Ch. 11: Grouping Things Together

- ANSI standard SQL Group functions:
  - AVG, COUNT, MAX, MIN, STDDEV, SUM, VARIANCE
- Others:
  - GROUPING (used with CUBE and ROLLUP, see Ch.13)
  - 9i statistical functions: CORR, COVAR_POP, COVAR_SAMP, CUME_DIST, DENSE_RANK, FIRST, FIRST_VALUE, GROUP_ID, GROUPING_ID, LAST, PERCENTILE_CONT, PERCENTILE_DISC, PERCENT_RANK, RANK, REGR, STDDEV_POP, STDDEV_SAMP, VAR_POP, VAR_SAMP
  - Obsolete? In 9i: GLB, LUB

Ch. 11: Grouping Things Together

- A group function returns a single row for an entire group of queried rows.
- NULL values are ignored in group functions.
- Without a GROUP BY clause the data returned for a group function is one row for the entire table.
- With a GROUP BY clause, one row is returned for each group your data is subdivided into.
- The WHERE clause is evaluated before data are grouped in GROUP BY clause.
- The HAVING clause is similar to the WHERE clause, but it works on entire groups of data.
- DISTINCT keyword forces only unique occurrences to be considered in data groups.
- When using a group by clause, these are the only selectable items:
  - constants,
  - group functions, and
  - group expressions -- these must match exactly the group by clause

Ch. 11: Grouping Things Together

- Order of clause execution within select statements:
  - If the statement contains a WHERE clause, removes all rows that do not satisfy it.
  - Group rows together based on the GROUP BY clause.
  - Calculate the results of the group functions for each group.
  - Choose and eliminate groups based on the HAVING clause.
  - Order the results based on the ORDER BY clause.

- Specify the GROUP BY and HAVING clauses after the WHERE clause. If both the GROUP BY and HAVING clauses are specified, they can appear in either order.

- A side effect of GROUP BY is a ordered result, same effect as having an ORDER BY clause of the GROUP BY columns.
Ch. 11: Grouping Things Together

- Views over groups, commonly used for percent of total calculations.

- Inline view example from p. 214:
  ```sql
  select categoryname, counter,
  (counter/bookcount)*100 "pct"
  from category_count,
  (select count(*) as bookcount from bookshelf)
  order by categoryname;
  ```

- Group expressions can be used in the HAVING clause, even those not used in the SELECT clause.

Ch. 12: When One Query Depends upon Another

- Equi-Joins vs. Correlated Subqueries
- Exists and IN Subqueries
- Outer Join, 8i and 9i syntax
- NOT IN replaced by Outer Join and NOT EXISTS
- NATURAL joins, INNER joins
- UNION, INTERSECT, MINUS

Ch. 12: Equi-joins vs. Correlated Subqueries

```sql
SELECT W.Name, W.Lodging
FROM Worker W,
Workerskill WS,
Lodging L
WHERE W.Name = WS.Name
  and W.Lodging = L.Lodging
  and Skill = 'COMBINE DRIVER'
  and Address LIKE '%EDMESTON%';

SELECT Name, Lodging
FROM Worker
WHERE Name IN
  (select Name FROM Workerskill
   where Skill = 'COMBINE DRIVER'
   and Lodging IN
    (select Lodging FROM Lodging
     WHERE Address
     LIKE '%EDMESTON%'));
```
Ch. 12: Another Correlated Subquery Example

The following statement returns data about employees whose salaries exceed the averages for their departments. The following statement assigns an alias to EMP, the table containing the salary information, and then uses the alias in a correlated subquery:

```
SELECT deptno, ename, sal
FROM emp x
WHERE sal >
    (SELECT AVG(sal)
     FROM emp
     WHERE x.deptno = deptno)
ORDER BY deptno
```

For each row of the EMP table, the parent query uses the correlated subquery to compute the average salary for members of the same department. The correlated subquery performs these steps for each row of the EMP table:
1. The DEPTNO of the row is determined.
2. The DEPTNO is then used to evaluate the parent query.
3. If that row’s salary is greater than the average salary for that row’s department, then the row is returned.

The subquery is evaluated once for each row of the EMP table.

Ch. 12: Exists and IN Subqueries

The EXISTS keyword is similar IN. EXISTS tests for the existence of any row. Unlike IN however, EXISTS does not match columns. Many times it is used with a correlated subquery.

