Simple SQL?

'SELECT' IS EASY

2010 RCRAInfo National Users Conference
Robert Reimer
Region 4
A great SELECT statement is like...
...a poem written for machines.

It must be elegant, compact, specific, artful, and robust.

And, like poetry, it can be annoyingly difficult to understand.

---

Defrauded I a Butterfly --
The lawful Heir -- for Thee.

(Emily Dickinson, #730)
FEAR NOT! If grade school children can do SQL SELECT, so can you!

SELECT [DISTINCT] columnname1 [, columnname2... ]
FROM tablename1 [, tablename2... ]
[WHERE condition1 [ and|or condition2... ] ]
[GROUP BY column-list ]
[HAVING conditions ]
[ORDER BY column-list ]

[] means the contents are optional, and

... means you can repeat this option many times
**SELECT Is Easy**

- SQL = Structured Query Language = “sequel”
- SELECT is a word in that computer language.
- So are: DISTINCT, FROM, WHERE, GROUP BY, HAVING, AND, OR, and ORDER BY
- SQL looks like English, but SQL is NOT English.
- A sentence using SELECT has a very restricted vocabulary and a very limited grammar, so you can be totally conversant pretty quickly.
- SELECT is the only part of SQL you’ll ever need with RCRAInfo.
- For our purposes, writing SELECT statements *is* writing SQL.
SELECT Is Easy

Who do we thank – blame? – for SQL?

Edgar Codd at IBM. He realized that...

• Computers hold more data than we can ever grasp at any given moment.

• Their data can be more efficiently stored if we understand the relationships between the data.

• Groups of similar data items should be stored in a row-and-column format called “tables.”

• Every table in a “tablespace” (a “database”) must somehow be relatable to at least one other table in that space.

• The best format for getting useful data out is the same format the data is already in: a results table.
A database is just a collection of several related tables.

A table is a collection of data that fit in a row-and-column format.
- Each column contains the same type of thing. e.g., a name, or a phone number, or...

A query is a question that we (poet programmers) ask the machine holding our database to answer. Our question must be specific, clear and complete.
- It can also be
  - elegant,
  - compact,
  - artful, and
  - robust.

The answer will be a new, temporary table of results.

A query starts with the word “SELECT”
SELECT Is Easy

- Once we have written a SELECT statement, how can we use it?
  - RCRAInfo’s BARRT (free)
  - Oracle’s SQL Developer (free)
  - Oracle’s Report Writer (maybe / maybe not free)
  - Quest’s TOAD ($900), or... ?

How to use BARRT: Bar My Ex View Back Con Run

From the RCRAInfo Main Menu, click on BARRT
At bottom of page, click on 'My Queries'.
In the 'Existing Queries' drop box, select 'RJZ - SNYS WITHOUT VIOLS', then click on VIEW.
At the bottom of the new page, click on 'Back to Post Queries'.
The Query should now fill the big box on screen.
Toward the bottom of this box, you can edit the State Postal Code ('NC' by default) or the START or END date for the search. Remember that dates MUST be in the format:
  'MM/DD/YYYY', including the single-quotes around them.
Once it is what you want, click 'Continue', then on the new page, click 'Run your Query'.
The Results page shows the results of your query.
You can convert this Screen output into a file by clicking on 'comma', 'semi-colon', or 'tab', then on the next page, click on "View a text file" and allowing it to download.
A 'tab' file can be opened in Excel with columns being properly separated.
You can also cut and paste the grid portion of the results page into Word or any other program.
**SELECT Is Easy**

- Three reserved words will show up in every useful SELECT ever written, namely...
  - **SELECT** ...
  - **FROM** ...
  - **WHERE** ...

- **SELECT** is followed by a list of the columns we want copied into our **results** table.

- **FROM** is followed by a list of the tables where the data we want “lives.”

