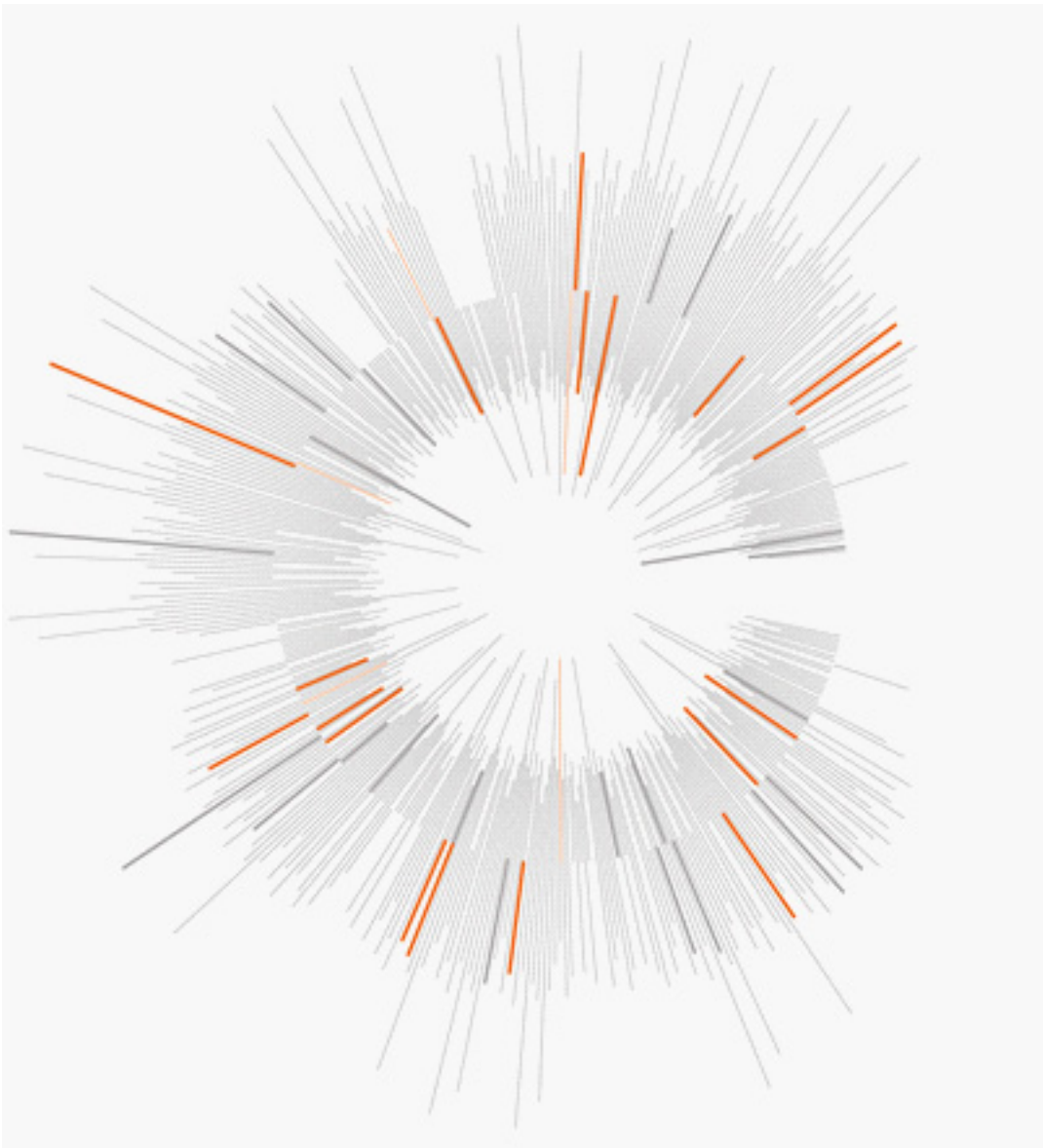


A guide to database subsetting

By Huw Price



Contents

Introduction.....	3
Why subset?.....	3
Setting up a subsetting project.....	9
Subsetting methods.....	11
Keeping subset databases up-to-date.....	16
Subsetting management.....	19
Data masking or data obfuscation?.....	22
Building the subset logic.....	24
Understanding the data model.....	28
Selecting the appropriate data.....	30
Primary key explosion.....	35
Interfacing with other applications.....	38
What next?.....	40
About Grid-Tools.....	40
About Huw Price.....	41

Introduction

Database subsetting is the process of extracting a cut down version of data from a larger database, typically a production database, and creating one or many smaller copies of that database for use in development or testing. Like most typical IT tasks, a database subsetting project needs to be well thought out and planned for it to be successful. In this paper, I have outlined some of the common tasks, methodologies and pitfalls of a typical project. Many companies have implemented subsetting with varying degrees of success and I will draw on a few examples throughout the paper.

As with all projects, there is a balance between practicality and pure theory. I am a strong believer in a pragmatic approach to project delivery and believe that database subsetting can be delivered in a short period of time; without losing too much application integrity. In addition, I think any work performed on a subsetting project should bring additional benefits in terms of overall data quality, and increased knowledge of the database structure and data flows throughout the application.

Why subset?

If an organization has been running an application for many years, there will inevitably be a build-up of information within the database, across any associated file storage and in data passed to other applications. Run times of month end jobs get longer and the need to constantly upgrade production and testing hardware is common.

Many home grown and virtually all packaged applications have no viable archiving strategy and data tends to grow inside databases. Over time your main transaction tables will include transactional data spanning many years. Users are usually very reluctant to remove data from an application as they are worried that there will be unforeseen consequences and cannot spare the time to investigate fully all effects of deleting data. If you are considering building an archiving strategy, you can consider using database subsetting as a step toward this; as all of the steps and methods can be transferred almost directly between the two projects. One man's subset is another man's archive!

While database growth in production can be managed by applying Moore's law, i.e. that technology will double exponentially every two years, it is rare that the same high spec equipment will be used in development and testing.

The result of this is that, typically, developers and testers will be using full size production data on low powered hardware. A job that runs in three hours in production could easily run for six in development. Reducing the run time of jobs in development is one of the many reasons why database subsetting should be considered.

Disk savings

Justifying cost is a major consideration for all project managers when looking to implement new procedures and methodologies. One perennial problem with infrastructure projects, such as database subsetting, is foots the bill? Project funding is usually controlled by users of the application who will carefully cost out a new feature, and project teams who are expected to bring in the new features on or under budget. Presenting a project proposal that would just seem to benefit the development team can be tricky and will usually be rejected, as immediate revenue generating projects will take precedence. For example, bringing to market a new type of investment bond as quickly as possible is often considered more important than taking your top analyst off a project for a month.

For these reasons, the cost of infrastructure projects tends to be spread across multiple cost centres, meaning that you now have to struggle to get agreement from all users, which can lead to politicking, and usually involves some kind of horse trading. Some organizations will already have budgets set aside for infrastructure projects and, if you are lucky and can tie the subsetting projects to cost savings, then you are in with a chance of implementing earlier rather than later.

One of the most effective cost justification areas is disk. In a large organization it is not uncommon for up to twenty copies of a production sized database to exist. These copies can be used for development, unit testing, regression testing, performance testing and so forth. Usually copies of production are taken once or twice a week and then the copy is moved into different regions when requested. When calculating the cost of disk, the management of the copies should be considered. While a one terabyte disk for your PC may cost \$800, the internal cost is much higher. Most organizations nowadays have a much more realistic cost per megabyte. While it might seem that a disk is cheap, the cost of managing the disk, backups, DBA time, disaster recovery etc. can cause total costs to grow quickly. A typical internal cost of \$200 per Gigabyte is not uncommon.

While you will not eliminate all full sized copies and you may actually increase the total number of test and development databases (see later), the cost savings are quite straight forward and can be expressed in a simple formula:

D = Cost per Gigabyte
 S = Database size in Gig
 NP = Number of current production copies
 Perc = Percentage reduction in size

Cost Savings = $NP * D * S - (NP * Perc * D * S)$

An example would be

D = \$200
 S = 1 Terabyte = 1,000 Gig
 NP = 18
 Perc = 5%

Cost saving = $1,000 * 18 * 200 - (1,000,000 * 0.05 * 200 * 18) = \$3,420,000$

In reality, you are likely to increase the number of database copies so this number may be closer to \$2.5Million. This, however, is still a significant number and should allow you to justify the initial and ongoing costs. It should be remembered, however, that this number only really makes sense at the time you are forced to buy more disk, which is usually when you are just about to run out of space. CIOs should try and plan ahead and try to implement subsetting prior to crisis points, allowing them to negotiate better deals with service and infrastructure providers.

Testing time

Testing with large production size databases creates many challenges, especially when it comes to testing batch processes. Typically these are end of period processes, for example, closing out this month's transactions and posting the balances to the general ledger. These batch jobs can run for a long time and if an error occurs can require the entire database to be reset or rolled back. It is not uncommon for batch jobs to run for several hours and for end to end processes to span tens of hours.

One of our customer's end of month processes took 30 hours to complete; testing any changes to this on a full size production copy meant that only one or two changes could be made to the production codes set, as testing just took too long. After building a subset database the month end run was reduced to 3 hours, allowing many more changes to their production code sets.

Number of testing databases

Typically many developers or testers will use the same database at the same time to test an application. From time to time users will request a database refresh and a new copy of production data is copied over and testing continues. The problem, of course, with this approach is that some testers may be in the middle of a complex series of tests where data has been entered manually, this data will be lost. The use of subset databases can solve this problem as different teams can 'draw down' subset databases without interfering with other users.

One of our customers, who had invested time in database subsetting, changed their development strategy and each developer would import a subset database into their own local RDBMs, in this case Oracle, for each new piece of development work. The DBAs automatically prepared several different flavours of subset databases each night and the developers would choose and import the appropriate one using standard database utilities.

Additional benefits

There are other benefits which are not immediately obvious but can dramatically improve the entire testing and development framework. These include:

- Data Comparison – Checking the underlying data during the testing process is an easy and low effort method to confirm that an application is working and that any data is being added, changed or deleted correctly. Using a table compare utility is simple

to implement and can be easily added to any kind of automated test harness. The problem, of course, with full size databases is that this comparison can run for a long time, however, comparing a few million rows in a subset database can take a few minutes and is a highly practical way of verifying application changes.

- Data Checking and Data Quality – As part of setting up the subset database you will need to define key business and referential relationships between tables. This information can be further used to look for any data errors in production and in testing databases. It is very common for data errors to creep into databases, especially large complex ones with multiple data interfaces. Being able to identify these data errors is a very useful feature. Just like data comparison, these data checks can be automated and included in any automated testing procedures.

Automatic Check For Overdraft Protection
FUNCTIONAL VARIATIONS

[Synthesis of NEW tests specified.]

Functional Variations for:

Give_OD:-Checking AND Big_Money AND not Current_OD AND Low_ODs AND
[Bus_Client OR Preferred].

[] - Refer to node(s): **INT-1**

1. If Checking and Big_Money and not Current_OD and Low_ODs
and [Bus_Client

OR Preferred]
then Give_OD.

2. If not Checking

(and Big_Money MASKed and not Current_OD MASKed and
Low_ODs

MASKed and [Bus_Client OR Preferred])
then not Give_OD.

3. If not Big_Money

(and Checking and not Current_OD and Low_ODs and
[Bus_Client

OR Preferred])
then not Give_OD.

4. If Current_OD

(and Checking and Big_Money and Low_ODs and [Bus_Client
OR Preferred])

then not Give_OD.

