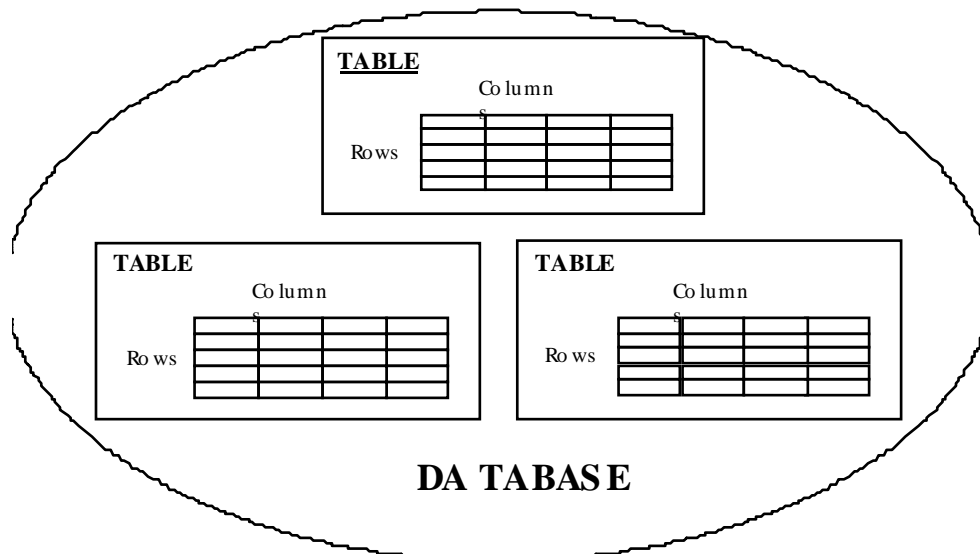


SQL USING WORD PROCESSING

I. INTRODUCTION

What is a database?

A database is a collection of information (data). The information in the database is arranged in tables. The tables consist of **rows**, sometimes called records, and **columns**, sometimes called fields. The database tables also contain **indexes**. Indexes make searches faster and allow sorts to take place. Indexes also make it possible for data in one table to be easily matched to corresponding data in another table. Most tables in the CARS database containing are indexed on their id number.



Examples of tables include:

- **id_rec**: Contains the names and general information of all individuals and entities
- **aa_rec**: Contains alternative address records
- **ed_rec**: Records attendance at another educational institution; includes class rank, gpa, and attendance dates.
- **emp_rec**: Contains records for non-university employment
- **profile_rec**: Contains biographical information such as birthdate and gender.
- **se_elm_rec**: Contains email address information for St. Edward's University faculty, staff and students.

Every constituent or record in the Jenzabar CX (CARS) database has information in at least one table – the **id_rec**. There are currently over 400,000 rows or records in the database, each with information stored in the **id_rec**. The columns or fields that make up the **id_rec** include the following:

Columns/Fields

Row/Record	id	fullname	addr_line1	city	st	zip
	136885	Adrian, Brenda	1514 W. 9 th Street	Austin	TX	78703
	400678	Smith, Joseph K.	101 E. 6 th St.	Austin	TX	78704
	417444	Garcia, Paul	220 Hwy 81	Dallas	TX	75551

The record for id 136885 is for Brenda Adrian. This record contains only information about this one constituent. The field fullname is different for every record in the database.

Every record in the table id_rec has a unique field, the id field. The id field consists of a serially assigned number from 1 – current number (440,000). This is the same ID number that is found on student and staff ID cards. This id field is also found in many other tables in the database. For example the profile_rec also has an id field.

Constituents with multiple connections to the university such as former student, alumni, staff, and donor may have data in many tables in the CARS database. In this case the person would have information stored in the id_rec, profile_rec, stu_acad_rec, program_enr_rec, emp_rec, ed_rec, se_elm_rec, aa_rec, alum_rec, hrempp_rec, pledge_rec, gift_rec. The person would likely have information in many other records also.

How am I able to get information on a constituent that includes all of the information in all of those tables?

Each of these tables is connected through a common field, the id field.

id_rec	id	fullname	city	st
	136885	Adrian, Brenda	Austin	TX
profile_rec	id	birth_date	sex	vet
	136885	11/13/1959	F	N

To access information across many tables for one constituent you need to connect these tables via the common field, the id field.

How do I get information from a database?

SQL, which stands for Structured Query Language, is one of the most common tools used to extract information from a database. SQL uses ordinary English words to obtain reports from database tables. Each SQL statement creates a table of rows and columns containing the data requested by the statement.

SQL is also at the heart of the ACE Report Writer, which is used to present information obtained using SQL statements in a format which is more comprehensive. Other available tools include Word Processing and the Data Dictionary Menu.

How can I use SQL to get information from the CARS database?

The CARS database includes a program called Word Processing that can be used to store statements that can be run in SQL. Word Processing is somewhat of a misnomer, because here

at St. Edward's we use Microsoft Word as our word processor. Some offices use the **EDT+** editor to write SQL statements and ACE Reports. You may also use **SEPic** as an editor.

II. DATA DICTIONARY MENU

How do I know what tables to use for my SQL statements?

The first thing to determine is specifically what information you want.

- What criteria are you using to select the information?
- How are you narrowing down your selections?
- Who is included? Why? Who is excluded? Why?
- How do you want it sorted or grouped?
- Do you want any calculations performed on the data?
- Should specific date ranges be included in the selection criteria?

The clearer you are on what you want, the easier it is to figure how out to get it. SO.... the first thing to do is write down what it is you're looking for. Then, you can use one, or any combination, of the three ways to determine which tables you will need:

1. If you have been doing **data entry**, or have access to data entry screens, you can look at the data entry screens to see what tables you've been using. The table name is displayed when first entering data entry and then again when editing a specific screen.
2. Another way to find out the names of the tables you need is to **ask someone**. Usually there is someone in your office who knows what tables have additional relevant information for your department or inquiry, and until you begin to get familiar with the tables yourself, this is a good way to learn. If there is no one in your office who can help you, you may call the Computer Center. **Be sure to apprise the person you ask with all the details of the information you need to ensure that you receive an accurate and complete response.**
3. You can also use items on the **Data Dictionary Menu** to look up field and table names.

What is the Data Dictionary Menu?

The Data Dictionary Menu has a selection of reports and query screens to help you find the information about the tables you need to create your SQL statements. The Data Dictionary Menu is located under the Utility Menu. It is currently option **h**).

As mentioned earlier, the database is broken down into tables, and the tables are broken into rows and columns. On the Data Dictionary Menu, the tables are referred to as **Files**, and the columns are referred to as **Fields**. You can use **Database Files** and **Database Fields** to determine which tables you will need to extract the desired information from the database.

Database Files

Selecting this menu option brings you to a PERFORM screen where you can query on a File (table) and see all of the attached Fields (columns). If you are unsure of a file name, you can use wildcards (*) to search. For example, entering ed* returns 4 rows. To move from row to row, press **N** or use the arrow keys to move to **Next** and press **<enter>**.

To see the **Detail** or **Database Fields** associated with each file, press **D** or use the arrow keys to move to **Detail** and press **<enter>**. To move from field to field, press **N** or use the arrow keys to move to **Next** and press **<enter>**.

Information on these screens that will be helpful to you when creating SQL reports includes:

- File Name (table name)
- Field Name
- Field Type
 - Char -- text fields of specific length
 - Date
 - Int – whole numbers
 - Float – decimal numbers
- Field Length

Database Fields

Selecting this menu option brings you a PERFORM screen where you can query on Fields (columns) and see which Files (tables) they are a part of. Some fields are in more than one file. These fields are generally indexed so that the tables can be linked through them. If you are unsure of a field name, you can use wildcards (*) to search. For example, entering ed* returns 18 rows found. To move from row to row, press **N** or use the arrow keys to move to **Next** and press **<enter>**.

Fields by File Report

Use this option to print (to paper or screen) the field information (including a description) for the selected file or files.

Fast Schema List

Once you know the file name, use this option to print the list of fields (columns), field type, and whether or not null values are accepted.