- **WHERE** is followed by a list of conditional equations which
  - tell the database how the tables we are using relate to each other, or
  - tell the database how to filter out just the results we want.
SELECT Is Easy

- **Punctuation** in a SELECT statement is simple, but picky.
  - **Commas** tell the database you are giving it a list.
  - **Single quotes** identify a series of connected alphanumeric characters, aka, a ‘string’.
  - **Double quotes** are NOT the same as single quotes. They have a use, but generally avoid them.
  - **Periods** tell the database which table you want it to use to find the particular column you are seeking.
  - **Parentheses** in a Select statement tell the database to pause and do something else entirely (namely, whatever is inside the parentheses) before continuing.
SELECT Is Easy

- What does a simple SELECT query look like?
  
  - `Select (things) from (these tables) where (they fit these criteria)`

- NOTE: “Line breaks” do NOT matter in queries. Each of these examples yields identical results:

  - `Select things from tables where criteria`
  - `Select things FROM tables WHERE criteria`
  - `select things from tables where criteria`
A comma in the SELECT list tells the database that our “results table” will have multiple columns.

- A comma in the SELECT list tells the database that our “results table” will have multiple columns.

- Note there is NEVER a comma at the end of the SELECT list – the thing AFTER every item in this list is either:
  - a comma, or
  - an alias which gives that column in our results table a new column name, or
  - the word “from”.

- In this case, the results table will have two columns: one column is called “thing1” and the other is called “newName”
Select
  thing1, thing2 newName
from
tables
where
criteria

Select
  handler_id, handler_name facility
from
  hreport_univ4
where
  region = ‘04’
SELECT Is Easy

Select

handler_id, handler_name facility

from

hreport_univ4

where

region = '04'

- The FROM section lists (with commas, if more than one) all the database’s “tables” you want to examine and use.

- The list can be any comma-separated list of...
  - “Real” tables in the database
  - Reporting tables (real tables made for ease of reporting)
  - Views (artificial tables in the database)
  - Manual lists of data (in parentheses) that look like tables
  - Sub-queries (other SELECT queries, in parentheses)
Select
handler_id, handler_name facility, ...
from
hreport_univ4, aevent4
where
region = ‘04’, ...

- If the FROM section lists more than one table, and two of those tables contain a column that carries the same column name, you will confuse the database if you SELECT “that” column.

- Avoid this problem by specifying which table should provide “that” column. Do this by SELECTing the table-dot-column name:

Select
aevent4.handler_id, handler_name facility, ...
from
hreport_univ4, aevent4 ...
SELECT Is Easy

Select
handler_id, handler_name facility
from
hreport_univ4
where
region = ‘04’

- The WHERE section tells the database how you want to limit the data in your results table.

- Two kinds of limits:
  - Relational limits: how multiple “from” tables should be related to each other, and
  - Boolean limits: how my results table should be limited

- Multiple limiting rules can be listed, but they are NOT connected with commas.
  - They are connected with “and” or “or”.


SELECT Is Easy

- Relational equations: “this column in this table should be matched exactly to that column in that table.”

- Relational equations are always “equal” equations:
  - \( \text{table2.column3} = \text{table5.column1} \)
  - \( \text{hbasic.handler_id} = \text{aevent4.handler_id} \)
  - \( \text{hbasic.activity_location} = \text{aevent4.event_activity_location} \)
SELECT Is Easy

- After relating two tables, the number of rows in the result equals the number of rows in the smaller table.
- If you want to “expand” the result to include all the rows in the larger table, you need to “puff up” the smaller table by adding a (+) after its name.

**EXAMPLE**
- *Hbasic has a row in it for every single handler.*
- *Aevent4 only has rows for handlers who have had corrective action events.* *(a smaller table)*

- This next WHERE equation gives us results with only handlers who have had corrective action events:
  
  \[
  \text{hbasic.handler_id} = \text{aevent4.handler_id}
  \]

- But this WHERE equation fills our results table with every single handler. If they have had a CA event, the CA SELECT columns will have data in them. If they have no CA events, those columns will contain “null”.

  \[
  \text{hbasic.handler_id} = \text{aevent4.handler_id (+)}
  \]
**SELECT Is Easy**

- Boolean equations: “if this equation is true for this particular piece of data, then include it in my results.”