5. If not Low_ODs

(and Checking and Big_Money and not Current_OD and
[Bus Client

Figure 1- A Data check report

- Data browsing and Data editing – Once you have built up a set of rules to define the subset database you can further use these rules to browse the data as a ‘relational data editor.’ Systems and data analysts are always browsing actual data to look for specific issues or investigate how data flows through the system.

SQL Window #4 (Data Target) - TRAVEL - o10GAMST:TRAVEL

Oracle Schema Explorer

- COUNTRIES
- CREDIT_CARDS
- DEPARTMENTS
- DISCOUNT_AGENCIES
- EMPLOYEES
- EXCHANGE_RATES
- EXTERNAL_SEAT_RESERVATIONS
- FARE_SCHEDULES
- FLIGHT_BOOKINGS
- FLIGHT_DEPARTURES
- FLIGHT_ROUTES
- FREQUENT_FLYER_PROFILES
- GTSUBSET_DRIVER
- HOTELS
- HOTEL_BOOKINGS
 - Columns
 - Related Tables
- HOTEL_CHAINS
- HOTEL_FACILITIES
- HOTEL_FINANCES
- HSD_BUSINESS_RELATIONS
- HSD_DEPARTMENTS
- HSD_EMPLOYEES
- HSD_ORDERS
- HSD_ORDER_LINES

SQL #1 (100) SQL #2 (24) New

Data in HOTEL_BOOKINGS

All 24 rows Lucida Console Row 1 of 2

Booking Id	Hot Id	Arrival Date	No Of Nights
252434	5935	19/01/1997 00:00:00	2
433982	3679	01/05/1999 00:00:00	1
433983	3699	01/05/1999 00:00:00	2
433984	5822	13/04/1999 00:00:00	2
433985	5839	12/04/1999 00:00:00	1
433986	5922	12/05/1999 00:00:00	2
433981	5761	01/04/1999 00:00:00	1
434001	7071	01/05/1999 00:00:00	1
434002	7071	12/05/1999 00:00:00	1
434003	5761	20/02/2009 00:00:00	7
434004	5761	20/02/2009 00:00:00	7
434005	5761	20/02/2009 00:00:00	7
434006	5761	20/02/2009 00:00:00	7
434007	5761	20/02/2009 00:00:00	7
434008	5761	20/02/2009 00:00:00	7

Figure 2- Data Browsing and Relational Data Editing

Setting up a subsetting project

Subsetting can be a complex task, however, if you follow a systematic approach it can be accomplished quite quickly. In my experience, you should spend no longer than a couple of weeks for a typical 200 table application and no longer than six weeks for a larger 1000 table app.

It will take a few iterations to get it right and there will be a degree of trial and error. “Did the end of day job run correctly? No, the batch set up parameters needed to be updated first. Ok, let’s run that process separately and try again.”

As part of a typical subset project, other issues will be thrown up. Most typically, these are data quality issues or problems with navigating to some data due to ambiguous or overlapping data relationships. These other issues can slow you down as they may involve having to repair problem data or adjust the application to resolve data ambiguities.

The team

A lead analyst needs to be assigned, preferably with knowledge of the application to be subsetted. Most analysts tend to have specialist knowledge of certain areas of a database and access to other subject area specialists is essential. The access to analysts or users does not need to be full time, but prompt response to questions about data is mandatory. The underlying data itself usually contains the information needed to subset; however, a few quick emails to verify assumptions would be recommended.

The lead test analyst needs to be included as part of the team as they may well already have some strong testing technology and techniques, as well as opinions on how the subset database can be used to its full advantage. They may well suggest setting up several separate databases for different phases of testing or projects.

The DBA team needs to be fully involved in the project. In many ways it makes sense for the DBA team to lead; they are the team that usually manages the setting up of the database itself and will already have strong expertise. However, DBAs, by the nature of their job, will be familiar with the structures of the tables, but not any underlying business relationships and data flows.

DBAs will certainly have opinions on the best way to manage the movement of data, run times of data copies, imports etc. They should really lead in determining the most appropriate technology and techniques to move data around see later.

System specialists need to be included, as they usually hold the purse strings when it comes to creating new file systems and deciding on which particular hardware the test databases are to be deployed.

The technology

Deciding on the appropriate technology is important and it may be worth you running a few test data migrations using different techniques. Your DBA will already have tried a few different approaches I’m sure. In my opinion, it is worth using standard database utilities to move data. These utilities are well supported by database vendors and are usually rock solid, well documented, and well tuned in terms of run times.

One general point worth noting is that data movement is usually much slower when data has to be converted between different character sets. You may not realize you are doing this however if you issue commands such as:

```
INSERT INTO NEW.XYZ SELECT * FROM OLD.XYZ
```

and compare this with

```
EXP TABLES=OLD.XYZ
IMP FROMUSER=OLD TOUSER=NEW
```

The run time of the export utility is several orders slower even though the I/O is the same. This is because the data has to be converted between character sets behind the scenes. If you perform as much work as you can inside the database you will usually see better performance.

That said, database utilities nowadays support parallel exports and loads, and can take advantage of database internals such as fast path loading, delayed index building etc. and do offer real flexibility.

Some tools on the market rely on a complex server infrastructure whereby all of the data is moved into flat file structures and then moved on into the test databases. As well as being more cumbersome, these tools tend to be much slower and, strangely, much more expensive (I guess you have to pay for the extra infrastructure!). If you are looking at these tools be sure to compare the run time of the tools versus standard database utilities.

Other key areas that need to be checked are:

- How do you wish to invoke the sub set process? Most sites already have a job scheduler and the easiest way is to use this, build up or generate a set of scripts using a tool and then use your normal job control tools to manage them.
- Will it be an ad-hoc or automated process? Setting up a daily or weekly export of subset data can be a good option. The individual testers can then import the subset data whenever they wish. If you link this with a control parameter table, which can contain specific row keys or general selection criteria e.g. export CUSIP 123456 or export CLAIMS in Zip 10011 (see further on for details of this technique), you can have the best of all worlds.
- Do you need to move image data or other non standard data types? If your database contains supporting images or document data, be aware of some of the restrictions of the data utilities. Exporting image data to flat file, then using that data with a flat file import utility may well create problems with non standard characters. Ideally you would like to avoid having to set up a separate image movement step.

Tool selection

There are numerous tools on the market to help you subset databases. If you want to see and try one I designed please take a look at http://www.grid-tools.com/datamaker_key.php and feel free to download a cut down copy at http://www.grid-tools.com/download_software.php

When selecting a toolset, here, in order, are my top criteria:

- Cost – Subsetting is a relatively straight forward task, even on the mainframe. Do NOT spend more than \$100K. The old adage “you don’t get fired for buying IBM!” is long gone.
- Being able to explore the data, and building a model as you go whilst validating the data, is extremely important. Data tends to have a life of its own and may not conform to a users’ external view of an application.
- The toolset should use native or standard database utilities to move data around. These are well supported and will normally already be in use at your site.
- The tool should be able to support multiple methods of data movement or architectures as you may well end up with a mixed approach to subsetting different databases.
- Try and avoid tools that use an intermediate architecture with proprietary file structures. It slows the process down and requires extra resources to administer.
- Make sure the subset processes can be run using your normal job schedulers.

Subsetting methods

Copying database internal files

If you think about it, the simplest way to create a subset database would be to copy a few underlying database files to a new machine, attach them to a new database and there would be your subset database. Copying files, even if they are a terabyte in size, is simple and quick.

Based on your particular RDBMS, this is sometimes possible. You may be using Oracle partitioning or have set up your SQL Server application to span multiple databases and be able to transport your subset partitions, or copy SQL Server mdf files, from production to development.

In reality, it is rare for applications to be designed from the ground up with subset (testing) or archiving (Information Lifecycle management) built in. Some enterprising DBAs may have implemented partitioning, however, the partition keys tend to be based on location or some simple date. When it comes to subsetting, it is rare that the simple partition methods can be of much use as usually the criteria for a subset database will span multiple partitions. The whole area of

partitioning is very interesting. If you wish to get in touch with me at huw.price@grid-tools.com, I will be happy to discuss some very pragmatic approaches to partition design that have proven extremely successful.

Archiving or purging data

A quick and pragmatic approach to building a subset database is to use existing or ad-hoc purge routines to remove large amounts of transactional data. The steps for this are quite straight forward:

1. Use database clone or copy technology, usually already in place, to create a copy of production.
2. Run purge routines to remove large quantities of data from the top 20 or so transactional tables.
3. Run a re-org to either migrate data to a new tablespace/data file or to reclaim space in existing data files.
4. Copy the underlying data files to your subset databases.

The key advantage of this method is that you are using the selection logic of the purge routine to retain a subset of data. The main drawbacks tend to be that either the purge routines run for a long time (deletes tend to be very I/O intensive) or that the purge logic is not as sophisticated as you may wish.

Extract and load

This is by far the most common method of building a subset database. Once you've built your sets and designed your criteria, the export utility will extract the pertinent data and create import files ready to be loaded. Typically these exported files are copied to another machine or to a central file system for testers to copy down and load as required.

Example 1 – A Unix Oracle Data pump parallel export

```

if [ "$1" = "" ] ; then
  echo
  echo \#\# ERROR SPECIFYING PARAMETERS
  echo \#\# TO RUN: Chains_export@p userid/password[@tnsname]
  echo
  echo
  exit -1
fi
echo Starting...
rm -f Chains_*.dmp*
sqlplus $1 @Chains_createdir.sql
expdp userid=$1 exclude=CONSTRAINT content=ALL directory=Chains parallel=4
dumpfile=Chains_TRAVEL_TRAVEL.HOTELS.dmp tables=TRAVEL.HOTELS >
Chains_TRAVEL_TRAVEL.HOTELS.out 2>&1 &
expdp userid=$1 exclude=CONSTRAINT content=ALL directory=Chains parallel=4
dumpfile=Chains_TRAVEL_TRAVEL.CITIES.dmp tables=TRAVEL.CITIES
parfile=Chains_export@p_CITIES.par > Chains_TRAVEL_TRAVEL.CITIES.out 2>&1 &
Chains_TRAVEL_TRAVEL.FLIGHT_DEPARTURES.out 2>&1 &
wait
echo Extract complete

```

Example 2 – A DB2 Mainframe JCL subset export

```

//*** *****
//*** 02 UNLOAD ***
//*** *****
//UNLOAD EXEC DEBATCH,SSN=DB2T,
//          PROC=DSNTIALL,
//          PLAN=ALTAULU,
//          PRTCL=*,LIB=HOLIB,PARM=SQL,COND=(4,LT)
//SYSRECO0 DD DSN=ALL2876.ITTH.D060911.T175000.SYSRECO0,
//          DISP=(NEW,CATLG,CATLG),DATACLASS=DPRMNG,UNIT=2MI2,
//          SPACE=(CYL,(1,5),RLSE)
//SYSRECO1 DD DSN=ALL2876.ITTH.D060911.T175000.SYSRECO1,
//          DISP=(NEW,CATLG,CATLG),DATACLASS=DPRMNG,UNIT=2MI2,
//          SPACE=(CYL,(1,5),RLSE)
//SYSRECO2 DD DSN=ALL2876.ITTH.D060911.T175000.SYSRECO2,
//          DISP=(NEW,CATLG,CATLG),DATACLASS=DPRMNG,UNIT=2MI2,
//          SPACE=(CYL,(1,5),RLSE)
//SYSFUNCH DD DSN=ALL2876.ITTH.D060911.T175000.SYSFUNCH,
//          DISP=(,CATLG,CATLG),SPACE=(TRK,(5,5),RLSE),
//          UNIT=STSDA
//SYKIN DD *
SELECT *
FROM TRAVEL.HOTEL_CHAINS
WHERE (CODE)
IN (SELECT L0.CODE
FROM TRAVEL.HOTEL_CHAINS L0
WHERE CODE = 'HY' )
;
SELECT *
FROM TRAVEL.HOTELS
WHERE (ID)
IN (SELECT L1.ID
FROM TRAVEL.HOTELS L1
INNER JOIN TRAVEL.HOTEL_CHAINS L0 ON L1.HCC_CODE = L0.CODE
AND LOCATION = 'CTY'
AND CODE = 'HY' )
;
SELECT *
FROM TRAVEL.AVAILABLE_ROOM_TYPES
WHERE (CURRENCY, HOT_ID, ROOM_TYPE)
IN (SELECT L2.CURRENCY, L2.HOT_ID, L2.ROOM_TYPE
FROM TRAVEL.AVAILABLE_ROOM_TYPES L2
INNER JOIN TRAVEL.HOTELS L1 ON L2.HOT_ID = L1.ID
INNER JOIN TRAVEL.HOTEL_CHAINS L0 ON L1.HCC_CODE = L0.CODE
AND LOCATION = 'CTY'
AND CODE = 'HY' )
;

```

If possible, extract as much data as possible in parallel. Make sure that when you extract the data the selection query or the table joins use indexes; maybe run an explain plan prior to running the extracts.