Some Commonly Used Tables	
Table Name	Some items it contains
id_rec	Full name and address; phone number
aa_rec	Alternative address records
emp_rec	Non-SEU Employee information
se_elm_rec	All SEU faculty, staff and current students login name or email address
profile_rec	Demographics such as age, sex, ethnicity, and citizenship

Schema for id_rec

id_rec

Column name	Type	Nulls
id	serial	no
prsp_no	integer	no
fullname	char(32)	no
name_sndx	char(4)	no
addr_line1	char(24)	yes
addr_line2	char(24)	yes
city	char(24)	yes
st	char(2)	yes
zip	char(10)	no
ctry	char(3)	yes
aa	char(4)	yes
title	char(4)	yes
suffix	char(4)	yes
ss_no	char(11)	no
phone	char(12)	yes
phone_ext	char(4)	yes
prev_name_id	integer	yes
mail	char(1)	yes
sol	char(1)	yes
pub	char(1)	yes
correct_addr	char(1)	yes
decsd	char(1)	yes
add_date	date	yes
ofc_add_by	char(4)	yes
upd_date	date	yes
valid	char(1)	yes
purge_date	date	yes
cass_cert_date	date	yes
own_by	char(4)	yes
owned	char(1)	yes
long_tmp	integer	yes
dup_rev	char(1)	yes
norel	char(1)	yes
no_ck	char(2)	yes
chg_uid	smallint	yes
prime_id	integer	yes
apptmp_id	integer	yes
pw_exists	char(1)	yes

Schema for aa_rec

aa_rec

Column name	Type	Nulls
aa_no	serial	no
id	integer	no
aa	char(4)	no
beg_date	date	no
end_date	date	yes
peren	char(1)	yes
line1	char(32)	yes
line2	char(32)	yes
line3	char(32)	yes
city	char(32)	yes
st	char(2)	yes
zip	char(10)	yes
ctry	char(3)	yes
phone	char(12)	yes
phone_ext	char(4)	yes
ofc_add_by	char(4)	yes
cass_cert_date	date	yes
chg_date	date	yes
chg_uid	smallint	yes
pcode	char(1)	yes

Schema for emp_rec

emp_rec			
Column name	Type		Nulls
emp_no	serial		no
id	integer		no
bus_id	integer		no
bus	char(32)		yes
pos	char(32)		yes
occ	char(4)		yes
beg_date	date		yes
end_date	date		yes
pd_ben	char(1)		yes
bus_phone	char(12)		yes
bus_ext	char(4)		yes
crpos_no	integer		no
stat	char(1)		yes
se_oc1	char(4)		yes
se_oc2	char(4)		yes
sp_emp	char(1)		yes
se_stat	char(1)		yes
chg_date	date		yes
chg_uid	smallint		yes
ctc_ok	char(1)		yes
retired	char(1)		yes

Schema for profile_rec

profile_rec			
Column name	Type		Nulls
id	integer		no
ethnic_code	char(2)		yes
sex	char(1)		yes
mrtl	char(1)		yes
birth_date	date		yes
age	smallint		yes
birthplace_city	char(24)		yes
birthplace_st	char(2)		yes
birthplace_ctry	char(4)		yes
res_st	char(2)		yes
res_cty	char(4)		yes
citz	char(4)		yes
visa_code	char(4)		yes
prof_visa_date	date		yes
prof_visa_no	char(10)		yes
res_ctry	char(3)		yes
vet	char(1)		yes
handicap_code	char(2)		yes
denom_code	char(4)		yes
church_id	integer		no
occ	char(4)		yes
news1_id	integer		yes
news2_id	integer		yes
prof_last_upd_date	date		yes
lang	char(1)		yes
prof_res_code	char(5)		yes
prof_res_date	date		yes
prof_vet_chap	char(4)		yes
grp_no	smallint		yes
password	char(8)		yes
priv_code	char(4)		yes
decsd_date	date		yes
photo	byte		yes
cit_ctry	char(3)		yes
vet_code	char(5)		yes
vet_num	char(10)		yes
film	smallint		yes
i9	date		yes
wr_off	char(2)		yes
no_note	char(2)		yes
no_edpy	char(2)		yes
nds1	char(2)		yes
collectech	money(16,2)		yes
manage	char(1)		yes
delinq_no	smallint		yes
delinq_amt	money(16,2)		yes
insure	char(2)		yes
ins_repa	char(1)		yes

fin_id	integer	yes
email_fin	char(1)	yes
have_car	char(1)	yes
aid_opt_fees	char(1)	yes
aid_eft	char(1)	yes
chg_date	date	yes
chg_uid	smallint	yes
per_res	char(1)	yes
reg_issue	date	yes
reg_num	char(10)	yes
need_i20	char(1)	yes
stu_visa	char(1)	yes

Schema for se_view_rec

Column name	Type	Nulls
id	integer	no
salut	char(32)	yes
line1	char(32)	yes
line2	char(32)	yes
line3	char(32)	yes
line4	char(32)	yes
line5	char(32)	yes
line6	char(32)	yes
st	char(2)	yes
zip	char(10)	yes
city	char(32)	yes
email	char(8)	yes
ctry	char(3)	yes
phone	char(12)	yes
phone_ext	char(4)	yes
wphone	char(12)	yes
wphone_ext	char(4)	yes
work_id	integer	yes
run_date	date	yes
run_code	char(8)	yes

Sample using Field by File Report

Mon May 4 1998

St. Edwards University

Page

1

11:44

DATABASE FIELDS FOR THE CARS DATABASE DICTIONARY

dbefield

For id_rec Schema

INFORMIX File Name

id_rec

Field Name Description	Type Index	Length	Status Date
cass_cert_date CASS Cert Date	date No Index	4	? 05/27/1997
Date the address was last CASS certified.			
chg_uid chg_uid	smallint No Index	2	? 08/17/1994
UID of person making last change (or adding record.			
own_by Owned by	char No Index	4	? 02/10/1996
The office code for the office that owns this record. This office is responsible for name and address changes.			
prime_id Primary id	integer No Index	4	? 09/29/1995
ID of primary record for bus/org/vendors, 0 (or null) if only one in group.			

For id_rec Schema

INFORMIX File Name			
id_rec			
Field Name	Type	Length	Status
Description	Index		Date
=====aa			
char		4	Active
Address Code - Id	No Index		08/17/1994
Identifies the type of address entered for this entity (see the alternate address table).			
add_date	date	4	Active
Add Date - Id Record	UNKNOWN INDEX		08/17/1994
Date identification information was added to the system.			
addr_line1	char	24	Active
Address Line 1	UNKNOWN INDEX		08/17/1994
First line of entity's address.			
addr_line2	char	24	Active
Address Line 2	UNKNOWN INDEX		08/17/1994
Second line of entity's address.			
city	char	24	Active
City	UNKNOWN INDEX		08/17/1994
City name where entity is located.			
correct_addr	char	1	Active
Correct Address - Y/N	UNKNOWN INDEX		08/17/1994
Y/N Indicates if this is a correct address for this entity.			
ctry	char	3	Active
Country Code	No Index		08/17/1994
Identifies country in which the entity is located (see country table).			
decsd	char	1	Active
Deceased - Y/N	No Index		08/17/1994
Y/N Indicates if this entity is deceased.			
dup_rev	char	1	Active
Dup rev	No Index		08/17/1994

For id_rec Schema

INFORMIX File Name			
id_rec			
Field Name	Type	Length	Status
Description	Index		Date
fullname	char	32	Active
Name	No Index		08/17/1994
Name of the entity being identified in this record. Format is - Last_Name,First_Name Middle_Initial, Suffix			
id	serial	4	Active
ID Number	No Index		08/17/1994
Serial system assigned number identifying this identification record.			
long_tmp	integer	4	Active
TMP_ID	No Index		08/17/1994
Temporary field used by the name update process.			
mail	char	1	Active
Mail - Y/N	UNKNOWN INDEX		05/27/1997
Y/N Indicates if this entity should receive mailings. NOTE: This column is notational only.			
name_sndx	char	4	Active
Phonetic matching name.	No Index		08/17/1994
Code used for phoneticly matching the name.			
no_ck	char	2	Active
No check	No Index		08/17/1994
No check code used by student accounts			
norel	char	1	Active
Restrict release	No Index		08/17/1994
Joins to norel_table to describe reason for restricted release of information			
ofc_add_by	char	4	Active
Office Added By	UNKNOWN INDEX		08/17/1994
Office which added the entity to the system.			

For id_rec Schema

INFORMIX File Name			
id_rec			
Field Name	Type	Length	Status
Description	Index		Date
phone	char	12	Active
Phone Number	UNKNOWN INDEX		08/17/1994
Area Code and Telephone number of the entity. Desired parenthesis and dashes should be included in field.			
phone_ext	char	4	Active
Phone Extension	UNKNOWN INDEX		08/17/1994
Extension to phone number of entity.			
prev_name_id	integer	4	Active
ID - Previous	UNKNOWN INDEX		08/17/1994
Identifies the person this individual was previously known as.			
prsp_no	integer	4	Active
ID Number - Prospect	No Index		08/17/1994
Identifies the original lead id number when this individual was only on the system as a lead.			
pub	char	1	Active
Publicity Mailings - Y/N	UNKNOWN INDEX		05/27/1997
Y/N Indicates if this person should receive publicity notices. NOTE: This column is notational only.			
purge_date	date	4	Active
Date ID May Be Removed	UNKNOWN INDEX		05/27/1997
Date that this entity is eligible to be removed from the system. NOTE: This column is notational only.			
sol	char	1	Active
Solicitation - Y/N	UNKNOWN INDEX		05/27/1997
Y/N Indicates if this person may receive solicitations. NOTE: This column is notational only.			
ss_no	char	11	Active
Social Security Number	Duplicates Allowed		08/17/1994
Outside agency identification number of entity. That is: SS Number for person, Fed Id for business, CEEB for Schools.			

