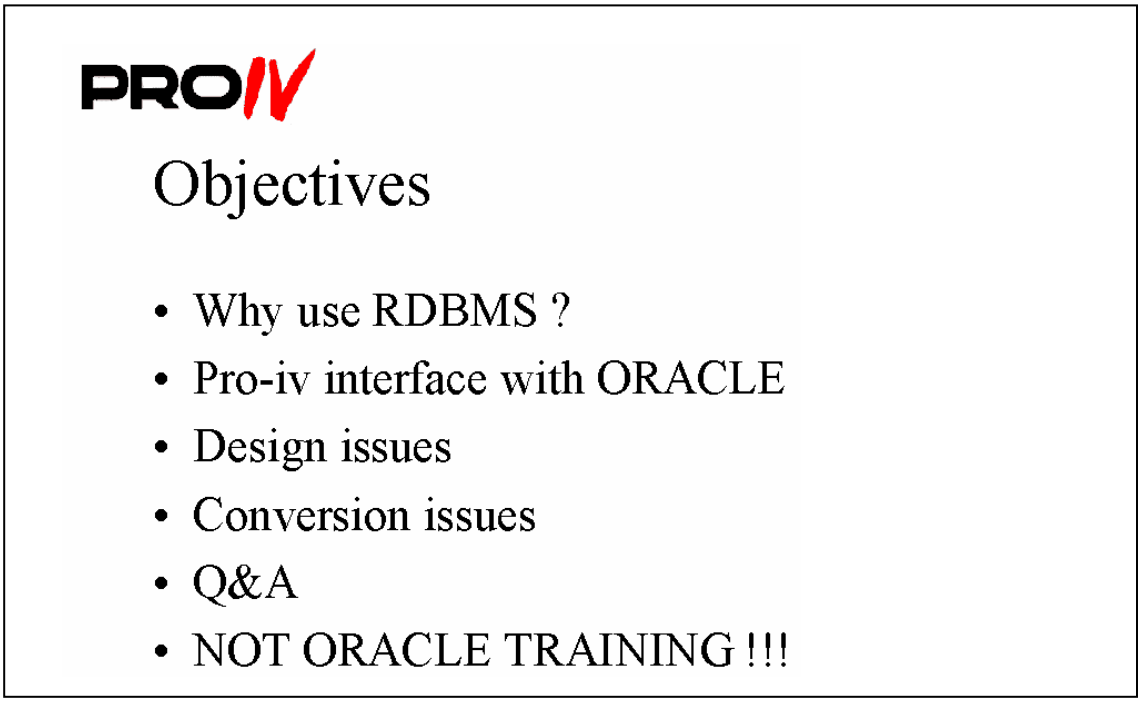
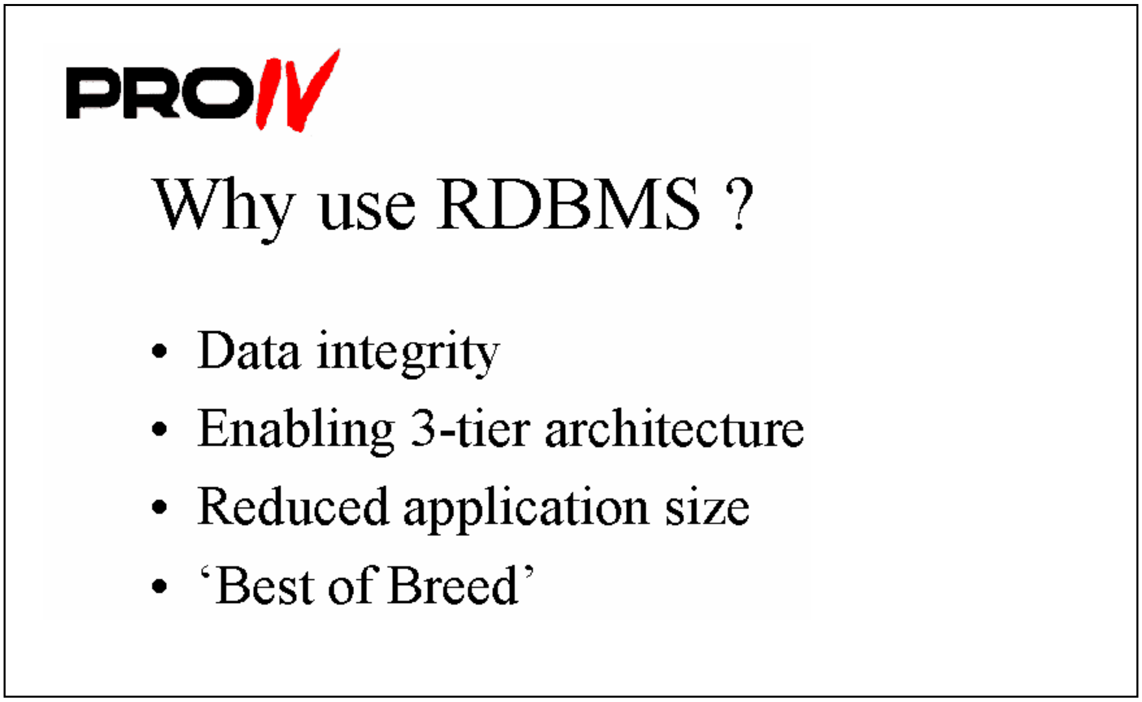
Relational Database Overview



PROIV

Objectives

- Why use RDBMS ?
- Pro-iv interface with ORACLE
- Design issues
- Conversion issues
- Q&A
- NOT ORACLE TRAINING !!!



Why use RDBMS ?

- Data integrity
- Enabling 3-tier architecture
- Reduced application size
- 'Best of Breed'

When moving an application to an RDBMS file system, you are likely to experience some of the following issues:

- Increase in system requirements. The database management system will inevitably require more memory, disk space and processor clock cycles.

- Application conversion. You will need to make some changes made to your application. Although many of these tasks can be automated, a full system test with performance tests will be required.

- Locking problems. The transaction/locking structure in RDBMS is different to that of an ISAM system, it is likely that some of your current functions will have concurrency problems.

- Training. To get the most from your system, your designers and programmers will need some SQL and database administration skills.

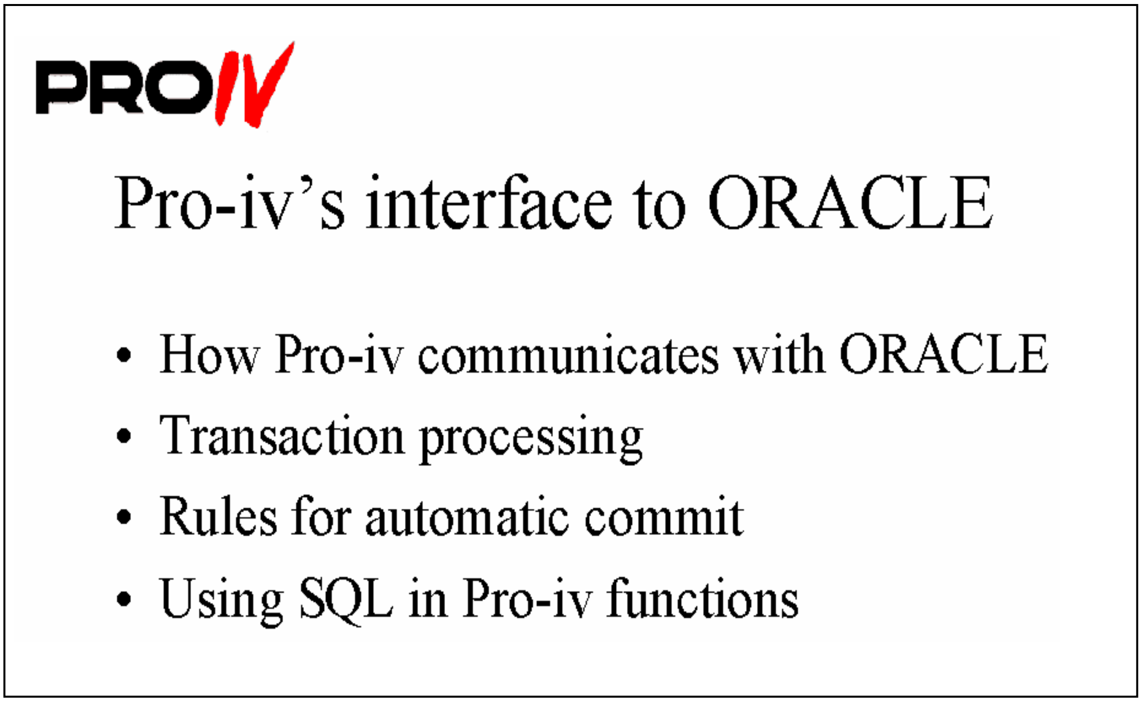
So, considering these issues, why would you move to a relational database?

- **Data Integrity**. Using a relational database, you can define a unit of work as a transaction. If this transaction does not complete, all updates belonging to that transaction will be 'rolled back'. With ISAM storage systems, you inevitably write code that processes workfiles to ensure that updates only occur at the end of a transaction. This code will no longer be necessary.

- **Enabling ‘3 tier’ architecture**. The 3 ‘tiers’ are Business rules, Presentation and Database. The marketplace is moving towards the 3-tier model, where any tier can easily be changed without affecting another. Pro-iv is your business rules tier. For the presentation tier, you can already choose between green screen, windows GUI and web browser. After a move to RDBMS, you will be able to easily move your application between databases further expanding your potential client base.

