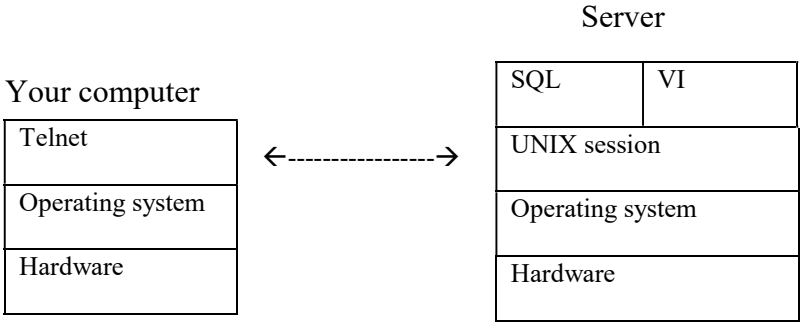
Relationship between SQL, vi, UNIX, sessions…
There is some confusion in the words being used to describe where they are when connected to a UNIX box.

Relation ship between your PC and the applications running on Telnet sessions



On your PC you invoke the Telnet communication application which dials you into a UNIX box. . Here you can run a UNIX application be it SQL or VI.

Do I need to talk about OSI seven layer model so that people use the terms application, session, … in the right context

Application
Presentation
Session
Transport
Network
Link
Physical

By using the SQL edit command you are suspending SQL and passing control to another application, in this case VI. After quitting from VI SQL resumes from where it left off.

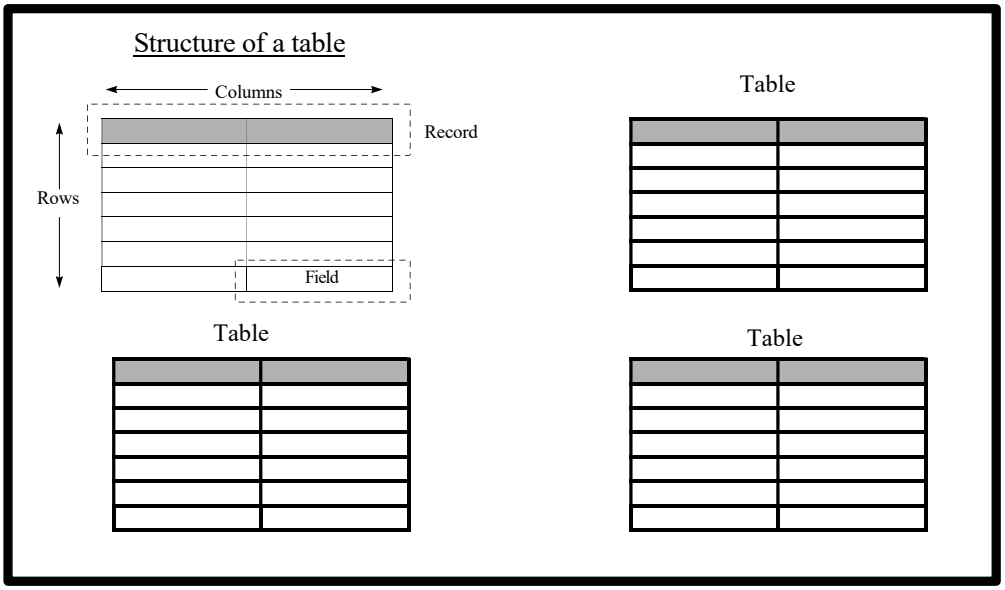
Introduction

All of the work we do here involves interaction with databases. Whether it be through a pretty windows interface with lots of boxes & buttons (aegis), the not so pretty CDMS or the really user hostile text based screens (UNIX & vi) that UAT have to work with.

Define a database. - *Get audience to define elements of a database*

Starting off simply, the outer structure is the database or schema, this is the container that everything gets stored in.
Within the schema like items of information are grouped into tables. Say Tperson, which describes a person and details, items like name & date of birth. The tables are built up from columns that hold different data items; There would be one column for name, another for date of birth. Each person would form a row of data across the columns. In old parlance a row in a table would comprise a record and the intersection of a particular column and row would be called a field. This is a lot simpler as a diagram.