INFORMIX File Name

id_rec

Field Name Description	Type Index	Length	Status Date
st State Code	char No Index	2	Active 08/17/1994
Identifies the state in which the entity is located. (See the state table).			
suffix Suffix to Name	char UNKNOWN INDEX	4	Active 08/17/1994
Identifies the suffix with which this entity should be addressed (see the suffix table).			
title Title Code	char UNKNOWN INDEX	4	Active 08/17/1994
Identifies the title with which this entity should be addressed (see the title table).			
upd_date Last Update Date - Id	date No Index	4	Active 08/17/1994
Date that id information was last updated.			
valid Valid - Id	char No Index	1	Active 08/17/1994
zip Zip Code	char Duplicates Allowed	10	Active 08/17/1994
Zip code of current location of entity.			

Location: /usr/carsi/modules/common/reports/dbefield

Revision: Developmental 10/01/93 19:26:04

III. WORD PROCESSING

How do I use Word Processing?

Word Processing is accessed by pressing **Shift-W** (Word Processing) at any CARS menu. On any Word Processing screen, there will be a list of permissible commands across the top of the screen. You may use these commands to maneuver through Word Processing.

When you select Word Processing, you will see a listing of **Cabinet Names**. Word Processing is structured using File Cabinets, Drawers, and Files, where Files exist inside Drawers, and Drawers exist inside File Cabinets. Some typical names for File Cabinets are **common** (everyone has access), **admissions** (or other departments or groups, where only people in that group have access), and **private** (where only you have access). Choose the desired Cabinet by typing its number.

```
Word Processing using SEpico
o - open cabinet                                C - Exit Word Processing
----- Cabinet Names -----
1. itec
2. common
3. display_reg
4. ac_ro
5. id_add
6. netmanag
7. helpdesk
8. hd_stu
9. comp_comp
10. helpcars
11. rpa
12. private
Enter Command: █
```

In Word Processing the menu options are always at the top of the screen above the horizontal line. In the screen above there are two options

- o to open one of the cabinets listed
- C to exit Word Processing

Enter the menu option at the Command prompt at the bottom of the screen.

Hint: The shortcut for opening a cabinet or file is to type the number to the left of the cabinet or file at the Command prompt. In this case type 12 at the Command prompt to open the private cabinet.

Once you have selected a Cabinet, you will see a listing of **Drawer Names**. (If this is the first time you've accessed your **Private File Cabinet** in Word Processing, there will be no Drawers listed.)

```

----- File Cabinet: private -----
o - open      r - rename      b - back screen      C - Close File Cabinet
n - new       d - delete      f - forward screen   X - Exit Word Processing
----- Drawer Names -----
1. letters
2. merge
3. reports
4. wastebasket
5. wpreports
Enter Command: █

```

Drawers are the equivalent of sub-directories or folders. You can create as many drawers you like, However, Word Processing has four names that are **reserved**.

RESERVED DRAWER NAMES in WORD PROCESSING and THEIR USES	
DRAWER NAME (must be exactly as shown)	DESCRIPTION
letters	Letter formats to be used by the letter processing system
wpreports	ACE reports used by the letter processing system
reports	ad hoc ACE reports and SQL statements
wastebasket	for deleted files

When the Drawers named reports and wpreports are created, a **SAMPLE** file is inserted into it. This **SAMPLE** file is an example of an ACE report, and can be copied and used as a format when creating your own ACE reports.

```

----- File Cabinet: private ----- Drawer: reports -----
o - open      c - copy      p - print      h - help      b - back screen
                r - rename      a - analyze      f - forward screen
                d - delete      l - locate      e - expand      C - Close drawer
                m - move      s - sort list   t - translate   X - Exit Word Proc
-----File Name-----Size--St--Modified-----File Name-----Size--St--Modified--
  1. 32sample      96  Nov 13 2002   22. ace37      878 T Oct 20 1998

```

What is the directory path for Word Processing files?

The directory is: **/opt/carsi/wp/filecabinetname/FileCabinet/Drawer/File**. For example, the directory path to the file **CARSacct** is:

/opt/carsi/wp/common/FileCabinet/requestforms/CARSacct

Another example: To access reports in the ITEC file cabinet reports drawer:

/opt/carsi/wp/itec/FileCabinet/reports

SQL using Word Processing Step by Step Instructions for accessing the Private File Cabinet and the Reports Drawer

1. Access Word Processing
 - a. From the Main CWIS Menu, select **Administrative Database (I)**
 - b. From any CARS menu, select **Word Processing** by pressing Shift-W.
2. Access your private File Cabinet
 - a. Type the number for the Cabinet called “private”
 - OR-
Press **o** for Open Cabinet and type **private**
 - b. Press **<Return>**
4. Open the file drawer **reports** by entering the number to the left of reports at the command line or typing **o** for Open Drawer and type **reports**
5. To create a new report type the filename at the Command prompt.
NOTE: File names in Word Processing are limited to **8** characters.
6. To edit an existing SQL report:
 - a. Press **o** for **open** and type in the file name. Press **<Return>**.
 - b. You will be in your editor (EDT or Pico).
 - c. Type the Select statement(s).
 - d. Exit the editor as usual.

IV. THE SQL SELECT STATEMENT

The Select Statement

The select statement consists of a series of **clauses** that tell SQL which items to take, from which tables to take them, using what criteria, sorted into what order, and placed where. **Every field that you want to appear in the output must be included in the select statement.**

The clauses in a Select Statement are:

1. **SELECT** (required) which items
 2. **FROM** (required) which tables
 3. **WHERE** (optional) using what criteria (restricts rows returned)
 4. **GROUP BY** (only used for aggregate functions)
 5. **HAVING** (only used with GROUP BY)
 6. **ORDER BY** (optional) sorted in what order
 7. **INTO TEMP** (optional - default is to screen) placed where
- ; Use a **Semi-colon** between each Select statement, except **after** the **last** statement

The Select Clause

The syntax for the SELECT clause is: (Syntax means “How do I say it so the computer understands it?”. Items in braces are not required, but can be options. A “|” means “or”.)

SELECT [ALL | DISTINCT | UNIQUE] **item1** [alias_item1], **item2** [alias_item2], etc.

The keywords ALL, DISTINCT, and UNIQUE allow us to retain or throw away duplicate rows found. The keywords are used as follows:

ALL (default) Retain all duplicate rows.

DISTINCT Retain one copy of each duplicate row.

UNIQUE Same as DISTINCT.

Each field (or column) in the select statement must be separated from the previous field by a **comma**. The last field in the select statement does not require a comma.

Example: This example will display the field's id, fullname and zip in the output.

```
select id,
       fullname,
       zip
```

Note that the formatting of the select statement does not matter. The following is exactly the same as the above example:

```
select id, fullname, zip
```

Formatting by indenting the field names just makes it easier for the human eye to read.

The fields you are selecting can be of one or more of the following:

- | | |
|----------------------------|--|
| columns | Values found in fields in the database. The type of data that is in a field can be important. See <u>SQL using Word Processing Handout IV. Reference</u> for information on the types of data in columns. |
| constants | Constants values. e.g., 42, “hello”, 3.14 |
| expressions | Columns, constants, keywords, and functions that are evaluated using the arithmetic operators. |
| Wildcards such as * | Use * to select all columns from a table. |

The From Clause

The syntax for the FROM clause is:

FROM [OUTER] **table1** [tbl1_alias], **table2** [tbl2_alias], table3 [tbl3_alias], etc.

- **Every table that is referenced in the Select clause must be listed in the From clause.**
- Use **commas** to separate table names.

Example: This example uses information from 3 tables.

```
from      id_rec,
          profile_rec,
          se_elm_rec
```

A table alias can be defined when desired. The table alias can be used in the Select clause and in the Where clause.

Example: In this example the id_rec table is referenced twice. The first time the table is aliased to student and the second time it is aliased to parent.

```
from      id_rec student,
          id_rec parent,
          profile_rec
```

A join condition MUST be defined in the Where clause when more than one table is indicated in the From clause.