- Boolean equations don’t always have “equals” in them:
  - The “test” could be > or < (gt or lt), or...
  - <> (either less than or greater than)
  - != (not equal) NOTE: ! means “not” in many cases
  - like (which requires a % in the matching text where things do not have to match: ... like ‘CA075%’
  - between (something AND something else)
  - exists (the data is not null)
  - null (the data is empty or missing – NOTE: 0 is not null.
  - is (another kind of equal sign) or is not
  - in () where a table inside the parentheses contains at least one matching member: ... STATE in (‘AL’,‘FL’,‘GA’,‘TN’)
After your Select / From / Where you can add an “ORDER BY”

- In the ORDER BY section, you list how you want the results table sorted for you, in the order you want its columns sorted.
  - Select ... From ... Where ...
  - order by state, location_city, handler_name

You can also add COMMENTS to your select statement.

- Adding notes makes understanding the SELECT statement easier, both now and later.
- The machine ignores all such comments.
- They are NOT put into the results table.

```
SELECT -- any double dash begins a comment
  HANDLER_ID, -- comments end when the line ends
  HANDLER_NAME,
  genstatus, -- use them often!
  FULL_ENFORCEMENT
FROM...
```
Handy FUNCTIONS can change what columns look like.

- **SUBSTR(column, starting-at, total-number-of-letters)**
  - Gives you a sub-string, or portion-of-a-string-of-characters in the column.
  
  ```sql
  substr(handler_id, 1, 2)
  ```

  - Example boolean equation:
  
  ```sql
  ...where substr(handler_id, 1, 2) in
  ('AL','FL','GA','KY','MS','NC','SC','TN')
  ```

- **UPPER(column)** gives you a results column with the original column’s contents converted into uppercase.

- **LOWER(column)** turns the results column’s contents into lowercase letters.
Functions can also help us get around the problem that a column is always and forever just one “type” of data, be it text or numeric or a date or...

Function can convert the “type” of a column in the database into a new type in our results table, or they can make boolean math easier.

- **TO_NUMBER( )** converts text-numbers into math-numbers
- **TO_CHAR( )** converts dates or numbers into text
- **TO_DATE( )** converts a formatted text into an actual date

Check for evaluation dates on-or-after October 1, 2008:

\[
\text{EVAL\_START\_DATE} > \text{TO\_DATE ( '10/01/2008' , 'MM/DD/YYYY' ) - 1}
\]
EXAMPLE: How to check for evaluation dates within the past 5 years?

Idea:
where \( \left( \text{today’s date} \right) - \left( \text{eval date} \right) \) \( \leq \) \( \text{(five years)} \)

Need:
1. today’s date, but as a number... that is, today’s julian date
2. eval date, also as a julian number
3. five years, also as a number... \( 365.25 \times 5 = 1826.25 = 1827 \)

So...
\[
\text{where (}
\text{to\_number( to\_char( \text{sysdate, 'J'}) ) -}
\text{to\_number( to\_char( \text{EVAL\_START\_DATE, 'J'}) )}
\text{) } \leq 1827
\]
The DECODE function lets you convert the contents of a column into completely different values, depending on what the original content is.

- decode( column_to_decode, contentMatch1 , substitute1 , contentMatch2 , substitute2 , ... , defaultSubstitute )

EXAMPLE: Corrective action events codes include: CA075HI, CA075ME... Suppose I wanted to sort my results table by these rankings, in a way that makes it clear to the results user.

```
SELECT Is Easy

DECODE( SUBSTR( event_code, 6, 2),
      'HI', '4-HIGH',
      'ME', '3-MEDIUM',
      'LO', '2-LOW',
      '1-UNRANKED') ranking

ORDER BY ranking
```
SELECT Is Easy

- **Concatenation** – joining two texts together to make one

- It is like pasting them end-to-end into one results column.