Structure of a data base



There are a large number of database applications on the market. Each one has a different look and uses its own language to manulipate data.
The need arose for a common language that could be used to talk to any database no matter its vendor. Structured Query Language was created to meet this need. Most database applications support SQL in addition to their native language. There is a standard SQL as defined by ANSI (American National Standards Institute). This is a core of functions that exist in every vendor's version. As long as the core functions are supported the package can be sold as ANSI compliment.

Throughout these notes the examples will use a table called Tperson.
Which without going into detail looks like

Name	DOB	Dept	Pin	Card_no	Salary

The following topics will be covered:

- Pulling data from a table
- Getting information out of a table
- Putting data into a table
- Changing data that already exists in a table
- Removing records from a table
- Creating a new table
- removing tables

Once these have been completed you should have a better understanding of the simpler aspects of SQL.

It should be come apparent that SQL follows every day language. Take note of the words you use to describe a problem and what you want out of it and then look those words up.
There is a help function in Oracle. Simply type in help followed by the word your interested in.

The select command.

Just as there is more to life than death & taxes, there is more to SQL than the select command but you can go along way without worrying about anything else.

The select command displays the contents of a table, row by row. At its simplest the statement is

select *column_name* from *table_name;*

This will display the contents of column column_name for all of the rows in table table_name. Most of the work done by UAT centres around using select.

Should you want more than one column shown type in the list of columns separating each one by a comma

select *column_name, another_column_name* from *table_name;*

When every column is wanted, typing in the entire column name can become really tedious, in recognition of this there is a short way to say every column. It is such a popular items of shorthand that most operating systems also use it when they want every available item to be picked. This useful abbreviation is the star, shift eight or the multiplication key in the numeric keypad. When the command interpreter sees a * it is read as everything, so every column will be shown. e.g.

SQL> select * from Tperson.

This will display all of the data (rows & column content) in the Tperson table.
Not all of the data is going to be wanted on every occasion. SQL can be told to display the data that matches certain selection criteria. Say pull out all the data for anyone called smith.
SQL> select name, DOB, salary from Tperson where name='Smith';

This will display all the information for anyone called Smith.

Wild cards in strings

Smith, a name for the masses. Let us consider the upper echelons of the social classes such as Smyth, Smithe and Smyhte. In describing these names we could say that they are like Smith. We can tell SQL that what we are looking for is like something. SQL needs to be told what elements of a name need to be matched exactly and what elements are allowed to deviate wildly. Just as in the select clause the star symbol was used to mean anything, there are wildcards that can be used to mean part of a name or word. The term wild card here is used exactly as it is in card games - a card that can pretend to be another, like twos & jokers in canasta.
To perform a search for all the esoteric spellings of Smith the following statement can be used

SQL> select distinct name from Tperson where name like 'Sm%';

You may have noticed a change at the start of the select statement (the second word). The distinct in the select clause will prevent duplicate results being displayed and the % represents anything or more precisely an undefined string of characters. So this query will display one instance of each name starting with Sm.

Say we now want all the occurrences of Smith and Smyth but no Smithe. Here the **%** wouldn't be exact enough as in this search only one character is allowed to deviate - the letter i. The wildcard for a single character is the underscore _ . This gives a statement of
SQL> select * from Tperson where name like 'Sm_th';

The **%** and _ symbols can be used in combination to select any pattern of characters.

Multiple search criteria

Having found information for the mode English name lets expand the search to include the rest of the UK. Here we have a list of names and we are looking for names that are in the list. There are two ways that people normally write lists, top down or left to right possibly using commas to separate individual items. SQL allows the use of lists.

SQL > Select * from Tperson where name in
2 ('Evans','Macdonald','Smith','Smithe','Smyth');

The items in the list need to be separated by commas and the list needs to be enclosed within brackets. The brackets can be regarded as the limits of the piece of paper you used to draft the list

Now we want to find everyone who does not have a name. This can be represented by either an entry of no characters '' or a null entry into a field. Although these two methods may look the same to a human on paper they are not always identical as far as the computer is concerned. How they are treated depends on the version and flavour of SQL being used. In the present incarnation of Oracle empty quotation marks and a null are treated as equivalent but this can not be guaranteed in future releases. To further complicate matters one null is not the same as another null.
So to be on the safe side when looking for people with no Name we have to perform two searches
SQL> select * from Tperson where name= '';
SQL> select * from Tperson where name is null;

