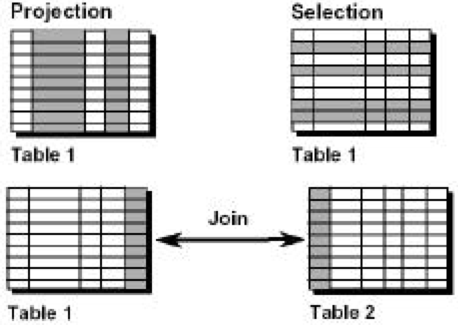
Structured Query Language (Part 2)

Capabilities of SQL SELECT Statements



Capabilities of SQL SELECT Statements

- A SELECT statement retrieves information from the database. Using a SELECT statement, you can do the following:
  - **Projection**: You can use the projection capability in SQL to choose the columns in a table that you want returned by your query.
  - **Selection**: You can use the selection capability in SQL to choose the rows in a table that you want returned by a query.
  - **Joining**: You can use the join capability in SQL to bring together data that is stored in different tables by creating a link between them.

Arithmetic Expressions

- Create expressions with number and date data by using arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

Using Arithmetic Operators

- Example
  - SELECT last_name, salary, salary + 300
    FROM employees;

LAST_NAME	SALARY	SALARY+300
King	24000	24300
Kochhar	17000	17300
De Haan	17000	17300
Hunold	9000	9300
Ernst	6000	6300
Lorentz	4200	4500
Gietz	8300	8600

Operator Precedence

- Multiplication and division take priority over addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to force prioritized evaluation and to clarify statements.

* / + −

Operator Precedence

- Example
  - SELECT last_name, salary, 12*salary+100
    FROM employees;

LAST_NAME	SALARY	12*SALARY+100
King	24000	288100
Kochhar	17000	204100
De Haan	17000	204100
Hunold	9000	108100
Ernst	6000	72100
Lorentz	4200	50500
Gietz	8300	99700

Using Parentheses

- Example
  - SELECT last_name, salary, 12*(salary+100)
    FROM employees;

LAST_NAME	SALARY	12*(SALARY+100)
King	24000	289200
Kochhar	17000	205200
De Haan	17000	205200
Hunold	9000	109200
Ernst	6000	73200
Lorentz	4200	51600
Gietz	8300	100800

Duplicate Rows

- The default display of queries is all rows, including duplicate rows.
- Example
  - SELECT department_id
    FROM employees;

DEPARTMENT_ID
90
90
90
60
60
60
50
50

Eliminating Duplicate Rows

- Eliminate duplicate rows by using the DISTINCT keyword in the SELECT clause.
- Example
  - SELECT DISTINCT department_id
    FROM employees;

DEPARTMENT_ID
10
20
50
60
80
90
110

Limiting Rows Using a Selection

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90
103	Hunold	IT_PROG	60
104	Ernst	IT_PROG	60
107	Lorentz	IT_PROG	60
124	Mourgos	ST_MAN	50

20 rows selected.

"retrieve all employees in department 90"

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

Limiting the Rows Selected

- Restrict the rows returned by using the WHERE clause.
- The WHERE clause follows the FROM clause.
- Syntax:
  - SELECT *|{[DISTINCT] column|expression [alias],...}
    FROM table
    [WHERE condition(s)];

Using the WHERE Clause

- Example
  - SELECT employee_id, last_name, job_id, department_id
    FROM employees
    WHERE department_id = 90;

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

Character Strings and Dates

- Character strings and date values are enclosed in
- single quotation marks.
- Character values are case sensitive, and date values are format sensitive.
- The default date format is DD-MON-RR.
- Example
  - SELECT last_name, job_id, department_id
    FROM employees
    WHERE last_name = 'Goyal';

Comparison Conditions

- Comparison conditions are used in conditions that compare one expression to another value or expression.
- Syntax:
  - WHERE expr operator value

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

Using Comparison Conditions

- Example
  - SELECT last_name, salary
    FROM employees
    WHERE salary <= 3000;

LAST_NAME	SALARY
Matos	[illegible]
Vargas	2500

Other Comparison Conditions

Operator	Meaning
`BETWEEN ...AND...`	Between two values (inclusive)
`IN(set)`	Match any of a list of values
`LIKE`	Match a character pattern
`IS NULL`	Is a null value

Using the BETWEEN Condition

- Use the BETWEEN condition to display rows based on a range of values.
- Example
  - SELECT last_name, salary
    FROM employees
    WHERE salary BETWEEN 2500 AND 3500;

Upper Limit

Lower Limit

LAST_NAME	SALARY
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500

Using the IN Condition

- Use the IN membership condition to test for values in a list.
- Example
  - SELECT employee_id, last_name, salary, manager_id
    FROM employees
    WHERE manager_id IN (100, 101, 201);

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100
124	Mourgos	5800	100
149	Zlotkey	10500	100
201	Hartstein	13000	100

Using the LIKE Condition

- Use the LIKE condition to perform wildcard searches of valid search string values.
- Search conditions can contain either literal characters or numbers:
  - % denotes zero or many characters.
  - _ denotes one character.
- You can use the ESCAPE identifier to search for the actual % and _ symbols.