Many sites do not like the additional load on production databases incurred by running extracts. I'm not convinced that the overhead is really worth worrying about, as usually an export is similar to running a medium complex batch report. As long as you run the extracts during a batch window there is not much system impact.

For sites that do not wish to impose additional load on production, it may be easier to subset a cloned copy of production. Many sites use mirroring technology or similar to create disaster recovery databases or reporting databases. You can then use these as the data source for your subset.

When using this method, you will have to decide who creates the table structures. Generally I prefer if DBAs use their normal utilities to create the database structures; these utilities are usually robust and well supported. If a DBA is called in to try and fix a database set up by you they may well leave you to sort out any problems. Some of the database export utilities do allow you to create the structures as part of the data migration; Oracle Export and Oracle Data Pump being two examples.

Shrink in situ

This is one of my favourite techniques as it is by far the fastest in terms of elapsed time, allowing you to reorganize any underlying tablespaces and apply database compression, while not requiring much work to manage any database procedural code or triggers. This method is particularly effective with packaged applications as there may well be many programmatic objects inside the database and making sure these get set up and enabled correctly can be very time consuming.

The key steps of this method are:

- Clone the entire production database
- Create shadow tables inside the database
- Insert data into the shadow tables
- Drop any foreign keys
- Drop the original tables
- Rename the shadow table to the original table
- Recreate any indexes and foreign keys
- Recompile any affected objects

Using this method, we can usually create a subset database 5 to 10 faster than using an export and import method. This method also allows you to mask or obfuscate the data at the same time (see data masking later), and allows you to create a staging database which can then be moved on to development and testing.

Hybrid in-situ and export/import

A variation on this method is a hybrid of shrinking and export/import. With this method, you must create a new schema within the core databases allowing the subset queries to write the data directly into the new schema with the appropriate selection criteria, for example, `INSERT INTO MOVE.CUSTOMER SELECT * FROM PROD.CUSTOMER WHERE STATUS = 'A'`.

Once the new schema is populated you can use normal database utilities to move the data from database to database.

Keeping subset databases up-to-date

One decision you must make when building subset databases is whether you wish to apply deltas or incremental updates to your subset database. If you make new subsets available on a frequent basis or on demand this should not really be needed.

Applying deltas to existing subsets

To apply a delta to a subset you can use some kind of log mining and playback tool (see later), or adjust the set criteria to pick up only new transactions since the last extract was run. However, this method has several problems, the key ones being:

- Do you also export updated transactional data?
- What do you do about deleted data?
- You may well have added new rows using database sequences, this could result in duplicate primary keys
- How do you reconcile any roll up or end of period processes that summarize data or build de-normalized copies of data?

Using data streams or log playback

Mining database logs and applying updates to remote databases is common practice nowadays. This technique is often used to keep reporting databases up to date, allowing database query activity to be moved to remote databases. This same technique can be used to apply any inserts, updates and, crucially, deletes to a subset database. For this to be effective, you will need to use some of the more modern log mining technologies, such as Oracle Streams; this technology allows

you to select which rows are to be updated. In the case of a subset, the same criteria used to build the original subset need to be applied to the streams control files.

If you are interested in this technique feel free to contact me for more details.

Create daily transaction data

A more effective technique than copying incremental data to your subset database is to create test data directly into the database. You can do this either by using capture playback tools such as AQA, QTP etc. and manually entering transactions through the UI, or using fully automated tools such as Datamaker. The advantages of this method are:

- You can base the transactions on real data; however, you can enhance and enrich that data to reach more complex parts of an application during testing. In our experience test cycles are reduced by over a third using this technique.
- You can create high volume data to simulate changes in business processing, for example, simulate merging legacy application data flows into a new application.
- You can generate data for new functionality. Up to now most users have avoided creating data, considering it too hard and too time consuming. However with the advent of new technology, the same work you used to create the subset can also be used to create the data.

Subsetting management

There are a few areas you need to be aware of in the day to day management of subsetting. These are detailed as follows.

Table differences

If you are moving data to a testing database which contains amended or new tables, in other words you have added some new columns and tables, some of the table migrations may fail. There are a few methods to solve this.

- When you create a new, not null, column assign a default value. The default value will then be applied during table loads.
- If you export the data to an intermediate external flat file structure you can amend the input load cards to apply a default value. Some tools, like Grid-Tools' Data Subset, will auto detect and restructure these for you.
- If you are using a staging area to migrate, you can apply the DDL differences to equalize the structures.

Reconcile Registered Objects

Version with Data Target | Data Target with Data Source | Version with Version

Reset [Icons] Register Changed & New Tables from

Compare Version: **Build 1** with Schema: **TRAVEL**

Object Name	Tags	New Objects
✓ ACCESS_CONTROLS	⚠ Changed	➕ ADDRESS
✓ ACCOUNT_PERIODS	↳ Copy of Default	➕ TEMP
✗ ADDRESS_USAGES	↳ Default	➕ TEMP2
✗ ADDRESSES	↳ DemoMultiDB	➕ TEST_CHAR
✓ AIRCRAFT_LAYOUTS	📦 FD_FFpayments	
✓ AIRCRAFT_TYPES	📦 FILE	
✓ AIRLINES	📦 GT CHILD ONLY	
✓ AIRPORTS	📦 GT PARENT ONLY	
✗ AVAILABLE_LANGUAGES	📦 GT RELATED	
✓ AVAILABLE_ROOM_TYPES	📦 GT UNRELATED	
✓ CAR_AVAILABILITY	📦 GTSample <1M	
✓ CAR_BOOKINGS	📦 GTSample <50K	
✓ CAR_HIRE_CHAINS	📦 GTSample <5K	
✓ CAR_HIRE_OFFICES	📦 GTSample Empty	
✓ CAR_RENTAL_PROFILES	📦 GTSubset Empty Tabl	
✓ CAR_TYPES	📦 GTSubset Ignore Tabl	
✗ CATEGORY_TRANSLATIONS	📦 GTSubset Large Table	
✓ CG_REF_CODES	📦 GTSubset Reference	
✓ CHEAPER_FARES	📦 GTSubset Small Table	
✓ CITIES	📦 Large	
✓ COUNTRIES	📦 LOV	
✗ COUNTRY_CODES	↳ Michelle	
✗ COUPON_USAGES	✗ Missing	
✗ COUPON_USAGES_NEW	📦 Reference	
✓ CREDIT_CARDS	↳ SDMmultiple	
✓ CUBE_FLIGHT_AIRPORTS	↳ Travel Demo	
✓ CUBE_HOTELS_AIRLINES	📦 Travel System	
✓ CUBE_HOTELSDRIVER_DIM	✓ Unchanged	
⚠ CUBE_TRIPS_TEST_1	📦 Used in Set	



Figure 4 – Table structure differences

Be aware of any de-normalization or aggregate processing

As part of your analysis of the database and application, you need to understand any column aggregation that occurs. This aggregation could be as simple as summing up the total item amounts for an order and storing it in the order, or could be a complex end of day process which accesses pricing and discount information. This de-normalized information may need to be changed after the subset database has been loaded. Usually I add in a few post process steps to update the columns directly or to run the batch processes to populate these columns.

Managing time sensitive data

In some circumstances you might be able to 'date shift' the data as it is being extracted or loaded. Date shifting allows you to test with up to date data, which improves testing as data tends not to be already out of date. This, for example, prevents an order already being three days overdue when you start testing.

There are a few things to consider:

- Are all your dates in date type fields?
- Do you use accounting periods to control your dates?
- Is the date of birth important for your testing, i.e. do you want to retain the age of people?
- Would you consider creating data with the correct date into your database?

If you are lucky you may well be able to apply date shifting as part of the subset:

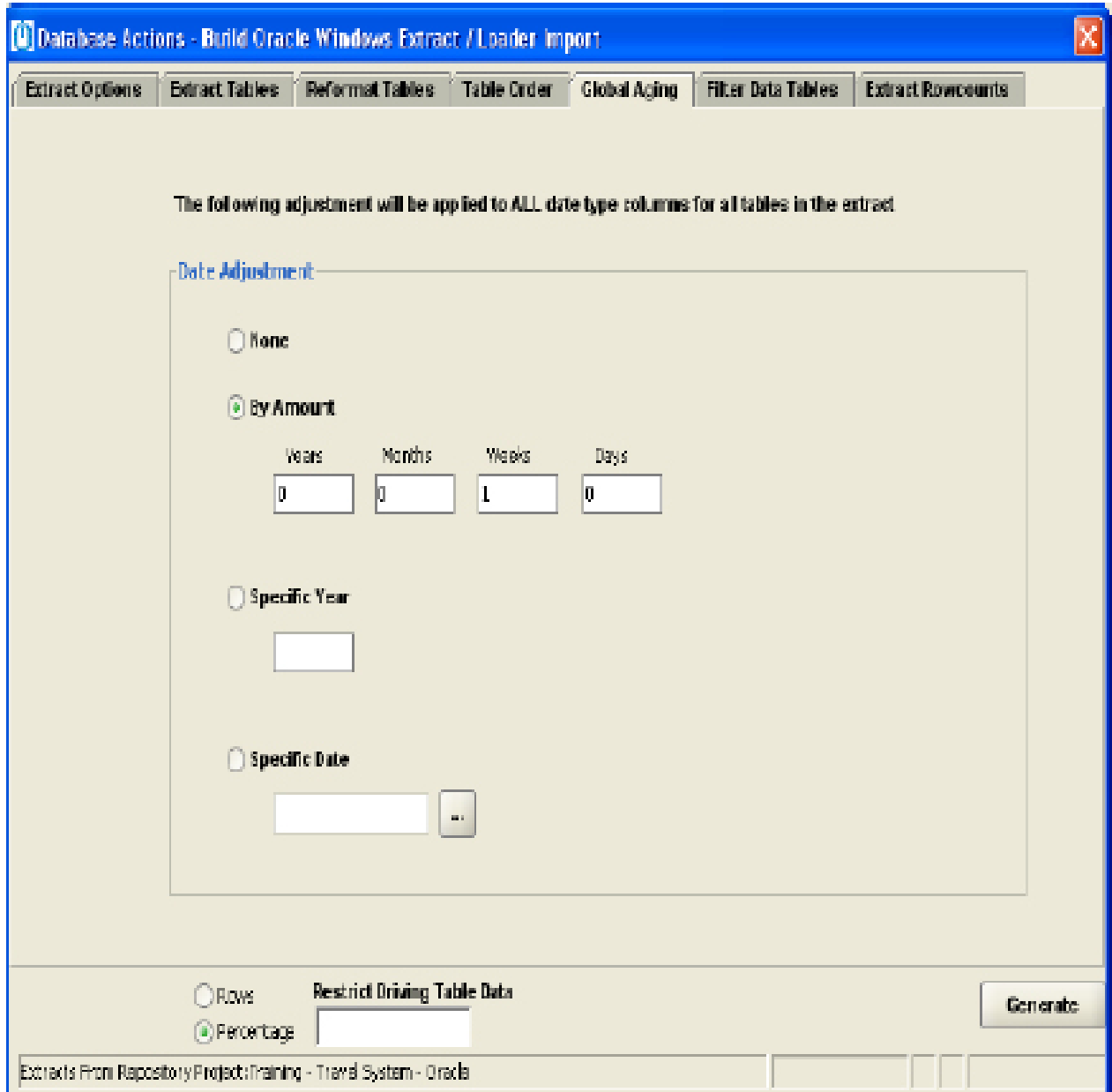


Figure 5– Date shift the data as part of the subset

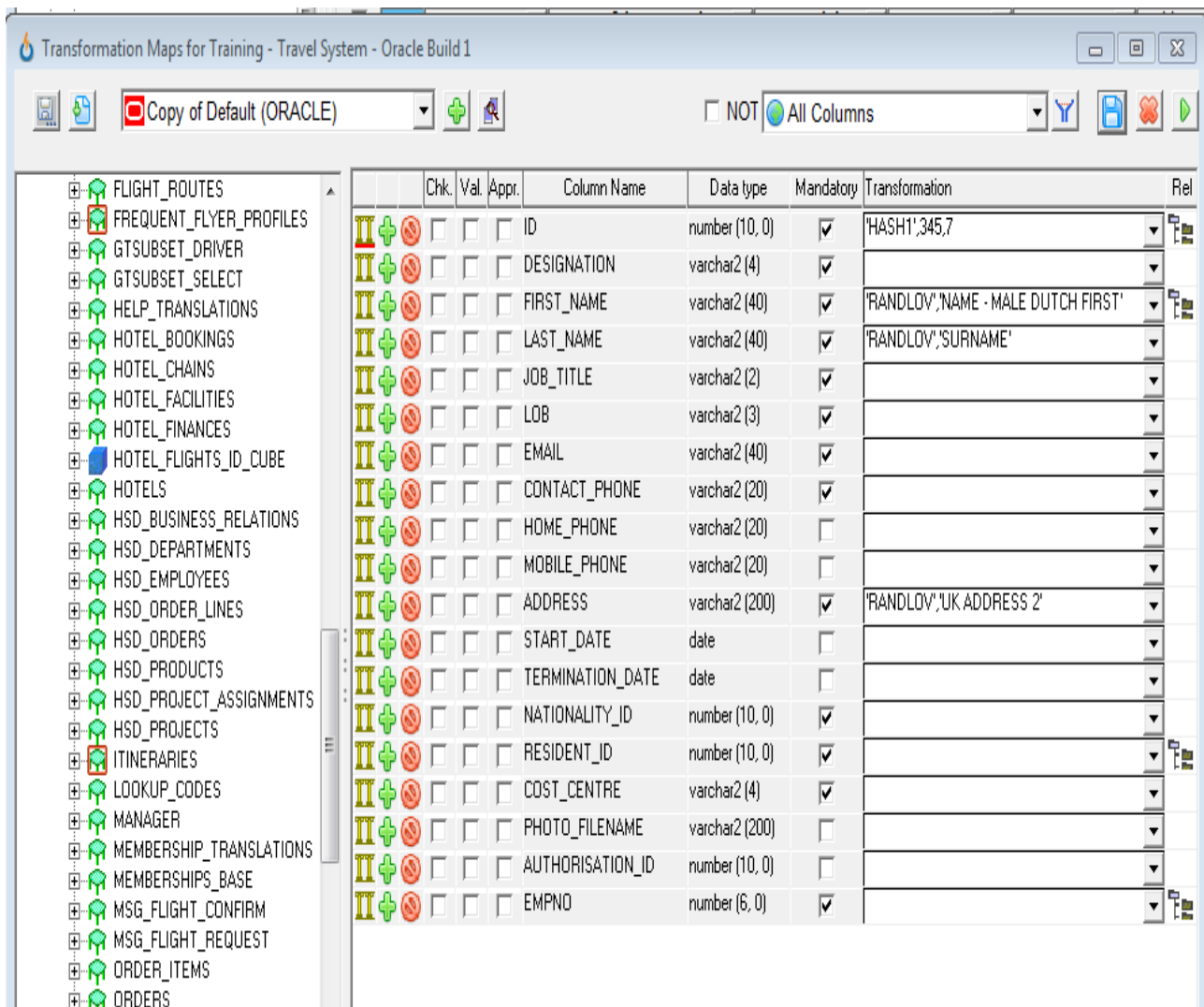
Data masking (also known as and data de-identification)

There has been a dramatic increase in legislation in the area of data privacy. This topic is a sizeable and I have written a separate paper on it (a copy can be obtained from our website or

alternatively, get in touch and I can send you a copy). In brief, data masking can be summarized as:

- You cannot identify an original customer, account or secure entity from the masked data.
- Overall data trends cannot be easily identified
- The minimum amount of data is used for testing.

In general, the work you have to do to set up a subsetting project flows almost directly into a data masking project. For example, if you have Customer_Account_Number in numerous locations you need to know this to subset and also to apply a masking function to all occurrences.



Transformation Maps for Training - Travel System - Oracle Build 1

Copy of Default (ORACLE) NOT All Columns

	Chk.	Val.	Appr.	Column Name	Data type	Mandatory	Transformation	Rel
FLIGHT_ROUTES				ID	number (10, 0)	<input checked="" type="checkbox"/>	'HASH1',345,7	
FREQUENT_FLYER_PROFILES				DESIGNATION	varchar2 (4)	<input checked="" type="checkbox"/>		
GTSUBSET_DRIVER				FIRST_NAME	varchar2 (40)	<input checked="" type="checkbox"/>	'RANDLOV','NAME - MALE DUTCH FIRST'	
GTSUBSET_SELECT				LAST_NAME	varchar2 (40)	<input checked="" type="checkbox"/>	'RANDLOV','SURNAME'	
HELP_TRANSLATIONS				JOB_TITLE	varchar2 (2)	<input checked="" type="checkbox"/>		
HOTEL_BOOKINGS				LOB	varchar2 (3)	<input checked="" type="checkbox"/>		
HOTEL_CHAINS				EMAIL	varchar2 (40)	<input checked="" type="checkbox"/>		
HOTEL_FACILITIES				CONTACT_PHONE	varchar2 (20)	<input checked="" type="checkbox"/>		
HOTEL_FINANCES				HOME_PHONE	varchar2 (20)	<input type="checkbox"/>		
HOTEL_FLIGHTS_ID_CUBE				MOBILE_PHONE	varchar2 (20)	<input type="checkbox"/>		
HOTELS				ADDRESS	varchar2 (200)	<input checked="" type="checkbox"/>	'RANDLOV','UK ADDRESS 2'	
HSD_BUSINESS_RELATIONS				START_DATE	date	<input type="checkbox"/>		
HSD_DEPARTMENTS				TERMINATION_DATE	date	<input type="checkbox"/>		
HSD_EMPLOYEES				NATIONALITY_ID	number (10, 0)	<input checked="" type="checkbox"/>		
HSD_ORDER_LINES				RESIDENT_ID	number (10, 0)	<input checked="" type="checkbox"/>		
HSD_ORDERS				COST_CENTRE	varchar2 (4)	<input checked="" type="checkbox"/>		
HSD_PRODUCTS				PHOTO_FILENAME	varchar2 (200)	<input type="checkbox"/>		
HSD_PROJECT_ASSIGNMENTS				AUTHORISATION_ID	number (10, 0)	<input type="checkbox"/>		
HSD_PROJECTS				EMPNO	number (6, 0)	<input checked="" type="checkbox"/>		
ITINERARIES								
LOOKUP_CODES								
MANAGER								
MEMBERSHIP_TRANSLATIONS								
MEMBERSHIPS_BASE								
MSG_FLIGHT_CONFIRM								
MSG_FLIGHT_REQUEST								
ORDER_ITEMS								
ORDERS								

Building the subset logic

Deciding on which data to extract is probably the most complex part of a project and will usually break down into a series of components. These are:

- The initial selection criteria, usually at an initial driving table.
- The sub tables associated with the driving tables and how these are spread out across the database. These sub tables may themselves have some kind criteria, which can enable you to issue more specific commands. For example, only extract active orders; do not extract more than a 100 items per bill etc.
- The intersection or union of different parts of an application. The subset database will be used by different testing teams each with their own needs. Each team will need data from other parts of the database and a superset of data needs to be built. You may, for example, have providers and claimants in a health care application; the common components of these would be claims. You would have to include a set of claims that satisfies the testing for both testing teams.
- Managing in-flight data. If an application is in constant usage you need to pick a time when transactional data is not flowing through the system or adjust your selection criteria to exclude this data.

Categorizing tables

Before spending too much on building your subsetting logic, a very worthwhile exercise is to categorize your tables.

Verify and Prepare Subset Schema - using Data Source connection : TRAVEL - o10GAMST

Schema Tables
Get Tables For Schema: TRAVEL

Save Scripts
Save Directory: C:\Users\zach\AppData\Local\Temp\...

Extract
 Repository File
 Open Extract

Small Limit: Set

Large Limit: Set

Table Name	Row Count	Small	Large	Reference	Subset	Ignore
ACCESS_CONTROLS	836	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ACCOUNT_PERIODS	789	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ADDRESS	10	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
AIRCRAFT_LAYOUTS	1307	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
AIRCRAFT_TYPES	23	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
AIRLINES	188	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
AIRPORTS	1903	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
AVAILABLE_ROOM_TYPES	182	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CAR_AVAILABILITY	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CAR_BOOKINGS	250	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CAR_HIRE_CHAINS	3	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CAR_HIRE_OFFICES	1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CAR_RENTAL_PROFILES	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CAR_TYPES	5	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CCP_ACCESS_CONTROLS	836	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CCP_ADDRESS	10	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CCP_GT_PARM	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CG_REF_CODES	36	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CHEAPER_FARES	25	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CITIES	1858	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
COUNTRIES	293	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 6 –Categorize Tables into Large, Small, Subset, Reference & Ignore

The categories I tend to use are as follows:

- Small – Move no matter what, it is not worth the effort subsetting them. I use about 10,000 as my threshold here.
- Large – Should be subsetted, over a million. It is probably worth obtaining some statistics from your DBA here. Tables which are growing rapidly should also be added to this category.
- Reference – Should be moved as it would affect the processing of new rows when testing. In other words, if you have a large pricing table, the tester would expect all prices to be available when testing not just the ones that have actually been used in production.
- Ignore – These tend to be work, transient or interface tables. These may well get populated during processing in the testing environment. This can also be used for tables which fall into the buckets “haven’t a clue what this is for” or “I think this an old table that isn’t used any more”. As you iterate through and validate what you can actually work with, the subset database will shrink.
- Subset – Tables to be subsetted.