Note that in the second line the equals sign has been replaced with the word 'is'. This is a quirk of dealing with null values. The way to remember this is to consider that null values are undefined or non-existent Saying that something equals undefined/non-existent does not sound right. Where as ' something is non-existent or something is undefined' sounds a lot better. *An undefined value is not the same as an uninitialised variable. Undefined exists and can be processed without causing a program to produce errant results. Uninitialised values can pick up the last value off the machines stack and cause all sort of interesting events to occur*

The two separate select statements can be rewritten as:
SQL> select * from Tperson where name= '' or name is null;

The joining word here is '**or**' where some of you may have expected to see the word '**and**'. To explain why this is so we must detour into the realm of truth tables. If you just want to take this on faith skip to the next page

AND

Input		Result
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

OR

Input		Result
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

The **And** table only gives a True(1) result if both of the inputs are True(1).

The **Or** table gives a True result if any of the inputs are True.

In terms of searches the **And** will tighten a search as both criteria of the search must be satisfied by any one row for that row to be selected. An **Or** will expand the scope of a search as any one row only has to satisfy one of the search criteria to be selected. If both conditions are satisfied then it still gets selected.

Returning to the SQL statement notice that each of the search conditions in the where clause are complete logical expressions within their own right. Each condition has both a left hand and a right hand term separated by an equals sign. The statement can not be written as
SQL> select * from Tperson where name = '' or is null;
Why ? Because the number of left hand terms is less than the number of right hand terms. They must balance. This is one of the things the command interpreter looks for before even attempting to run the statement.

Like a mathematical formula the search criteria of the where clause are processed from left to right subject to the rules of precedence. A slight exception to this is that any **And**s will be processed before any **Or**s. So think carefully about the order of multiple search criteria and use brackets to control the order of the processing. The use of brackets will also make reading the code easier
To illustrate the difference that evaluating **And** before **Or** consider the following
A or B and C will be processed as get the rows that satisfy the both search criteria B and search criteria C and add to these the rows that satisfy search criteria A. Where as what the writer intended was get all the rows that satisfy either (or both) of search criteria A or search criteria B and then find which of these also satisfy search criteria C.
It comes as no suprise that all the wordy stuff above can be abbreviated to nice simple formulas:
A or B and C != (A or B) and C
A or B and C = A or (B and C)

The != operator is something not covered yet so lets look at

Other comparison operators

Just being able to see if things are equal makes for cumbersome select statements. Just as we can decide if something is more or less than something else so can SQL. The table below lists all of the available selection criteria.

Operator	Meaning	Opposite
=	equal to	!= ^= <>
!= ^= <>	not equal to	=
<	less than	>=
<=	less than or equal to	>
>	greater than	<=
>=	greater than or equal to	<
between … and …	between two values	not between
in (list)	any in a list of values	not in (list)
like	match a character pattern	not like
is null	is a null value	is not null

Ordering the data

Using the 'Order by' clause the results of a query can be sorted and then displayed in either acceding or descending order. For lowest to highest the statement is
SQL> select name, DOB from Tperson where name='Smith' order by DOB;
This will display the name and DOB for people in Tperson startin with the youngest and progressing through to the oldest
To get the results in order of highest to lowest add desc to the end of the statement.
There can be more than one column specified in the order by parameter
No matter what sort order you choose any null values will be displayed first

Automatic data conversion

If a statement is entered and some of the data is in the wrong format for the field SQL will make an attempt to convert said data to the correct type. CDMS holds just about everything as a character string. When correctly typing a pin number into a statement the pin is enclosed in single speech marks. If the speech marks are not used then SQL sees a numeric input when it expects a string and helpfully converts the number into a string rather than falling over and complaining. There is a drawback with this in that although the pin had a number of leading zeros. SQL ignores the zeros since they are not significant and just converts the remaining significant digits into the string. Consequently any searches using this statement will fail.
e.g. 00012345 typed in when it should have been '00012345'
SQL sees this as 12345
SQL converts this to '12345'
'00012345' is not the same as '12345' as the three leading zeros are missing from the string and the length of the strings are different.
Murphy's laws will always apply to automatic data conversion.

Statistical functions

Having selected, compared and sorted our data it would be nice to get some information from it.

There are five stats type functions these are:

Function	Example	
avg	avg(salary)	average value of salary column
count	count(*)	number of rows selected
max	max(salary)	highest value of salary
min	min(salary)	lowest value of salary
sum	sum(salary)	total salary bill

These functions return the result based on all of the rows selected by the search criteria.

