Structured Query Language

SQL

Writing SQL Statements

- SQL statements are not case sensitive
- SQL statements can be on one or more lines
- Keywords cannot be abbreviated or split across lines
- Clauses are usually placed on separate lines
- Tabs and indents are used to enhance readability

Data Definition Language

Data Definition Language (DDL) statements are used to define the database structure or schema.

- **CREATE -** to create objects in the database
- **ALTER -** alters the structure of the database
- **DROP -** delete objects from the database
- **TRUNCATE -** remove all records from a table, including all spaces allocated for the records are removed
- **RENAME -** to rename an object

Data Manipulation Language

Data Manipulation Language (DML) statements are used for managing data within schema objects.

- **SELECT -** To retrieve data from the database
- **INSERT -** To insert data into a table
- **UPDATE -** updates existing data within a table
- **DELETE -** deletes all records from a table

Data Control Language

Data Control Language (DCL) statements.

- **GRANT -** gives user's access privileges to database
- **REVOKE -** withdraw access privileges given with the GRANT command

Eg:
grant select, update on Table to User ;

revoke update on Table to User;

Create Table Syntax

```
CREATE TABLE table_name
(
column_name1 data_type (width),
column_name2 data_type (width),
column_name3 data_type (width),
....
); // without specifying constraint
```

```
CREATE TABLE <table_name> (
<column_name1> <data type>[(<width>)]
[constraint <constraint name>< constraint
type>],
<column_name2> <data type >[(<width>)],
…
 <column_nameN> <data type >[(<width>)] );
```

Data Types

- Text types – VARCHAR(size), CHAR(size),
- Number types – INT, DOUBLE, FLOAT, NUMBER

- Date types – DATE, TIME

SQL Constraints

- Constraints are used to limit the type of data that can go into a table.
- Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

SQL NOT NULL & UNIQUE Constraint

- The NOT NULL constraint enforces a column to NOT accept NULL values.
- The UNIQUE constraint uniquely identifies each record in a database table.

- CREATE TABLE table_name
  (
  attribute1 datatype NOT NULL,
  attribute2 datatype NOT NULL UNIQUE,
  attribute3 datatype
  …
  );

SQL PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a database table.
- Primary keys must contain **unique** values.
- A primary key column cannot contain **NULL** values.
- Each table should have a **primary key**, and each table can have only **ONE** primary key.

- CREATE TABLE table_name1
  (
  attribute11 datatype NOT NULL PRIMARY KEY,
  attribute12 datatype NOT NULL,
  attribute13 datatype
  …
  or CONSTRAINT pk_attr PRIMARY KEY(attr12, attr13)
  );

SQL FOREIGN KEY Constraint

- A FOREIGN KEY in one table points to a PRIMARY KEY in another table.
- CREATE TABLE table_name2
  (
  attribute21 datatype NOT NULL PRIMARY KEY,
  attribute22 datatype NOT NULL,
  attribute23 datatype

  attribute11 datatype FOREIGN KEY REFERENCES table_name1(attribute11)

  …

  or CONSTRAINT fk_attr FOREIGN KEY (attr11) REFERENCES table_name1(attr11)

  );

INSERT data into Table

- INSERT INTO table_name
  VALUES ('value1', 'value2',
  'value3',...);

Or

- INSERT INTO table_name ('&column1',
  '&column2', '&column3',...);

ALTER Table statement

- **To add a column in a table**
- ```
  ALTER TABLE table_name
  ADD column_name datatype;
  ```

- **To delete a column in a table**
- ```
  ALTER TABLE table_name
  DROP COLUMN column_name;
  ```

- **To change the data type of a column in a table**
- ```
  ALTER TABLE table_name
  MODIFY column_name datatype;
  ```


  ```

ALTER Table statement

- To allow naming of a PRIMARY KEY constraint
- CREATE TABLE table_name
  (
  coloum1 int,
  coloum2 varchar(255),
  CONSTRAINT constraint_name PRIMARY KEY (coloum1));
- **To allow naming of a PRIMARY KEY constraint**

ALTER TABLE table_name
ADD CONSTRAINT constraint_name PRIMARY KEY (coloumn1);

To Drop contraint

- ALTER TABLE table_name
  DROP CONSTRAINT constraint_name;

To allow naming of a FOREIGN KEY constraint

- ALTER TABLE table_name
  ADD CONSTRAINT constraint_name
  FOREIGN KEY (coloumn_name)
  REFERENCES base table_name(coloumn_name);

- **To DROP FOREIGN KEY Constraint**
- ALTER TABLE table_name
  DROP CONSTRAINT constraint_name;

REFERENTIAL INTEGRITY OPTIONS

- We can specify CASCADE, SET NULL on referential integrity constraints (foreign keys)

Example :