Using the LIKE Condition

- Example
  - SELECT last_name
    FROM employees
    WHERE last_name LIKE '_o%';

LAST_NAME
Kochhar
Lorentz
Mourgos

Using the NULL Conditions

- Test for nulls with the IS NULL operator.
- Example
  - SELECT last_name, job_id, commission_pct
    FROM employees
    WHERE commission_pct IS NULL;

LAST_NAME	JOB_ID	COMMISSION_PCT
King	AD_PRES	
Kochhar	AD_VP	
De Haan	AD_VP	
Gietz	AC_ACCOUNT	

Logical Conditions

- A logical condition combines the result of two component conditions to produce a single result based on them or inverts the result of a single condition.
- A row is returned only if the overall result of the condition is true.

Operator	Meaning
AND	Returns TRUE if *both* component conditions are true
OR	Returns TRUE if *either* component condition is true
NOT	Returns TRUE if the following condition is false

Using the AND Operator

- AND requires both conditions to be true.
- Both conditions must be true for any record to be selected.
- All character searches are case sensitive.
- Character strings must be enclosed in quotation marks.

AND Truth Table

- The following table shows the results of combining two expressions with AND:

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Example of AND Operator

- Example
  - SELECT employee_id, last_name, job_id, salary
    FROM employees
    WHERE salary >=10000
    AND job_id LIKE '%MAN%';

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
149	Zlotkey	SA_MAN	10500
201	Hartstein	MK_MAN	13000

Using the OR Operator

- OR requires either condition to be true.
- Either condition can be true for any record to be selected.

The OR Truth Table

- The following table shows the results of combining two expressions with OR:

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Example of OR Operator

- Example
  - SELECT employee_id, last_name, job_id, salary
    FROM employees
    WHERE salary >= 10000
    OR job_id LIKE '%MAN%';

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
124	Mourgos	ST_MAN	5800
149	Zlotkey	SA_MAN	10500
174	Abel	SA_REP	11000
201	Hartstein	MK_MAN	13000
205	Higgins	AC_MGR	12000

Using the NOT Operator

- The NOT operator can also be used with other SQL operators, such as BETWEEN, LIKE, and NULL.
- ... WHERE job_id NOT IN ('AC_ACCOUNT', 'AD_VP')
- ... WHERE salary NOT BETWEEN 10000 AND 15000
- ... WHERE last_name NOT LIKE '%A%'
- ... WHERE commission_pct IS NOT NULL

The NOT Truth Table

- The following table shows the result of applying the NOT operator to a condition:

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Example of NOT Operator

- Example
  - SELECT last_name, job_id
    FROM employees
    WHERE job_id NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');

LAST_NAME	JOB_ID
King	AD_PRES
Kochhar	AD_VP
De Haan	AD_VP
Mourgos	ST_MAN
Zlotkey	SA_MAN
Whalen	AD_ASST
Hartstein	MK_MAN
Fay	MK_REP
Higgins	AC_MGR
Gietz	AC_ACCOUNT

Rules of Precedence

- Override rules of precedence by using parentheses.

Order Evaluated	Operator
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	NOT logical condition
7	AND logical condition
8	OR logical condition

Rules of Precedence

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id = 'SA_REP'
OR     job_id = 'AD_PRES'
AND    salary > 15000;
```

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000
Abel	SA_REP	11000
Taylor	SA_REP	8600
Grant	SA_REP	7000

Rules of Precedence

- Use parentheses to force priority.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  (job_id = 'SA_REP'
OR     job_id = 'AD_PRES')
AND    salary > 15000;
```

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000

ORDER BY Clause

- The order of rows returned in a query result is undefined.
- The ORDER BY clause can be used to sort the rows.
- If you use the ORDER BY clause, it must be the last clause of the SQL statement.
- You can specify an expression, or an alias, or column position as the sort condition.
- Sort rows with the ORDER BY clause
  - ASC: ascending order (the default order)
  - DESC: descending order
- The ORDER BY clause comes last in the SELECT statement.

Default Ordering of Data

- The default sort order is ascending:
  - Numeric values are displayed with the lowest values first: for example, 1–999.
  - Date values are displayed with the earliest value first: for exa mple, 01-JAN-92 before 01-JAN-95.
  - Character values are displayed in alphabetical order: for example, A first and Z last.
  - Null values are displayed last for ascending sequences and first for descending sequences.
- Reversing the Default Order
  - To reverse the order in which rows are displayed, specify the DESC keyword after the column name in the ORDER BY clause.

Sorting in Ascending Order

- Example
  - SELECT last_name, job_id, department_id, hire_date
    FROM employees
    ORDER BY hire_date;

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
King	AD_PRES	90	17-JUN-87
Whalen	AD_ASST	10	17-SEP-87
Kochhar	AD_VP	90	21-SEP-89
Hunold	IT_PROG	60	03-JAN-90
Ernst	IT_PROG	60	21-MAY-91
[illegible]	AD_[illegible]	90	13-JA[illegible]

Sorting in Descending Order

- Example
  - SELECT last_name, job_id, department_id, hire_date
    FROM employees
    ORDER BY hire_date DESC;