Once you have completed the categorization you can perform some pre-validation of your potential subset database. The types of issues that can be resolved prior to actually running are:



Figure 7 – Example of a pre subset check

You need to be aware of database constraints when loading data into your subset database. Generally, I would suggest disabling foreign key constraints and then re-enabling them after population; life's too short! However you can calculate the load order based on the constraints.

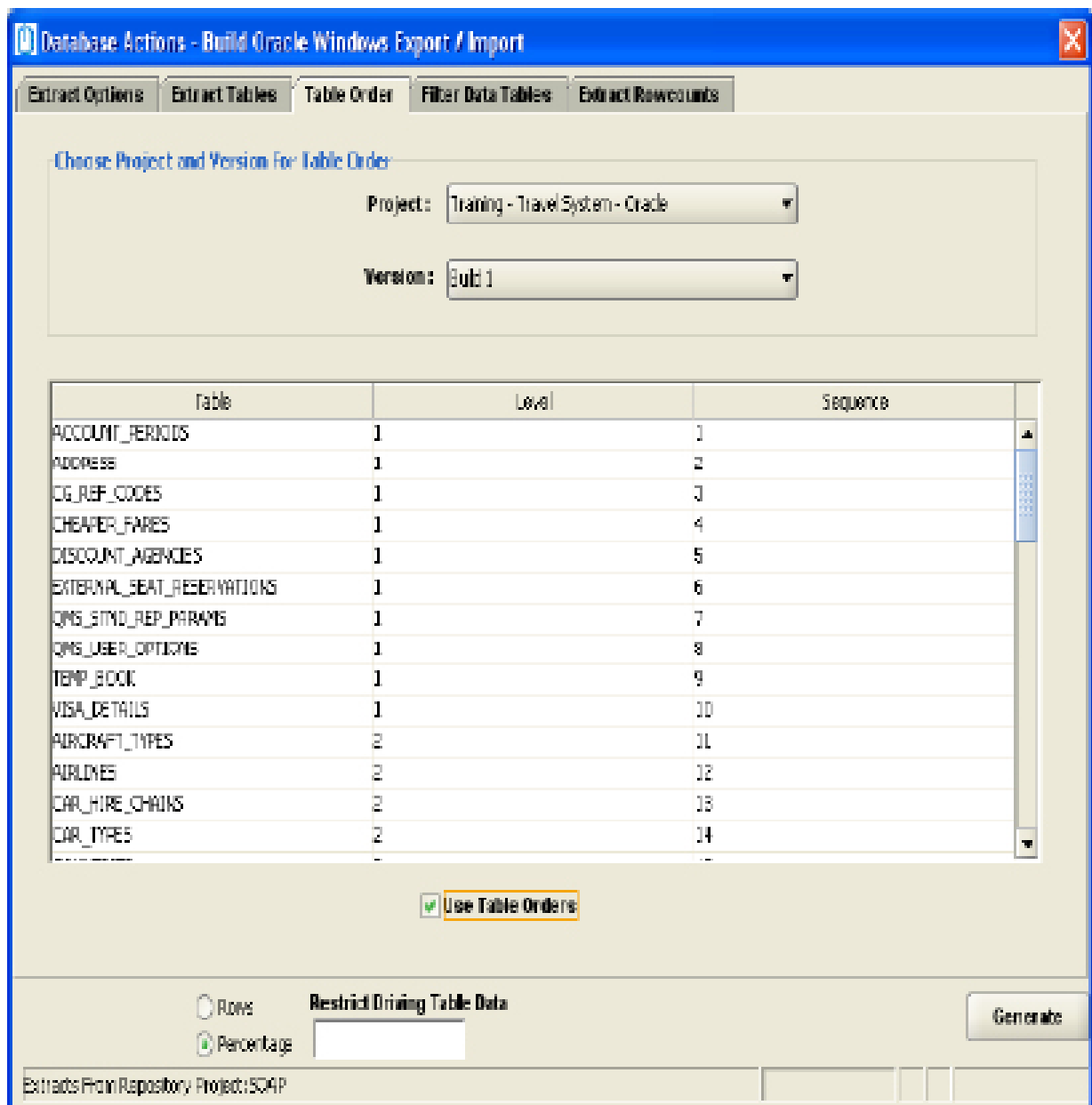


Figure 8 – A calculated load order

This load order should be used when loading the data into your subset database.

Understanding the data model

Before beginning a subsetting project you will need to spend time understanding your data. In reality you do not need to understand all of the data relationships (see categorization above), however, understanding the main transactional data is key to building a useful test database. To begin building up a picture you will need to gather all of the available ‘free’ information surrounding the key tables to be subsetting. This includes:

- Foreign Keys. How are tables related in the database?
- Documentation. This is usually held in a variety of formats and applications, however, they are rarely current.
- User knowledge. What is the users understanding of how and where key data is held and displayed?
- Naming standards. A surprisingly good source of information. Column names in tables can give a strong hint to their use and relationship to other columns.

Once you have gathered a basic picture, you will need to investigate the data itself to verify any documentation and try to understand in detail both where the data is held, and how it relates to other data. There are a number of problems common across most systems. These include:

- Data columns being used for multiple purposes. It is quite common for limitations in an application to be overcome by creative use of fields. Thus a field used for one purpose contains data to identify data for other uses.
- Invalid Data (see figure 1). As applications and databases evolve and merge with other systems, data may be created that is invalid. Users usually have an idea that this invalid data exists but have made the decision to ignore the data problems as there is no critical problem that would justify the time to clean it up. When it comes to subsetting, you need to decide whether to extract the invalid data or leave it behind. Orphan records are a typical example of this; they will generally not be extracted as they do not link to parent rows.

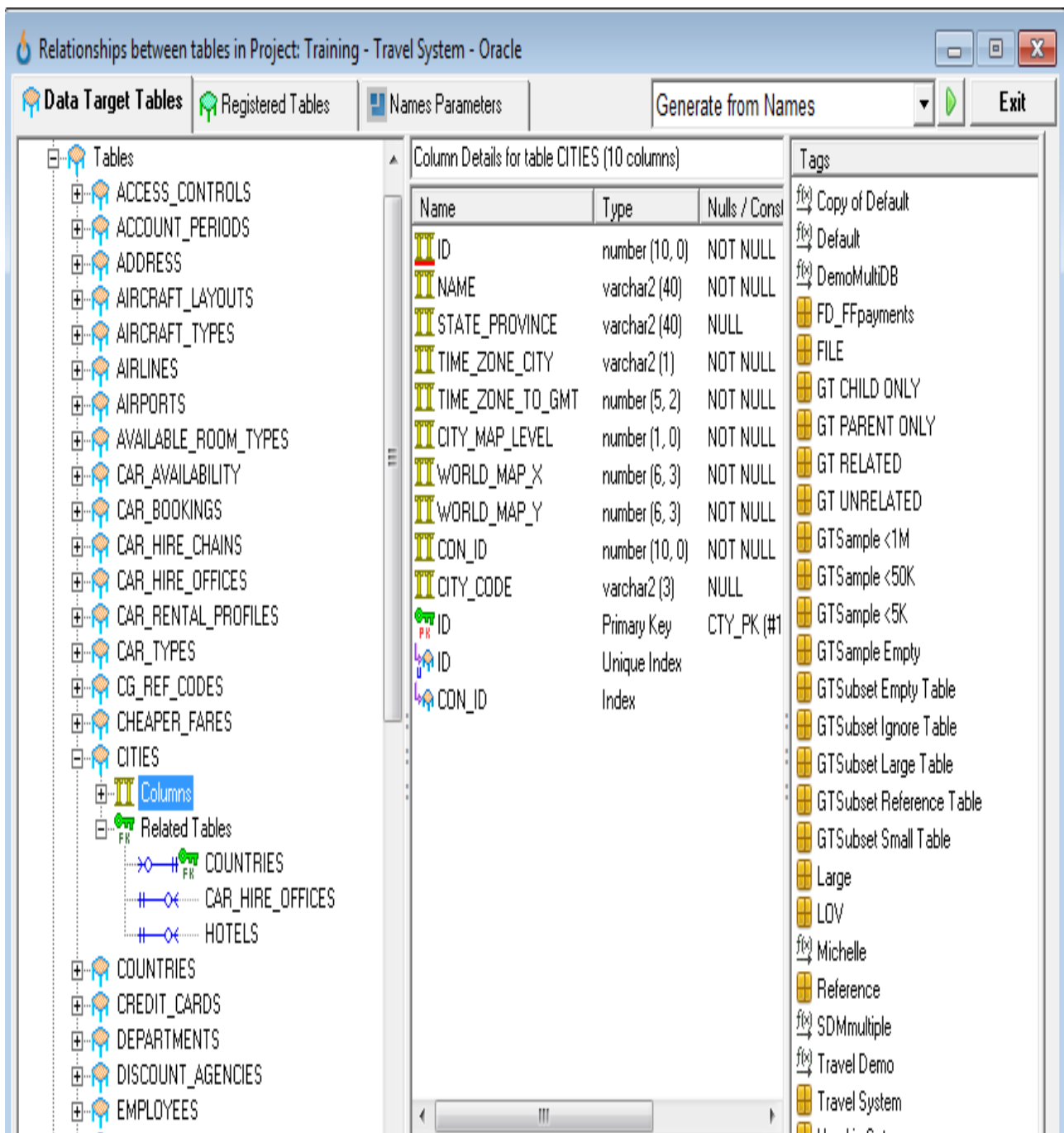


Figure 9 – Relationships and Table categorization

Selecting the appropriate data

As mentioned earlier you may well end up with several extract definitions each for a different logical component or business usage within the database.