- **Double-pipe** (pipe is the vertical bar above back-slash \ ) is used to concat things, positioned between two text items:

  `LOCATION_STREET_NO || ' ' || LOCATION_STREET1`

- Note that the middle of this is: quote / space / quote
  - I can insert text (even if it is just a space) into my results column!
You can have the machine scan your results table and give you summaries of specified columns:

- \( \text{max( actual\_date ) newest\_actual\_date} \)
- \( \text{min( actual\_date ) oldest\_actual\_date} \)
- \( \text{sum( received\_TONS) sum\_of\_Tons} \)
- \( \text{count( unique handler\_id ) num\_of\_handlers} \)
- \( \text{avg( received\_tons) average\_no\_of\_tons} \)

These functions require you to add a line at the END of your select statement, telling the machine in which groups of data it should find your max (or min, or sum, or...):

- `group by` ...

**EXAMPLE:** -- when was the most recent CA725 at facilities in GA?

```sql
select handler_id, max(actual_date) from corrective_action_view
where ca_event like 'CA725%'
and state = 'GA'
and group by handler_id
```
SELECT Is Easy

- You might still want to limit what displays in your results table after “aggregating” it with one of the “group by” functions.

- Do this by adding a HAVING section after the GROUP BY.

- A HAVING section reads just like a boolean WHERE section, but the WHERE does its work before the GROUP BY, and a HAVING does its work after.

EXAMPLE:

```sql
SELECT state, sum(tons_generated) total_tons
FROM br_reporting
WHERE state in ('AL','FL','GA','KY','MS','NC','SC','TN')
    and report_cycle = '2007'
GROUP BY state
HAVING total_tons > 100,000
```
**SELECT Is Easy**

Three Places to find Useful Table Information in RCRAInfo

1. Main Menu --> News Alerts --> Documentation --> Current Structure Charts (in Miscellaneous column)

2. While on the "input" page for an interesting report, click on Help (the logo). The help screen that opens gives you every detail about that report’s SELECT logic.

3. Click on Help, then click on the "Technical Help" tab along the top, then on "Data Element Dictionary" in the left column, then on "DED Reports" underneath that, then on "Reporting Tables" beneath that, then select a reporting table, select the report type "Column Information report", then click GO. Save the resulting .pdf to your hard drive for future reference.
Useful RCRAInfo code snippet #1: GROUP OF IDS

This “where” lets us limit our select to just the handlers who are members of the “Group of IDs” named 'BURKS R4 CERCLA OFFSITE', created by the user ‘RJZ’

... where
  handler_id in
    ( select handler_id  -- create a sub-query of ids
        from hid_groups  -- from this RCRAInfo table
        where user_id = 'RJZ'  -- that are in this group...
        and group_name = 'BURKS R4 CERCLA OFFSITE'
        and active_status = 'Y')  -- and the group is active

...
Useful RCRAInfo code snippet #2: SUBQUERIES

This “from/where” lets us select from all permitting data concerning final determinations for operating permits, post-closure permits, or emergency permits at all federal facilities.

Select ...

... from

  permitting_view

where

  (permit_event like 'OP200%' or -- final determination op permit
   permit_event like 'PC200%' or -- final determined post-clos
   permit_event like 'EP200%' -- final determined emergcy
  ) and

  actual_date is not null -- that actually happened

  and handler_id in -- and only get handlers who...

  (select handler_id
   from hreport_univ3
   where owner_type = 'F') -- are federal facilities

...
Useful RCRAInfo code snippet #3: COUNTING GENERATORS

Count up how many different-status generators there are every state and report the results in a useful fashion...

```sql
select location_state,
    count(decode(fed_waste_generator,'3','X', null)) CESQG,
    count(decode(fed_waste_generator,'2','X', null)) SQG,
    count(decode(fed_waste_generator,'1','X', null)) LQG
from hreport_univ4
group by location_state
```

The results table has four columns, namely:

- location_state, CESQG, SQG, LQG

The table is filled with each state’s postal code, followed by its number of each of the three types of generators.
SELECT Is Easy
Questions?