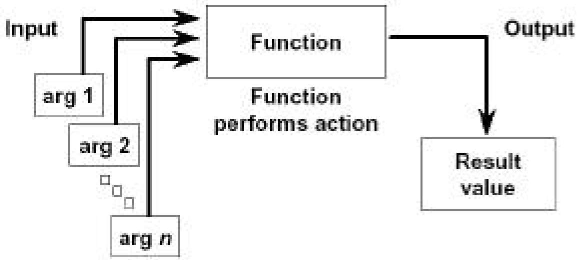
LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
Zlotkey	SA_MAN	80	29-JAN-00
Mourgos	ST_MAN	50	16-NOV-99
Grant	SA_REP		24-MAY-99
Lorentz	IT_PROG	60	07-FEB-99
Vargas	ST_CLERK	50	09-JUL-98
Taylor	SA_REP	80	24-MAR-98
Matos	ST_CLERK	50	15-MAR-98
Fay	MK_REP	20	17-AUG-97
Davies	ST_CLERK	50	29-JAN-97
Abel	SA_REP	80	11-MAY-96
King	AD_PRES	90	17-JUN-87

Sorting by Multiple Columns

- The order of ORDER BY list is the order of sort.
- You can sort by a column that is not in the SELECT list.
- Example
  - SELECT last_name, department_id, salary
    FROM employees
    ORDER BY department_id, salary DESC;

LAST_NAME	DEPARTMENT_ID	SALARY
Whalen	10	4400
Hartstein	20	13000
Fay	20	6000
Mourgos	50	5800
Rajs	50	3500
Higgins	110	[illegible]
Gietz	110	8300
Grant		7000

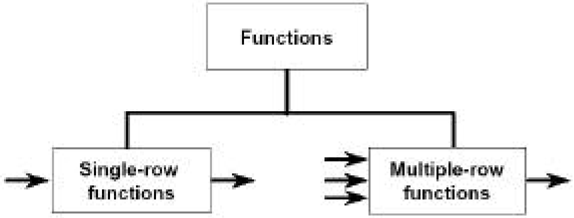
SQL Functions



SQL Functions

- Functions are a very powerful feature of SQL and can be used to do the following:
  - Perform calculations on data
  - Modify individual data items
  - Manipulate output for groups of rows
  - Format dates and numbers for display
  - Convert column data types
- SQL functions sometimes take arguments and always return a value.

Two Types of SQL Functions



Two Types of SQL Functions

- Single-Row Functions
  - These functions operate on single rows only and return one result per row. There are different types of single-row functions.
- Multiple-Row Functions
  - Functions can manipulate groups of rows to give one result per group of rows. These functions are known as group functions.

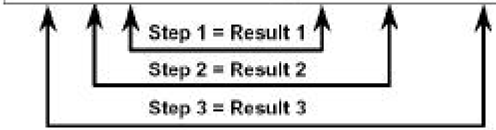
Single-Row Functions

- Single row functions:
  - Manipulate data items
  - Accept arguments and return one value
  - Act on each row returned
  - Return one result per row
  - May modify the data type
  - Can be nested
  - Accept arguments which can be a column or an expression
    - function_name [(arg1, arg2,...)]

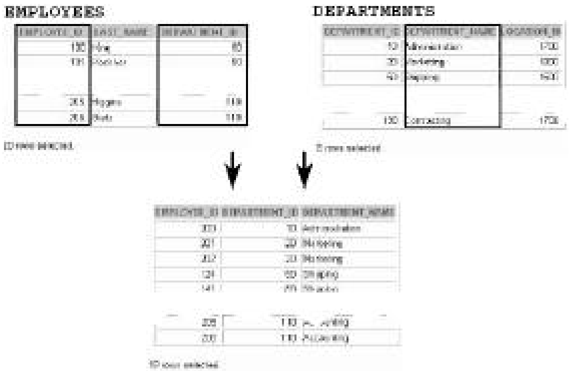
Nesting Functions

- Single-row functions can be nested to any level.
- Nested functions are evaluated from deepest level to the least deep level.

```
F3(F2(F1(col,arg1),arg2),arg3)
```



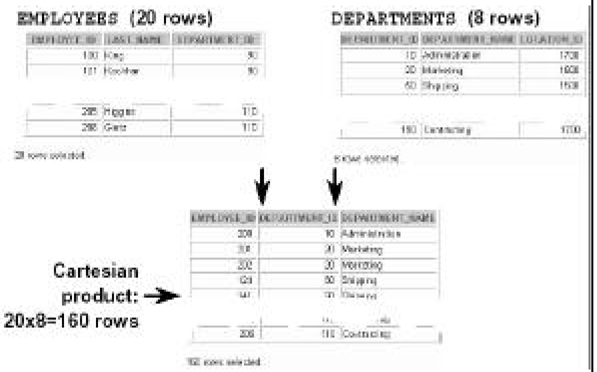
Obtaining Data from Multiple Tables



Cartesian Products

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition in a WHERE clause.

