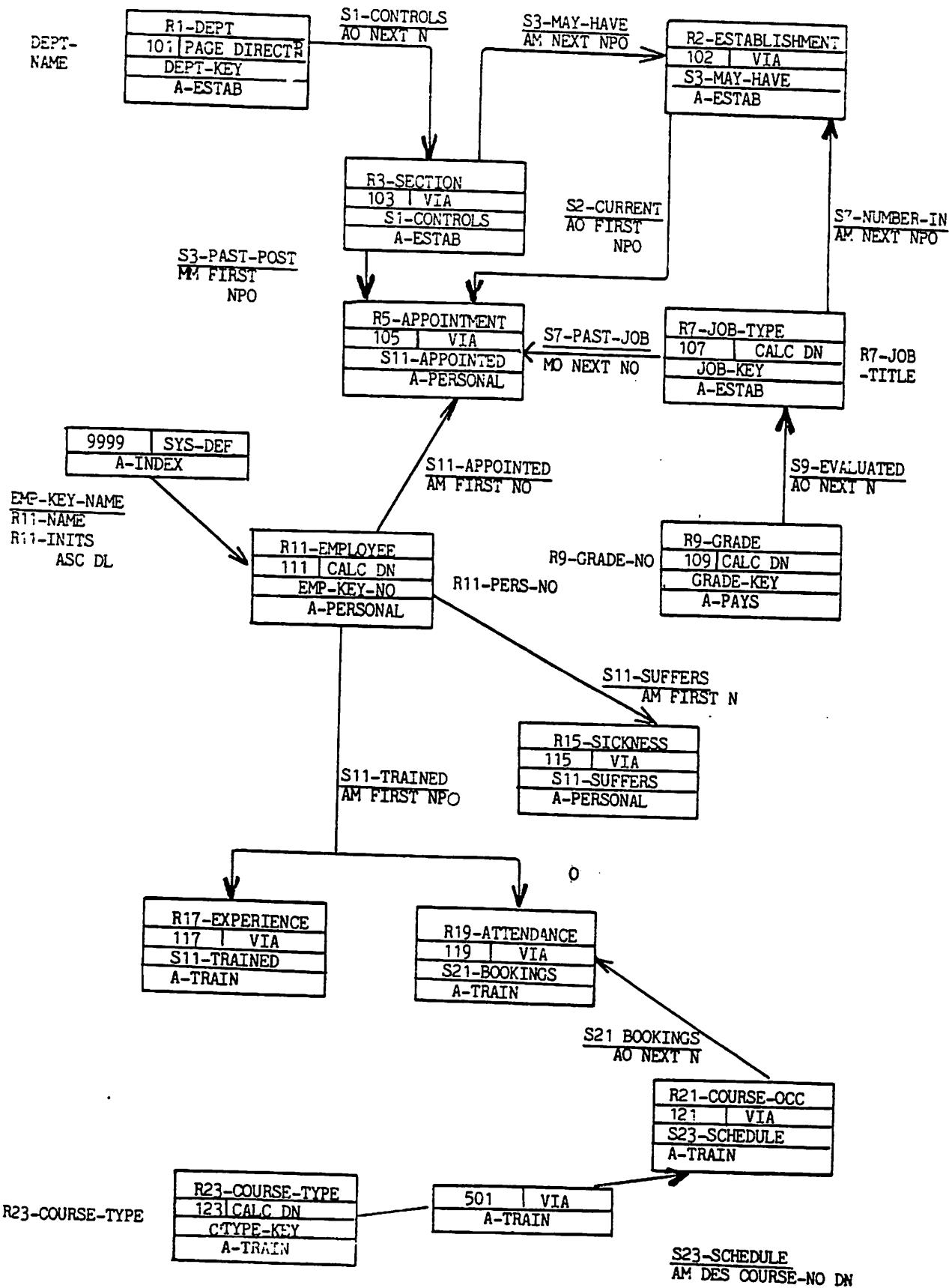
You are required to establish the total number of pointers in one occurrence of each of the following record types:

R1-DEPT, R3-SECTION, R7-JOB-TYPE
R5-APPOINTMENT, R11-EMPLOYEE,
R19-ATTENDANCE, R21-COURSE-OCC,
R23-COURSE-TYPE

Code = 2 extn

Member = Own pointer

Indexed = Own 2 pointers
Member



NØ267

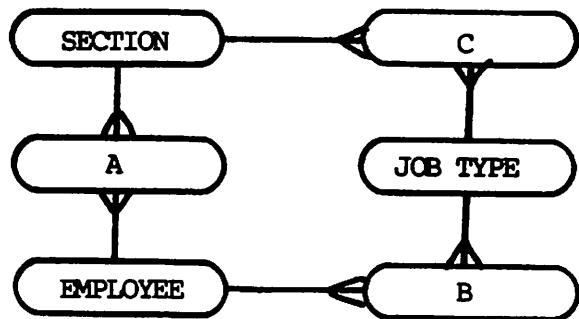
SOLUTION 30

R1-DEPT	3	2 (CALC chain), 1 S1-CONTROLS
R3-SECTION	5	1 S1-CONTROLS, 2 S3-MAY-HAVE, 2 S3-PAST-POST
R7-JOB-TYPE	6	2 (CALC chain), 2 S7-NUMBER-IN, S9-EVALUATED, 1 S7-PAST-JOB
R5-APPOINTMENT	10	3 S3-PAST-POST, 3 S2-CURRENT 2 S7-PAST-JOB, 2 S11-APPOINTED
R11-EMPLOYEE	6	2 (CALC chain), 1 S11-APPOINTED 1 S11-SUFFERS, 2 S11-TRAINED
R19-ATTENDANCE	4	3 S11-TRAINED, 1 S21-BOOKINGS
R21-COURSE-OCC	2	1 S23-SCHEDULE, 1 S21-BOOKINGS
R23-COURSE-TYPE	4	2 (CALC chain), 2 to Index record

Comments

1. Establishment and other associated Record types

The incomplete Data Model given in the question contains a structure around SECTION and EMPLOYEE which contains Many to Many Relationships. If these are exploded, the structures look like this:

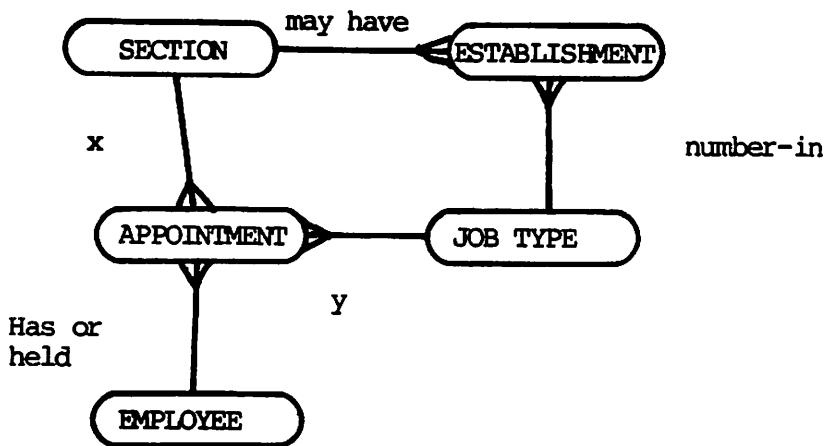


Where A - contains detail of each Employee in a given post eg Date appointed & Date terminated.

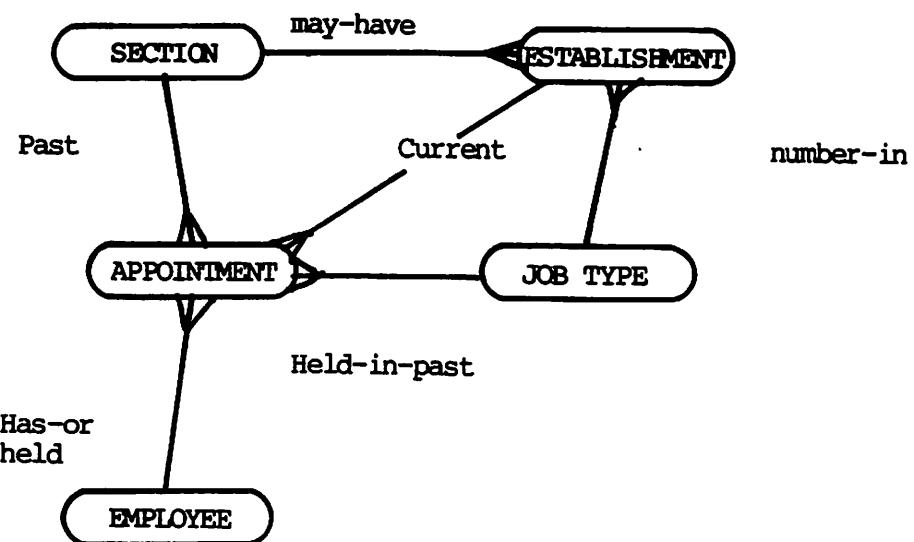
B - contains details of each Employee holding a given job.
eg date appointed and date terminated.

C - contains details of each Job type associated with a Section, eg Number of Posts of that type in that Section by Establishment and the Number filled at present.

There seems to be a good case for combining A and B to produce:



Finally there is a need to distinguish between the past situations and the current one as far as the connections between Employee, Establishment and Section go. If Relationships x and y are regarded as PAST, then a new Relationship between Establishment and Appointment could give the current situations:



So an Establishment Entity could represent JUNIOR SECRETARIES in the PARKS Section with Attributes specifying an establishment of 6 with 4 of the posts filled. Then the Current Relationship would lead to 4 Appointment Entities each owned by the appropriate Junior Secretary Employee.

2. Keys

Given the pathways needed to answer the enquiries specified in the questions, all keys shown are required except DEPT-KEY ON DEPT. This has been included because IDMS Records at the top of a structure, ie which have no Owners, usually need to have a Logical Key. This point will be clearer when the physical detail of IDMS is considered.

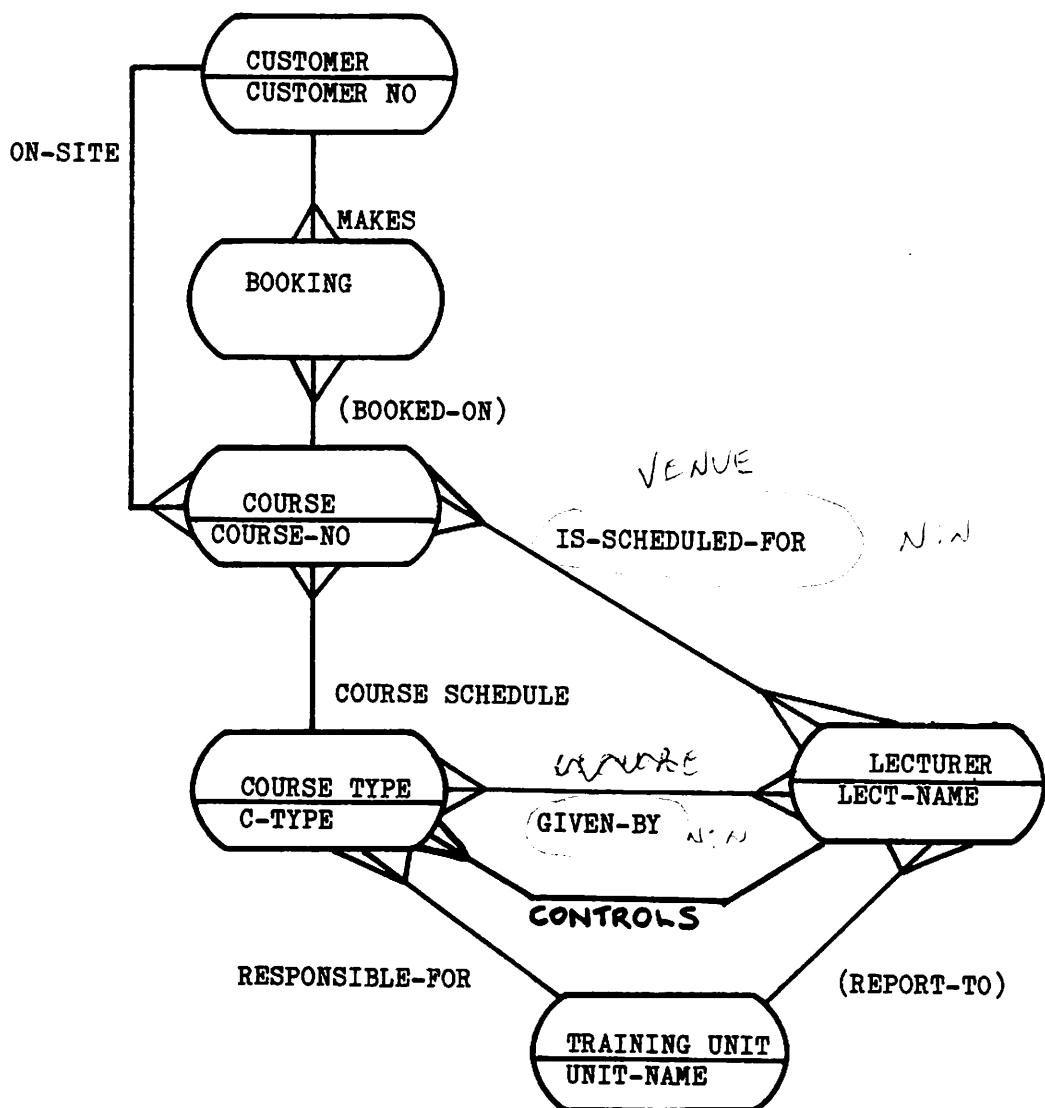
3. Address

Because each Employee is 1 to 1 with his address, the latter has been taken into the Employee Record as field(s).

DATABASE DESIGN EXERCISE 40

A system is needed to enable a Training Establishment to record, monitor and control course bookings, course attendance and lecturer scheduling.

Data Analysis has produced the 'soft-box' model below.



Entity Descriptions

TRAINING UNIT - the Training Establishment consists of 11 Training Units, each responsible for various courses. Each unit has on average 10 Lecturers working for it at any one time. In time a Lecturer may move to another unit, widening his ability in the types of course he can give.

COURSE TYPE - this is a specific type of course eg IDMD as opposed to IDMP. Each training unit offers approximately 20 different course types. Some run frequently and some very seldom. All are of 1 weeks duration. There are 230 course types at present.

LECTURER - a person able to give a particular type of course or courses, and being scheduled to give forthcoming courses. Past lecturer schedules are held for one year, before being archived. Some lecturers are Course Controllers, ie responsible for the other lecturers that take that particular course. (There is one controller per Course Type.) Although reporting to one training unit, a Lecturer may be called upon to give a course 'belonging' to another unit. This may be due to past commitments or some special ability.

COURSE - is an occurrence of a course type. It can be given by one lecturer alone or by several. Each course occurrence is uniquely identified.

A course when held at the Training Establishment is known as an "in-house" course. Courses held at venues arranged by Customers are known as "on-site" courses. Different information is recorded for in-house and on-site courses. All on-site course occurrence codes are suffixed by S and in-house courses are suffixed by H. Approximately 1600 in-house courses run each year, the on-site courses being in the region of 350 a year.

BOOKING - is a place on an in-house course reserved for one student. Most in-house courses average 15 bookings. Individual bookings are not recorded for on-site courses. Clearly it is not until the beginning of the course that we can confirm actual course attendance. Any bookings not taken up on the day of start of the course will be deleted.

CUSTOMER - an organisation that sends employees as students on the various courses. Alternatively a customer can request courses to run on-site ie at his own venue.

The system must support both on-line and batch enquiries. The requirements of the system include the production of the following information:

1. Course List

Showing all bookings made on a particular course occurrence, displaying student names and their companies. For an on-site course, the company name and expected number of students only is shown.

COURSE - KEY

2. Course Attendance

As above, but for actual attendees. This is only requested after the course has run.

COURSE - KEY

3. Course/Lecturer Schedule

WHEN SET

Showing for a particular course occurrence, the Lecturer(s), their training unit and the Course Controller.

COURSE - KEY

4. Weekly Schedule

Showing all the "in-house" courses due to run in a particular week (given the start date), with the expected number of students on each.

COURSE - DATE - KEY

5. On-Site List

Showing all the on-site courses to be run in a particular week (given the start date), with customer and venue.

COURSE - DATE - KEY

6. Ability List

All the lecturers able to run a particular course type, detailing their training unit.

C - TYPE - KEY

7. Course Controllers List

For each course type running in any week, the controller of that course type will receive a list of courses of that type and Lecturers giving those courses. The controller may in fact be one of the lecturers scheduled. If a course occurrence is an on-site course, then the venue will also be shown together with the customer name.

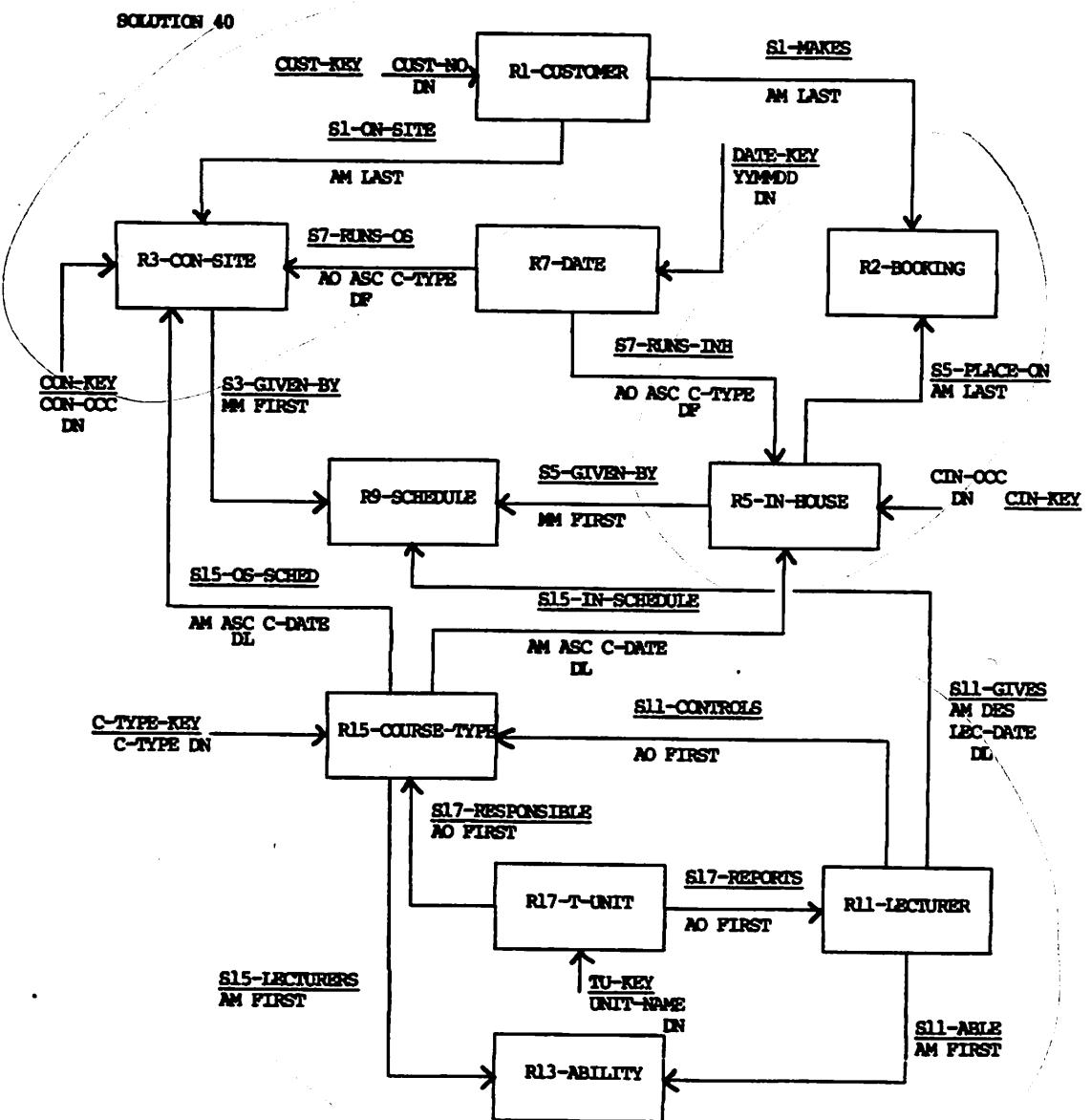
C - TYPE - KEY

Transaction Volumes

All enquiries, with the exception of enquiry (5) can be made on-line or batch. The batch output being slightly different in format from the on-line displays.

	on-line	batch
Enquiry (1)	30 day	1 per week for 35 in-house 8 on-site
Enquiry (2)	4 week	1 per week for 35 in-house
Enquiry (3)	20 day	1 per week for 43 courses
Enquiry (4)	1 day	1 per week for 35 courses
Enquiry (5)	-	1 per week for 7 courses
Enquiry (6)	10 week	3 monthly
Enquiry (7)	10 day	1 per week for 40 courses

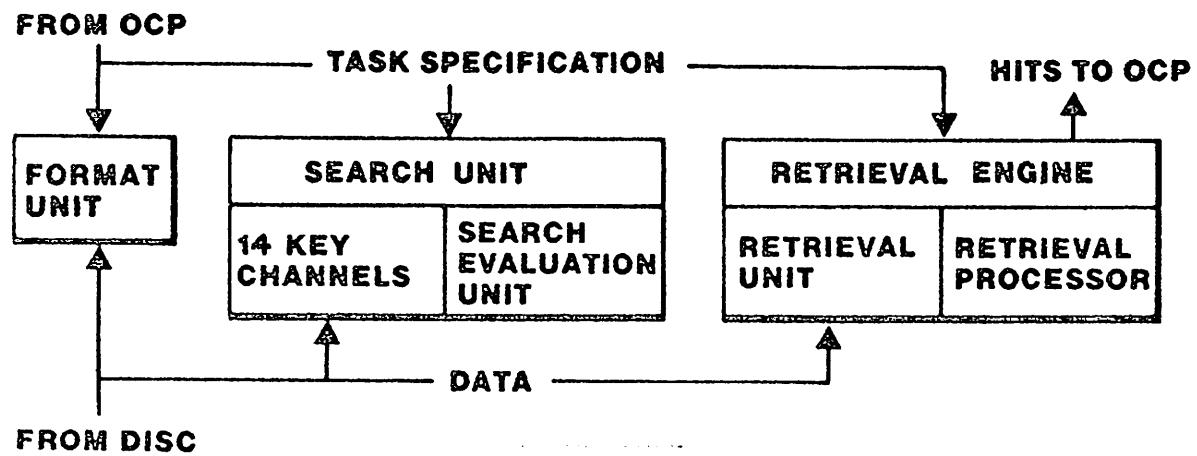
Produce a logical schema to support the necessary processing.