Ch. 12: Outer Join, Oracle 8i syntax

The outer join extends the result of a simple join. An outer join returns all rows that satisfy the join condition and those rows from one table for which no rows from the other satisfy the join condition. Such rows are not returned by a simple join.

To write a query that performs an outer join of tables A and B and returns all rows from A, apply the outer join operator (+) to all columns of B in the join condition.

For all rows in A that have no matching rows in B, Oracle returns NULL for any select list expressions containing columns of B.

This is the basic syntax of an outer join of two tables:

```
SELECT table1.column
FROM table1, table2
WHERE table1.column = table2.column(+)
```
Ch. 12: Outer Join Rules

- The (+) operator can only appear in the WHERE clause, not in the select list, and can only be applied to a column of a table or view.
- If A and B are joined by multiple join conditions, the (+) operator must be used in all of these conditions.
- The (+) operator can only be applied to a column, rather than to an arbitrary expression, although an arbitrary expression can contain a column marked with the (+) operator.
- A condition containing the (+) operator cannot be combined with another condition using the OR logical operator.
- A condition cannot use the IN comparison operator to compare a column marked with the (+) operator to another expression.
- A condition cannot compare a column marked with the (+) operator to a subquery.
- If the WHERE clause contains a condition that compares a column from table B to a constant, the (+) operator must be applied to the column so that the rows from table A for which Oracle has generated NULLs for this column are returned.
- In a query that performs outer joins of more than two pairs of tables, a single table can only be the NULL–generated table for one other table. For this reason, you cannot apply the (+) operator to columns of B in the join condition for A and B and the join condition for B and C.

Ch. 12: Outer Join, 8i vs. 9i syntax

- NEW 9i syntax, right outer join:
  ```sql
  select B.Title, MAX(…)
  from BOOKSHELF_CHECKOUT BC
  right outer join
  BOOKSHELF B
  on BC.Title = B.Title
  group by B.Title;
  ```
  This also returns same result:
  ```sql
  select B.Title, MAX(…)
  from BOOKSHELF_CHECKOUT BC
  right outer join
  BOOKSHELF B
  using (Title)
  group by B.Title;
  ```
- Left outer join – changes the driving table to the left (2nd) table.
- Full outer join – can be achieved by a right and a left outer join queries combined together with a UNION operator.

- Pre 9i syntax:
  ```sql
  select B.Title, MAX(…)
  from BOOKSHELF_CHECKOUT BC,
  BOOKSHELF B
  where BC.Title(+) = B.Title
  group by B.Title;
  ```
Ch. 12: NOT IN replaced by Outer Join

- Left and right queries return the same result, but NOT IN query on left is much less efficient.

```sql
select Title
from BOOKSHELF
where Title NOT IN
    (select Title
     from BOOKSHELF_CHECKOUT)
order by Title;
```

- NOT IN queries use full table scans.
- Outer join queries can use indexes.

Ch. 12: NOT IN replaced by NOT EXISTS

- NOT EXISTS queries eliminate rows that cannot be joined, uses a correlated subquery which can utilize indexes rather than full scans.

```sql
select Title
from BOOKSHELF
where Title NOT IN
    (select Title
     from BOOKSHELF_CHECKOUT)
order by Title;
```

Ch. 12: NATURAL and INNER joins

- Both are new Oracle9i syntax alternatives to traditional equi-join syntax.

- NATURAL join uses columns of same name in join.
  ```sql
  select Title
  from BOOK_ORDER natural join BOOKSHELF;
  ```

- INNER join.
  ```sql
  select BO.Title
  from BOOK_ORDER BO inner join BOOKSHELF B
  on BO.Title = B.Title;
  ```
Ch. 12: UNION, INTERSECT, MINUS

- **UNION** returns distinct rows for the combination of two select statements.
- **UNION ALL** returns all rows for the combination of two select statements regardless of duplication.
- **INTERSECT** returns distinct rows for the combination of two select statements where data matches.
- **MINUS** returns the rows from one select statement excluding the rows of a second select statement.

Ch. 12: Set Operator Examples

- **An obvious example:**
  ```sql
  select Name from Longtime
  minus
  select Name from Prospect
  ```

- **A subtle example:**
  ```sql
  select Name, Lodging from Longtime
  minus
  select Name, Address from Prospect
  ```

- **Columns must be compatible for set operations, but not necessarily the same.**

Ch. 13: Some Complex Possibilities

- **New 9i Statistical Functions**
- **New 9i Temporary Tables**
- **ROLLUP, GROUPING, CUBE (Oracle 8i)**
- **START WITH, CONNECT BY, LEVEL (Oracle 7.x)**
Ch. 13: Statistical Functions

- RANK is one of many new statistical functions in 9i.
  ```sql
  select RANK(3)
  within group (order by Counter desc)
  from CATEGORY_COUNT;
  ```

- PERCENT_RANK is another new statistical function.
  ```sql
  select PERCENT_RANK(8)
  within group (order by Counter desc)
  from CATEGORY_COUNT;
  ```

- These have been added for new Oracle data warehousing capabilities.