The Where Clause

Even though a Where clause is technically an option, given the large numbers of records that exist in the database tables, the Where clause is, in reality, more of a requirement than an option. The Where clause allows you to specify exactly which records you wish to see. For example, you can choose which major(s), session(s), year(s), or other precise criteria you wish to use to define your list of records.

The columns evaluated in a Where Clause **DO NOT** need to be in the **SELECT** clause.

The Where clause uses three types of conditions:

1. The Comparison Condition
2. The Join Condition
3. The Condition with Subquery

AND and **OR** are used when multiple conditions in a Where clause are necessary.

A Comparison Condition can be preceded by the keyword **NOT**, in which case the inverse of the condition must be satisfied.

Example: In this statement, id_rec is the table name and id is the field name. The statement id_rec.id=profile_rec.id is an example of a **join statement**. The where statement also contains a comparison condition that limits records returned to those with id's greater than 400000.

```
where id_rec.id = profile_rec.id
      and id_rec.id > 400000
```

Example 1 A simple SQL statement: This example will return on output all rows or records which have an id number greater than 400000. The fields that will be displayed on output will be the id and fullname fields.

```
select      id,  
           fullname  
from        id_rec  
where       id > 400000
```

Field Types

In the schema the type of each field is listed. The most common types are

- integer – whole numbers
- serial – auto-generated numbers such as id
- char – text fields with specific lengths
- date

When using a field in a where clause it is necessary to include double quotes “ ” around date and character fields. It is not necessary to do so with integer or serial fields.

Running SQL Reports in the CARS System

1. Access Administrative Data Base (I)
2. Select the **Utility Menu**.
3. Select **Letters/Labels/Reports**. This puts you in the correct menu to run your SQL reports.
4. Access Word Processing (i.e., Shift-W for Word Processing).
5. Select your **private** File Cabinet by typing in the number in front of it and pressing **<Return>**.
6. Select the **reports** drawer by pressing the number in front of it and pressing **<Return>**.
- 7.a. To create a new file:
Press o for **open**, **<enter>**, and enter the **file name**. Remember, the file name must be less than or equal to 8 characters.
- 7.b. To edit an existing file:
Type the number in front of it and press **<Return>**.
8. Type and/or edit your SQL statement.
9. Check it for typos and syntax errors (e.g., missing commas).
10. Exit the editor, saving the file.
11. Press **Shift-X** to Exit WPVI and press **<Return>** or press **Control-Z** to suspend Word Processing and go back to the Utilities menu..
12. Press **<Return>** to continue.
13. From the Letters/Labels/Reports menu select **[j]-WPVI SQL Report**.
14. Type in the name of the File Cabinet and the report. NOTE: You do not have to type the name of the drawer. The system knows that it is in the Reports Drawer. Therefore, for this class, type *private/filename*.
15. Press **PF1**.
16. Press **<Return>** to acknowledge that you are running an SQL statement and not an ACE Report.
17. Type **file** in as the mode to run the report to a file.
18. Press **PF1** to execute the report.
19. If the report runs correctly, you will receive an email with the subject: Output for 'reportname' created.
20. If the report does not run correctly, read the message(s) in your email. They will give you some indication as to the problem.
21. To re-edit the report if you have suspended Word Processing go to **Tasks** and select **Word Processing**. Press **PF1** and then **<Enter>**. If you do not have Word Processing suspended press **W** to access Word Processing.

Fields Which Exist in More than One Table

If the name of any field exists in more than one of the tables in the From clause, you must identify which table you want to pull the field from. Place a period (.) between the table and column name when typing the column in the select clause. The format is **table_name.field_name**. To identify that the field id is in the table id_rec use this syntax: id_rec.id

Example 2: Field id exists in 2 tables

In this example, the field id is in both id_rec and profile_rec. The id field to be used has been identified as id_rec.id.

```
select      id_rec.id,
           fullname,
           sex
from        id_rec,
           profile_rec
where      id_rec.id = profile_rec.id
           and id_rec.id > 400000
```

Truncating Field Length

When a requested column has character data, you can truncate the results you see, in order to shorten the display. Use brackets immediately after the column name. The first number in the brackets indicates which character to begin with. The second number indicates which the ending character is.

Example 3: Truncating field lengths in output

In this example, only the first through the 10th character of the fullname field would be displayed.

```
select  id_rec.id,
        fullname[1,10],
        sex
from    id_rec,
        profile_rec
where  id_rec.id = profile_rec.id
       and id_rec.id > 400000
```

Functions Available in the Select Clause

Aggregate Functions

Aggregate functions allow you to perform operations (such as the calculations of sums or averages) using all of the selected columns collectively.

count(*)	Number of rows total
sum(col)	Total of all values in column/field col
avg(col)	Average of all values in column col
max(col)	Returns the maximum value of all values in the column
min(col)	Returns the minimum value of all values in the column

Example 4: Simple count

This query returns the number of constituents with ID #'s greater than 400000 who live in Dallas.

```
select count(*)
from id_rec
where city = "Dallas"
      and id > 400000
```

Example 5: Another count

This query returns the number of female constituents with ID #'s greater than 400000 who live in Dallas.

```
select count(*)
from id_rec,
     profile_rec
where id_rec.id = profile_rec.id
      and city = "Dallas"
      and sex = "F"
      and id_rec.id > 400000
```

Example 5A. Using the Min function

This query returns the oldest person in the list (or the min birth_date).

```
select min(birth_date)
from id_rec,
     profile_rec
where id_rec.id = profile_rec.id
      and city = "Dallas"
      and sex = "F"
      and id_rec.id > 400000
      and birth_date is not NULL
```

Date Functions

Date functions allow you to insert and/or perform calculations on columns containing dates.

date	Returns the date-type value of the expression passed to it
day	Returns the day of the month of the date passed to it
month	Returns the month of the year of the date passed to it
year	Returns the year of the date passed to it
today	Returns the current date
mdy	Returns a date-type value of the three parameters passed to it
weekday	Returns a value between 0-6 denoting the day of the week (0 – Sunday through 6 – Saturday)

Example 6

Use the date functions to produce a list of people with March birthdays.

```
select  id_rec.id,
        fullname,
        month(birth_date) bmonth,
        day(birth_date) bday
from    id_rec,
        profile_rec
where   id_rec.id = profile_rec.id
        and city = "Dallas"
        and sex = "F"
        and id_rec.id > 400000
        and month(birth_date) = "03"
```

Using Relational Operators in the Where Clause

Operator	Operation
=	equals
!= or <>	not equal
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

Example 7: Using Relational Operators

This example uses the >= and < operators to return all constituents who live in the zip code range 78703-0000 through 78703-9999.

```
select  id,
        fullname,
        zip
from    id_rec
where   zip >= "78703" and zip < "78704"
```

Comparison Condition

Use the comparison condition in the **where clause** to select or filter the rows that you want by comparing the item in a column to another item, a range of items or a string of characters, or other conditions. If the item matches, the row is selected. The comparison conditions are:

Operator	Operation
AND	If one condition is true AND another condition is true, then and only then, perform the task.
OR	If either condition is true OR both are true, then perform the task.
NOT	Reverses the truth value of the search condition.
NOT takes precedence over both AND and OR. AND takes precedence over OR.	
BETWEEN	Use to test to see if a value falls BETWEEN two expressions.
IN	Use to test to see if a value is IN a list of values: IN(value1, value2, value3)
LIKE	Use to test to see if a string value is LIKE another string.
MATCHES	Use to test to see if a string value MATCHES another string.

expression **BOOLEAN OPERATOR** expression

Example: sex = "F" and birth_date > "01/01/1980"

expression **[NOT] IN** (item1, item2, item3)

Example: st in("TX", "OK")

column_name **[NOT] LIKE** "string"

column_name **[NOT] MATCHES** "string"

column_name **[NOT] NULL**

- To limit the number of records returned use the AND operator.
- To increase the number of records returned use the OR operator

Example 8: Using AND to limit values returned

This example counts all constituents in Austin in the 78703 zip code and whose ID is greater than 400000.

```
Select count(*)
From id_rec
where (city = "Austin"
and (zip >="78703" and zip < "78704"))
and id > 400000
```

Example 9: Using OR to limit values returned

This example counts all constituents in Austin or in the zip code range 78703 whose ID is greater than 400000.

```
Select count(*)
From id_rec
where (city = "Austin"
OR (zip >="78703" and zip < "78704"))
AND id > 400000
```

Example 10: Using NOT to limit values returned

This example selects all zip codes except 78703 in the city of Austin whose ID is > 400000.

```
select count(*)
from id_rec
where (city = "Austin"
      AND NOT (zip >="78703" and zip <"78704"))
      AND id > 400000
```

Example 11: Using IN

This example uses IN to return all constituents who live in Texas or Louisiana.

```
select id,
       fullname,
       st
from id_rec
where st IN ("TX", "LA")
      and id > 400000
```

Using Wildcards to Search Text

Wildcards are available for **MATCHES** and **LIKE**.