S - 6
N0266-5
© International Computers Limited 1986

S-6
N0266-5

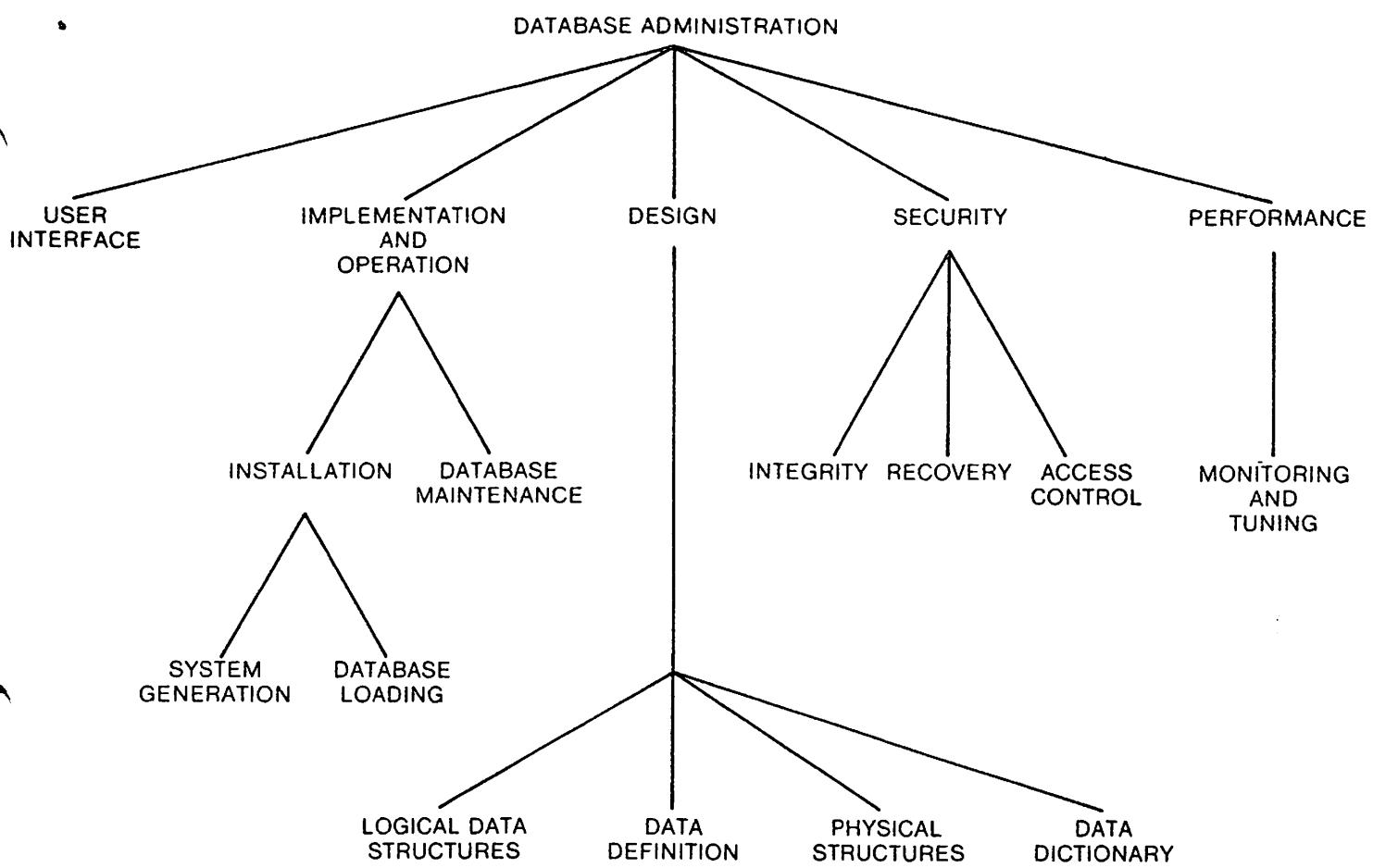
STRUCTURE OF CAFS HARDWARE



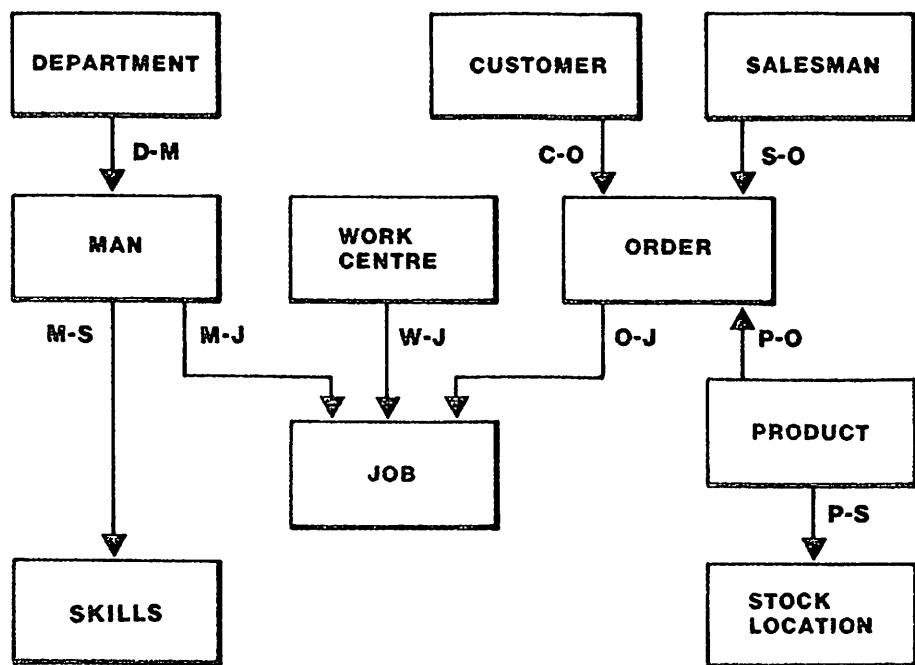
- ④ OCP sets up task specification aimed at all records in an IDMS area or sequence of records in a file

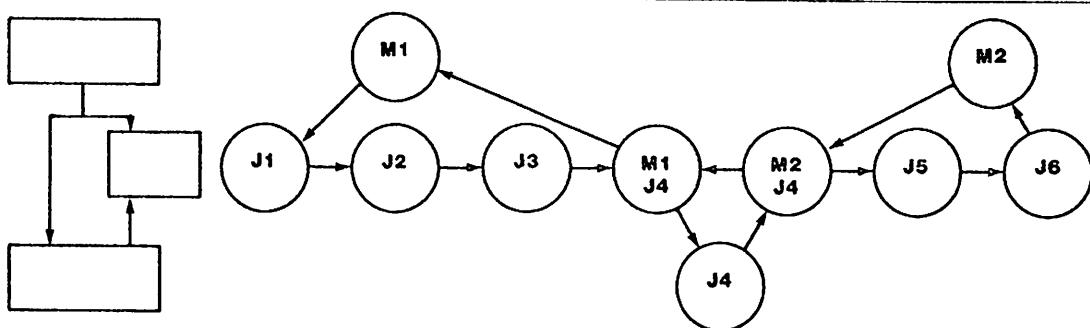
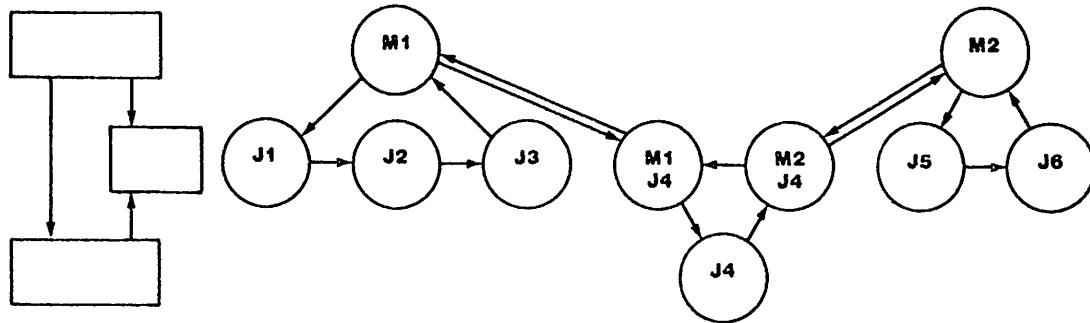
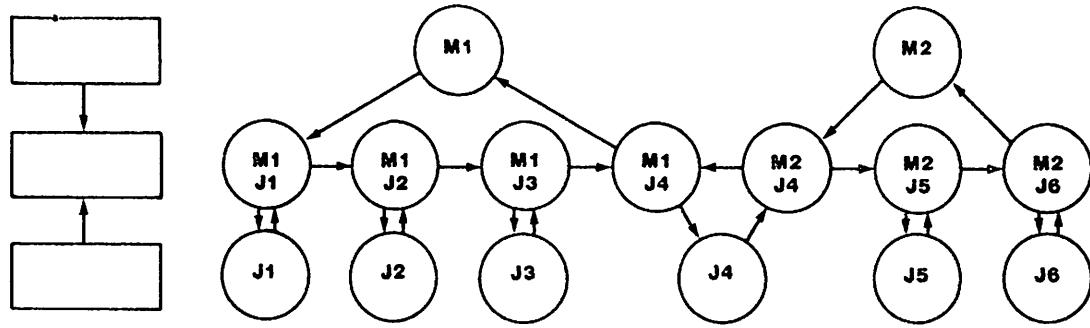
Selection criteria for each of up to 14 fields AND/ORed, EQUAL, > <, present/absent, fuzzy, totals, etc.

- ⑤ Search criteria loaded into key channels and formats of data into format unit (from DDS)
- ⑥ Data flows through CAFS hardware
 - Format unit detects record/field start/end
 - Search engine scans data in parallel for hits
 - Retrieval engine assembles data and totals
 - Sends record to OCP if a hit



A PRODUCTION ENVIRONMENT : SHOWING SET STRUCTURES





EXERCISE 45

The major on-line transaction for the course booking system is a Customer making a Booking for an in-house Course. This transaction is in two phases (ie 2 message-pairs):

For a particular Course Type, find the earliest in-house Course occurrence with a free place. On average two Course occurrences need to be examined to find a free place.

For that Course occurrence, create a Booking record.

For the records and sets involved in this transaction, determine:

The record placement modes.

The set modes.

The set pointers.

The objectives of physical design are to minimise disc accesses and minimise contention, while ensuring that record clustering is not too great.

For the determined placement modes, estimate the number of disc accesses required per day for this transaction.

Provide solutions for both IDMS and IDMSX.

100 times/day

see

S-6 EX 4 p

R1-CUSTOMER	
101	CACC-DN
<u>CUST KEY</u>	
<u>CUST - AREA</u>	

✓
ST-MNUSS
ATM LAST
N10

R2-BOOKING	
102	VIA
<u>SS-PLACE-ON</u>	
<u>IN-HOUSE</u>	

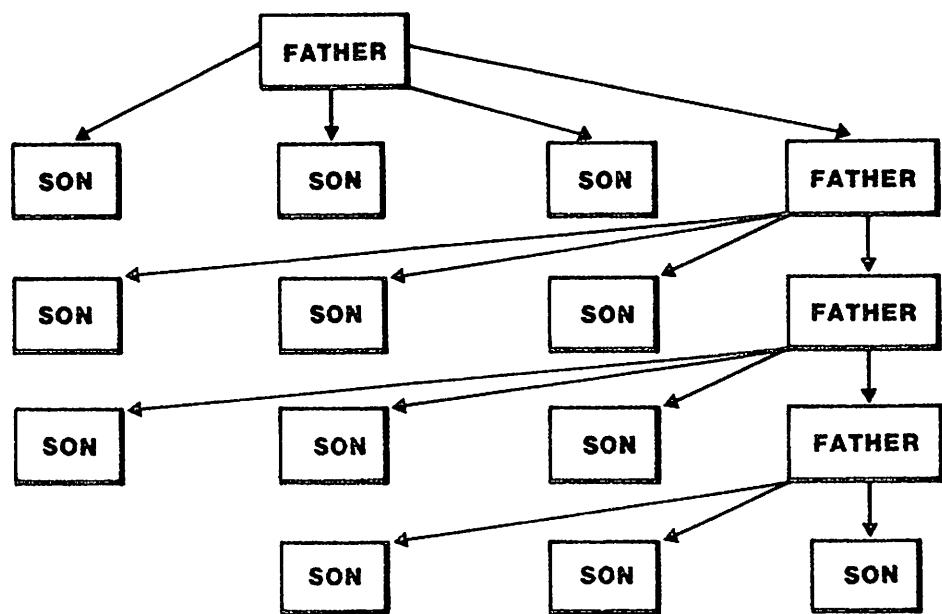
SS-PLACE-ON
ATM LAST
N10

R3-IN-HOUSE	
105	CACC-DN
<u>C-IN-KEY</u>	
<u>IN-HOUSE</u>	

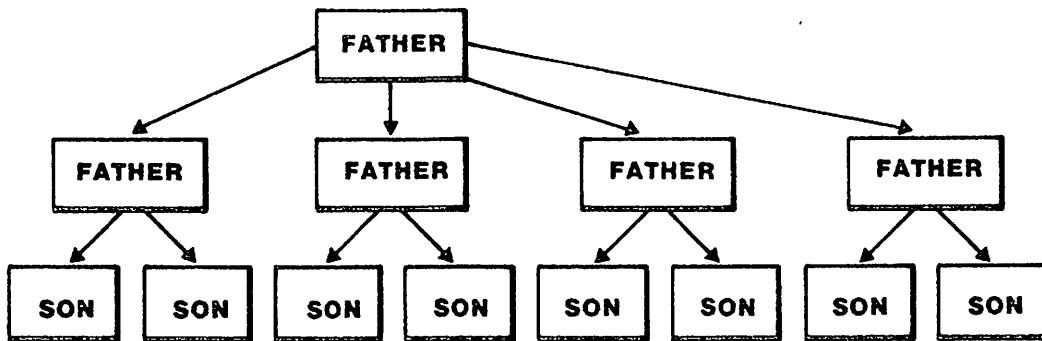
R15-COURSE-SCHED	
115	CACC-DN
<u>C-TRNG-KEY</u>	
<u>ADMIN-AREA</u>	

→
S15-IN-SCHEDULE
ATM ASC
C-DATE DL
N10 ←

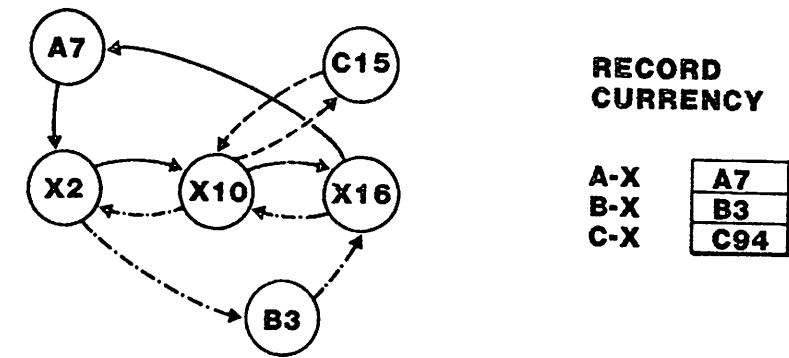
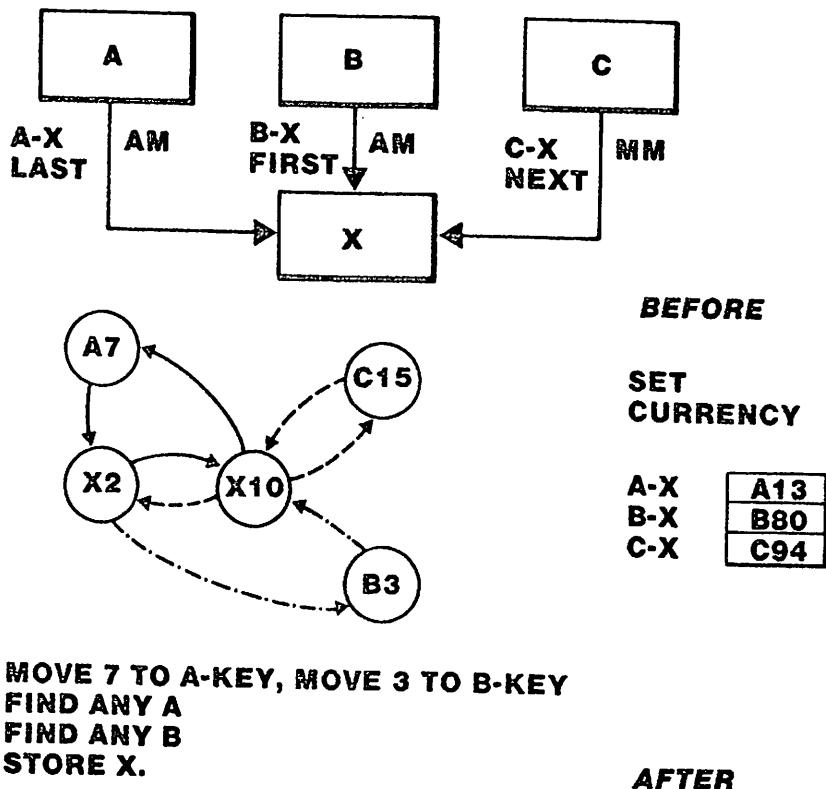
INDEX AFTER SEQUENTIAL LOAD (each son 50% full)

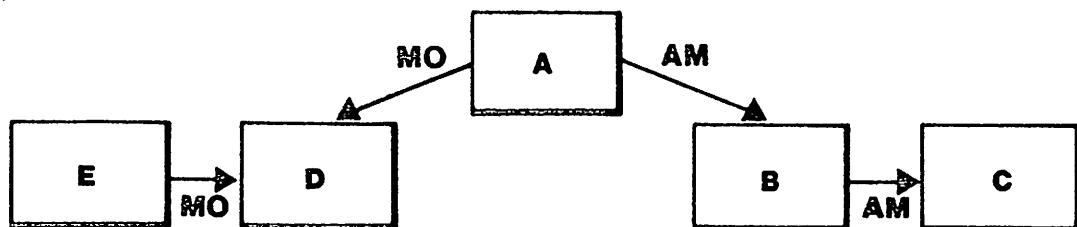


INDEX AFTER 'INDEX TIDY' (75% packing density)



STORING A NEW MEMBER RECORD



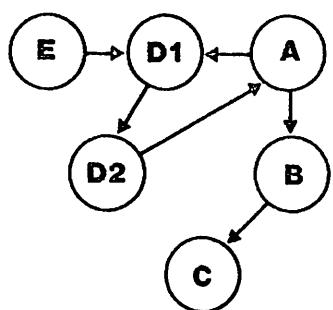


BEFORE

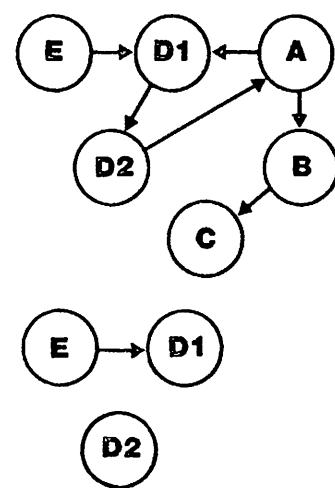
FUNCTION

AFTER

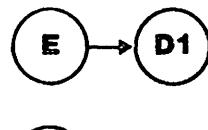
ERASE A



ERASE A PERMANENT



ERASE A SELECTIVE



ERASE A ALL



Table 4.1
IDMS page sizes and available space utilisation

Page size	EDS 100, 100D, 200		EDS 80, FDS 160, 640		Excellent Fit	Multiple of 512 bytes
	Pages per track	% Usage	Pages per track	% Usage		
19616	—	—	1*	100.00		
13028	1*	99.98	1	66.42		
9736	1	74.72	2*	99.27		
6444	2*	98.91	3*	98.55		
6144	2	94.31	3	93.96	YES	6.0
5632	2	86.45	3	86.13		5.5
5120	2	78.59	3	78.30		5.0
4800	2	73.68	4*	97.88		
4608	2	70.73	4	93.96		
4252	3*	97.90	4	86.70		
4096	3	94.31	4	83.52		4.0
3812	3	87.77	5*	97.17		
3584	3	82.52	5	91.35		3.5
3152	4*	96.76	6*	96.41	YES	
3072	4	94.31	6	93.96		3.0
2684	4	82.39	7*	95.78		
2560	4	78.59	7	91.35		2.5
2496	5*	95.78	7	89.07		
2332	5	89.49	8*	95.11		
2056	6*	94.67	9*	94.33	YES	
2048	6	94.31	9	93.96		2.0

** indicates a page size obtained by dividing the track into n pages and rounding down to a multiple of 4 bytes. This procedure gives values of 3156 for EDS100 etc. and 3152 for EDS80 etc., so the latter is chosen.