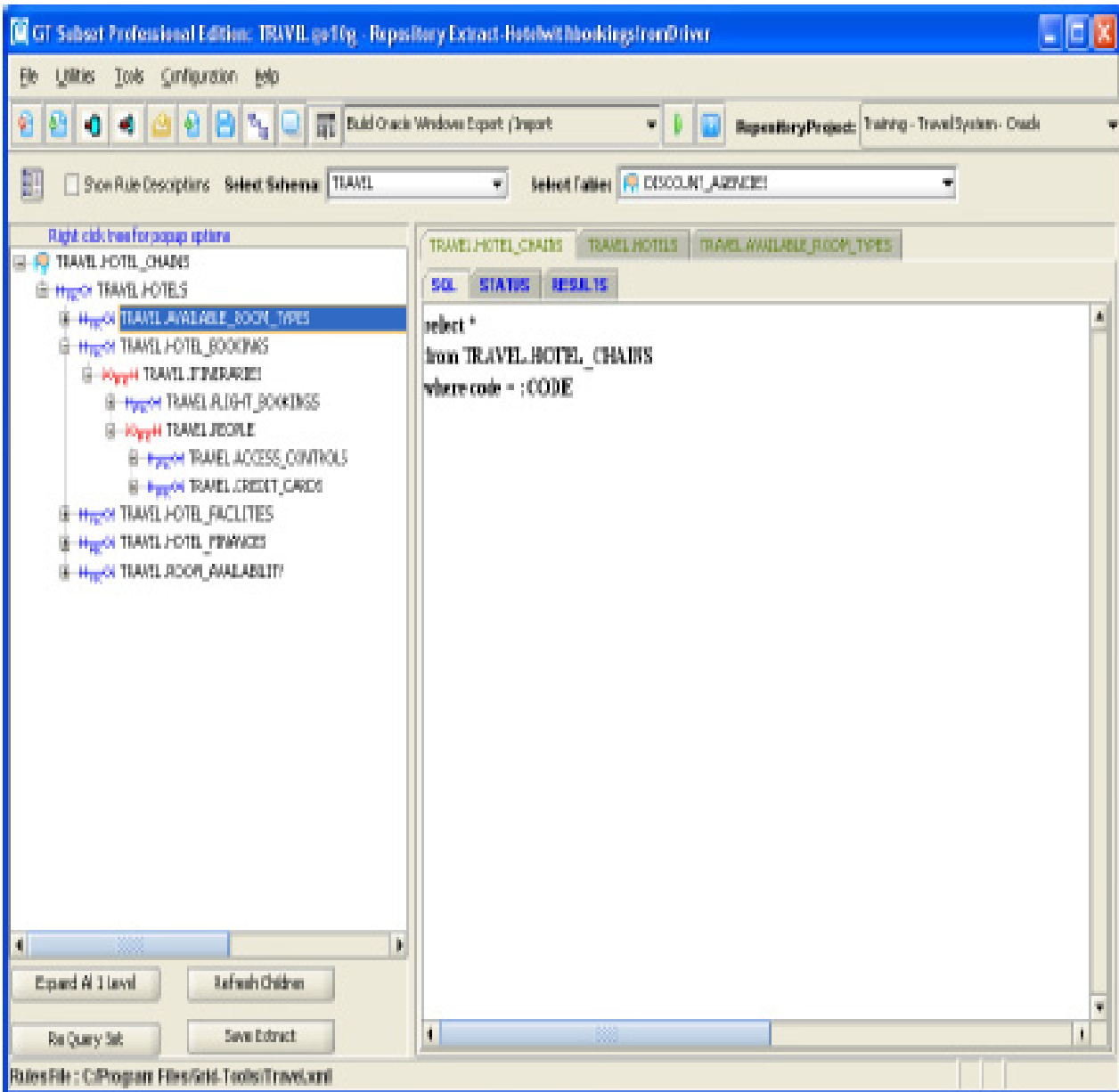


Figure 10 – An example of a subset definition

These extracts may well be run separately but must use some common set of driving criteria (see later). In the example above, we are starting with a particular HOTEL_CHAIN and working out from there. You can see that we will be extracting data from the table ITINERARIES, which is a parent (coloured red). You could potentially move from ITINERARIES back to other HOTELS and then onto other HOTEL_CHAINS, however, in this case we have just brought in the associated flight bookings.

In general, when building the set, ignore any of the tables of category Small, Reference or Ignore in the extract definition as this will usually simplify the process. Be aware of parent relationships; you may wish to extract parent tables in a separate extract.

For simple extracts, define standard parameters which drive the extracts, for example:

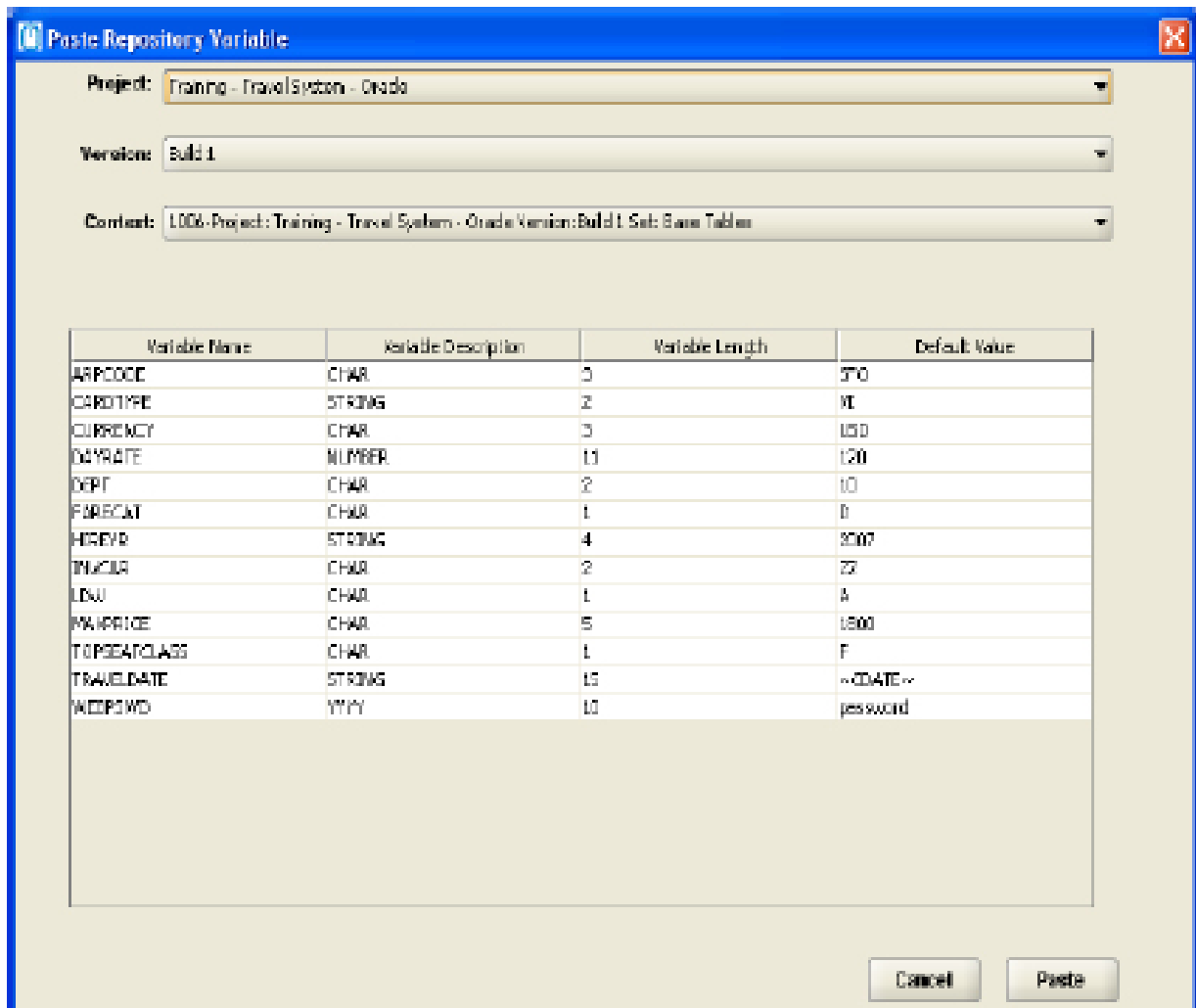


Figure 11 – Example driving parameters

Users can then set these prior to extraction and the extracts will extract the appropriate data.

Data intersection, data cubes and data chains

In Figure 10, you can see an example of a definition where it is possible to go back up to a parent and down to a child and so forth; HOTEL_CHAIN -> HOTEL -> ITINERARIES -> HOTELS -> HOTEL_CHAINS etc. If this chaining continues you would eventually extract the whole database. Being able to logically break the chain has to be a key part of your design; for example, you may wish to limit the extract criteria to only 3 levels. Testers would then have to restrict themselves to querying down no more than these three levels. A technique I prefer to use is to set up a driving table, such as the one below.