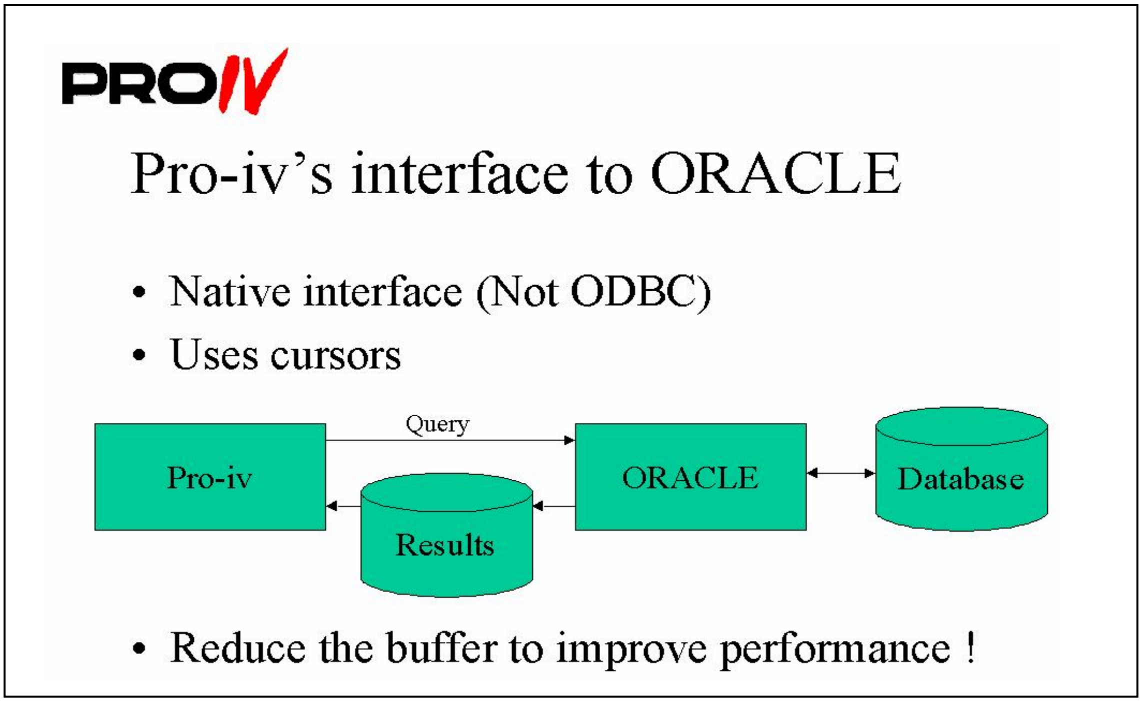
- **Reducing application size and complexity**. When you use RDBMS, your application should actually reduce in size and become more manageable. Functionality that processed workfiles and index files can be removed.

- **Best of breed**. You are already using Pro-iv, which gives your clients the ability run on their favorite platform and to be able to choose the client interface. Using a relational database gives you access to many tools that link directly to your database. Take reporting tools. Pro-iv reports are very basic. To supply tailorable management reports really requires a business intelligence tool. If you are using a popular database, it is very easy to add business intelligence capabilities to your product offerings. As well as being a sales opportunity, it is a good chance to pick up more back-end consultancy work.



Pro-iv’s interface to ORACLE

- How Pro-iv communicates with ORACLE
- Transaction processing
- Rules for automatic commit
- Using SQL in Pro-iv functions



Pro-iv's interface to ORACLE

- Native interface (Not ODBC)
- Uses cursors

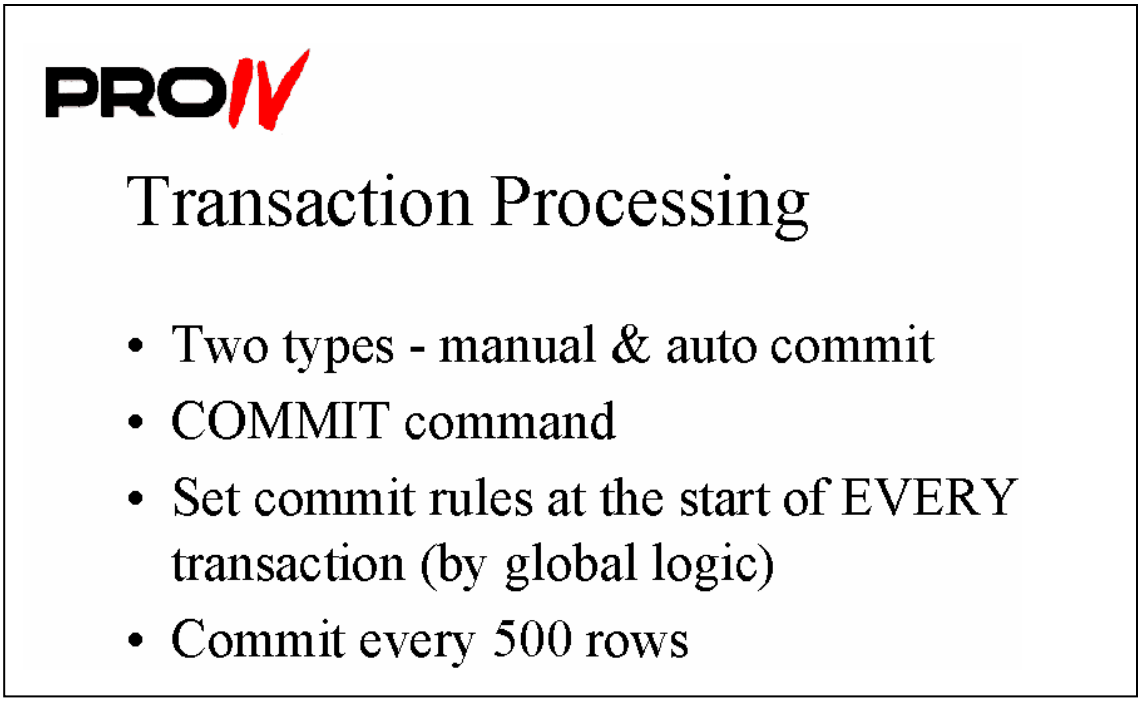
Query
Pro-iv
ORACLE
Database
Results

- Reduce the buffer to improve performance !

- **Native Interface**. When Pro-iv interfaces to a database, it does so directly, ODBC is not used! Performance of the interface is not a problem.

- **Uses cursors**. The cursor is an area of memory in which an SQL statement and its execution parameters are stored. Pro-iv uses the programmatic interfaces to control the cursors. This gives Pro-iv control over the phases of the SQL statement execution.

- **Reduce the buffer to improve performance!** If you look at the above diagram, you can see how SQL commands are handled using the Pro-iv/ORACLE interface. An SQL command is issued which is handled by ORACLE. ORACLE will interrogate the database and pass the results back to Pro-iv. These results are then processed by the standard Pro-iv timing cycle.
  Let's consider a paging update. An SQL or SEL- command would be issued. This would then be processed by ORACLE and the results passed back to Pro-iv. All of this happens just after execution of the default logic. These results would then be processed by sort select logic and after that, each row would be read. In sort select and after read no error; rows can still be de-selected. Your goal when embedding SQL into Pro-iv functions is to reduce the size of the results passed back from ORACLE to Pro-iv. There is no point in passing a row to Pro-iv to merely delete it! (More on this later).



PROIV

Transaction Processing

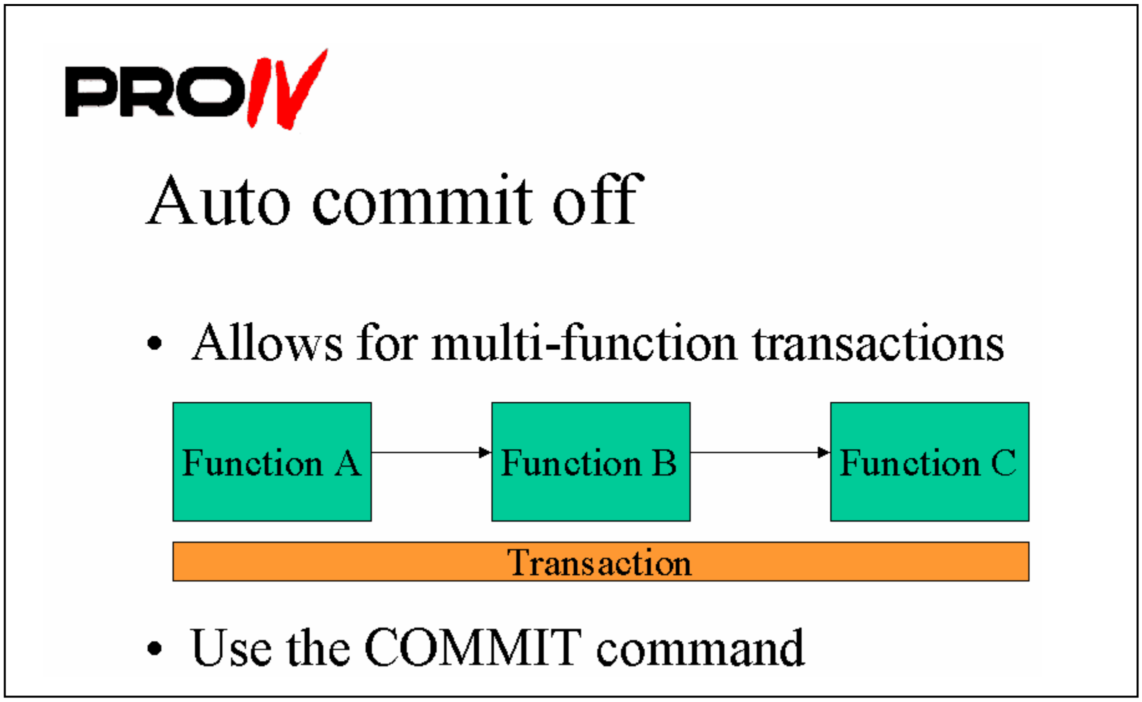
- Two types - manual & auto commit
- COMMIT command
- Set commit rules at the start of EVERY transaction (by global logic)
- Commit every 500 rows

- **Two types – manual & auto commit**. There are 2 ways that transactions can be defined within Pro-iv. You can use manual commit and automatic commit. Manual commit leaves the definition of the transaction boundaries entirely up to the programmer. Automatic commit uses a set of pre-defined rules to decide where the transaction boundary will be.