'YES' in the 'Excellent Fit' column indicates that the page size indicated by ** coincides on both families.

'% Usage' is calculated relative to the capacity if each track is formatted into one block.

Table C

COMPATIBILITY OF BLOCKSIZES

K Blocksize bytes	EDS200	EDS80	MDSS	FDS300	FDS2500
1. 1024	****		****	****	****
2. 2048	****	***	****	****	***
3. 3072	****	**	****	**	****
4. 4096	****	*	**	**	****
4252	****	**	**	****	****
5. 5120				**	***
6. 6144	****	**	****	*	****
6356	****		****	**	****
7. 7168				****	****
7340				****	****
8. 8192			*	**	***
9095			***	****	****
9. 9216			***	****	*
9300			****	****	*
12. 12288	***			****	
12564	****			****	*

Key: **** 95% or better
 *** 90 - 95%
 ** 85 - 90%
 * 80 - 85%

Note: Calculated by comparing given blocksizes against "perfect" blocksizes.

Disc Capacities for FDS300 Drive

Blocks /Track	Block Size (bytes)	Kb	'Perfect' Blocksize	Capacities				Packing %-		Suitable Allocation Unit Sizes
				Track (bytes)	Cylinder (bytes)	-Volume (Mb)-	Block Volume			
						Max	Mean			
1	24576	24	38676	24576	245760	201.03	195.62	63.5	64.4	1
2	19092	P	19092	38184	381840	312.35	299.36		100.0	1
2	18432	18	19092	36864	368640	301.55	289.38	96.5	96.5	1
2	17406	17	19092	34816	348160	284.79	274.00	91.2	91.2	1
2	16384	16	19092	32768	327680	268.04	258.54	85.8	85.8	1
2	15360	15	19092	30720	307200	251.29	243.00	80.5	80.5	1
2	14336	14	19092	28672	286720	234.54	227.37	75.1	75.1	1
2	13312	13	19092	26624	266240	217.78	211.39	69.7	69.7	1
3	12564	P	12564	37692	376920	308.32	295.51		98.7	1
3	12288	12	12564	36864	368640	301.55	289.38	97.8	96.5	1
3	11264	11	12564	33792	337920	276.42	266.28	89.7	88.5	1
3	10240	10	12564	30720	307200	251.29	243.00	81.5	80.5	1
4	9300	P	9300	37200	372000	304.30	292.02		97.4	2,1
4	9216	9	9300	36864	368640	301.55	289.38	99.1	96.5	2,1
4	8192	8	9300	32768	327680	268.04	258.54	88.1	85.8	2,1
5	7316	P	7316	36580	365800	299.22	287.15		95.8	1
5	7168	7	7316	35840	358400	293.17	281.70	98.0	93.9	1
5	6144	6	7316	30720	307200	251.29	243.00	84.0	80.5	1
6	6036	P	6036	36216	362160	296.25	284.66		94.8	3,2,1
6	5120	5	6036	30720	307200	251.29	243.00	84.8	80.5	3,2,1
7	5076	P	5076	35532	355320	290.65	279.28		93.1	1
8	4404	P	4404	35232	352320	288.20	277.28		92.3	4,2,1
8	4096	4	4404	32768	327680	268.04	258.54	93.0	85.8	4,2,1
9	3860	P	3860	34740	347400	284.17	273.40		91.0	3,1
10	3412	P	3412	34120	341200	279.10	268.87		89.4	5,2,1
10	3072	3	3412	30720	307200	251.29	243.00	90.0	80.5	5,2,1
11	3060	P	3060	33660	336600	275.34	265.24		88.2	1
12	2772	P	2772	33264	332640	272.10	262.12		87.1	6,4,3,2,1
13	2516	P	2516	32708	327080	267.55	258.07		85.7	1
14	2292	P	2292	32088	320880	262.48	253.17		84.0	7,2,1
15	2100	P	2100	31500	315000	257.67	248.85		82.5	5,3,1
15	2048	2	2100	30720	307200	251.29	243.00	97.5	80.5	5,3,1
25	1024	1	1044	25600	256000	209.41	203.52	98.1	67.0	5,1
38	512		532	19456	194560	159.15	155.84	96.2	51.0	1
61	128		148	7808	78080	63.87	63.32	86.5	20.4	1

Disc Capacities for FDS2500 Drive

Blocks /Track	Block Size (bytes)	Kb	'Perfect' Blocksize	Capacities				Packing %	Suitable Allocation Unit Sizes
				Track (bytes)	Cylinder (bytes)	-Volume (Mb)-	Max Mean		
1	24576	24	47444	24576	368640	325.51	317.77	51.8	52.3
2	23476	P	23476	46952	704280	621.88	593.00	100.0	1
2	22528	22	23476	45056	675840	596.77	570.41	96.0	96.0
2	21504	21	23476	43008	645120	569.64	545.13	91.6	91.6
2	20480	20	23476	40960	614400	542.52	520.40	87.2	87.2
2	19456	19	23476	38912	583680	515.39	495.54	82.9	82.9
2	18432	18	23476	36864	552960	488.26	470.57	78.5	78.5
2	17408	17	23476	34816	522240	461.14	445.47	74.2	74.2
2	16384	16	23476	32768	491520	434.01	419.76	69.8	69.8
3	15476	P	15476	46428	696420	614.94	586.39	98.9	1
3	15360	15	15476	46080	691200	610.33	582.68	99.3	98.1
3	14336	14	15476	43008	645120	569.64	545.13	92.6	91.6
3	13312	13	15476	39936	599040	528.95	507.99	86.0	85.1
3	12288	12	15476	36864	552960	488.26	470.57	79.4	78.5
4	11476	P	11476	45904	688560	608.00	580.46	97.8	2,1
4	11264	11	11476	45056	675840	596.77	570.41	98.2	96.0
4	10240	10	11476	40960	614400	542.52	520.40	89.2	87.2
4	9216	9	11476	36864	552960	488.26	470.57	80.3	78.5
5	9076	P	9076	45380	680700	601.06	573.83	96.7	1
5	8192	8	9076	40960	614400	542.52	520.40	90.3	87.2
6	7476	P	7476	44856	672840	594.12	567.88	95.5	3,2,1
6	7168	7	7476	43008	645120	569.64	545.13	95.9	91.6
7	6356	P	6356	44492	667380	589.30	563.27	94.8	1
7	6144	6	6356	43008	645120	569.64	545.13	96.7	91.6
8	5492	P	5492	43936	659040	581.93	556.89	93.6	4,2,1
8	5120	5	5492	40960	614400	542.52	520.40	93.2	87.2
9	4820	P	4820	43380	650700	574.57	549.84	92.4	4,2,1
10	4276	P	4276	42760	641400	566.36	542.62	91.1	5,2,1
10	4096	4	4276	40960	614400	542.52	520.40	95.8	87.2
11	3860	P	3860	42460	636900	562.38	538.82	90.4	1
12	3476	P	3476	41712	625680	552.48	529.95	88.8	6,4,3,2,1
13	3188	P	3188	41444	621660	548.93	526.55	88.3	1
13	3072	3	3188	39936	599040	528.95	507.99	96.4	85.1
14	2932	P	2932	41048	515720	543.68	521.51	87.4	7,2,1
15	2676	P	2676	40140	602100	531.65	510.58	85.5	5,3,1
16	2484	P	2484	39744	596160	526.41	505.54	84.6	5,4,2,1
17	2324	P	2324	39508	592620	523.28	503.13	84.1	1
18	2164	P	2164	38952	584280	515.92	496.05	83.0	9,6,3,2,1
18	2048	2	2164	36864	552960	488.26	470.57	94.6	78.5
31	1024	1	1044	31744	476160	420.45	407.12	98.1	67.6
46	512		532	23552	353280	311.95	304.98	96.2	50.2
74	128		148	9472	142080	125.46	124.32	86.5	20.2

SERVICE DESCRIPTION LANGUAGE

① SERVICE NAME

- identifies this description

② BUFFER SECTION

- specify number of buffers

③ AREA SECTION

- specify areas to be available
- delayed page locking (X)
- page reserve
- virtual store (X)

④ LOCK SECTION

- specify size of lock tables
- update not allowed
- timeout
- maximum concurrency

⑤ ACCESS CONTROL SECTION + Diary mandatory ?

- specify database procedures for:
- service start
 - application start
 - error detection
 - files in and out

⑥ JOURNAL SECTION (X)

- choice of:
- central journal
- area journals with QBLs
- duplexing

METHOD OF USING TRAVERSAL CONCEPT

For each record type:

- ① list the different traversal types for each operation pathway accessing the record**
- ② note the frequency with which each one occurs**
- ③ calculate the number of times the record is accessed per traversal type**
- ④ assemble list in descending order of volume (modified by any other priority factors)**
- ⑤ apply flowchart logic steps to traversal volume list**

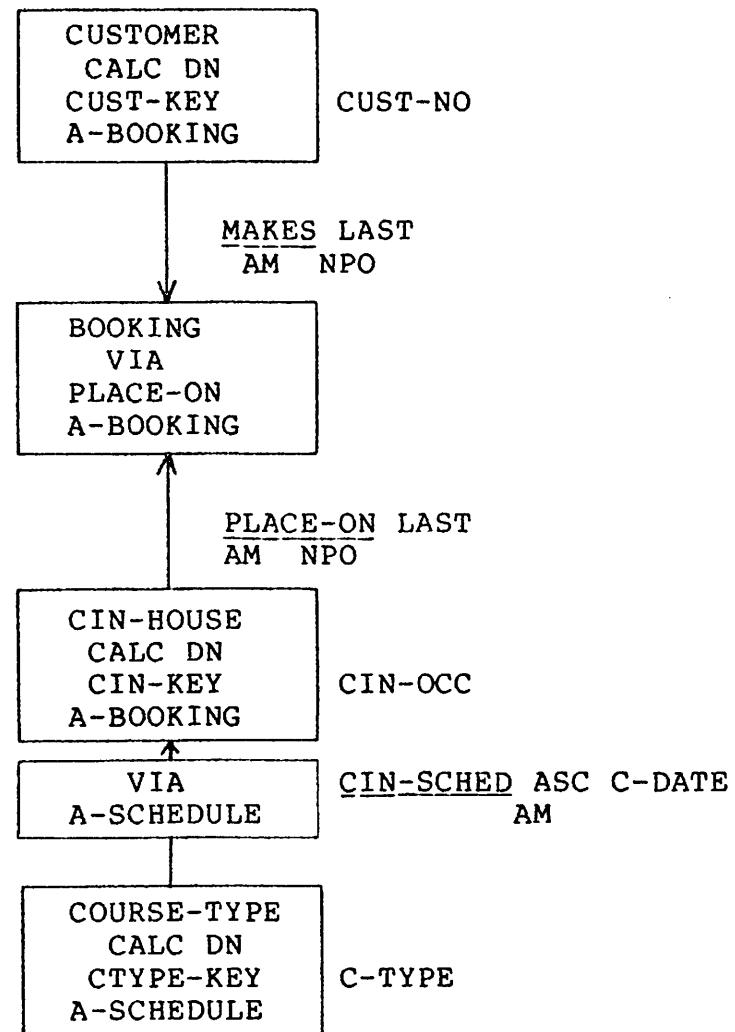
IDMSDUMP REPORT

ICL 2900 IDMS (C) COPYRIGHT CULLINANE 1975 AND ICL 1982.
SECURITY DUMP - DATABASE UTILISATION REPORT (DUMP TAKEN AT 82/12/21 15:45:18)
DUMPED 12 PAGES, FROM AREA AREA-1
MAXIMUM PAGE SIZE IS 1,024 CHARACTERS
SPACE USED 7,072 CHARACTERS
EMPTY SPACE LEFT 4,736 CHARACTERS
FILE UTILISATION IS 60 PERCENT

PAGE 1

RECORD TYPE	SPACE DISTRIBUTION PER RECORD			LOGICALLY DELETED		
	OCCURRENCES	TOTAL SPACE USED	PERCENT OF TOTAL USED	OCCURRENCES	TOTAL SPACE USED	PERCENT OF TOTAL USED
SR2	1	16	0	0	0	0
SR3	1	144	2	0	0	0
SR201	1	20	0	0	0	0
SR206	23	3,312	47	0	0	0
SR207	6	936	13	0	0	0
SR209	24	1,440	20	0	0	0
SR1201	1	220	3	0	0	0
SPACE INV	1	984	14			

SOLUTION EXERCISE 45



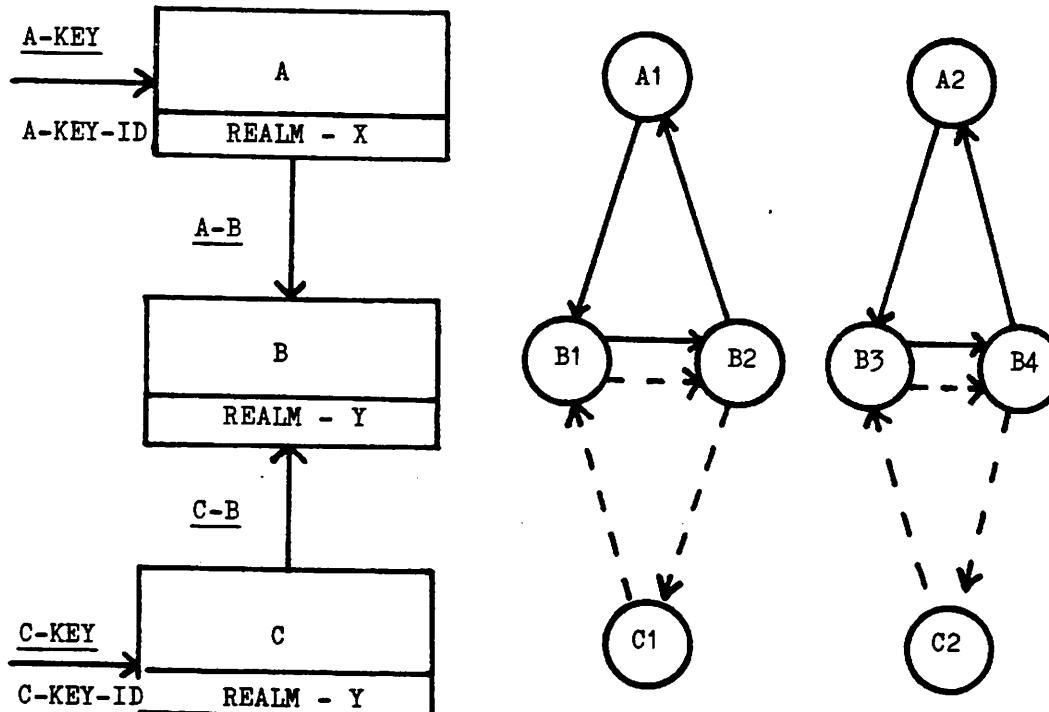
DML NAVIGATION EXERCISE 50

1. Given the data structures and record occurrences shown below, list the records retrieved by each GET in the DML sequence:

```

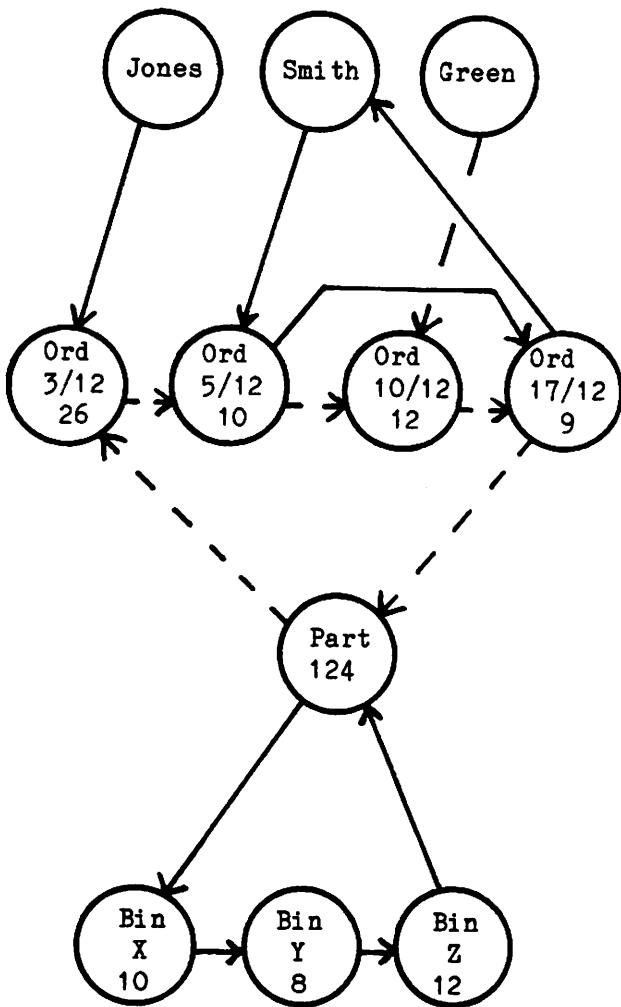
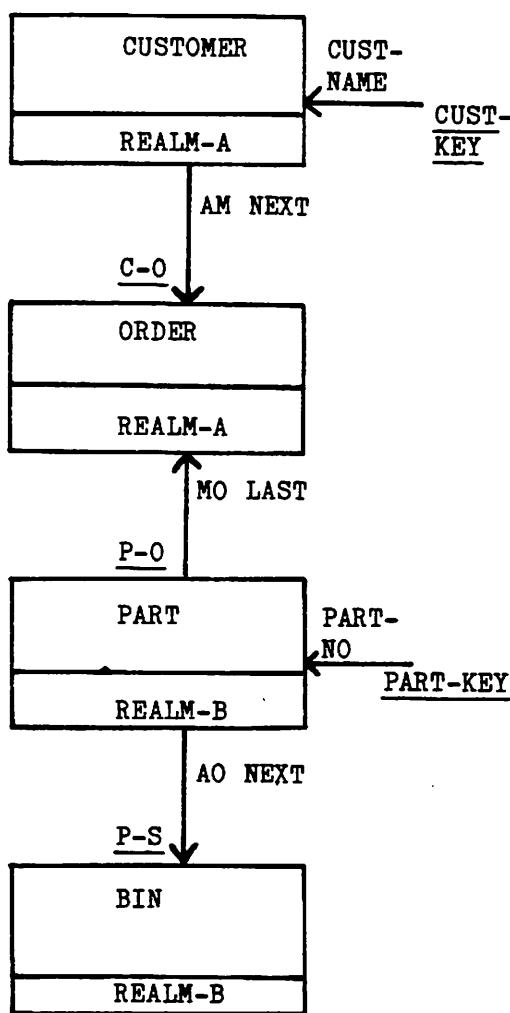
MOVE "A1" TO A-KEY
FIND ANY A USING A-KEY
GET A
FIND NEXT B WITHIN A-B
GET B
MOVE "C2" TO C-KEY
FIND ANY C USING C-KEY
GET C
FIND NEXT B WITHIN C-B
GET B
FIND NEXT B WITHIN A-B
GET B

```



2. Write the sequence of logic (showing DML functions in full) which, given a part number, will:

- establish available stock for part no 124 and then
- discover those customers whose orders for part 124 cannot be met in full (stock is allocated to orders in date sequence).