```
CREATE TABLE orders_item
(
order# NUMBER(10),
item# NUMBER(10),
qty NUMBER(6),
CONSTRAINT oi_onopk PRIMARY KEY(order#,item#),
CONSTRAINT oi_onofk foreign key (order#) REFERENCES
orders(order#) on delete cascade,
CONSTRAINT oi_inofk foreign key (item#)  REFERENCES
item(item#) on delete set null
);
```

DROP statement

- **To delete a table**
- `DROP TABLE table_name;`

- **To delete the data inside the table, and not the table itself?**
- `TRUNCATE TABLE table_name;`

- **DROP and TRUNCATE are DDL commands, whereas DELETE is a DML command.**
- **DELETE operations can be rolled back (undone), while DROP and TRUNCATE operations cannot be rolled back.**

SELECT data from Table

- `SELECT column_name(s) FROM table_name;`

- `SELECT * FROM table_name;`

Displaying Table Structure

- `DESCRIBE table_Name;`

UPDATE statement

```
UPDATE table_name
SET column1=value1, column2=value2,...
WHERE some_column=some_value;
```

DELETE statement

- ```
  DELETE FROM table_name
  WHERE some_column=some_value;
  ```


  ```

Transaction Control

- `COMMIT;`
- `ROLLBACK;`

Aggregate Functions

SQL Aggregate Functions

- SUM
- AVG
- MIN
- MAX
- COUNT

Behavior of Aggregate function

- Aggregate functions are functions that take a collection of values as input and return a single value.

- Behavior of Aggregate Functions:
  - Operates - on a single column
  - Return - a single value.
  - Used only in the SELECT list and in the HAVING clause.

Behavior of Aggregate function

- Accepts:

  - ✓ DISTINCT : consider only distinct values of the argument expression.

  - ✓ ALL : consider all values including all duplicates.

Example: SELECT COUNT(DISTINCT column_name)

Input to Aggregate Function

- SUM and AVG :
  - ➔ Operates only on collections of numbers .
- MIN , MAX and COUNT
  - ➔ Operates on collection of numeric and non-numeric data types.

Staff

sno	fname	lname	salary	position
SL100	John	White	30000.00	Manager
SL101	Susan	Brand	24000.00	Manager
SL102	David	Ford	12000.00	Project Manager
SL103	Ann	Beech	12000.00	Project Manager
SL104	Mary	Howe	9000.00	Project Manager

SUM()

Returns: The sum of the values in a specified column.

Example: Find the total/sum of the Managers salary

Query:

```
SELECT  SUM( salary) AS sum_salary
FROM Staff
WHERE Staff.position = 'Manager';
```

Result:

sum_salary
54000.00

AVG()

Returns: The average of the values in a specified column.

Example: Find the average of the Project Managers salary
.

Query:

```
SELECT  AVG( DISTINCT salary) AS avg_salary
FROM  Staff
WHERE Staff.position = 'Project Manager';
```

Result:

avg_salary
10500.00

// Error in Result
// avg_salary = 11000.00
// What is wrong?

Revised Query for AVG()

Query:

```
SELECT  AVG(ALL salary) AS avg_salary
FROM  Staff
WHERE Staff.position = 'Project Manager';
```

Result :

avg_salary
11000.00

CAUTION: Using DISTINCT and ALL in SUM() and AVG()

Staff

sno	fname	lname	salary	position
SL100	John	White	30000.00	Manager
SL101	Susan	Brand	24000.00	Manager
SL102	David	Ford	12000.00	Project Manager
SL103	Ann	Beech	12000.00	Project Manager
SL104	Mary	Howe	9000.00	Project Manager

MIN() and MAX()

Returns: MIN() returns the smallest value of a column.
MAX() returns the largest value of a column.

Example: Find the minimum and maximum staff salary.

Query:

```
SELECT  MIN( salary) AS min_salary, MAX (salary)  AS
max_salary
FROM  Staff;
```

Result:

min_salary	max_salary
9000.00	30000.00

COUNT()

Returns: The number of values in the specified column.

