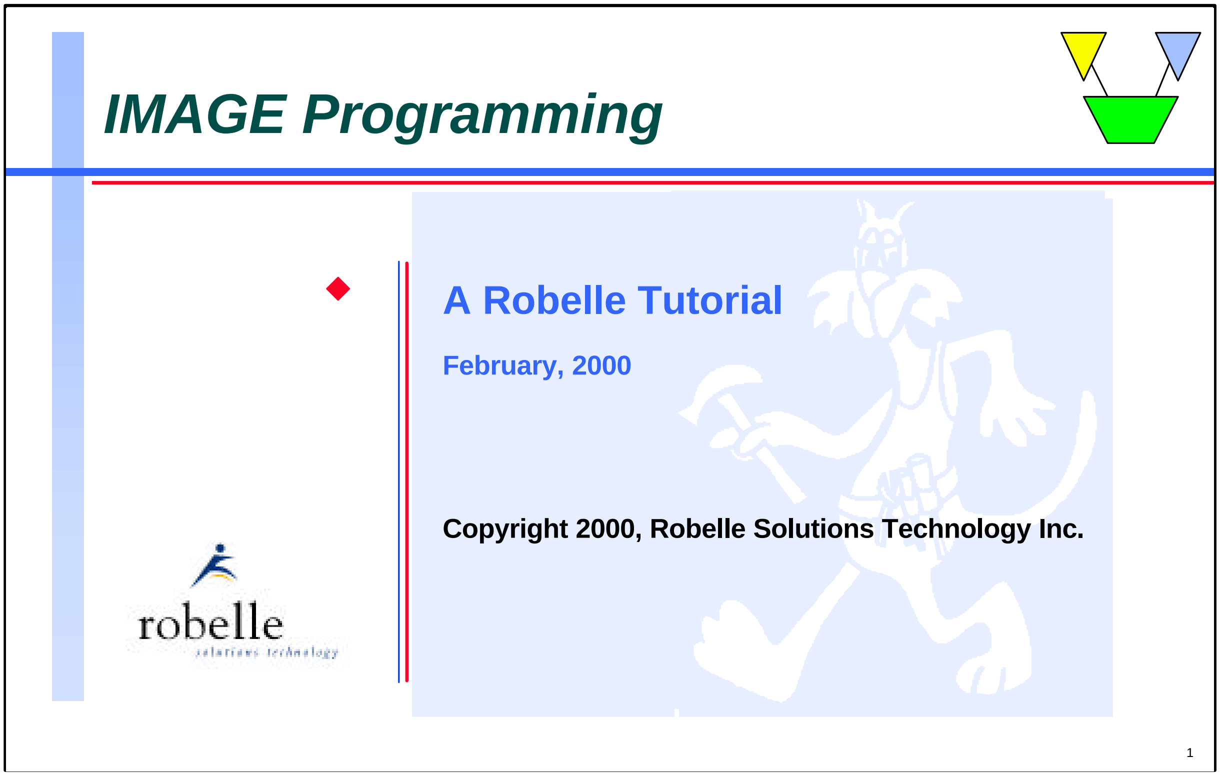


IMAGE Programming

A Robelle Tutorial

February, 2000

For Techies

References

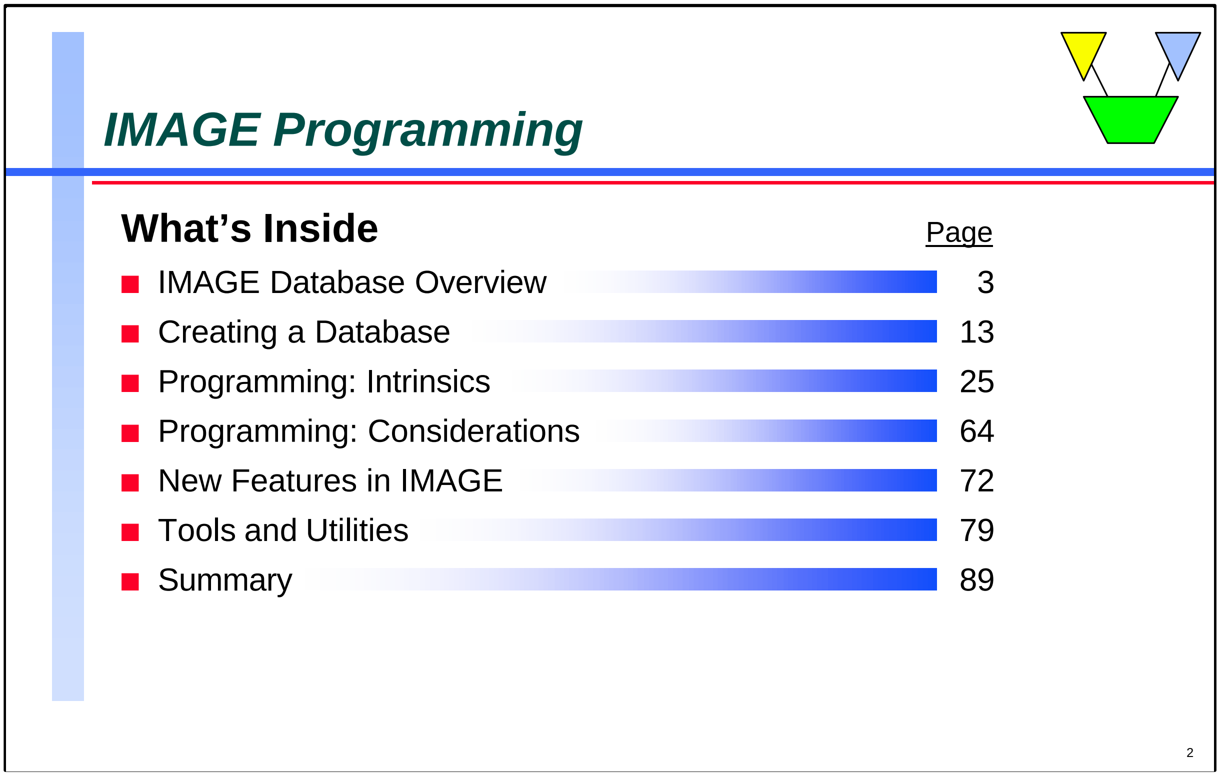


IMAGE Programming

What's Inside

Page

For Techies

References

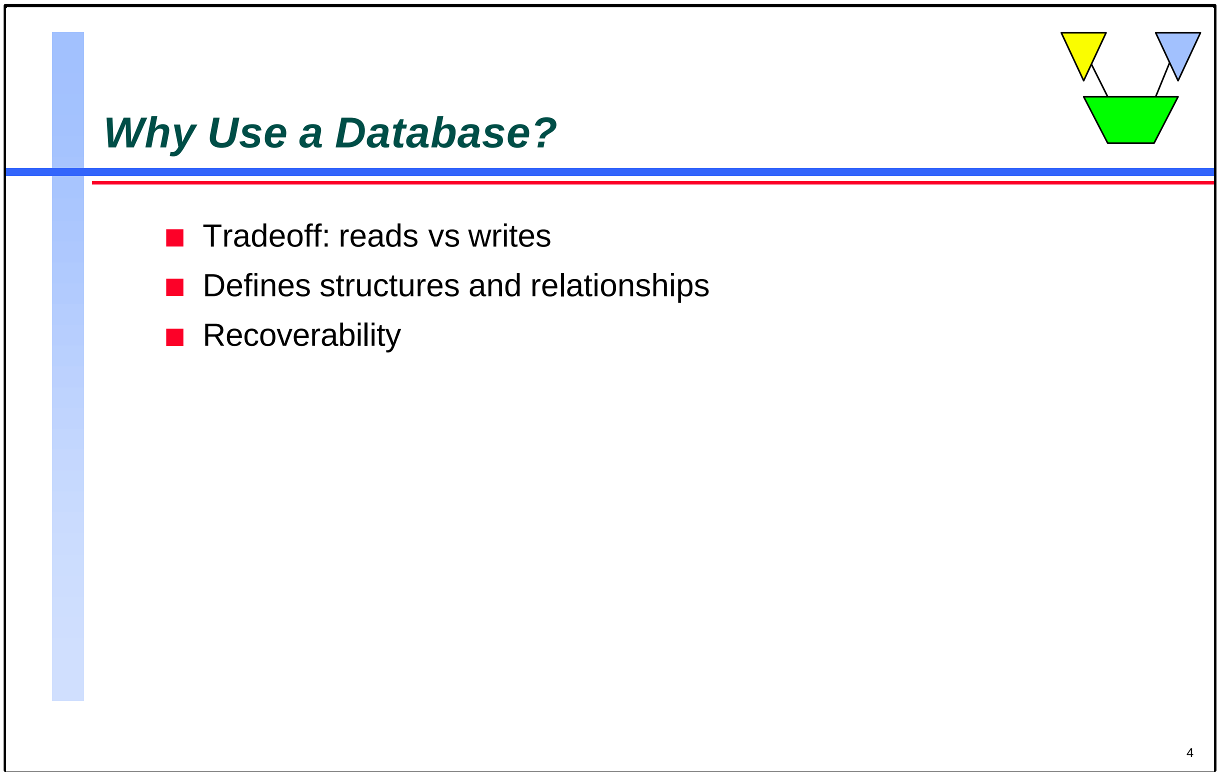
IMAGE Database overview

In this section

	Page
What is a database?	4
Structure	5
Record retrieval	8
Item formats	11

For Techies

References



Why Use a Database?

- Tradeoff: reads vs writes
- Defines structures and relationships
- Recoverability

4

Why not store data in un-indexed flat files? Because you can't read the entire file serially every time a customer calls with a query. Databases are a way to retrieve particular records from a large amount of data in a fast and efficient way. This means that index structures have to be built when records are added, so they’re available when those records need to be retrieved. It’s a tradeoff between slightly less efficient writes vs much more efficient reads. Typically, fields that are required for on-line retrievals (customer-no, product-no, invoice-no, etc) are indexed.

Databases also define structure and relationships between the different data values stored. Individual datasets store the same logical information, with a rigidly defined structure that allows different applications to read and write them in a consistent way.

Because IMAGE databases are integral to the HP 3000, they have a range of special facilities to apply appropriate security, integrity and recoverability. Files are flagged as “privileged” so special security applies. Database logging can be enabled, for auditing and recovery. Backups and recovery are subject to special checks and controls.

For Techies

References

Structure

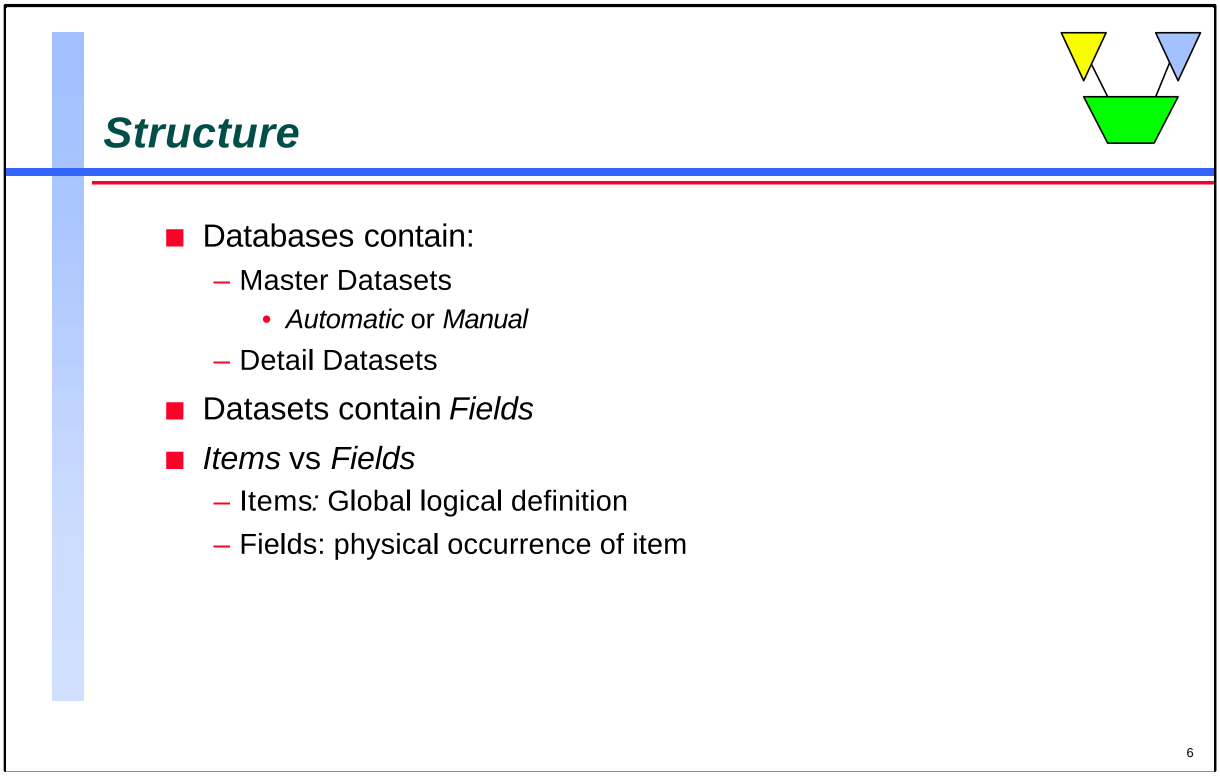
- Look for unique value fields
  - put in master datasets
- Use Unique fields to retrieve associated records
  - in detail datasets

Use product-number (unique) to retrieve associated invoice-line records for that product. Similarly, customer-no to retrieve invoices for that customer. Unique values (Customer record, product record) are stored in masters and used to retrieve repeating values (invoice-lines) which are stored in details.

Masters have only one key, but details can have multiple keys. I.e. they can have more than one field defined as a key. So if you require more than one access path to the data, it would need to be stored in a detail dataset, even though the data values might be unique. In these cases, it’s common to define “Automatic Masters” for those (unique) keys.

For Techies

References



Structure

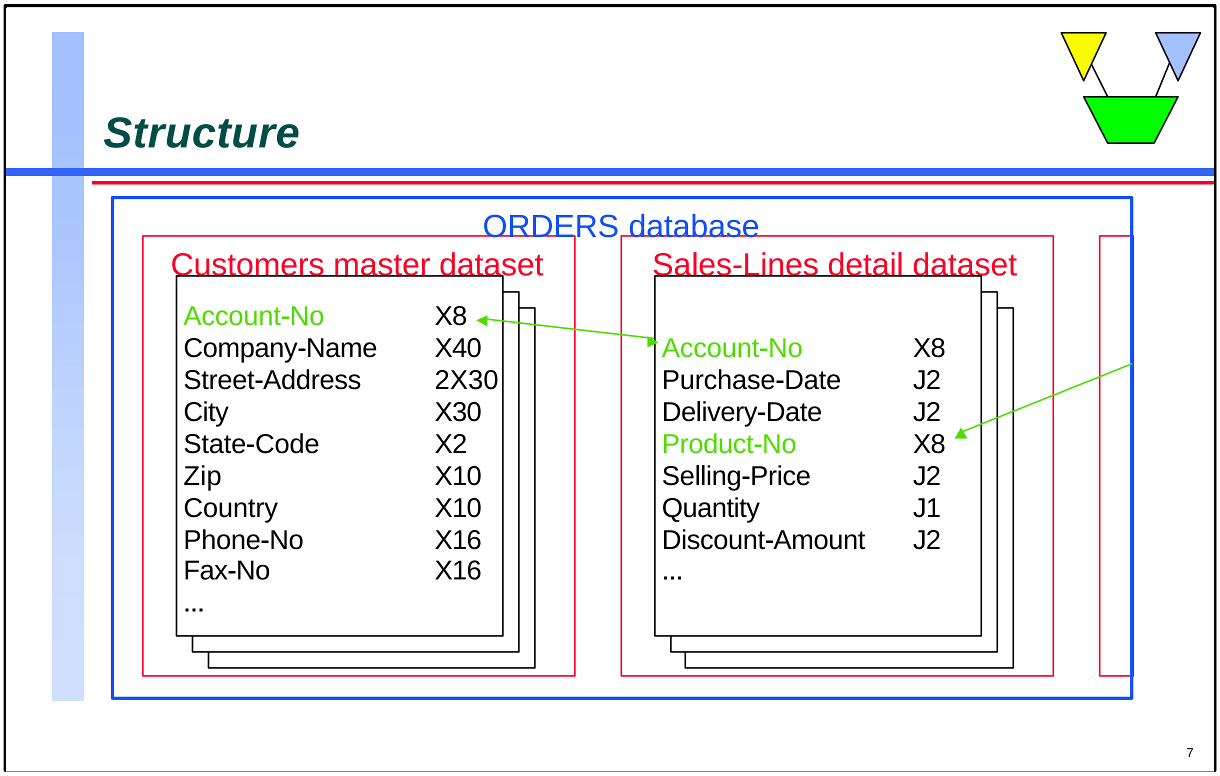
- Databases contain:
  - Master Datasets
    - *Automatic* or *Manual*
  - Detail Datasets
- Datasets contain *Fields*
- *Items* vs *Fields*
  - Items*:* Global logical definition
  - Fields: physical occurrence of item

When creating a database, you first have to declare the items of data that will be stored. Items are logical data values, e.g. Customer Name, Zip Code, product-description, sales-price, phone-no, etc. Then you define how these logical entities will be stored, by defining the datasets (Automatic Master, Manual Master, or Detail), and, for each dataset, which fields they will contain, and in what sequence.

For Techies

References

The database declaration language is described in section 2.



Structure

ORDERS database

Customers master dataset

Account-No	X8
Company-Name	X40
Street-Address	2X30
City	X30
State-Code	X2
Zip	X10
Country	X10
Phone-No	X16
Fax-No	X16
...	

Sales-Lines detail dataset

Account-No	X8
Purchase-Date	J2
Delivery-Date	J2
Product-No	X8
Selling-Price	J2
Quantity	J1
Discount-Amount	J2
...	

7

A **database** is a collection of related datasets. There can be up to 199 datasets in a database.

Each **dataset** is made up of **data entries**, also known as **records**. There can be up to 255 items in a dataset record. All the records in a dataset have identical layout.

Each **data item** in a record has specific attributes regarding what kind of data it can hold, how much of that data it can hold, and in what format the data is stored. E.g., eight uppercase letters, or binary numeric capable of holding ten digits, or 'packed' numeric capable of storing five digits.

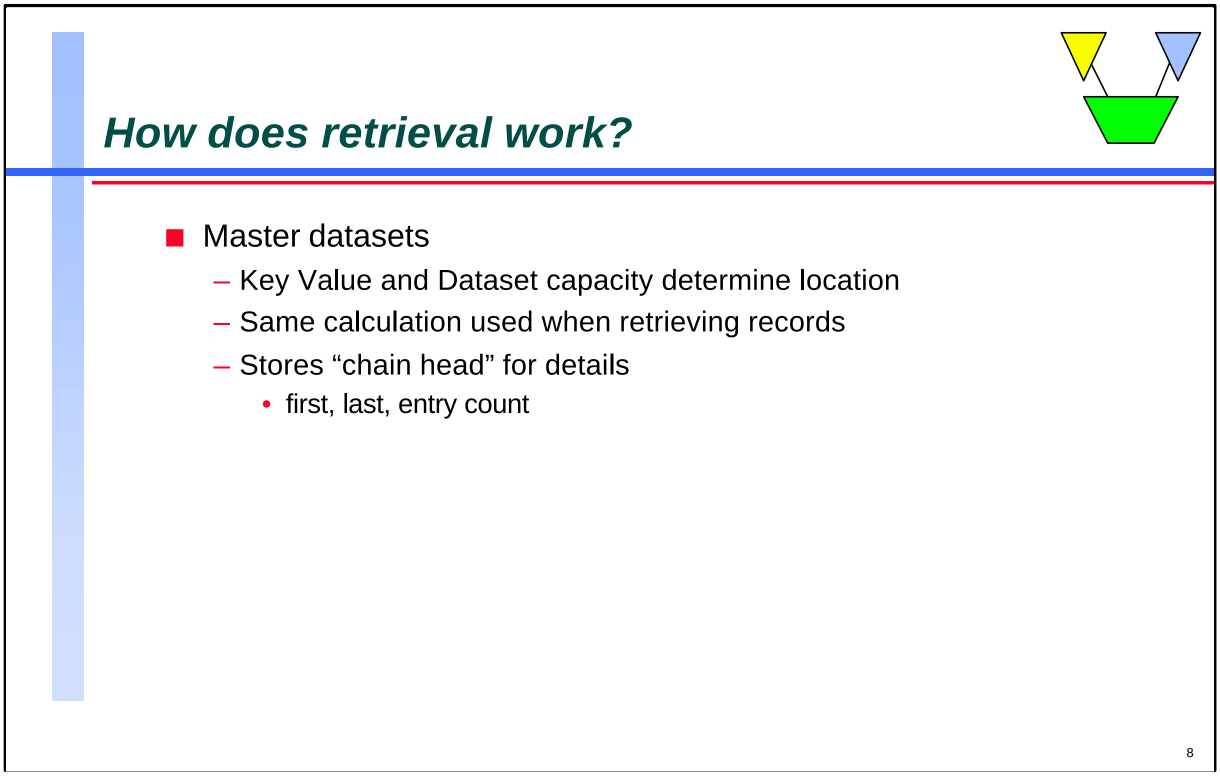
Master dataset entries are accessed by their **key**. The key is the piece of data that uniquely identifies a data entry. E.g., the single record for customer account number 12345. In a master dataset only one field is designated as the key.

Detail datasets are also accessed by their keys. Any detail dataset may have up to 16 items designated as keys. All the records in a detail dataset that have the same key make up a **chain** of detail records. E.g., all the sales records for customer 12345, or all the sales records for product BHGF-227.

Each key in a detail dataset relates back to a master dataset. This relationship forms a **path**. Therefore a detail dataset may be related to up to 16 master datasets.

For Techies

References



How does retrieval work?

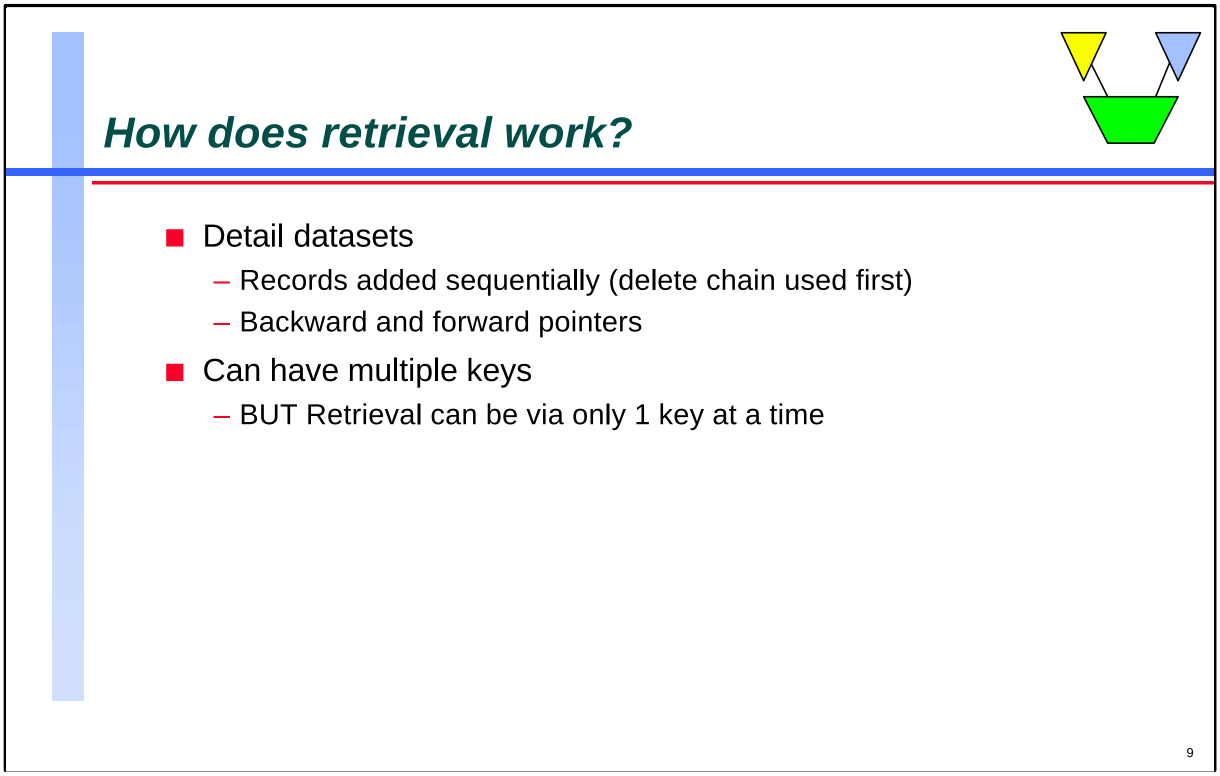
- Master datasets
  - Key Value and Dataset capacity determine location
  - Same calculation used when retrieving records
  - Stores “chain head” for details
    - first, last, entry count

When writing a new entry into a master dataset, IMAGE determines its position by applying a formula based on the key value and the dataset capacity. It uses the same calculation to establish the record’s location when retrieving. This means that the full key value is required when retrieving records, and that key values must be unique. It also means that a Master dataset can have only one key, as a record cannot be in 2 locations at the same time.

Each Master Dataset record stores information about its associated records in detail datasets. This is called the “Chain Head”, and stores the location of the first detail record, the last detail record, and the number of detail records with that key value. A master dataset can store the “chain head” for multiple associated detail datasets. For example, a CUSTOMER-MASTER dataset could store chain head information for ADDRESS-DETAIL and INVOICE-DETAIL, each of which have CUSTOMER-NUMBER as a key.

For Techies

References



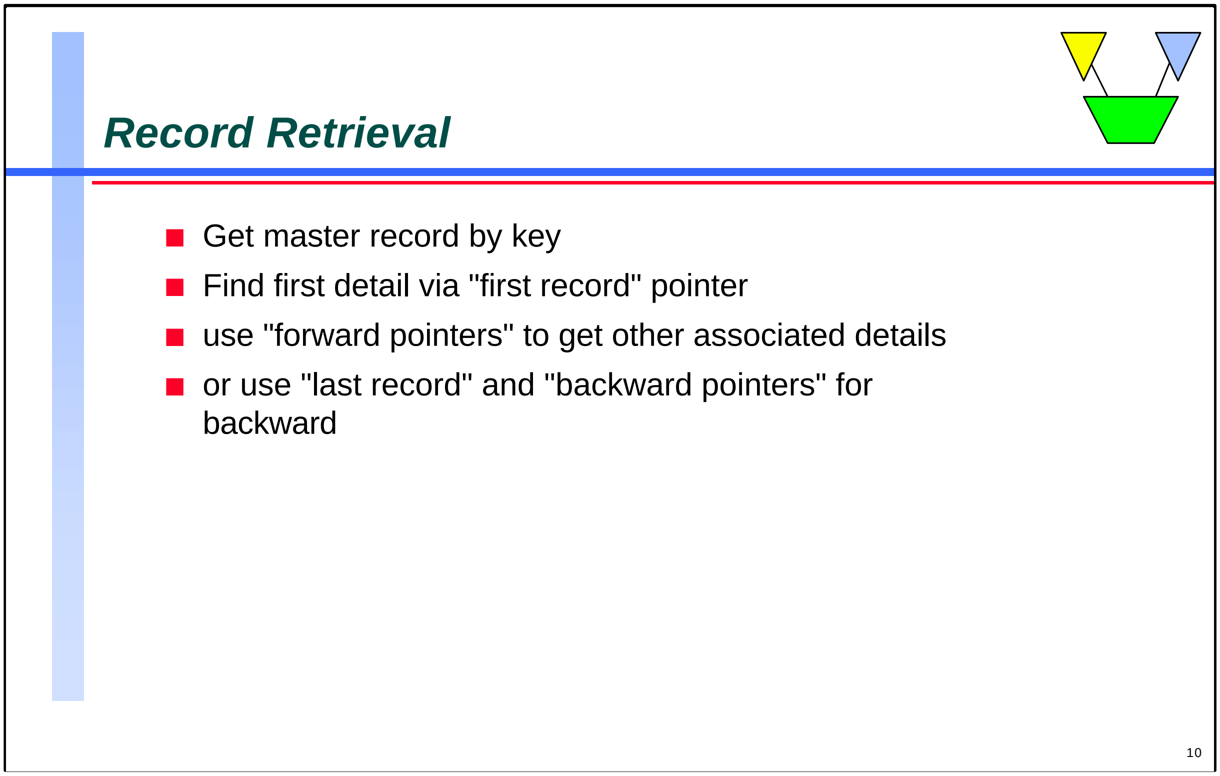
How does retrieval work?

- Detail datasets
  - Records added sequentially (delete chain used first)
  - Backward and forward pointers
- Can have multiple keys
  - BUT Retrieval can be via only 1 key at a time

New records are added sequentially to a detail dataset. If entries have been deleted, those deleted locations are used first, in order to keep the dataset’s “highwater mark” as low as possible (sequential reads typically read up to the “highwater mark”). Each detail record stores backward and forward pointer for each key field, I.e. pointing to the location of the previous and next entries in the chain of entries with that key value. Detail Datasets can have multiple key fields, but only one set of pointers can be followed, so retrieval can be done on only one key at a time.

For Techies

References



Record Retrieval

- Get master record by key
- Find first detail via "first record" pointer
- use "forward pointers" to get other associated details
- or use "last record" and "backward pointers" for backward

10

Let’s say, for example, that you need to report all the sales you have made to a particular customer:

1) Retrieve that customer’s CUSTOMER-MASTER record by finding its position using the ACCOUNT-NUMBER key field.

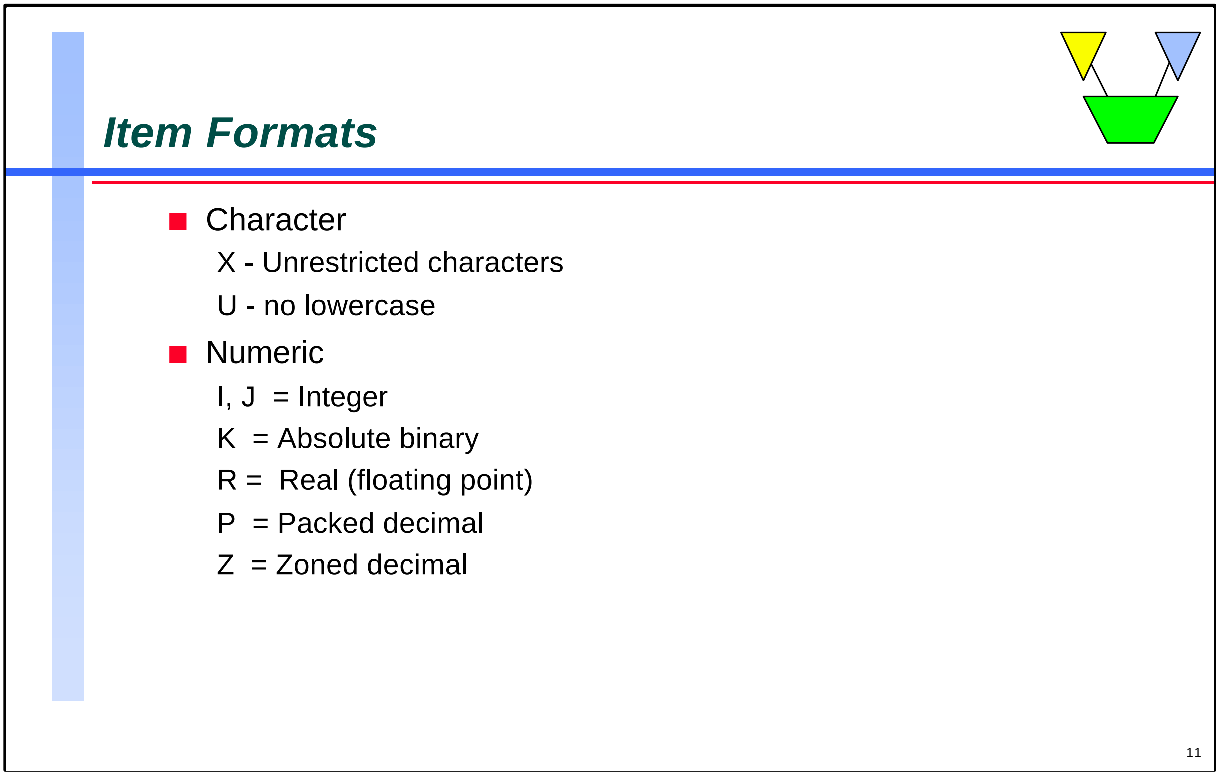
2) Use the chain-head information from the master record to retrieve the first SALES-DETAIL record.

3) Use the forward pointer from the first detail record to retrieve the second detail record.

4) Use each subsequent detail record’s forward pointer to retrieve the next record, until all detail records have been retrieved.