Generating a Cartesian Product



Types of Join

Oracle Proprietary Joins (8*i* and prior):

- Equijoin
- Nonequijoin
- Outer join
- Self join

SQL: 1999 Compliant Joins:

- Cross joins
- Natural joins
- Using clause
- Full or two sided outer joins
- Arbitrary join conditions for outer joins

Defining Joins

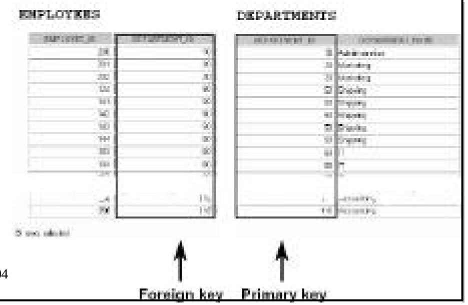
- When data from more than one table in the database is required, a join condition is used. Rows in one table can be joined to rows in another table according to common values existing in corresponding columns, that is, usually primary and foreign key columns.
- To display data from two or more related tables, write a simple join condition in the WHERE clause.
- Syntax:
  - SELECT table1.column, table2.column
    FROM table1, table2
    WHERE table1.column1 = table2.column2;

Guidelines

- When writing a SELECT statement that joins tables, precede the column name with the table name for clarity and to enhance database access.
- If the same column name appears in more than one table, the column name must be prefixed with the table name.
- To join n tables together, you need a minimum of n-1 join conditions. For example, to join four tables, a minimum of three joins is required. This rule may not apply if your table has a concatenated primary key, in which case more than one column is required to uniquely identify each row.

What Is an Equijoin?

- Equijoins are also called **Simple Joins** or **Inner Joins**.



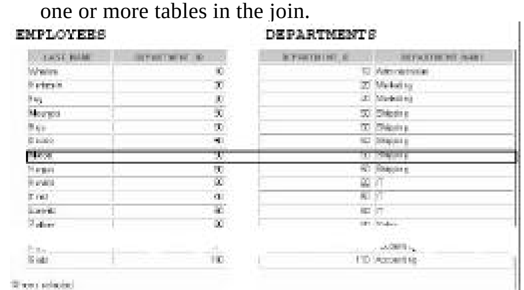
Retrieving Records with Equijoins

- Example
  - SELECT employees.employee_id, employees.last_name, employees.department_id, departments.department_id, departments.location_id
    FROM employees, departments
    WHERE employees.department_id = departments.department_id;

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500
...				
205	Higgins	110	110	1700
206	Gietz	110	110	1700

Additional Search Conditions Using the AND Operator

- In addition to the join, you may have criteria for your WHERE clause to restrict the rows under consideration for one or more tables in the join.



Additional Search Conditions Using the AND Operator

- Example
  - SELECT last_name, employees.department_id, department_name
    FROM employees, departments
    WHERE employees.department_id = departments.department_id
    AND last_name = 'Matos';

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Matos	50	Shipping

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Improve performance by using table prefixes.
- Distinguish columns that have identical names but reside in different tables by using column aliases.

Using Table Aliases

- Qualifying column names with table names can be very time consuming, particularly if table names are lengthy.
- You can use table aliases instead of table names.
- Just as a column alias gives a column another name, a table alias gives a table another name.
- Table aliases help to keep SQL code smaller, therefore using less memory.

Using Table Aliases

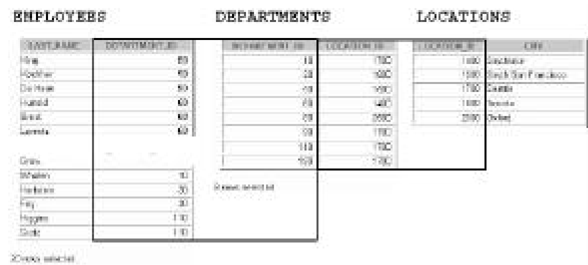
- Simplify queries by using table aliases
- Improve performance by using table prefixes
- Example
  - SELECT e.employee_id, e.last_name, e.department_id, d.department_id, d.location_id
    FROM employees e, departments d
    WHERE e.department_id = d.department_id;

Guidelines of Using Table Aliases

- Table aliases can be up to 30 characters in length, but the shorter they are the better.
- If a table alias is used for a particular table name in the FROM clause, then that table alias must be substituted for the table name throughout the SELECT statement.
- Table aliases should be meaningful.
- The table alias is valid only for the current SELECT statement.

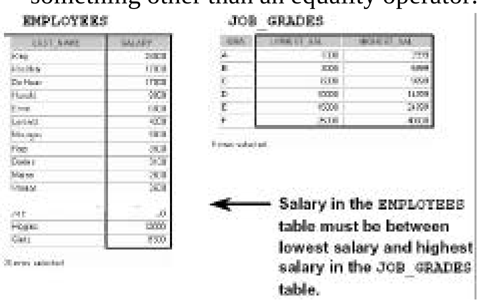
Joining More than Two Tables

- To join n tables together, you need a minimum of n-1 join conditions.



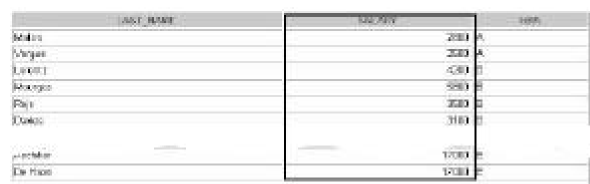
Nonequijoins

- A nonequijoin is a join condition containing something other than an equality operator.



Retrieving Records with Nonequijoins

- Example
  - SELECT e.last_name, e.salary, j.grade_level
    FROM employees e, job_grades j
    WHERE e.salary BETWEEN j.lowest_sal
    AND j.highest_sal;



Self Joins

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
101	Kochhar	100
102	De Haan	100
124	Mourgos	100
149	Zlotkey	100
201	Hartstein	100
200	Whalen	101
206	Gietz	205

19 rows selected.

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
100	King
100	King
100	King
100	King
101	Kochhar
205	Higgins

MANAGER_ID in the WORKER table is equal to EMPLOYEE_ID in the MANAGER table.