Group by

If you want the average salary for each department you could run a number of different selects; one for each department.
Instead get SQL to group together rows having the same department code and to display the average for each group

SQL> select dept, avg(salary) from Tperson group by department;

Dept	avg(Salary)
sales	22,100
admin	13,000
CEO	1,000,000.10

Having

Just as select uses the where clause to select certain rows there is a similar function to allow select of specified groups. The Having clause compares a property of the group against a value (which could be the result of a subquery).

If for the above salary example we wanted to exclude departments that consisted of only one person the statement would become

```
SQL> select dept, avg(salary) from Tperson group by dept
  2  having count(*) >=2;
```

This cunningly conceals the CEO's salary with out the code drawing attention to the fact.

The order of execution in a statement is choose rows based on the where clause, group rows together inline with the group by. Apply the having condition and then sort the groups based on the order by clause.

Group by and having are realy temperamental so I tend not to use them.

Involving more than one table

Often the information need will be split between various tables and a query needs to be performed that spans these tables. Rows in one table may be joined to rows in another table by common values in corresponding columns. This is why just about every table in CDMS contains a pin or card number.

The expression joining the tables is placed in the where clause of the select statement and basically says where thing in table A equals thing in table B.

Because a column can occur in both tables SQL needs to be told precisly which column is to be used. This is done by prefixing the table name and a dot to the column name giving *table.column* as the full name of the desired column. If you want all the columns from one table and just a few from the other the statement would look like

```
SQL> select Table_A.*, dept, salary from Table_A, Tperson
  2  where Table_A.pin = Tperson.pin;
```

This statement will select all the columns from Table_A and then dept and salary from Tperson.

Most joins between tables will use the equals condition but any relationship operator could be used e.g.

where x.sal > y.sal

which will join each row in Y to those rows in X whose value of sal is greater. Why you would want to do this is beyond me.

If the audience has grasped this quickly

When working across more than one table you may get the error message
'single row subquery returns more than one row'
The following statement, which finds card issue details for members working off of their CIDs suffers from this error.

```
SQL> select * from card_issue_histories
  1  where card_hldg_id =
  2  (select card_number_histories.card_hldg_id from
  3  card_number_histories, memberships
  4  where card_number_histories.mbrp_no=memberships.mbrp_no
  5  and cid = (select * from jasons_pins));
```

This is because each of the nested selects are actualy being used to generate a list to be passed to the preceding select statement. So the equals signs in lines 1 and 5 need to be replaced with **in** which is the comparsion operator used for processing lists.

Abbreviating table names

Although table names are often needed to avoid ambiguity in a query they can be really tedious to enter. Especially with some of the descriptive names used in CDMS

Alias

A table can be given a temporary name in the from clause and this temporary name can be used elsewhere in the query.

```
SQL> select D.* , ename, job
  2  from emp E, Tperson D
  3  where E.dept = D.dept
  4  and E.dept in ('sales','admin','CEO');
```

Notice that an alias appears to be used (in line one) before it has been defined (in line two). This is all right because SQL works through the syntax and structure of a statement before executing it. An error will be reported if you use the full table name and an alias has been defined. It is a stupid quirk so just get used to it.

Synonyms

Within the select statement the table was assigned an alias to shorten its name and save typing. The alias only exists within that statemet. It can not be used outside of the statement. To give a table a short name that can be used all the time it can be given a synonym.

SQL> create synonym *alias* for *table*

Synonyms can not be shared between users. Each user will need to create the same synonym. Public synonyms can only be created by the data base administrator and they are bound to object to people ignoring the schemas naming conventions.

Functions, system variables and chopping things up

If you know that someone is in the database but you can't find then on a Name search. Or you know that an airport is in a time zone five hours away but as to plus or minus you haven't got a clue. How do you go about selecting the information.

In both of these cases we want to force the data into a mould to circumvent discrepancies in how we want to examine it and what it really looks like.

There are a whole host of functions that allow data to be reshaped and chopped up.

If we take the name MacDonald as an example. This could appear as
Macdonald
MacDonald
Mac Donald
Ignore the problem of the space, in the third spelling, the concern here is matching the capitalisation of their name. Its' format all depends on the whim of the data entry clerk. So grab hold of the name and turn it all into capitals.

SQL> Select * from Tperson where upper(Name)='MACDONALD';

The opposite to upper is of course lower. If forcing the capitalisation of a world please set the comparison string to the correct case.