For Techies

References



Item Formats

- Character
  - X - Unrestricted characters
  - U - no lowercase
- Numeric
  - I, J = Integer
  - K = Absolute binary
  - R = Real (floating point)
  - P = Packed decimal
  - Z = Zoned decimal

Sample Suprtool FORM listings:

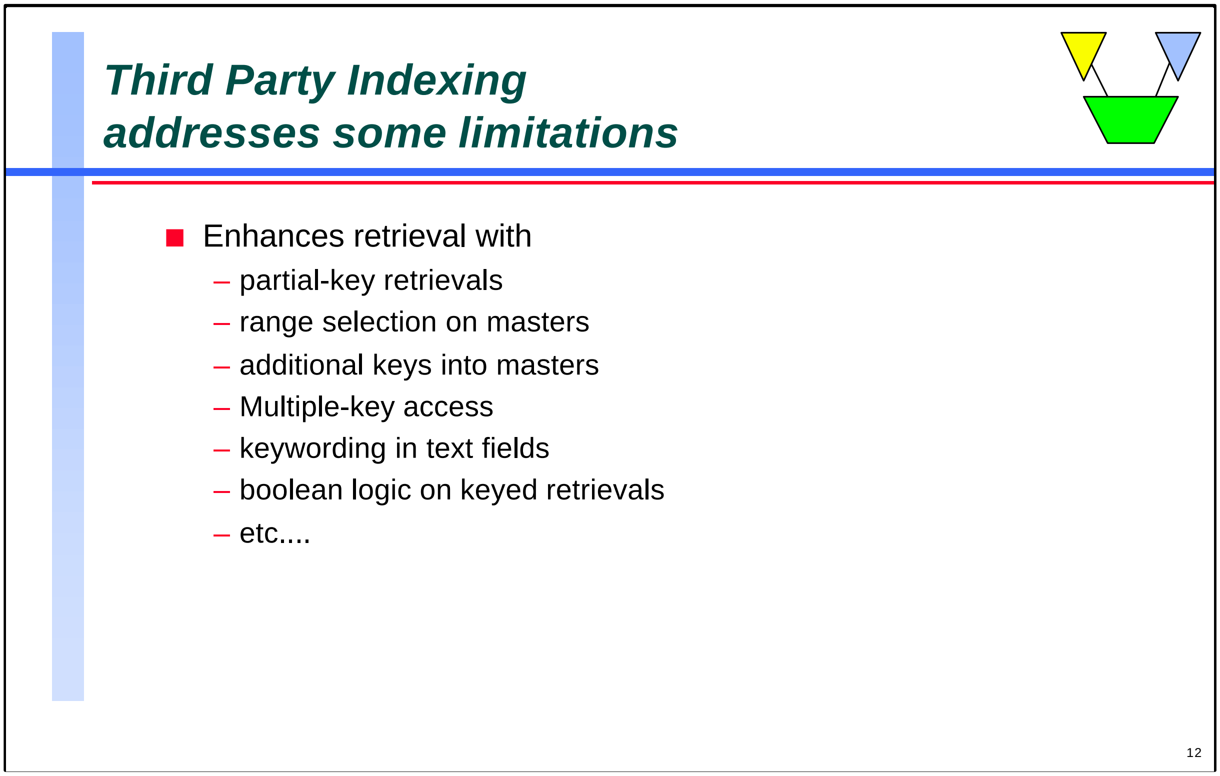
```
>form m-product
Database: STORE.DATA.ACCOUNT

    M-PRODUCT        Master                  Set# 2
       Entry:                    Offset
          PRODUCT-DESC        X30      1
          PRODUCT-MODEL       X10     31
          PRODUCT-NO          Z8      41  <<Search Field>>
    Capacity: 307 (12)  Entries: 13  Bytes: 48
>form d-sales
Database: STORE.DATA.ACCOUNT

    D-SALES          Detail                  Set# 5
       Entry:                    Offset
          CUST-ACCOUNT        Z8       1  (!M-CUSTOMER)
          DELIV-DATE          J2       9
          PRODUCT-NO          Z8      13  (M-PRODUCT)
          PRODUCT-PRICE       J2      21
          PURCH-DATE          J2      25
          SALES-QTY           J1      29
          SALES-TAX           J2      31
          SALES-TOTAL         J2      35
    Capacity: 602 (14)  Entries: 8  Highwater: 8  Bytes: 38
```

Note that J and I fields are shown in “words” (I.e. 2 bytes per word). E.g. J2 is 4 bytes, J1 is 2 bytes. Packed fields are shown in “nibbles” (4 bits), e.g. P28 is 14 bytes long.

Examine “offset” to determine physical length in bytes.

For Techies

References



Third Party Indexing addresses some limitations

- Enhances retrieval with
  - partial-key retrievals
  - range selection on masters
  - additional keys into masters
  - Multiple-key access
  - keywording in text fields
  - boolean logic on keyed retrievals
  - etc....

12

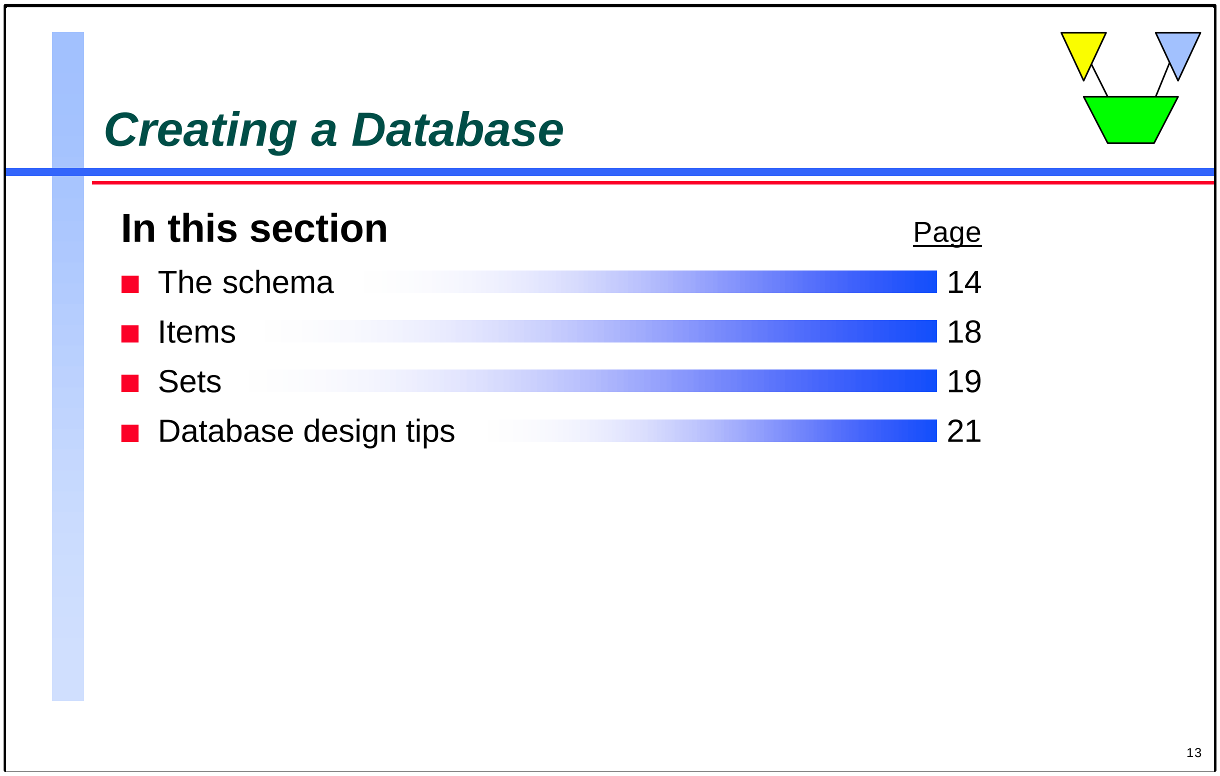
Third Party Indexing tools (Omnidex and Superdex) allow you to build additional index structures to enhance record retrieval options. Additional overhead when writing records is offset by more powerful and efficient record retrievals. These tools are not packaged with IMAGE, but Hewlett-Packard has worked with their suppliers to tightly integrate their operation with IMAGE. For more information, contact:

Dynamic Information Systems Corporation (http://www.disc.com/) for Omnidex

Bradmark Technologies Inc (http://www.bradmark.com/) for Superdex

For Techies

References



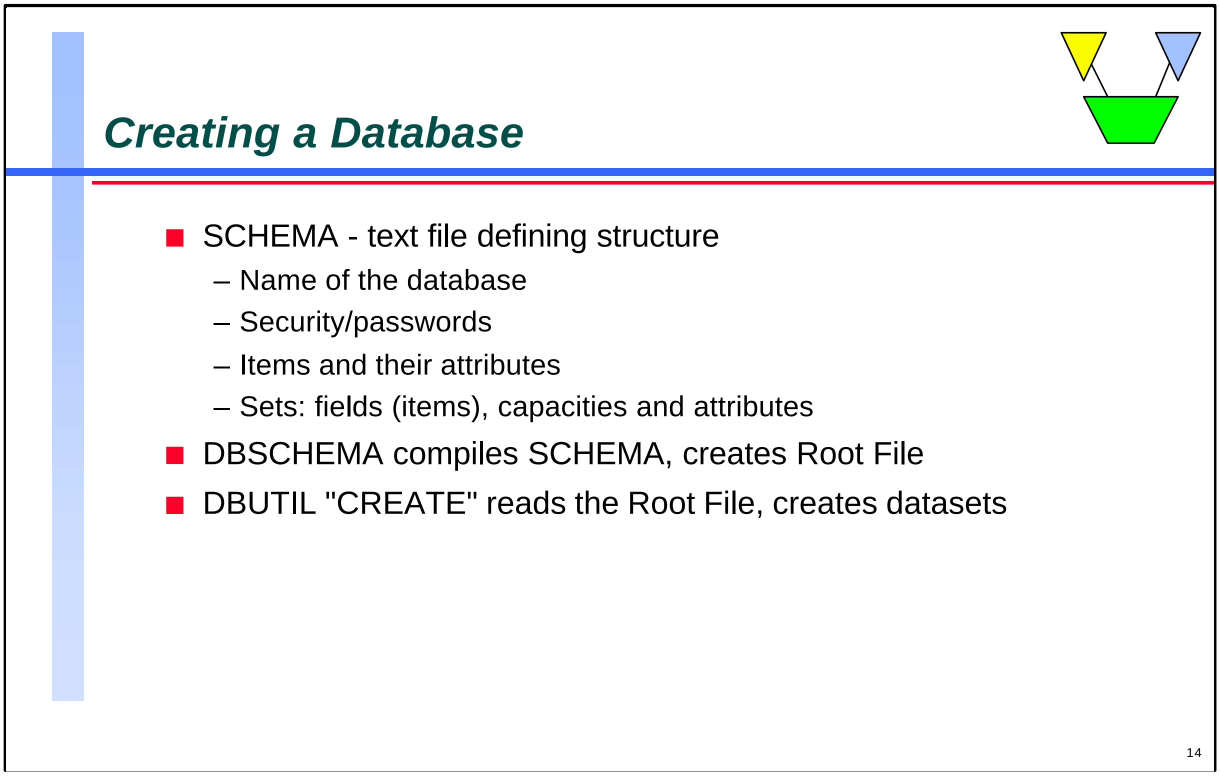
Creating a Database

In this section

	Page
■ The schema	14
■ Items	18
■ Sets	19
■ Database design tips	21

For Techies

References



Creating a Database

- SCHEMA - text file defining structure
  - Name of the database
  - Security/passwords
  - Items and their attributes
  - Sets: fields (items), capacities and attributes
- DBSCHEMA compiles SCHEMA, creates Root File
- DBUTIL "CREATE" reads the Root File, creates datasets

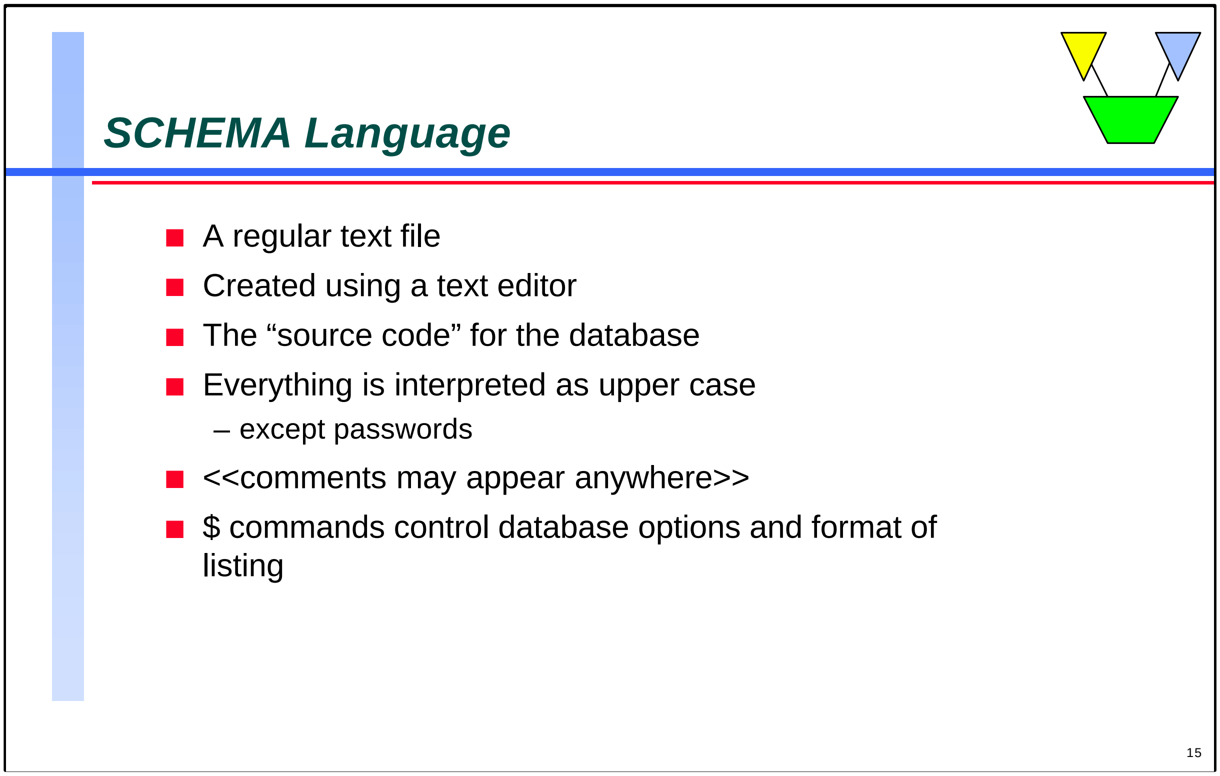
The schema of a database is used initially to define the database layout.

Later on in the life of a database, its layout can be altered using a database terraforming tool such as Adager or DBGeneral. These tools can generate a new schema file to match the altered database layout.

Many tools generate database structure listings in a format almost identical to a schema. For example, Query FORM command and Suprtool FORM command both produce schema-like output.

For Techies

References



SCHEMA Language

- A regular text file
- Created using a text editor
- The “source code” for the database
- Everything is interpreted as upper case
  - – except passwords
- <<comments may appear anywhere>>
- $ commands control database options and format of listing

The entire schema contents is interpreted as uppercase by the DBSCHEMA program. This is why you need to code uppercase dataset names and field names in your DB intrinsics.

Comments may appear anywhere except within other comments. A comment can start on one line and end on another.

$PAGE

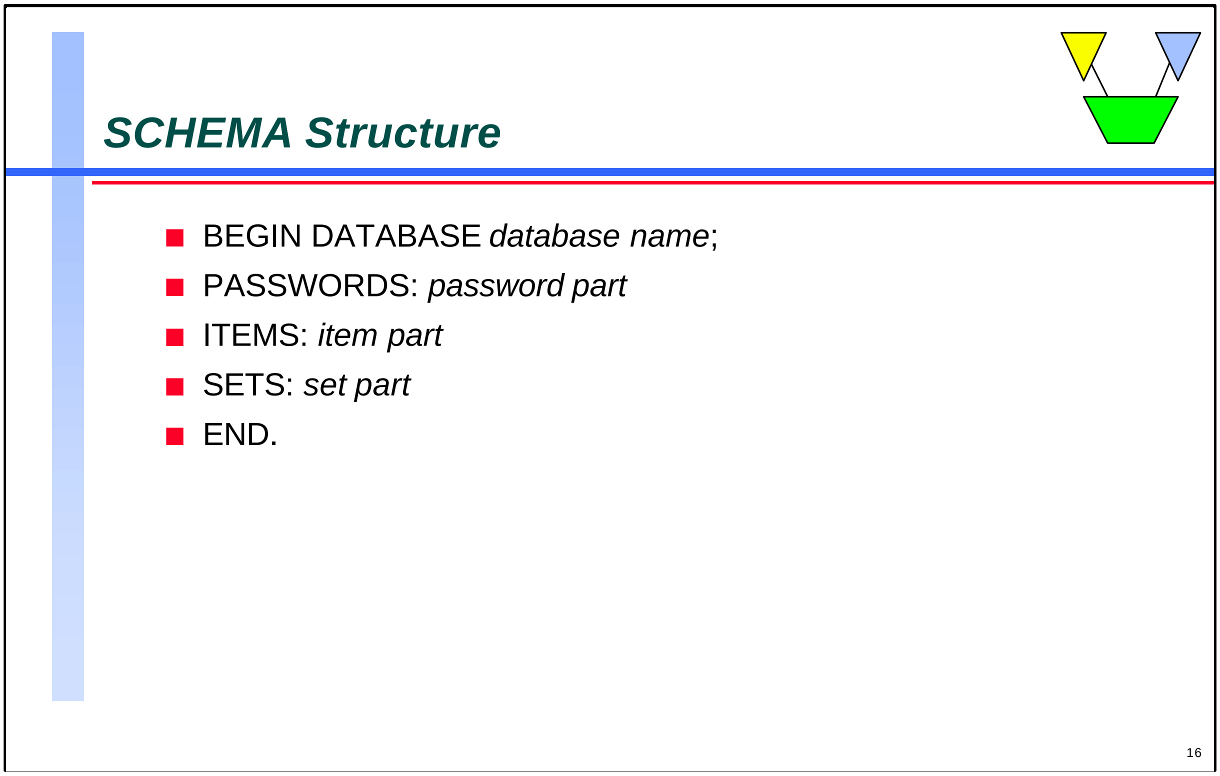
$TITLE

$CONTROL

For Techies

References

$CONTROL options are detailed later in this section.



SCHEMA Structure

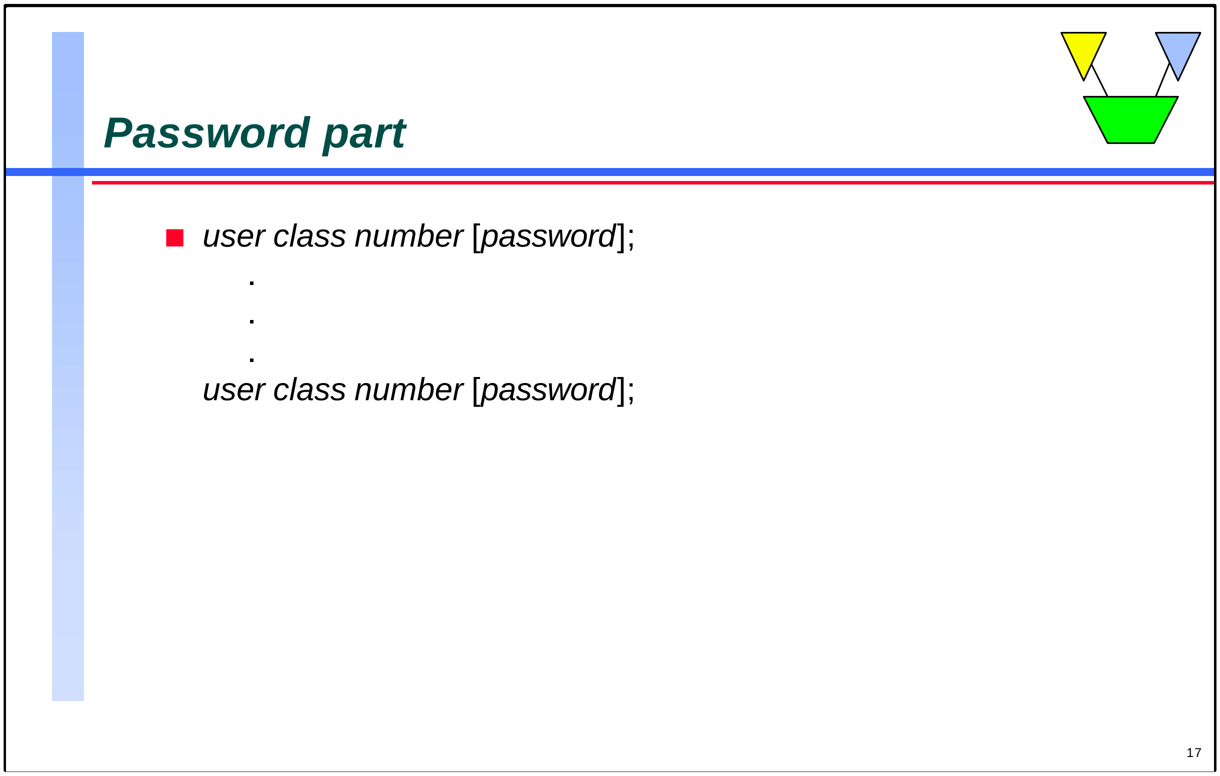
- BEGIN DATABASE *database name*;
- PASSWORDS: *password part*
- ITEMS: *item part*
- SETS: *set part*
- END.

The database name can be up to 6 alphanumeric characters, starting with a letter.

The password part, item part, and set part each repeat as many times as there are passwords, items, and sets.

For Techies

References



Password part

- *user class number* [*password*];

  .

  .

  .

  user class number [*password*];

Each user class number / password combination defines a user class, which is used later in the ITEMS and SETS parts to protect the items and sets .

Numbers range from 1 through 63 and are arbitrary. I.e, higher numbers don't grant more or less access than low numbers.

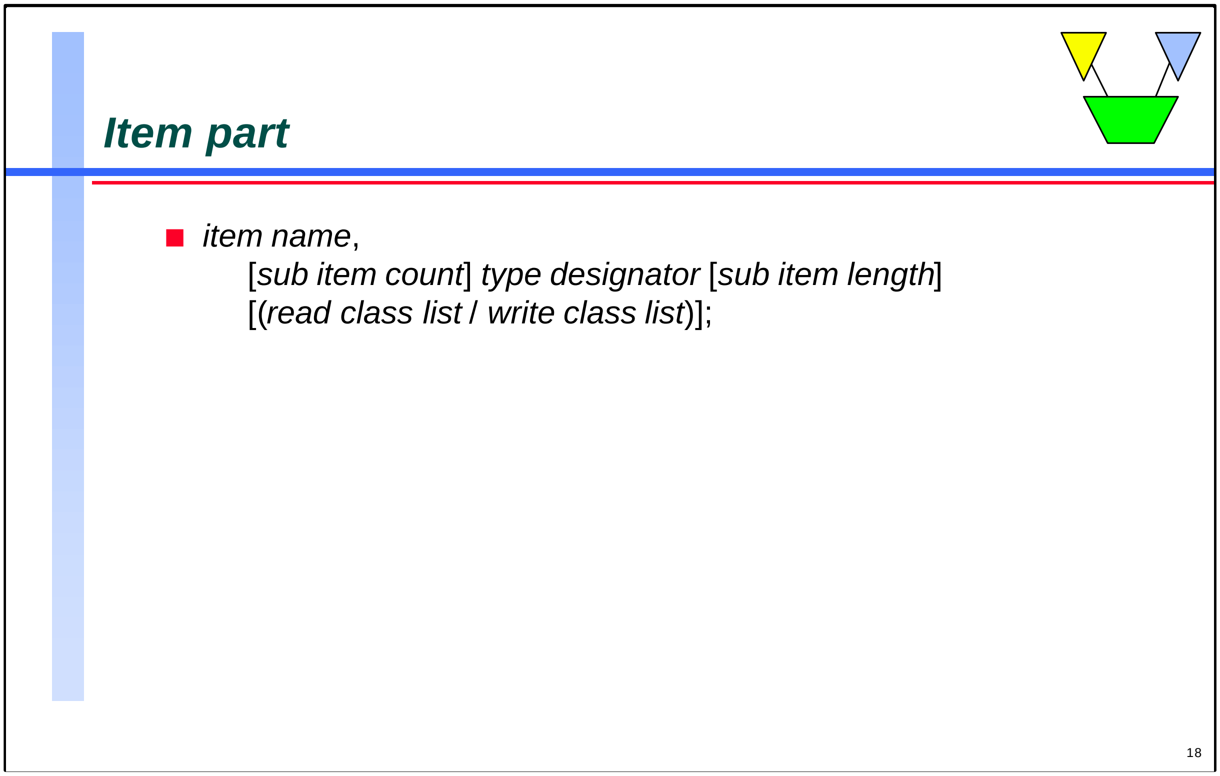
Passwords are from 1 to 8 characters including lowercase characters and excluding carriage return, slash, semicolon, and blank. Passwords are case sensitive. I.e., reader, Reader, and READER are all different passwords. The passwords are arbitrary and have no inherent meaning. I.e., a password of "Writer" may not grant write access at all.

```
PASSWORDS:     1 ReadOnly;
               2 WRITER;
               10 CREDIT;
               15 CLERKS;
               20 y2k;
               42 nmd643bc;
               55 Barney;
               63 DO-ALL;
```

For Techies

Use DBUTIL to change the passwords of existing databases.

References



Item part

- *item name*,
  [*sub item count*] *type designator* [*sub item length*]
  [(*read class list* / *write class list*)];

Repeat as needed, one per item.

```
SALES-TOTAL,              J2;
STATE-CODE,               X2 (1/2);
STREET-ADDRESS,          2X25;
SUPPLIER-NAME,            X20;
SUPPLIER-NO,              Z8;
UNIT-COST,                P8 (1,2,5/2);
POSTAL-CODE,              X6;
```

Data item names may be up to 16 characters long, starting with a letter. Characters after the first must be chosen from letters A through Z, digits 0 through 9, or + - * / ? ' # % & @

The names are not case sensitive.

Data type designators are E, I, J, K, P, R, U, X, and Z.

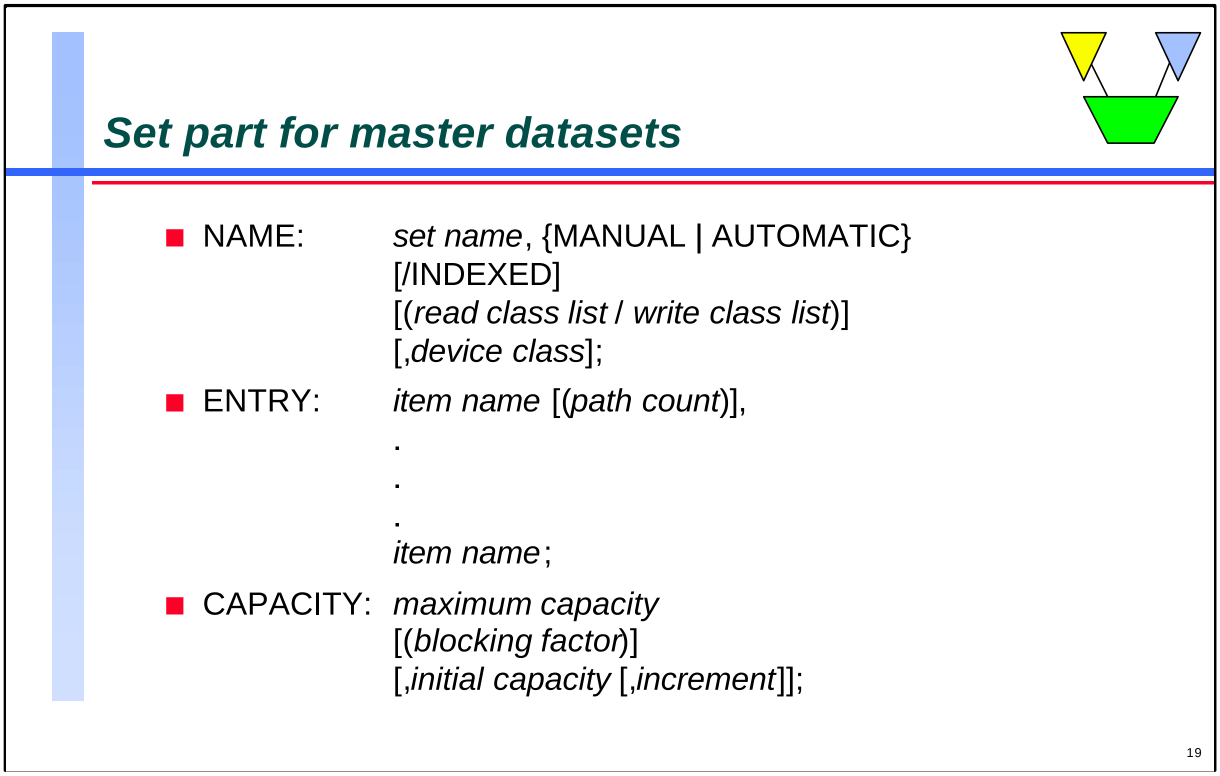
Unit	Size	Type
Word	16 bits	E, I, J, K, R
Byte	8 bits	U, X, Z
Nibble	4 bits	P

The item must always be a whole number of 16-bit words, or an even number of bytes. E.g., X25 is invalid, but X24 or X26 is okay, and 2X25 is okay, too.

For Techies

This tutorial lists a "word" as being 16 bits. Other HP documents may show a word as being 32 bits. The definition of a "word" has changed over the years as the hardware and operating system have advanced.

References



Set part for master datasets

- NAME: *set name*, {MANUAL | AUTOMATIC}
  [/INDEXED]
  [(*read class list* / *write class list*)]
  [,*device class*];
- ENTRY: *item name* [(*path count*)],
  .
  .
  .
  item name;
- CAPACITY: *maximum capacity*
  [(*blocking factor*)]
  [,*initial capacity* [,*increment*]];

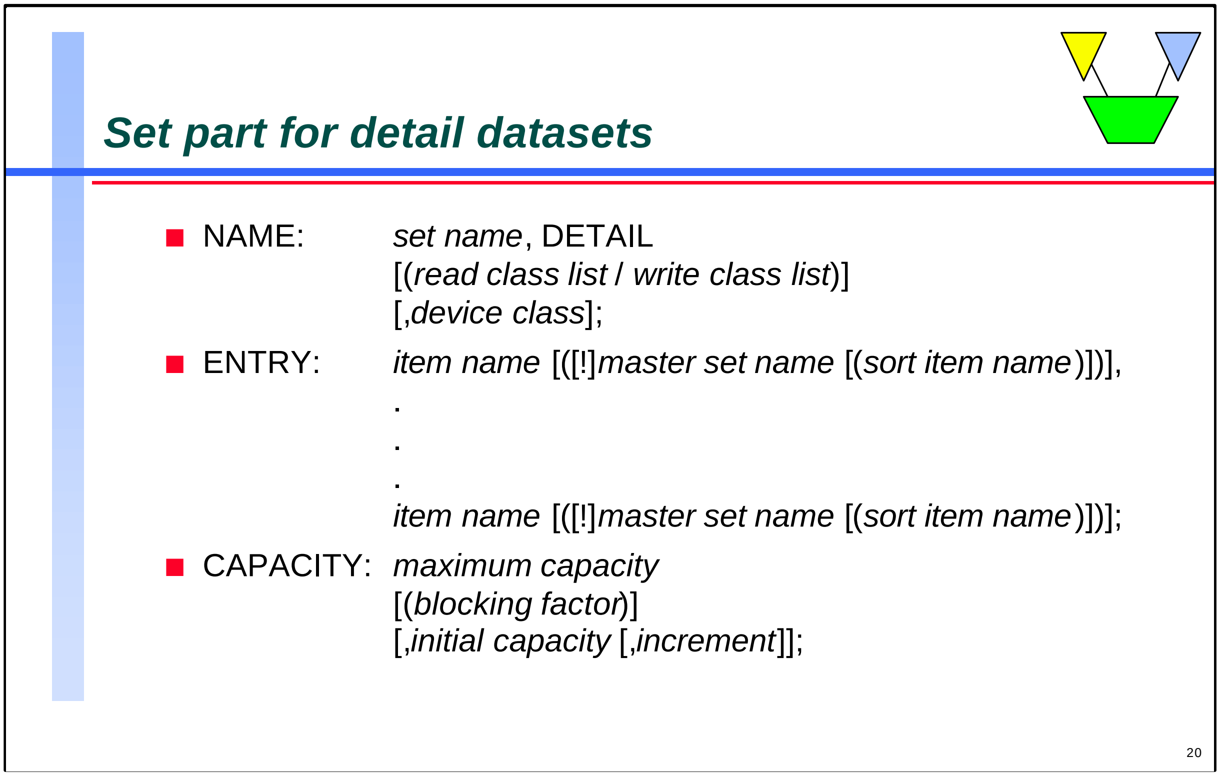
The ENTRY lists the items that make up the field list of the record. An automatic master has only one field. A manual master has up to 255 fields. Exactly one field must be designated as the key, indicated by the (path count). A path count of zero indicates a stand-alone manual master, not linked to any detail datasets.

```
NAME:  M-SUPPLIER,       MANUAL (1/2);
ENTRY:
       SUPPLIER-NAME
      ,STREET-ADDRESS
      ,CITY
      ,STATE-CODE
      ,POSTAL-CODE
      ,SUPPLIER-NO(1)       <<KEY FIELD>>
      ;
CAPACITY:  211;

NAME:  A-DATE,           AUTOMATIC (1/2);
ENTRY:
       PURCH-DATE(5)
      ;
CAPACITY:  10007,5003,25%;
```

The increment can be an absolute number or a percentage of the initial capacity.