- **COMMIT command** The COMMIT command is used to commit the current transaction within a function. The COMMIT command can be used in both manual and auto commit modes.

- **Set commit rules at the start of EVERY transaction**. It is not mandatory to set the type of commit at the start of every function. It is, however very good practice to do so. Some organizations have a global logic for auto commit and a global logic for manual commit. One of these global logics should appear at the start of every transaction. If you do not set the type of transaction, then Pro-iv will keep the last setting used. This will cause application execution problems.

- **Commit every 500 rows**. There's a lot of processing involved when ORACLE performs a commit. You can issue a commit after each row is updated but this would be extremely slow. On the other hand, if you do not commit often enough, you could cause a rollback (depending on the size of your transaction log files). As a general recommendation, a commit should be issued after 500 rows have been changed. In any case, the commit should only be issued at a sensible point in the application. Another important issue is that row locks are not released until a commit is issued.



Auto commit off

- Allows for multi-function transactions

Function A → Function B → Function C

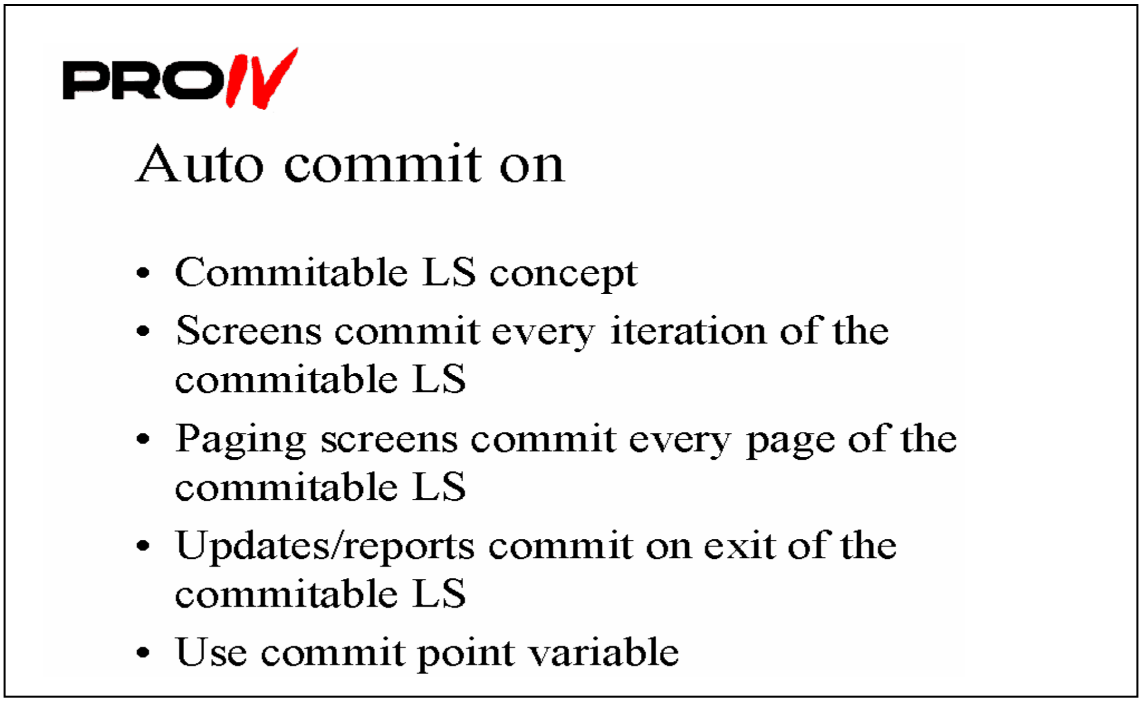
Transaction

- Use the COMMIT command

Let’s take a look at how you use transaction processing in Pro-iv, firstly manual commit. For manual commit, the auto commit facility in Pro-iv must be turned OFF.

- **Allows for multi-function transactions**. When you use automatic commit, Pro-iv assumes that your transaction is enclosed within one function. This is not always the case. In the above example, the transaction is spread across 3 functions. This allows you to write a system with small modular functions, and still benefit from transaction processing.

- **Use the COMMIT command.** When you use manual commit, it is vital to use the COMMIT command. In the above example the commit command would be issued at the end of Function C. If you do not issue a commit, then the rows updates may get committed in another function. All row locks would be held until that commit took place. If a commit is not issued and the user logs off Pro-iv, then ALL outstanding updates would be rolled back.
  The commit command should never be put between the read and write cycles. If this is done, a ‘Fetch out of sequence’ error will occur as Pro-iv tries to write back to a cursor that no longer exists (commit clears the cursors). A commit can be done in after write no error of the last table in update mode in a characteristic. It can also be done in exit logic of the characteristic or function. Be careful when you put the commit command anywhere else!



Auto commit on

- Commitable LS concept
- Screens commit every iteration of the commitable LS
- Paging screens commit every page of the commitable LS
- Updates/reports commit on exit of the commitable LS
- Use commit point variable

When using automatic commit, it is vital that you understand where Pro-iv will issue a commit.

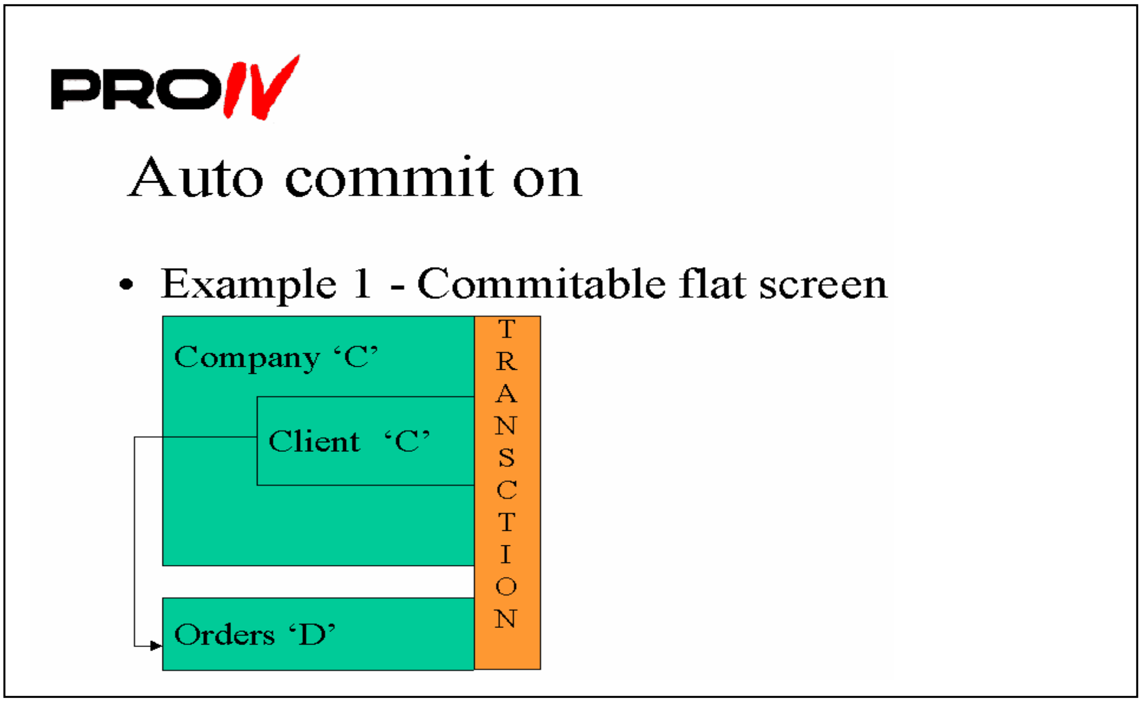
- **Commitable LS concept**. With the relational database interface to Pro-iv, we have introduced the concept of the ‘commitable LS’. The commitable LS is the characteristic in which a commit will be issued. There can be only ONE commitable LS at any time. The commitable LS is the highest characteristic with a table being modified. There can be more than one commitable LS in a function but only one can be commitable at any time. The issuing of a commit is dependent on the type of characteristic.

- **Screens commit every iteration of the commitable LS**. If a ‘flat’ screen is currently the commitable LS, then a commit will be issued after each write in that characteristic. All updates in characteristics nested or called from this characteristic will be included within the transaction.

- **Paging screens commit every page of the commitable LS**. With a paging screen, a commit is performed for every page of data. The commit will occur when the user hits F3 or page up/down.