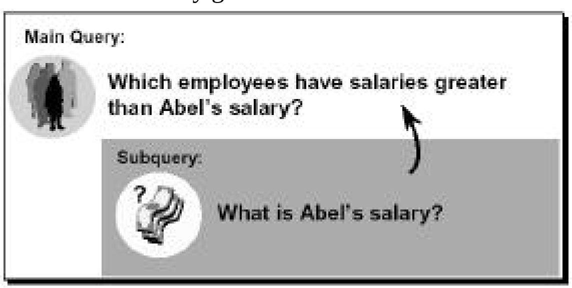
Joining a Table to Itself

- Example
  - SELECT worker.last_name
    FROM employees worker, employees manager
    WHERE worker.manager_id = manager.employee_id;

W.LAST_NAME||'WORKSFOR'||M.LAST_NAME
Kochhar works for King
De Haan works for King
Mourgos works for King
Zlotkey works for King
Hartstein works for King
Whalen works for Kochhar
Higgins works for Kochhar
Hunold works for De Haan
Fay works for Hartstein
Gietz works for Higgins

Using a Subquery to Solve a Problem

- Who has a salary greater than Abel's?

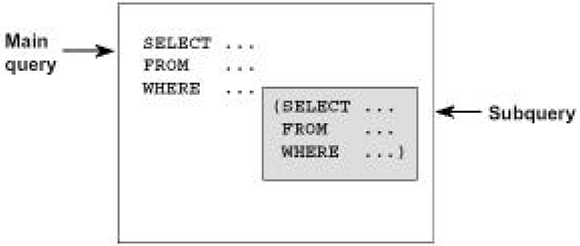


Using a Subquery to Solve a Problem

- Suppose you want to write a query to find out who earns a salary greater than Abel's salary.
- To solve this problem, you need two queries: one to find what Abel earns, and a second query to find who earns more than that amount.
- You can solve this problem by combining the two queries, placing one query inside the other query.
- The inner query, also called the subquery, returns a value that is used by the outer query or the main query. Using a subquery is equivalent to performing two sequential queries and using the result of the first query as the search value in the second query.

What is a Subquery?

- A subquery is a SELECT statement that is embedded in a clause of another SQL statement, called the parent statement.



Using a Subquery

- A subquery is a SELECT statement that is embedded in a clause of another SELECT statement.
- You can build powerful statements out of simple ones by using subqueries.
- They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

Subquery Syntax

- The subquery (inner query) executes once before the main query.
- The result of the subquery is used by the main query (outer query).
- Syntax
  - SELECT select_list
    FROM table
    WHERE expr operator
    (SELECT select_list
    FROM table);

Subquery Syntax

- You can place the subquery in a number of SQL clauses, including:
  - The WHERE clause
  - The HAVING clause
  - The FROM clause
- In the syntax: ***operator*** includes a comparison condition such as >, =, or IN

Note on Subquery

- Comparison conditions fall into two classes: single-row operators (>, =, >=, <, <>, <=) and multiplerow operators (IN, ANY, ALL).
- The subquery is often referred to as a nested SELECT, sub-SELECT, or inner SELECT statement.
- The subquery generally executes first, and its output is used to complete the query condition for the main or outer query.

Using a Subquery

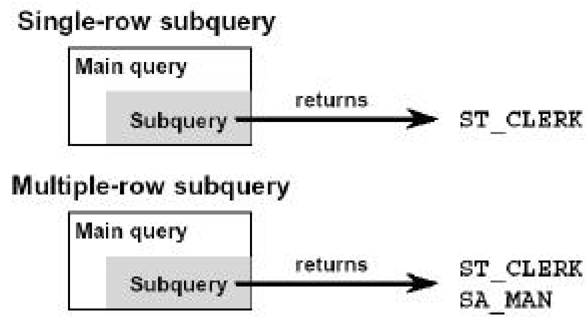
```
SELECT last_name
FROM   employees   11000
WHERE  salary >
                 (SELECT salary
                  FROM employees
                  WHERE  last_name = 'Abel');
```

LAST_NAME
King
Kochhar
De Haan
Hartstein
Higgins

Guidelines for Using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition.
- The ORDER BY clause in the subquery is not needed unless you are performing top-n analysis.
- Use single-row operators with single-row subqueries and use multiple-row operators with multiple-row subqueries.

Types of Subqueries



Types of Subqueries

- Single-row subqueries
  - Queries that return only one row from the inner SELECT statement
- Multiple-row subqueries
  - Queries that return more than one row from the inner SELECT statement
- There are also multiple-column subqueries: queries that return more than one column from the inner SELECT statement.