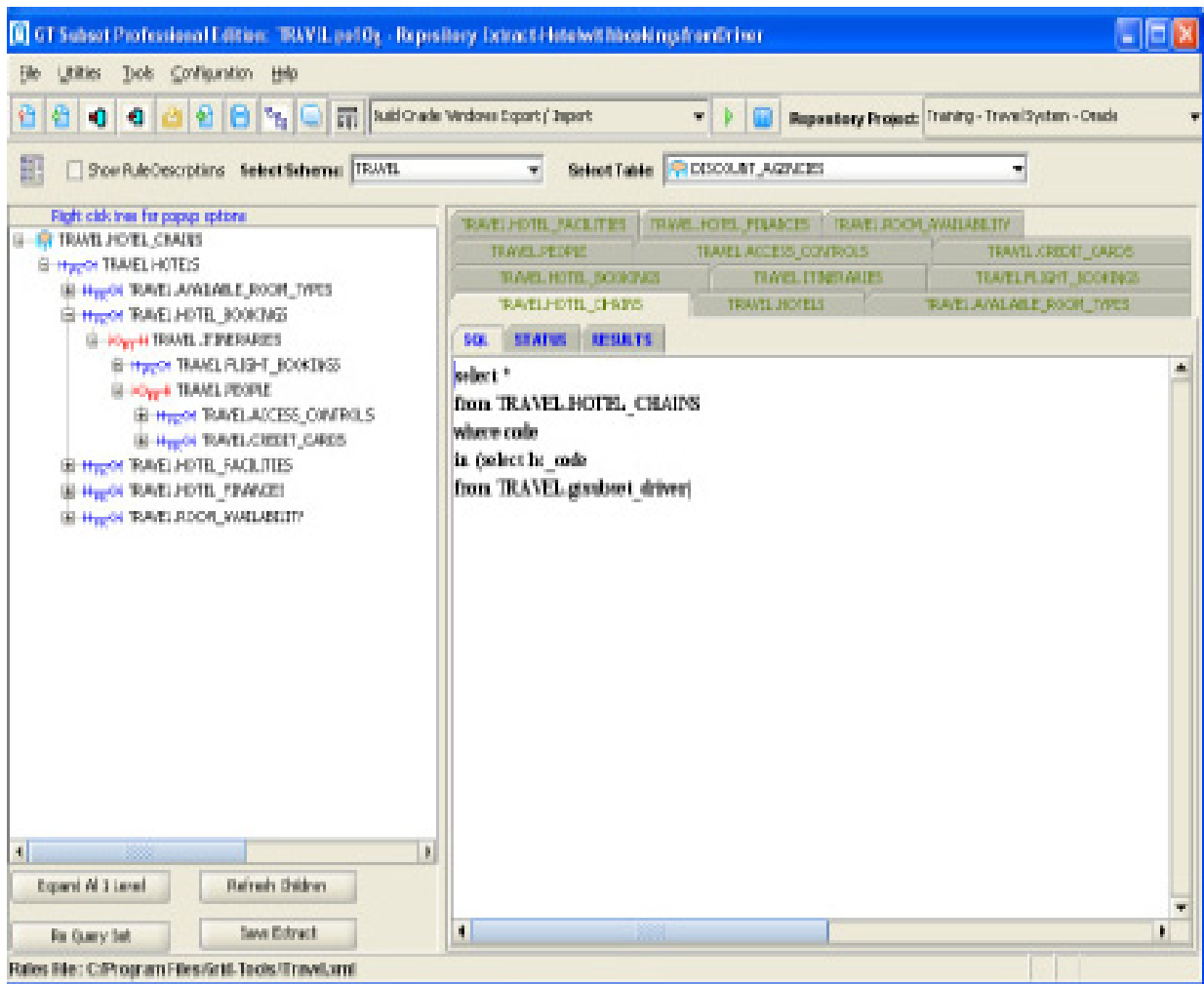


Figure 12 – A subset using a sub table to drive the extract

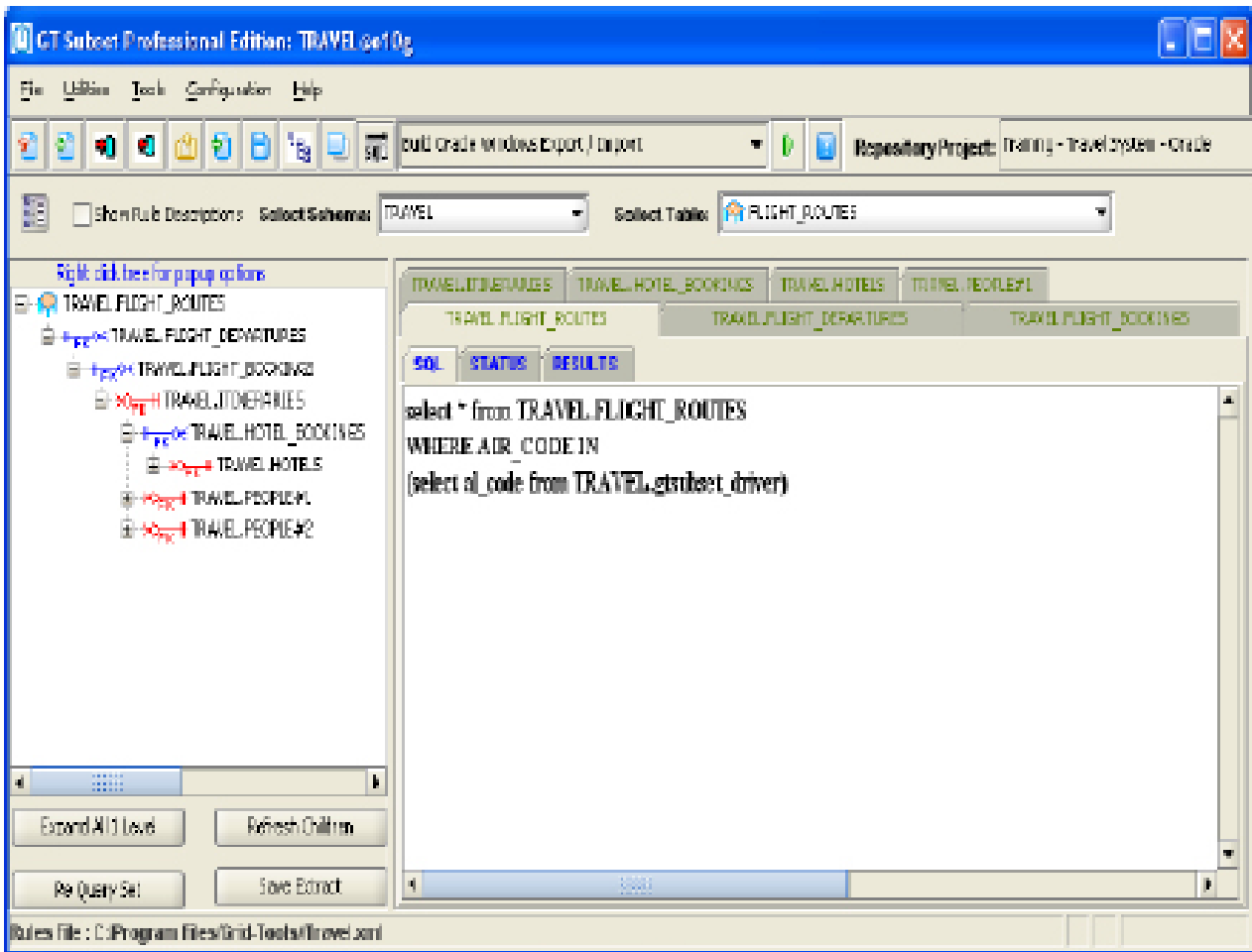


Figure 13 - A subset using a sub table to drive the extract

The extracts use the columns in the driver table to select specific data. The table GTSUBSET_DRIVER contains a list of primary keys used to drive several subsets. See figures 12 and 13 above.

Primary key explosion

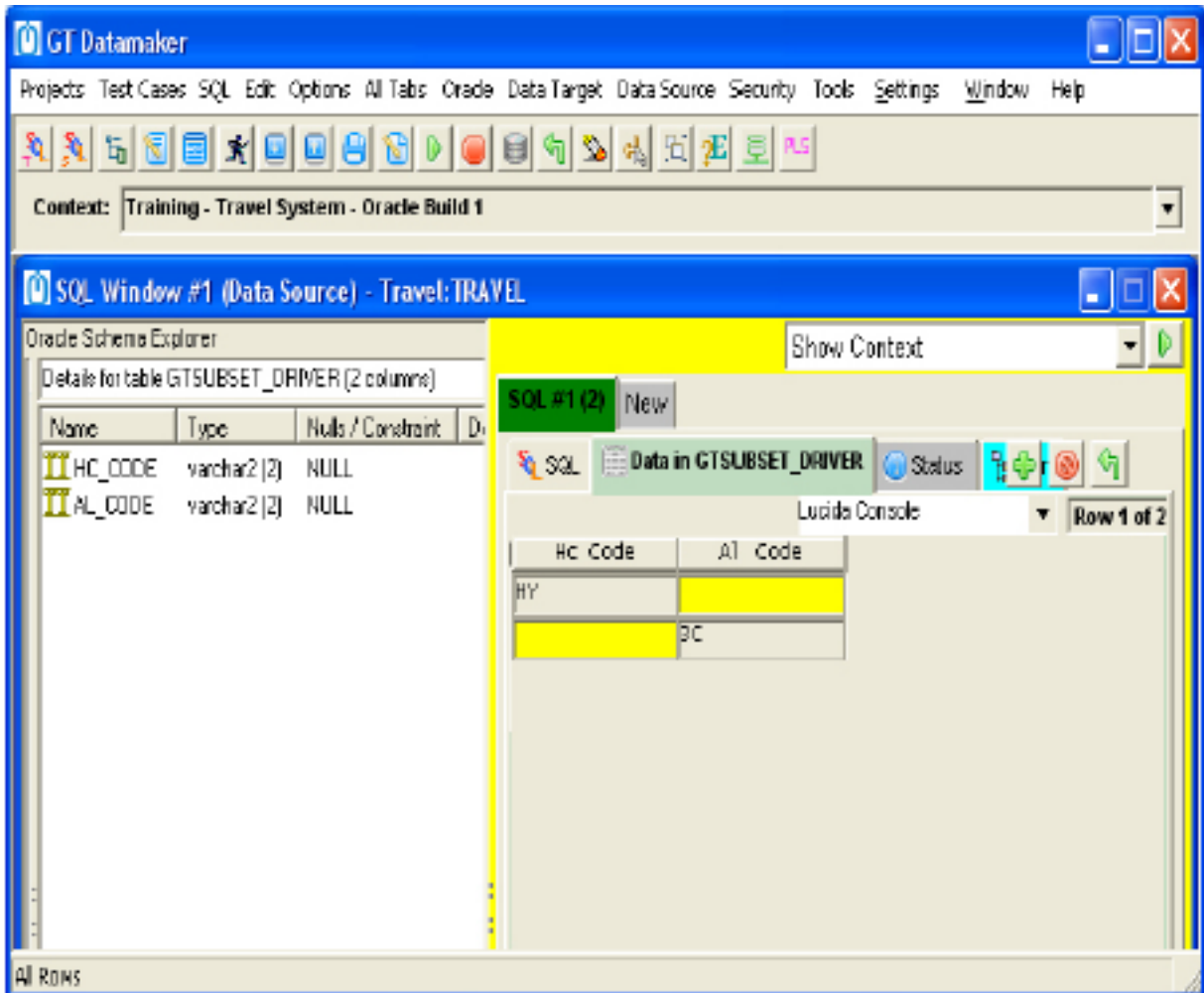


Figure 14 – The driving table defines which specific hotel chains and specific airlines are to be extracted

Users will enter specific values into the driver table. These are then used by travelling across the database model to explode the table and find other driving keys. In effect, you are finding chains of data and then controlling the depth of the data discovery.

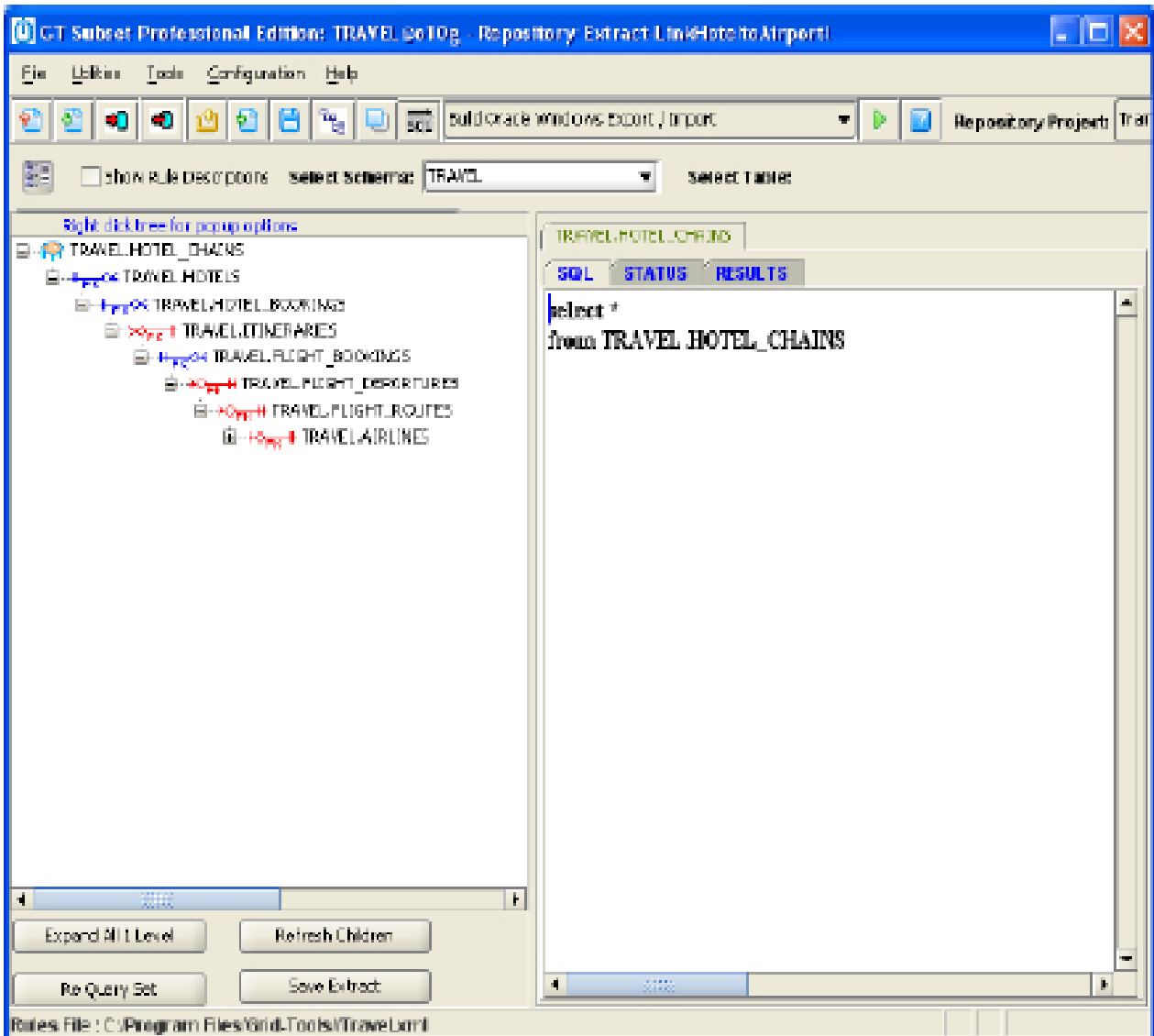


Figure 15 – The path between Hotel_Chains and Airlines is defined based on the data model

Based on the data path, you can work out which airlines have bookings associated with specific hotel chains.