- **Updates/reports commit on exit of the commitable LS**. A report, update or Lsupdate characteristic will commit on completion of the characteristic

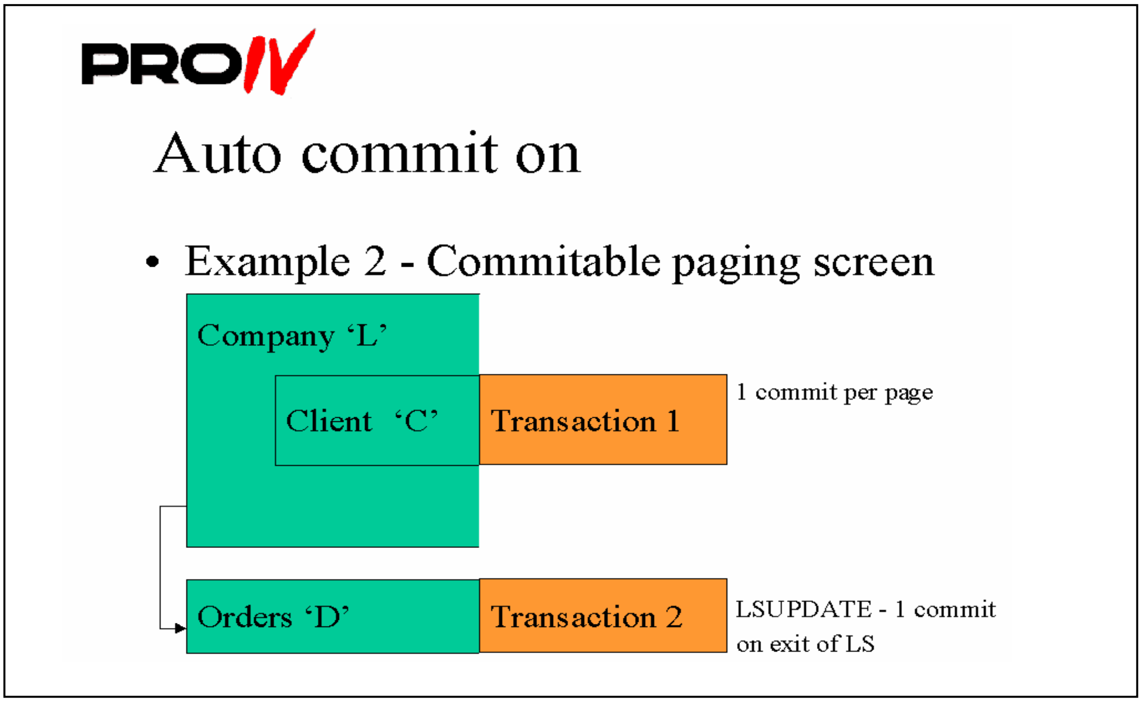
- **Use commit point variable**. When using automatic commits, you can still use the commit command. This is especially important in large updates. You can use Pro-iv’s ‘commit point’ facility. This allows you to enter a numeric variable name. Pro-iv will then automatically maintain a counter in that field. This field can then be interrogated within the characteristic and used to determine whether a commit is required.



Auto commit on

- Example 1 - Commitable flat screen

- **Example 1 – Commitable flat screen**. In this case, we can see that the highest level characteristic with a table being updated is characteristic one. All other characteristics are nested/called within that characteristic. All updates will be commited after each iteration of characteristic one.



Auto commit on

- Example 2 - Commitable paging screen

Company ‘L’

Client ‘C’

Transaction 1

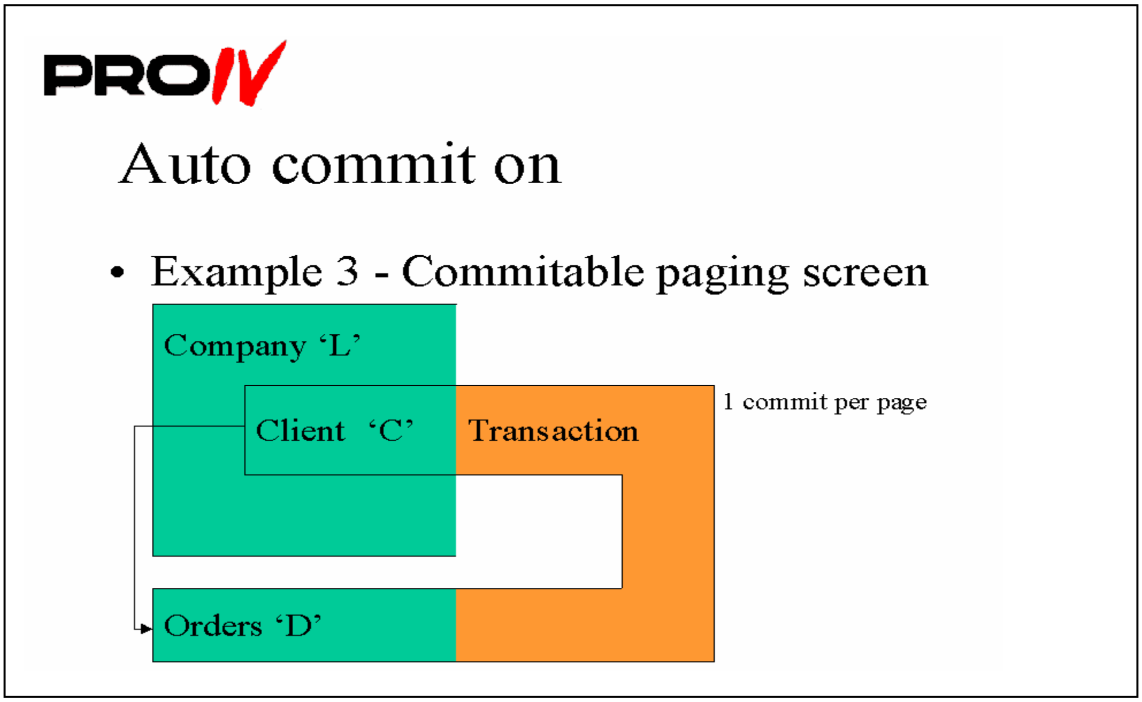
1 commit per page

Orders ‘D’

Transaction 2

LSUPDATE - 1 commit on exit of LS

- **Example 2 – Commitable paging screen**. This is very similar to example 1; in this case, the COMPANY table is now in lookup mode. This means that the highest characteristic with a table being updated is characteristic 2. In this characteristic, a commit will be issued for each page of data and on exit of the characteristic.
  When characteristic 2 has completed, there is NO commitable LS open.
  When characteristic 3 is called, this will become the commitable LS. A commit will be issued on exit of this characteristic.



PRO/V

Auto commit on

- Example 3 - Commitable paging screen

Company ‘L’

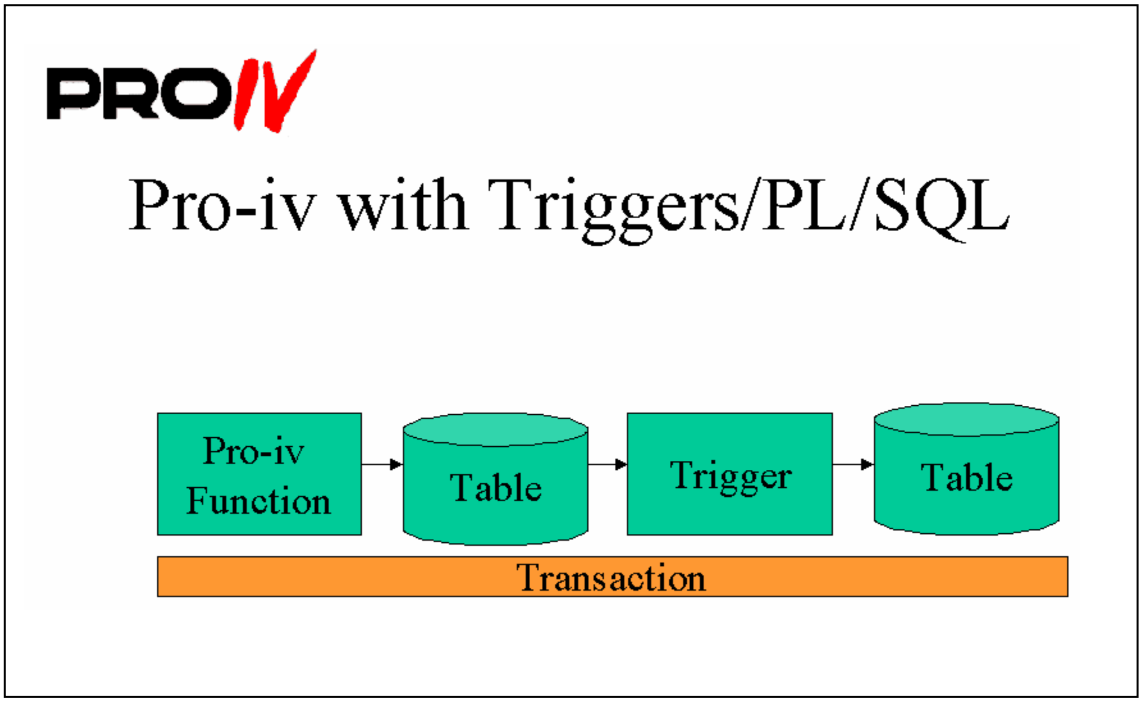
Client ‘C’

Transaction

1 commit per page

Orders ‘D’

- **Example 3 – Commitable paging screen**. In this example, we have changed the position of the call to characteristic 3. The updates in characteristic 3 will now be a part of the transaction in commitable LS number 2.