To select all the Macs from the data base we could use Mac% or just use the first three characters of the string.

SQL> select * from Name where ltrim(Name,3)='Mac';

or to be avoid any capitalisation problems

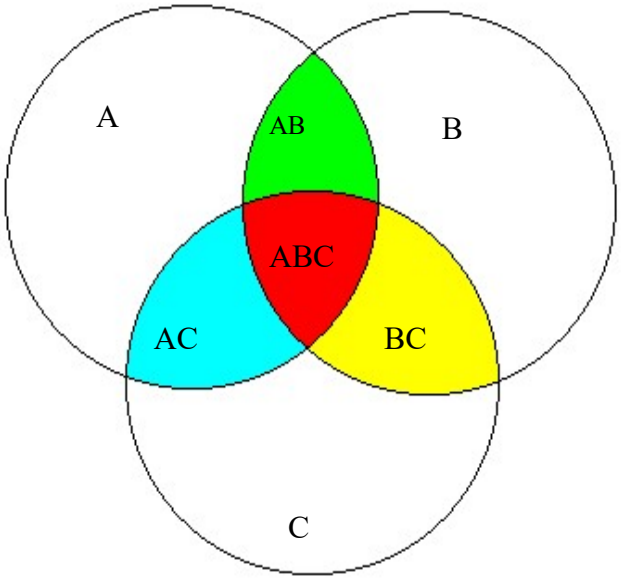
SQL> select * from Name where upper(ltrim(Name,3))='MAC';

In addition to chopping things up from the left we can chop from the right

SQL> select * from Name where upper (rtrim(Name,7))='DONALD';

SETS

For those of you who were half asleep that is s e t s. Cast your minds back to secondary school maths and drawing boxes with inter-locking circles. Well there is actually a use for Venn diagrams. They can be used where a selection is being made to find data occurring in some of tables but not in another



Defining the sets
A=Beer, only, drinkers
B=Larger, only, drinkers
C=Cider, only, drinkers

AB drink beer or larger
AC drink beer or cider
BC drink snake bites

ABC probably drink anything with an OH group in it.

The above Venn diagram discribes the drinking preferances of the BA exec club. There were four options. Ah ha you say there are only three circles. Yes, but everything outside of the three circles comprises the fourth data set or people who do not drink beer, larger or cider.

Now taking a work based example there are three tables that should contain data for all the members of a scheme. (All three tables contain the members' pin.) However by using the count clause you have found that the number of members in C does not equal the number of members in both A or B. This is represented by the white area of section C.

SQL provides three commands for dealing with this type of problem:

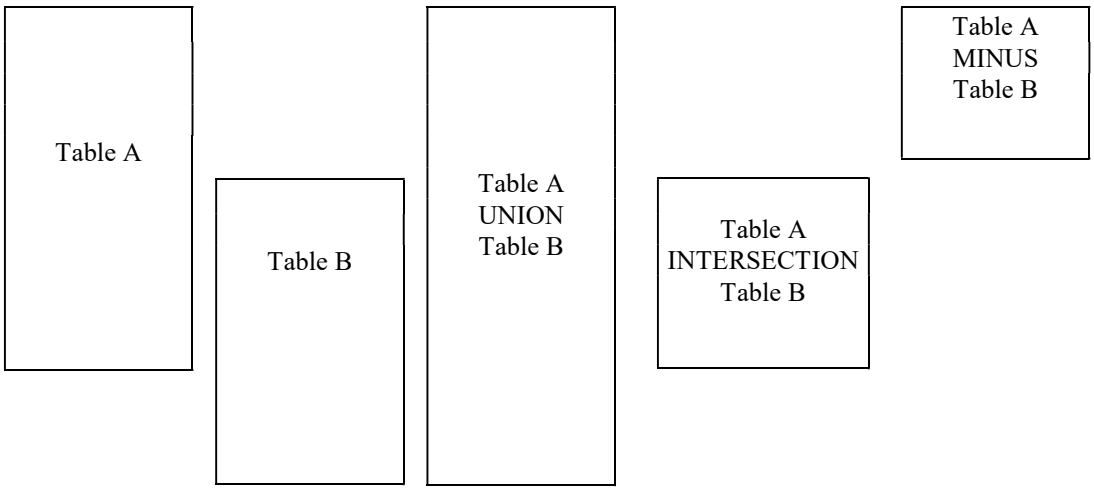
- Union returns all the distinct rows in that occur either of the queries
- Intersection returns the rows that satisfy both queries
- Minus selects all the rows returned by the first query that are not also returned by the second query

To find the rows that are unique to table C the statement will look like

```
SQL> select pin from C minus
  1 ( select pin from A union select pin from B );
```

This query will find all the pins in table C and then remove from this any pins that it finds by looking in either of table A and table B.