For Techies

References



Set part for detail datasets

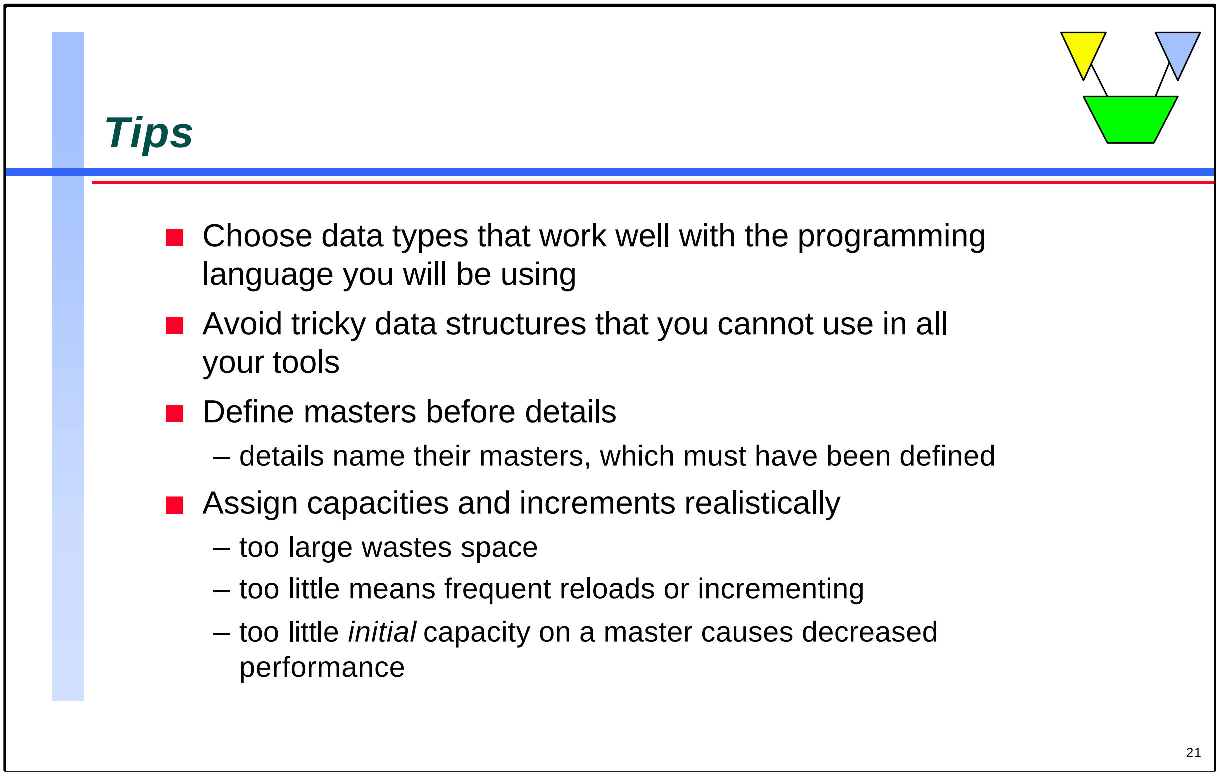
- NAME: *set name*, DETAIL
  [(*read class list* / *write class list*)]
  [,*device class*];
- ENTRY: *item name* [([!]*master set name* [(*sort item name*)])],
  .
  .
  .
  item name [([!]*master set name* [(*sort item name*)])];
- CAPACITY: *maximum capacity*
  [(*blocking factor*)]
  [,*initial capacity* [,*increment*]];

A detail dataset may have up to 255 fields. Up to 16 fields may be designated as search items, linked back to master datasets.

```
NAME:  D-INVENTORY,      DETAIL (1/2);
ENTRY:
       BIN-NO
      ,LAST-SHIP-DATE
      ,ON-HAND-QTY
      ,PRODUCT-NO(M-PRODUCT)    <<KEY FIELD>>
      ,SUPPLIER-NO(!M-SUPPLIER) <<PRIMARY KEY>>
      ,UNIT-COST
      ;
CAPACITY:  450;     <<2 * CAP(M-SUPPLIER)>>
```

For Techies

References

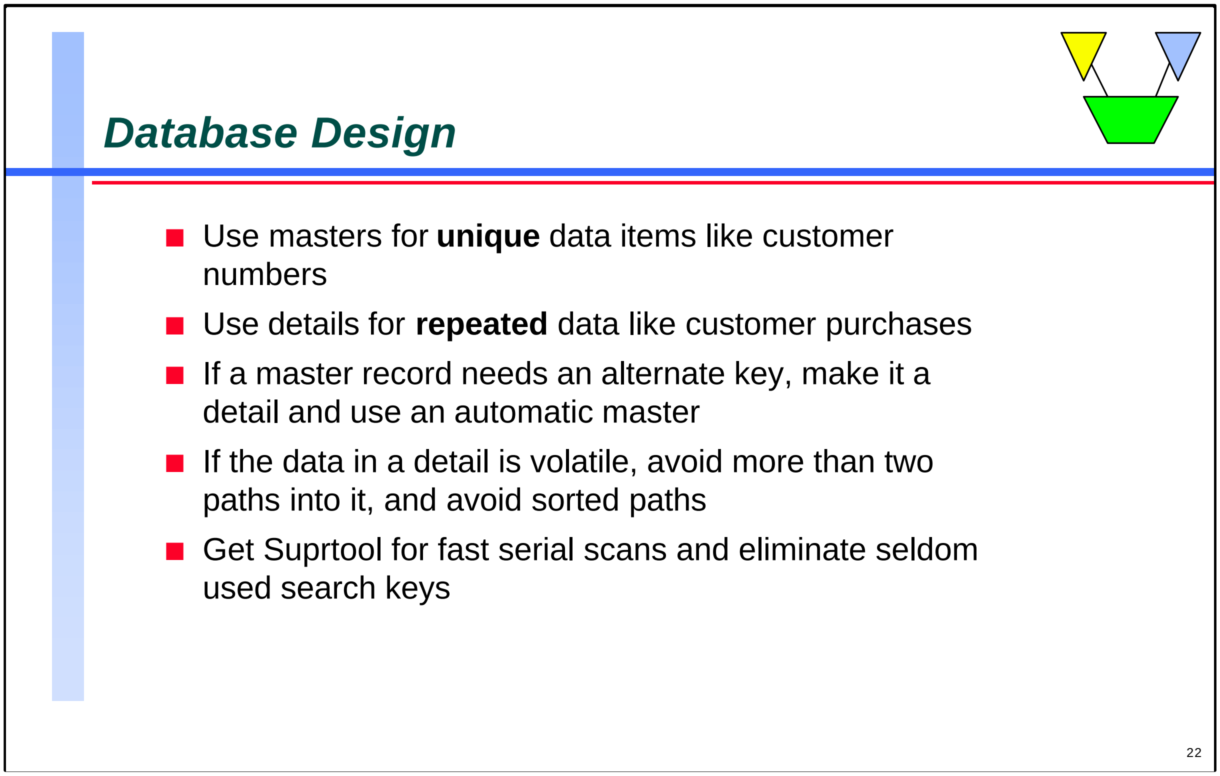


Tips

- Choose data types that work well with the programming language you will be using
- Avoid tricky data structures that you cannot use in all your tools
- Define masters before details
  - details name their masters, which must have been defined
- Assign capacities and increments realistically
  - too large wastes space
  - too little means frequent reloads or incrementing
  - too little *initial* capacity on a master causes decreased performance

For Techies

References



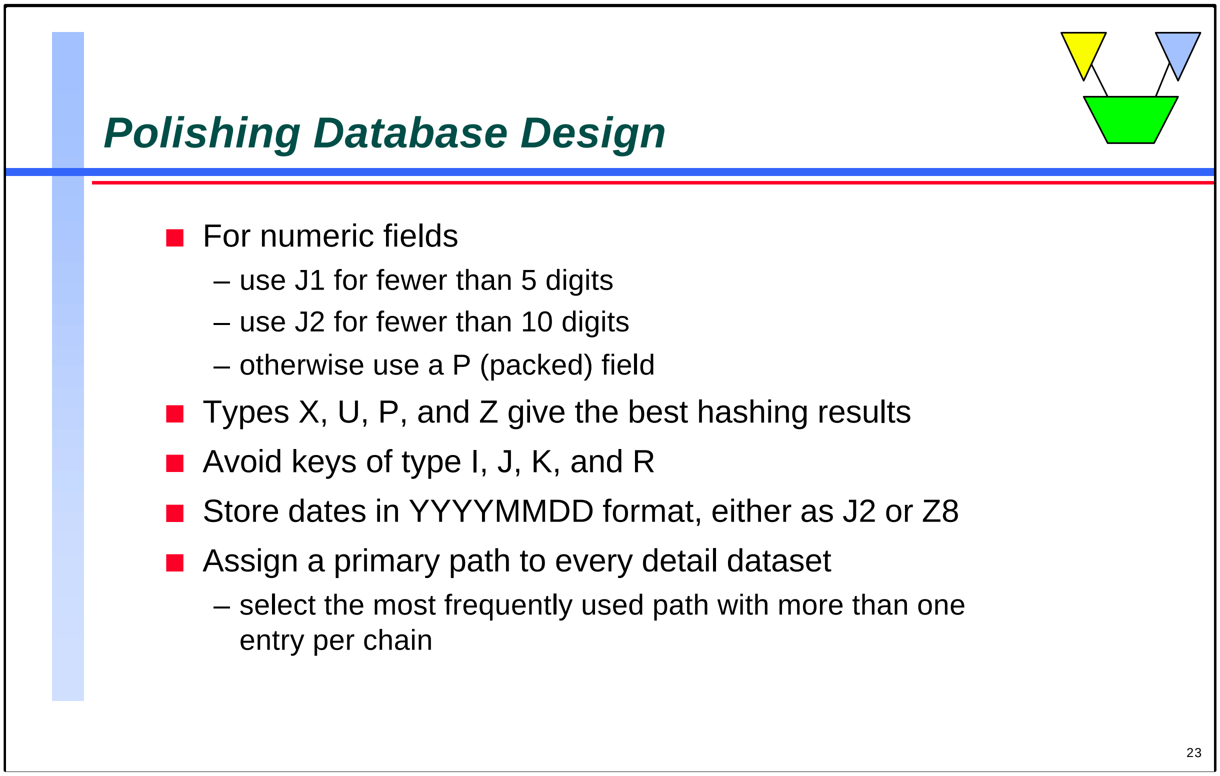
Database Design

- Use masters for **unique** data items like customer numbers
- Use details for **repeated** data like customer purchases
- If a master record needs an alternate key, make it a detail and use an automatic master
- If the data in a detail is volatile, avoid more than two paths into it, and avoid sorted paths
- Get Suprtool for fast serial scans and eliminate seldom used search keys

For Techies

References

See Adager's web site for good discussions on database design: http://www.adager.com/TechnicalPapers.html



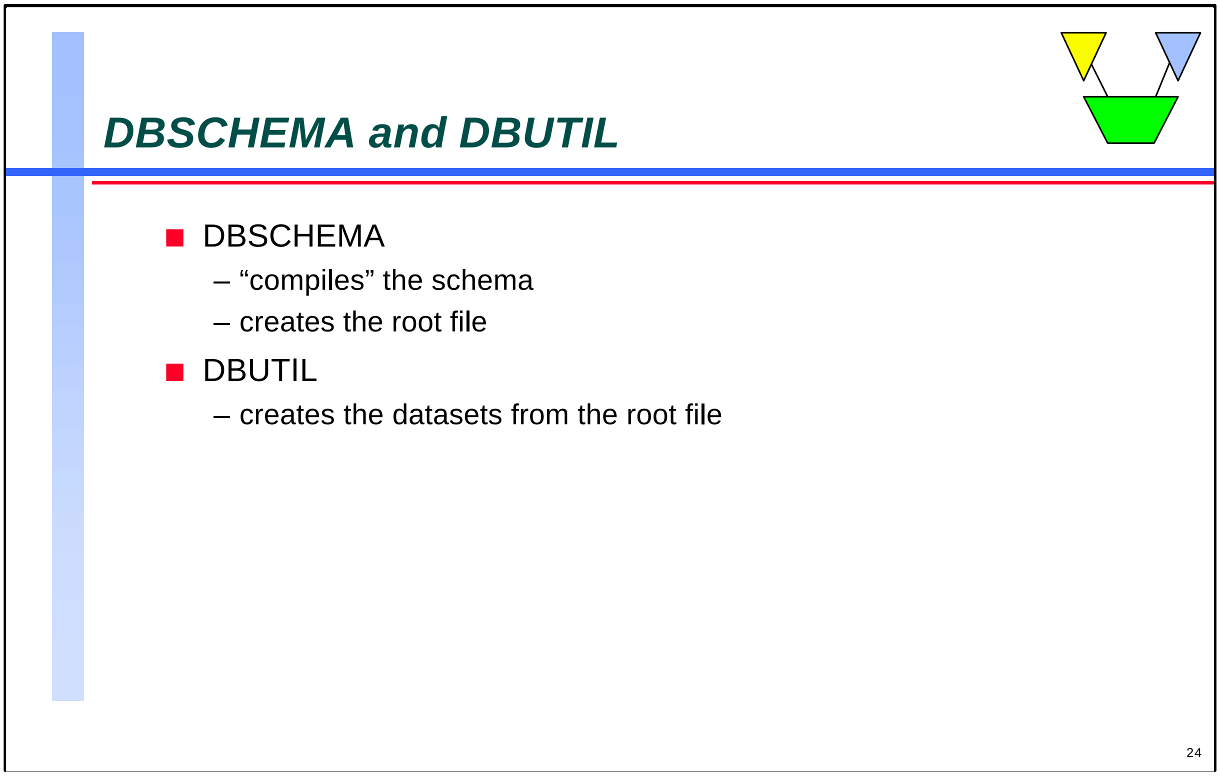
Polishing Database Design

- For numeric fields
  - – use J1 for fewer than 5 digits
  - – use J2 for fewer than 10 digits
  - – otherwise use a P (packed) field
- Types X, U, P, and Z give the best hashing results
- Avoid keys of type I, J, K, and R
- Store dates in YYYYMMDD format, either as J2 or Z8
- Assign a primary path to every detail dataset
  - – select the most frequently used path with more than one entry per chain

For Techies

References

See Robelle’s web site for a wealth of IMAGE and MPE information: http://www.robelle.com/library/smugbook/mpetips.html



DBSCHEMA and DBUTIL

- DBSCHEMA
  - “compiles” the schema
  - creates the root file
- DBUTIL
  - creates the datasets from the root file

$PAGE “this is the title of a new page starting now”

$TITLE “this will be the title for the next page eject”

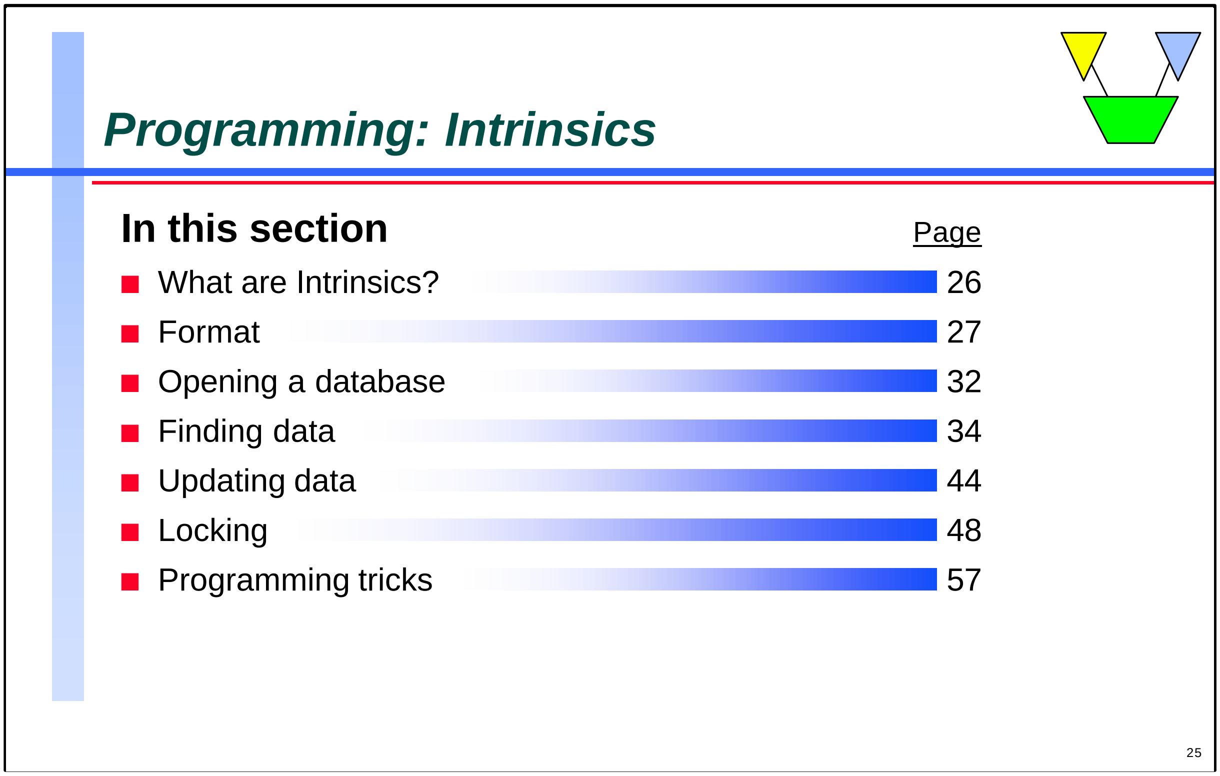
$CONTROL LIST | NOLIST
ERRORS=*n*
LINES=*nn*
ROOT | NOROOT
BLOCKMAX=*nnn*
TABLE | NOTABLE
JUMBO | NOJUMBO

```
!file dbstext=filename
!file dbslist;dev=lp
!run dbschema.pub.sys;parm=3
!
!run dbutil.pub.sys
create basename
exit
```

For Techies

When creating the database, make sure you are logged on as the user who will be the creator of the database, in the group where you want the database created. Later, only this creator user will be able to use DBUTIL on this database.

References



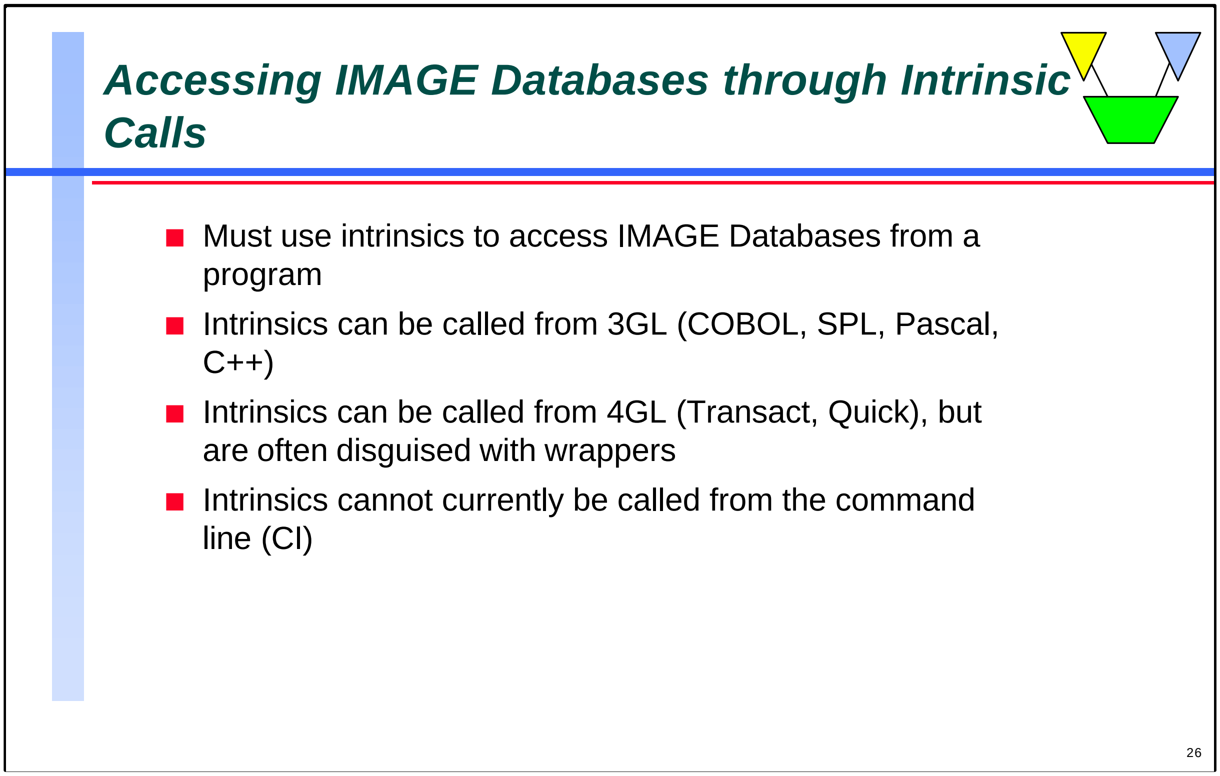
Programming: Intrinsics

In this section

	Page
What are Intrinsics?	26
Format	27
Opening a database	32
Finding data	34
Updating data	44
Locking	48
Programming tricks	57

For Techies

References



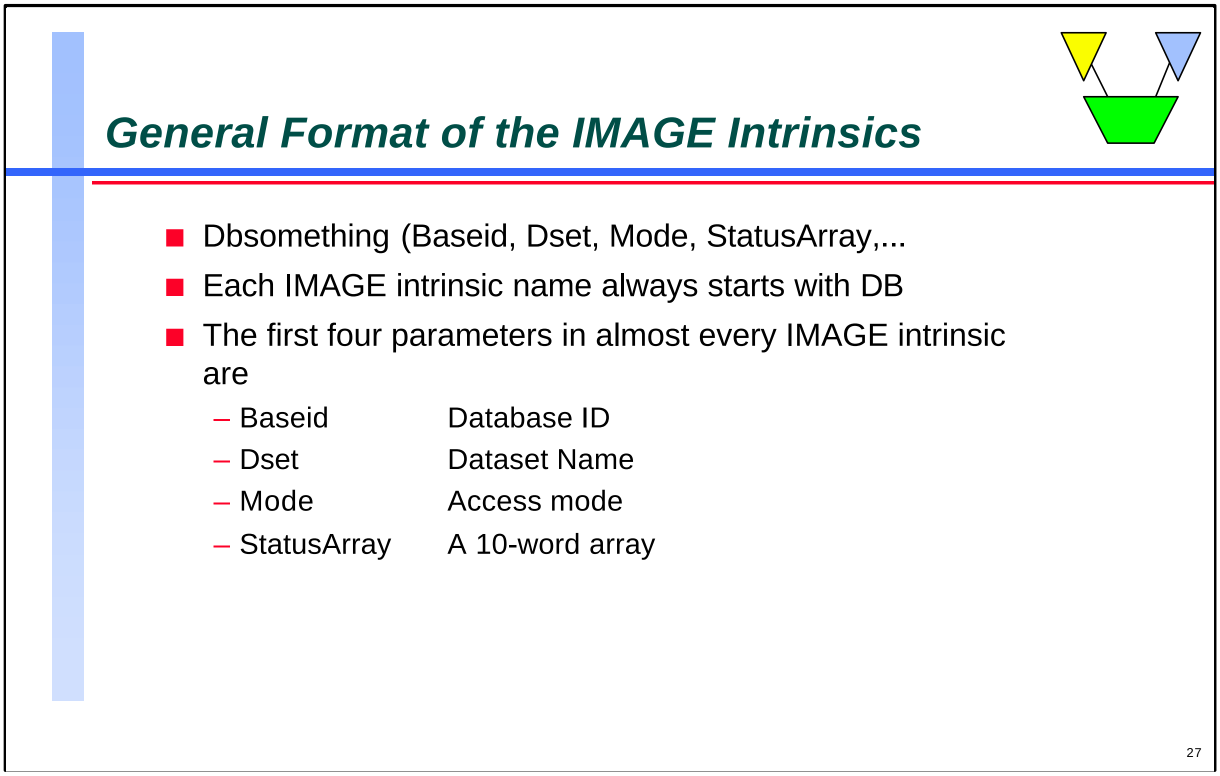
Accessing IMAGE Databases through Intrinsic Calls

- Must use intrinsics to access IMAGE Databases from a program
- Intrinsics can be called from 3GL (COBOL, SPL, Pascal, C++)
- Intrinsics can be called from 4GL (Transact, Quick), but are often disguised with wrappers
- Intrinsics cannot currently be called from the command line (CI)

Intrinsics are procedures that can be called from a programming language. On other operating systems these might be known as procedures, subroutines, methods, APIs, etc. HP calls them “intrinsics”.

For Techies

References



General Format of the IMAGE Intrinsics

- Dbsomething (Baseid, Dset, Mode, StatusArray,...
- Each IMAGE intrinsic name always starts with DB
- The first four parameters in almost every IMAGE intrinsic are

– Baseid	Database ID
– Dset	Dataset Name
– Mode	Access mode
– StatusArray	A 10-word array

All parameters must begin on a word boundary.

In COBOL, it’s best to use 01 levels, with Sync32 option set on in the compile.

```
$OPTIONS SYNC32

01  BASE.
      05  base-id       pic x(2).
      05  base-name     pic x(8).
```

As pointed out previously, all IMAGE structures begin on a word boundary.

For Techies

As used here, a word is 16 bits, and a doubleword is 32 bits.

References

The BaseID

- First parameter for each intrinsic is the Baseid.
  - bytes 1 and 2 Initially spaces
  - bytes 3 through 9 The database name
    Up to 6 characters plus a terminator.
- First two bytes assigned a unique value by IMAGE after database is opened.
- The databasename must end in a semicolon or space. E.g., “`FOO;`”

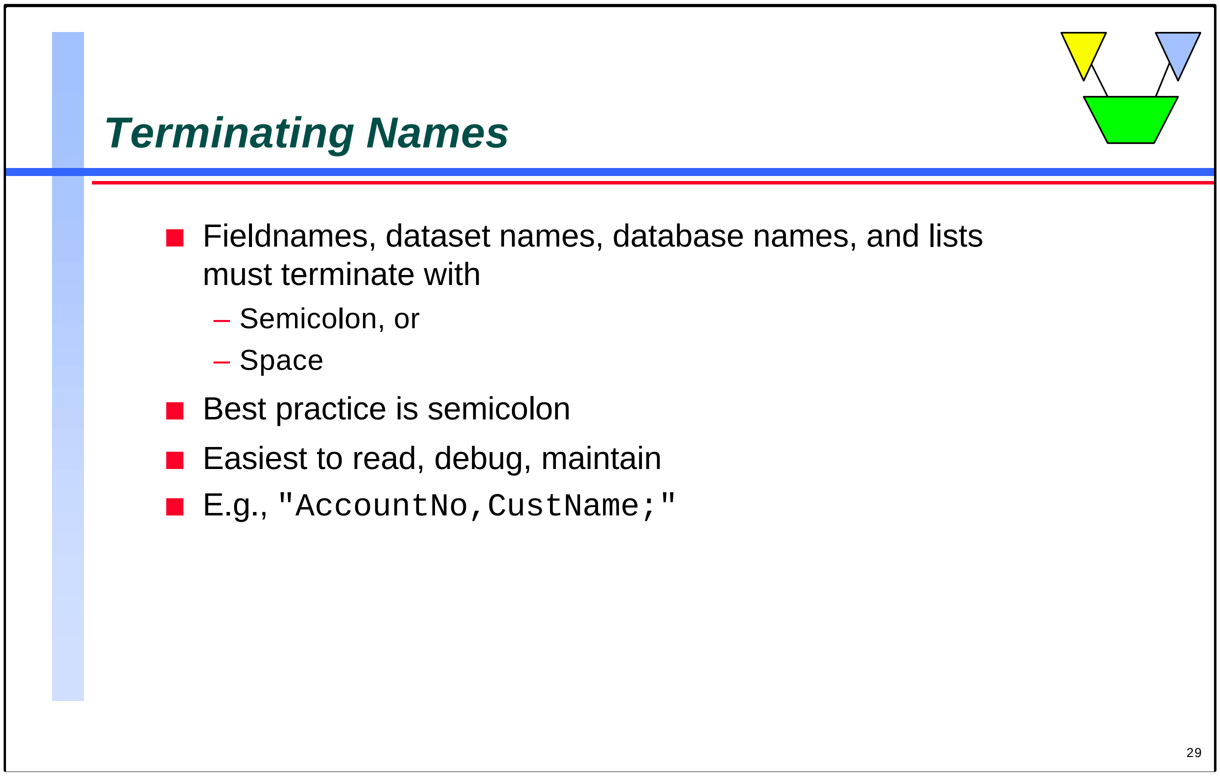
You may end the database name with a semicolon or a space. Space is not recommended because of programming maintenance headaches! It’s a lot easier to spot a semi-colon than a space.

You can pass the database variable to other objects within your process.

One example where this is useful is when your program has COBOL subprograms. You pass the base from subprogram to subprogram via the Linkage Section.

For Techies

References



Terminating Names

- Fieldnames, dataset names, database names, and lists must terminate with
  - Semicolon, or
  - Space
- Best practice is semicolon
- Easiest to read, debug, maintain
- E.g., `"AccountNo,CustName;"`

Dataset and Item names are alphanumeric and can be up to 16 characters long The only special character allowed is a dash.

Eg.

M-CUSTOMER

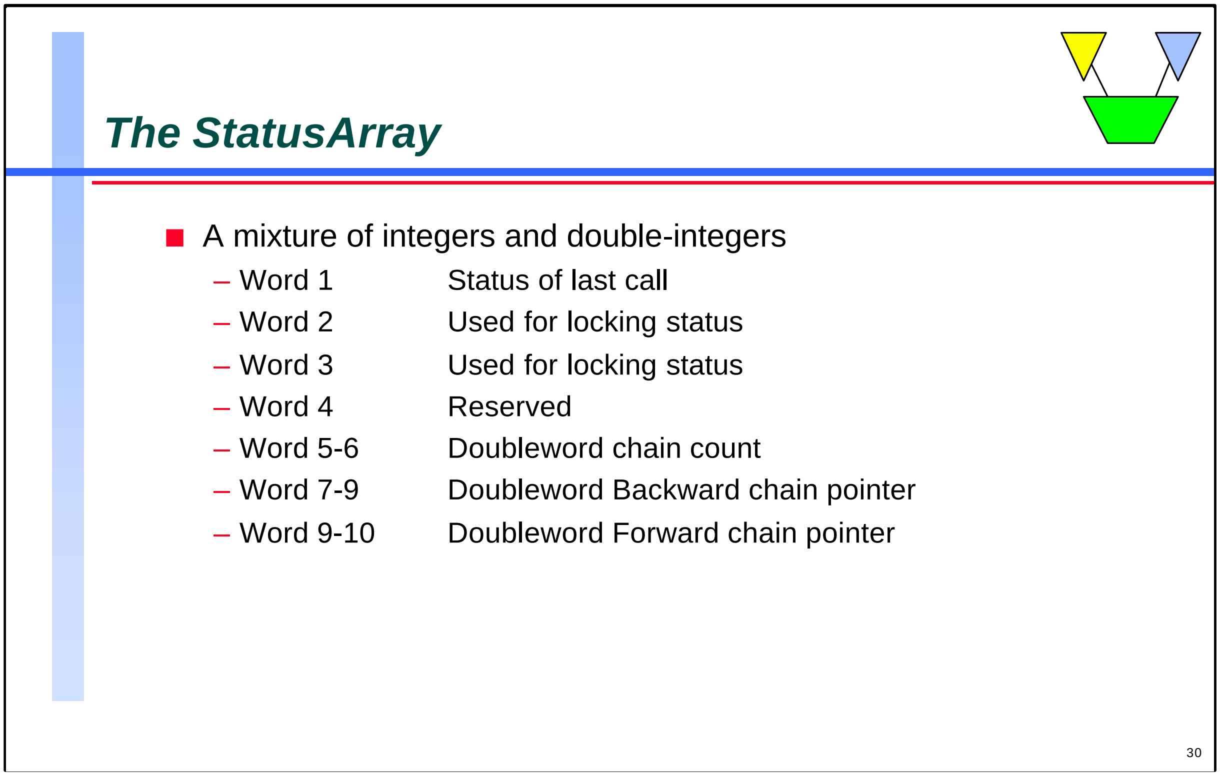
D-PRODUCTS

ACCOUNT-NO

IMAGE always upshifts the names in the list. You can call the intrinsics with mixed case if you'd like.

For Techies

References



The StatusArray

- A mixture of integers and double-integers

– Word 1	Status of last call
– Word 2	Used for locking status
– Word 3	Used for locking status
– Word 4	Reserved
– Word 5-6	Doubleword chain count
– Word 7-9	Doubleword Backward chain pointer
– Word 9-10	Doubleword Forward chain pointer

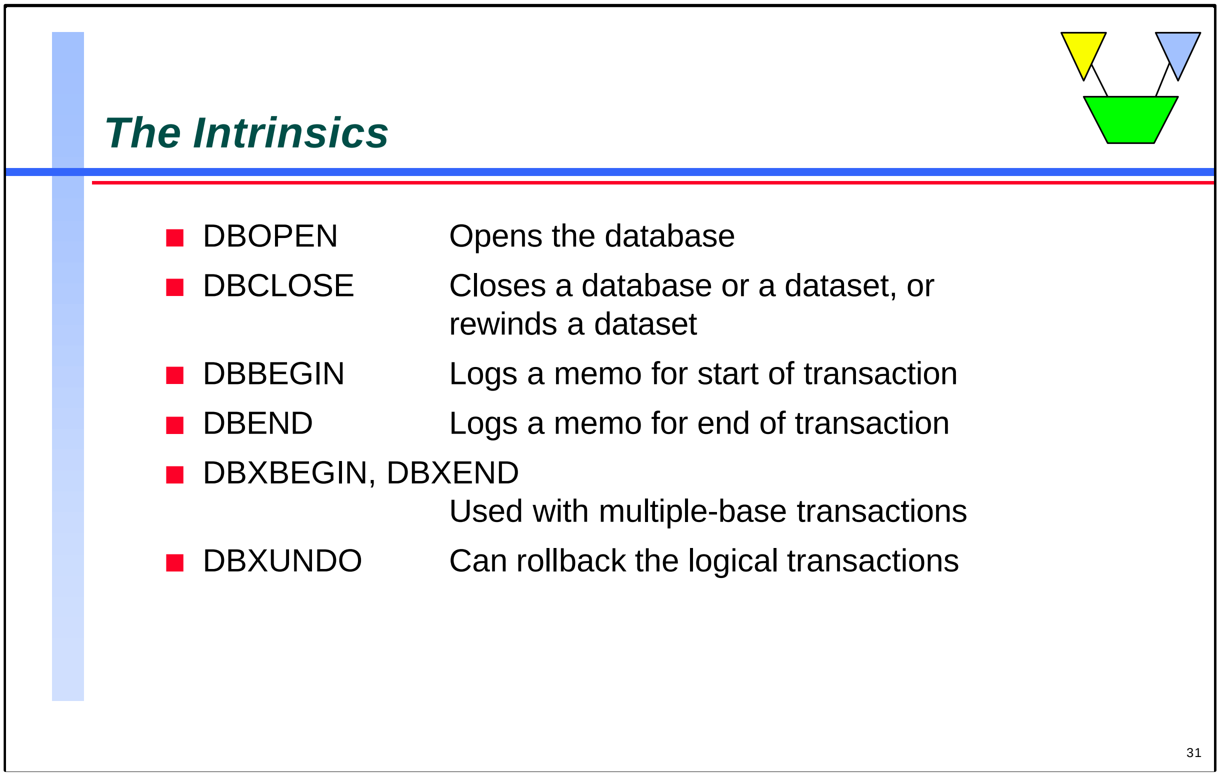
30

You’ll be using the the first word in the status array the most!

IMAGE will always return a value of zero in the first word of the status array IF the completion of the intrinsic is successful.

For Techies

References



The Intrinsics

- DBOPEN Opens the database
- DBCLOSE Closes a database or a dataset, or rewinds a dataset
- DBBEGIN Logs a memo for start of transaction
- DBEND Logs a memo for end of transaction
- DBXBEGIN, DBXEND
  Used with multiple-base transactions
- DBXUNDO Can rollback the logical transactions

Opening a database is expensive in terms of I/O....don’t do it more often than you have to.

