Finding those Pesky Data Issues

Tyler Hawkes

Business Intelligence Engineer

Session Stuff

Questions?

- Interrupt Me!

Am I saying it wrong?

- Set the record straight!

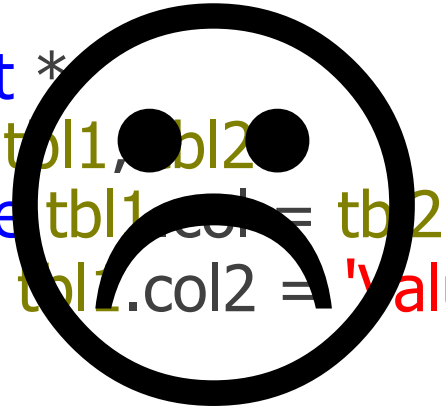
Some Assumptions

- High Performance
- No create privileges
- Imperfect transactional system
- Odd business requirements
- Examples

A thought about ANSI Standards

ANSI 89

```
select *
from tbl1, tbl2
where tbl1.col1 = tbl2.col(+)
  and tbl1.col2 = 'Value'
```



- Set Theory?

ANSI 92 or later

```
select *
from tbl1
  left join tbl2
    on tbl1.col1 = tbl2.col1
where tbl1.col2 = 'Value'
```

- Atomic clauses
- Plain English
- Portable
- Full joins

Why I love the “With” Clause

- Small executable units
- More join control
- Executed once
- Re-use of data and code
- Forces naming of data sets
- Extends Analytics

Finding rows causing conversion errors

```
select val + col
from tbl
```

```
select val, col
from tbl
where regexp_like(val,'[^[:digit:]]')
    or regexp_like(col,'[^[:digit:]]')
```

--OR--

```
select val, col
from tbl
where regexp_like(val||col,'[^[:digit:]]')
```

Connect by what?
Create your own data

Finding the next Friday, Jan. 4th

```
with ilv as (
select trunc(sysdate) + level dt
from dual
connect by level <= 10000
)
select dt
from ilv
where to_char(dt,'Mon dd fmDay') = 'Jan 04 Friday'
```

DT
1/4/2013
1/4/2019
1/4/2030
1/4/2036

Connect by what?
Create your own data

Translating days to business days

```
with date_range as (
  select to_date('20091231','yyyymmdd') + level dt
  from dual
  connect by level <= 10000
)
select dt
, case when to_char(dt,'d') in (1,7)
    then trunc(dt + 1,'d') + 1
  else dt end business_day
from date_range
```

Connect by what?
Looping through data

Finding unique punctuation

```
with punct as (  --return a single row of punctuation values
select to_char(wm_concat(distinct
  regexp_replace(order_id||invoice_id,'[^[:punct:]]'))) val
from
  ... tables
)
```

VAL
"#,#&,#,&,#...#.,&,#,-,#-,#-&,#--,&,(...-,-- ..././/.//$.../$.:.::.<,?,\,V,_,`

Connect by what?
Looping through data

Finding unique punctuation

```
, all_values as (--copies punct.val into as
                 --many rows as punct.val is long
select punct.val
, level row_number
from punct
connect by level <= length(punct.val)
```

VAL	ROW_NUMBER
"#,#&,#,&,#...#,.&,#,-,#-,#-&,#--,&,(...-,--.../,//,//$.../$..:,::,<,?,\,\/,_,`	1
"#,#&,#,&,#...#,.&,#,-,#-,#-&,#--,&,(...-,--.../,//,//$.../$..:,::,<,?,\,\/,_,`	2
"#,#&,#,&,#...#,.&,#,-,#-,#-&,#--,&,(...-,--.../,//,//$.../$..:,::,<,?,\,\/,_,`	3

Connect by what?
Looping through data

Finding unique punctuation

```
--returns a distinct list of punctuation used
select distinct
substr(val, row_number, 1) punct
from all_values;
```

PUNCT
:
<
"
\
/
?
_
.
-
(
.
$
#
&
`

Connect by what?
Looping through data

Finding unique punctuation

```
with punct as (  --return a single row of punctuation values
select to_char(wm_concat(distinct
  regexp_replace(order_id||invoice_id,'[^[:punct:]]'))) val
from
  ... tables
)
, all_values as (--copies punct.val into as many rows as punct.val is long
select punct.val
, level row_number
from punct
connect by level <= length(punct.val)
)
--returns a distinct list of punctuation used
select distinct substr(val, row_number, 1) punct
from all_values;
```