Example: Count number of staffs who are Manager.

Query: SELECT COUNT(sno) AS sno_count
FROM Staff
WHERE Staff.position = 'Manager';

Result:

sno_count
2

Use of COUNT() and SUM()

Example: Find the total number of Managers and the sum of t
salary.

Query: SELECT COUNT(sno) AS sno_count , SUM(salary) A
sum_salary
From Staff
WHERE Staff.position = 'Manager';

sno	fname	lname	salary	position
SL100	John	White	30000.00	Manager
SL101	Susan	Brand	24000.00	Manager

COUNT

SUM

COUNT() and SUM() continued

- Result:

sno_count	sum_salary
2	54000.00

Staff

sno	fname	lname	salary	position
SL100	John	White	30000.00	Manager
SL101	Susan	Brand	24000.00	Manager
SL102	David	Ford	12000.00	Project Manager
SL103	Ann	Beech	12000.00	Project Manager
SL104	Mary	Howe	9000.00	Project Manager

COUNT(*)

Input: There is no input to this function.

Returns: It counts all the rows of a table , regardless of whether Nulls or the duplicate occur.

Example: How many Project Manager salary is more than 9000.00

Query:
```
SELECT COUNT(*) AS Count_Salary
FROM Staff
WHERE Staff.position = 'Project Manager'
AND
Staff.salary > 9000.00
```

COUNT(*) continued…

- Result:

Count_ Salary
2

Usage of Aggregation Functions

- Use of GROUP BY
- Use of HAVING

Use of Group by Clause

- The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups.
- The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

Staff

sno	bno	fname	lname	salary	position
SL100	B3	John	White	30000.00	Manager
SL101	B5	Susan	Brand	24000.00	Manager
SL102	B3	David	Ford	12000.00	Project Manager
SL103	B5	Ann	Beech	12000.00	Project Manager
SL104	B7	Mary	Howe	9000.00	Project Manager

GROUP BY

Example: Find the number of staff working in each branch and the sum of their salaries.

Query:

```
SELECT bno, COUNT(sno) AS count, SUM(salary) AS sum
FROM Staff
GROUP BY bno
ORDER by bno;
```

Result:

bno	count	sum
B3	2	42000.00
B5	2	36000.00
B7	1	9000.00

SQL'S ROLE

bno	sno	salary	COUNT (sno)	SUM (salary)
B3	SL100	30000.00	2	42000.00
B3	SL102	12000.00		
B5	SL101	24000.00	2	36000.00
B5	SL103	12000.00		
B7	Sl104	9000.00	1	9000.00

USE OF HAVING

HAVING clause: It is designed to be used with GROUP BY so that it can restrict the groups that appear in the final result table.

Example: For each branch office with more than one member of staff, find the number of staff working in each branch and the sum of their salaries.

Query: SELECT bno, COUNT(sno) AS count, SUM(salary) AS sum
FROM Staff
GROUP BY bno
HAVING COUNT(sno) > 1
ORDER by bno;

Having Clause continued….

bno	COUNT (sno)	SUM (salary)
B3	2	42000
B5	2	36000
B7	1	9000

- Result table after performing GROUP BY bno clause.

bno	count	sum
B3	2	42000
B5	2	36000

- Final result table after performing HAVING COUNT(sno) > 1 ORDER by bno;

Nested Queries or Sub Queries

- **Syntax**

```
select <column(s)>
from table                          ←------OUTER QUERY
where <condn> operator
        (select <column>
          from table);              ←----INNER QUERY
```

Operator can be any one of >, =, or IN

Comparision condition may be

single row operators like >, =, >=, <, <=, <>

Multiple row operators like IN,ANY,ALL

Using a Subquery to Solve a Problem

Who has a salary greater than Abel's?

Main Query:

Which employees have salaries greater than Abel's salary?

Subquery

What is Abel's salary?

Subquery Syntax

```
SELECT    select_list
FROM      table
WHERE     expr operator
                      (SELECT       select_list
                      FROM          table);
```

- The subquery (inner query) executes once before the main query.
- The result of the subquery is used by the main query (outer query).