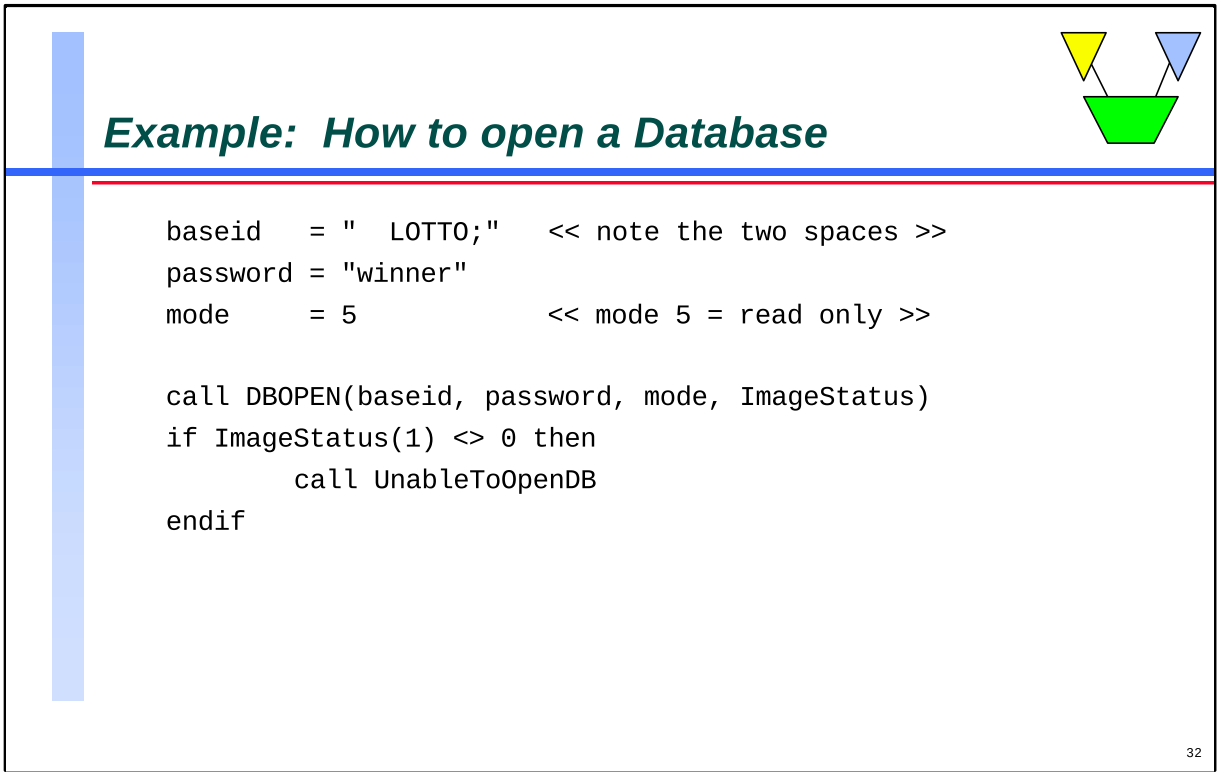
The DBXBEGIN, END and UNDO intrinsics allow you create logical transactions with multiple databases.

For Techies

Planning and programming multiple-db transactions can be tricky. Read up on the subject before attempting this!

References

TurboImage/XL Database Management System Reference Manual: see TurboIMAGE/XL Library Procedures... DBXBEGIN



Example: How to open a Database

```
baseid   = "  LOTTO;"   << note the two spaces >>
password = "winner"
mode     = 5            << mode 5 = read only >>

call DBOPEN(baseid, password, mode, ImageStatus)
if ImageStatus(1) <> 0 then
        call UnableToOpenDB
endif
```

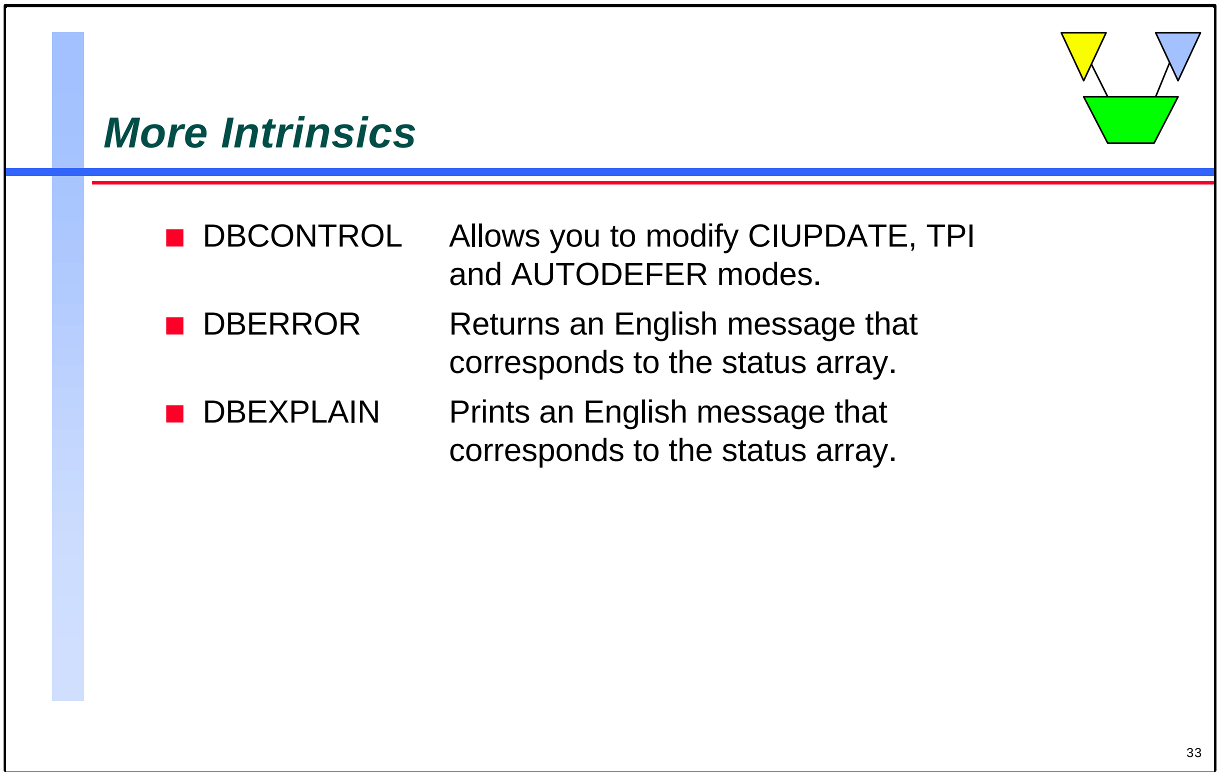
Basic Data Base Open Modes

Mode	Access
1	Modify, allow concurrent modify
2	Update, allow concurrent update
3	Read and write, Exclusive Access !!!
4	Modify, allow concurrent read
5	Read, allow concurrent modify
6	Read, allow concurrent modify
7	Read, exclusive access !!!
8	Read, allow concurrent read

Modes 1, 3 and 5 are the most commonly used.

For Techies

References



More Intrinsics

- DBCONTROL Allows you to modify CIUPDATE, TPI and AUTODEFER modes.
- DBERROR Returns an English message that corresponds to the status array.
- DBEXPLAIN Prints an English message that corresponds to the status array.

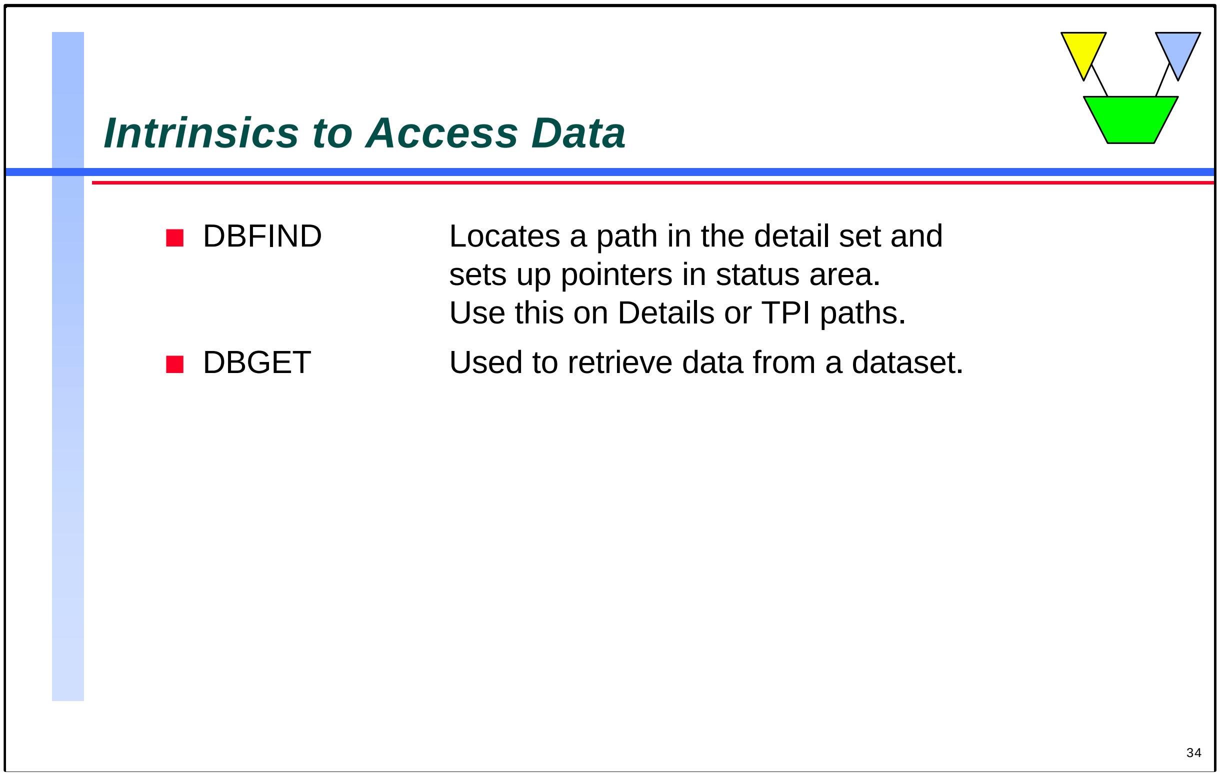
DBEXPLAIN prints an error message right to your terminal session. This isn’t very useful for block-mode type programs, such as those using VPLUS.

DBERROR is great when you need to get the error message into a variable for displaying. Allow for a very long message...at least 256 bytes.

For Techies

Use DBCONTROL Mode 7 to activate Database Deadlock Protection. This only works in MPE/iX 6.0 and greater.

References



Intrinsics to Access Data

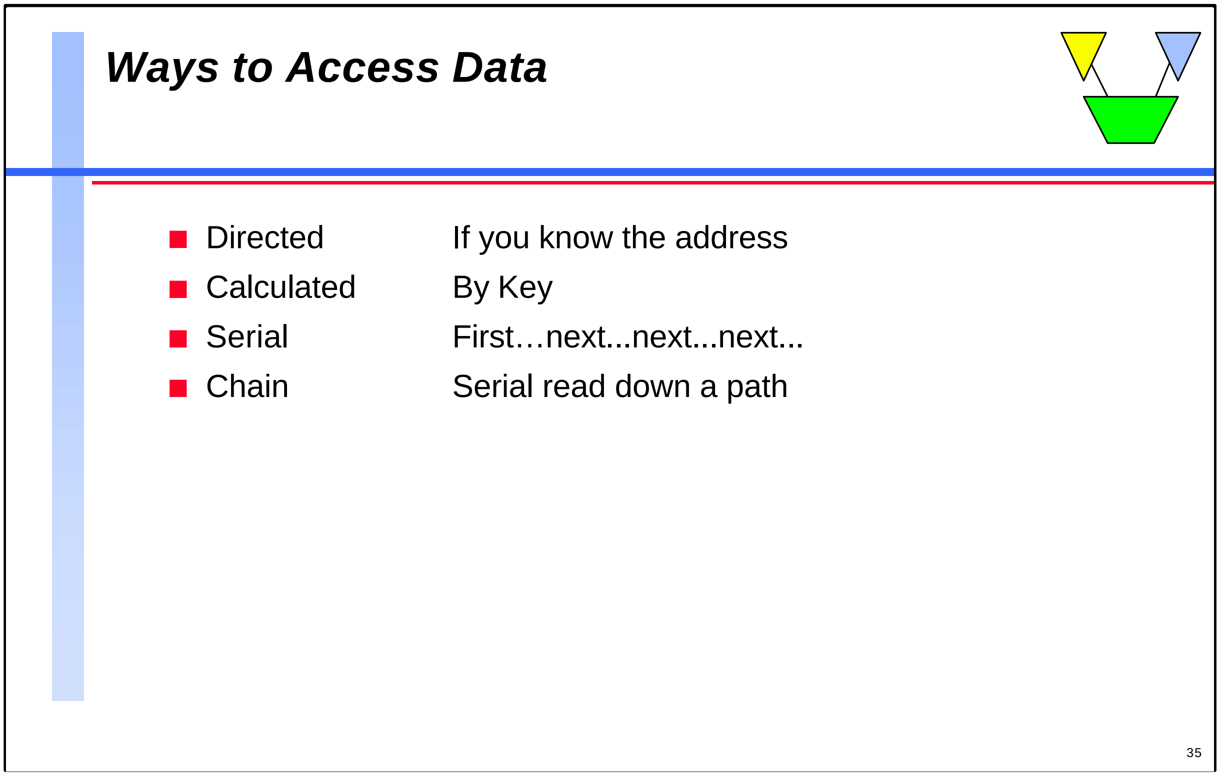
- DBFIND — Locates a path in the detail set and sets up pointers in status area. Use this on Details or TPI paths.
- DBGET — Used to retrieve data from a dataset.

You can use DBFIND on IMAGE paths in DETAIL sets, or TPI keys in DETAIL and MASTER sets.

You can also use DBFIND to search B-TREE structures in MASTER sets.

For Techies

References



Ways to Access Data

Directed	If you know the address
Calculated	By Key
Serial	First…next...next...next...
Chain	Serial read down a path

Directed reads are only useful when you know that the location of the data is NOT going to change.

Other processes running against the database may change the location of the data..

For Techies

References

A Quick Look at DBFIND

- DBFIND(baseid, dset, mode, ImageStatus, item,
  argument)
  - item: The name of an item that is a path in the detail set
  - argument: The value of the item that you want to find
  - Mode is always 1
    EXCEPT for TPI and B-TREE

DBFIND is used to locate *detail* dataset records by key. DBFIND does not return any data records to the program (that is done by DBGET). The format of the item and argument must match. If the item that you are searching is binary, then pass a binary reference.

Eg.

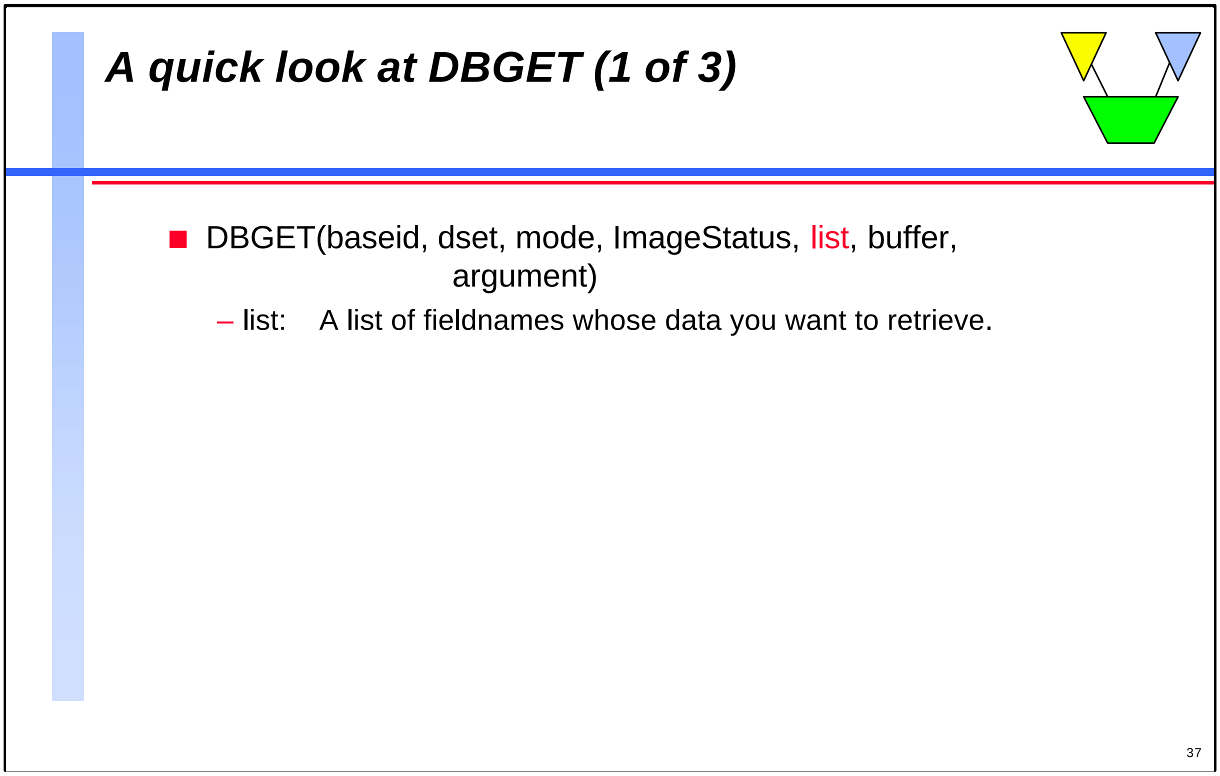
```
01  bn-day-of-year          pic x(16)
                            value “Day-Of-Year;”.
01  day-of-year        comp pic s9(4).

    call DBFIND using Baseid,
                      SetName,
                      Mode_1,
                      StatusArray,
                      bn-day-of-year,
                      day-of-year
```

For Techies

TPI modes can be found in the 3rd-Party Vendor documentation.

References

An excellent resource for B-TREE programming can be found in the TurboImage/XL Database Management System Reference Manual - MPE/iX 6.0



A quick look at DBGET (1 of 3)

- DBGET(baseid, dset, mode, ImageStatus, list, buffer,
  argument)
  - list: A list of fieldnames whose data you want to retrieve.

37

DBGET is used to return data records to your program. The data that is returned is determined by the list of fields named.

For Techies

References

A sidebar on List

- The List of fieldnames can be one of four things:
  - "@;"   All items
  - "*;"   Previous List
  - "field_name,field_name,...;"
  - an ordered list of item numbers (seldom used)

Some examples of lists....

“Account-no;”

“Company-name,Contact-Name;”

“@;”   (All items in the dataset record)

“*;”   (Use the list of names set up in a previous intrinsic call)

Using * is more efficient for subsequent calls to the same dataset, as IMAGE doesn’t have to parse and do error-checking against the list.

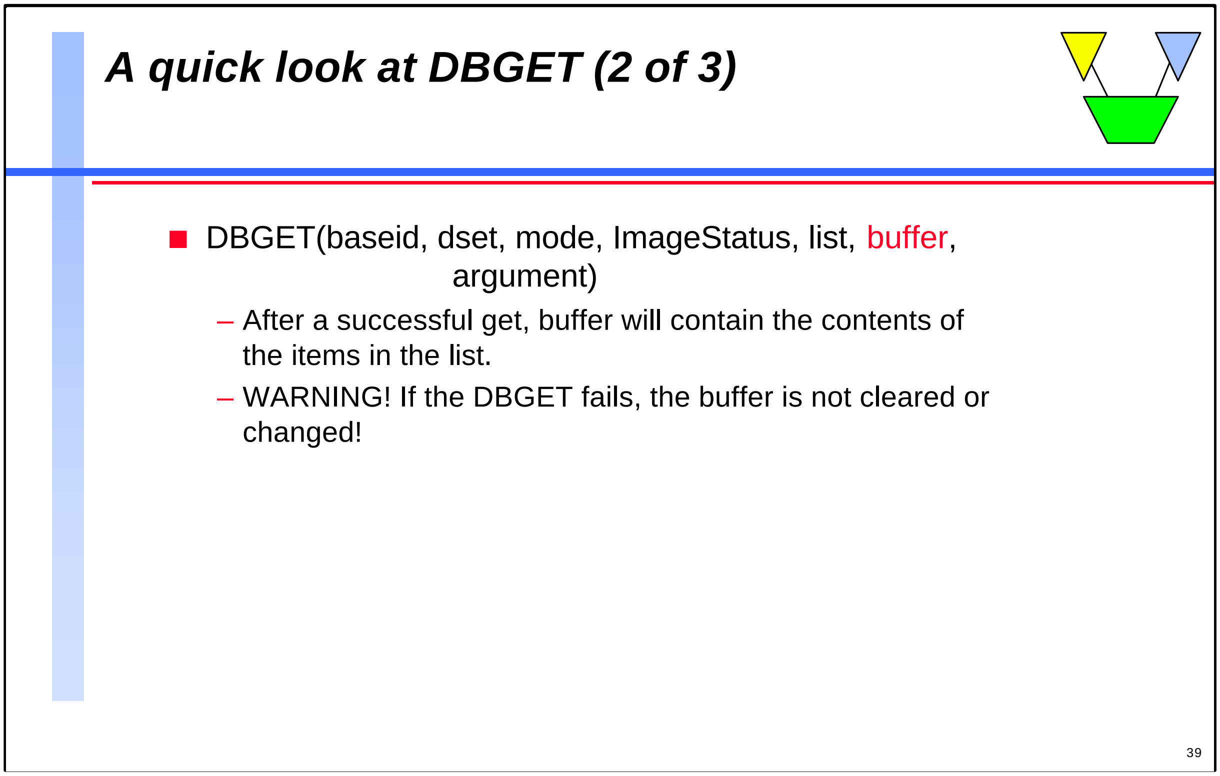
IMAGE will remember the previous list for each dataset!

WARNING! Make sure that you have already defined a list before using * - otherwise IMAGE will use a NULL list. Your DBPUTs will merrily return a IMAGE Status of zero - successful completion - when you didn’t really put anything to the set!

Eg: First call with dbget, list = “account-no,company-name;”
Next call with dbget, list=“*;”

For Techies

References



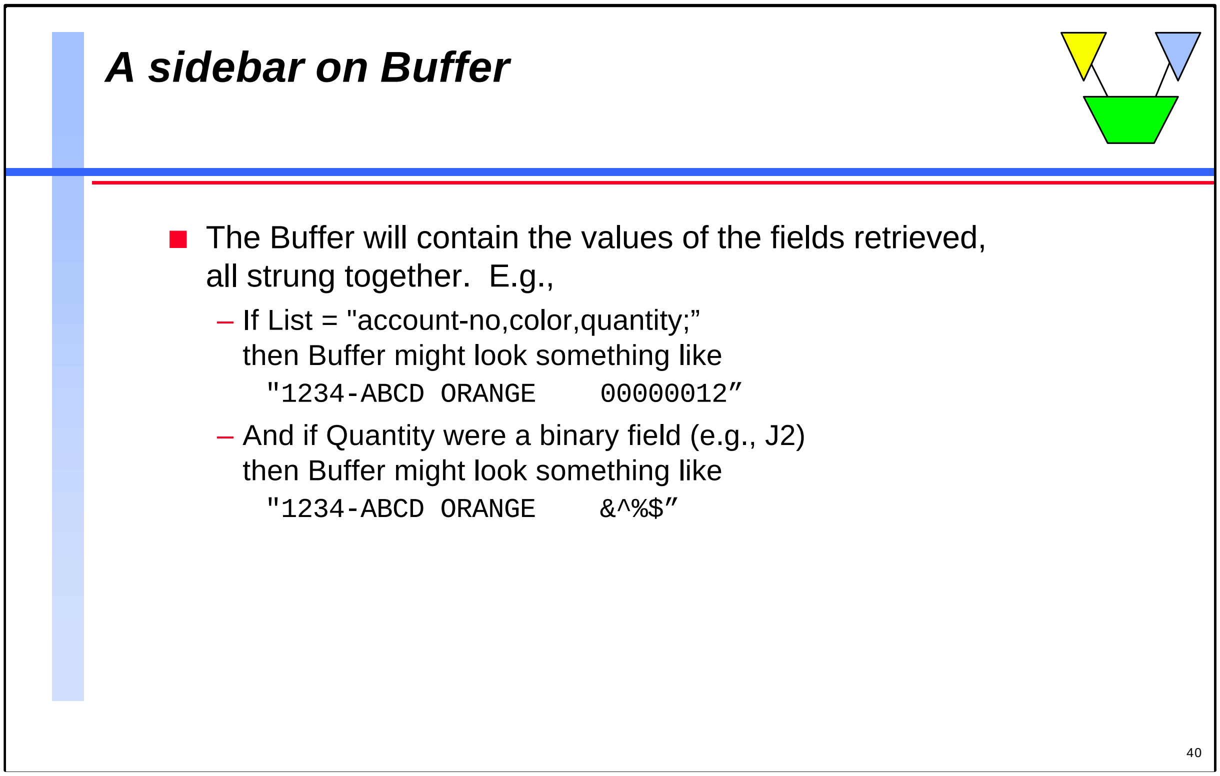
A quick look at DBGET (2 of 3)

- DBGET(baseid, dset, mode, ImageStatus, list, buffer, argument)
  - After a successful get, buffer will contain the contents of the items in the list.
  - WARNING! If the DBGET fails, the buffer is not cleared or changed!

WARNING! The buffer size must be at least as large as the amount of data you are expecting. If the buffer is too small, IMAGE will happily write the contents into the other memory areas that follow the buffer.

For Techies

References



A sidebar on Buffer

- The Buffer will contain the values of the fields retrieved, all strung together. E.g.,
  - If List = "account-no,color,quantity;" then Buffer might look something like
    ```
    "1234-ABCD ORANGE    00000012”
    ```
  - And if Quantity were a binary field (e.g., J2) then Buffer might look something like
    ```
    "1234-ABCD ORANGE    &^%$”
    ```

The buffer will contain the contents of all of the items in the list concatenated by their size. The data will be placed in the buffer according to the size and format defined in the schema. There are no field separators.

For Techies

References

A quick look at DBGET (3 of 3)

- DBGET(baseid, dset, mode, ImageStatus, list, buffer, argument)
  - argument: Contains the search value of the key field.
  - Used only for directed or calculated reads
    - Use the record # for Mode 4 and 8
    - Use the key value for Mode 7
    - Mode 7 is the most commonly used mode

DBGET Modes:

1	Re-Read
2	Serial Read
3	Backward Serial Read
4	Chained Read, or Next Qualified Entry Read (TPI and B-Trees)
5	Chained Read
6	Backward Chained Read
7	Calculated Read - Master Sets Only
8	Primary Calculated Read - Use with Caution The record you retrieve may not be the one you're looking for!

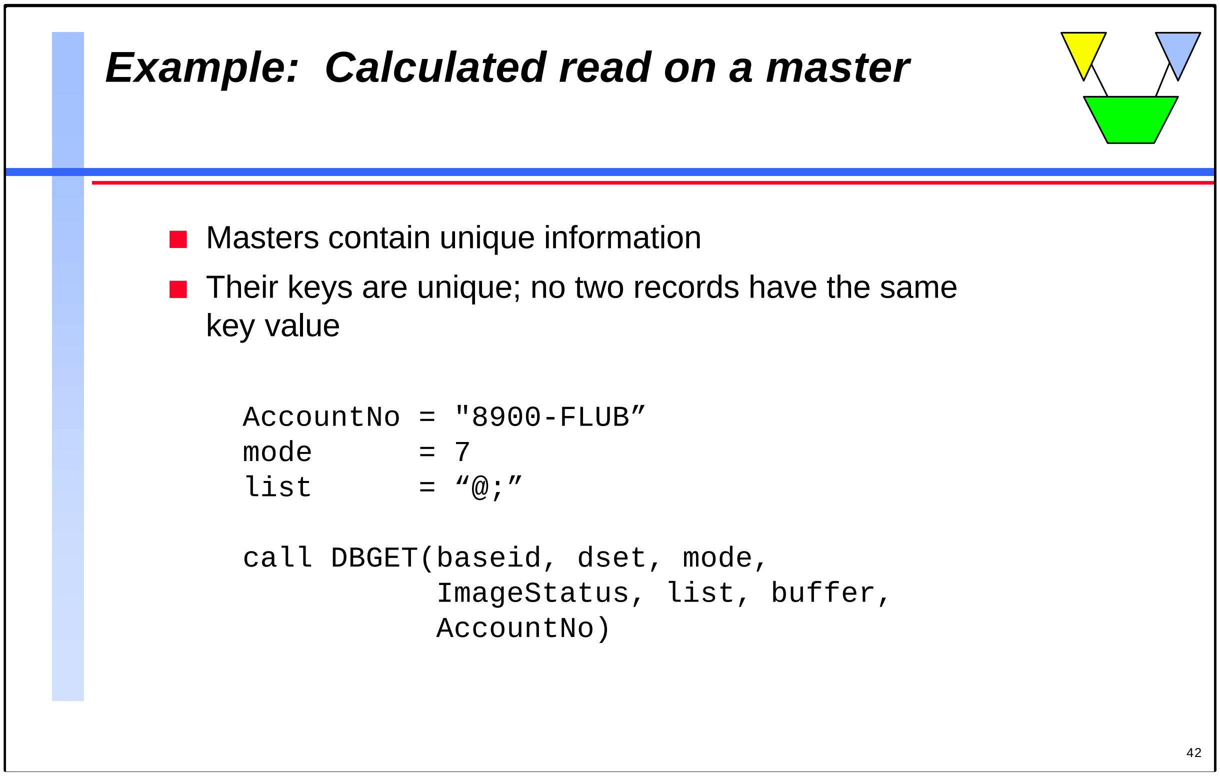
We mostly use modes 2 and 7 for Master sets, and modes 5 and 6 for Detail sets.

For Techies

Mode 8 DBGETs are useful when producing statistics on the health of your database. Use Mode 7 for everyday use.

References

Migrating Secondaries, in slide XX of this presentation.



Example: Calculated read on a master

- Masters contain unique information
- Their keys are unique; no two records have the same key value

```
AccountNo = "8900-FLUB”
mode      = 7
list      = “@;”

call DBGET(baseid, dset, mode,
           ImageStatus, list, buffer,
           AccountNo)
```

```
AccountNo = "8900-FLUB”
mode      = 7
list      = “@;”

call DBGET(baseid, dset, mode, ImageStatus,
           list, buffer, AccountNo)

if ImageStatus(1) = 0 then
   <<dbget retrieved 8900-FLUB>>
elsif ImageStatus(1) = 17 then
   <<dbget did not find 8900-FLUB>>
else
   <<dbget failed abnormally>>
endif
```

For Techies

References

Reading Up or Down a Chain of Details

- We need to FIND the chain head before we can read it.

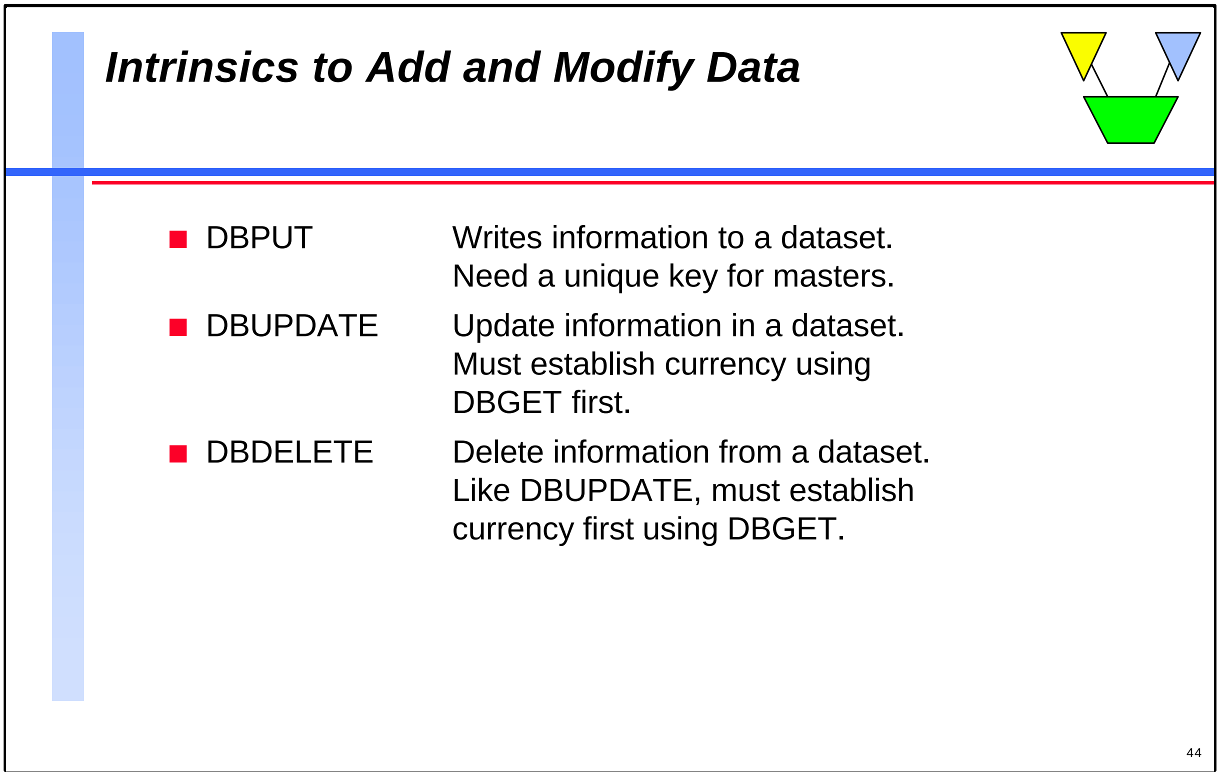
```
call DBFIND(baseid, dset, mode_1,
            ImageStatus, item, argument)
if ImageStatus(1) = 0 then
   call DBGET(baseid, dset,
              mode_direction,
              ImageStatus, list,
              buffer, dummy)
endif
```

- mode = 5 indicates to read the chain forwards.
- mode = 6 indicates to read the chain backwards.