Single-Row Subqueries

- Return only one row
- Use single-row comparison operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

Single-Row Subqueries

- Example
  - SELECT last_name, job_id
    FROM employees
    WHERE job_id =
    (SELECT job_id
    FROM employees
    WHERE employee_id = 141);

LAST_NAME	JOB_ID
Rajs	ST_CLERK
Davies	ST_CLERK
Matos	ST_CLERK
Vargas	ST_CLERK

Executing Single-Row Subqueries

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =                ST_CLERK
                (SELECT job_id
                 FROM   employees
                 WHERE  employee_id = 141)
AND    salary >                2600
                (SELECT salary
                 FROM   employees
                 WHERE  employee_id = 143);
```

LAST_NAME	JOB_ID	SALARY
Rajs	ST_CLERK	[illegible]
Davies	ST_CLERK	[illegible]

Using Group Functions in a Subquery

- You can display data from a main query by using a group function in a subquery to return a single row.
- The subquery is in parentheses and is placed after the comparison condition.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =
                (SELECT MIN(salary)
                 FROM   employees);
```

2500

LAST_NAME	JOB_ID	SALARY
Vargas	ST_CLERK	2500

The HAVING Clause with Subqueries

- The Oracle Server executes subqueries first.
- The Oracle Server returns results into the HAVING clause of the main query.

```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
HAVING   MIN(salary) >
                       (SELECT MIN(salary)
                        FROM employees
                        WHERE department_id = 50);
```

2500

DEPARTMENT_ID	MIN(SALARY)
10	4400
20	6000
[illegible]	[illegible]

Errors with Subqueries

- One common error with subqueries is more than one row returned for a single-row subquery.
- To correct this error, change the = operator to IN.

```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
                (SELECT MIN(salary)
                 FROM   employees
                 GROUP BY department_id);
```

Single-row operator with multiple-row subquery

```
ERROR at line 4:
ORA-01427: single-row subquery returns more than
one row
```

Problems with Subqueries

- A common problem with subqueries is that no rows are returned by the inner query.

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Haas');
```

```
no rows selected
```

Subquery returns no values

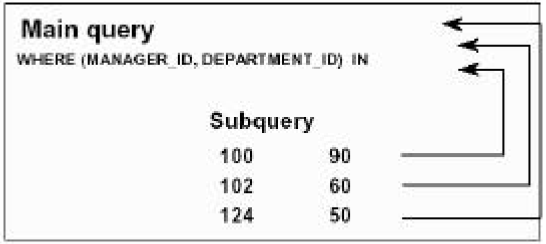
Multiple-Row Subqueries

- Subqueries that return more than one row are called multiple-row subqueries.
- You use a multiple-row operator, instead of a single-row operator, with a multiple-row subquery. The multiple-row operator expects one or more values.

Operator	Meaning
IN	Equal to any member in the list
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery

Using the IN Operator in Multiple-Row Subqueries

- Each row of the main query is compared to values from a multiple-row and multiple-column subquery.



Using the ANY Operator in Multiple-Row Subqueries

- The ANY operator (and its synonym the SOME operator) compares a value to each value returned by a subquery.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees        9000, 6000, 4200
WHERE  salary < ANY
                     (SELECT salary
                      FROM   employees
                      WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
124	Mourgos	ST_MAN	5800
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
206	Gietz	AC_ACCOUNT	8300

10 rows selected.

Using the ALL Operator in Multiple-Row Subqueries

- The ALL operator compares a value to every value returned by a subquery.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL        9000, 6000, 4200
                     (SELECT salary
                      FROM   employees
                      WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

Using a Subquery in the FROM Clause

- You can use a subquery in the FROM clause of a SELECT statement, which is very similar to how views are used.
- A subquery in the FROM clause of a SELECT statement is also called an **Inline View**.
- A subquery in the FROM clause of a SELECT statement defines a data source for that particular SELECT statement, and only that SELECT statement.

Using a Subquery in the FROM Clause

```
SELECT  a.last_name, a.salary, a.department_id, b.salavg
FROM    employees a, (SELECT   department_id,
                      AVG(salary) salavg
                      FROM     employees
                      GROUP BY department_id) b
WHERE   a.department_id = b.department_id
AND     a.salary > b.salavg;
```

LAST_NAME	SALARY	DEPARTMENT_ID	SALAVG
Hartstein	13000	20	9500
Mourgos	5800	50	3500
Hunold	9000	60	6400
Zlotkey	10500	80	10033.3333
Abel	11000	80	10033.3333
King	24000	90	19333.3333
Higgins	12000	110	10150

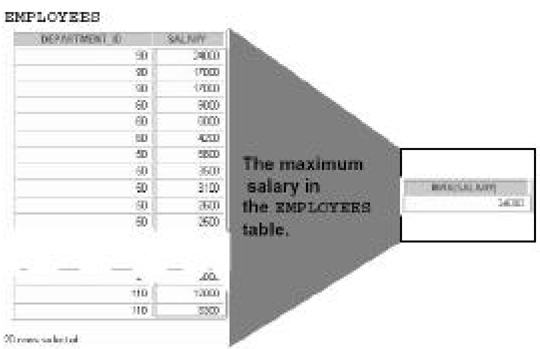
7 rows selected.

Null Values in a Subquery

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
                          (SELECT mgr.manager_id
                           FROM   employees mgr);

no rows selected
```


What are Group Functions?



Group Functions

- Group functions operate on sets of rows to give one result per group.
- Unlike single-row functions, group functions operate on sets of rows to give one result per group.
- These sets may be the whole table or the table split into groups.