SOLUTION 50

1. Records retrieved:

A1 B1 C2 B3 B4

The currency setting are shown below.

CURRENCY TABLE

DML VERB	CRU	CURRENT A-B	CURRENT C-B
FIND ANY A	A1	A1	NULL
FIND NEXT B WITHIN A-B	B1	B1	B1
FIND ANY C	C2	B1	C2
FIND NEXT B WITHIN C-B	B3	B3	B3
FIND NEXT B WITHIN A-B	B4	B4	B4

2. MOVE "124" TO PART-NO.
FIND ANY PART USING PART-KEY.

IF REC-NOT-FOUND

MOVE ZERO TO TOT-QTY.

→ OBTAIN NEXT BIN WITHIN P-S.

IF END-OF-SET

ADD B-QTY TO TOT-QTY

→ OBTAIN NEXT ORDER WITHIN P-O

IF END-OF-SET

SUBTRACT O-QTY FROM TOT-QTY

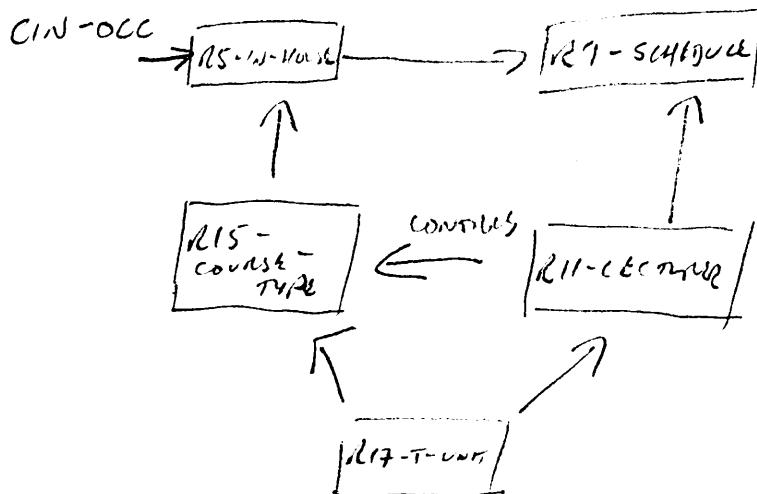
IF TOT-QTY NEGATIVE

→ OBTAIN OWNER WITHIN C-O
DISPLAY CUST-NAME.

MORE DML NAVIGATION EXERCISE 60

Referring to the Schema from Exercise 40, outline the DML which will support Enquiry 3. Use occurrence diagrams to clarify the access paths which would show that VIDMD.123 will be given by D Jukes only. The Course Controller is M Whitmarsh. Both lecturers work for the Systems Training Unit.

for course show lecturer, training unit & controller



MOVE "VIDMD.123" TO CIN-OCC

FIND ANY RS USING CIN-OCC

FIND FIRST RA WITHIN S5-GIVEN BY

SCHED-LOOP

OBTAIN OWNER WITHIN S11-CONTROLLER

DISPLAY LECTURER

FIND NEXT RA WITHIN S5-GIVEN BY

ON DB-END-OF-SET

GOTO END-OF-SCHED .

GOTO SCHED-LOOP

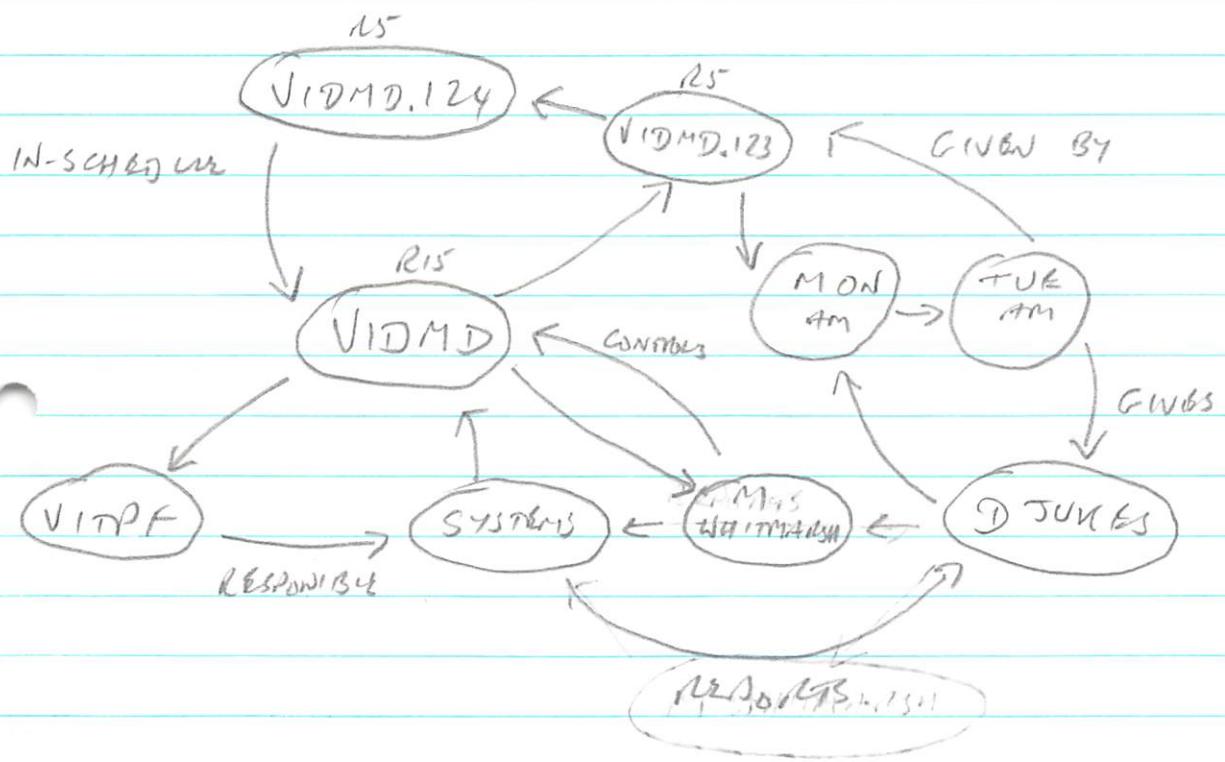
END-OF-SCHED OBTAIN OWNER WITHIN S15-SCHEDULE

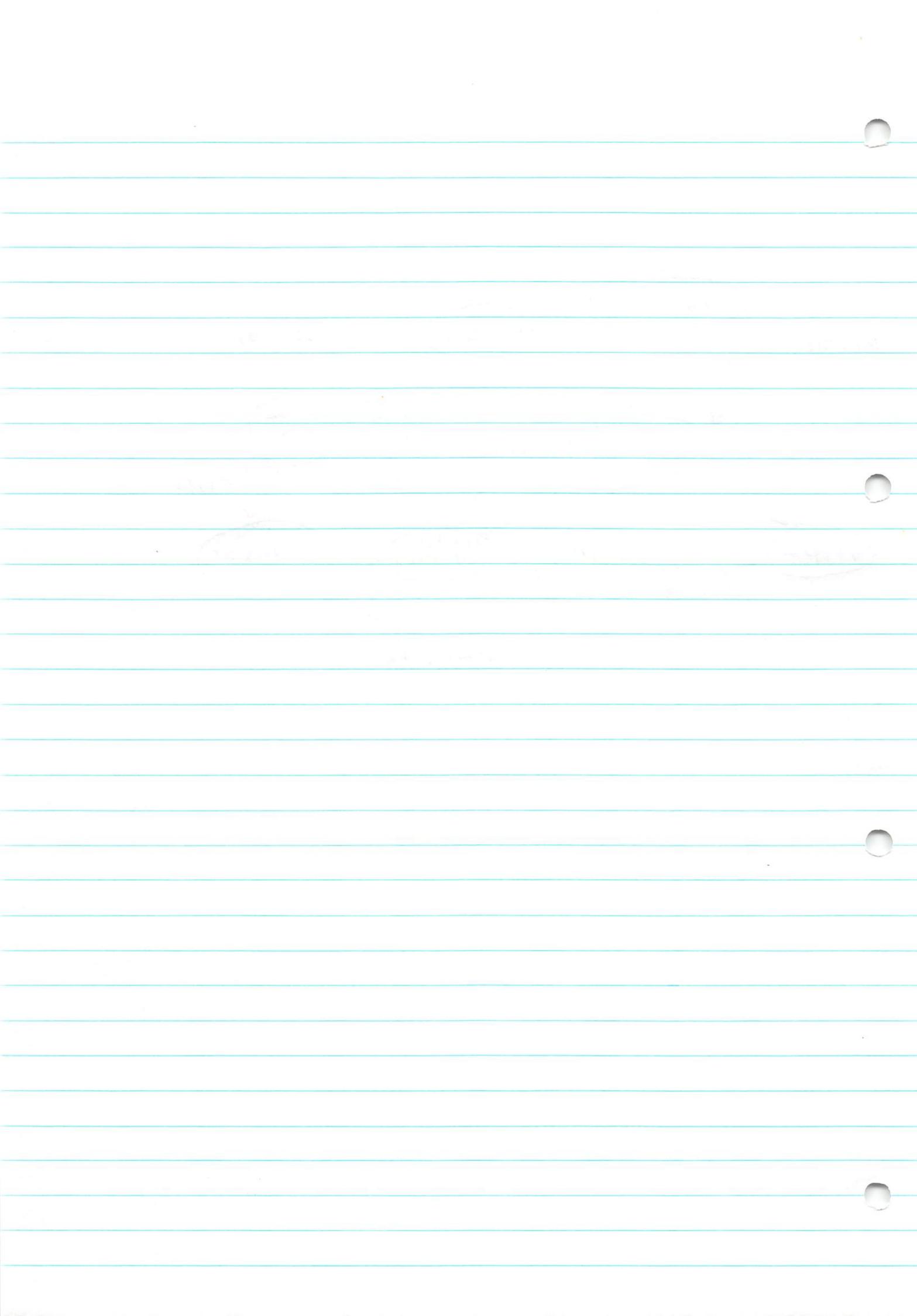
OBTAIN OWNER WITHIN S11-CONTROLLER

DISPLAY CONTROLLER

OBTAIN OWNER WITHIN S17-RESPONSIBILITY

DISPLAY TRAINING-UNIT



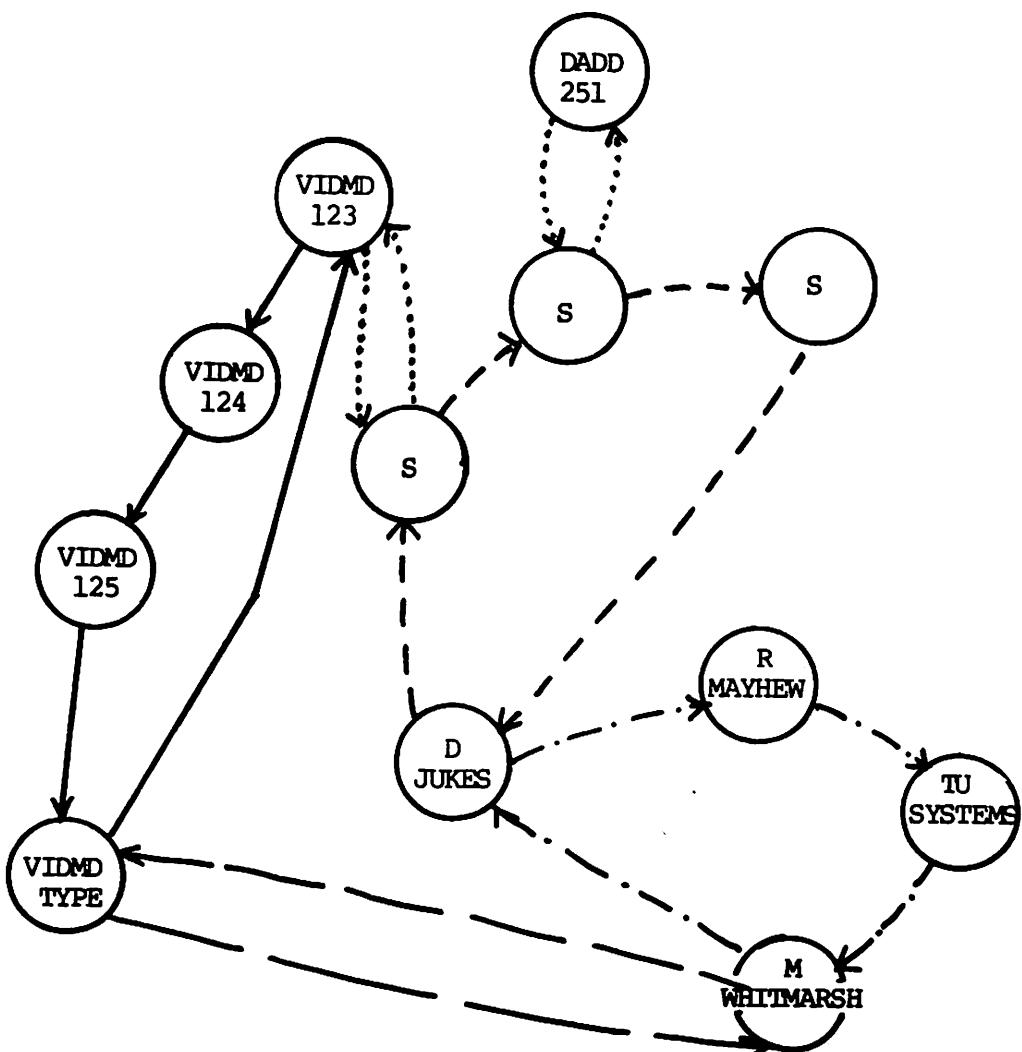


SOLUTION 60

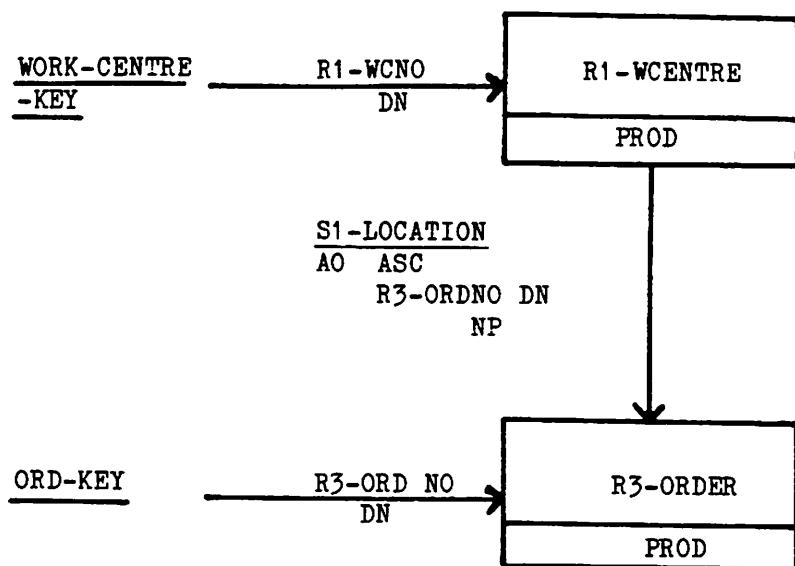
```

MOVE COURSE-OCC TO CIN-OCC
FIND ANY R5-IN-HOUSE USING CIN-KEY
IF DB-REC-NOT-FOUND
    GO TO →
    OBTAIN OWNER WITHIN S15-IN-SCHEDULE
    OBTAIN OWNER WITHIN S11-CONTROLS
Note. Course Controller found.
L-NEXT
    FIND NEXT R9-SCHEDULE WITHIN S5-GIVEN-BY
    IF DB-END-OF-SET
        GO TO ←
        OBTAIN OWNER WITHIN S11-GIVES
Note. Lecturer found.
        OBTAIN OWNER WITHIN S17-REPORTS
Note. Training Unit found.
        GO TO L-NEXT

```



SUCCESS UNITS EXERCISE 70



Currently the computer department run the 'Order Location' sub-system once a week in batch mode. Run time averages about 45 minutes.

No thought was given to the Success Unit concept in designing the program structure as the attached flow chart shows (Figure 1).

Clarify what could be wrong with this design from the points of view of:

- Recovery
- Locking
- Database consistency.

Draw a simple flow chart showing an approach which corrects the errors you have found.

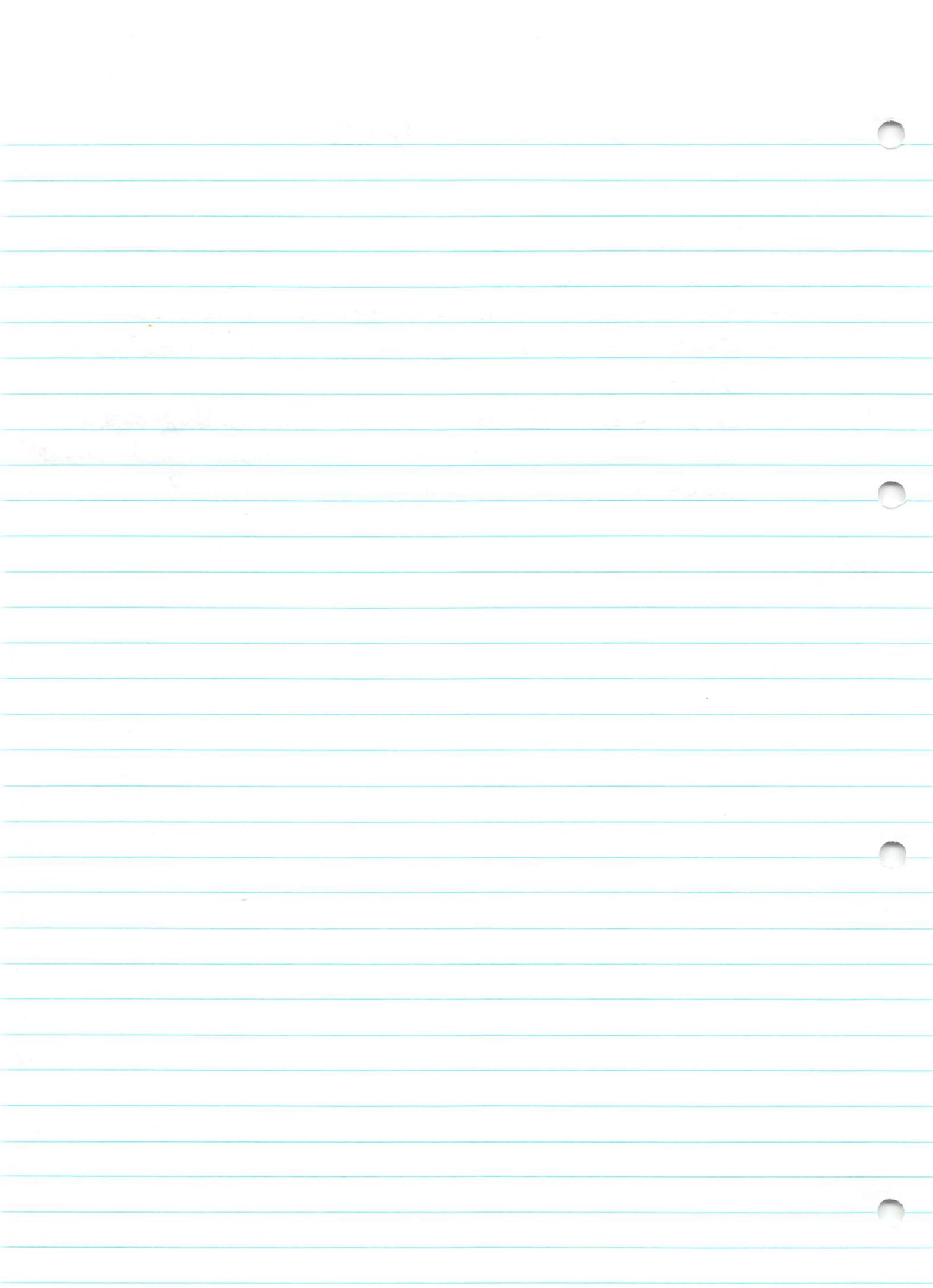
Ex 60

Recovery

- Could commit after each file rec has been processed freeing locks.
- Could use IDMS + RECON checkpointing
- Check for existence of New Work Centre at start
- Sort, take OrderNumber matching WorkCentre
- If Missing off D13 just reads next record what happens to open file recs??