The following diagram represents the results of applying each of the operations to a pair of tables



The order that the **minus** operation is applied to tables makes a large difference to the result. If table A is minused from table B nothing is returned. There is nothing in table A that does not occur in table B.

There are some constraints when using union, intersect and minus. All the queries must select the same number of columns and corresponding columns must be of the same data type. The column names need not match though.

To order the results add a single order by clause at the end of the statement. The sort column must be named in the select clause.

DATES

Dates are kept as a date:time entry and displayed in the format DD-MON-YY. This can be somewhat confusing, as you will not know what century the date is in. The default behaviour of oracle is to treat any year of 50+ as being in the 20th century and any year of below 50 as being in the 21st century. This behaviour can be changed by the data base administrator so it is never a wise idea to trust this principle. To show the full year component of a date when asking SQL to provide you with a data you would use the to_char command.

SQL> to_char(date_column,'dd-mon-yyyy');

This statement consist of three sections:

to_char - which is the command
Date_column - which is the column name from which the date is obtained
'dd-mon-yyyy' - which is the formatting string for the display

The formatting string can be constructed from the following elements

Element	Meaning
YYYY	Year
YYY YY or Y	Last 3, 2 or 1 digit(S) of year
MM	Month number
MONTH or MON	Month name or three letter abbreviation
DDD DD D	Day of year, month or week
DAY or DY	Name of day or three letter abbreviation
HH24	Hour of day using 24 hour clock
/ . ,	Punctuation used in display

The capitalisation of elements within the formatting string determines how that element will be displayed. If the item is in upper case in the format string then that is how it will appear on the screen.

To get a date into a table use or to define the criteria dates in a select statement use the to_date function. The following creates a date entry of midday on December 31st 2000.

SQL> to_date('31-dec-2000','dd-mon-yyyy')

There are a restricted number of arithmetic functions that can be performed on dates :

date + number	Adds a number of days onto the date, producing a date
date - number	Subtracts a number of days onto the date, producing a date
date - date	Subtracts one date from another, producing a number of days

One date can not be added to another

To get SQL to display four digit years instead of the usual two type in the following line matching the spacing and case exactly

alter session set NLS_DATE_FORMAT='dd-mon-yyyy'

Insert, update and delete

So far everything we have looked at has involved manipulating existing data. Nothing has changed the actual contents of a table. All that is about to change. The following sections will give you untold power to wreak havoc with BA's data. Or for the less megalomaniac allow you to create test data.

Insert

The insert command creates a new row in an already existing table and populates that row with the data from the values clause of the statement. It creates a new record in an existing table. The commands format is:

insert into table values (*a list of data values*)

The number of values in the list must equal the number of columns and appear in the same order as the columns in the table. It does not matter if the data items are nulls. Data items are separated by commas and remember to enclose character and date strings in single quotation marks

```
SQL> insert into Tperson values
     1 ('00012345','Smythe',to_date('23-jan-2010','dd-mon-yyyy'),
     2 76819437,null);
```

If most of the row is to consist of null values rather than type loads of nulls the insert command can be instructed to populated selected fields with the supplied values and to then populate the non specified fields with nulls.

```
SQL> insert into Tperson (name) values
     1 ('Smythee');
```

Would result in the creation of the following row in the Tperson table
null, Smythee, null, null, null

When creating tracking items or new exec club members through CDMS it is the insert command that does the work of getting the information into the database.

Update

To change values in a row that already exists use is made of the update command. The command is made up of three clauses:

update *table*
set *field = value, field = value*
where *logical expression;*

If the where clause is missed out the update will be applied to the entire table.

Delete

Having created and amended rows the remaining item to discuss is removing them. The delete command has two clauses

delete from *table*
where *logical expression*

Missing out the where clause when using delete from is not a good idea as every row will be removed. The table will still exist, just with no rows.