PUNCT
:
<
-
\
/
?
_
.
-
(
.
$
#
&
`

Work Smarter not Harder Leveraging Analytics

- No Group By
- No Self-Joins
- Improve Performance
- Simplify SQL
- Allow Complex Logic
- Extensible

Analytic Favorites

- row_number, sum, count, and lag

```
select level
, trunc((level- 1)/6) div_partition
, row_number () over (partition by trunc((level- 1)/6) order by level) row_number
, sum(level) over (partition by trunc((level- 1)/6)) sum
, sum(level) over (partition by trunc((level- 1)/6) order by level) running_sum
, mod(level - 1, 5) mod_partition
, count(*) over (partition by mod(level - 1, 5)) cnt
, lag(level) over (partition by mod(level - 1, 5) order by level) lag
from dual
connect by level <= 12
order by level
```

Oracle's Analytic Template and a complete list of analytic functions can be found here

Analytic Favorites Results

LEVEL	DIV_PARTITION	ROW_NUMBER	SUM	RUNNING_SUM	MOD_PARTITION	CNT	LAG
1	0	1	21	1	0	3	
2	0	2	21	3	1	3	
3	0	3	21	6	2	2	
4	0	4	21	10	3	2	
5	0	5	21	15	4	2	
6	0	6	21	21	0	3	1
7	1	1	57	7	1	3	2
8	1	2	57	15	2	2	3
9	1	3	57	24	3	2	4
10	1	4	57	34	4	2	5
11	1	5	57	45	0	3	6
12	1	6	57	57	1	3	7

Analytic Example #1 : Finding unique violations and their culprit

The Statement

```
select ci.course_instance
, i.individual
, d.dt
, mi.mapped_individual
, beg_it.beg_ind_type
, it.ind_type
, end_it.end_ind_type
, f.enrolled_cnt
, f.prsnt_cnt
, f.not_prsnt_cnt
, f.unenrolled_cnt
from
  ... tables
```

The Hard Way - Poor performance

```
with ilv as (
  select (<- The Statement)
)
, grp as (
  select course_instance, individual, dt
  from ilv
  group by course_instance, individual, dt
  having count(*) > 1
)
select *
from ilv
  inner join grp
    using (course_instance, individual, dt)
```

Analytic Example #1 : Finding unique violations and their culprit

The Easy Way

```
with ilv as (
select (<- The Statement)
, count(*) over (partition by d.dt,
   ci.course_instance, i.individual)
cnt
from
  ... tables
)
select *
from ilv
where cnt > 1
order by cnt desc, 1, 2, 3
```

The Template

```
with ilv as (
select natural_key
, ... other columns
, count(*) over (partition by natural_key) cnt
from
  ... tables
)
select *
from ilv
where cnt > 1
order by cnt desc, natural_key
```

Analytic Example #2 : Resolving *-1 relationships down to 1-1

The Hard Way – Cost: 856,038

```
with addr as (
  select a.*
  from customer_address a
  where addr.eff_dt = (
    select max(eff_dt)
    from customer_addresses a
    where addr.customer_id = a.customer_id
      and addr.seq = a.seq
  )
)
select *
from customers c
  left join addr
    on c.customer_id = addr.customer_id
    and c.primary_address_seq = addr.seq
```

The Easy Way – Cost: 11,716

```
with addr as (
  select a.*
   , row_number() over (partition by customer_id
     , seq order by eff_dt desc) row_number
from customer_address a
)
select *
from customers c
  left join addr
    on c.customer_id = addr.customer_id
    and c.primary_address_seq = addr.seq
    and addr.row_number = 1
```

Analytic Example #2 : Resolving *-1 relationships down to 1-1

The Template

```
with ilv as (
  select t.*
  , row_number() over (partition by joining_columns
    order by chronological_or_value desc) row_number
from many_tbl t --child table
)
select *
from one_tbl o --parent table
  left join ilv
    on ilv.joining_columns= o.joining_columns
    and ilv.row_number = 1
```