Sub Queries

- Single Row Sub Query
  - inner query returns only one row
- Multiple Row Sub Query
  - inner query returns more than one row

Example: display the names of the employee working for account department.

```
Select name from employee
Where dno= (select dno from department
Where dname='account')
```

Output

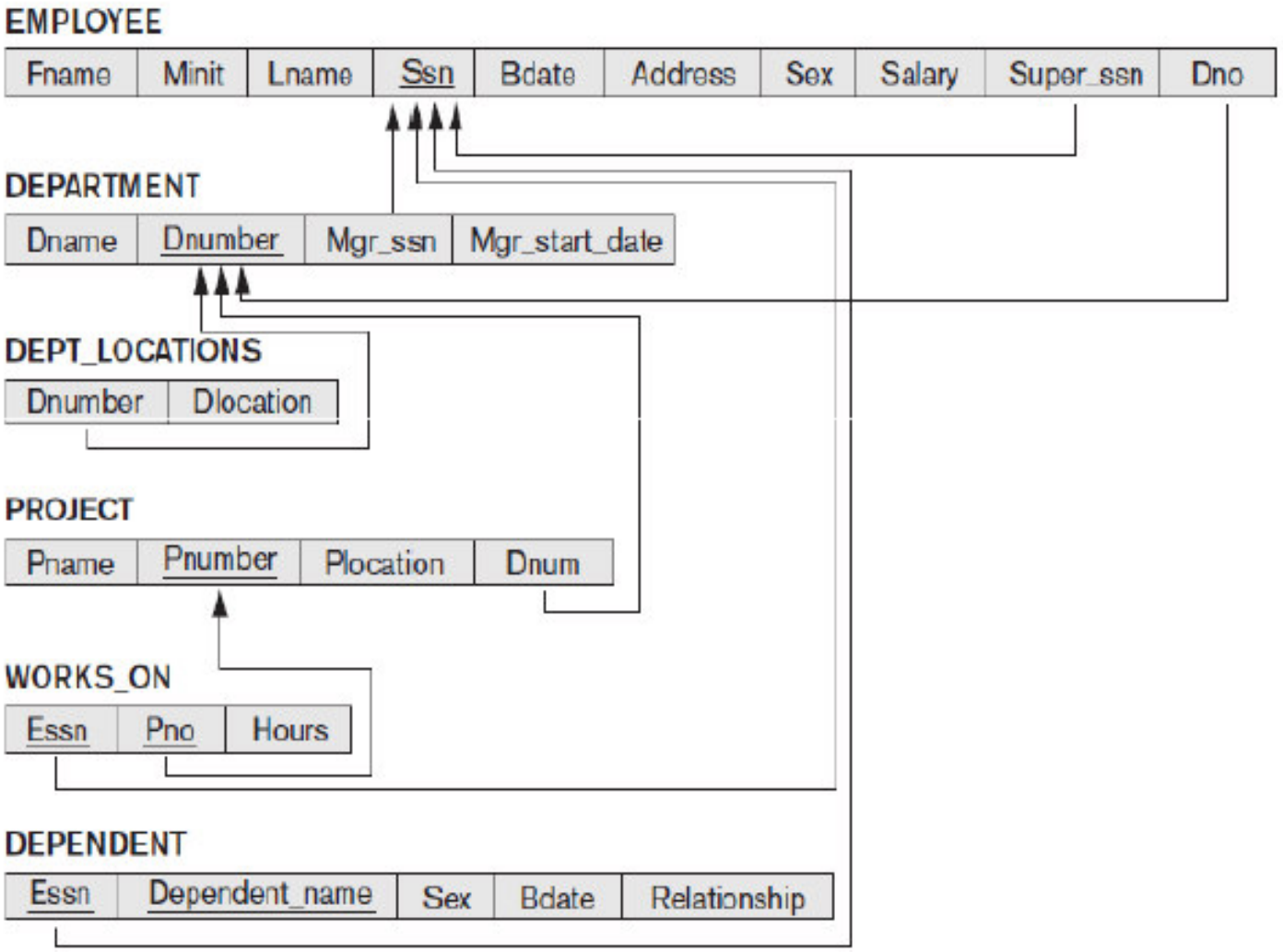
Name
Raj
Prasad
Reena

Note: inner query should give single row value (= sign)

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

COMPANY database schema



Using a Subquery

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

11000

LAST_NAME
King
Kochhar
De Haan
Hartstein
Higgins

Types of Subqueries

- **Single-row subquery**

- **Multiple-row subquery**

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

Executing Single-Row Subqueries

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

ST_CLERK

2600

LAST_NAME	JOB_ID	SALARY
Rajs	ST_CLERK	3500
Davies	ST_CLERK	3100

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.