Some elements in the ImageStatusArray will contain address information about the previous and next records in the chain. Changing these programmatically will have no effect - the ImageStatus array is one-way, from the calling intrinsic back to the program.

For Techies

References



Intrinsics to Add and Modify Data

- DBPUT — Writes information to a dataset. Need a unique key for masters.
- DBUPDATE — Update information in a dataset. Must establish currency using DBGET first.
- DBDELETE — Delete information from a dataset. Like DBUPDATE, must establish currency first using DBGET.

For Techies

References

Format of DBPUT

- DBPUT(baseid, dset, mode_1, ImageStatus, list, buffer)
  - list and buffer must correspond
  - WARNING: Fields not referenced in the list will be filled with Binary Zeroes
- For a master, a record with the key must not already exist
- For a detail linked to manual masters, a manual master record with the key must already exist

A null list will cause the intrinsic to put nothing to the database, and end up with a successful return. Be careful while using the * list construct!

It’s a great idea to use the * for subsequent calls, as parsing fieldnames and determining security has a lot of overhead.

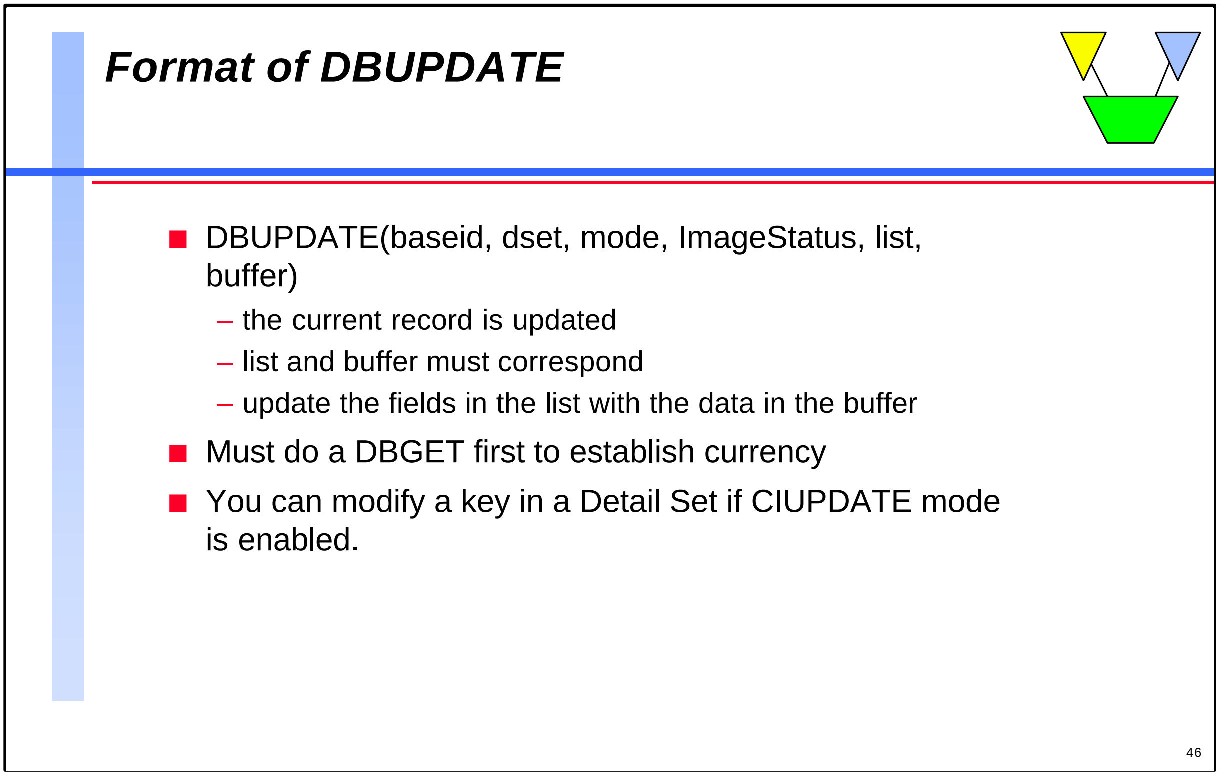
You can’t do a DBPUT to AUTOMATIC MASTERS.

Doing a DBPUT to a DETAIL set will automatically update any connected MASTER sets, including AUTOMATIC MASTERS.

Unless you’re accessing the database in Mode 3 (exclusive access), you must use a DBLOCK before doing a DBPUT.

For Techies

References



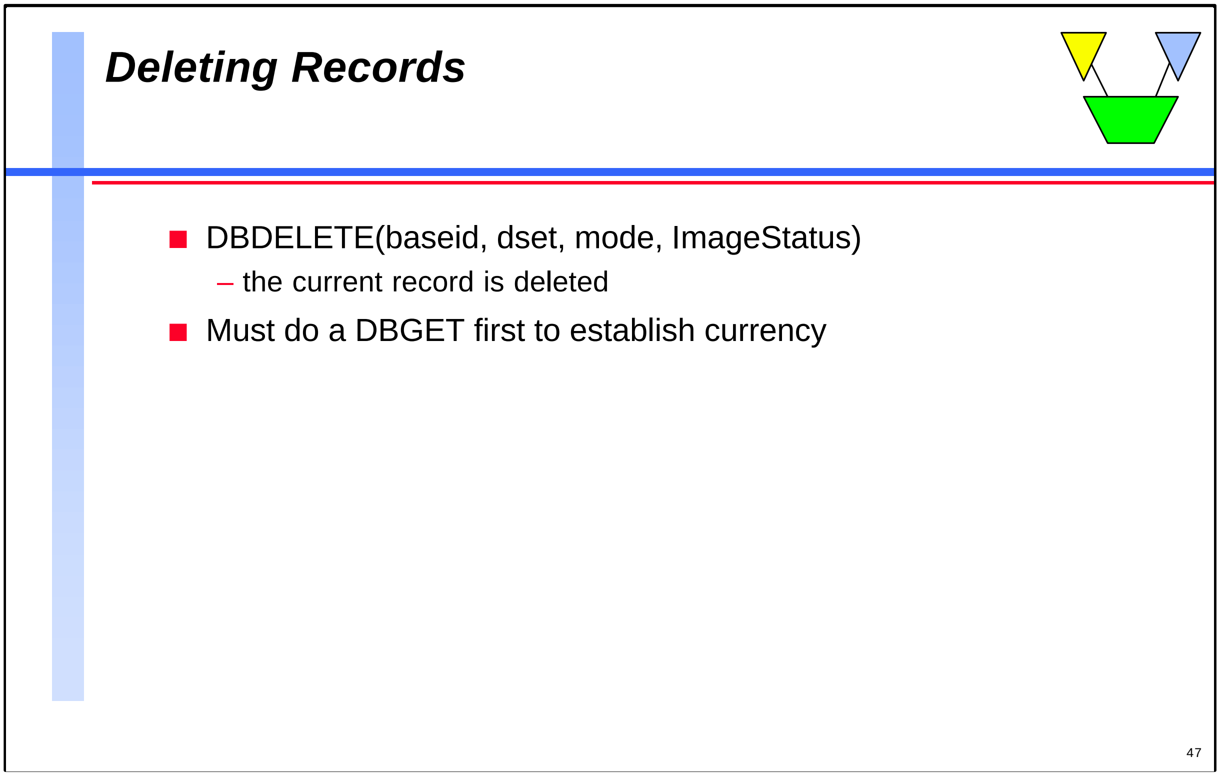
Format of DBUPDATE

- DBUPDATE(baseid, dset, mode, ImageStatus, list, buffer)
  - the current record is updated
  - list and buffer must correspond
  - update the fields in the list with the data in the buffer
- Must do a DBGET first to establish currency
- You can modify a key in a Detail Set if CIUPDATE mode is enabled.

Unless you're accessing the database in Mode 3 (exclusive access), you must use a DBLOCK before doing a DBUPDATE

For Techies

References



Deleting Records

- DBDELETE(baseid, dset, mode, ImageStatus)
  - the current record is deleted
- Must do a DBGET first to establish currency

47

Unless you’re accessing the database in Mode 3 (exclusive access), you must use a DBLOCK before doing a DBDELETE.

Performing a DBDELETE doesn’t physically remove the record, but it adds the record to the Delete Chain. This makes the space available for a future DBPUT.

If you accidentally delete some records, you can sometimes get them back with a 3rd-party tool - but don’t count on it.

The calling sequence should be

DBLOCK

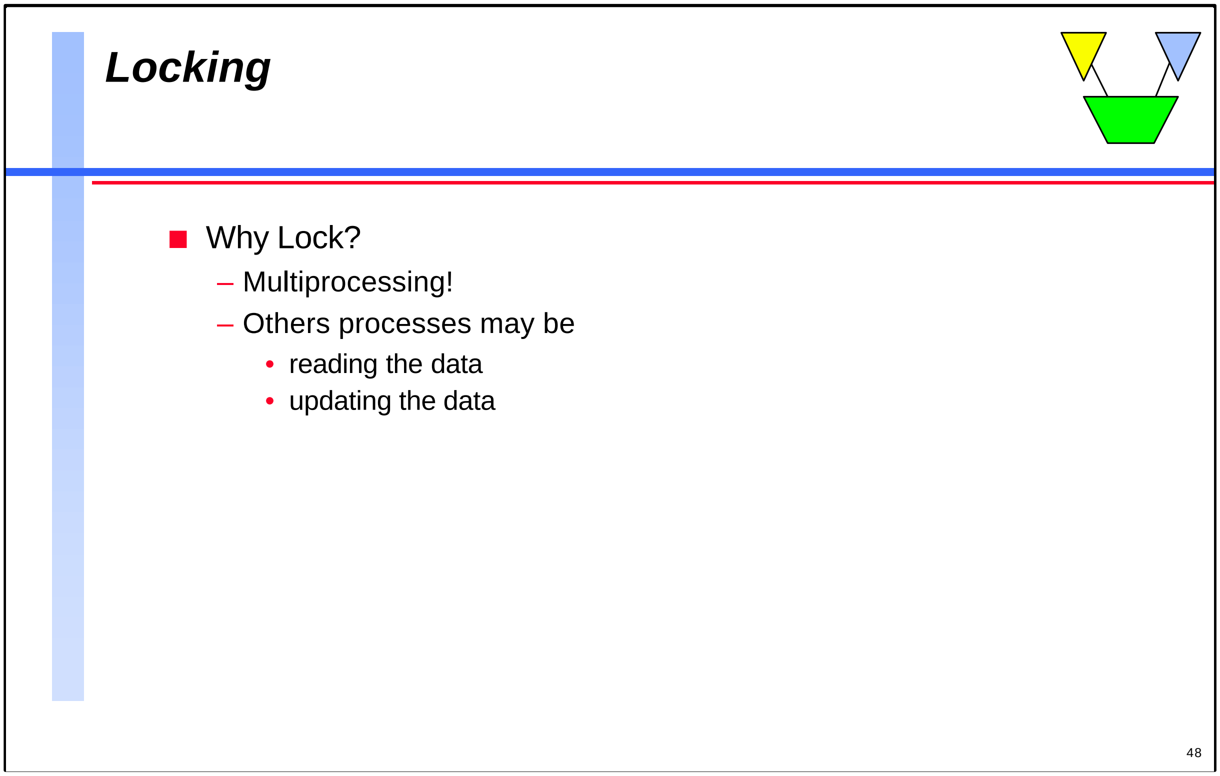
DBGET - get the record

DBDELETE

DBUNLOCK

For Techies

References



Locking

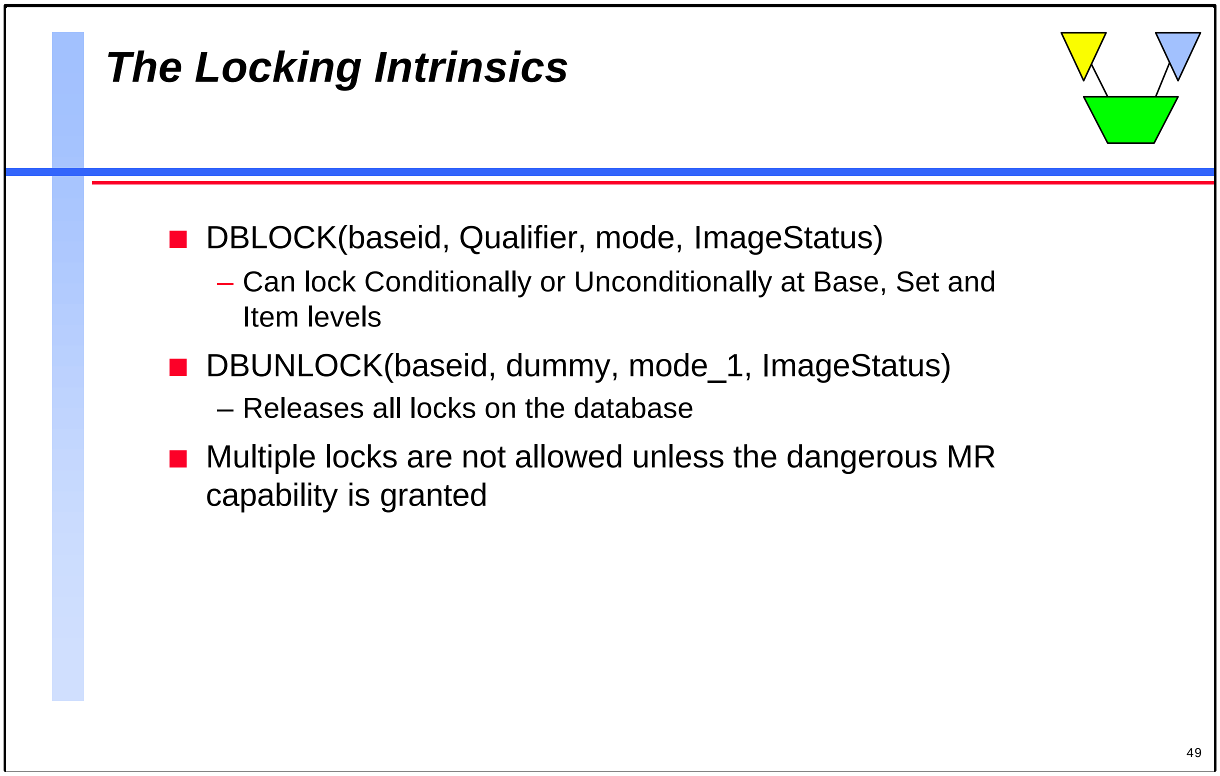
- Why Lock?
  - Multiprocessing!
  - Others processes may be
    - reading the data
    - updating the data

Although not enforced for reads, you should also lock around DBGETs if other processes will be changing the data.

If you do not lock around DBGETs and other processes are modifying the data, you may get a Broken Chain error #18.

For Techies

References



The Locking Intrinsics

- DBLOCK(baseid, Qualifier, mode, ImageStatus)
  - Can lock Conditionally or Unconditionally at Base, Set and Item levels
- DBUNLOCK(baseid, dummy, mode_1, ImageStatus)
  - Releases all locks on the database
- Multiple locks are not allowed unless the dangerous MR capability is granted

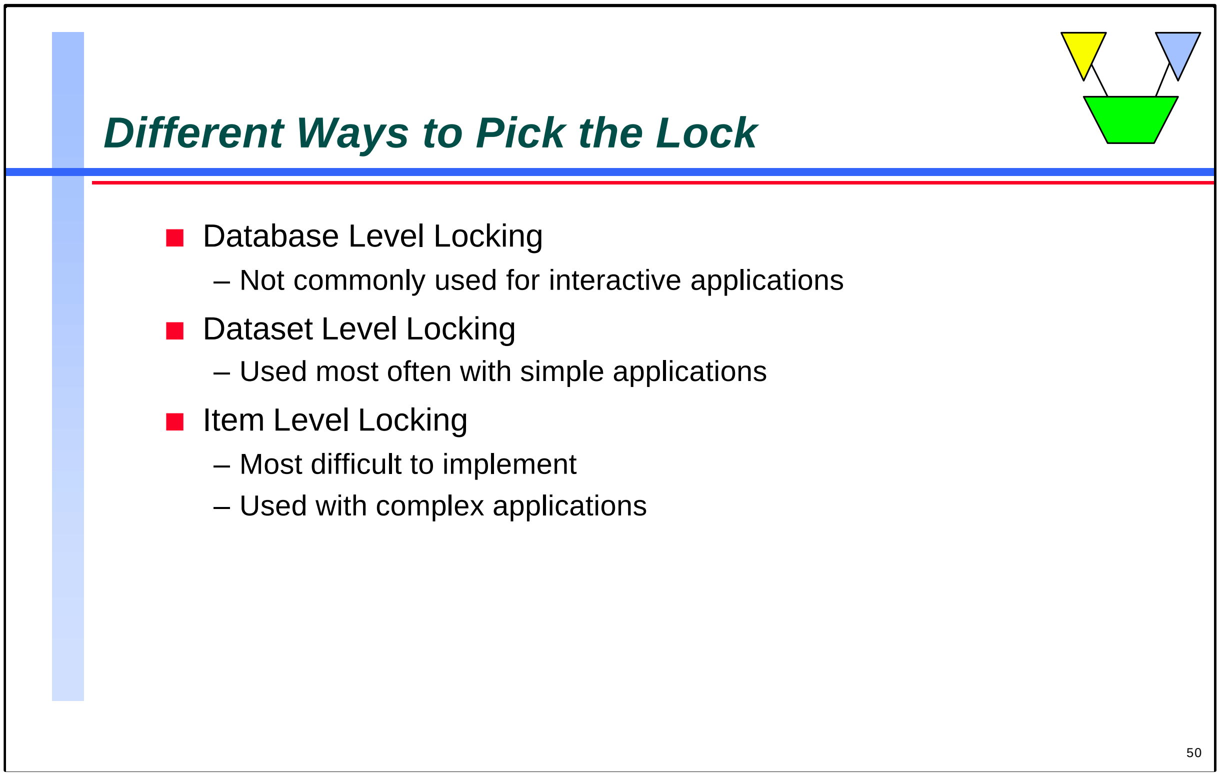
You may only call DBLOCK once per open database if MR (Multiple Resource Identification Number, or Multi-RIN) capability is not given to the program.

You should really only perform one lock on a database at a time. Careless locking strategies can result in a DATABASE DEADLOCK, which may require re-booting of the system.

To help detect Database Deadlocks, use DBCONTROL mode 7 after the database is open if you plan on using multiple locks. Some deadlocks are not detectable.

For Techies

References



Different Ways to Pick the Lock

- Database Level Locking
  - Not commonly used for interactive applications
- Dataset Level Locking
  - Used most often with simple applications
- Item Level Locking
  - Most difficult to implement
  - Used with complex applications

Locking is optional for reads. But if other applications may be modifying the data, you should lock around reads, too.

Database level locking is best used in batch applications when there will be no other accessors to the database. With trends towards 24 by 7 database access required, this method is becoming less prevalent.

Dataset level locking is best used for very simple applications.

Item level locking is best used for complex systems. This method can be quite tricky, especially if you have applied item-level security. It requires care on the part of the programmer, and coding bugs are more likely to happen using this method of locking than with others.

There are limits as to the number of items that you can lock, dependant upon their size, and the size of the DataBaseGlobal file that is created at runtime.

Implement standards by setting up subroutines that perform the locking.

For Techies

Disable the Break key in online applications that perform locking.

If a user presses Break during a lock, other processes will be waiting for the lock to release. This could take a long time for someone to notice.

References

Updating Some Account Records: When Do We Lock?

1 Ask for account#
2 Get account data
3 Display data to clerk
4 Wait for clerk to enter changes
5 Accept updated data from clerk
6 Update records with new data

This is a seemingly-simple transaction: update some data based on changes entered by a clerk.

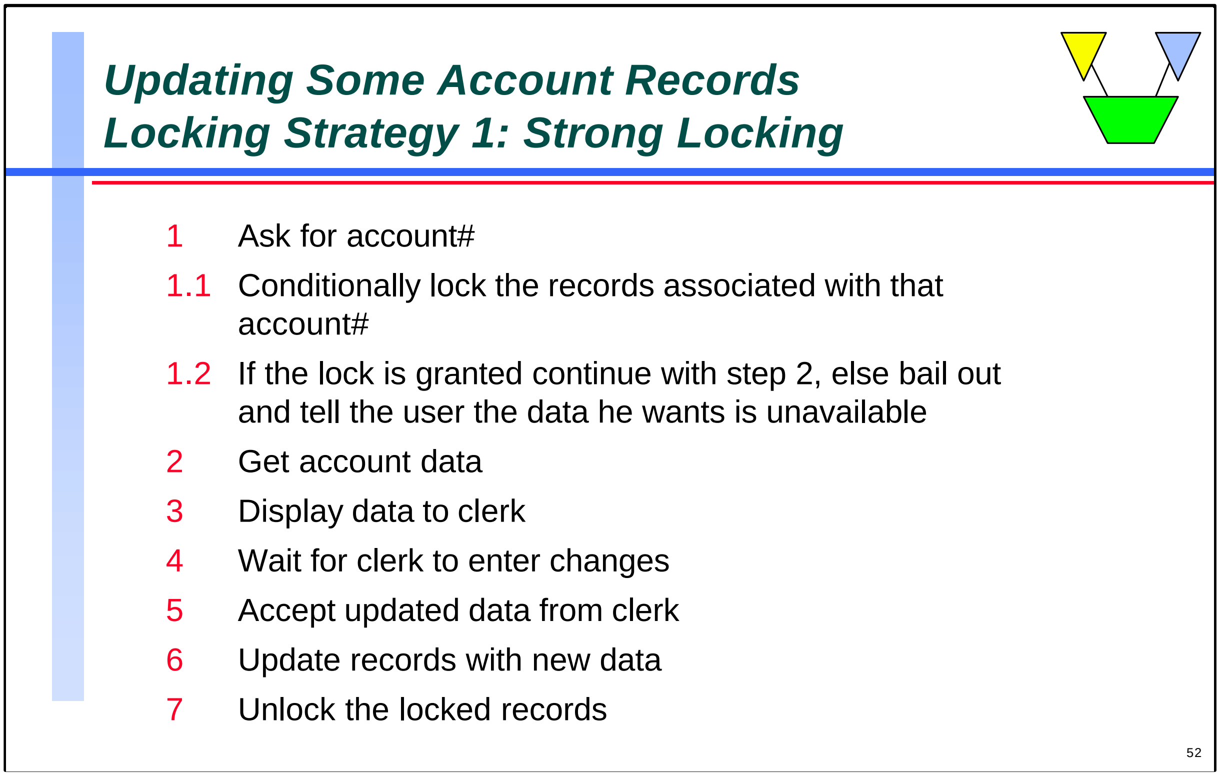
But IMAGE forces us to lock the data before updating it. Where do we put the DBLOCK?

If we request a lock any time before step 5, we’ll be locking the data for an unknown amount of time, because we don’t know how long the clerk will take to enter the changes. Maybe he’ll look up some information, answer the phone, go for lunch, … In the meantime, other users may be locked out of the database, dataset, or that account, depending on what kind of lock we request.

On the other hand, if we wait that long to lock, how can we be sure that the data has not been changed by someone else while the clerk was trying to decide what to enter?

For Techies

References



Updating Some Account Records
Locking Strategy 1: Strong Locking

1 Ask for account#

1.1 Conditionally lock the records associated with that account#

1.2 If the lock is granted continue with step 2, else bail out and tell the user the data he wants is unavailable

2 Get account data

3 Display data to clerk

4 Wait for clerk to enter changes

5 Accept updated data from clerk

6 Update records with new data

7 Unlock the locked records

Two interesting things here: conditional locking, and locking only the minimum that is needed to complete the transaction.

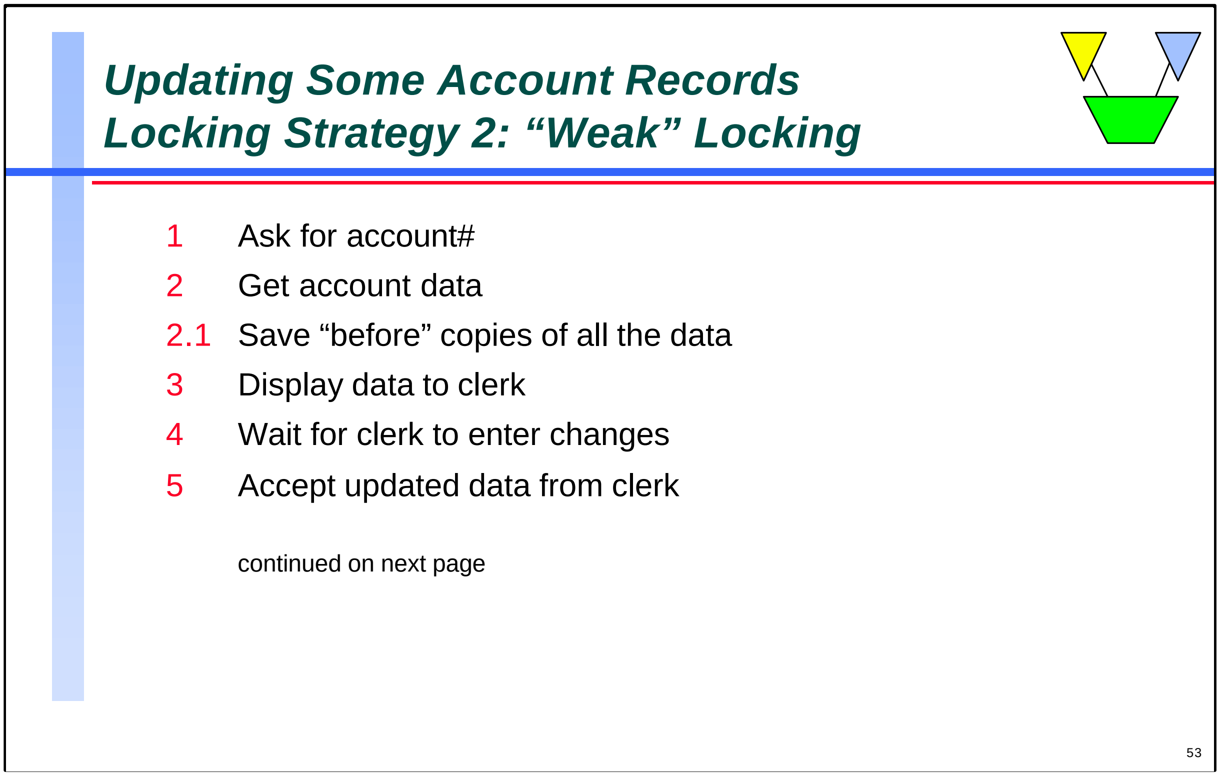
With a regular, unconditional lock, you either get the lock right away, or if the entry is locked you wait in line for your turn. There’s no telling how long you’ll be waiting, and you cannot un-request an unconditional lock once you’ve asked for it. Your program is stuck waiting and cannot do anything else.

With a conditional lock, you either get the lock right away, or you are notified that you cannot get the lock because someone else already has the entry locked. You can then take some kind of action, such as notifying the user to try again later.

There are three levels of locking: database, dataset, or predicate. With database or dataset level locks, the entire database or dataset is unavailable to other processes while the lock is held. With predicate level locking you specify exactly what you want to lock. For example, you can lock only account 12345, which allows other clerks to update other accounts while one clerk takes a break in the middle of updating account 12345.

For Techies

References



Updating Some Account Records
Locking Strategy 2: “Weak” Locking

1 Ask for account#

2 Get account data

2.1 Save “before” copies of all the data

3 Display data to clerk

4 Wait for clerk to enter changes

5 Accept updated data from clerk

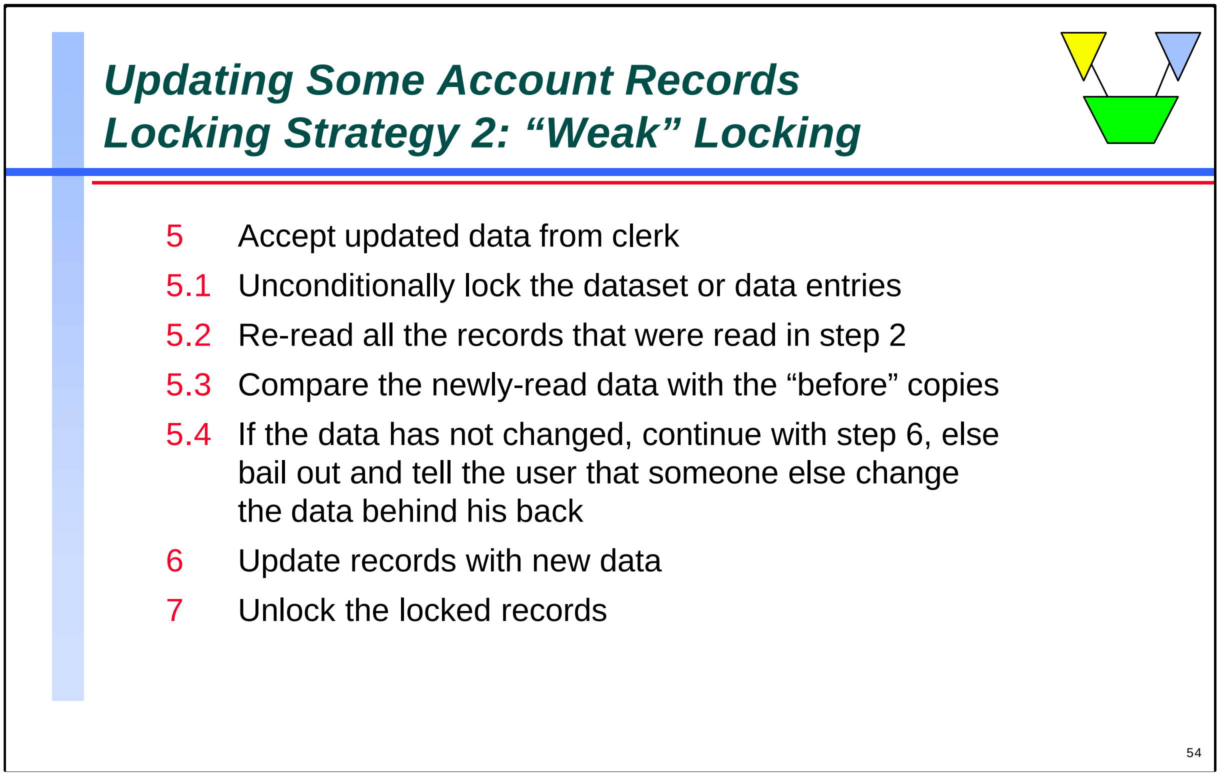
continued on next page

In this scenario we do not lock anything during the user think time.

But we do make a reference copy of the data that we initially retrieved, so that we can compare it later.

For Techies

References



Updating Some Account Records
Locking Strategy 2: “Weak” Locking

5 Accept updated data from clerk

5.1 Unconditionally lock the dataset or data entries

5.2 Re-read all the records that were read in step 2

5.3 Compare the newly-read data with the “before” copies

5.4 If the data has not changed, continue with step 6, else bail out and tell the user that someone else change the data behind his back

6 Update records with new data

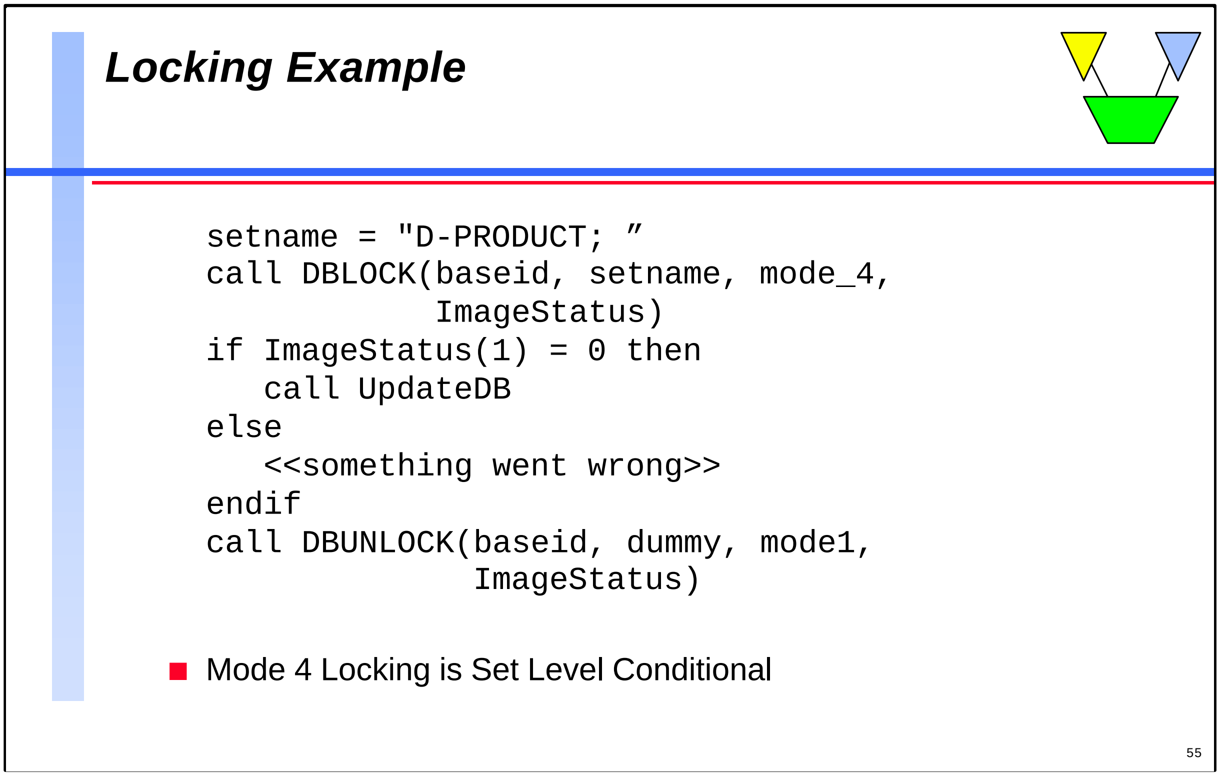
7 Unlock the locked records

After the user has finished thinking, it’s critical that we re-read the original data to be sure that it hasn’t been deleted or altered by somebody else in the meantime. We compare the newly-retrieved records with the record images we saved during the first retrieval.

As long as the same records exist with exactly the same data, we can apply the updates the user entered. But if anything has changed or disappeared, we cannot proceed any further. The changes are based on now-obsolete data, and the transaction must be cancelled.