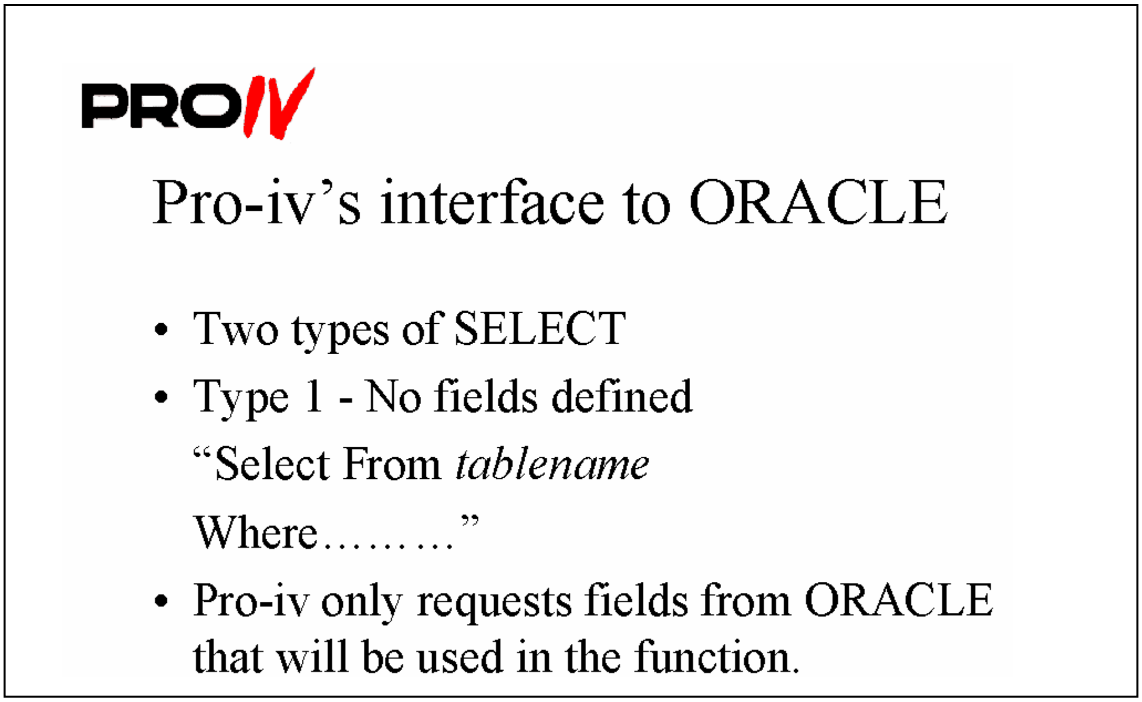


Pro-iv with Triggers/PL/SQL

Pro-iv Function → Table → Trigger → Table

Transaction

- **Pro-iv with Triggers/PL/SQL**. When you write your system, you may use triggers and procedural SQL (PL/SQL). A trigger is attached to a table and is executed on a pre-defined action such as a row insert. When Pro-iv inserts a row to such a table, the trigger procedure will execute and further table updates may occur. When this happens, the updates from the trigger belong to the same transaction as the Pro-iv function. This also applies when you call a PL/SQL procedure; all of the updates contained within that procedure would be a part of the current Pro-iv transaction.



PRO/IV

Pro-iv's interface to ORACLE

- Two types of SELECT
- Type 1 - No fields defined
  "Select From *tablename*
  Where………"
- Pro-iv only requests fields from ORACLE that will be used in the function.

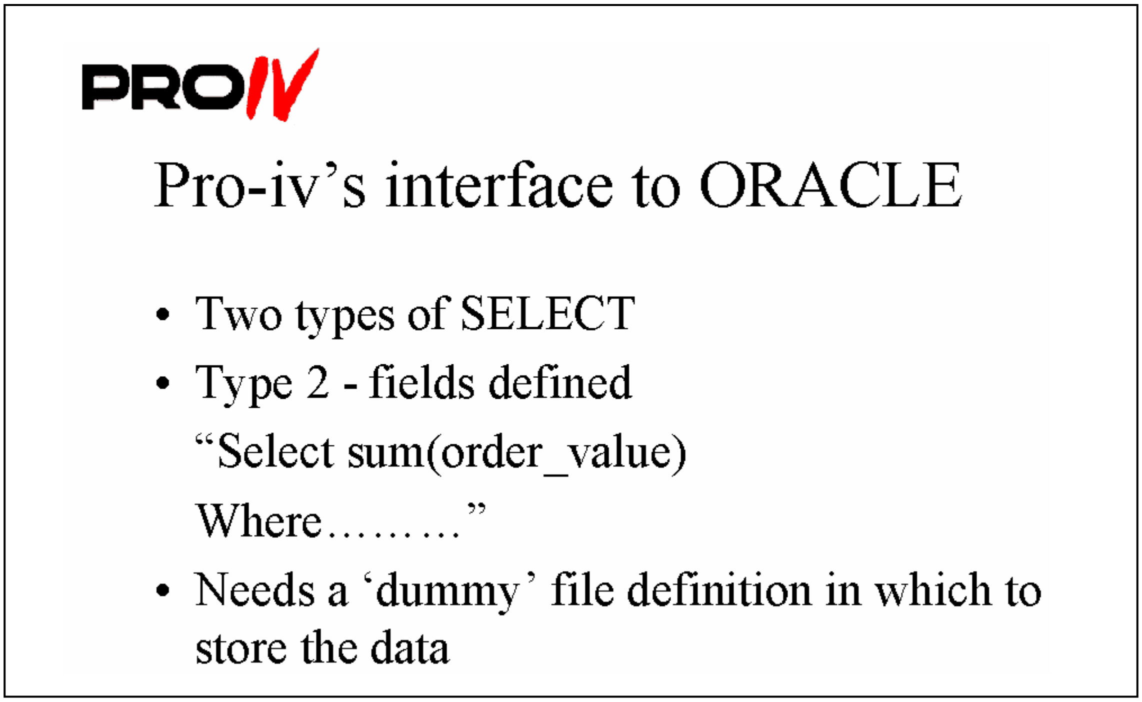
Pro-iv accesses ORACLE using SQL commands. SQL is ALWAYS used by Pro-iv. When you use SEL-ONLY in a function, Pro-iv turns that into an SQL command. There is therefore no difference in operation when using embedded SQL to SEL- commands. Being able to embed SQL commands gives you much more control over row selection but care must be taken as SQL commands can cause lots of extra work for ORACLE. This is where good SQL training will pay dividends.

- **Two types of SELECT**. There are two types of select statement that can be used in Pro-iv. The two types of select have different purposes. The first type is used to select rows from a table, returning single or multiple rows. The second type is used to collect summary data from a single/multiple tables. This will usually return one row. Any select statement is valid, including nested queries.

- **Type 1 – No fields defined**. A type 1 SQL statement is used to select rows for processing by the Pro-iv function. When you embed SQL in a Pro-iv function, you do not specify the column names. Your SQL statement will look something like this:

  ```
  SELECT FROM CLIENT
  WHERE COMPANY_ID = :$COMP_ID
  ```
  NB. ***When using this type of SELECT statement, the table name is the name of the Pro-iv file definition.***

- **Pro-iv only requests fields from ORACLE that will be used in the function**. The reason that it is not necessary to define column names is that Pro-iv automatically detects which columns are required in the function. Only these columns will be passed back to Pro-iv.



PROIV

Pro-iv's interface to ORACLE

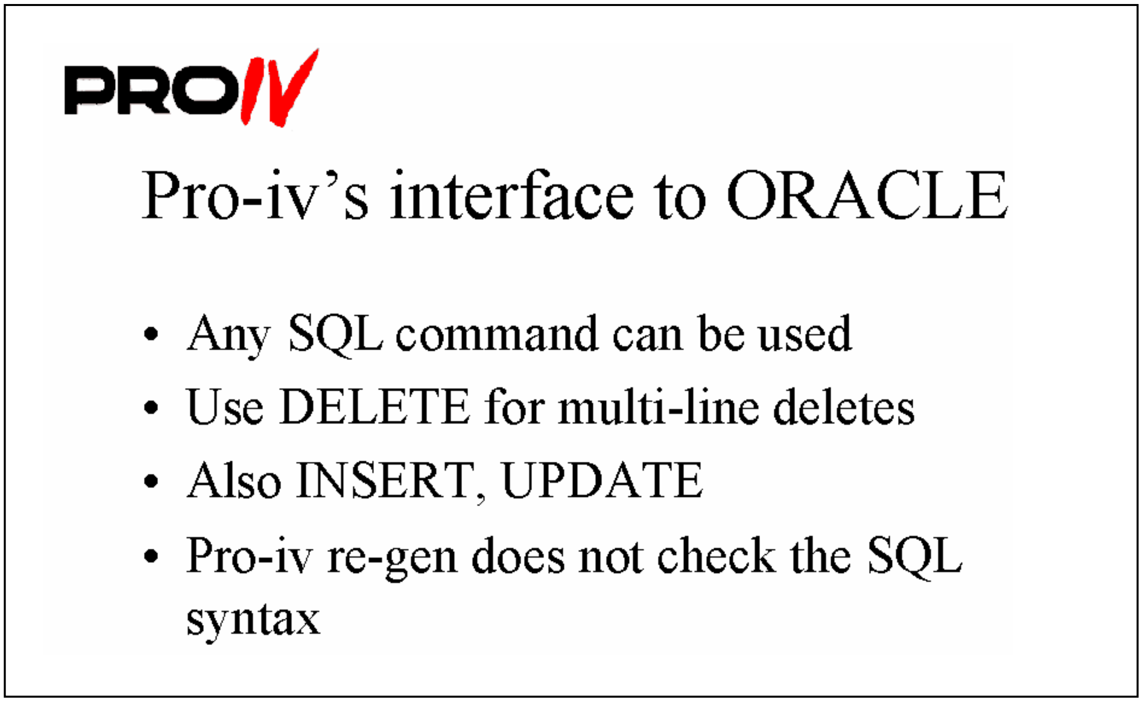
- Two types of SELECT
- Type 2 - fields defined
  "Select sum(order_value)
  Where………"
- Needs a 'dummy' file definition in which to store the data

- **Type 2 – fields defined**. This type of select statement is typically used to return summary data. An example would be to return the total value of a customers' orders. This would have the following syntax:

  ```
  SELECT SUM(ORDER_VALUE) FROM ORD_LINE
  WHERE ORDER_NUM IN (SELECT ORDER_NUM FROM ORDERS
                                         WHERE CUSTOMER_ID = :$CUSTOMER)
  ```

  This SQL finds all of the orders for the selected customer id and then reads all of the order lines, calculating the total order value for the customer. The reason that we use this technique is that only one column is passed from ORACLE to Pro-iv, drastically reducing the buffering and improving the performance.