Locking

Consistency



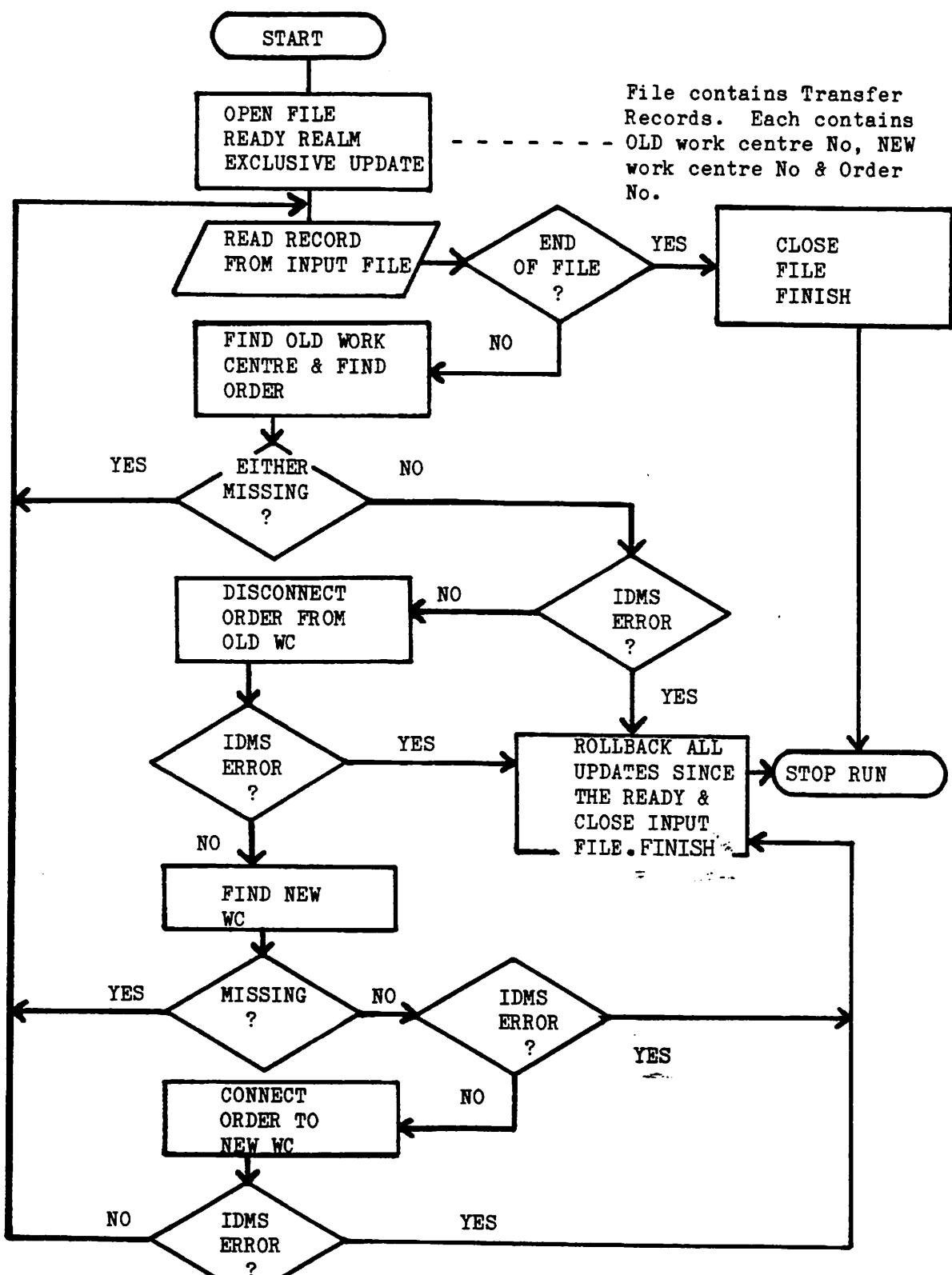


Figure 1 Flowchart of old System's Program Logic

SOLUTION 70

The original program design meant that the WHOLE RUN was one Success Unit. As a result:

1. Recovery implications

Any failure at any point in the 45 minutes of the run means that all updates since the start of the run would be reversed. After the problem had been rectified, the whole run would then have to be started from the beginning again.

2. Locking implications

EXCLUSIVE UPDATE means that any other user would have to wait until the run was complete (up to 45 minutes) or until Recovery was completed if the run failed.

3. Database consistency

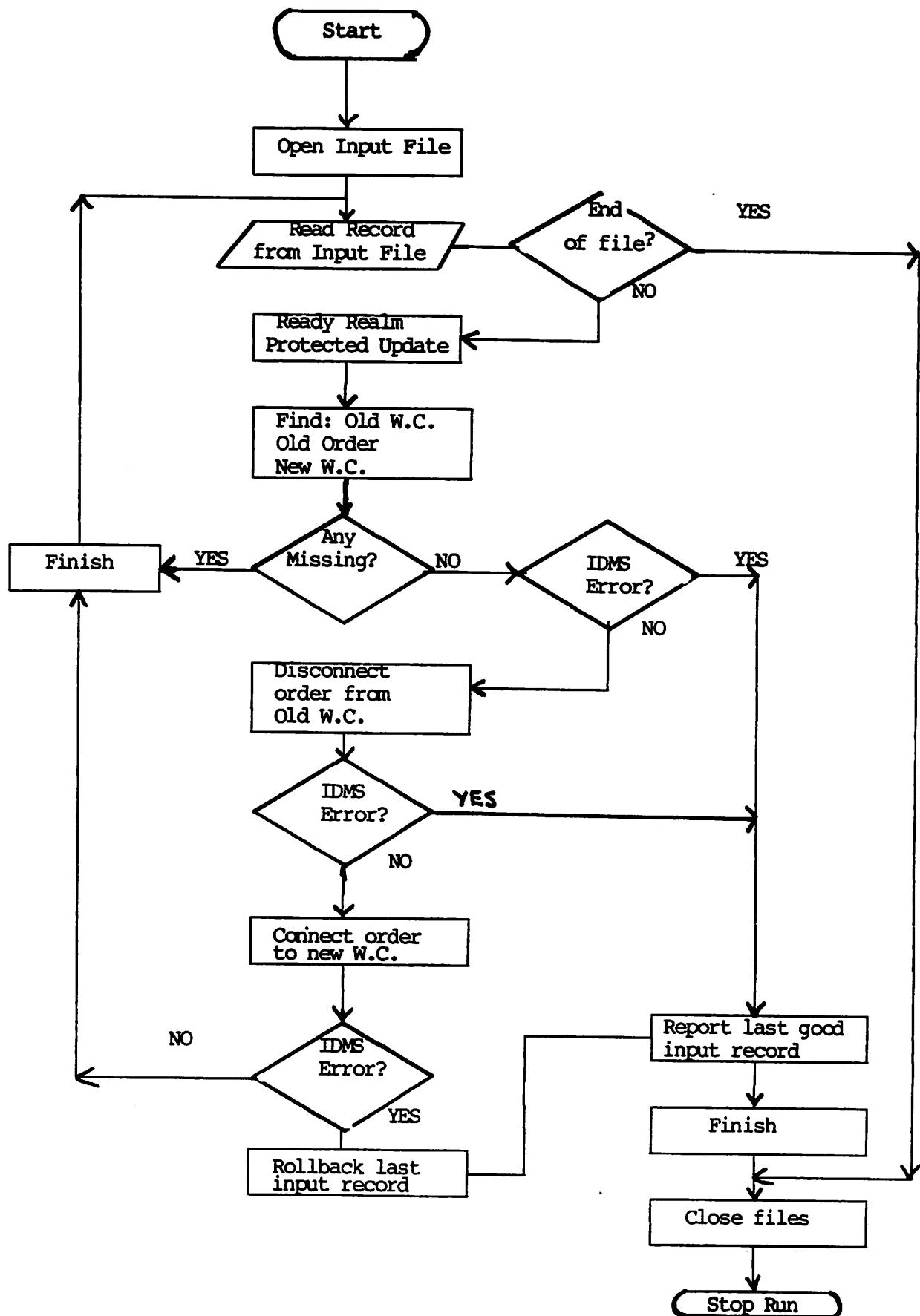
The way the Logic is written means that if an Order is disconnected but then the "new Work Centre Number" turned out to be incorrect, the run is continued with the Order hanging in limbo. In fact this design means that the only hope for ensuring Database Consistency on this point, is some subsequent failure which will cause all updates to be reversed, including this one!

The attached flowchart shows an approach where:

- a) each Input Transaction is a separate Success Unit
- b) the Realm is opened for Page Locking
- c) an Order is not disconnected until both Work Centre Numbers are known to be correct.

As a result, any failure causes the loss of only the current transaction and the results of previous successful transactions are preserved. The Reporting of "last successful Transaction" plus some re-start logic (not shown for simplicity) would allow the run to continue at the appropriate point after the failure had been corrected.

In addition, the Database Consistency is improved and other users are either not delayed at all, or just for the duration of a Transaction.



IDMS DESIGN EXERCISE 80

A TV rental company is implementing an IDMS database system which is intended to improve the Fault Repair service offered to customers. The essence of success in this system is to provide fast response to customer's Fault reporting.

Data analysis has produced the detailed information which follows, along with the soft-box model shown in Figure 1. The initial design has produced the Schema and Storage diagrams (Figures 2 and 3).

Examine the information and then consider:

1. What factors in the proposed design will produce contention, ie several transactions competing for the same data at the same time? Consider:
 - a) the use of areas
 - b) the different requirements of retrieval and update processing
 - c) the run time interactions of the different types of transactions.

Suggest ways in which the resulting problems may be avoided. Your tuning suggestions should be designed to give transaction type 1 the highest priority.

2. How could the success unit concept be used to structure transaction 1?
3. Draw a subschema diagram to support the processing involved in transaction 6.

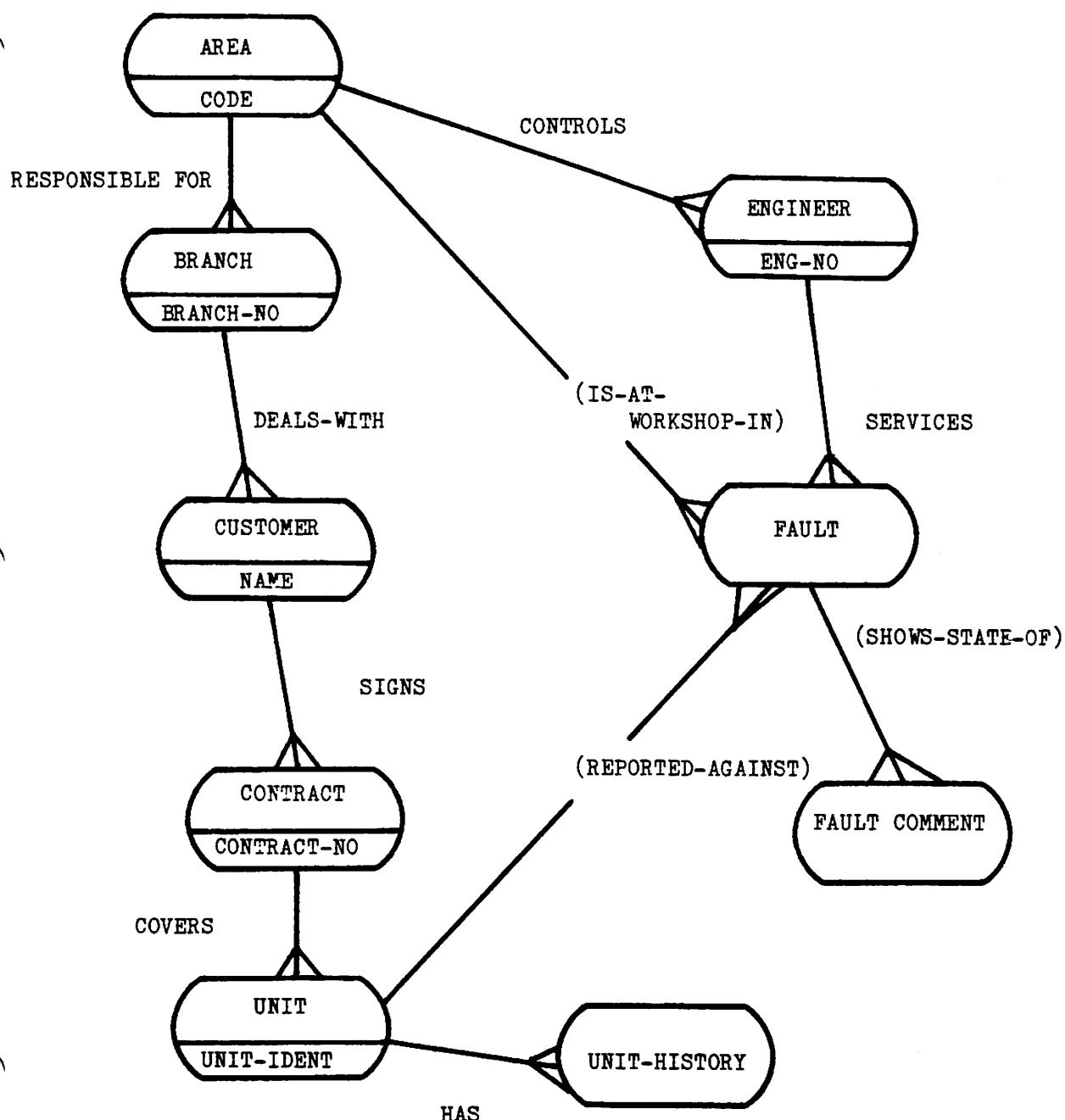


Figure 1 Soft-box model of Rental System

Entity descriptions

- AREA - A geographical area controlling a number of Branches and Engineers.
- BRANCH - that part of the business which deals directly with Customers.
- ENGINEER - reports to an Area Office and is responsible for servicing Customers' TV faults in that area. If the TV in question cannot be repaired by him at the Customer's premises then he can send the set to the area workshop. Repairs complete, the Fault is cleared.
- CUSTOMER - some person (or organisation) who rents TV sets.
- CONTRACT - a signed agreement between the Customer and the business for rental of one or more TVs. While most customers have one contract for one TV, a few (eg hotel chains) may have many contracts each for many TVs.
- UNIT - uniquely identified by Unit number, is a particular TV set rented out to a Customer.
- FAULT - occurs on a set, and is recorded against the appropriate Unit. After a fault has been cleared by an Engineer, the Fault should be summarised as Unit History and the Fault together with any Fault Comments should be erased.
- FAULT COMMENT - used to trace the progress of a FAULT. It can be an Engineer's comment on the FAULT or a further report from a customer before the FAULT was cleared.
- UNIT HISTORY - detail of a specific problem on a TV set together with the remedial action taken by the Engineer.

Six transactions are described below. Of these the first "Customer Fault Reporting" is of critical importance and a fast response is vital.

1. Customer Fault Reporting

The customer will report:

- His Name and the Unit Number of the Faulty Set. (The latter is stamped on the back of the set and no fault report will be accepted unless this number is quoted by the Customer.)
- A simple description of the fault.

The processing logic for the transaction must then:

- a) Check that the customer exists. *key*
- b) Check that the Unit exists. *key*
- c) Check that there is a valid contract between that Customer and that unit. *FIND OWNERS*
- d) If all these checks succeed, then check whether the fault has been reported already. If so, record the repeated complaint as a Fault Comment. If not, record the Fault details in the database. *RECORD SET*

2. Customer signs a contract

Check customer exists with acceptable credit rating and input details of the contract and the Unit(s) if OK.

3. Schedule Faults

Examine outstanding Faults and assign to Engineers. This is done at hourly intervals (300 per hour) between 9am and 3pm. Faults reported after 3 pm are assigned overnight and repaired the next day. *AREA OWNERS TO MINS WORK*

4. Engineer collects fault information

He collects details of the Faults assigned to him together with Fault Comments and addresses (from the contract).

5. Engineer reports on a visit to a Customer

The report will specify one of the following:

- a) Fault cleared
- b) Set taken to workshop
- c) Set collected from workshop and returned to customer.
- d) Fault comment, eg set repairable on site but required part not available at that time.

6. Archive cleared Faults

BATCH

Summarise cleared Faults as Unit Histories and erase the Faults together with any associated Fault Comments.