For Techies

References



Locking Example

```
setname = "D-PRODUCT; ”
call DBLOCK(baseid, setname, mode_4,
            ImageStatus)
if ImageStatus(1) = 0 then
   call UpdateDB
else
   <<something went wrong>>
endif
call DBUNLOCK(baseid, dummy, mode1,
              ImageStatus)
```

- Mode 4 Locking is Set Level Conditional

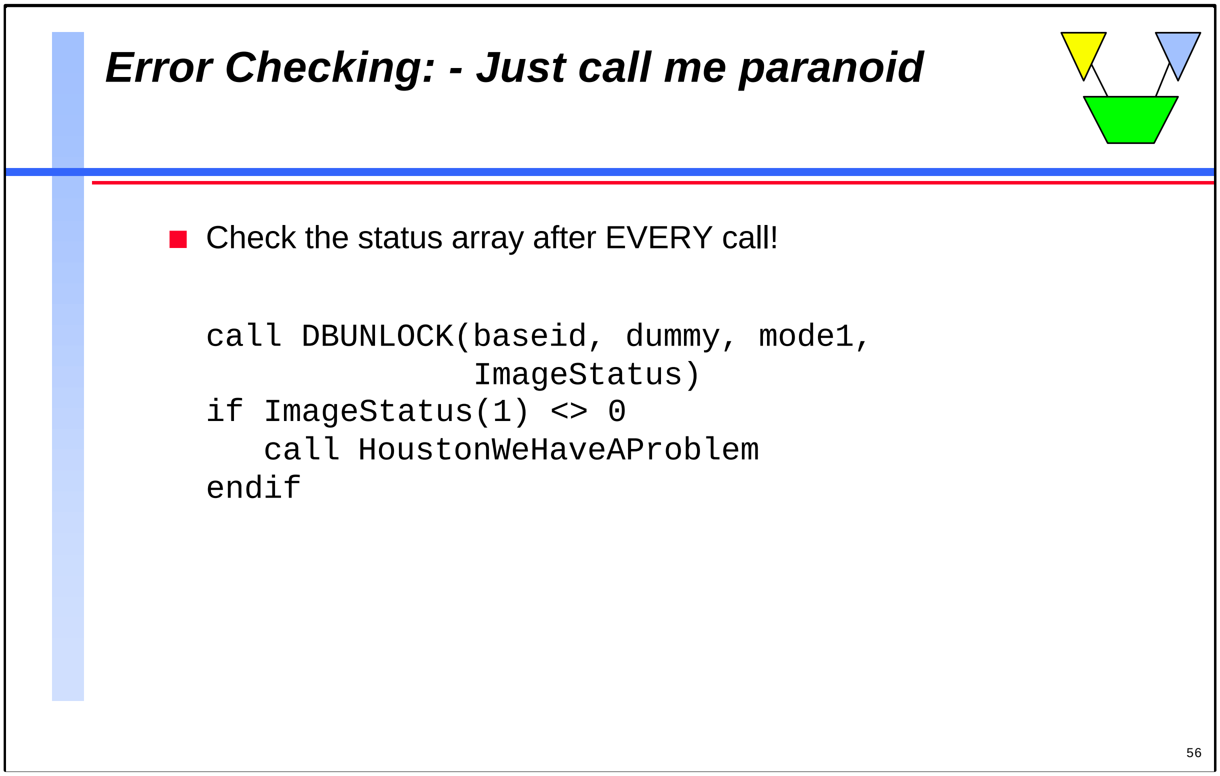
55

The most popular locking strategy is to lock the entire set while performing an update or deletion.

Put the DBGET inside the lock...if it’s outside, someone else may modify the data before you do!

For Techies

References



Error Checking: - Just call me paranoid

- Check the status array after EVERY call!

```
call DBUNLOCK(baseid, dummy, mode1,
              ImageStatus)
if ImageStatus(1) <> 0
   call HoustonWeHaveAProblem
endif
```

The reasons for so much error checking:

- Data sets will fill up
- Disc drives fill up
- Disc drives can fail
- Other hardware can fail
- Programmers can make mistakes in their logic

Of course the last item in the list is the most prevalent cause of errors.

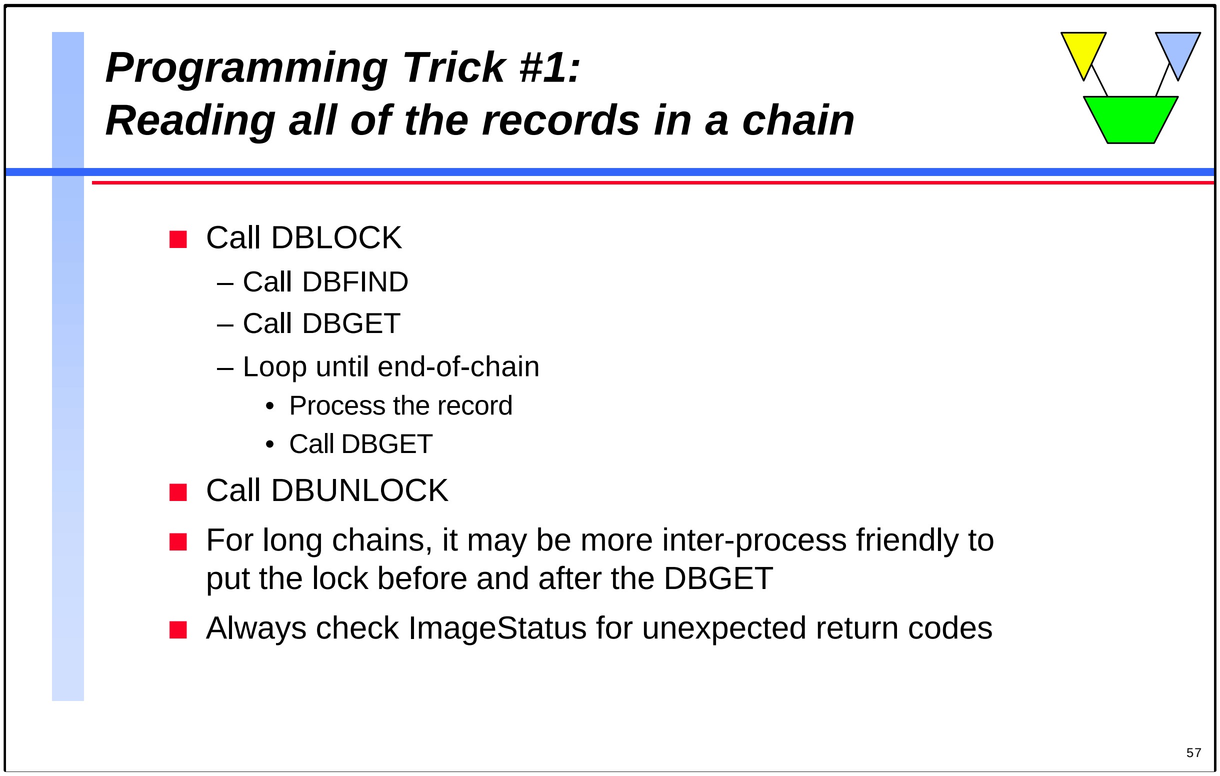
It is a good idea to code a standard error-detection routine that looks for unexpected errors AND ALWAYS USE IT. Eg.

```
sub CheckForWeirdErrors(ImageStatus) as Boolean
   if ImageStatus = 0 or 10 or 11 or 14 or 17 or 20 or 22
      return true
   else
      call WereInTroubleNow
      return false
   end if
end sub
```

Lazy progamming practices will produce bugs.

For Techies

References



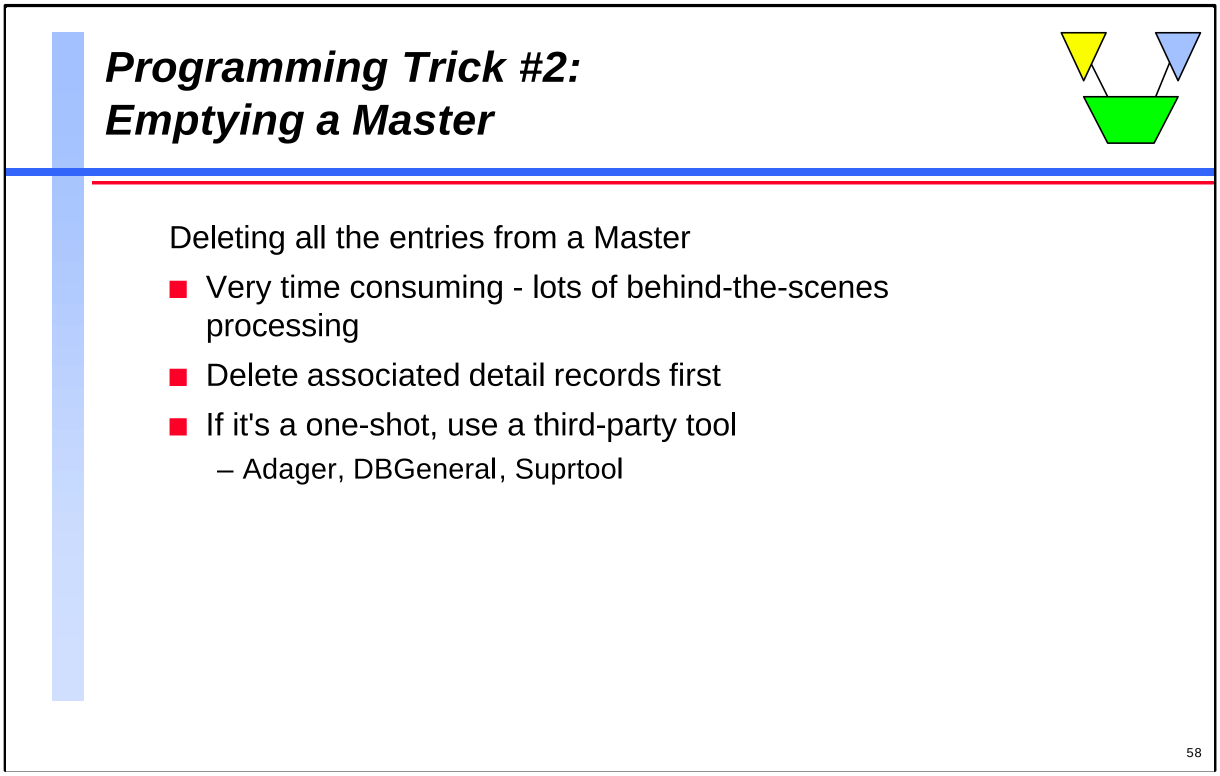
Programming Trick #1: Reading all of the records in a chain

- Call DBLOCK
  - – Call DBFIND
  - – Call DBGET
  - – Loop until end-of-chain
    - Process the record
    - Call DBGET
- Call DBUNLOCK
- For long chains, it may be more inter-process friendly to put the lock before and after the DBGET
- Always check ImageStatus for unexpected return codes

```
argument = "4509-FOOP"
call DBFIND(baseid, dset, mode_1, ImageStatus,
            item, argument)
if ImageStatus = 0
   call DBGET(baseid, dset, mode_5,
              ImageStatus, list, buffer, dummy)
   while ImageStatus = 0
      call ProcessTheBuffer
      call DBGET(baseid, dset, mode_5,
                 ImageStatus, list, buffer,
                 dummy)
   endwhile
endif
```

For Techies

References



Programming Trick #2: Emptying a Master

Deleting all the entries from a Master

- Very time consuming - lots of behind-the-scenes processing
- Delete associated detail records first
- If it's a one-shot, use a third-party tool
  - Adager, DBGeneral, Suprtool

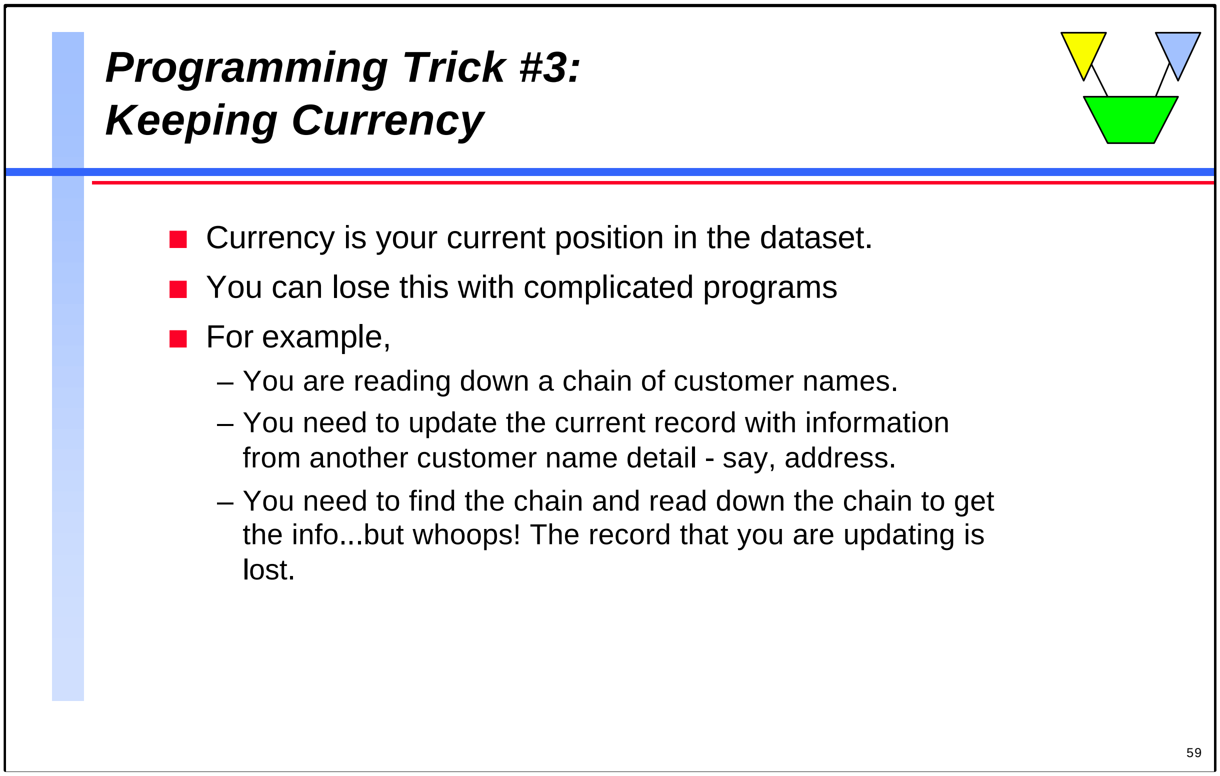
58

```
dblock'dataset
more'records = true
read'mode = serial
while more'records do
   dbget(baseid,dset,read'mode,status,
         list,buffer,dummy)
   read'mode = serial
   if status = end'of'file then
      more'records = false
   else
      if status <> 0 then
         fatal'error
      else
         if status'synonym'count > 1 then
            read'mode = reread
         endif
         dbdelete(baseid,dset,mode1,status)
         if status <> 0 then
            fatal'error
         endif
      endif
   endif
endwhile
dbunlock'dataset
```

For Techies

Often many records will hash to the same location in a Master. IMAGE creates a Secondary Chain. When a Master record is deleted that has a secondary chain, the next item in the chain is physically moved to the new empty location. This is called a ‘Migrating Secondary’.

References

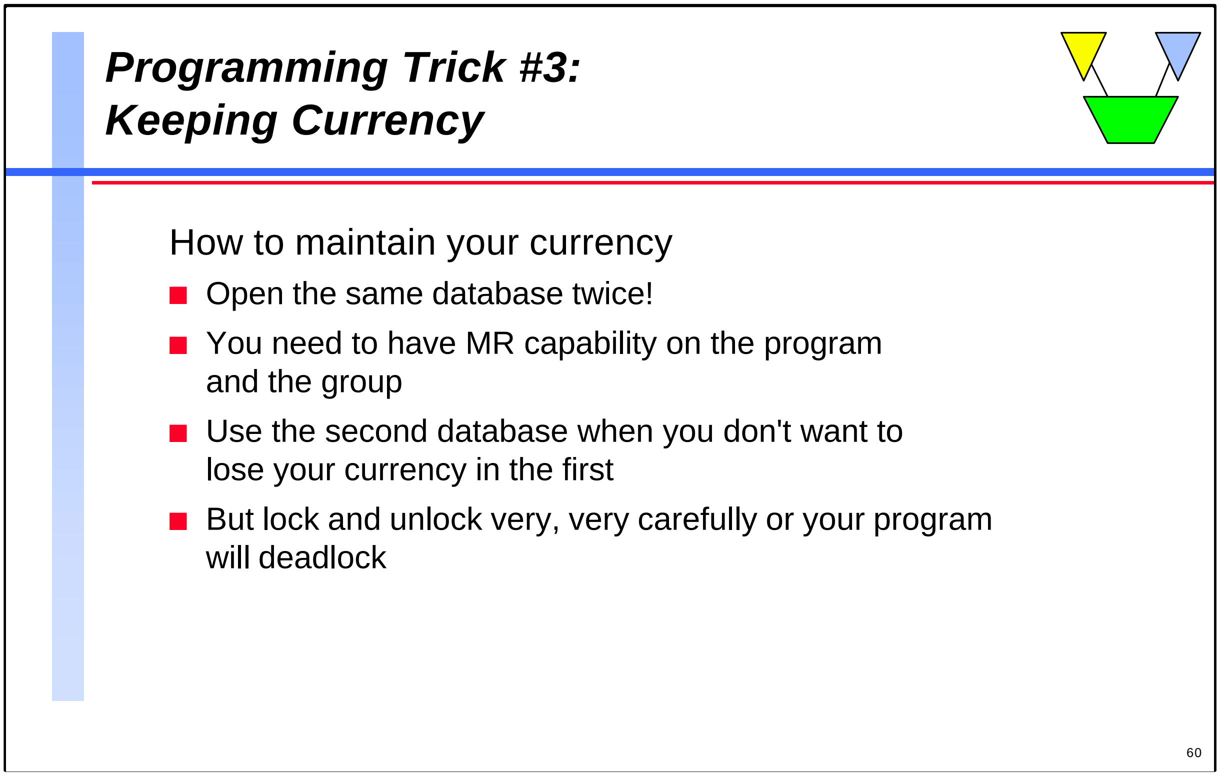


Programming Trick #3: Keeping Currency

- Currency is your current position in the dataset.
- You can lose this with complicated programs
- For example,
  - You are reading down a chain of customer names.
  - You need to update the current record with information from another customer name detail - say, address.
  - You need to find the chain and read down the chain to get the info...but whoops! The record that you are updating is lost.

For Techies

References



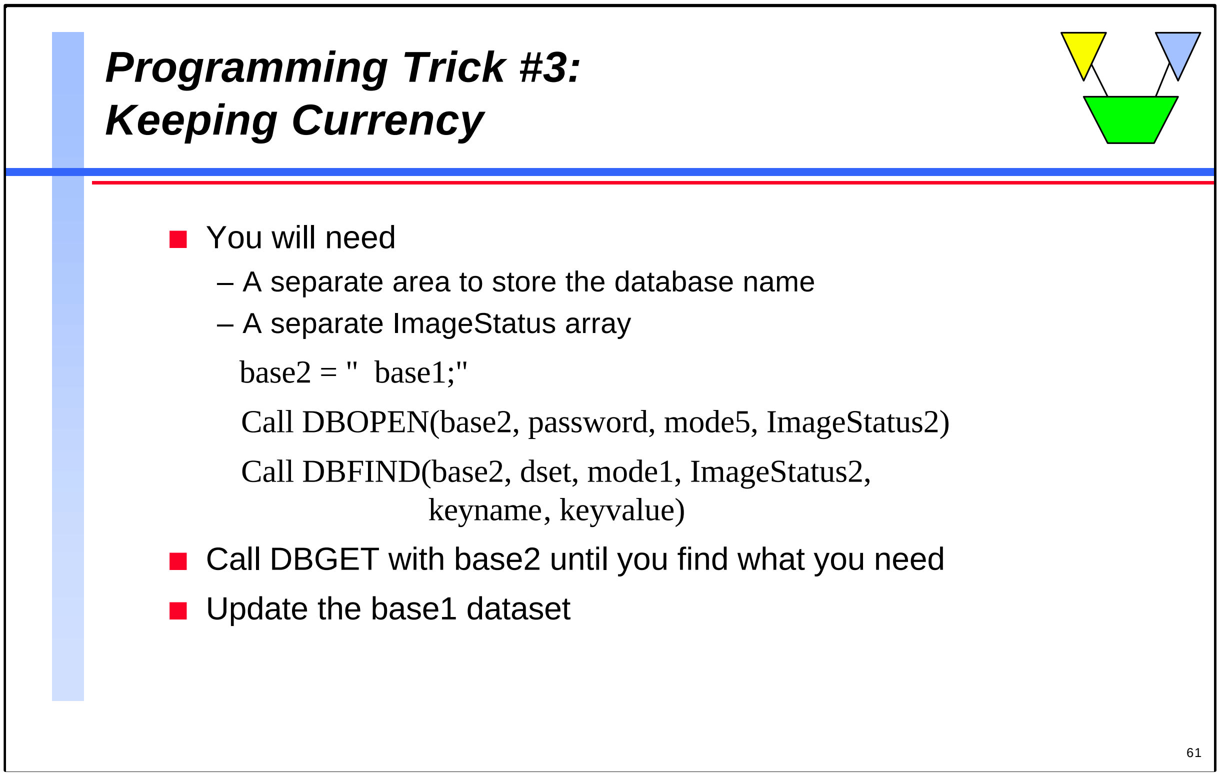
Programming Trick #3: Keeping Currency

How to maintain your currency

- Open the same database twice!
- You need to have MR capability on the program and the group
- Use the second database when you don't want to lose your currency in the first
- But lock and unlock very, very carefully or your program will deadlock

For Techies

References



Programming Trick #3: Keeping Currency

- You will need
  - A separate area to store the database name
  - A separate ImageStatus array

```
base2 = "  base1;"
Call DBOPEN(base2, password, mode5, ImageStatus2)
Call DBFIND(base2, dset, mode1, ImageStatus2,
                         keyname, keyvalue)
```

- Call DBGET with base2 until you find what you need
- Update the base1 dataset

Don't do lots of DBOPEN's!

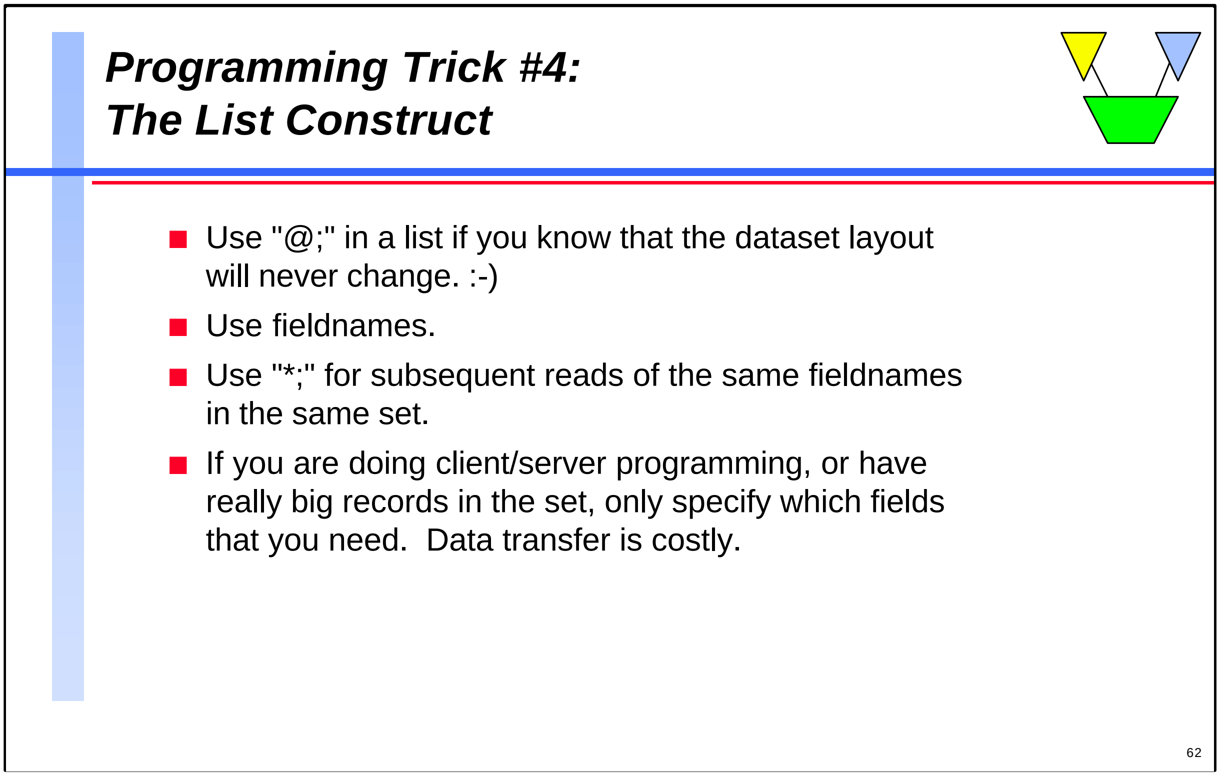
- Resource hogs
- If you must open the database twice, do it only ONCE and leave it open

Here’s an expensive loop:

```
LOOP until Record-Is-Found
   CALL DBOPEN
   CALL DBFIND
   CALL DBGET
   CALL DBCLOSE
```

For Techies

References

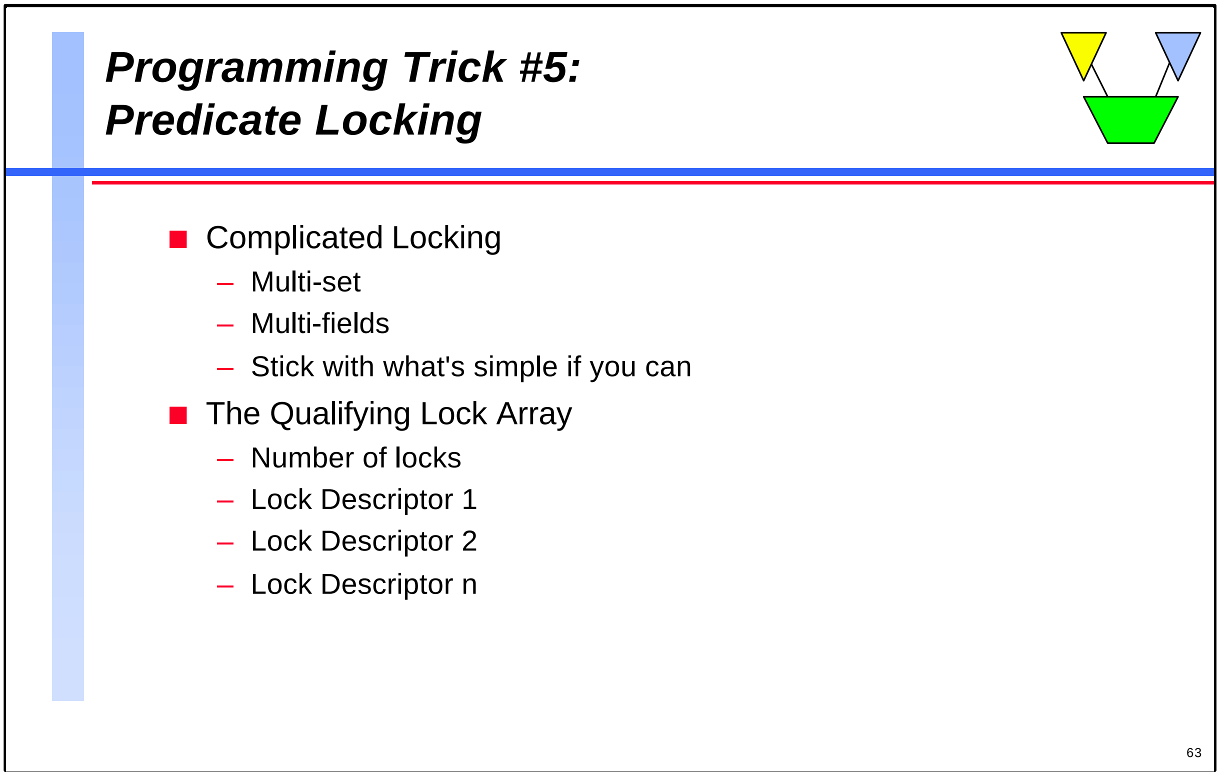


Programming Trick #4: The List Construct

- Use "@;" in a list if you know that the dataset layout will never change. :-)
- Use fieldnames.
- Use "*;" for subsequent reads of the same fieldnames in the same set.
- If you are doing client/server programming, or have really big records in the set, only specify which fields that you need. Data transfer is costly.

For Techies

References



Programming Trick #5: Predicate Locking

- Complicated Locking
  - Multi-set
  - Multi-fields
  - Stick with what's simple if you can
- The Qualifying Lock Array
  - Number of locks
  - Lock Descriptor 1
  - Lock Descriptor 2
  - Lock Descriptor n

The Lock Descriptor

Word 1	Length in words of Lock Descriptor	
2-9	Dataset Name	
10-17	Fieldname	
18	Relational Operator	<= >= =
19	Value	

Predicate (multi-set, multi-fields) locking is trickier, but is considered a part of a “strong” locking strategy.

DBUNLOCKs are practically free.

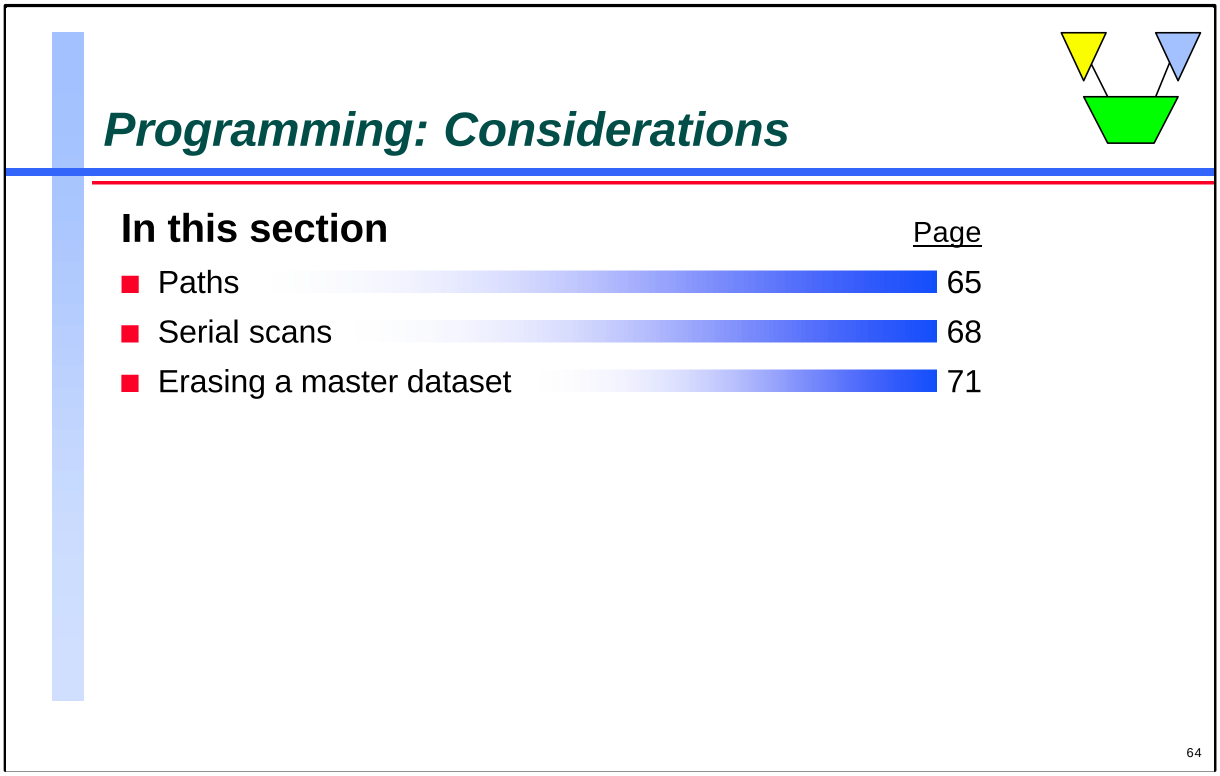
You can DBUNLOCK as often as you want.

DBUNLOCK releases ALL the locks

Be careful using locking when your process has two databases open! You may deadlock yourself.

For Techies

References



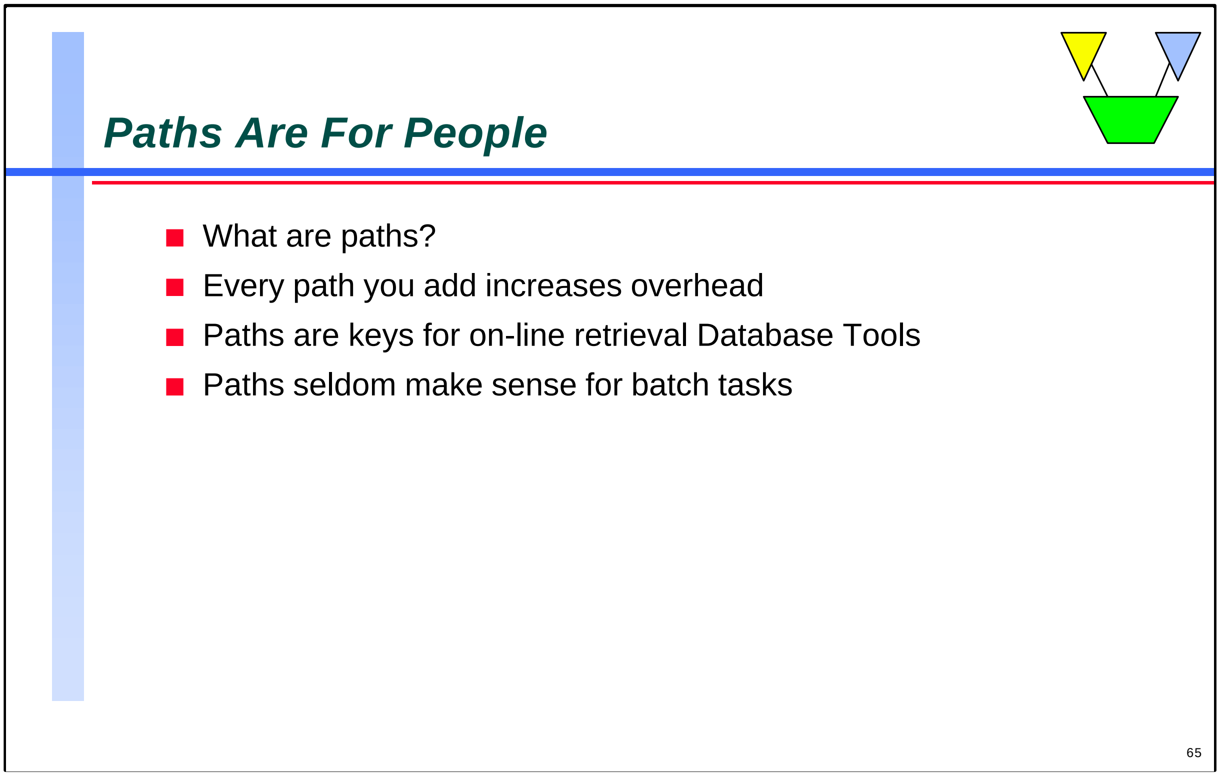
Programming: Considerations

In this section

	Page
■ Paths	65
■ Serial scans	68
■ Erasing a master dataset	71

For Techies

References



Paths Are For People

- What are paths?
- Every path you add increases overhead
- Paths are keys for on-line retrieval Database Tools
- Paths seldom make sense for batch tasks

A path is a structural relationship between entries in a master dataset and related entries in detail datasets. Within detail datasets, entries having the same search item value are also linked together forming a chain. IMAGE takes care of maintaining the information in these structures. Because of that, every path you add increases the amount of work IMAGE has to do whenever the information changes.