Analytic Example #3 : Looking for uniqueness – with a group by

The Hard Way

```
with ilv as (
select distinct region
, order_id
, app_src
from order_lines
)
select region
, order_id
, count(*)
from ilv
group by region
, order_id
having count(*) > 1
```

The Easy Way

```
select region
, order_id
, app_src
, count(*) group_row_cnt
, count(*) over (partition by region
  , order_id) cnt_distinct_group
, sum(count(*)) over (partition by region
  , order_id) total_partition_row_cnt
from order_lines
group by region
, order_id
, app_src
order by cnt_distinct_group desc, 1, 2
```

Analytic Example #3 : Looking for uniqueness – with a group by

The Template

```
select group_by_columns
, count(*) group_row_cnt
, count(*) over (partition by parent_natural_key)
    cnt_distinct_group
, sum(count(*)) over (partition by parent_natural_key)
    total_partition_row_cnt
from tbl t
group by group_by_columns
order by cnt_distinct_group desc, parent_natural_key
```

Analytic Example #4 : Ensuring no duplicate values

The Hard Way – 14 sec

```
select case when c.contract_id =
    (select max(contract_id)
     from contract ct
     where ct.vendor = c.vendor
      and f.dt between c.start and c.end)
  then f.spend end spend
, c.pct * f.spend as savings
from facts f
  left join contract c
    on f.vendor = c.vendor
    and f.dt between c.start and c.end
```

The Easy Way – 300 ms

```
with ilv as (
select f.spend
, c.pct * f.spend as savings
, row_number() over (partition by f.rowid
    order by c.contract_id desc) row_number
from facts f
  left join contract c
    on f.vendor = c.vendor
    and f.dt between c.start and c.end
)
select case when row_number = 1
  then spend end spend
, savings
from ilv
```

Fixing data with “smart” joins

- “Unknown” Customer Types
- Evaluated at runtime
- Inexpensive

Fixing data with “smart” joins

Before:

```
select f.*
, nvl(beg_ct, -1) beginning_customer_type
from fact f
  inner join customers c
    on f.customer_key = c.customer_key
  left join customer_hist beg_cc
    on c.customer_id = beg_cc.customer_id
    and quarter_start_date between beg_cc.eff_start_date and beg_cc.eff_end_date
  left join customer_type_dim beg_ct
    on nvl(beg_cc.married, 0) = beg_ct.married
    and nvl(beg_cc.years_in_college, 0) = beg_ct.years_in_college
    and case when c.rewards_member_date <= quarter_start_date
            then 1 else 0 end = beg_ct.rewards_member
    and trunc(months_between(quarter_start_date, c.birth_date) / 12) = beg_ct.age
    and c.member_type = beg_ct.member_type;
```

Fixing data with “smart” joins

Before:

```
left join customer_type_dim beg_ct
    on nvl(beg_cc.married, 0) = beg_ct.married
    and nvl(beg_cc.years_in_college, 0) = beg_ct.years_in_college
    and case when c.rewards_member_date <= quarter_start_date
            then 1 else 0 end = beg_ct.rewards_member
    and trunc(months_between(quarter_start_date,
            c.birth_date) / 12) = beg_ct.age
    and c.member_type = beg_ct.member_type
```

Fixing data with “smart” joins

```
left join customer_type_dim beg_ct
    on nvl(beg_cc.married, 0) = beg_ct.married
    and least(nvl(beg_cc.years_in_college, 0), 8) = beg_ct.years_in_college
    and case when c.rewards_member_date <= quarter_start_date
            then 1 else 0 end = beg_ct.rewards_member
    and case when trunc(months_between(quarter_start_date,
             c.birth_date) / 12) between 18 and 100
            then trunc(months_between(quarter_start_date, c.birth_date) / 12)
            else -1 end = nvl(beg_ct.age,-1)
    and case when trunc(months_between(quarter_start_date,
               c.rewards_member_date)/12)>5
               or c.member_type = 'Gold' then 'Gold'
             when trunc(months_between(quarter_start_date,
               c.rewards_member_date)/12)>10
               or c.member_type = 'Platinum' then 'Platinum'
             else 'Regular' end = beg_ct.member_type;
```