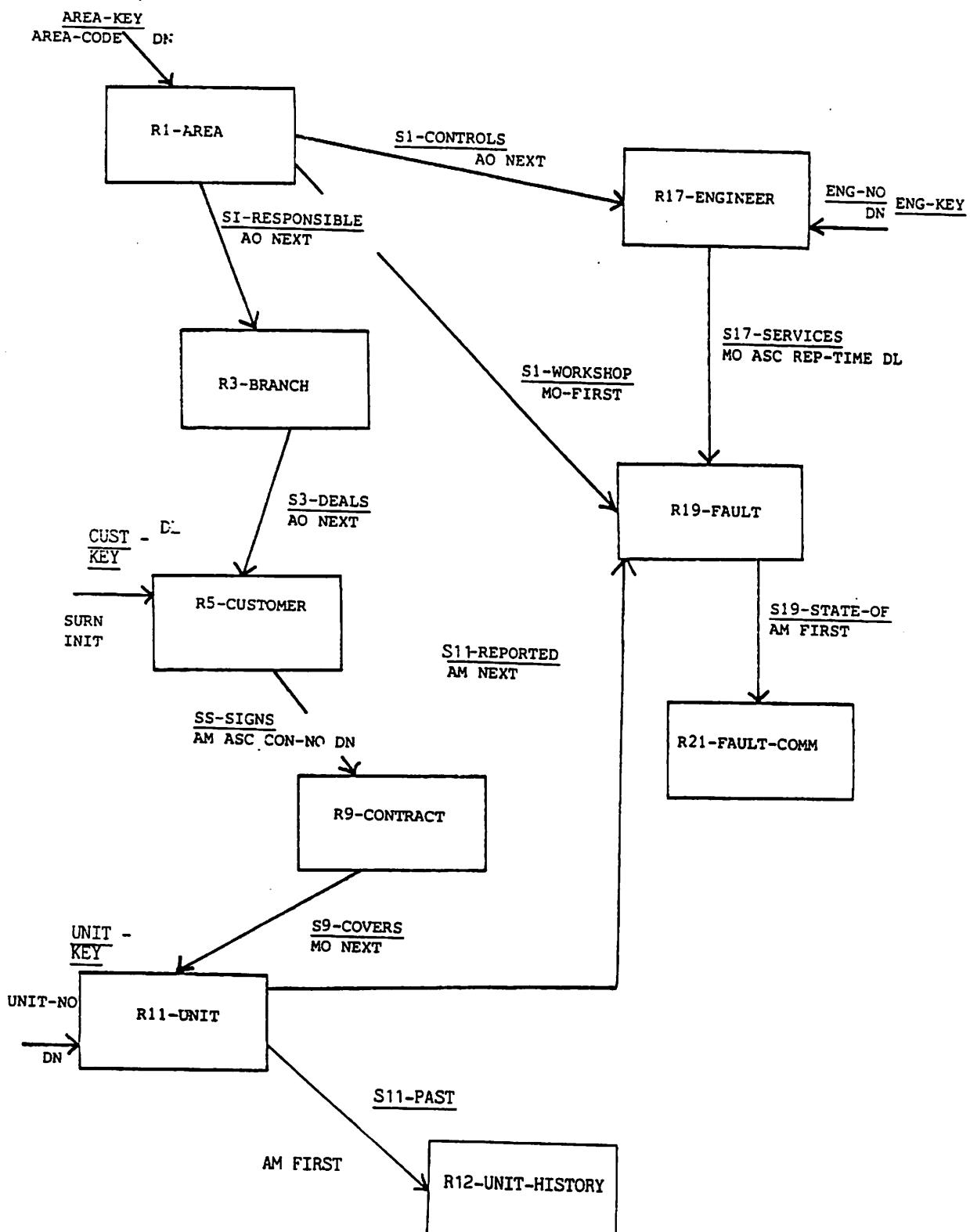


FIGURE 2 - TV RENTAL - SCHEMA DIAGRAM

NO267

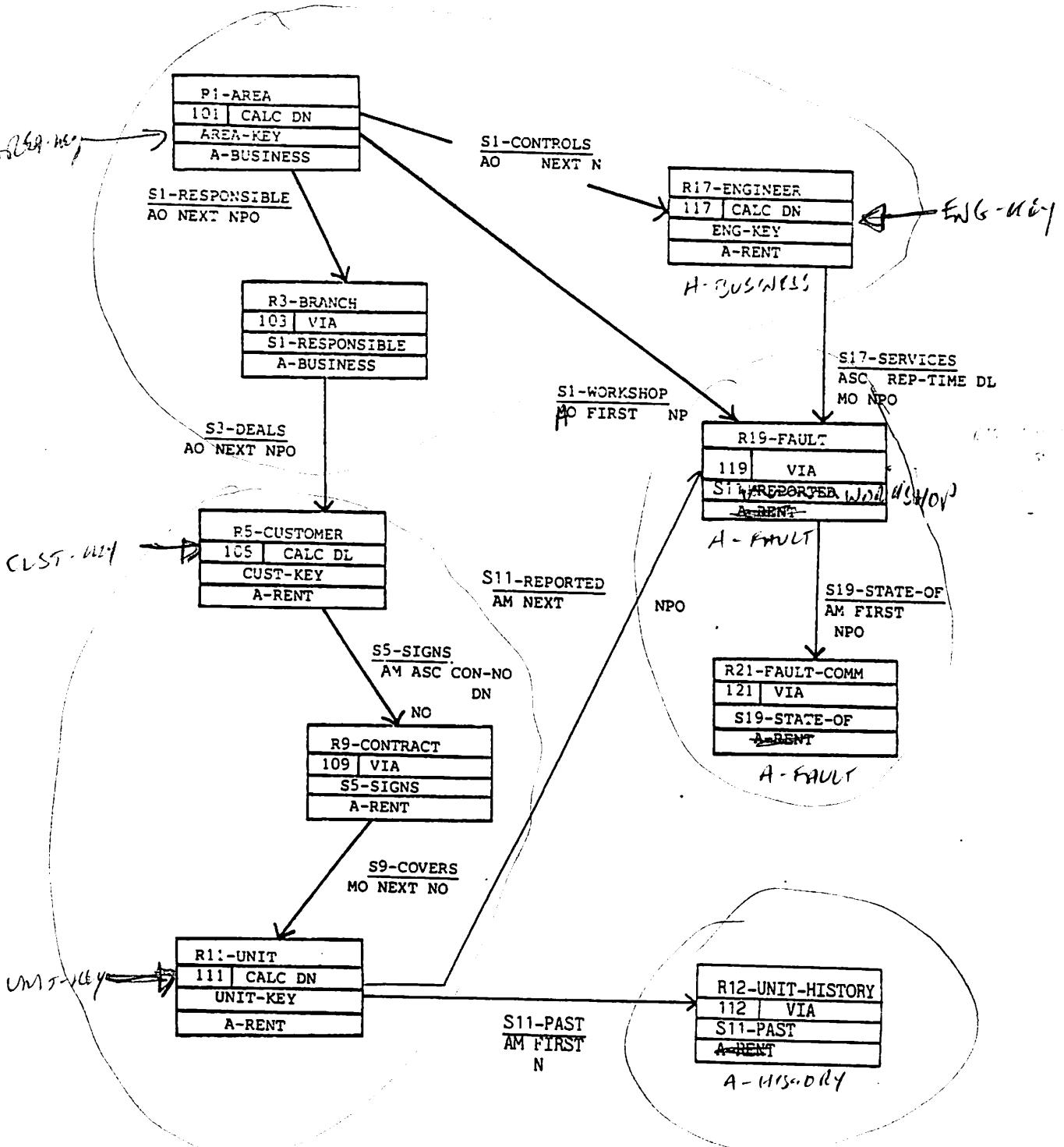


Figure 3 TV Rental - Data Storage Diagram

SOLUTION 80

1. Reducing Contention

As designed, the Records are all in one large Area (except for the rarely used R1-AREA and R3-BRANCH records). All the transaction types update except transaction 4 ("Engineer Collects Fault information") so even with Page Locking a degree of delay or even deadlock may occur.

In general, a larger number of smaller Areas is better than one or two big ones. The smaller physical units can be shared by more simultaneous transactions without contention occurring and Recovery procedures like dumping will be faster. However, achieving such smaller Areas and using them effectively may require careful examination of how the various Record types are processed together with some ingenuity. In this case for example, the following techniques could be used:

- a) Consider the mix of transaction types between on-line and batch use and between day and night use. In fact transaction type 2 ("create new contract") and (especially) transaction type 6 ("Archive Cleared Faults") can be done batch overnight. This means that the processing requirements of those transactions need not be given much weight when allocating Record types to Areas. In addition there will be no Locks from those transaction types interfering with the on-line work.
- b) Consider whether one or more read-only Areas can be created so that any mix of on-line transactions can share their use at the same time. This is possible with an Area comprising R3-BRANCH, R5-CUSTOMER, R9-CONTRACT, R11-UNIT and R12-UNIT-HISTORY Records providing some way can be found to link R11-UNIT Records to R19-FAULT Records without using set pointers. If a "Soft-Pointer" is used in R19-FAULT by making the Record CALC on UNIT-NUMBER, then the link can be made without having to change anything in the R11-UNIT Record. Duplicates of the Key could occur if several different faults were reported on the same set. Hence the CLAC Key could be, say, DUPLICATES LAST to keep the reports in date sequence.
- c) Ensure that all transaction types use as economic a navigation strategy as possible. For example, transaction type 3 ("Scheduling Faults to Engineers") seems to present some problems in finding out just which Faults are unallocated at present. An Area sweep could be very slow and could set too many locks. One solution would be to connect new faults to an UNALLOCATED Set linking them to the relevant R1-AREA Record as they were created. When allocation occurs, only occurrences of that Set need be searched, each R19-FAULT Record being disconnected from that Set as it was connected to an R17-ENGINEER Record.

Figure 1 shows a Storage Diagram which implements these ideas. Notice that the updating part of the data structure has been split into TWO Areas. This has the advantage that FAULT COMMENT Records will not be kept out of the Page containing their FAULT record by intruding ENGINEER or AREA Records. It has the disadvantage of set connections crossing Area boundaries which complicates re-organisations.

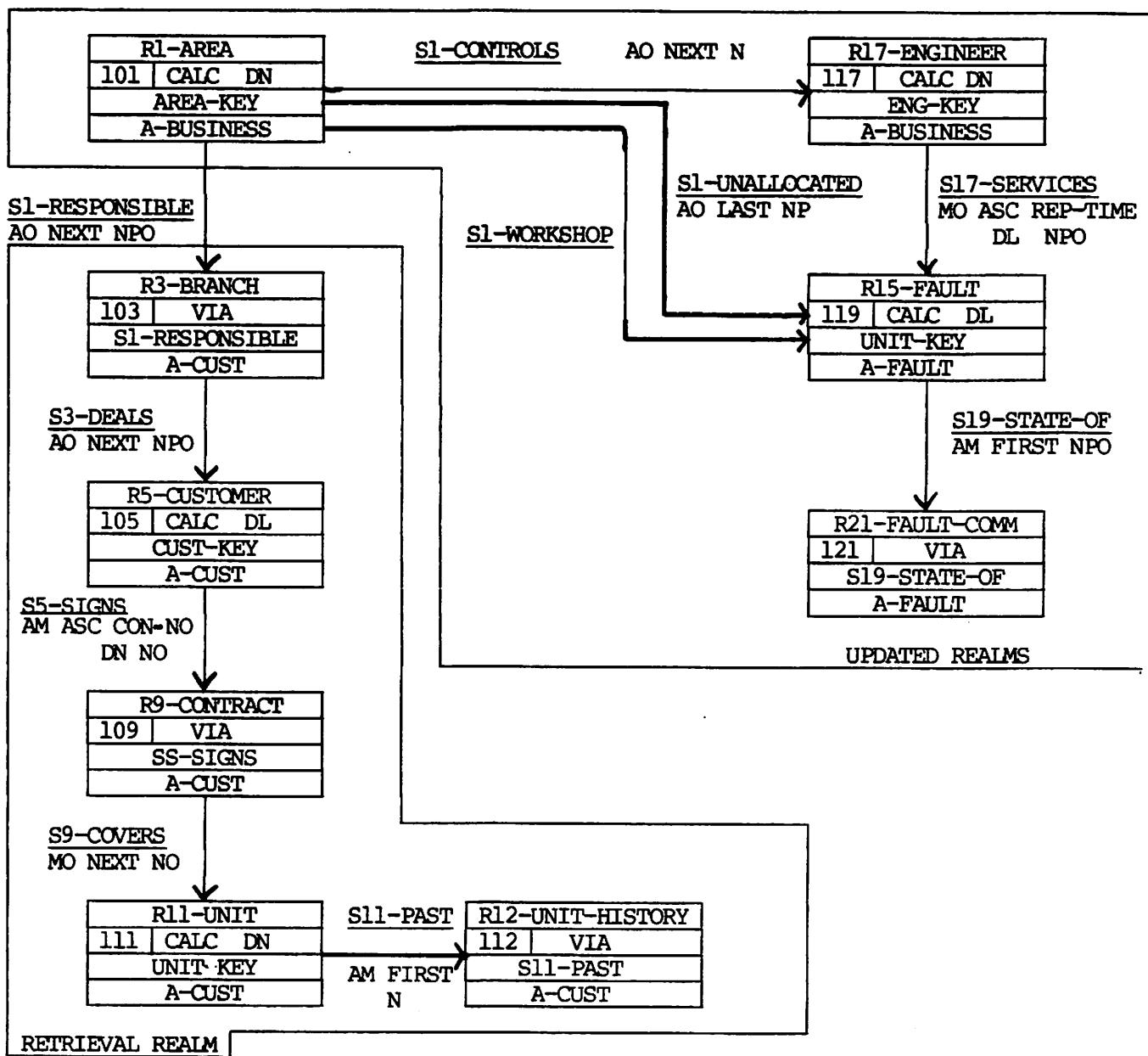


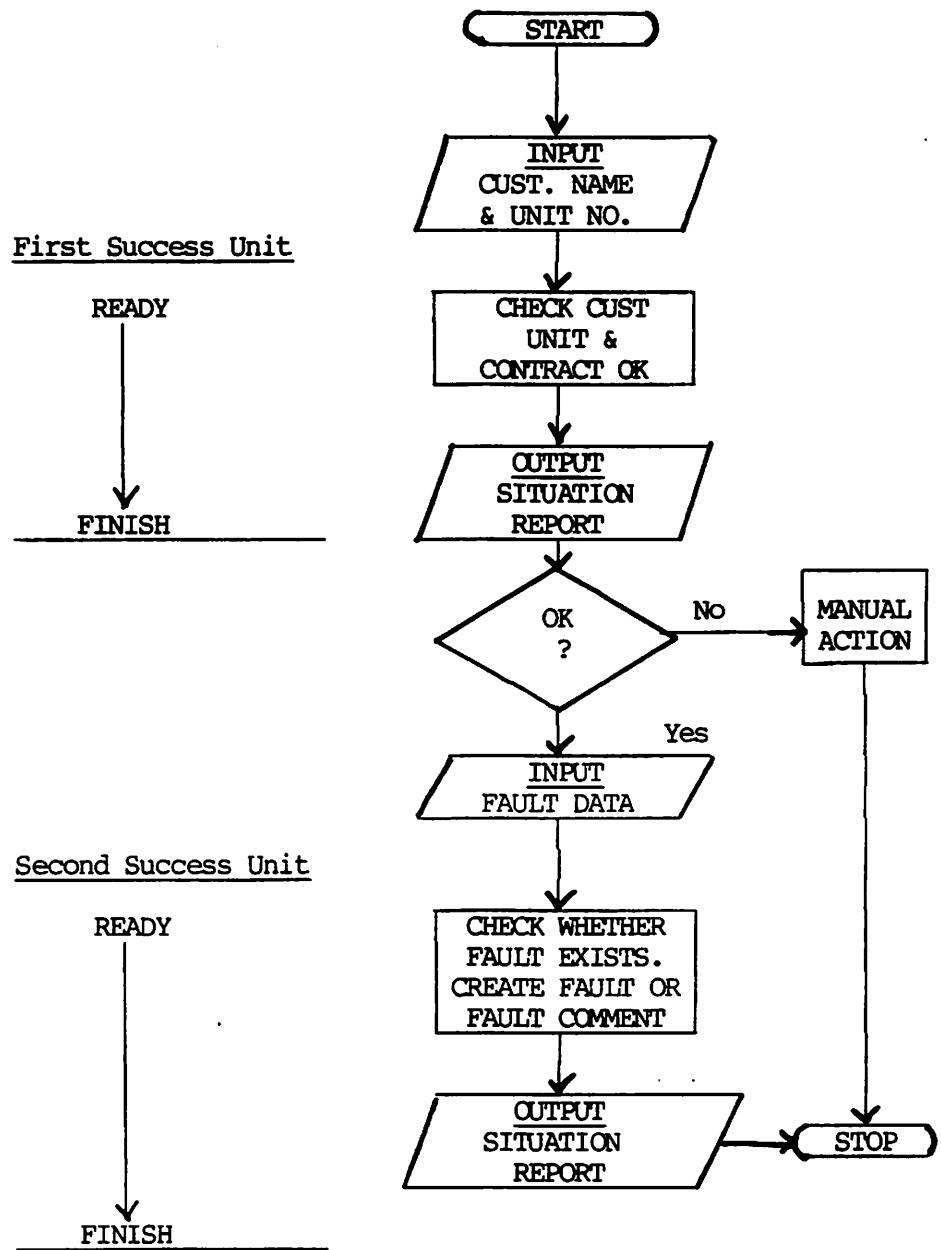
Figure 1 TV Rental System - Tuned Storage Diagram

2. Success Unit Structure

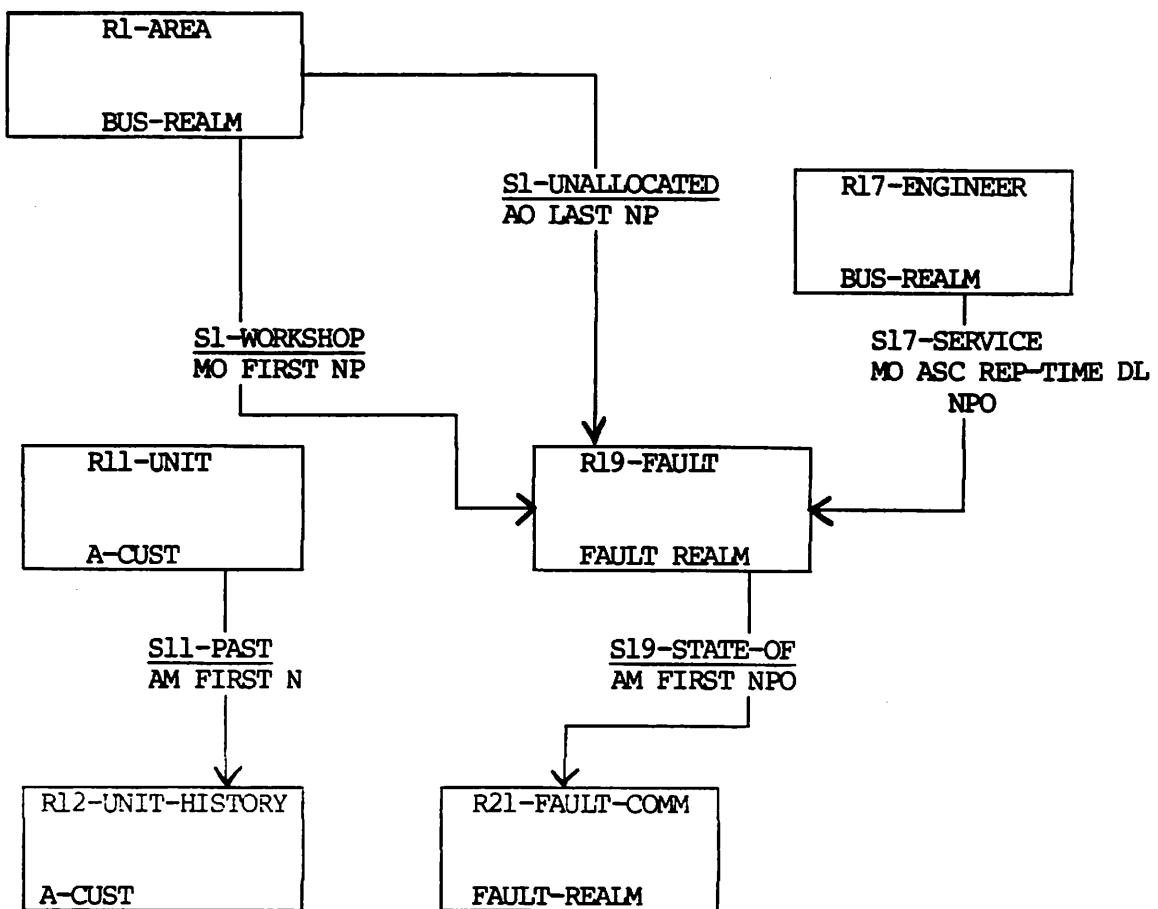
Transaction type 1 could be done as a single message pair, ie as one Success Unit. However, it is assumed in the flow chart below that it is more convenient for the terminal operator to structure the work into two message pairs:

- a) The major checking processes, eg does the Unit exist?
- b) The input of update information.

This has two advantages. First, the update data need only be input if it is actually required. Second, if the checks fail, further dialogue with the customer may allow the problem to be corrected in flight and the transaction to be completed. However this approach does increase the message traffic since it is likely that the second message pair will usually occur.



3. Subschema Diagram to support transaction 6 ("Archive Faults")



- Realms **BUS-REALM**, **FAULT-REALM** and **A-CUST** to be readied in update mode.
- **FAULT** and **FAULT-COMM** records will be physically erased, no logical deletions will occur.
- **UNIT HISTORYs** will be created.
- The 'cleared' **FAULTS** to be erased may be found by:
 - . an area scan checking for **FAULT** records with a 'cleared' indicator
 - . alternatively, the design could include a super-chief record to own all cleared **FAULTS**. Transaction 6 would only need to walk this set, in this case.

AREA SIZING EXERCISE 90

See 9-20

We now wish to assess the storage diagram used in Exercise 30 to ensure that performance specifications are met. The first stage of the assessment is to estimate the disc space needed.

The page size is to be 3100 bytes and all areas are to be packed at 70%.

- ① Calculate the size of the A-PERSONAL area and the A-INDEX area.

Record details are:

$L + \text{prefix} + \text{Line Index Entry}$

<u>Record Type</u>	<u>Data Length</u>	<u>Number Occurrences</u>
R5-APPOINTMENT	60	40,000
R11-EMPLOYEE	116	20,000
R15-SICKNESS	56	1 per Employee
EMP-KEY-NAME index	3024 (max. length as defined in Storage Schema)	Note. EMP-KEY-NAME's Logical key = 28 bytes

Note. The maximum index record length includes the 16 byte prefix. A 60% packing density should be used when estimating the number of entries per index record.

$$N = L_1 O_1 + L_2 O_2 + \dots + L_n O_n + 8(O_1, O_2, \dots, O_n)$$

P-40

$$\times \frac{100}{D}$$

$$= 4034 \text{ pages}$$

$$\text{Space M needed} = \frac{2 \times 4034}{3100 - 40}$$

$$\times \frac{100}{2} \\ \frac{2}{8068}$$

$$= \frac{8068}{3060}$$

$$= 3 \text{ pages}$$

E-22

NO267

SOLUTION 90**A-PERSONAL AREA**

Record	Pointers	Data Length	Total	No. of occs.	Total Volume
R5-APPOINTMENT	40	60	100	40,000	4,000,000
R11-EMPLOYEE	24	116	140	20,000	2,800,000
R15-SICKNESS	4	56	60	20,000	1,200,000

8,000,000

$$\frac{8,000,000 + (8 \times 80,000)}{(3100 - 40)} = 2824 \text{ Pages} \\ (\text{rounded up})$$

Plus 70% Packing Density

$$2824 \times \frac{100}{70} = 4035 \text{ Pages} \\ (\text{rounded up})$$

Plus Space Management Pages

$$\frac{(4035 \times 2)}{(3100 - 40)} = 3 \text{ Pages} \\ (\text{rounded up})$$

= 4038 Pages

A-INDEX AREA

Length of each index entry	= 28 + 4 (for db key)
	= 32 bytes
No. of entries to be covered	= 20,000
Size of all the entries	= 20,000 X 32
	= 640,000 bytes
Index record size	= 3024 bytes
Index packing density	= 60%
Total byte requirement	= $\frac{640,000 * (3024 + 20)}{.6 * (3024 - 16)}$
	= 1,079,433 bytes
Area packing density	= 70%
Total no. pages required	= $\frac{1,079,433}{.7 * (3100 - 40)}$
	= <u>504 pages</u>

Note

This calculation uses the standard index sizing formula:

$$\frac{A * (M + 20)}{P * (M - 16)}$$

(refer to course handout Chapter 9)

A tidy index has been assumed, so no allowance has been made for higher level index records or space management pages, as their impact will be insignificant.