---

Oracle example: 19-13 Hypothetical Rank and Distribution Syntax from 19 SQL for Analysis in Data Warehouses
http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96520/analysis.htm

Using the list price data from the products table used throughout this section, you can calculate the RANK, PERCENT_RANK and CUME_DIST for a hypothetical sweater with a price of $50 for how it fits within each of the sweater subcategories. The query and results are:

```sql
SELECT prod_subcategory,
       RANK(50) WITHIN GROUP (ORDER BY prod_list_price DESC) AS HRANK,
       TO_CHAR(PERCENT_RANK(50) WITHIN GROUP (ORDER BY prod_list_price),'9.999') AS HPERC_RANK,
       TO_CHAR(CUME_DIST (50) WITHIN GROUP (ORDER BY prod_list_price),'9.999') AS HCUME_DIST
FROM products
WHERE prod_subcategory LIKE 'Sweater%'
GROUP BY prod_subcategory;
```

<table>
<thead>
<tr>
<th>PROD_SUBCATEGORY</th>
<th>HRANK</th>
<th>HPERC_RANK</th>
<th>HCUME_DIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sweaters - Boys</td>
<td>16</td>
<td>.911</td>
<td>.912</td>
</tr>
<tr>
<td>Sweaters - Girls</td>
<td>1</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Sweaters - Men</td>
<td>240</td>
<td>.351</td>
<td>.352</td>
</tr>
<tr>
<td>Sweaters - Women</td>
<td>21</td>
<td>.783</td>
<td>.785</td>
</tr>
</tbody>
</table>
Ch. 13: Temporary Tables

- New Oracle 9i feature.
- Used to support aggregation/rollups of data.
- To create a temp table, use the CREATE GLOBAL TEMPORARY TABLE command instead of CREATE TABLE.
- For temp table for this session, use ON COMMIT PRESERVE ROWS clause.
- For temp table only for this transaction, use ON COMMIT DELTE ROWS clause.

```sql
create global temporary table YEAR_ROLLUP (
    Year NUMBER (4),
    Month VARCHAR2 (9),
    Counter NUMBER)
on commit preserve rows;
```

Ch. 13: ROLLUP, GROUPING, CUBE

- New Oracle 8i feature, provides OLAP like capability.
- Provides similar feature as SQLPLUS’s compute sum/break on.

- ROLLUP – used in GROUP BY clause, adds rows to the groups that includes total and subtotal rows.
- GROUPING – used in SELECT, WHERE, HAVING clauses, returns a value of 1 if the row is produced by the ROLLUP function in the GROUP BY clause. Often GROUPING is combined with DECODE.
- CUBE – used instead of ROLLUP in GROUP BY clause. This will return a subtotal line for all combinations the group by columns.

- More Oracle documentation and examples:
  **SQL for Aggregation in Data Warehouses**
  http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96520/aggreg.htm
Ch. 13: Hierarchical Queries

- CONNECT BY – used to indicate relationship within hierarchy.
  PRIOR keyword indicates parent.
- START WITH – indicates root node.
- LEVEL – pseudocolumn function that returns 1 for root, 2 for children of root, 3 for next child level, etc. Usually combined with LPAD function.

```
SELECT employee_id, last_name, manager_id, LEVEL
FROM employees
CONNECT BY PRIOR employee_id = manager_id;
```

<table>
<thead>
<tr>
<th>EMPLOYEE_ID</th>
<th>LAST_NAME</th>
<th>MANAGER_ID</th>
<th>LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Kochhar</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>108</td>
<td>Greenberg</td>
<td>101</td>
<td>2</td>
</tr>
<tr>
<td>109</td>
<td>Faviet</td>
<td>108</td>
<td>3</td>
</tr>
<tr>
<td>110</td>
<td>Chen</td>
<td>108</td>
<td>3</td>
</tr>
</tbody>
</table>

- More Oracle documentation and examples: Hierarchical Queries
  [Link](http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96540/queries4a.htm#2053937)