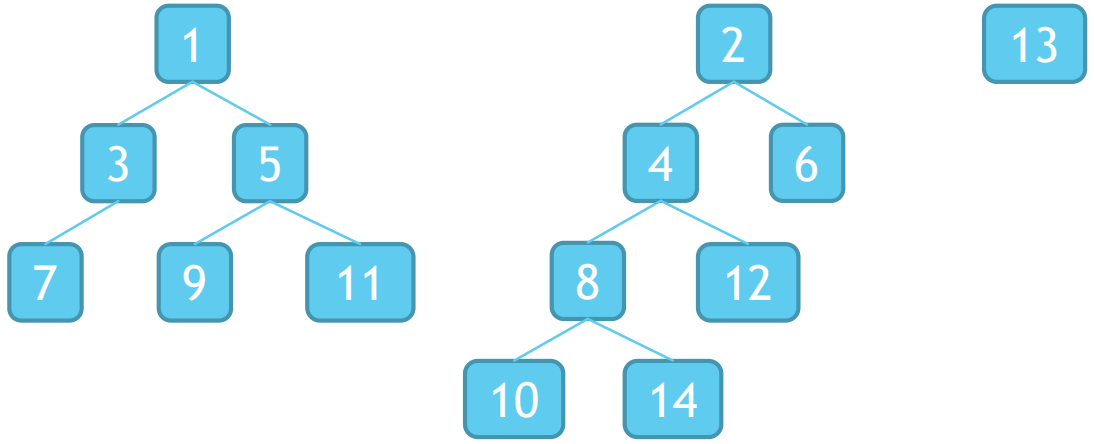
Fixing data with “smart” joins

```
select f.*
, nvl(beg_ct, -1) beginning_customer_type
from fact f
  inner join customers c
    on f.customer_key = c.customer_key
  left join customer_hist beg_cc
    on c.customer_id = beg_cc.customer_id
    and quarter_start_date between beg_cc.eff_start_date and beg_cc.eff_end_date
  left join customer_type_dim beg_ct
    on nvl(beg_cc.married, 0) = beg_ct.married
    and least(nvl(beg_cc.years_in_college, 0), 8) = beg_ct.years_in_college
    and case when c.rewards_member_date <= quarter_start_date then 1 else 0 end = beg_ct.rewards_member
    and case when trunc(months_between(quarter_start_date, c.birth_date) / 12) between 18 and 100
      then trunc(months_between(quarter_start_date, c.birth_date) / 12) else -1 end = nvl(beg_ct.age,-1)
    and case when trunc(months_between(quarter_start_date, c.rewards_member_date)/12)>5
              or c.member_type = 'Gold' then 'Gold'
            when trunc(months_between(quarter_start_date, c.rewards_member_date)/12)>10
              or c.member_type = 'Platinum' then 'Platinum'
            else 'Regular' end = beg_ct.member_type;
```

Dealing with ragged hierarchies

Naturally Occurring Hierarchies

Horizontal & Vertical



Dealing with ragged hierarchies

Prep Script

```
create table ragged_hier (
  child_key int,
  parent_key int,
  constraint pk_ragged_hier primary key (child_key)
);

begin
insert into ragged_hier values (1, null);
insert into ragged_hier values (2, null);
insert into ragged_hier values (3, 1);
insert into ragged_hier values (4, 2);
insert into ragged_hier values (5, 1);
insert into ragged_hier values (6, 2);
insert into ragged_hier values (7, 3);
insert into ragged_hier values (8, 4);
insert into ragged_hier values (9, 5);
insert into ragged_hier values (10, 8);
insert into ragged_hier values (11, 5);
insert into ragged_hier values (12, 4);
insert into ragged_hier values (13, null);  -- orphan
insert into ragged_hier values (14, 8);
commit;
end;
```

Table Data

CHILD_KEY	PARENT_KEY
1	
2	
3	1
4	2
5	1
6	2
7	3
8	4
9	5
10	8
11	5
12	4
13	
14	8

Dealing with ragged hierarchies
Get Oracle to do the heavy lifting

```
with vert_hier as (--Get all parent child relationships and level 0 (self) relationships
select connect_by_root child_key as child_key
, parent_key
, level as level_relationship
, connect_by_root child_key || sys_connect_by_path(parent_key, ' -> ') as path
from ragged_hier --add a previous in-line view called ragged_hier having child and parent keys
where parent_key is not null
connect by nocycle child_key = prior parent_key
union all --The union produces all level zero rows, which is where a record equals itself
select child_key --dimension surrogate key
, child_key
, 0 as level_relationship
, to_char(child_key,'fm999999999999') as path
from ragged_hier --Get all rows from the dimension table
)
```