Wildcards which work with **LIKE** are:

% matches 0 or more characters
matches any single character

Wildcards which work with **MATCHES** are:

***** matches 0 or more characters
? matches any single character (except NULL)
[] matches any character of those enclosed in brackets
[^] matches any character EXCEPT those enclosed in brackets

Example 12: Using MATCHES to search for character strings

This example searches for all fullnames that begin with "James".

```
select fullname
from id_rec
where fullname matches "James*"
```

Example 13: Using Matches to search for character strings

This example searches for all records with a first name of "James".

```
select fullname
from id_rec
where fullname MATCHES "*, James"
```

Example 14: Using Matches with [] to search for character strings

This example searches for all records with a last name beginning with “A” and a first name of “James”.

```
select fullname
from id_rec
where fullname MATCHES “[A]*, James”
```

Null Values

Use the NULL comparison for fields that have no data. Check the schema to see which fields allow NULL.

Example 15: Looking for fields with null values

This example looks for all constituents who have a NULL value in the state field. This will return all non-USA addressed constituents.

```
select id,
       fullname,
       city,
       st
from id_rec
where st IS NULL
```

Join Conditions

Joins consist of setting two fields, which two tables have in common, equal to one another. In the CARS database the field **id** is the most common field which is used as it is present in almost all tables.

A rule of thumb is that you have one fewer Join conditions than you have tables listed in the FROM clause. So, if you have two tables in the From clause, you need one Join condition as part of the Where clause.

When the columns that you are joining have the same names in both tables, the column name must be preceded by the table name and a period in the Join Condition of the Where clause.

The effect of the join is to create a temporary composite table in which each pair of rows (one from each table) satisfying the join condition are linked together to form a single row.

Table Joins can be affected using a Two-Table Join, a Multiple-Table Join, a Self-Join (when the table is joined to itself for calculation) or an Outer Join.

Example 16: Using Joins for a Two-Table Join

One joining statement is used to join 2 tables. Only records with both an id_rec and a profile_rec will be returned.

```
select id_rec.id,
       id_rec.fullname,
       profile_rec.birth_date
from id_rec,
     profile_rec
where id_rec.id = profile_rec.id
      and id_rec.id > 400000
```

Example 17: A Multiple Table Join

Two joining statements are used to join 3 tables. Only records with an id_rec, and emp_rec and a se_view_rec will be returned. This will return all ids, names, logins (for email addresses) and employment information for people employed by Dell.

```
Select  id_rec.id,
        id_rec.fullname,
        emp_rec.bus,
        se_view_rec.email
from    id_rec,
        emp_rec,
        se_view_rec
where   id_rec.id = emp_rec.id
        and id_rec.id = se_view_rec.id
        and se_view_rec.run_code="STUDENT"
        and (emp_rec.end_date is null
             or emp_rec.end_date = " "
             or emp_rec.end_date >= today)
        and id_rec.id > 400000
```

TABLE JOIN RULE

The number of JOIN CONDITIONS in the WHERE clause must equal AT LEAST ONE LESS than the number of TABLES in the FROM clause. If there are 3 tables in the from clause, there must be 2 joins in the where clause.

Outer Joins

An OUTER join is a join between two tables where one of the tables may not have any corresponding data, but the data in the other table is still relevant to the query. If there is no corresponding row in the OUTER table, NULL will be entered for the value from the OUTER table. An example of where an OUTER join would be useful would be where information is being gathered on employees and their dependents. While some employees may not have dependents, the information for the employee would still be necessary and valid.

Example 18

An outer join is used to join tables which have active se_elm_recs, and prints the emp_rec bus and pos fields if present. Only records with an id_rec and a se_view_rec will be returned. This will return all ids, names, logins (for email addresses) and employment information for constituents with active logins.

```
Select  id_rec.id,
        id_rec.fullname,
        emp_rec.bus,
        se_view_rec.email
from    id_rec,
        outer emp_rec,
        se_view_rec
where   id_rec.id = emp_rec.id
        and id_rec.id = se_view_rec.id
        and se_view_rec.run_code="STUDENT"
        and (emp_rec.end_date is null
             or emp_rec.end_date = " "
             or emp_rec.end_date >= today)
```

Subquery Conditions

A subquery is a SELECT statement that is nested in the WHERE clause of another SELECT statement. The YES or NO value returned by the subquery determines whether the row is selected by the query.

The syntax for a Subquery can be any of the following:

Where "expression" "relational operator" [ALL | ANY | SOME] (Select Statement)

Where "expression" [NOT] In (Select Statement)

Where [NOT] EXISTS (Select Statement)

Using the syntax above, these keywords have the effect of:

ALL	determining if a comparison is true for every value returned.
ANY	determining whether a condition is true for at least one of the values returned.
SOME	the same as ANY.
EXISTS/IN	making the subquery true only if the nested SELECT finds at least one row.

Example 19: Using a Subquery

This selection will print the fullname and age of all constituents whose age is greater than the average age of all constituents.

```
Select  fullname,
        age
from    id_rec,
        profile_rec
where   id_rec.id = profile_rec.id
        and age >
        (select avg(age)
         from profile_rec)
```

Group By Clause

The Group By clause is used in select statements containing aggregate functions. Each item in the Group By clause produces a single row of results for that group.

If you use a GROUP BY clause, each of the column names that you use must be in the SELECT CLAUSE, unless columns are used as part of an aggregate or time expression.

Do not put constant expressions or BYTE or TEXT column expressions in the GROUP clause.

Group By Rules

A Group By clause can only be used with an Aggregate Function.

A having clause is only used with a Group By clause.

All columns in the Select Statement must be part of the Group by clause EXCEPT the column containing the aggregate.

Example 20: Using the GROUP BY clause

This query returns the number of constituents with id greater than 400000, by state.

```
select  st, count(*) number
from    id_rec
where   id > 400000
GROUP BY st
```

Having Clause

The HAVING clause usually complements a GROUP BY clause by applying one or more qualifying conditions to groups after they are formed, similar to the way the WHERE clause qualifies individual rows. This restricts the groups that are returned.

One advantage to using a HAVING clause is that you can include aggregates in the qualifying conditions whereas you cannot include aggregates in the qualifying condition of a WHERE clause.

Example 21: Using the HAVING clause

This query returns the number of constituents by state having counts greater than 10.

```
select st, count(*) number
from id_rec
where id > 400000
GROUP BY st
having count(*) > 10
```

Order By Clause

The Order By clause is used to determine the sorted order of your output.

The syntax of the Order by Statement is

ORDER BY column1 [ASC | DESC], column2 [ASC | DESC], etc.

Using the keyword **DESC** following a column name causes the retrieved data to be sorted in descending order. The default is **ASC**, and therefore does not need to be indicated.

Each column in the sort is separated by a comma. List the columns in the order in which you would like to have the sort performed.

All columns used to make up the Order By key must have been included in the Select Clause.

Example 22: Using ORDER BY clause

```
select st, count(*) number
from id_rec
where id > 400000
GROUP BY st
having count(*) > 10
order by st
```

A number (indicating the column's order in the select statement) can be used instead of the column name.

Example 23: Using ORDER BY clause with a column number. This will sort in count(*) order.

```
select st, count(*) number
from id_rec
where id > 400000
GROUP BY st
having count(*) > 10
order by 2
```

Example 24: Using ORDER BY clause with DESC

```
select st, count(*) number
from id_rec
where id > 400000
GROUP BY st
having count(*) > 10
order by 2 DESC
```

Into Temp Clause

The situation often arises where you will want to use results from one Select Statement in part of another. Remember that the Select Statement produces results in a table (rows and columns) format. Often the best way to test your Select statements is to break them down where one feeds into another. To do this, use the **Into Temp Clause**.

The syntax of the Into Temp clause is:

INTO TEMP <temp table name> WITH NO LOG

The temporary table will disappear when you exit from SQL. The WITH NO LOG is used to reduce the overhead of transaction logging since you're not really making any changes to the database that need to be tracked.

Example 25: Using the INTO TEMP Clause

This example prints id, birth_date and employer name on output. Output is sorted in birth_date order.

```
select id,
       fullname
from id_rec
where id_rec.id > 400000
into temp a with no log;

select a.id,
       a.fullname,
       birth_date
from a,
     profile_rec
where a.id = profile_rec.id
      and birth_date > "01/01/1980"
into temp b with no log;

select b.*,
       emp_rec.bus
from b,
     outer emp_rec
where b.id = emp_rec.id
order by birth_date, fullname
```

Refining the Select Statement

The select statement can use any of the arithmetic operators below to form **expressions**.