Paths are designed to provide quick access to existing data. Fast access is critical to on-line applications but not necessarily for batch processes.

For Techies

References

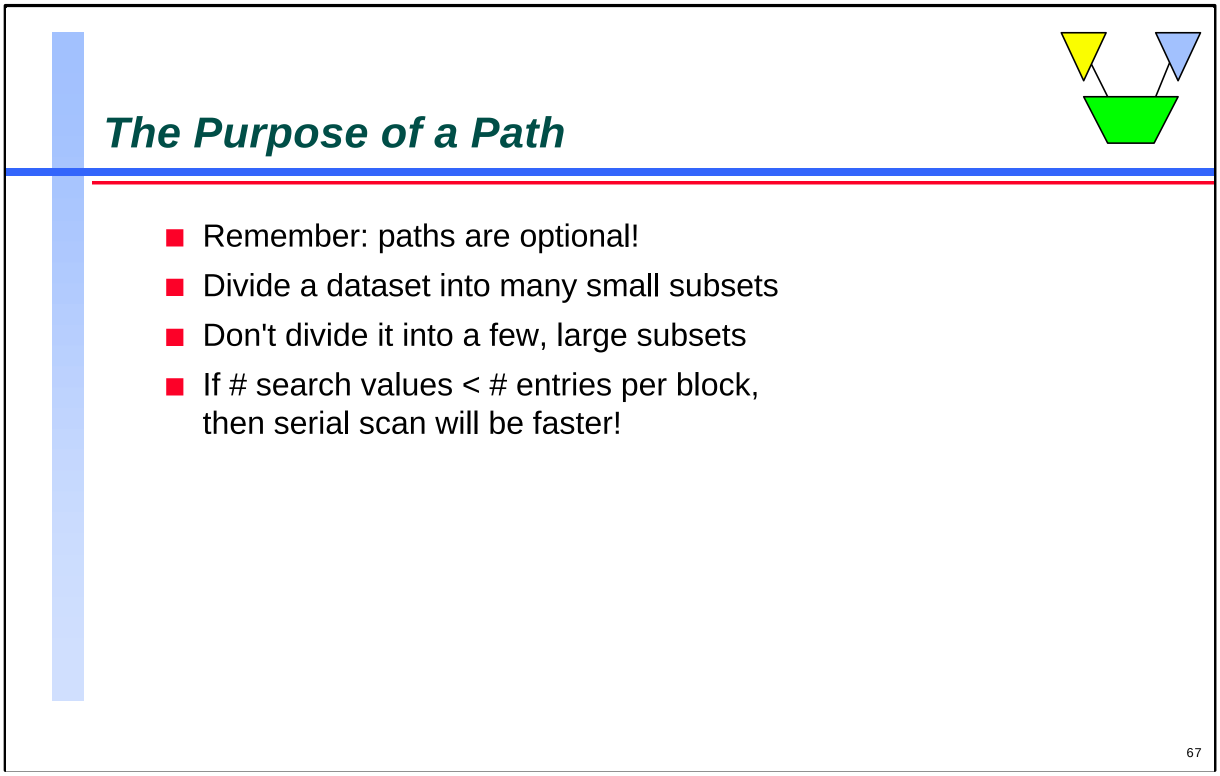
How Many Inquiry Paths Should I Have?

- What is the volatility of the dataset?
- How fast are new entries added?
- How often does the data change?
- How often is the data queried by the key?
- High Read/Write ratio => more keys

IMAGE has to work hard whenever information in a path changes. That is when you insert or delete entries and when you update a search item value. You should estimate the volatility of the dataset in order to decide how many paths you need and which items to use. Volatility is the ratio between read and write requests. If the ratio is low e.g. 1 read/1 write, you should keep the number of paths to a minimum and keep the overhead down. If the volatility ratio is high e.g. 1000 reads/1 write, you can add paths to provide retrieval alternatives.

For Techies

References



The Purpose of a Path

- Remember: paths are optional!
- Divide a dataset into many small subsets
- Don't divide it into a few, large subsets
- If # search values < # entries per block, then serial scan will be faster!

A good access path is one that produces many short chains rather than a few long chains. Remember also that a dataset does not require a path. You could very well create a stand-alone detail dataset that is not linked to any master.

If you have to retrieve all the entries in a dataset, it might be faster to do a serial scan than to do a chained read with each key value. That’s more so if the number of unique values is less than the blocking factor. Each unique value would require 1 read whereas a serial scan takes only 1/N reads (where N is the blocking factor). In other words, each read in a serial scan retrieves many unique values.

For Techies

References

Example of a "Good" Serial Scan

- There are 2.3 million detail entries of 300 bytes
- There are 13 records per block
- We have 162,000 order numbers to search for
- We want the lines sorted by order number
- How many disk reads for chained access?
- How many disk reads for serial scan?
- What if you use Nobuf/MR (Suprtool)?

Here is a quick way to estimate the number is disc I/Os required to retrieve 162,000 orders and related detail lines. To get at the details, you first need to read the master entry using DBFIND. This causes one I/O for each order number.

Then, there are 260,000 detail lines for these orders. Unless the detail dataset has been just before the job, the detail entries are scattered all over the place in the detail dataset. It's quite possible that each detail line will require one I/O.

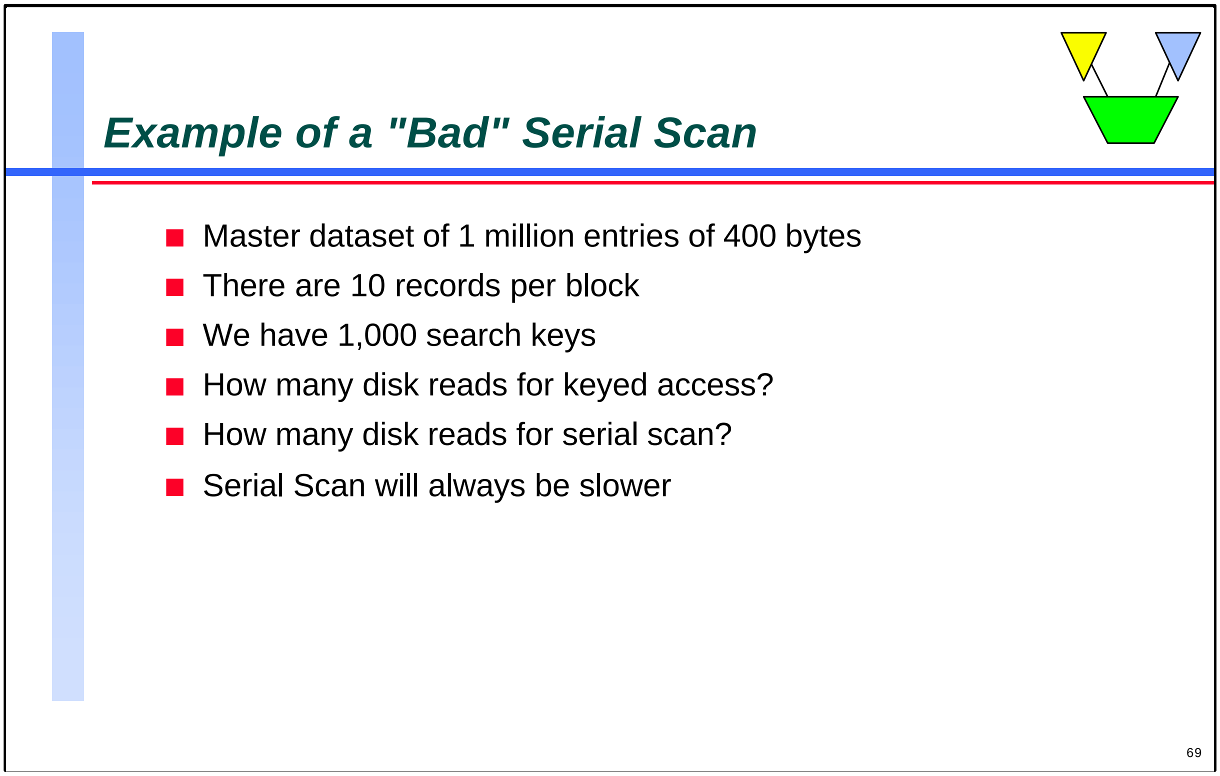
So, 162,000 DBFINDs plus 260,000 DBGETs add up to 422,000 disc reads.

Instead, you could do a serial scan straight out of the detail dataset. Each serial read retrieves one block at a time. Assuming there are 13 records per block, a serial scan only takes 176,924 disc reads (compared to 260,000). Since we do not have to access the master dataset, we are also saving 162,000 disc reads. You have eliminated 245,076 I/Os. That's very good.

An extract utility such as Suprtool can do better than that because it reads multiple blocks in each disc read. This technique is called MR/NOBUF (multi-record, no buffer). In this example, Suprtool could read more than 150 entries in one I/O. The number of disc reads then comes down to around 15,000.

For Techies

References



Example of a "Bad" Serial Scan

- Master dataset of 1 million entries of 400 bytes
- There are 10 records per block
- We have 1,000 search keys
- How many disk reads for keyed access?
- How many disk reads for serial scan?
- Serial Scan will always be slower

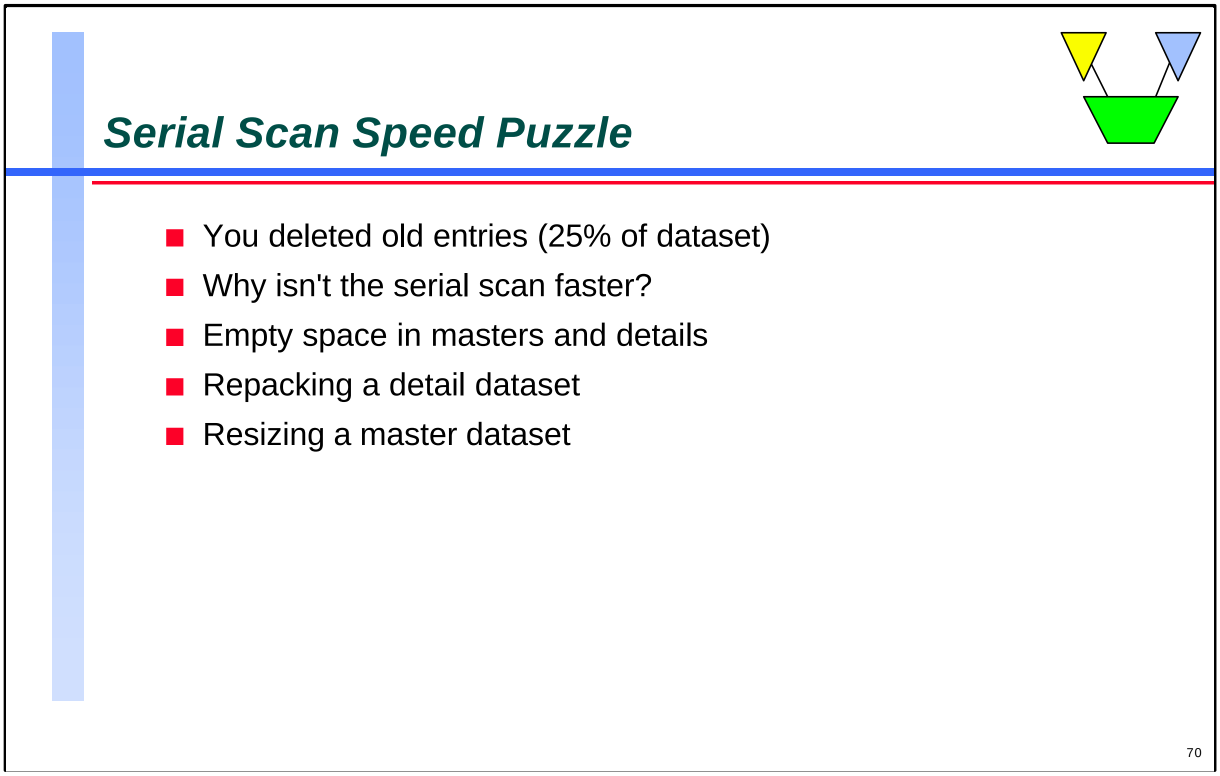
The opposite is also true: there are times when a chained read is going to be faster than a serial scan.

IMAGE is able to go directly at each entry using the hashing algorithm. This means the program only needs one I/O for each key value.

A serial scan would take 100,000 disc reads because it would retrieve only 10 records each time. Even Suprtool would take longer. It would be able to read about 120 records per I/O or about 6,500 disc accesses.

For Techies

References



Serial Scan Speed Puzzle

- You deleted old entries (25% of dataset)
- Why isn't the serial scan faster?
- Empty space in masters and details
- Repacking a detail dataset
- Resizing a master dataset

You have an archive program that scans all your datasets, masters and details, to delete obsolete entries. You are hoping that removing these unneeded entries will speed up full serial scans. To your surprise, nothing has changed. Why is that?

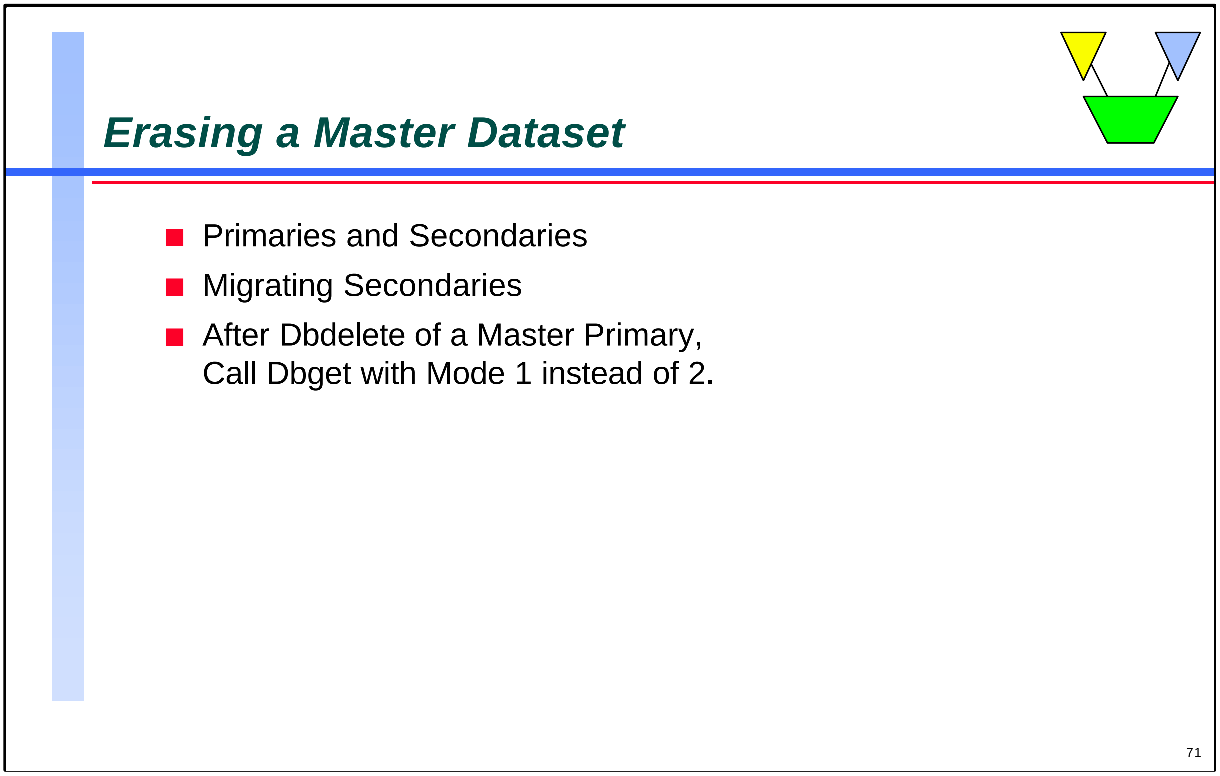
Entries in a master dataset are placed in random location based on the hashing algorithm. Entries could be anywhere so a serial scan on a master dataset always reads up to the dataset capacity. The only way to speed up a serial scan is to reduce the size of the master, if that’s feasible. Remember that changing the capacity forces all existing entries to be re-hashed. You might be helping serial scan but you might be hurting calculated reads at the same time.

Entries in a detail dataset are placed sequentially as they are created. A brand new detail dataset where you simply add new entries does not have any empty location. IMAGE keeps track of the next empty location using the highwater mark. A serial scan typically reads up to the highwater mark.

When the archive program deletes old entries, these locations are marked as available again and are added to the delete chain. This chain is only used and maintained by IMAGE itself. If you add a new entry, IMAGE simply follows the delete chain and inserts the entry in the first empty spot. So, entries that make up the delete chain can be anywhere in the dataset. But IMAGE still reads up to the highwater mark. To fix this, you have to repack the dataset in order to remove the delete chain and reset the highwater mark.

For Techies

References



Erasing a Master Dataset

- Primaries and Secondaries
- Migrating Secondaries
- After Dbdelete of a Master Primary,
  Call Dbget with Mode 1 instead of 2.

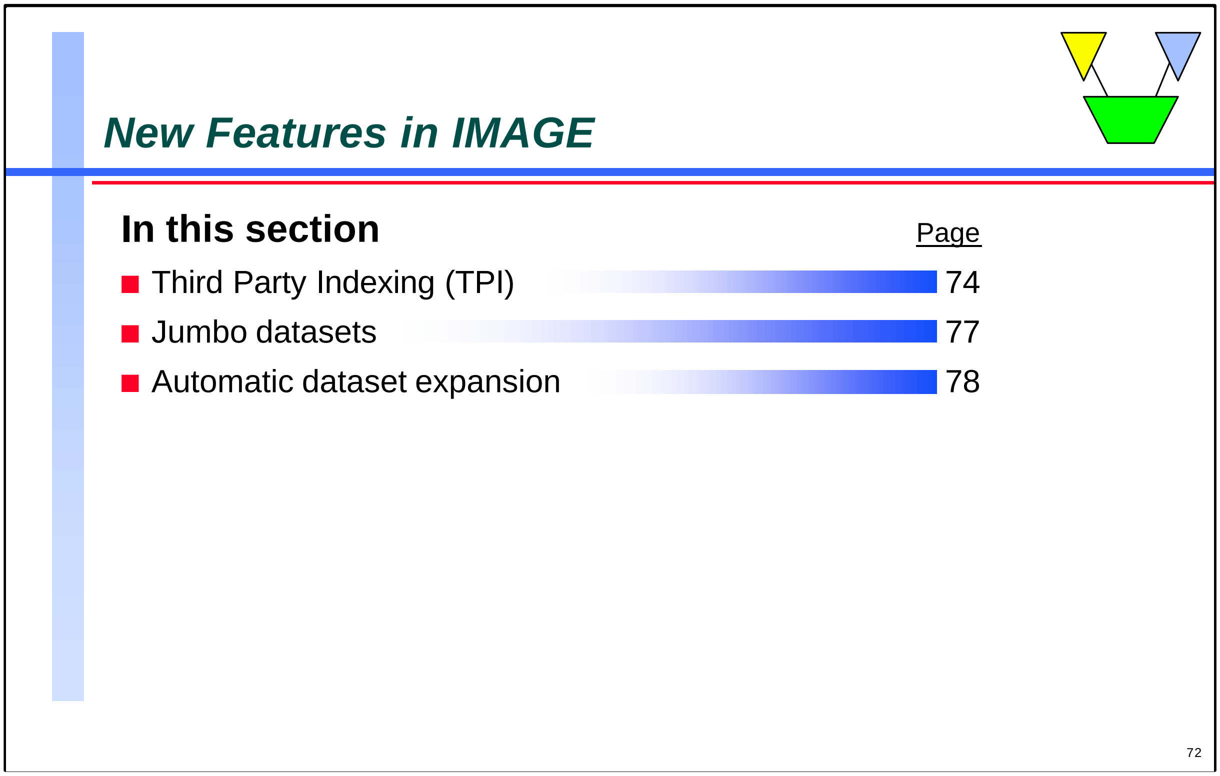
71

There are 2 types of entries in a master dataset: primaries and secondaries. Primary entries are stored in the location calculated by the hashing algorithm. Secondary entries are stored in a location other than their primary address because that location was already occupied by a primary entry. Secondaries for a primary location are linked and form what is called a synonym chain. One IMAGE rule states that an entry must reside in its primary location, if that location is available.

This means that, if you delete a primary entry, the first secondary in the synonym chain (if any) is moved to the primary location. This is known as a migrating secondary. So, if you are trying to clean up a master dataset, you should call DBGET with mode=1 after a DBDELETE to re-read the same location. Maybe a new entry has been moved there.

For Techies

References



New Features in IMAGE

In this section

	Page
■ Third Party Indexing (TPI)	74
■ Jumbo datasets	77
■ Automatic dataset expansion	78

72

For Techies

References

IMAGE has changed and grown

- Access
  - Omnidex
  - Superdex
  - TPI
  - B-Trees
- Capacity
  - Jumbo Datasets
- Maintenance or Uptime
  - DDX
  - MDX

Even though IMAGE has been faithfully storing and serving up data for over two decades, it has been improved and expanded to meet the data demands that customers have had. These areas of improvement have been in the general areas of Access, Capacity and Maintenance.

Relational Access was one of the first enhancements made to IMAGE. This was done by a Third-Party software vendors.

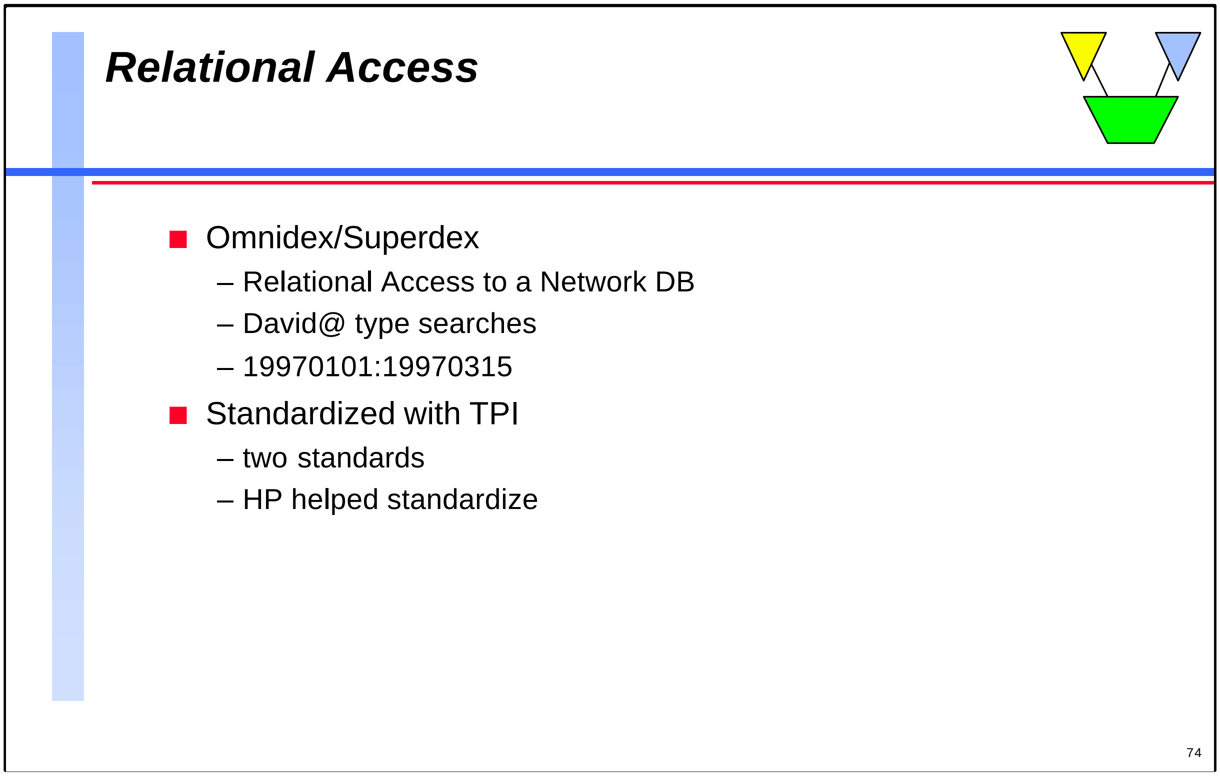
As data grew and larger applications were being hosted on the HP 3000 capacity became a concern. This was was addressed with Jumbo Datasets in MPE/iX 5.0.

In conjunction with capacity, 24 x 7 operations with no maintenance downtime, became important. Detail and Master Dataset eXpansion features were added to keep the application running and prevent downtime for database maintenance.

Since its inception in the mid-seventies, IMAGE has grown and changed in the areas of access, capacity and maintenance, but you still can open a database with the same dbopen call you wrote over twenty years ago.

For Techies

References



Relational Access

- Omnidex/Superdex
  - Relational Access to a Network DB
  - David@ type searches
  - 19970101:19970315
- Standardized with TPI
  - two standards
  - HP helped standardize

74

IMAGE is what is known as a network database, which did not lend itself to allow the type of access that some users had been used to. These types of accesses, such as partial-key retrieval, or in-between values would require either a serial read thru the entire dataset, which was time and resource consuming and not practical, or a KSAM file that would be kept in sync with the data in the dataset. The KSAM file would then point directly to the record in the dataset.

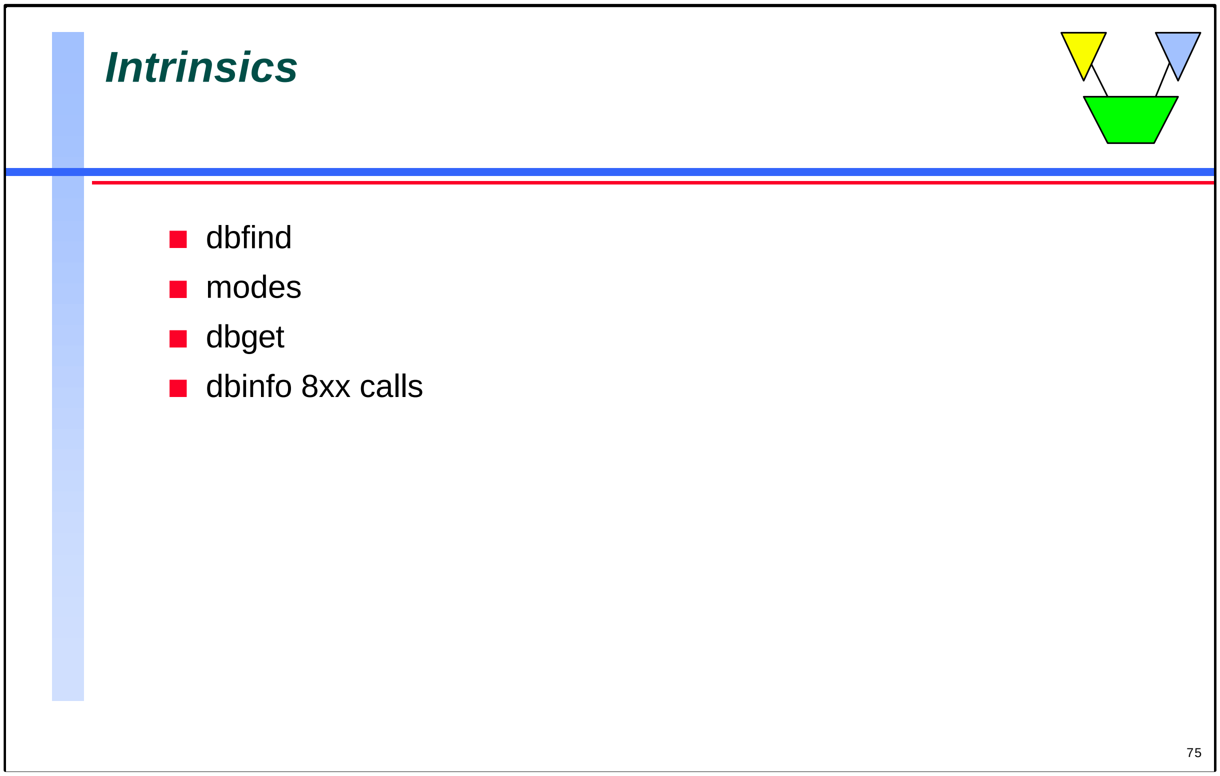
Two third-party products emerged as methods of providing this indexed access or relational queries of their data, while still using image or at least image-like intrinsic calls.

While these products flourished, one annoying problem for developers was that they did differ in their implementation and required that any program run with a lib= or xl= statement in order to insure that their version of dbfind or dbput, would not only add to the database but also keep the index in sync. Some programs would not run with this statement and therefore cause application problems and databases would have to be re-indexed.

DISC and Bradmark agreed to standardize the intrinsic calls and modes for each as well as provide a standard set of dbinfo calls to gain more information about an index. HP agreed to modify IMAGE to no longer require that the xl= statement be necessary, therefore insuring that the index was always in-sync with the database.

For Techies

References



Intrinsics

- dbfind
- modes
- dbget
- dbinfo 8xx calls

The indexing products, as mentioned previously would allow for Relational types of accesses. Simple partial-key retrievals were done thru the standard modes of access, such as dbfind mode 1, followed by a dbget, mode 5 (read chain). The argument passed to dbfind would contain certain characters which would trigger the indexed search, such as “David@”. A dbfind with this type of search on an index would retrieve all entries beginning with David:

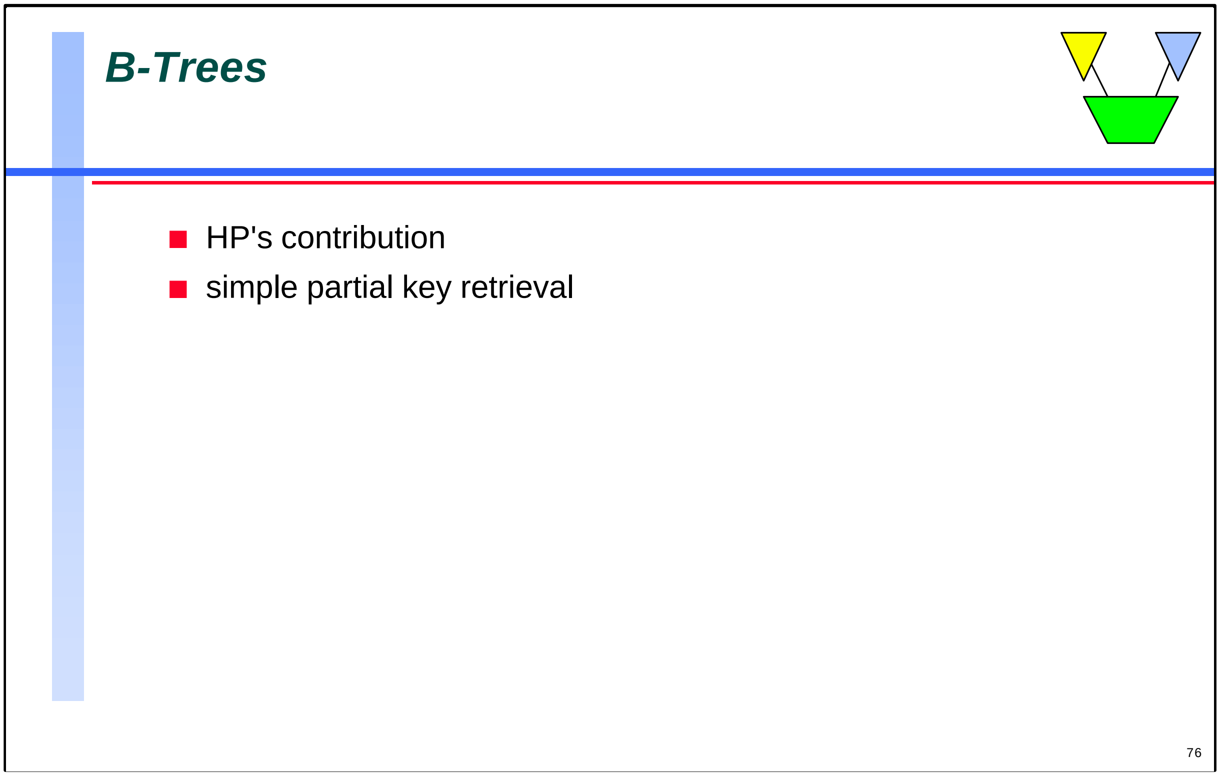
David Greer
David Duchovny
David Lo

The only change to the application, would be to insure that it would be able to logically handle n number of entries and be able to read down chains for certain types of queries. The modes specified to dbfind typically help control either the type of data being retrieved, (mode 1 for byte type, mode 11 for binary/ numeric data) or help control the type of lookup, such as >, >= (mode 2nn), >, >= (mode 3nn) etc. In dbfind modes 1, 12 and 21 the argument can contain many different operators, some examples are:

Conditional operators such as: @, any variable number of number or alpha characters, ?, any single alphanumeric character, #, any single numeric character. Retrieval operators such as >=, <= and <>. Boolean operators such as AND, OR, NOT.