Dealing with ragged hierarchies
Get Oracle to do the heavy lifting

```
, child_leaf as (  --gets all child keys that do not exist as a parent, which are leafs
select child_key
from ragged_hier c
where not exists (
   select 1
   from vert_hier vh
   where level_relationship > 0 --get rid of all self referencing rows
     and c.child_key = vh.parent_key -- check to make sure the child_key never exists in the parent column
 )
)
, parent_root as (  --gets all parent keys that do not exist as a child, which are roots
select distinct child_key as parent_key --must select the primary key
from ragged_hier p
where not exists (
   select 1
   from vert_hier vh
   where level_relationship > 0  -- get rid of all self referencing rows
     and p.child_key = vh.child_key-- check to make sure the parent_key never exists in the child column
 )
)
, leaf_root as (--join parent_root and child_leaf to determine indicators
select vh.*
, case when c.child_key is null then 0 else 1 end as child_is_leaf
, case when p.parent_key is null then 0 else 1 end as parent_is_root
from vert_hier vh
  left join child_leaf c
    on vh.child_key = c.child_key
  left join parent_root p
    on vh.parent_key = p.parent_key
)
, master as (--get master keys for each child. /*this and the final query can be skipped if masters are not needed*/
select lr.child_key
, lr.parent_key as master_key
from leaf_root lr
where lr.parent_is_root = 1
)
select lr.child_key
, lr.parent_key
, m.master_key
, lr.level_relationship
, lr.child_is_leaf
, lr.parent_is_root
, lr.path
from master m
  inner join leaf_root lr
    on lr.child_key = m.child_key
order by master_key desc, parent_key, level_relationship, child_key;
```

This is the rest of the code to produce a complete vertical hierarchy including masters. Path isn't needed, but is useful for debugging.

Dealing with ragged hierarchies
Get Oracle to do the heavy lifting

CHILD_KEY	PARENT_KEY	MASTER_KEY	LEVEL_RELATIONSHIP	CHILD_IS_LEAF	PARENT_IS_ROOT	PATH
13	13	13	0	1	1	13
2	2	2	0	0	1	2
4	2	2	1	0	1	4 -> 2
6	2	2	1	1	1	6 -> 2
8	2	2	2	0	1	8 -> 4 -> 2
12	2	2	2	1	1	12 -> 4 -> 2
10	2	2	3	1	1	10 -> 8 -> 4 -> 2
14	2	2	3	1	1	14 -> 8 -> 4 -> 2
4	4	2	0	0	0	4
8	4	2	1	0	0	8 -> 4
12	4	2	1	1	0	12 -> 4
10	4	2	2	1	0	10 -> 8 -> 4
14	4	2	2	1	0	14 -> 8 -> 4

CHILD	PARENT	MASTER	LEVEL	LEAF	ROOT	PATH
14	4	2	2	1	0	14 -> 8 -> 4

1	1	1	0	0	1	1
3	1	1	1	0	1	3 -> 1
5	1	1	1	0	1	5 -> 1

Chronological Groups – Handling Special Group By Cases

This SQL will not work

```
select min(dt) eff_start_date
, max(dt) eff_end_date
, customer
, married
, years_in_college
from customer_hist
group by customer
, married
, years_in_college
```

Raw data

CUSTOMER	MARRIED	YEARS_IN_COLLEGE	DT
X	0	4	1/1/1999
X	1	4	1/1/2000
X	1	4	1/1/2003
X	1	4	1/1/2005
X	1	4	1/1/2008
X	0	4	1/1/2010

Target Data

EFF_START_DATE	EFF_END_DATE	CUSTOMER	MARRIED	YEARS_IN_COLLEGE
1/1/1999	1/1/2000	X	0	4
1/1/2000	1/1/2010	X	1	4
1/1/2010	12/31/9999	X	0	4