+	Addition
-	Subtraction
*	Multiplication
/	Division

Any item can be **aliased** to a more comprehensive name by typing the name of the alias immediately after the name of column (before the comma separator).

Example 26: Arithmetic operator and a Column Alias

In this example, the column alias “age” will be used for the expression “(today – birth_date)/365”. The expression (today – birth_date) returns a number that has to be divided by 365 to show age.

```
select id,  
       (today – birth_date)/365 age,  
       sex  
from   profile_rec  
where  id > 400000
```

Miscellaneous Commands

Self Joins

You can join a table to itself, creating a self-join. To do so, you must list the table names twice in the FROM clause and use table aliases to differentiate the two readings of the table. This is useful if you need to reference the same fields twice in one query. For example, printing out the fullname of the student, the student’s address and the fullname of a parent and the parent’s address.

Example 27: Using Self Joins

This query prints out the id and name of the student as well as their employer

```
Select id_rec.id,  
       id_rec.fullname stuname,  
       emp_rec.bus_id busid,  
       bus.fullname busname  
from   id_rec,  
       id_rec bus,  
       emp_rec  
where  id_rec.id = emp_rec.id  
       and emp_rec.bus_id = bus.id  
  
order by busname
```

V. ACE REPORT WRITER

OVERVIEW - ACE REPORT SECTIONS

An ACE Report Consists of the following sections. Keywords have been **bolded**. Note that the keyword *end* terminates each section.

DATABASE (required)

What database will you be working with
end

DEFINE

Define variables

end

OUTPUT

Define output page setup (if different from default).

end

SELECT (required)

Read data from database to create records that will be passed into FORMAT.

end

FORMAT (required)

Indicate how data is to be displayed.

end

DATABASE Section - Required

The database that you will be working with is defined in this section. For practical purposes at St. Edward's, the database will always be defined as CARS_DB.

```
database
    CARS_DB
end
```

Defining the CARS Database

DEFINE Section - Optional

Variables are specified in this section. The value of a variable may be set later in the report by using the keyword LET.

Variable are written in the format:

variable variable_name variable_type

*For variable types, refer to the **Reference** section of **SQL using WPVI** under **Data Type**.*

Functions may also be set in the DEFINE section. See **ACE Report Writer: Reference** for the list of the standard CARS functions that are passed in the macro REP_DEFINE.

By using REP_DEFINE, you have access to the standard CARS functions. You can also include your own variables to run the report to your specifications.

```
define
    variable name char(20)
    variable cash money
    variable counts smallint
end
```

Defining Variable Parameters

OUTPUT Section - Optional

The default ACE output places text out to 80 columns. A column in this instance is the space one character fills on the screen or printer (e.g., the number of columns the word *character* requires is 9).

In the SAMPLE report, the macro REP_OUTPUT will set your report to CARS standard output. To review these values, see *Standard Output Parameters Contained in REP_OUTPUT* in the **Reference** section.

If you wish, you can set your own margins and page length in this section.

```
output
  top margin          1
  bottom margin      1
  left margin         0
  right margin       80
  page length        24
end
```

Output Parameters for Standard Screen Output

SELECT Section - Required

The select statement is a standard SQL (Structured Query Language) statement. It may be most helpful to you to test your SQL statements to ensure that you are receiving the output you want before incorporating them into your ACE report.

Because the Format section of the report supports aggregate functions, you may also wish to not perform the aggregate functions in your select statements. For example, if you wish to see a list of the names of students, grouped and counted by city, you would perform the aggregate in the Format section. However, if you wish to see just the city and count, you could perform the aggregate in the Select section.

If you are planning on using Group breaks in the Format section, an Order By clause must be included in the last Select section.

Also note that the final Select section in an ACE Report does not have a semicolon (;) at the end of it. Instead the keyword *end* is used. If the *Into Temp* clause is used, then semicolons are placed at the end of each Select statement and the *end* keyword is used to indicate that the Select *SECTION* is complete.

FORMAT Section - *Required*

The format section uses two types of clauses: Timing clauses and Action clauses. **Action Clauses** tell ACE what to do. **Timing Clauses** tell ACE when to do it.

Action Clauses

Print

Print tells ACE to produce output. Each **print** statement produces one line of data. Separate items on a line with a comma. **Print** can be used with column name(s), arithmetic operators, constants, functions such as date and time, and graphic characters by using their ASCII values. In a character field, the number of columns used for output will be determined by the size of the field in the table. To print character text without trailing blanks, use the keyword *clipped* after the field name.

In a numeric field, the format in which the data displays can be changed by using the **using** keyword. See **Reference - Formatting Numeric Data** for information on how to change the format of numeric fields. When utilizing **Using**, the format must be placed in double quotes.

print field1	Prints the contents of field 1
print field1, 1 space, field2 prints the contents of field 2	Prints the contents of field 1, a space, then prints the contents of field 2
print field1 * field2 result	Multiplies field 1 times field 2 and prints the result
print "This report was printed on", date	Prints the text in quotes, and then the current date using the format Day Mon DD YYYY (e.g., Mon Sept 13 2002)
print time	Prints the current time using the format HH:MM:SS
Note: Values for date and time can be truncated.	For example "print time[1,5]" prints in the format HH:MM.
print ascii 33	Prints an ! (33 is the ASCII value for the exclamation mark)
print field1, 10 spaces, field2	Put 10 spaces between fields 1 and 2. Prints on the same line (unless the size of field 1 and/or 2 prohibits)
print col 10, field1, col 30, field2	Prints field 1 beginning at column 10, and field 2 beginning at column 30

print	Prints a blank line (Using Skip n lines is recommended)
print field1 clipped, field2	Removes trailing blanks from field 1
print num_field using “\$\$\$\$.&&”	Prints a floating dollar sign, and put in a decimal place

Using the PRINT Statement

Print is also used with aggregate functions. When the aggregate is not to be used on ALL of the data, the Action Clause *After Group Of* is used to separate information. See *Timing Clauses - After Group Of* for more information on grouping.

print count	Displays a total number of all rows
print group count	Displays a total number of all rows in the group
print total of field1	Displays sum of field 1
print group avg of field1	Displays average of field 1 in the group
print percent	Displays percentage of each row in the group

Print Statements Using Aggregates

Skip

The Format clause **Skip** can be used to tell ACE to leave blank lines and/or go to the top of the next page. While blank lines can be accomplished using multiple print statements, it is preferable to use **skip**. Example 8 shows the usage of the **skip** statement:

skip 5 lines	Produces 5 blank lines
skip to top of page	Produces a page break

Using SKIP

Timing Clauses

TIMING CLAUSES	
FIRST PAGE HEADER	Prints only on the first page
PAGE HEADER	Prints as a header on every page
BEFORE GROUP OF	Used on sorted data, when sort group begins
ON EVERY ROW	How to display every row of data
AFTER GROUP OF	Used on sorted data, when sort group ends
ON LAST ROW	What to do when the data ends
PAGE TRAILER	Prints as a footer on every page
ACTION CLAUSES	
PRINT	Prints item(s) then moves to next line. Multiple items on the same line are separated by a comma.
SKIP	Skips lines, or to top of page
NEED X LINES	Used when you want all lines to print on one page; keeps lines together
LET	Assigns a value to a variable
IF	Only executes if condition is true

Example : A Basic form for creating ACE Reports

```
{
This is a sample ace report for creating reports.
Modify the select and format sections to obtain the desired results.
}
database
CARS_DB
end
define
REP_DEFINE
end
output
REP_OUTPUT
end
{ Select statements go here}
{ order statements go here}
end

format
    page header
    REP_FORMAT
    REP_HEADER("ENTER REPORT TITLE")
    skip 1 line
    print "Column Headers"
    print "-----"

on every row
    print { enter fields to be printed}

on last row
    REP_LAST_REC

page trailer
    REP_TRAILER

end
```

Example 1: A simple ACE report

```
{ List of all ids and names of all constituents with an address in the Austin zip code range and id
  > 400000}
database
    CARS_DB
end
define
    REP_DEFINE
end
output
    REP_OUTPUT
end
select id_rec.id,
       fullname
from   id_rec
where  id_rec.id > 400000
       and (zip >="78701" and zip < "78799")
order by fullname
end

format
page header
REP_FORMAT
REP_HEADER("Constituents with ID's Greater than 400000 in Austin")
skip 1 line
print  col 1, "ID",
       col 10, "Name"
print  col 1, "-----",
       col 10, "-----"

on every row
print  col 1, id,
       col 10, fullname
on last row
    REP_LAST_REC
page trailer
    REP_TRAILER
end
```