SOLUTION 90 INDEX AREA SIZING - Alternative solution

Assuming that the record index is to cover:

20000 records
28 byte logical key
60% index record packing density
3100 page size of area to hold the index
70% index area packing density

Assuming one index record per page

each index record will have

3100 - 40 - 8 - 12 - 16 bytes available for index entries

= page - header & - line index - pointers - red tape
size trailer entry

= 3024 bytes available for index entries

each index entry occupies

28 + 4 bytes

= logical key + database key

= 32 bytes

each index record may hold a maximum of

$\frac{3024}{32}$ entries

= 94 entries (rounded down)

assuming a 60% index record packing density, each index record will only be used to hold

$94 * \frac{60}{100}$ entries

= 56 entries (rounded down)

The index will have a total of 20000 entries (one for each record)

so 20000 index records are needed
 56

= 358 index records (rounded up)

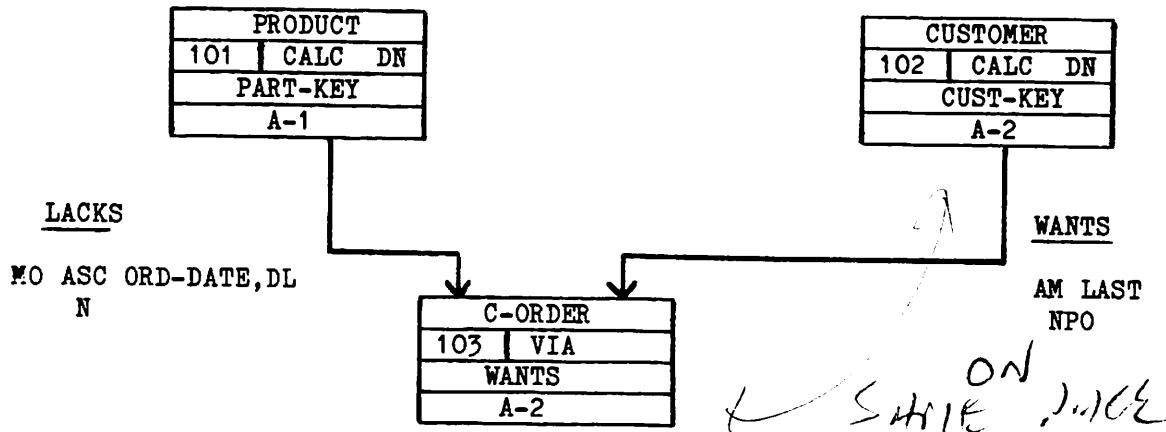
this requires 358 pages assuming the area is completely full. With a 70% area packing density we need

358 * 100 pages
 70

= 512 pages (rounded up)

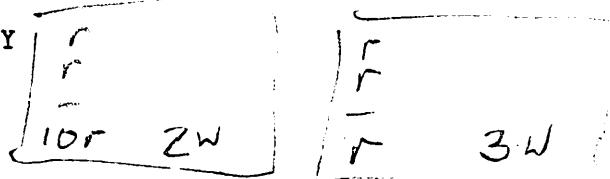
This calculation assumes that the impact of space management pages and higher levels of index are insignificant.

DISC ACCESSES EXERCISE 100



Skeleton DML to create a new Order might be:

OBTAINT ANY CUSTOMER USING CUST-KEY
 OBTAIN ANY PRODUCT USING PART-KEY
 STORE C-ORDER
 CONNECT C-ORDER TO LACKS



Given the data structure above and the skeleton DML to process it, estimate:

1. The number of disc transfers (pages), both for reading and writing. (Assume that Orders are added in date sequence and that there are enough Page Buffers to avoid thrashing.)
2. The disc transfers when the LACKS set design options have been changed to:

set order = LAST
 pointers = NP

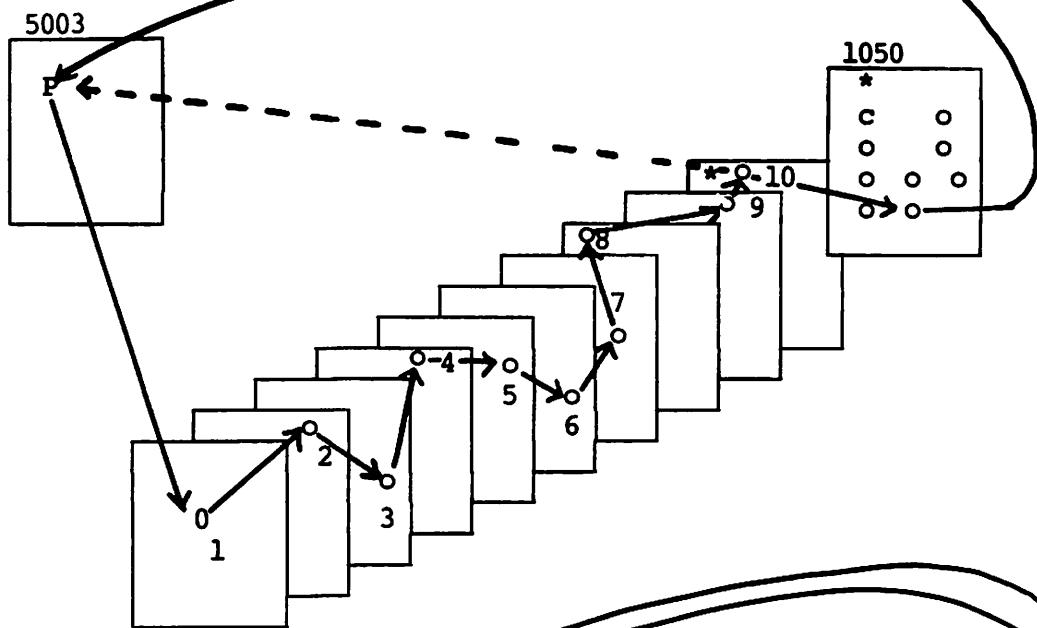
only read lost in set.

Note. Assume an average of 10 orders per LACKS set.

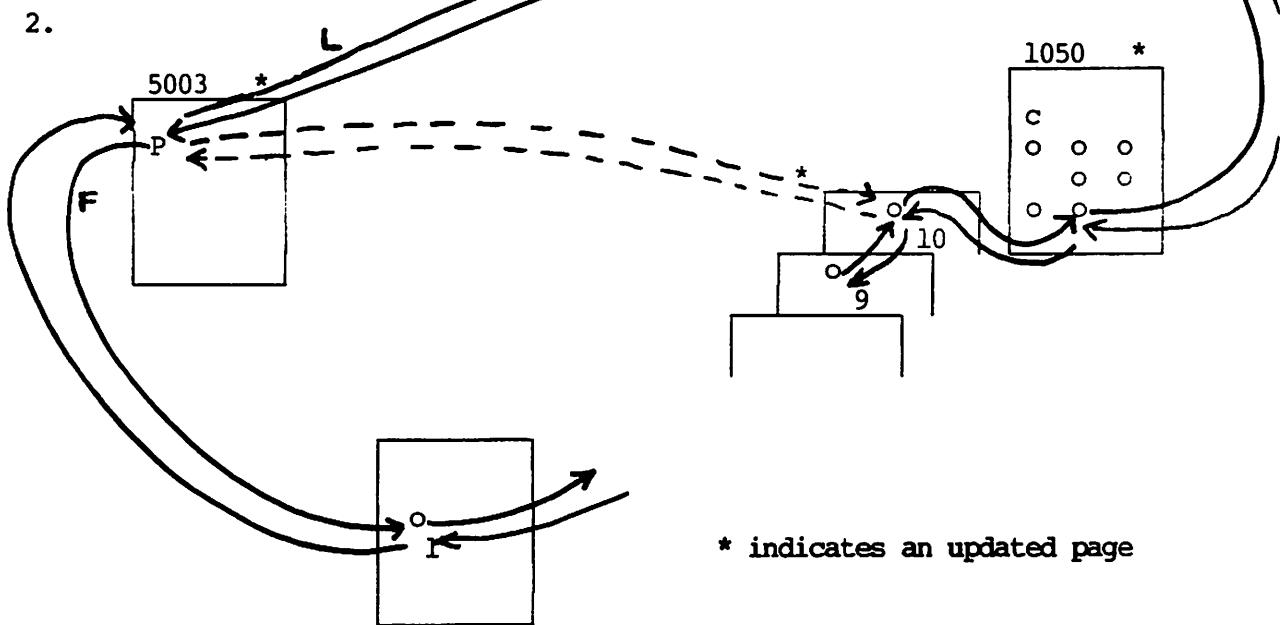
SOLUTION 100

1.	DML	Disc Reads	Disc Writes	
	OBTAİN ANY CUSTOMER	1	-	
	OBTAİN ANY PRODUCT	1	-	
	STORE C-ORDER	-	1	
	CONNECT C-ORDER	10 12	1 2	= 14 =
2.	OBTAİN ANY CUSTOMER	1	-	
	OBTAİN ANY PRODUCT	1	-	
	STORE C-ORDER	-	1	
	CONNECT C-ORDER	1 3	2 3	= 6 =

1.



2.



* indicates an updated page

IDMS DESIGN EXERCISE 110

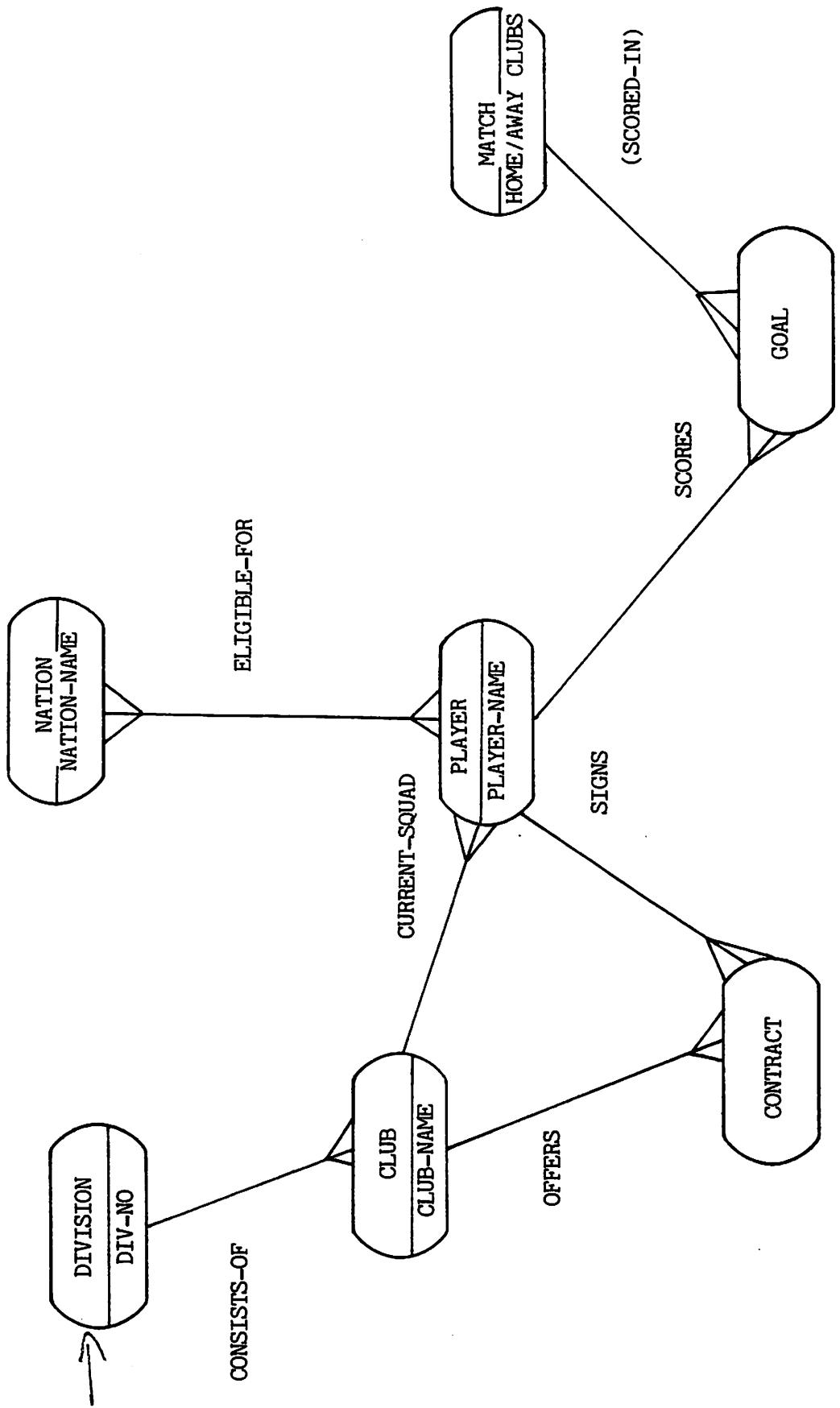
This exercise is based on a computer bureau which is to set up an IDMS database to hold information about the English Football League.

The purpose of the exercise is to highlight various aspects of the design process, not to carry out a full analysis of the environment. The data model and transactions, therefore, do not necessarily include all the functions of the real football league, but are sufficient for the purpose of this exercise.

The Football League database, once implemented, will enable sports writers and football officials to make enquiries, mostly on-line, as described below. The Data Analysis phase has been completed and the results, including the soft-box diagram, are included here.

Having checked that the Data Model can support the transactions, translate the model in terms of its required logical and physical characteristics, producing:

1. A Schema Diagram.
2. A Storage Diagram.



NO267

Figure 1 'Soft-box' representation of Football League

The Data Analysis phase of development has established:

1. The 'soft-box' representation of the Data Model illustrated in Figure 1.

2. Entity Descriptions:

DIVISION "One of the divisions of the English Football League, comprising those Clubs which compete against one another in the League competition during a particular season."

CLUB "A Football Club which is, or has been in the past, a member of the English Football League."

Note. We do not remove CLUBs which fail to be re-elected, until a further ten years have elapsed.

PLAYER "A footballer who has, or has had, a contract with a League CLUB."

Note. We shall remove players from the database only when they have not held a contract for at least ten years.

CONTRACT "A legal contract between a PLAYER and any CLUB, whether or not that Club is a League CLUB."

MATCH "A game played between two League CLUBs, and counting towards the result of the League competition."

GOAL "A goal, whether a Normal or 'Own-goal', scored during a MATCH."

NATION "A Nation entitled to play in the World Cup and/or other 'international' competitions."

3. The Entities shown in the diagram have the following properties:

<u>Entity</u>	<u>Identifying Attribute(s)</u>	<u>Some other Attributes</u>	<u>Volume</u>
Division	Division No.		4
Club	Club Name	Home Ground	Ave. 20 per Division
Player	Player Name	Date of Birth, Position, Height, Weight	20 per Club
Contract	Contract Number	Date Signed, Date Terminated	30 per Club
Match	Home Club Name, Away Club Name	Date, Start Time	380 per Division each season
Goal	Time Scored	'Own goal' indicator	Ave. 2 per match
Nation	Name		150

Note that:

- although some Players are eligible to play for more than one Nation, almost all are eligible for only one. Also, most Players, probably about 90%, are eligible for either England, Scotland, Wales or Ireland
- although the database will hold information extending over several seasons, there is no requirement to record which Division any Club was in in any previous season, and there is no requirement to hold any related information (eg winners in past years).

4. The types of enquiry to be supported are:

On-line:

- i) List all players in a given Division. 24 per week
- ii) How many players, currently with a given Club, are eligible to play for a given Nation? 300 per week
- iii) Which Division is a given Player in? 600 per week
- iv) What was the result of a given match, listing the goals in the sequence of occurrence, together with the scorer of each. 720 per week

Batch:

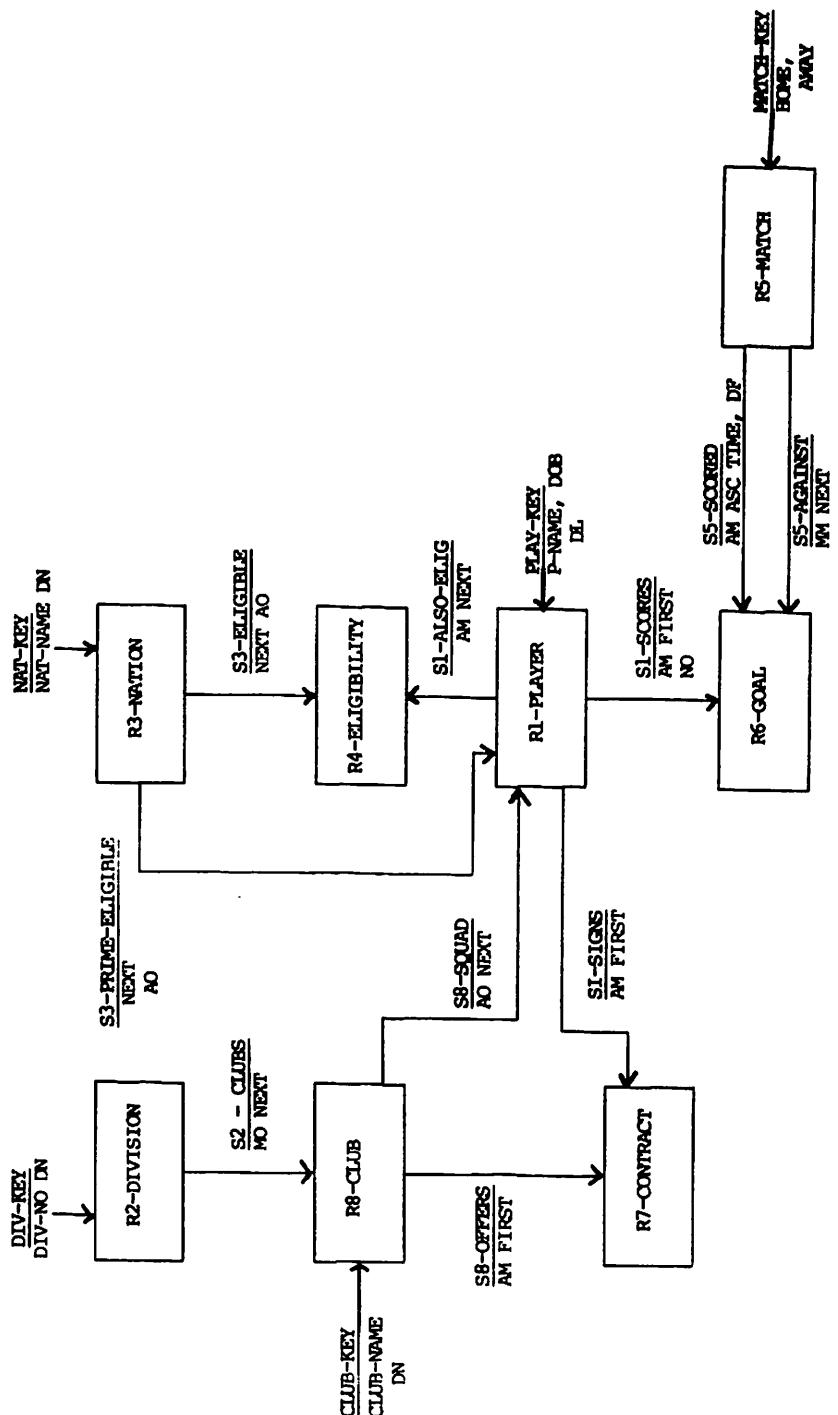
- v) List all players eligible for a given Nation. 1 per week
- vi) List all contracts (with non-league as well as League Clubs) for a given Player including Club details. 5 per week

IDMS DESIGN EXERCISE 110 - POSSIBLE SOLUTION

Sequence of Events

1. Translate Data Model into IDMS RECORDS and SETS.
2. VALIDATE the Data Structure with the known TRANSACTIONS
3. Determine entry points and KEYS.
4. Assign SET MEMBERSHIP and SET ORDER options.
5. Produce Schema Diagram (Figure 1).
6. Analyse record VOLUMES and set POPULATIONS and map ACCESS PATHS onto the schema Diagram (Figure 3).
7. Determine PLACEMENT of records where possible.
8. Work out ACTIVITY LISTS for those records whose placement is not obvious.
9. Include any 1:1 Access records and/or INDEX records.
10. Allocate POINTERS for chained Sets.
11. ASSIGN records to AREAS.
12. Produce Storage Diagram (Figure 2).
13. Check efficiency and TUNE if necessary.
14. AMEND Schema and/or Storage diagrams.

Descriptions of the decisions affecting both the Logical and Physical Design follow.