- **Needs a 'dummy' file definition in which to store the data**. All select statements are coded into before read or default logic. You need to put a 'dummy' file definition into the function to store the result. Using type 2 select statements, a file definition that matches the returning data needs to be used.
  In this case, a 'dummy' file definition with one numeric field is required. After the read, the numeric field will contain the total order value for the customer.



Pro-iv’s interface to ORACLE

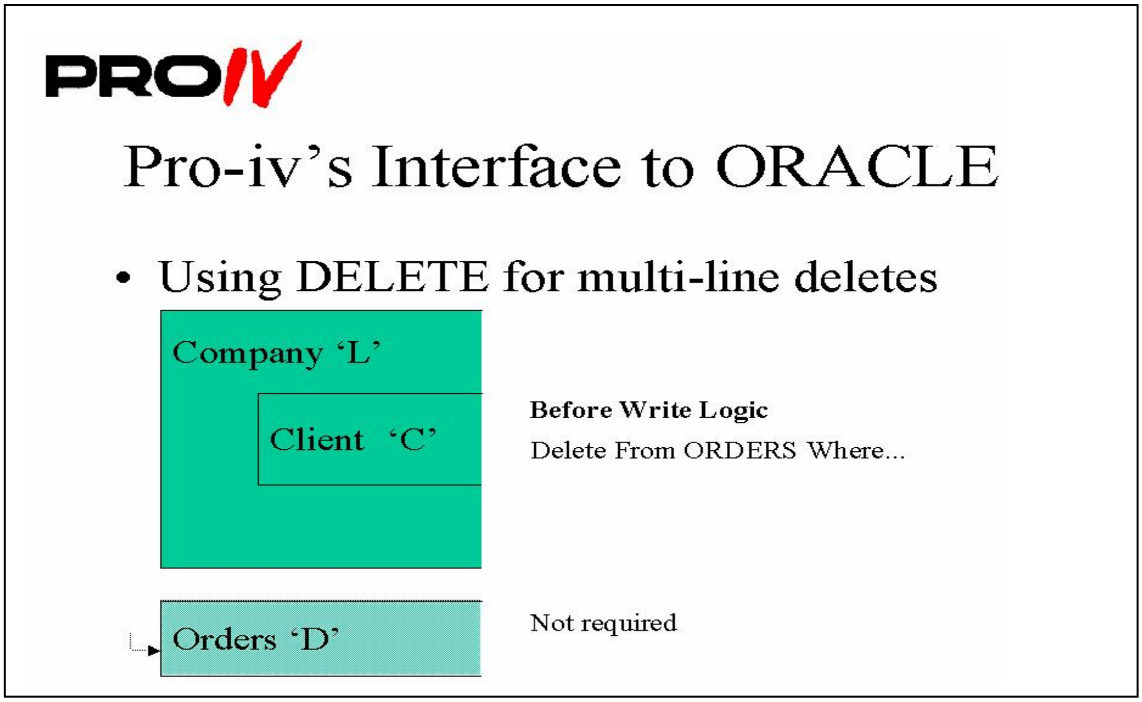
- Any SQL command can be used
- Use DELETE for multi-line deletes
- Also INSERT, UPDATE
- Pro-iv re-gen does not check the SQL syntax

- **Any SQL command can be used**. So far, we have looked at the use of SELECT statements in before read logic and default logic. We can put other SQL statements into our functions in ANY logic. This includes the use of DDL,DML and calls to PL/SQL. We must remember to put the SQL in a logical place in terms of the transaction.

- **Use DELETE for multi-line deletes**. An example of using SQL statements to improve the performance of a function is the DELETE command. When you use Pro-iv delete mode, the row is requested from ORACLE, passed back to Pro-iv (for possible deselection) and then deleted. This is a lot of unnecessary work if you merely want to delete the row. In fact, the row does not need to be passed back to Pro-iv at all. See the next page for an example.

- **Also UPDATE, INSERT**. As above, you should also give consideration to the UPDATE and INSERT commands in SQL.

- **Pro-iv re-gen does not check the SQL syntax**. Pro-iv does not check your SQL. You should ensure that every embedded SQL is tested before making a function ‘live’



PRO/IV

Pro-iv's Interface to ORACLE

- Using DELETE for multi-line deletes

Before Write Logic

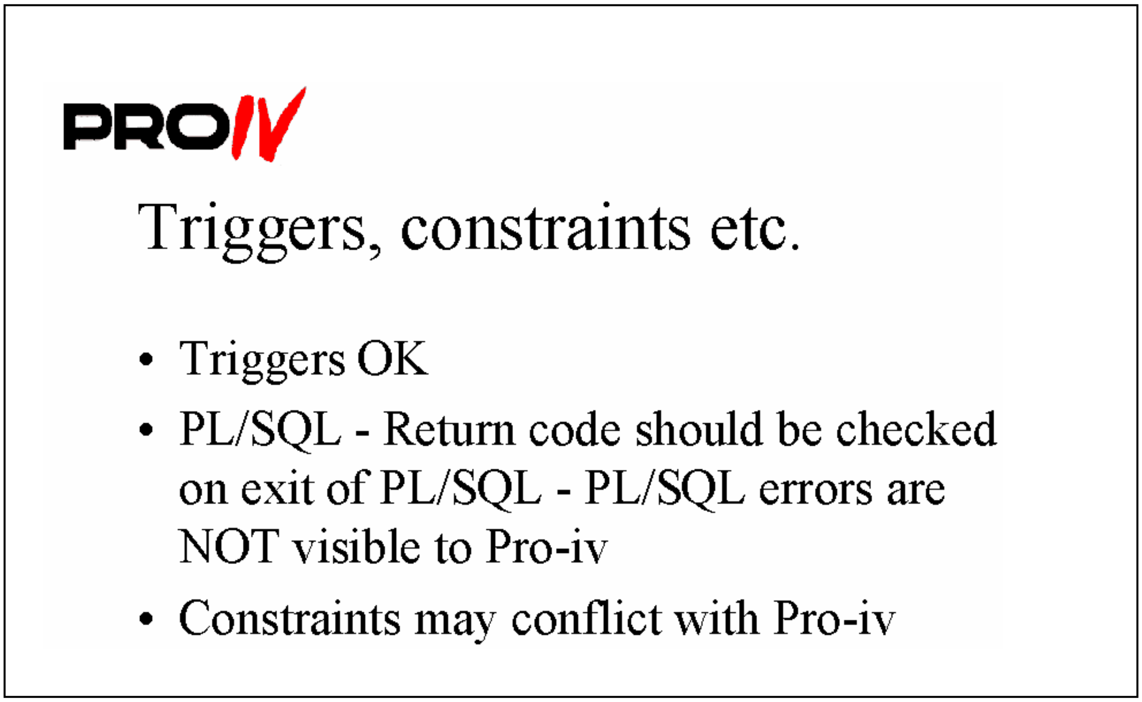
Delete From ORDERS Where...

Not required

- **Using DELETE for multi-line deletes**. Let's consider an update that deletes all orders for a client. We have an ORDER table and an ORD_LINE table for order lines.
  With ISAM, we would have a many time characteristic to delete each of the ORDER and ORD_LINE tables. With SQL we can put the following logic in after write no error of the client table:

  ```
  DELETE FROM ORD_LINE
  WHERE ORDER_NUM IN (SELECT ORDER_NUM FROM ORDERS
      WHERE CLIENT_ID = :$CLIENT)
  DELETE FROM ORDER
  WHERE CLIENT_ID = :$CLIENT
  ```

  In this case, there is no buffering of data and no pre-delete reads. Performance will be far superior. The only drawback is that you will not see this type of delete in a file/function cross-reference.



Triggers, constraints etc.

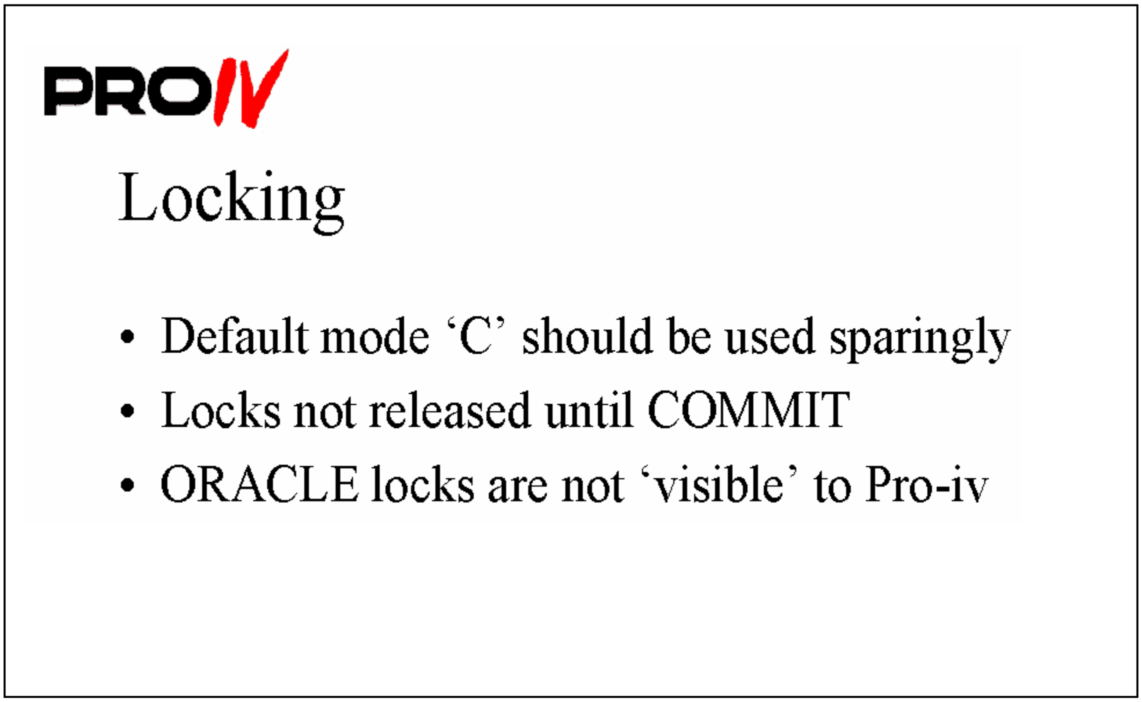
- Triggers OK
- PL/SQL - Return code should be checked on exit of PL/SQL - PL/SQL errors are NOT visible to Pro-iv
- Constraints may conflict with Pro-iv

Some database objects work well with Pro-iv, others do not.