Running an ACE Report

1. Access Administrative Data Base (I)
2. Select the **Utility Menu**.
3. Select **Letters/Labels/Reports**. This puts you in the correct menu to run your ACE reports. The path to Letters/Labels/Reports is **U(Utilities)/b(Letters/Labels/Reports)**.
4. Access Word Processing (i.e., Shift-W for **Word Processing**).
5. Select your **private** File Cabinet by typing in the number in front of it and pressing **<Return>**.
6. Select the **reports** drawer by pressing the number in front of it and pressing **<Return>**.
 - To create a new file:
Copy the sample file and edit your new copy. (Remember to first remove the sample SQL statement and format)
 - To edit an existing file:
Type the number in front of it and press **<Return>**.
7. Type and/or edit your ACE Report
8. Check it for typos and syntax errors (e.g., missing commas; missing *ends*).
9. Exit the editor, saving the file.
10. Use **t** to translate the file. Translating the file puts it into a format that the ace report writer can read. If there are problems with the translation, a *filename.err* file will be created. Use this file to determine the corrections that need to be made to the original file. Make the corrections in the original report file (not in the error file) and re-translate. **You must translate the file before initially running it, and again each time you make modifications!!!**
11. Press **Shift-X** to Exit Word Processing and press **<Return>**. To suspend Word Processing press **Control Z**.
12. Press **<Return>** to continue.
13. Select [e] WPVI ACE Report.
14. Type in the name of the File Cabinet and the report. Note: You do not have to type the name of the drawer. The system knows that it is in The Reports Drawer. Therefore, for this class, type *private/filename*.
15. Press **PF1**.
16. Type **file** in as the mode to print the report to the screen.
17. Press **PF1** to execute the report.
- 18.a. If the report runs correctly, you will receive an email telling you that output was created..
- 18.b. If the report does NOT run correctly, read the message(s) in email. They will give you some indication as to the problem. Then go back to step 7.b. to re-edit the report.

Example 2: Same selection criteria as Example 1. but with the addition of ID # formatting and the use of the `_full_name()` function.

```
format
page header
    REP_FORMAT
    REP_HEADER("Constituents with ID's Greater than 400000 in Austin")
skip 1 line
print col 1, "ID",
      col 10, "Name"
print col 1, "-----",
      col 10, "-----"

on every row
print col 1, id using "<<<<<<<",
      col 10, _full_name(fullname)

on last row
    REP_LAST_REC

page trailer
    REP_TRAILER

end
```

Need X Lines

Need x Lines tells ACE that the next x lines need to be printed on the same page. If there is not enough room on the page, ACE will skip to the next page before continuing.

```
need 4 lines

print name
print address1
print address2
print city, state, zip
```

Using NEED x LINES

Timing Clauses

As previously mentioned, *Timing Clauses* tell the ACE Report Writer when to perform the statements in the Action Clauses. The Timing clauses are listed below in the logical order in which they are to be used your ACE Reports.

First Page Header is used to define information that you wish to have appear on the first page *only*. If First Page Header is used in Conjunction with Page Header, the Page Header information will begin on the second page.

Page Header is used to define information that you wish to have appear at the top of every page. Such information can include the report title, page number, date and/or time printed, and column headers for print out.

```
FORMAT
page header
    print _justify("", "This is my Report", "")
    print _justify("", "Printed on ", date, time[1,5]", "")
    print _justify("", "page, rep_pageno using "###"", "")
    skip 2 lines
    print    col 10, "Name",
            col 30, "Phone No."
on every row
    print    col 10, name clipped,
            col 30, phone
end
```

Using PAGE HEADER in Format

Example 5: Using Before Group of clause to sort by employer.

Add the fields for employer information to the select statement and then sort by bus.

```
select id_rec.id,
       fullname,
       se_view_rec.line2,
       se_view_rec.city,
       se_view_rec.st,
       se_view_rec.zip
from   id_rec,
       se_view_rec
where  id_rec.id=se_view_rec.id
       and run_code="STUDENT"
       and id_rec.id > 400000
       and (se_view_rec.zip >="78701" and se_view_rec.zip < "78799")
into temp a with no log;
```

```
select a.*,
       emp_rec.bus
from   a,
       emp_rec
where  a.id = emp_rec.id
order by bus desc, fullname
end
```

```
format
page header
REP_FORMAT
REP_HEADER("Constituents with ID's Greater than 400000 in Austin")
let rep_text="Sorted by Employer in Descending Order "
print REP_JUSTIFY(" ",rep_text clipped,"")
```

```
skip 1 line
print col 1, "ID",
      col 10, "Name",
      col 45, "Address"
print col 1, "-----",
      col 10, "-----",
      col 45, "-----"
```

```
before group of bus
print col 1, bus
```

```
on every row
need 2 lines
```


After Group Of

After Group Of is used much the same way as *Before Group Of*. Use the **After Group Of** to perform aggregate functions.

```
FORMAT
before group of state
    print state
    skip 1 line
before group of city
    print city
    skip 1 line

on every row
    print name clipped, phone

after group of city
    print "The total for ", city, " is : ", group count using "####"
    skip 1 line

after group of state
    print "The percent in ", state, " is : " percent using "##.##",
    "%"

end
```

Using the AFTER GROUP OF Clause

Example 6: Adding After Group of clause allows for printing subtotals of groups. The group must be listed in the Order by statement.

```
select id_rec.id,
       fullname,
       se_view_rec.line2,
       se_view_rec.city,
       se_view_rec.st,
       se_view_rec.zip
from   id_rec,
       se_view_rec
where  id_rec.id=se_view_rec.id
       and run_code="STUDENT"
       and id_rec.id > 400000
       and (se_view_rec.zip >="78701" and se_view_rec.zip < "78799")
into temp a with no log;

select a.*,
       emp_rec.bus
```


On Last Row

Use the **On Last Row** clause to perform calculations on all of the data, or make statements that may refer to the report as a whole.

```
FORMAT
...
on last row
    print "Total Names Listed : " count using
    "#####"
end
```

Using the ON LAST ROW Clause

Example 7: Using On Last Row to get a total count of all records processed.

```
on last row
skip 2 lines
    print col 1, "Total: ",
    col 60, count using "#####"
```

The clause, **Page Trailer** allows you to put footer information in your report. Such information can include page numbers, and date and/or time printed.

```
FORMAT
...
page trailer
    print col 55, date, 3 spaces, time[1,5]
    print col 70, pageno
end
```

Using the PAGE TRAILER Clause

{Example 8 : Using After Group of and On Last Row to print subtotals and totals only.}

```
format
page header
REP_FORMAT
REP_HEADER("Counts of Constituents with ID's Greater than 400000 in Austin")
let rep_text="Total Employed by Employer"
print REP_JUSTIFY(" ",rep_text clipped,"")
skip 1 line
after group of bus
    print col 20, bus, ":",
        col 60, group count using "#####"

on last row
skip 2 lines
    print col 20, "Total: ",
        col 60, count using "#####"
page trailer
    REP_TRAILER
end
```

If Statements

Circumstances arise when an action is only performed when a condition is true. Using the **IF** statement allows selective processing of information. The **IF** statement evaluates the condition, and performs the action when the condition is true. If the condition is false, an **ELSE** statement can be included for an alternative action. If multiple statements are to be processed, the keywords **begin** and **end** tell ACE to perform all actions indicated.

```
if (salary > 30000) then
begin
    print name
    print "Director Level"
    let exec_count = exec_count + 1
end

else
begin
    print name
    print "Not Executive Material"
end
```

Using the Conditional IF/THEN/ELSE

Example 9: Adding If/Then/Else statements to output. Change join in second select statement to an outer join.

```
select id_rec.id,
       fullname,
       se_view_rec.line2,
       se_view_rec.city,
       se_view_rec.st,
       se_view_rec.zip
from   id_rec,
       se_view_rec
where  id_rec.id=se_view_rec.id
       and run_code="STUDENT"
       and id_rec.id > 400000
       and (se_view_rec.zip >="78701" and se_view_rec.zip < "78799")
into temp a with no log;

select a.*,
       emp_rec.bus
from   a,
       outer emp_rec
```

```

where a.id = emp_rec.id
order by bus, fullname
end
format
page header
REP_FORMAT
REP_HEADER("Constituents with ID's Greater than 400000 in Austin")
let rep_text="Sorted by Employer"
print REP_JUSTIFY(" ",rep_text clipped,"")

skip 1 line

after group of bus
  if bus = " " or bus is NULL
  then
    print col 1, "Total with No Employer on Record:",
      col 60, group count using "#####"
  else
    print col 1, "Total Employed by ", bus, ":",
      col 60, group count using "#####"

on last row
  skip 2 lines
  print col 20, "Total: ",
    col 60, count using "#####"

page trailer
  REP_TRAILER
end

```

Using ACE Report Output as Input to Another Application such as Excel

It is often useful to be able to output data from the CARS system in Comma Delimited Format for export to another application such as Excel. Comma Delimited Format (.csv files in Excel) is a standard format consisting of one record per row, with a comma between each field.