For Techies

References



B-Trees

- HP's contribution
- simple partial key retrieval

There are some companies who did not require the advanced level of relational access, or did not like non HP solutions and still wanted to do partial-key retrieval lookups.

HP added B-Trees (indexed access to IMAGE data) in C.07.10 version of IMAGE that was available on MPE/iX 5.5 Express 4 release.

The method of indexing was done via a KSAM/XL file, which kept the index in sync with the database. The index file is kept in the Posix space with the same name as the dataset, but has the extension .idx. While not a sophisticated as the full relational capabilities of Omnidex or Superdex, it does provide simple partial-key retrieval access.

Similar to the two TPI products the type of access is controlled by the mode specified in the dbfind call and the contents of the argument. For example dbfind mode 1, will work as it does without a B-Tree, or can be used for a B-Tree search if BTREEMODE1 is ON, which can be set in dbutil or can be set programatically with a dbcontrol call. Other modes are, four (4) which allows a B-Tree index search on numeric fields. Mode 10 is a standard IMAGE dbfind mode 1 search regardless of the value of the BTREEMODE1 setting. Modes 21 and 24 are the same as 1 and four but are faster, but they do not return the number of entries qualified.

B-Trees searches also allow wildcard searches, with the following argument syntax, (<, <=, >, >=, “PK” or [] for between).

For Techies

References

Capacity

- Jumbo Datasets
- Super Jumbo Datasets

With the limit of anyone dataset being only 4GB (say this with a smile), for some users and applications this limit of 4GB for a dataset was simply not enough. Some application vendors were required by law to keep up to five years of data online.

The solution to this became known as jumbo data sets. Instead of a single file for a dataset, HP devised a way to have multiple files store data. Due to structural limitations, the new limit for datasets became 40 Gb, instead of 4Gb.

For jumbo datasets, the data is stored in files in the POSIX name space with the extension, “.001”, “.002” and so on. For example if the dataset CUSTMAST is the second dataset in the Orders database is a Jumbo master with two “chunk” files you would have three files that make up the dataset:

1) Order02   {chunk control file no data stored}

2) Order02.001 {chunk number 1, contains data}

3) Order02.002 { chunk number 2, contains data}

The first file is sometimes referred to the chunk control file, contains no “CUSTMAST” data, but rather information about the dataset. The data is actually stored in the two “chunk” files.

For Techies

References

Maintenance/Uptime

- Capacity changes when needed
- DDX
- MDX
- Three values
  - Maximum capacity
  - Initial capacity
  - increment

As some applications were required to be on-line 24 hours, 7 days a week, and the nature of datasets required them to have a fixed capacity which had to be all allocated, problems would arise when a particular dataset would reach it's capacity. Applications would then have to be shut down, in order for more room to be added to that particular dataset, which in some cases would take not only days but hours.

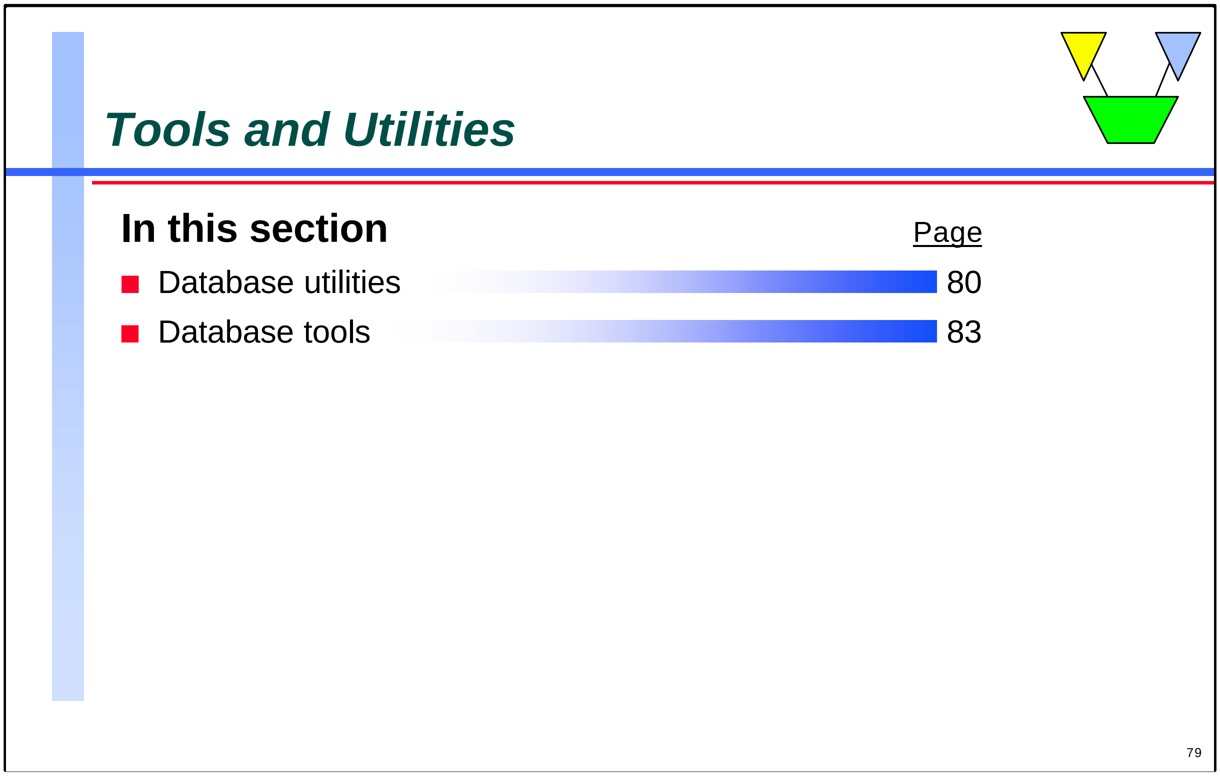
Database Administrators, would then be forced to create datasets with huge capacities which would all be allocated in order for them to not have to bring down at an unscheduled or unplanned point in time. This, however, was not an efficient use of resources.

It was proposed that databases be allowed to expand as needed, so as to reduce the waste of disc space. From this DDX (Detail Dataset eXpansion) and MDX (Master Dataset eXpansion) were born.

A dataset can be defined with the following values, maximum capacity, initial capacity and an increment. The maximum capacity is the maximum number of entries the dataset can hold. The initial capacity is the number of spaces that the dataset can hold, once this limit is reached the dataset will expand by the number of entries specified by the increment, which can be expressed as a percentage of the initial capacity.

For Techies

References



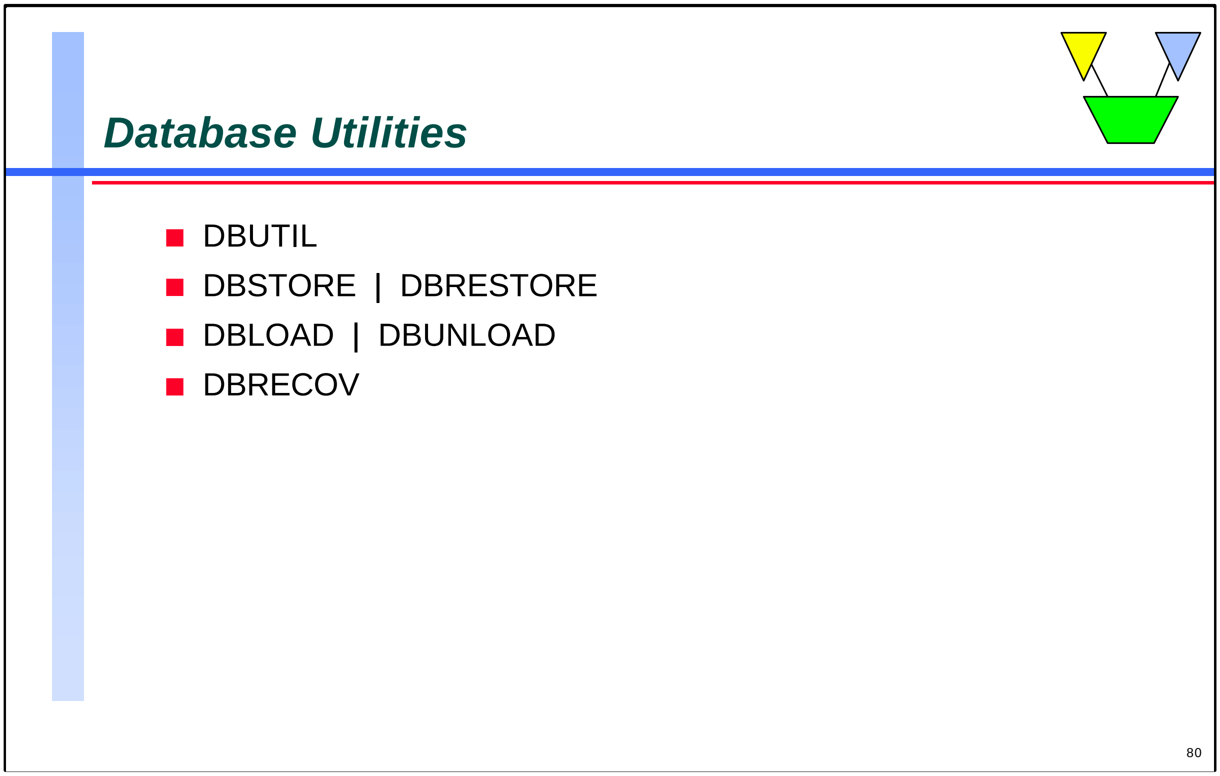
Tools and Utilities

In this section

	Page
■ Database utilities	80
■ Database tools	83

For Techies

References



Database Utilities

- DBUTIL
- DBSTORE | DBRESTORE
- DBLOAD | DBUNLOAD
- DBRECOV

80

Utilities to manage your database - set passwords, create and purge, load and unload, store and restore and recover.

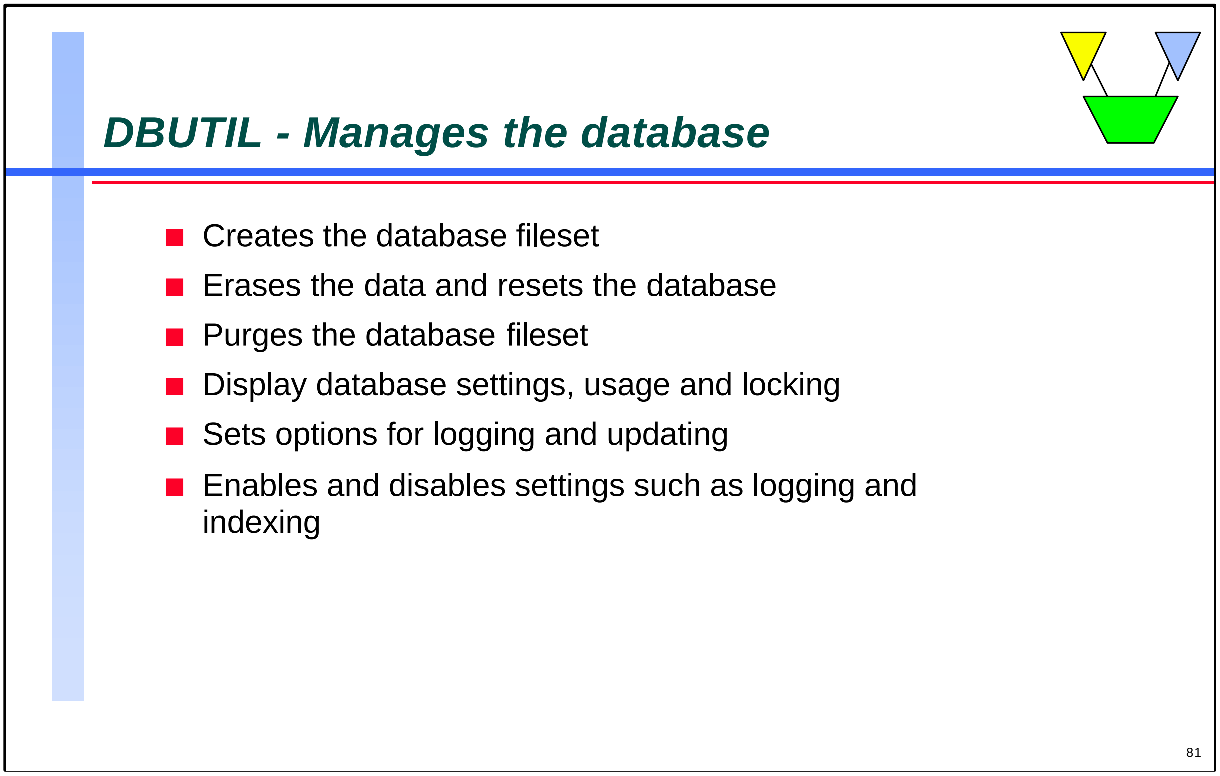
DBSTORE and DBRESTOR are backup tools.

DBUNLOAD and DBLOAD are for dumping the data from a dataset into a file, and reloading from that file. Seldom needed now that we have database altering tools like Adager.

DBRECOV is to recover a database after a system failure. You must have transaction logging enabled on the database. Restore the database from a backup, then use DBRECOV to reapply the transactions from the logfile.

For Techies

References

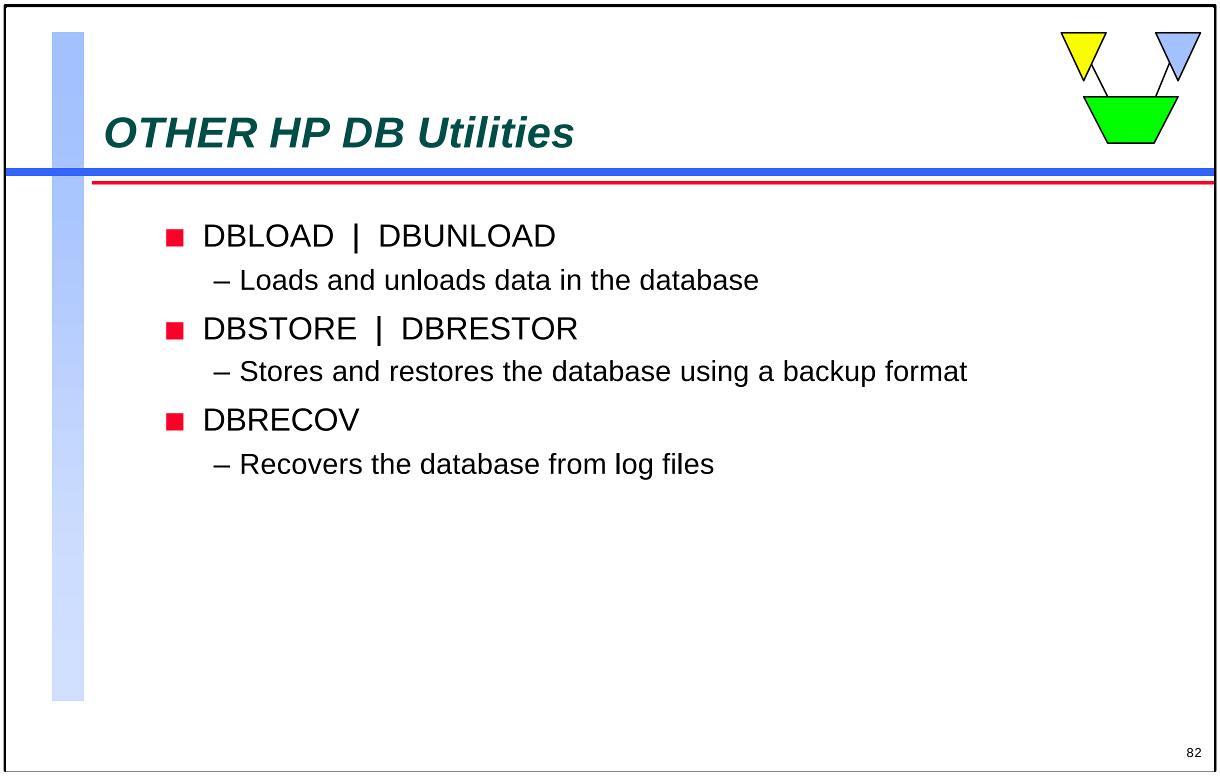


DBUTIL - Manages the database

- Creates the database fileset
- Erases the data and resets the database
- Purges the database fileset
- Display database settings, usage and locking
- Sets options for logging and updating
- Enables and disables settings such as logging and indexing

For Techies

References



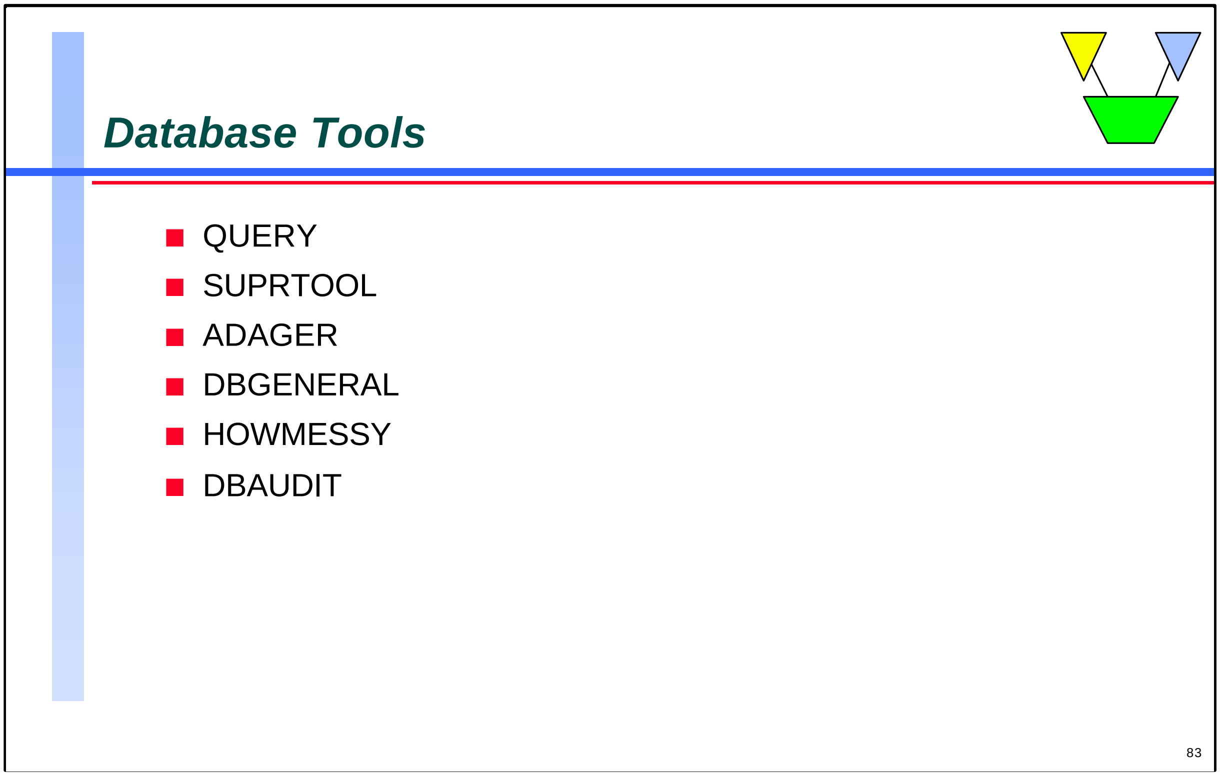
OTHER HP DB Utilities

- DBLOAD | DBUNLOAD
  - – Loads and unloads data in the database
- DBSTORE | DBRESTOR
  - – Stores and restores the database using a backup format
- DBRECOV
  - – Recovers the database from log files

Never restore only part of a database to try to recover data.

For Techies

References



Database Tools

- QUERY
- SUPRTOOL
- ADAGER
- DBGENERAL
- HOWMESSY
- DBAUDIT

Tools to manipulate the database and the data in it and optimize its performance.

SUPRTOOL, the “database handyman”, is a general-purpose reading, writing, sorting, extracting, selecting, transforming, web-enabling, linking, tool for data files.

Adager and DBGeneral are for maintaining and restructuring databases.

HowMessy reports on internal efficiency of database pointers.

DBAudit produces reports from transaction logfiles.

For Techies

References

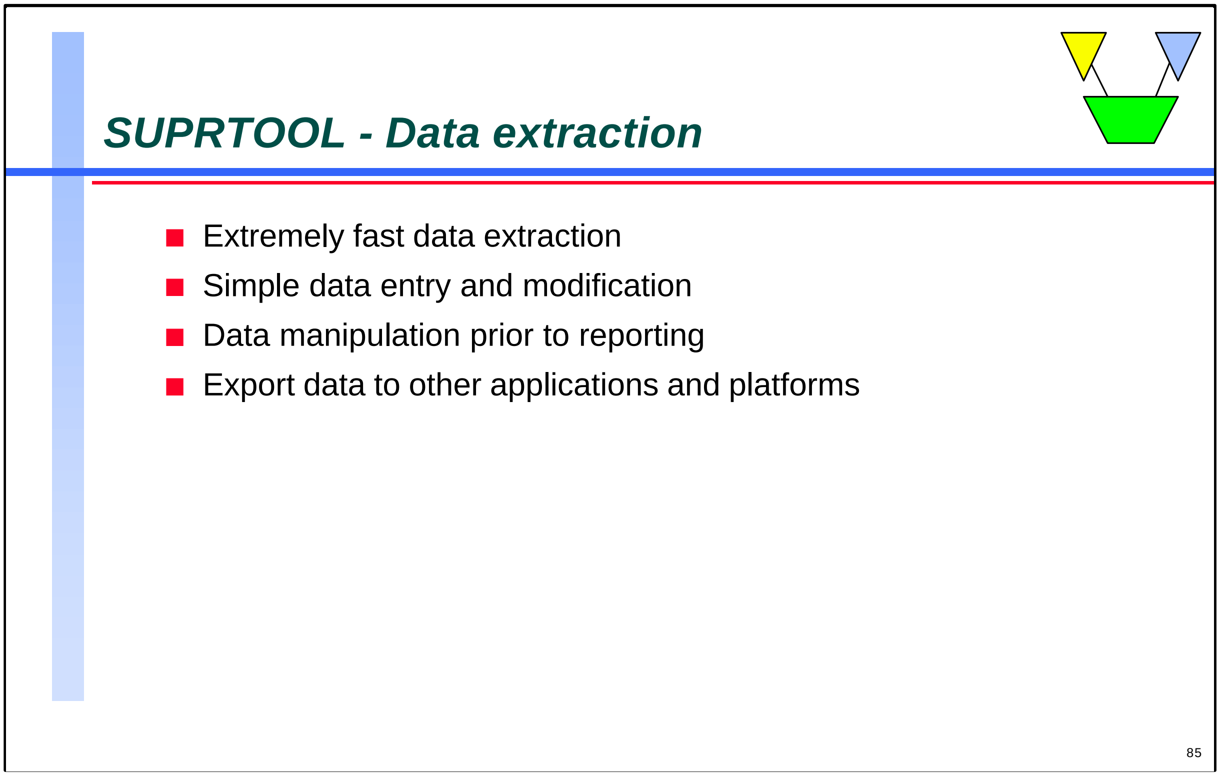
QUERY - Database access tool

- Simple data entry capability
- Modification and deletion of data
- Data retrieval
- Simple data reporting

Useful for adding test data to a database, producing simple reports, finding invalid data and deleting unwanted records.

For Techies

References



SUPRTOOL - Data extraction

- Extremely fast data extraction
- Simple data entry and modification
- Data manipulation prior to reporting
- Export data to other applications and platforms

85

Use Suprtool for its extremely fast selection and sorting ability prior to generating reports with tools such as COBOL or Cognos.

Use it to format data prior to exporting it to other applications and platforms - e.g. Excel, Oracle

For Techies

References

ADAGER | DBGENERAL

- Perform database tuning
- Add paths, fields, items, sets
- Move or copy databases
- Change capacities
- Repack paths
- Reblock datasets
- Build a schema

These utilities allow you to make changes to the database structure without having to unload the data from the database. For large databases this is essential.

For Techies

References

HOWMESSY - Database performance analyzer

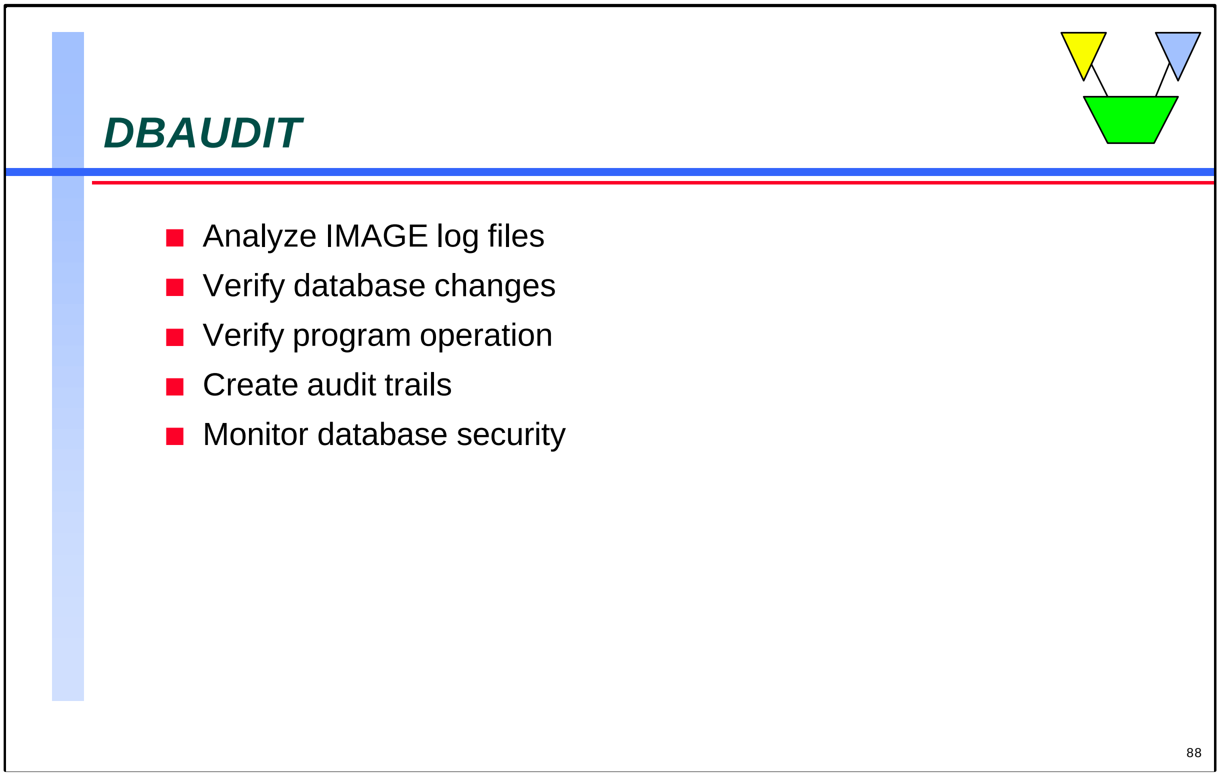
- Migrating Secondaries
- Inefficient Paths
- Insufficient capacities
- Poorly designed key fields

Run HOWMESSY against your databases regularly (e.g. monthly) to analyze performance and check for capacities which may be nearing their limits.

For Techies

References

See the IMAGE Performance tutorial, also at this conference.



DBAUDIT

- Analyze IMAGE log files
- Verify database changes
- Verify program operation
- Create audit trails
- Monitor database security

For Techies

References

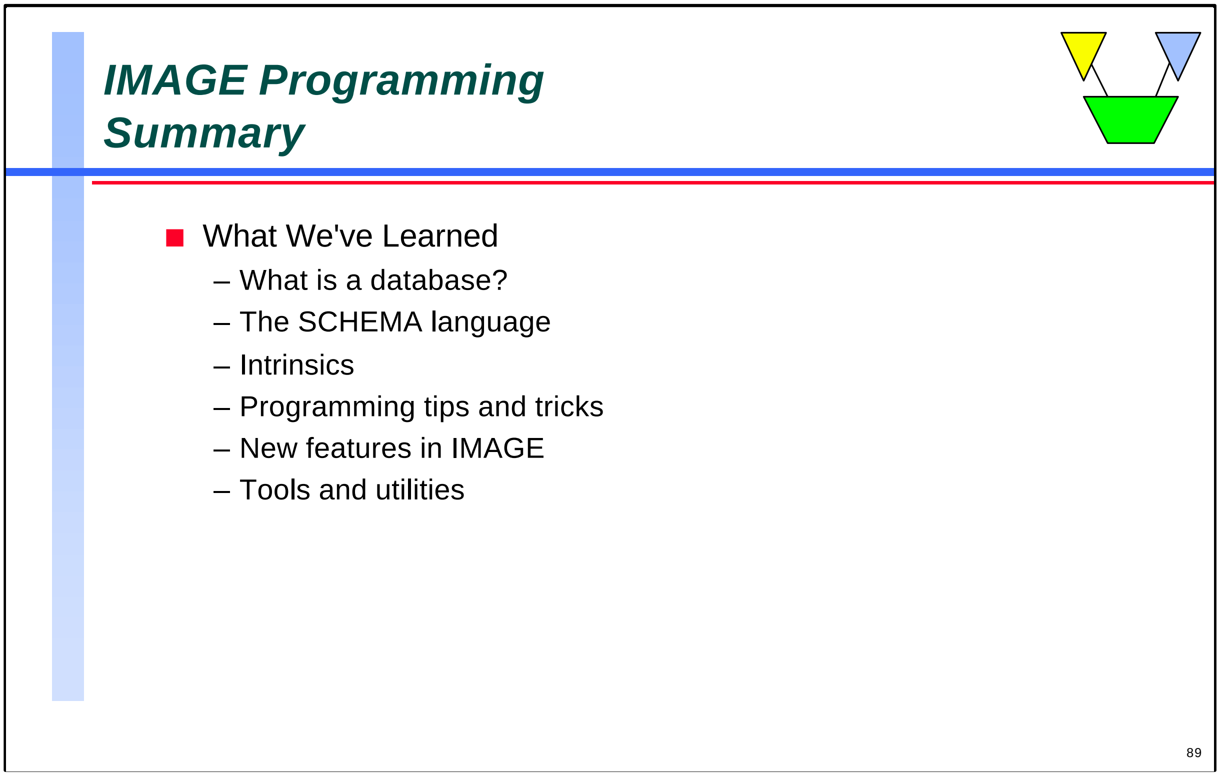
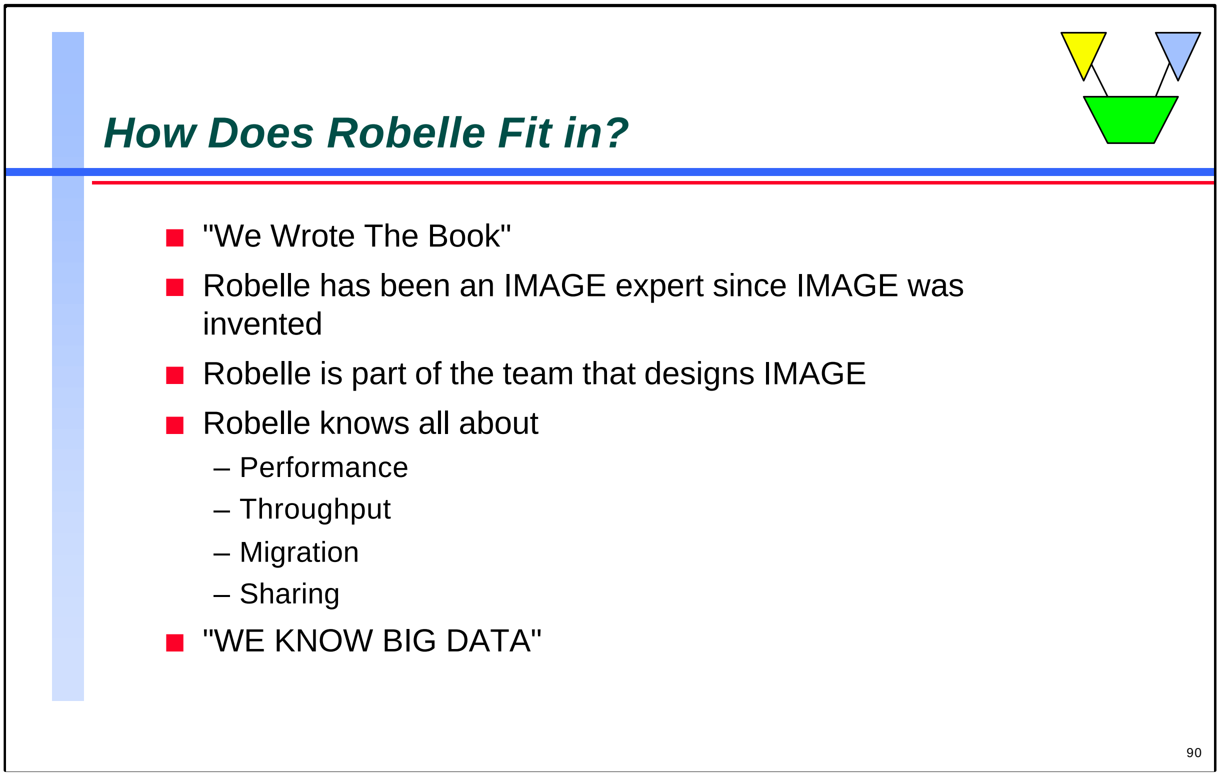


IMAGE Programming Summary

- What We've Learned
  - What is a database?
  - The SCHEMA language
  - Intrinsics
  - Programming tips and tricks
  - New features in IMAGE
  - Tools and utilities

For Techies

References



How Does Robelle Fit in?

- "We Wrote The Book"
- Robelle has been an IMAGE expert since IMAGE was invented
- Robelle is part of the team that designs IMAGE
- Robelle knows all about
  - Performance
  - Throughput
  - Migration
  - Sharing
- "WE KNOW BIG DATA"

For Techies

References

Where to Get More Information

- The IMAGE/3000 Handbook
  - ISBN 0-914243-00-4
    Published by Wordware
    P.O. Box 14300, Seattle WA 98114
- HP 3000 Quick Reference Guide
- TurboImage /XL Database Management System Reference Manual
  - Available on the web from HP at
    http://docs.hp.com/dynaweb/smpe/ for MPE/ix 6.0
- HP3000-L mailing list
  - Ask Questions! Answers are generous...

For Techies

References