Using Group Functions in a Subquery

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

2500

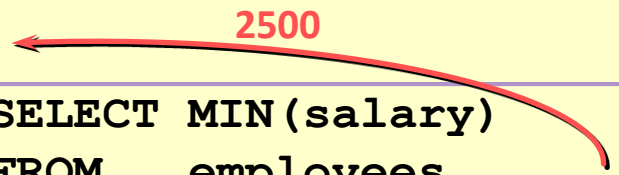
LAST_NAME	JOB_ID	SALARY
Vargas	ST_CLERK	2500

The HAVING Clause with Subqueries

- You can use subqueries not only in the WHERE clause, but also in the HAVING clause.
- DBMS executes the subquery, and the results are returned into the HAVING clause of the main query.

The HAVING Clause with Subqueries

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



What is Wrong with this Statement?

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

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

Single-row operator with multiple-row subquery

Errors with Subqueries

- One common error with subqueries is more than one row returned for a single-row subquery.
- The `WHERE` clause contains an equal (=) operator, a single-row comparison operator expecting only one value.
- The = operator cannot accept more than one value from the subquery and therefore generates the error.

Will this Statement Return Rows?

```
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

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
```

Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

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

Multiple Row Sub Query:

The table gives an idea to use ANY and ALL

Operator	Meaning	Example
<ANY	Less than the maximum	E<ANY(5,3,8): E is less than any single item in the list (5,3,8).even 7 qualifies. because 7<8.
>ANY	More than the minimum	E>ANY(5,3,8): E is greater than any single item in the list (5,3,8).even 4 qualifies. Because 4>3.
=ANY	Same as IN	E=ANY(5,3,8). All values in the list qualify
<ALL	Less than the minimum	E<ALL(5,3,8). Anything below 3 qualifies
>ALL	More than the maximum	E>ALL(5,3,8). Anything greater than 8 qualifies

Using the ANY Operator in Multiple-Row Subqueries

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

9000, 6000, 4200

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
144	Vargas	ST_CLERK	2500

...

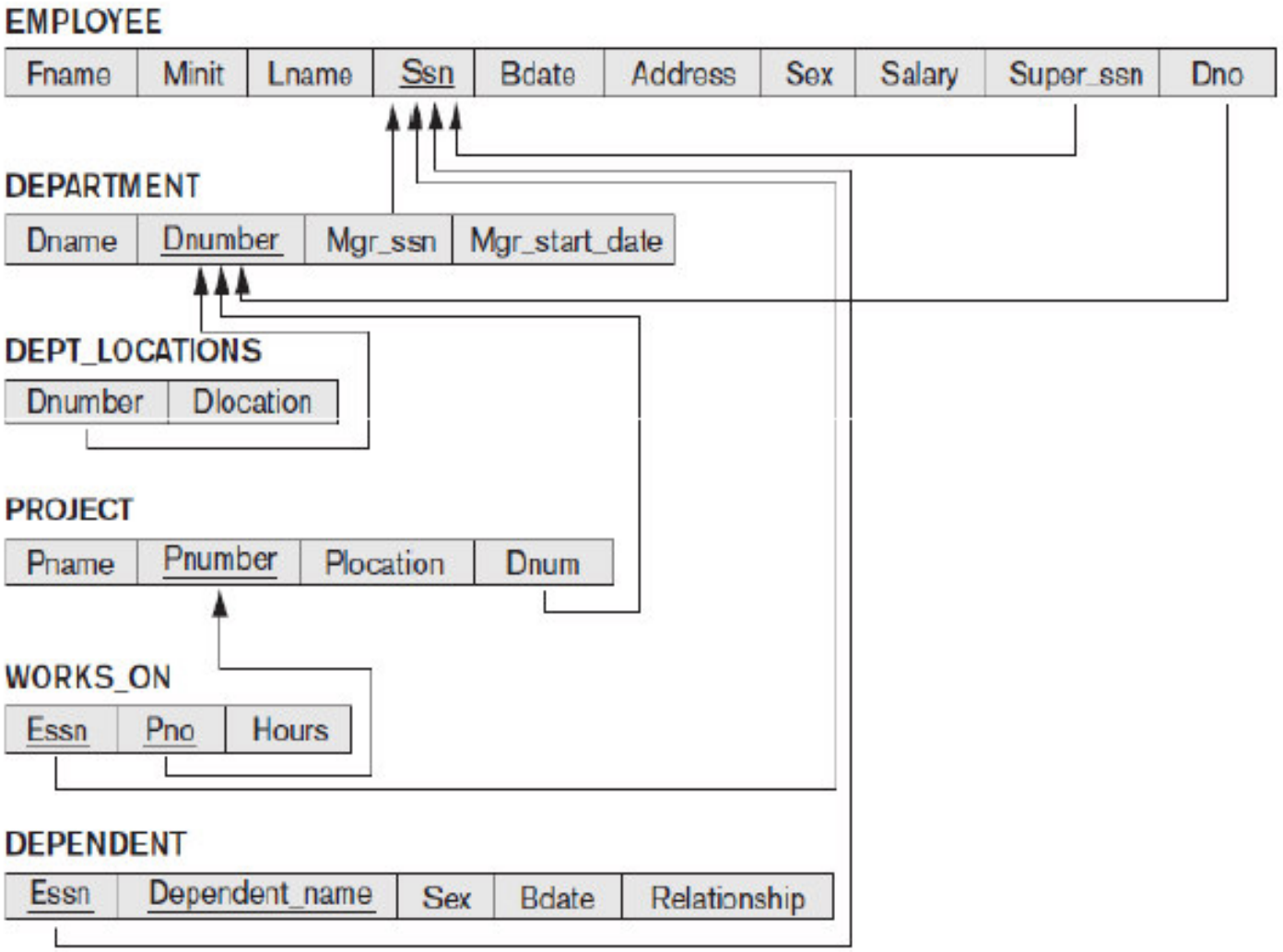
10 rows selected.