Example: firstname, lastname, email address

Example 10. This report creates a comma delimited file for all academic accounts with an ID > 400000. The output will include id, firstname, lastname, and email address.

{Logins and names of students in comma delimited format}

```
database CARS_DB end
```

```
define
```

```
    REP_DEFINE
```

```
end
```

```
output
```

```
    top margin    0
```

```
    bottom margin 0
```

```
    left margin   0
```

```
    right margin  100
```

```
end
```

```
select id_rec.id,
```

```
    fullname,
```

```
    email
```

```
from id_rec,
```

```
    se_view_rec
```

```
where se_view_rec.id = id_rec.id
```

```
    and run_code = "STUDENT"
```

```
    and email is not null
```

```
    and id_rec.city = "Dallas"
```

```
order by 1
```

```
end
```

```
format
```

```
on every row
```

```
print column 1, id using
```

```
"#####",",",_first_name(fullname),",",_last_name(fullname),",",email
```

```
clipped,"@acad.stedwards.edu"end
```

VI. REFERENCE

Data Type

As previously discussed, data in a database is stored in tables. Tables are set up in rows (also know as records) and columns (also known as fields). Fields are defined by the type of data stored in them. The syntax of SQL statements can vary depending on the type of data in the field. Data types include:

smallint Uses 2 bytes. Stores whole numbers between -32,767 - +32,767.

integer Uses 4 bytes. Stores whole numbers between -2,147,483,647 to +2,147,483,647.

serial Same as integer. Each new row in table will receive next serial #.

smallfloat Uses 4 bytes. Fractional number of 8 significant digits.

float Uses 8 bytes. Fractional number of 16 significant digits.

decimal Fractional number with up to 32 significant digits. Defined with a total width and digits to right of decimal. Decimal(8,4) is 8 total digits, 4 to the right of the decimal. Default is decimal(16,0).

money Stored as decimal but displayed with \$. Default is money(16,2).

date Stored as mm/dd/yyyy

char Stores character representations of data. Define size based on largest possible value to go into column.

Functions and Variables Contained in REP_DEFINE

FUNCTIONS		
FUNCTION	USE	DESCRIPTION
<code>_getcars</code>		Calls environmental variables
<code>_midstring</code>	<code>_mid_string(col)</code>	Center a piece of text
<code>_first_name</code>	<code>_first_name(fullname)</code>	Extract first name from fullname
<code>_full_name</code>	<code>_full_name(fullname)</code>	Extract name in the form "First M Last) from fullname
<code>_last_name</code>	<code>_last_name(fullname)</code>	Extract last name from fullname
<code>_dashdays</code>		For use with an array of days of the week
<code>rep_justify</code>	<code>rep_justify("","","")</code>	Justifies line; requires 3 parameters
<code>_formatrcs</code>		Formats RCS header and source lines
VARIABLES		
VARIABLE	TYPE	DESCRIPTION
<code>rep_record</code>	smallint	Used in the macros REP_LAST_REC and REP_TRAILER
<code>rep_width</code>	smallint	Used in the macro PAGE_FORMAT
<code>rep_pageno</code>	smallint	Used in the macro PAGE_FORMAT
<code>rep_skip_to_top</code>	smallint	Used in the macro REP_SKIP_TO_TOP_OF_PAGE
<code>rep_page</code>	char (8)	Used in the macro REP_FORMAT to set a page number and format
<code>rep_text</code>	char (132)	A character string that can be used as a temporary buffer

Standard Output Parameters Contained in REP_OUTPUT

STANDARD DEFINITION OF	IS
top margin	3
bottom margin	1
left margin	0
right margin	80
page length	66

Default Formatting

Data Type	Defaults As:
smallint	6 spaces. Including - if negative value
serial	11 spaces. Including - if negative value
integer	
smallfloat	14 spaces (11 before, 1 for, and 2 after the decimal point.)
float	
decimal	Number of digits plus 2 (for decimal point and sign)
money	Number of digits plus 3 (for decimal point, sign, and dollar sign)
char	As defined. (e.g., char(12) will have 12 spaces. A value having less than 12 spaces will have trailing blanks.)

Formatting Numeric Data

FORMATTING CHARACTERS FOR NUMERIC DATA				
Group	Character	Description	Example	Produces
FILL	*	Inserts a * for blank characters	1234 using “*****”	**1234
	&	Inserts a 0 for blank characters	1234 using “&&&&&&”	001234
	#	Inserts a blank for blank characters (right justifies)	1234 using “#####”	1234
	<	Left justifies	1234 using “<<<<<<<”	1234
LITERAL	,	Insert once to place comma	1234 using “##,##”	12,34
	.	Insert once to place decimal	1234 using “###.#”	123.4
SIGN	-	If specified once, then is fixed. If used as a fill character, then floats	-1234 using “-#####” -1234 using “_____”	- 1234 -1234
	+	If specified once, then is fixed. If used as a fill character, then floats	1234 using “+#####” 1234 using “+++++++”	+ 1234 +1234
	()	If specified once, then is fixed. If used as a fill character, then floats	-1234 using “(#####)” -1234 using “((((()”	(1234) (1234)
	\$	If specified once, then is fixed. If used as a fill character, then floats	1234 using “\$#####” 1234 using “\$\$\$\$\$\$\$”	\$ 1234 \$1234

Aggregate Functions

Aggregate functions work on the data from the Select section. If subtotals (sub-counts, sub-averages, etc.) are required, data must have been grouped using the Order By clause in the Select statement.

Aggregate	Description
COUNT	Prints the number of rows
MIN	Prints the minimum value in a column
MAX	Prints the maximum value in a column
PERCENT	Prints the percentage of all rows in a group
TOTAL	Prints the sum of a field in a group
AVG	Prints the average of a field in a group

SQL USING WORD PROCESSING
I. INTRODUCTION	1
What is a database?	1
Columns/Fields.....	2
How do I get information from a database?	2
How can I use SQL to get information from the CARS database?	2
II. DATA DICTIONARY MENU	3
How do I know what tables to use for my SQL statements?.....	3
What is the Data Dictionary Menu?	3
Database Files	4
Database Fields	4
Fields by File Report.....	4
Fast Schema List	4
Sample using Field by File Report.....	11
III. WORD PROCESSING	16
How do I use Word Processing?	16
SQL using Word Processing Step by Step Instructions for accessing the Private File Cabinet and the Reports Drawer.....	18
IV. THE SQL SELECT STATEMENT	19
The Select Statement.....	19
The Select Clause.....	19
The From Clause.....	20
The Where Clause.....	20
Field Types.....	21
Running SQL Reports in the CARS System	22
Fields Which Exist in More than One Table.....	23
Truncating Field Length.....	23
Functions Available in the Select Clause	23
Aggregate Functions	23
Date Functions.....	24
Using Relational Operators in the Where Clause.....	25
Comparison Condition	26
Using Wildcards to Search Text	27
Null Values	28
Join Conditions	28
Outer Joins	29
Subquery Conditions.....	30
Group By Clause.....	31
Group By Rules.....	31
Having Clause.....	31
Order By Clause.....	32
Into Temp Clause.....	33
Refining the Select Statement	34
Miscellaneous Commands	34
Self Joins	34
V. ACE REPORT WRITER.....	35
OVERVIEW - ACE REPORT SECTIONS	35
DATABASE Section - <i>Required</i>	36
DEFINE Section - <i>Optional</i>	36
OUTPUT Section - <i>Optional</i>	37
SELECT Section - <i>Required</i>	37
FORMAT Section - <i>Required</i>	38

Action Clauses.....	38
Print.....	38
Skip.....	39
Timing Clauses.....	40
Running an ACE Report	43
Need X Lines.....	44
Timing Clauses	46
Before Group Of.....	47
On Every Row.....	49
After Group Of.....	50
On Last Row	52
If Statements	54
Using ACE Report Output as Input to Another Application such as Excel.....	56
VI. REFERENCE	57
Data Type.....	57
Functions and Variables Contained in REP_DEFINE	58
Standard Output Parameters Contained in REP_OUTPUT	59
Default Formatting.....	59
Formatting Numeric Data	60
Aggregate Functions	61

Introduction to SQL and ACE Report Writing from CARS