Types of Group Functions

- Each of the functions accepts an argument.
- The following table identifies the options that you can use in the syntax:

Function	Description
AVG([DISTINCT|ALL]n)	Average value of n, ignoring null values
COUNT({*|[DISTINCT|ALL]expr})	Number of rows, where expr evaluates to something other than null (count all selected rows using *, including duplicates and rows with nulls)
MAX([DISTINCT|ALL]expr)	Maximum value of expr, ignoring null values
MIN([DISTINCT|ALL]expr)	Minimum value of expr, ignoring null values
STDDEV([DISTINCT|ALL]x)	Standard deviation of n, ignoring null values
SUM([DISTINCT|ALL]n)	Sum values of n, ignoring null values
VARIANCE([DISTINCT|ALL]x)	Variance of n, ignoring null values

Guidelines for Using Group Functions

- DISTINCT makes the function consider only non-duplicate values; ALL makes it consider every value including duplicates. The default is ALL and therefore does not need to be specified.
- The data types for the functions with an expr argument may be CHAR, VARCHAR2, NUMBER, or DATE.
- All group functions ignore null values. To substitute a value for null values, use the NVL, NVL2, or COALESCE functions.
- The Oracle Server implicitly sorts the result set in ascending order when using a GROUP BY clause. To override this default ordering, DESC can be used in an ORDER BY clause.

Group Functions Syntax

- Syntax
  - SELECT [column,] group_function(column), ...
    FROM table
    [WHERE condition]
    [GROUP BY column]
    [ORDER BY column];

Using the AVG and SUM Functions

- You can use AVG and SUM for numeric data.
- Example
  - SELECT AVG(salary), MAX(salary), MIN(salary), SUM(salary) FROM employees WHERE job_id LIKE '%REP%';

AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
8150	11000	6000	32600

Using the MIN and MAX Functions

- You can use MIN and MAX for any data type.
- Example
  - SELECT MIN(hire_date), MAX(hire_date) FROM employees;

MIN(HIRE_DATE)	MAX(HIRE_DATE)
17-JUN-87	29-JAN-00

Using the COUNT Function

- The COUNT function has three formats:
  - COUNT(*)
  - COUNT(expr)
  - COUNT(DISTINCT expr)

Using the COUNT Function

- COUNT(*) returns the number of rows in a table that satisfy the criteria of the SELECT statement, including duplicate rows and rows containing null values in any of the columns. If a WHERE clause is included in the SELECT statement, COUNT(*) returns the number of rows that satisfies the condition in the WHERE clause.
- COUNT(expr) returns the number of non-null values in the column identified by expr.
- COUNT(DISTINCT expr) returns the number of unique, non-null values in the column identified by expr.

Using the COUNT(*) Function

- Example
  - SELECT COUNT(*)
    FROM employees
    WHERE department_id = 50;

COUNT(*)
5

Using the COUNT(expr) Function

- COUNT(expr) returns the number of rows with non-null values for the expr.
- Example
  - SELECT COUNT(commission_pct)
    FROM employees
    WHERE department_id = 80;

COUNT(COMMISSION_PCT)
3

Using the COUNT (DISTINCT expr) Function

- Use the DISTINCT keyword to suppress the counting of any duplicate values within a column.
- COUNT(DISTINCT expr) returns the number of distinct non-null values of the expr.
- Example
  - SELECT COUNT(DISTINCT department_id)
    FROM employees;

COUNT(DISTINCTDEPARTMENT_ID)
7

Group Functions and Null Values

- All group functions ignore null values in the column.
- The average is calculated based only on the rows in the table where a valid value is stored in column.
- Example
  - SELECT AVG(commission_pct)
    FROM employees;

AVG(COMMISSION_PCT)
.2125

Creating Groups of Data

Creating Groups of Data: GROUP BY Clause Syntax

- You can use the GROUP BY clause to divide the rows in a table into groups. You can then use the group functions to return summary information for each group.
- Syntax:
  - SELECT column, group_function(column)
    FROM table
    [WHERE condition]
    [GROUP BY group_by_expression]
    [ORDER BY column];
- In the syntax: ***group_by_expression*** specifies columns whose values determine the basis for grouping rows

Guidelines

- If you include a group function in a SELECT clause, you cannot select individual results as well, unless the individual column appears in the GROUP BY clause. You receive an error message if you fail to include the column list in the GROUP BY clause.
- Using a WHERE clause, you can exclude rows before dividing them into groups.
- You must include the columns in the GROUP BY clause.
- You cannot use a column alias in the GROUP BY clause.
- By default, rows are sorted by ascending order of the columns included in the GROUP BY list. You can override this by using the ORDER BY clause.

Using the GROUP BY Clause

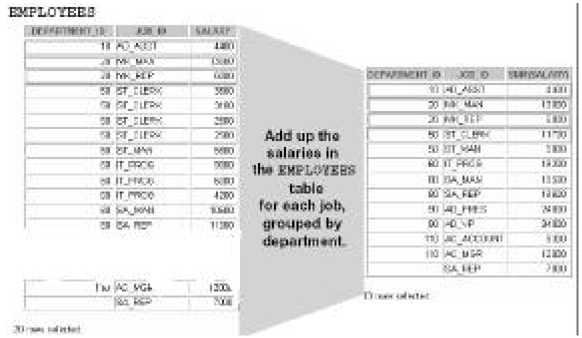
- When using the GROUP BY clause, make sure that all columns in the SELECT list that are not group functions are included in the GROUP BY clause.
  - The FROM clause specifies the tables that the database must access.
  - The WHERE clause specifies the rows to be retrieved. Because there is no WHERE clause, all rows are retrieved by default.
  - The GROUP BY clause specifies how the rows should be grouped.
- The GROUP BY column does not have to be in the SELECT clause.