- **Triggers**. Triggers are very useful objects. You could add a trigger to your system that deleted all order lines when the order header is deleted. This fits in well with the Pro-iv transaction method.

- **PL/SQL**. PL/SQL is very efficient for mass updates. Care should be taken when calling PL/SQL. If a PL/SQL fails, then Pro-iv is not aware of the failure. There are system variables that contain the last ORACLE error code. These should be cleared before calling the PL/SQL and interrogated once the PL/SQL has completed. If there is an error code in these system variables, then you know that the PL/SQL failed.

- **Constraints**. Constraints put restrictions on data. A constraint may exist on a column to say that it can have a value from 1 to 100. If a Pro-iv function allows a value outside of that range, an error and rollback will occur upon commit. It is sensible to keep all of the business logic within Pro-iv. Constraints are not recommended.



Locking

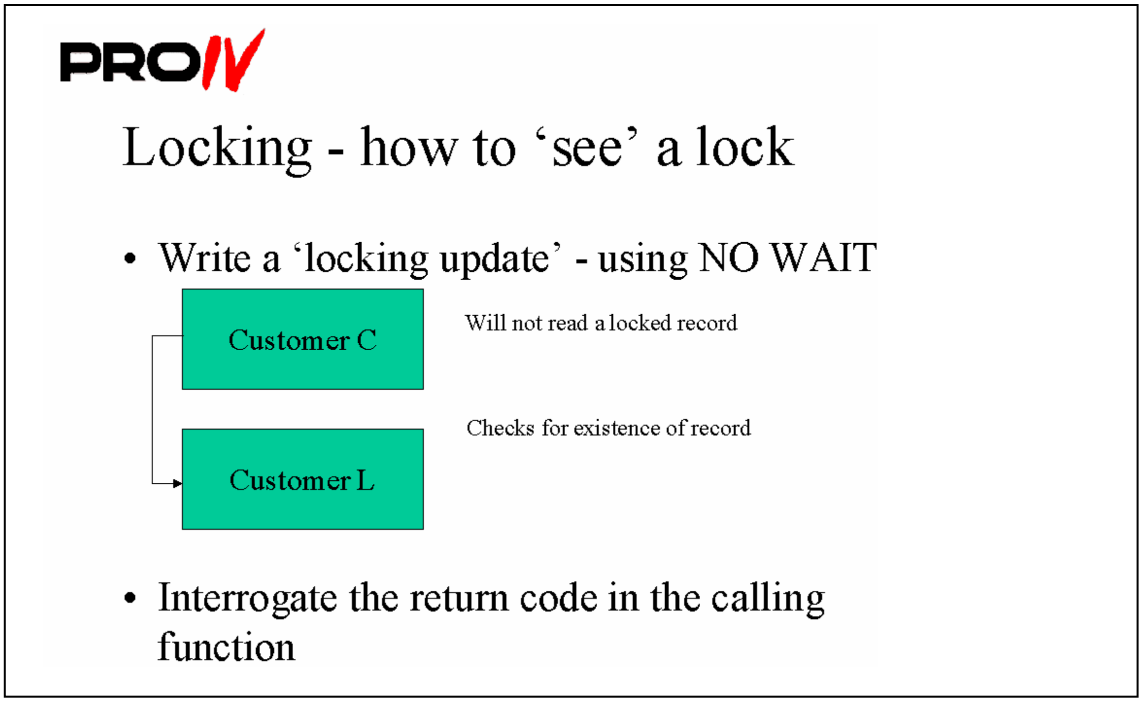
- Default mode ‘C’ should be used sparingly
- Locks not released until COMMIT
- ORACLE locks are not ‘visible’ to Pro-iv

There is a difference in the way that locking is handled when you use Pro-iv with RDBMS. This is one of the major issues that you have to deal with when converting. Understanding this issue is fundamental in a successful implementation of RDBMS.

- **Default mode ‘C’ should be used sparingly**. In a Pro-isam paging screen, with a default mode of change, your rows are displayed without being locked. Only when you actually enter a row is it locked. This is different when you used RDBMS. In this case, all rows that you see have been read with update intent. This means that they are all locked. If two users try to access the same set of data in change mode, one users’ screen will lock until the second user has finished. This typically occurs in simple table maintenance functions. Maybe you have a country code maintenance paging screen. If it has default mode ‘C’, then it will be a single-user screen with RDBMS !!

- **Locks not released until COMMIT**. With RDBMS, locks are held until a commit is performed, another reason to be careful about transaction boundaries.

- **ORACLE locks are not ‘visible’ to Pro-iv**. Pro-iv requests information from ORACLE. If the information is not returned, a ‘waiting’ message is displayed. Pro-iv does not see ORACLE locks and therefore locking logic in Pro-iv will not be executed. You can write a small global update in Pro-iv to see if the row is locked. See the next page for details on how to write the update.



PROIV

Locking - how to ‘see’ a lock

- Write a ‘locking update’ - using NO WAIT

Customer C

Will not read a locked record

Customer L

Checks for existence of record

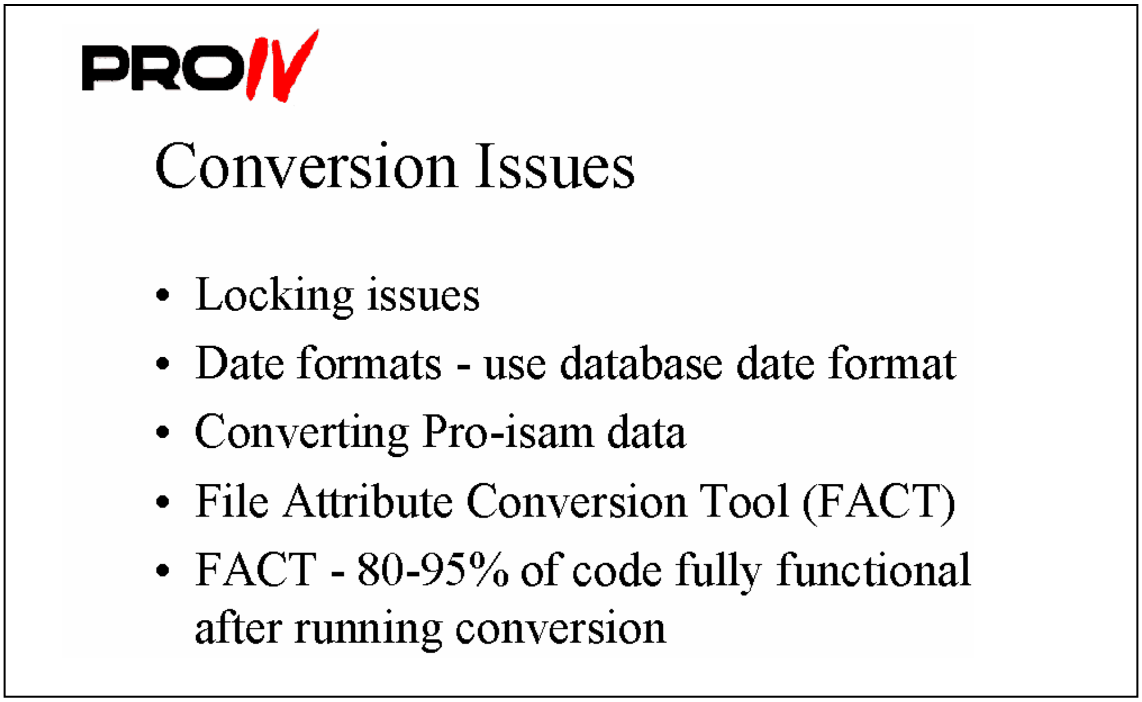
- Interrogate the return code in the calling function

- **Write a ‘locking update’ – using NO WAIT**. When we encounter a row lock, the system will wait until the lock is released. This is not always acceptable. The above example shows a global update that checks for a lock on the CUSTOMER table. The first characteristic reads the table in change mode. The NO WAIT clause is specified in the SQL in default logic. This will read the row in change mode if it is available and cause a read error if it is locked. If we get a read error, we know that the row is either locked or missing. We read the row in lookup mode to check that it exists.

- **Interrogate the return code in the calling function**. By the time the global update is complete, we know the status of the row. We can then pass a return code back to the calling function for example**:**
  1) Row successfully locked.
  2) Row already locked
  3) Row not found

  This return code should be interrogated in the calling function and appropriate action taken.

- Be careful calling this update when auto commit is ON. If the transaction has not already started in the calling function, this update will be the commitable LS. In that case, a commit will be issued on exit of this update, clearing the lock.