S - 23
N0266-5
© International Computers Limited 1986

S-23
N0266-5

Figure 1 The English Football League - Schema Diagram

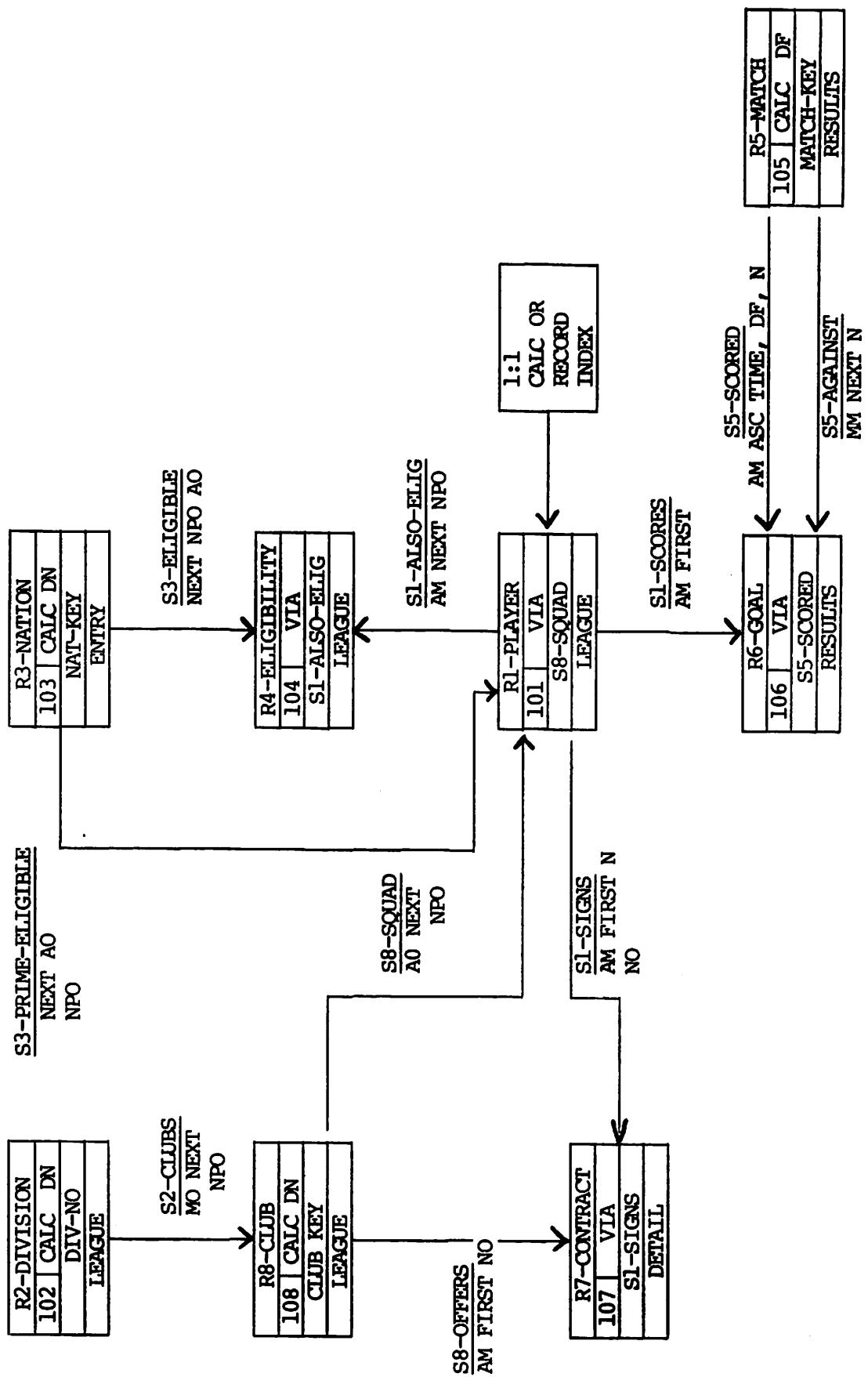


Figure 2 The English Football League - Data Storage Diagram

LOGICAL DESIGN DECISIONS

Translating the Data Model into Records and Sets

Each entity and relationship shown in the data model is first translated into records and sets. Exceptions to this rule apply to:

- the ELIGIBLE-FOR relationship, which being a many:many must be decomposed. A relationship record ELIGIBILITY participating in the S3-ELIGIBLE and S1-ALSO-ELIG sets is used to represent the relationship. Most of the time, however, this relationship exists as a one:many and therefore can be optimised.
Each PLAYER will exist in an occurrence of the S3-PRIME-ELIGIBLE set, with further eligibilities represented by the presence of an ELIGIBILITY record connected into both sets
- the inclusion of a second set between MATCH and GOAL to handle own-goals. This is to support enquiry (iv) with minimum navigation. Alternately a field within the GOAL record could be used to indicate whether the goal counted towards the home or away team.

Considering record design: by repeating Division number in PLAYER, navigation to DIVISION can be eliminated (enquiry (iii)). However, this does need extra control in checking that when a PLAYER transfers CLUBs, the Division number is updated, if necessary. A similar decision can be taken for CLUB.

Keys

Keys will be defined for five of the records, allowing entry point access at implementation time. The are:

- DIV-KEY - Division number
- CLUB-KEY - Club name. There could be a requirement for a secondary key here on CLUB code. If for instance club codes are held in the MATCH record and for transaction 4, full Club names are to be displayed, then access could be made directly to the CLUB record to retrieve the full name. This would save complex navigation from MATCH to CLUB.
- NAT-KEY - should this be Nation name? Can all users spell ZIMBABWE? If a nation code was used, this could lead to confusion between AUST(ria) and AUST(ralia). Two keys could be employed.

- PLAY-KEY - Player name only poses the probability of duplicate entries, which makes processing more difficult. Including initials and date of birth in the key, although theoretically still allowing duplicates, greatly reduces the probability.
- MATCH-KEY - If Home/Away Club names or codes are to be used, the processing must allow for duplicates; alternatively a third element (Season?) can be included in the key.

Set membership (Insertion/Retention)

- A Goal would be meaningless except in the context of a Match and a Player. So GOAL is an AM member of the sets S5-SCORED and S1-SCORES. (Note however that this implies that we must know who scored each goal before we store it in the database: simply knowing the match final score is not enough. It also implies that any errors in storing must be corrected by erasure and re-storage.) There will have to be manual connection of GOALS into S5-AGAINST.
- Making Players automatic members of S3-PRIME-ELIGIBLE implies again that we must know, at the time of storing each Player, at least one Nation for which he is eligible. Alternatively, we could hold a "dummy" Nation to own queries and unknowns. However, Players can legitimately change owners in this set, since a Player with multiple eligibility might eventually play for a Nation other than the one to which he was linked "directly". Also, correction of errors by erasure and re-storage is not practical, since it would involve erasing all contracts at the same time. So Retention must be Optional.
- Making Players automatic members of S8-SQUAD implies that we store a Player only when he has signed a contract with a recognised Club (and not, for example, when he is merely offered a contract). Optional Retention recognises the possibility of transfer or retirement.
- Similarly, Contracts' automatic membership of S1-SIGNS implies that Contracts are stored when they are accepted, not when they are offered. Since by its nature a Contract cannot be "transferred" to a different Club or Player, it has mandatory retention in both S1-SIGNS and S8-OFFERS.

the following words were written on the back of the
card:

Franklin D. Roosevelt
President of the United States
and his wife
Eleanor Roosevelt

Bugle Hill

Montgomery

Ala.

522 420

The address on the front of the card was:
Franklin D. Roosevelt
President of the United States
and his wife
Eleanor Roosevelt

Bugle Hill

Montgomery

Ala.

522 420

The address on the back of the card was:
Franklin D. Roosevelt
President of the United States
and his wife
Eleanor Roosevelt

Bugle Hill

Montgomery

Ala.

522 420

The address on the front of the card was:
Franklin D. Roosevelt
President of the United States
and his wife
Eleanor Roosevelt

Bugle Hill

Montgomery

Ala.

522 420

The address on the back of the card was:
Franklin D. Roosevelt
President of the United States
and his wife
Eleanor Roosevelt

Bugle Hill

Montgomery

Ala.

522 420

- Once a Player has been stored in the database, we must record all his subsequent contracts, whether with a League Club or a non-League one, in order to be able to give details of his career.

This can be handled in several ways:

- . record the non-League Club in the Database, keeping Contracts as automatic members of S8-OFFERS. But Clubs must now be manual rather than Automatic members of S2-CLUBS, OR we must similarly have "Divisions" other than the four English League ones in the Database. (A half-way house would be to have either a single dummy "non-league" Division to own all non-League Clubs or to have a single "non-league" Club to own all non-League Contracts)
- . make Contracts manual members of S8-OFFERS, and carry details of non-League Clubs in the non-League Contract records. (This in turn implies either two Contract record types for league and non-League Contracts, or a variable-length record)
- Clubs' automatic membership of S2-CLUBS implies that we store a Club only when it is elected to the League, (but see previous point) while Optional retention enables us to promote or relegate.

Set Order

- There is no useful order of Players or Eligibilities within S3-ELIGIBLE, or Eligibilities within S1-ALSO-ELIG, or of Players within S8-SQUAD. The only interesting order of Clubs within S2-CLUBS is their current ranking in the competition, which changes daily, and can be dealt with simply by sorting to that sequence before displaying. So all these sets can have an order of NEXT or FIRST.
- In S5-SCORED, Goals can be ordered in Ascending order of the time scored to facilitate generation of progress scores. Since S5-SCORED is clustered, average membership is very low, and orders are never updated once established correctly, extra processing due to having to Connect in sorted sequence is trivial.
- Similarly in S1-SCORES, a Descending sequence of Date and Time would involve trivial updating overhead, despite the very large set memberships of some set occurrences, since Goals will be stored in approximately Ascending Date/Time sequence. This would keep each Player's goals in sequence with the more recent ones (presumably the more interesting) towards the front of the set. On the other hand, an order of FIRST would achieve much the same effect with less overhead, and may be good enough given no specific requirement at this stage.

Digitized by srujanika@gmail.com

102 P. S. T. Lai

在於此，故其後人之學，亦復以爲子思之傳。蓋子思之學，實出於孟子，而孟子之學，又實出於子思也。

卷之三

१०८ अनुवाद विजय कुमार

34721 44-1974

卷之三

and the other day I was at a conference in the mountains of Colorado. The people there were very friendly and helpful. They gave me a lot of information about the area and its history. I also met some interesting people who shared their knowledge and experiences with me. Overall, it was a great experience and I learned a lot.

- Similar considerations apply to S1-SIGNS and S8-OFFERS, particularly if Date of Termination were the key, with "low values" indicating "no fixed termination date, still current".
- There are only three set types which can have large memberships, and may be sorted: S8-OFFERS, S1-SCORES and S1-SIGNS. None has a requirement to be searched on the sort key, and S1-SIGNS will not be scattered. So even assuming ISMS-X, all sets are likely to be Chained rather than Indexed.

The Physical Considerations

Initially map the priority transactions onto the Schema Diagram in the form of access paths. This will be useful when analysing traversals to records and drawing up activity lists.

Calculate the expected population for each set and note on the diagram. Also estimate the record volumes, taking into consideration growth factors where possible.

Placement of Records

Initially the transactions should be mapped on to the logical schema diagram (Figure 3) showing the access paths through the data structure. This aids the designer when choosing the placement, because entry points are clearly illustrated and it highlights heavily traversed record types eg PLAYER. Analysis of activities on frequently used records can then be carried out as discussed later.

DIVISION, NATION and MATCH are all top-level, direct entry records and could all be CALC.

Considering record accesses only, ELIGIBILITY could be via S3-ELIGIBLE or S1-ALSO-ELIG, and GOAL via S1-SCORES or S5-SCORED. However, S3-ELIGIBLE and S1-SCORES have a number of very large occurrences whereas occurrences of S1-ALSO-ELIG and S5-SCORED are low. Placing ELIGIBILITY via S1-ALSO-ELIG and GOAL via S5-SCORED therefore ensures more efficient clustering. The same argument can be used for CONTRACT and the S1-SIGNS set.

AMERICAN MUSEUM OF NATURAL HISTORY
NEW YORK CITY

9032 - *Phalaenopsis amabilis* (L.) Lindl. (syn. *P. amabilis* Lindl.)

காலத்திலே குறிப்பிட்டு வரும் சம்பந்தமாக அதை விரிவாக விவரிதிப்பது முன் கூறுவது என்று அறியப்படுகிறது.

Almond Butter, 1 lb. - \$1.00 - 1 lb. - \$1.00 - 1 lb. - \$1.00
Peanut Butter, 1 lb. - \$1.00 - 1 lb. - \$1.00 - 1 lb. - \$1.00

Activity lists will be used to determine the placement of PLAYER and CLUB.

PLAYER

All six transactions access PLAYER. Transactions 1 to 4 are on line and assumed to take priority. So we will just consider these.

Transaction 1

Direct to DIV	24
S2-CLUBS to CLUB	24 x 20 480
S8-SQUAD to PLAYER	480 x 20 9600

Transaction 2

90% of PLAYERS are eligible for the UK countries. If we assume that 90% of this transaction is for UK countries, the transaction can be supported in one of two ways:

- for UK countries enter at CLUB and then look at PLAYERS. There are only 20 PLAYERS : CLUB. The number of PLAYERS to NATION can be calculated:

$$\text{No. of PLAYERS} = 4 \times 20 \times 20 = 1600 \\ (\text{DIV}) \quad (\text{CLUB}) \quad (\text{PLAYERS})$$

$$90\% \text{ PLAYERS} = 1440$$

$$\text{UK NATION : PLAYER} = 1440 : 4 = 1 : 360$$

Hence it is more efficient to search a set of 20 members than 360.

- for non UK countries, 146 in all, the NATION to PLAYER ratio is

$$10\% \text{ PLAYER} = 160$$

$$\text{NATION : PLAYER} = 1 : 1.1$$

Thus it is more efficient to search a set of 1.1 members than 20 (from CLUB). So non UK enquiries will enter at NATION.

These PEPs will be developed by our partners and will be used to inform and support the development of the new PEPs.

“我就是想在你身上找点东西，好让我自己也觉得不是白活了。”

100-3 100-3 100-3
100-3 100-3 100-3

10. The following table summarizes the results of the study. The first column lists the variables, the second column lists the descriptive statistics, and the third column lists the results of the regression analysis.

Figure 10. The effect of the number of hidden neurons on the performance of the neural network.

After reading the above, you will have a clear idea of what is involved in the preparation of a good thesis.

19. The following is a list of the names of the members of the Board of Education.

19. The following table shows the number of hours worked by each of the 1000 workers in the firm.

19. *Leucosia* *leucostoma* *leucostoma* *leucostoma* *leucostoma* *leucostoma* *leucostoma*

1996. 1997. 1998. 1999. 2000. 2001. 2002. 2003. 2004. 2005. 2006. 2007. 2008. 2009. 2010. 2011. 2012. 2013. 2014. 2015. 2016. 2017. 2018. 2019. 2020.

It is estimated that about 10% of the world's population is infected with *H. pylori*.

Journal of Clinical Endocrinology and Metabolism, Vol. 132, No. 10, October 1997, pp. 3033–3039.

of *Archidium*, but it is a more or less distinct genus, and I have no objection to its being so regarded.

10. The following table shows the number of hours worked by each employee.

68-10
S. S. 10

本办法所称的“重大危险源”，是指具有较大危险性的危险化学品生产、经营、储存和使用单位。

Query on UK

Enter at CLUB

Volume is 90% of 300 = 270

| | |
|--------------------------|-----------------|
| Direct to CLUB | 270 |
| S8-SQUAD to PLAYER | 270 x 20 = 5400 |
| S3-ELIGIBILITY to NATION | 5400 |

Query on non-UK

Enter at NATION

Volume is 10% of 300 = 30

| | |
|--------------------------|---------------|
| Direct to NATION | 30 |
| S3-ELIGIBILITY to PLAYER | 30 x 1.1 = 33 |
| S8-SQUAD to CLUB | 33 |

Transaction 3

Division number held as field within PLAYER.

| | |
|------------------|-----|
| Direct to PLAYER | 600 |
|------------------|-----|

S - 30
N0266-5

© International Computers Limited 1986

1054

1.0750 02 2009

Digitized by Google

MAY 2002

2

THE JOURNAL OF CLIMATE

◎ 中国古典文学名著全集·《水浒传》

1994. *Journal of Management Education*.

166 of 166 pages - Page 166 of 166 pages - Page 166 of 166 pages - Page 166 of 166 pages

卷之三

卷之三

Chlorophyll a

BOOKS RECEIVED
RECENT ADDITIONS

卷之三

— Page 23 —

- 97 -

卷之三

www.sagepub.com/journals/submitmanuscript.aspx

1920-21 - 1921-22 - 1922-23 - 1923-24 - 1924-25 - 1925-26 - 1926-27 - 1927-28

Transaction 4

| | | |
|-----------|-----------------|------|
| | Direct to MATCH | 720 |
| S5-SCORED | to GOAL | 1440 |
| S1-SCORES | to PLAYER | 1440 |

Activity on PLAYER:

| | |
|-------------------|---|
| Using S8-SQUAD | 9600 |
| | 5400 |
| | 33 (borrowed) |
| | <u>15033</u> from club) |
| Using S3-ELIGIBLE | 5400 (borrowed |
| | 33 from NATION) |
| | <u>5433</u> |
| Direct to PLAYER | <u>600</u> |
| Using S1-SCORES | 1440 (ignored =
ACCESS to
an owner) |

Since the highest activity on PLAYER is caused by traversing the S8-SQUAD set, it is reasonable to place PLAYER via that set for efficiency of access.

However, direct entry is required, so inclusion of a 1.1 CALC access mechanism or a record index for IDMS-X users, will provide for both requirements.

UNQUOTE OR TWO AND THREE CROWNS DRAINED FROM SYSTEM. THE UN-
TIES IN THE PIPING LINE FLOWED DOWN THE TUBE AND
DID NOT REACH THE HORN. A SMALL CROWNS WOULD HAVE BEEN
BLOWN IN AND

THE UNQUOTE OR TWO AND THREE CROWNS WAS DRAINED
OUT OF THE TUBE AND THE CROWNS WOULD HAVE BEEN
A SMALL CROWNS. A SMALL CROWNS IS ONE CROWN WHICH
WILL NOT REACH THE HORN.

UNQUOTE OR TWO AND THREE CROWNS WAS DRAINED
BECAUSE IT WAS NEEDED. BECAUSE IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE. IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE. IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE.

UNQUOTE OR TWO AND THREE CROWNS WAS DRAINED
BECAUSE IT WAS NEEDED. BECAUSE IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE. IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE. IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE.

UNQUOTE OR TWO AND THREE CROWNS WAS DRAINED
BECAUSE IT WAS NEEDED. BECAUSE IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE. IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE.

UNQUOTE OR TWO AND THREE CROWNS WAS DRAINED
BECAUSE IT WAS NEEDED. BECAUSE IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE. IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE.

UNQUOTE OR TWO AND THREE CROWNS WAS DRAINED
BECAUSE IT WAS NEEDED. BECAUSE IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE. IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE.

UNQUOTE OR TWO AND THREE CROWNS WAS DRAINED
BECAUSE IT WAS NEEDED. BECAUSE IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE. IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE.

UNQUOTE OR TWO AND THREE CROWNS WAS DRAINED
BECAUSE IT WAS NEEDED. BECAUSE IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE.

UNQUOTE OR TWO AND THREE CROWNS WAS DRAINED
BECAUSE IT WAS NEEDED. BECAUSE IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE. IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE.

UNQUOTE OR TWO AND THREE CROWNS WAS DRAINED
BECAUSE IT WAS NEEDED. BECAUSE IT WAS NEEDED TO GET THE
PIPE SEPARATED FROM THE TUBE.

Pointers

For maximum efficiency, NEXT, PRIOR and OWNER pointers can be included on all sets, space permitting. However, careful examination of each set will highlight those sets where a compromise between space and efficiency can be made:

- S5-SCORED and S5-AGAINST have low average and maximum memberships, are clustered, and have no requirement to disconnect. They might safely dispense with Prior pointers. Similarly, they might dispense with owner pointers also.
- The members of S8-OFFERS, S1-SIGNS, S1-SCORES and S3-ELIGIBLE also are rarely or in some cases never disconnected or erased except when the owner is erased. Might they too dispense with Prior pointers? The argument is perhaps strongest for the clustered Contracts with S1-SIGNS, and weakest for the Players within S3-ELIGIBLE, where logical deletion might cause inefficiency.
- Are Prior pointers really justified for CLUBS with S2-CLUBS, since disconnection occurs predictably only once a year, and unpredictably only should a Club cease to exist unexpectedly?

Areas

- LEAGUE comprises DIVISION, there are only 4 occurrences. CLUB and PLAYERS are to be clustered on same page. ELIGIBILITY is VIA PLAYER, and the number of occurrences so small, if they exist at all, that they would not affect the efficient clustering of PLAYER records.
- DETAIL contains only CONTRACTS. Although clustered according to PLAYER we do not want them on the same page as this would destroy the compact clustering of PLAYERS and CLUB. Therefore they are in their own area having the additional benefit of the file being required only for the batch runs, otherwise off lined.
- RESULTS comprises MATCH and GOAL, retaining their compact clustering with no 'outside' interference.
- ENTRY will hold the 1:1 CALC occurrences or the indexes for PLAYER. It can also usefully contain NATION. If, however, a Nation code was held as a field within PLAYER and ELIGIBILITY then NATION records would not be needed for the on-line enquiries, in which case NATION could be included in DETAIL area instead.

As an alternative to separate areas, individual page ranges could be allocated to particular record type(s).