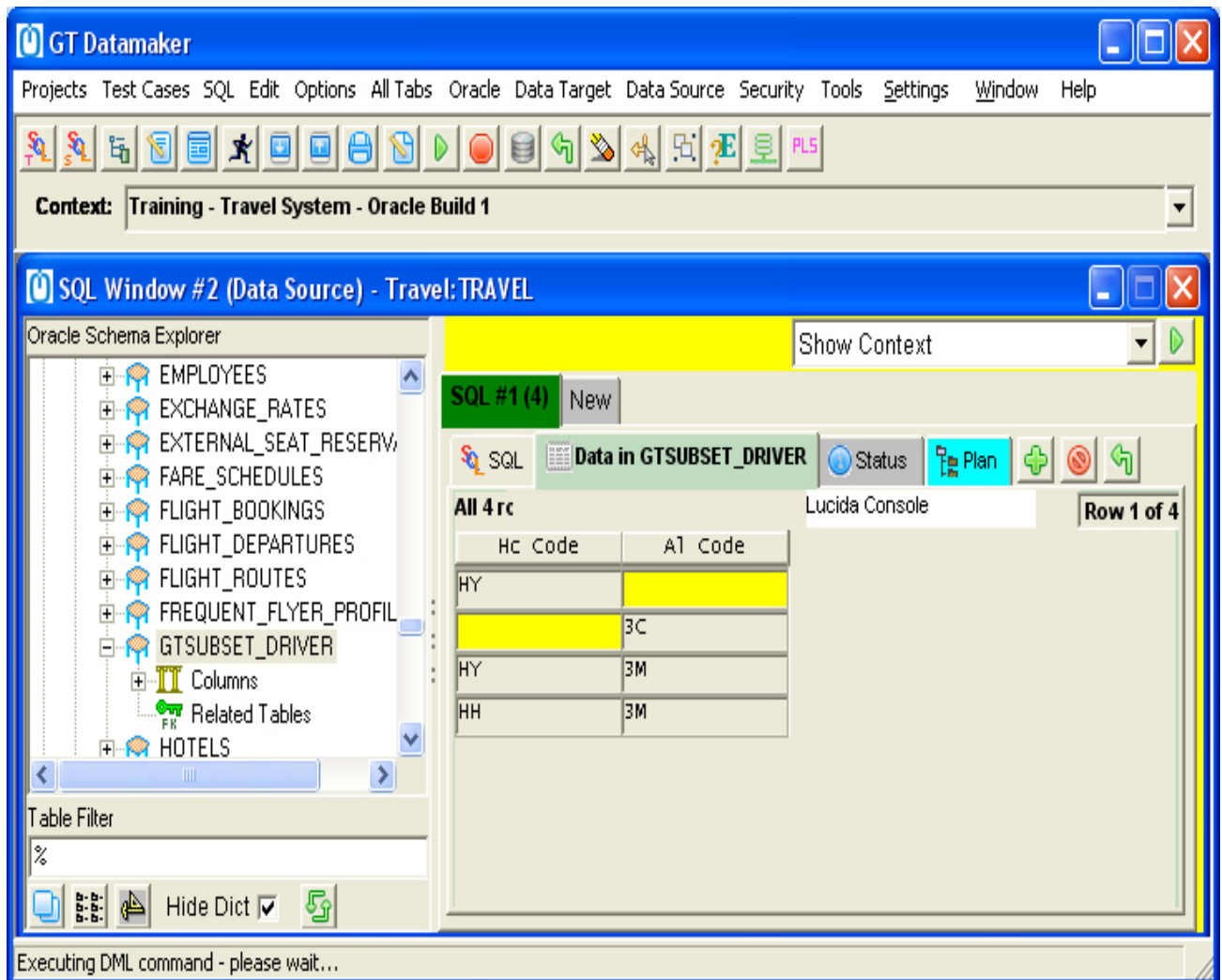


Figure 17 – Driving data after two data explosions

This technique of using a driving table to manage the major selection criteria is a very practical method of dealing with complex cross application data intersections. Users can easily see the rows of data that will be extracted and edit this table prior to the actual extraction and population of the subset database. Users quickly get used to populating this table, running the data explosion, verifying that the keys are correct and making any small amendments. Like most problems, if you break it down into smaller logical chunks it is easier to understand and implement.

Interfacing with other applications

In the real world it is rare that your database application is stand alone; most applications nowadays have quite complex interrelations with upstream and downstream data feeds.

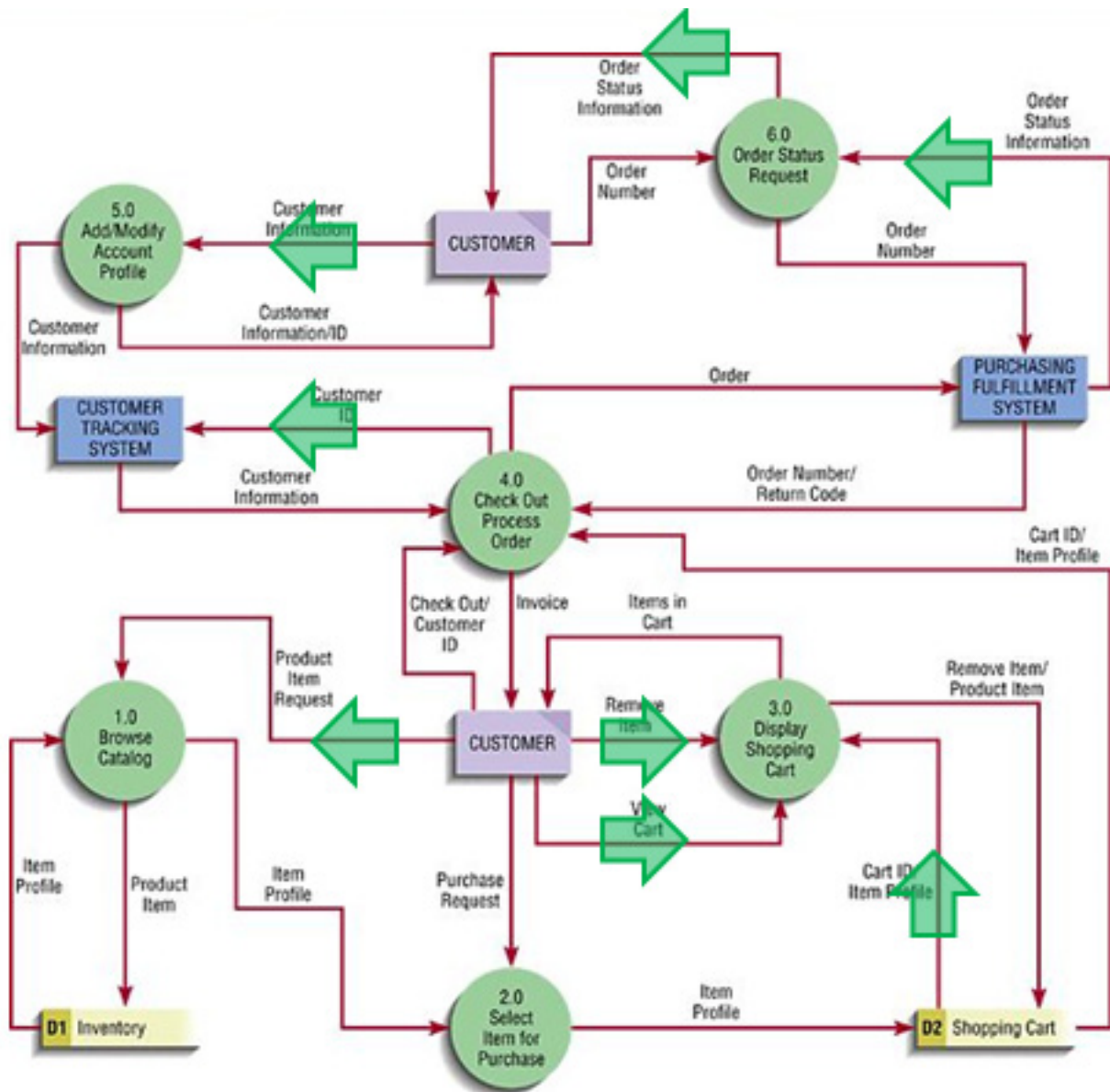


Figure 18 – A typical application data flow diagram

As part of your testing strategy you will need to identify:

- Other databases to be subsetted

- Upstream data file input, for example, interface files containing Customer updates
- Cross Database lookups, for example, Pricing lookups and Product availability
- Downstream files produced by your application
- Other applications that rely on your database. Each one of these cross application interfaces will have to be managed carefully. Some techniques that have proven very effective are:
 - When you subset multiple related databases of different types, for example Oracle and DB2, using a pre populate step to build a common driving table with primary keys from both databases is very effective. The driving table data explosion can be run repeatedly on both databases to build a common list. For example, build a list of Customers from the Customer system and then build a list of matching Orders from the Order system
 - Make sure any processes which accept flat file input do not rely on matching records, so for example, if you choose to use a copy of the production flat file containing customer updates, the file processing program must be able to gracefully reject and report on missing customers in your test database
 - If your application has to make call outs to external services, for example SOA requests using a WSDL, consider using a mock service to return consistent values every time. Often sites struggle to validate if a test has run successfully as there too many moving parts.

In reality, you may have to be quite innovative in solving some of the cross application interface problems.

What next?

Once you've taken a more detailed look at your specific databases feel free to get in touch and ask any specific questions I have not covered in this paper. Happy subsetting!

About Grid-Tools

Grid-Tools is a UK company that has become the shining light for testing quality and efficiency in system development programs and other non-production environments. Using a combination of its Datamaker software product and in-house data creation and management expertise, Grid-Tools helps companies plan and execute testing and QA processes and data needs within projects by being adopted as part of the testing strategy and infrastructure. Grid-Tools achieves this through the effective provision of test case analysis and data using a range of algorithms, techniques and

know-how to deliver testing strategies and data that are fit for purpose in terms of timeliness, coverage, volume, consistency and logical coherence. Partners and end-user organizations alike have benefited from working with Grid-Tools by being able to plan testing strategies within system development programs that are effective and flexible and that deliver programs on time, on budget and with fewer defects - all as a result of the improved quality of testing processes, integration and the timely availability of high quality test data. Grid-Tools is also able to work with existing data, using its expertise and data management and creation software tools to provide datasets optimized for coverage, shape, size and regulatory compliance (HIPAA, PCI DSS and GLBA etc) for use in system testing and other non- or pre-production environments.

Grid-Tools is privately funded and is based in the UK with local sales and support staff based in the US and India. The company has over 100 customers spread across the banking, insurance, healthcare and government industries in Europe and the US. Grid-Tools works very closely with a small number of partners to ensure that customers receive the best quality and practices possible in delivering test strategies, data and management to their development and other non-production environments.

About Huw Price

With over a 30 year career, Huw Price has been the lead technical architect for several US and European software companies. Specializing in test automation tools, he has launched numerous innovative products which have re-cast the testing model used in the software industry. Huw has provided high-level architecture design support to multinational banks, major utility vendors and health care providers.

A strong believer in balancing pragmatism with a visionary approach, he has been able to rapidly bring new products to market while maintaining strong quality control. Huw's newest venture, Grid-Tools, has quickly redefined how large organizations need to approach their test data management strategy.

Grid Tools

11 Oasis Business Park
Eynsham
Oxfordshire
OX29 4TP

UK: +44 01865 884 600

US: +1 866 563 3120

E: info@grid-tools.com

www.grid-tools.com

Find us on Facebook

Follow us on Twitter

Join the Datamaker circle on LinkedIn

Subscribe to our blog