Using the ALL Operator in Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
                (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

COMPANY database schema



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

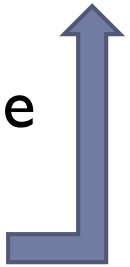
DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Sub Queries

Example 2: suppose we want to display name of the highest paid employee.

```
select name from employee
where salary >
 (select salary from employee
 where name='deepak');
```



20000

Output: **Name**

raj

Prasad

Inner query gives a single row with a value of 20000

Examples..

```
Select name,salary from employee
where salary < ANY
(Select salary from employee
 where dno=3)
```

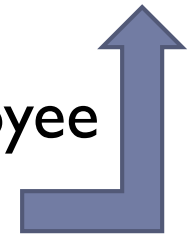
<30000,32000,8000>

Output:

name	salary
raj	8000
ravi	30000
smith	20000

Examples..

```
Select name,salary from employee
 where salary < ALL
 (Select salary from employee
  where dno=3)
```



<30000,32000,8000>

OUTPUT:

If anyone draws salary lower than minimum value in the set, their names will be displayed

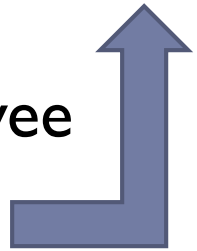
Examples..

Select name,salary from employee

where salary > ALL

(Select salary from employee

where dno=3)



<30000,32000,8000>

- OUTPUT:

If anyone draws salary more than maximum value in the set, their names will be displayed