Chronological Groups – Handling Special Group By Cases

```
with denserank as (
select ch.*
, dense_rank() over (partition by customer
    order by married, years_in_college) dense_rank
from customer_hist ch
)
```

Chronological Groups – Handling Special Group By Cases

CUST	MARRIED	COLLEGE	DT	DENSE
X	0	4	1/1/1999	1
X	1	4	1/1/2000	2
X	1	4	1/1/2003	2
X	1	4	1/1/2005	2
X	1	4	1/1/2008	2
X	0	4	1/1/2010	1

Chronological Groups – Handling Special Group By Cases

```
, lag_dense as (
select denserank.*
, nvl(lag(dense_rank) over (partition by customer
    order by dt), 0) lag_dense_rank
from denserank
)
```

Chronological Groups – Handling Special Group By Cases

CUST	MARRIED	COLLEGE	DT	DENSE	LAG
X	0	4	1/1/1999	1	0
X	1	4	1/1/2000	2	1
X	1	4	1/1/2003	2	2
X	1	4	1/1/2005	2	2
X	1	4	1/1/2008	2	2
X	0	4	1/1/2010	1	2

Chronological Groups – Handling Special Group By Cases

```
, chg as (
select lag_dense.*
, sum(case when dense_rank <>
    lag_dense_rank then 1 end)
    over (partition by customer
    order by dt) sum_dense_rank_change
from lag_dense
)
```

Chronological Groups – Handling Special Group By Cases

CUST	MARRIED	COLLEGE	DT	DENSE	LAG	SUM
X	0	4	1/1/1999	1	0	1
X	1	4	1/1/2000	2	1	2
X	1	4	1/1/2003	2	2	2
X	1	4	1/1/2005	2	2	2
X	1	4	1/1/2008	2	2	2
X	0	4	1/1/2010	1	2	3

Chronological Groups – Handling Special Group By Cases

```
select min(dt) eff_start_date
, nvl(lead(min(dt)) over (partition by customer order by min(dt))
    , to_date('99991231','yyyymmdd'))  eff_end_date
, customer
, max(married) married
, max(years_in_college) years_in_college
from chg
group by customer
, sum_dense_rank_change
```

Chronological Groups – Handling Special Group By Cases

EFF_START_DATE	EFF_END_DATE	CUSTOMER	MARRIED	YEARS_IN_COLLEGE
1/1/1999	1/1/2000	X	0	4
1/1/2000	1/1/2010	X	1	4
1/1/2010	12/31/9999	X	0	4

Doing your own Type 2 changes – How hard could it be?

```
with current_changes as (  --Get only the rows that have changed
 select customer
 , married
 , years_in_college
 from ext_customer_hist
 minus
 select customer
 , married
 , years_in_college
 from customer_hist
 where eff_end_date = to_date('99991231','yyyymmdd')
   and customer in (select customer from ext_customer_hist)
)
```

Doing your own Type 2 changes – How hard could it be?

```
select case when not exists (
       select 1 from customer_hist ch
       where ch.customer = cc.customer)
     then to_date('00010101','yyyymmdd')
     else sysdate
   end eff_start_date --Year 1 for new customers
, to_date('99991231','yyyymmdd') eff_end_date
, cc.*
, null row_id
from current_changes cc
```

Doing your own Type 2 changes – How hard could it be?

```
union all
  select eff_start_date
  , sysdate - interval '1' second eff_end_date
  , customer
  , married
  , years_in_college
  , rowid row_id
  from customer_hist
  where eff_end_date = to_date('99991231','yyyymmdd')
    and customer in (select customer from current_changes)
  order by 3, 1
```

Doing your own Type 2 changes – How hard could it be?

```
merge into customer_hist t -- target
using (/*All that previous SQL here (3 slides)*/) s -- source
on (t.rowid = s.row_id) --merge the data
when matched then
update set t.eff_end_date = s.eff_end_date
when not matched then
insert (eff_start_date, eff_end_date
   , customer, married, years_in_college)
values (s.eff_start_date, s.eff_end_date
, s.customer, s.married, s.years_in_college);
```

Conclusions

- In-Line Views
- Connect By
- Analytics
- Data issues
- SQL Transformations