Rollback & commit

In case of mistake, e.g. deleting an entire table, you have two options hari-kiri or undo the changes. To undo something type in

SQL> rollback;

and behold all of your work will be undone. That is everything changed since you invoked SQL. Oops

Loosing all of your work because of one mistake is a touch harsh. In order to keep changes that you know are correct use

SQL> commit;

Any rollbacks will now only undo changes made after the last commit. Commit also makes any changes that you have made visible to other users of the database.

If you are updating a table for someone else to work on commit the changes immediately or else your colleague will be looking at an incorrect version of the table.

If you are typing in data and being confident that you are not going to make a mistake then it is possible to have every line committed once it is typed in.
SQL> set autocommit on;
If you discover a mistake then the error needs to be corrected using the update command.

Should a SQL session be abruptly terminated (510, act of gods…) a rollback will be performed automatically.

&

Consider the following statements
SQL> select pin from Tperson where card_no = '45791795';
SQL> select pin from Tperson where card_no = '75197268';
SQL> select pin from Tperson where card_no = '35741945';

There are times when you want to execute the same select option with only one of the criteria changing. Instead of having to constantly edit the statement it would be easier to get the statement to prompt you for the new value.

There are two ways to do this a pretty way and the quick way.
The quick way is instead of populating the 'where X = ' clause of the select with a value you put in a variable name with an ampersand as a prefix to the variable name. The variable name can be anything bar reserved words and symbols.

SQL> select pin from Tperson where card_no=&card;

If the statement is run a series of events similar to that below occurs:

Enter value for card:
12345678 ←*user input*
old 1: select pin from Tperson where card_no=&card
new 1: select pin from Tperson where card_no=12345678

The &card in the old line has been replaced in the new line by the input of 12345678.

The pretty way involves producing a more descriptive prompt than Enter value for

___

As it involves more than one SQL statement it is best written as a script.
SQL> accept card prompt 'Enter the member's card number'
SQL> select pin from Tperson where card_no=&card;

The accept line will display your descriptive prompt and get the users input then pass this value across to the select statement.

These examples have used numeric input so none of them would actually work on CDMS because of its tendency to hold everything as character strings. To get the users input to be treated as a string the &card needs to be put inside of speech marks.

SQL> select pin from Tperson where card_no='&card';

A statement can contain multiple variables. As the number of variables increases the higher the demand for more descriptive prompts. There is a tendency to use &1 as the variable name. If someone else needs to use your SQL scripts then they will need to fully understand the script in order to work out what input is being requested by &1.

Create

So far everything we have worked with was already in existence and we have just modified the tables contents.
Occasionaly you may need a table to keep data on a temporary basis. Adding tables into the BA system is bound to upset the data base administrator. So this is how to do it

create *table name* (*column1 name column1 type* (*column1 width*) *column1 constraint*, *column2 name column2 type* (*column2 width*) *column2 constraint*, …)

To create a table you name the table and its columns. You specify what kind of information each column will hold and the maximum width of each column.
The column type defines what format the data is stored in.

Column type	how specified	description
char	char(size)	max of 240 characters
	varchar(size)	same as char
date	date	date between 1/1/4712BC and 31/12/4712AD
number	number	can be expressed as +/- or sign & magnitude
decimal	decimal(scale)	scale is the number of decimal places to be used
float	float	same as decimal
integer	integer	number with no decimal part
smallint	smallint	

Some of the above terms are being used in a slightly different way to their usage in programming languages. In many languages a small int is a small whole number that has been allocated one byte of storage space in memory. It can have a range of 0 to 255 or -127 to +128.
One common data type is missing. Boolean, which is used for holding logic values of true or false. CDMS tends to use type char and assigning either a Y or N in place of logic values. Another convention is to use a small int to hold either 1 or 0.

The constraint setting allows for data to be checked against certain conditions before it is inserted into the table. If the data falls outside of the constraint condition then the whole insert or update operation will fail. The most common constraint encountered is ‘not null’ which ensures that a value is entered into the field. The constraint could be any one of the comparison operators discussed earlier and link to values in another table.

Limits on table dimension

Item	Limit
Columns in a table	254
Total sum of characters across all columns in a row	126,495
Characters in a field 240	240
Digits in a number field 105	105
Date range	1/1/4712BC to 31/12/4712AD
Length of a column name	30 characters

Each table takes up disk space. Even before it is populated. When a database is created it is allocated a certain amount of space and all of its tables, indexes and constraints have to fit in this space. I learnt SQL on a VAX. Not the vacuum cleaner that washes although that probably has has much computing power as the college vaxes had. When creating a new database on the vax you actually had to define how much disk space was allocated to that database and where it had to look for the disk space.