Example of GROUP BY Clause

- Example
  - SELECT department_id, AVG(salary)
    FROM employees
    GROUP BY department_id
    ORDER BY AVG(salary);

DEPARTMENT_ID	AVG(SALARY)
50	3500
10	4400
60	6400

Grouping by More Than One Column



Using the GROUP BY Clause on Multiple Columns

- You can return summary results for groups and subgroups by listing more than one GROUP BY column.
- You can determine the default sort order of the results by the order of the columns in the GROUP.
  - The SELECT clause specifies the column to be retrieved.
  - The FROM clause specifies the tables that the database must access.
  - The GROUP BY clause specifies how you must group the rows.

Using the GROUP BY Clause on Multiple Columns

- Example
  - SELECT department_id dept_id, job_id, SUM(salary)
    FROM employees
    GROUP BY department_id, job_id;

DEPT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
[illegible]	[illegible]	[illegible]
	SA_REP	

Illegal Queries Using Group Functions

- Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause.

```
SELECT department_id, COUNT(last_name)
FROM   employees;
```

Column missing in the GROUP BY clause

```
SELECT department_id, COUNT(last_name)
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```


Illegal Queries Using Group Functions

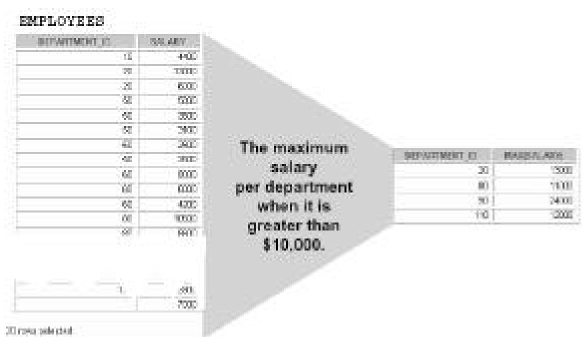
- You cannot use the WHERE clause to restrict groups.
- You use the HAVING clause to restrict groups.
- You cannot use group functions in the WHERE clause.

```
SELECT department_id, AVG(salary)
FROM   employees
WHERE  AVG(salary) > 8000
GROUP BY department_id;
```

Cannot use the WHERE clause to restrict groups

```
WHERE  AVG(salary) > 8000
       *
ERROR at line 3:
ORA-00934: group function is not allowed here
```


Excluding Group Results



Excluding Group Results: The HAVING clause

- You use the HAVING clause to specify which groups are to be displayed, and thus, you further restrict the groups on the basis of aggregate information.
- Syntax
  - SELECT column, group_function
    FROM table
    [WHERE condition]
    [GROUP BY group_by_expression]
    [HAVING group_condition]
    [ORDER BY column];
- In the syntax: ***group_condition*** restricts the groups of rows returned to those groups for which the specified condition is true

Excluding Group Results: The HAVING clause

- The Oracle Server performs the following steps when you use the HAVING clause:
  - Rows are grouped.
  - The group function is applied to the group.
  - The groups that match the criteria in the HAVING clause are displayed.
- The HAVING clause can precede the GROUP BY clause, but it is recommended that you place the GROUP BY clause first because that is more logical. Groups are formed and group functions are calculated before the HAVING clause is applied to the groups in the SELECT list.

Using the HAVING Clause

- Example
  - SELECT department_id, MAX(salary)
    FROM employees
    GROUP BY department_id
    HAVING MAX(salary)>10000;

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

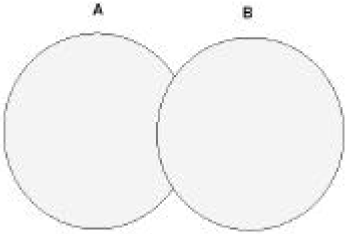
Nesting Group Functions

- Example
  - SELECT MAX(AVG(salary))
    FROM employees
    GROUP BY department_id;

MAX(AVG(SALARY))
19333.3333

The UNION SET Operator

- The UNION operator returns all rows selected by either query.
- Use the UNION operator to return all rows from multiple tables and eliminate any duplicate rows.



Guidelines

- The number of columns and the data types of the columns being selected must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
- UNION operates over all of the columns being selected.
- NULL values are not ignored during duplicate checking.
- The IN operator has a higher precedence than the UNION operator.
- By default, the output is sorted in ascending order of the first column of the SELECT clause.

Using the UNION Operator

- Example
  - SELECT employee_id, job_id
    FROM employees
    UNION
    SELECT employee_id, job_id
    FROM job_history;

EMPLOYEE_ID	JOB_ID
100	AD_PRES
101	AC_ACCOUNT
[illegible]	AD_V[illegible]
176	SA_REP
200	AC_ACCOUNT
200	AD_ASST

Appendix

Table Descriptions and Data

DEPARTMENTS Table

DESCRIBE departments

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

DEPARTMENTS Table

```
SELECT * FROM departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

8 rows selected.

EMPLOYEES Table

```
DESCRIBE employees
```

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

EMPLOYEES Table

```
SELECT * FROM employees;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000			90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000		100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000		100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000		102	60
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000		103	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200		103	60
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	5800		100	50
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	3500		124	50
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	3100		124	50
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	2600		124	50
144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	2500		124	50
149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	10500	.2	100	80
174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	11000	.3	149	80
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	8600	.2	149	80
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	.15	149	
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400		101	10
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000		100	20
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000		201	20
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000		101	110
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300		205	110

20 rows selected.