PROIV

Conversion Issues

- Locking issues
- Date formats - use database date format
- Converting Pro-isam data
- File Attribute Conversion Tool (FACT)
- FACT - 80-95% of code fully functional after running conversion

By now, you should be getting some idea of the issues that you will encounter when converting your system. Here's a summary of the main issues:

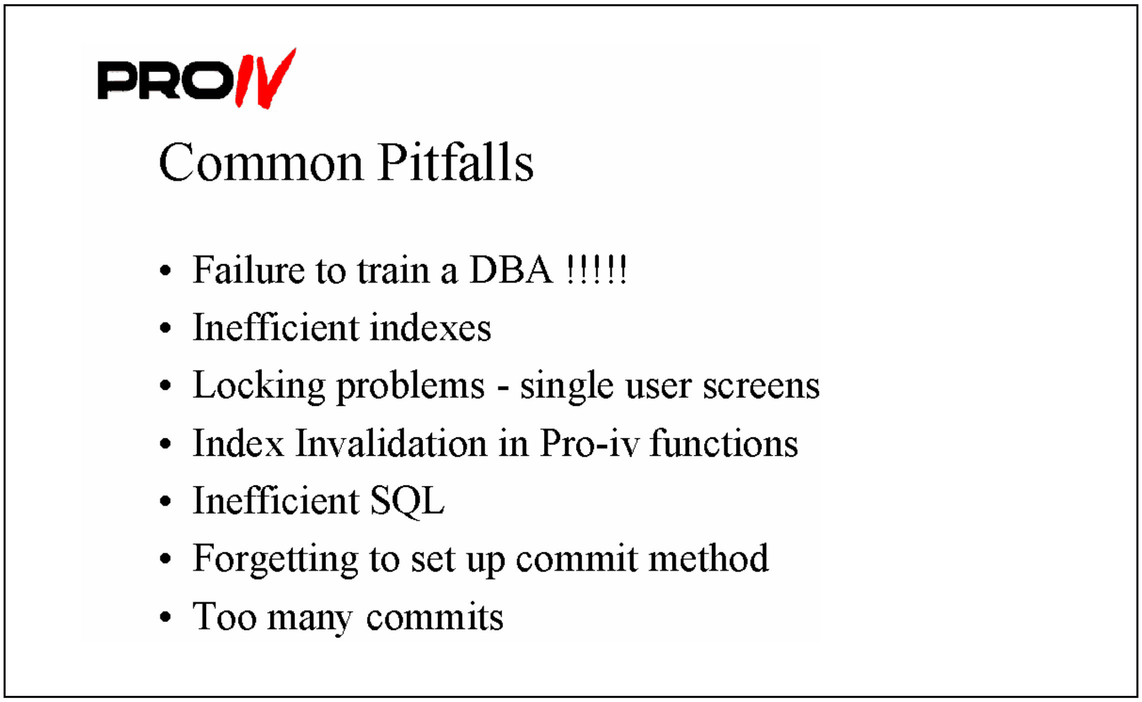
- **Locking Issues**. As we have seen, converting your functions as they are may leave you with some locking issues.

- **Date formats**. ORACLE has its own date format. Specify this format in your external type and Pro-iv will automatically convert the columns back to Pro-iv format when reading/writing the table. It is important to do this as other tools and PL/SQL will not understand Pro-iv dates.

- **Data conversion**. If you are converting data from Pro-isam, then you will need to check the data format in the Pro-isam files before converting. For instance, your Pro-iv file may have a number with a format of 5.2. Pro-iv uses variable length fields and it is possible that you have data outside the boundaries of the field definitions. For example you may actually have 309.827 in your 5.2 field. It is necessary to fix these problems before converting your data as your database cannot handle these inaccuracies.

- **File Attribute Conversion Tool (FACT)**. This is a set of tools produced by Pro-iv that assist in the conversion of your system. They handle conversion between Pro-isam, Btreive, C-Isam, Ingres, Oracle and Dec RMS. The tools automate file definition conversion, logical file update removal and the creation of file conversion programs.

- **FACT – 80-95% of code fully functional after running conversion**. After running the FACT tools, your system will be operational. At this point, you should then look at fine tuning the locking, transaction processing and performance aspects of your system.



PROIV

Common Pitfalls

- Failure to train a DBA !!!!!
- Inefficient indexes
- Locking problems - single user screens
- Index Invalidation in Pro-iv functions
- Inefficient SQL
- Forgetting to set up commit method
- Too many commits

- **Failure to train a DBA**. You don't need a DBA with Pro-isam. Put your files in a directory and that's your administration complete! With ORACLE you need a trained DBA. Tablespaces, clusters, extents, rollback segments, redo log files, control files, database buffer cache, etc - these settings all have an impact on your system. Your DBA will also be very useful at implementation time.

- **Inefficient indexes**. This also comes under the DBA role. Indexes are different when using RDBMS. For example, an index on a column with a low number of values e.g. male and female will actually slow data access down.

- **Locking problems**.

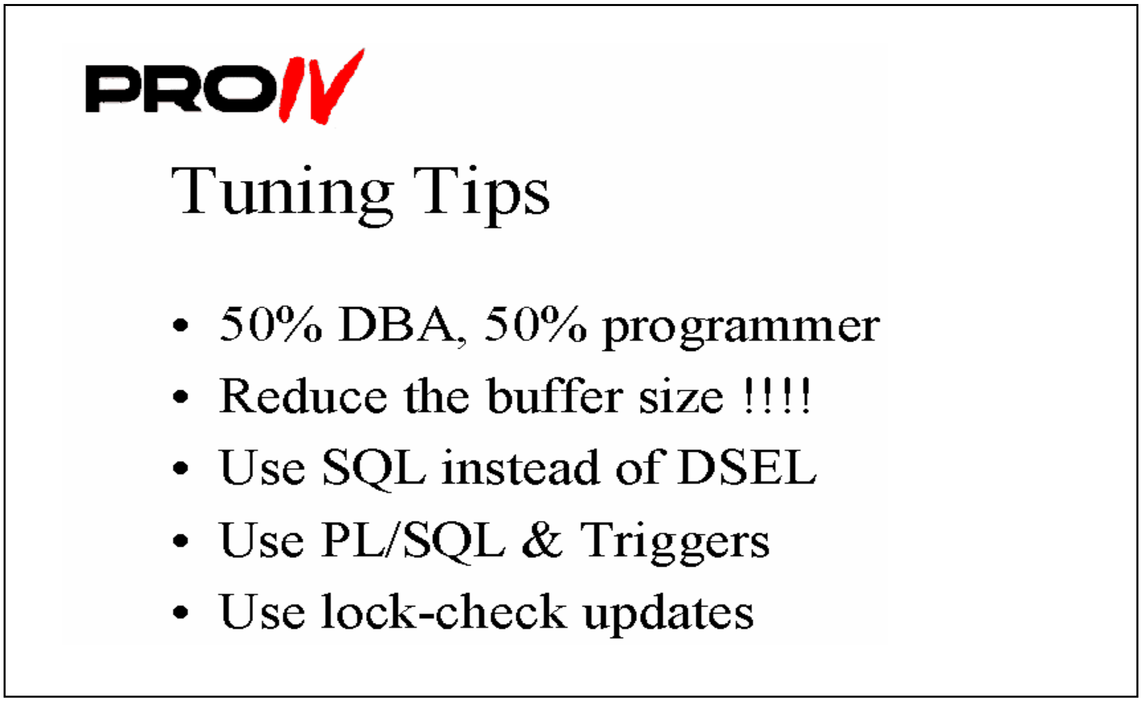
- **Index Invalidation in Pro-iv functions**. Your programmers now have the power of SQL at their fingertips. They must be careful which SQL commands they use. SELECT clauses such as LIKE, SOUNDEX, greater than, less than and substring will 'invalidate' indexes when used on key columns. For instance:

  SELECT FROM CUSTOMER
  WHERE CUSTOMER_NAME LIKE 'PRO%'

  An index on CUSTOMER_NAME will not be used in this select because it has been invalidated with the LIKE clause. Instead the whole customer table will be read and the relevant rows returned. This is very bad for performance.

- **Forgetting to set up the commit method**. It is a very common mistake to forget to set up the commit method in a function. The function will operate differently when auto commit is on or off. This is why we recommend using global logics on entry to each transaction to set the commit method.

- **Too many commits**. Another common problem is to go ‘over the top’ in terms of commits. Only issue a commit when you need one, too many commits will affect the performance of the entire system.



Tuning Tips

- 50% DBA, 50% programmer
- Reduce the buffer size !!!!
- Use SQL instead of DSEL
- Use PL/SQL & Triggers
- Use lock-check updates

Finally, let's take a look at some of the more pertinent tuning tips:

- **50% DBA, 50% programmer**. So far we have discussed how to write code using the Pro-iv/RDBMS interface. The way that your tables are organized has a profound impact on performance, as does the physical organization of the database on the disks. Do not underestimate the power of your DBA in the tuning process

- **Reduce the buffer size**. In every characteristic, in every program, you should look for ways to reduce the size of the data sent from ORACLE to Pro-iv. You should ensure that the programmers fully understand the impact of passing unnecessary data to Pro-iv.

- **Use SQL instead of DSEL**. To reduce the buffer size, only the required rows should be returned from ORACLE. You do this by coding good SQL. Don't worry about the size of the SQL statements, as long as you don't invalidate the index, it is always faster to de-select rows using SQL than passing them back to Pro-iv for de-selection. Sorting of data should be done in SQL using the ORDER BY clause. Use sort select sparingly

- **Use PL/SQL and Triggers**. For mass updates, PL/SQL is faster than Pro-iv. Triggers are also useful ways of performing common updates.

- **Use lock-check updates**. These updates, when called from before read logic, enable you to give control back to the user when you hit a row lock.

PROIV

Conclusions

- Not ‘rocket science’
- Get trained
- Use FACT
- Call Pro-iv for assistance !!