Views

There are times when you are constantly selecting just a couple of columns from a table. Or in the case of OCD data that you expect to be contained within one table is split across three. Instead of constantly writing the same select the data you can define a view of the table that just displays what you want. A view is a virtual table that takes up no space and is always updated with the latest changes to its component data. You may say that as all of my queries are saved as scripts what use is a view. What if you want to perform some mathematical functions on the results or compare one table against another using the intersection function. Both of these would require significant and clumsy rewrites of the script.

create view *name* (*columns in view)* as select *columns* from *table;*

To query a view just treat it as a normal table

A view can not contain an order by clause within its definition. If the rows need to be ordered you can do this when querying the view.

If a view is to span across two or more tables the relationship or join between the tables must be defined

```
SQL> create view projstaff  (name, dept, project, project_number) as
  2  select Tperson.name, dept, pname, projno
  3  from Tperson, proj
  4  where Tperson.name = proj.name;
```

This would create a view that shows what departments are loosing staff to which project
The words in brackets after the create clause become the column names of the view. These columns will use the variable type of the column in the select clause that appears in the same position in the list.
Line 4, the where clause, defines the join between the two tables. The columns selected to be the join must be of the same type.

Views can be joined to other views or tables.

Nasty stuff

Again this section is intended for just UAT and is realy a filler should things run ahead of schedule or to cover any questions people might ask

Inserting from one table to another

```
insert into Tmove_to
(sample_date, rain, city,noon_temp, midnight_temp)
select
date, rain, city, noon, midnight
from Tweather
where rain<10 and (noon+midnight)<16;
```

If populating mutliple tables with similar data the simplest method is to create a tempory table, type your data into this and then take the data from this table..

Sequences

There are time when the data in one column will increment by a set amount form the previous column. The members of this sequence could be typed in by hand or you could have SQL generate the numbers automaticaly

create sequence EmployeeID increment by 1 start with 100000;

To make use of the sequence

insert into Tperson (pin) values (EmployeeID.nextval);

Every time nextval is called the sequence is increased by the increment value.
If a number needs to be used without incrementing it use currval.

To stop a sequence use the drop sequence command. To skip a block of numbers the sequence needs to dropped and recreated to start after the skipped block

Rename

To change the name of a table, view or synonym to something memorable rather than 1 or fred which you used while experimenting use
SQL> rename *old* to *new*

Spool

This command makes all a copy of everything that appears on the screen to the named file.
SQL>spool *filename*

To stop the screen capture you use somewhat predictably spool off.
The file will appear in filel where it can be edited.
Apparently spool out prints the file but at what printer is anyone's guess. It is not always obvious where your printer is. When first using Lotus notes my default printer was in Glasgow and I worked in Woking.

Comments

In programming languages a facility exists to allow for lines of lines of text to be in the code that are purely for human use. The computer ignores them.
The advantage of comments are that they can be used in scripts to describe lines of code or explain complex statements. If a statement is being used to test a particular condition why not put a comment in saying so. If a script is written to test a PR/CR put that in the comments at the start of the file.
There are two ways to mark a line as a comment.
The most common is
The REMARK statement. Placing this at the start of a line causes SQL to ignore the whole line. Or use the notation from the C language of /* */
the /* starts the comment and */ ends the comment. It is best to open and close the comments markers for each line. e.g.
/* start of comments */
/* end of comments */
The results of opening the comment on one line and closing it on another are unpredictable.
Sticking to the remark method is the safest option since it describes its own funcion.

A comment can be attached to a table or a column. These comments do not get used much it is covered here for completeness.

comment on table *table name* is *comment*
comment on column *column name* is *comment*

to view the comments

select * from all_tab_comments
select * from all_col_comments

Find out what tables are in the schema

To list all the tables in the database
select table_name from dba_tables
in older versions of oracle select * from dtab
Some times they try and be clever and restrict acess to the system tables. In most enviroments with the execption of JE06 the above commands do not work. So try
select table_name from all_catalog where table_type='table';

If you are interested in only the tables you have created replace the 'all' with 'user'.

FINISH

You now know about SQL as much as I do without having to sit through two years worth of college lectures. Thank you for not heckling and I hope something covered today helps.