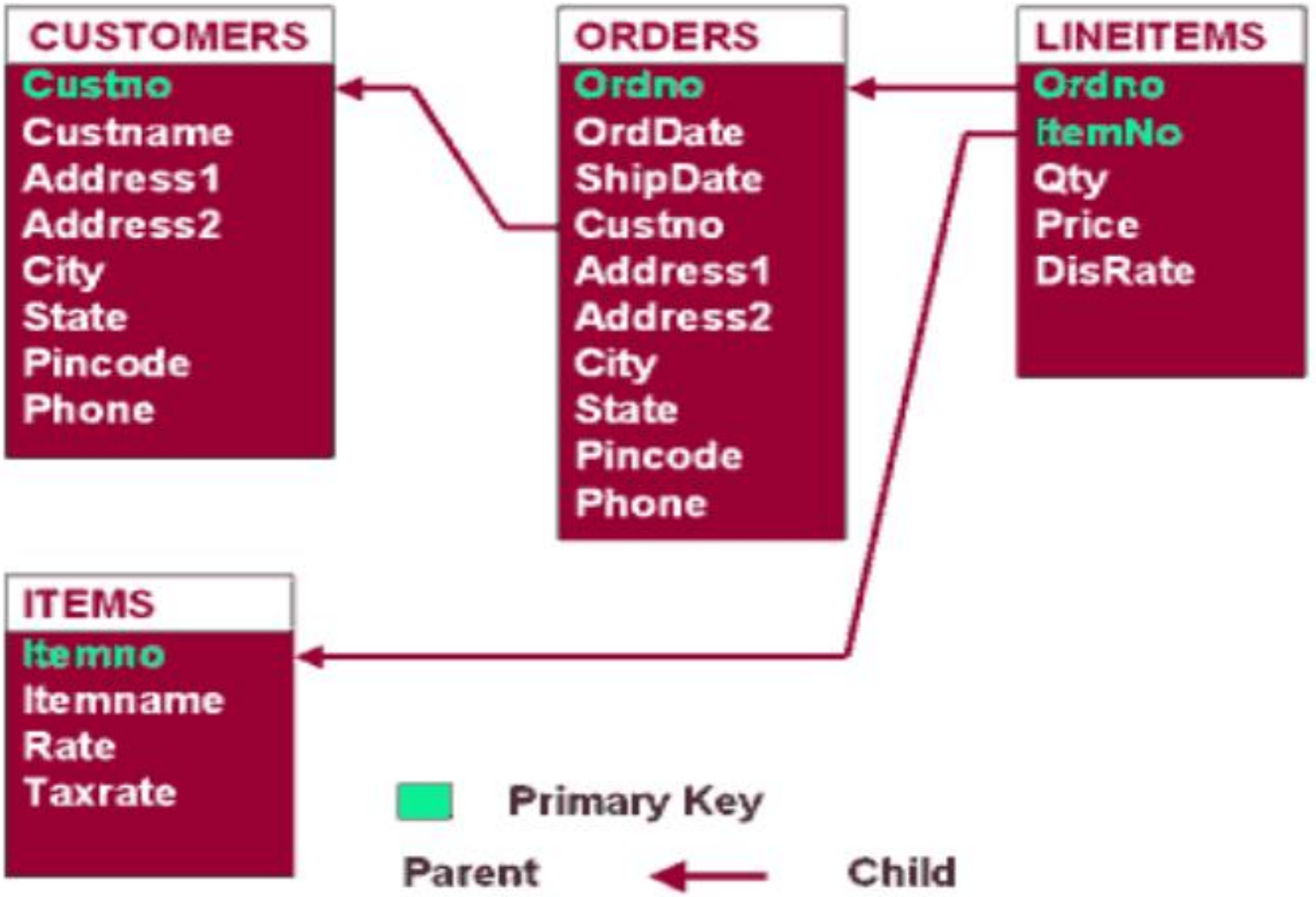
Note: we can say that >ALL means greater than the greatest and <ALL means less than the lowest value

Some Queries

- **Qa:** Returns the names of employees whose salaries are more than the salaries of all those whose supervisor ssn is '333445555'.
- **Aa:** SELECT FNAME FROM EMPLOYEE WHERE SALARY > **ALL** (SELECT SALARY FROM EMPLOYEE WHERE SUPERSSN='333445555');

- **Qb:** Return all employee names who have the same name as their supervisors.
- **Ab:** SELECT E.FNAME FROM EMPLOYEE E WHERE E.SSN **IN** (SELECT SUPERSSN FROM EMPLOYEE WHERE E.FNAME = FNAME);

Purchase Order Processing DB



Queries[Group by and Having Clause]

1. Display customer name and total amount of items purchased by customer.
2. Display no.of orders placed by customers residing in bangalore.
3. Display highest no.of orders placed by a single customer.
4. Display customer who has placed more than 2 orders in a single month.
5. Display state,no.of customers in the state where the customer name contains the word 'nike'.
6. Display orderno,average of price by taking into orders that were placed in the last 15 days.
7. Display orderno,custname,orderdate,no.of date between shipdate and orderdate for orders that have been shipped
8. Display custno,date,no.of orders placed
9. Display orderno,orderdate,custno,name for all the orders where the order contains order for itemno 5.
10. Display itemno,total no.of units sold,maxprice,minprice

Queries[Group by and Having Clause]

11. Display orderno,max price in the order for the orders where the amount of items is more than 10000
12. Display custno,date on which first order was placed and the gap between first order and last order in days
13. Display orderno for orders where atleast one product is having rate more than 5000 and total no.of units is more than 10
14. Display total no of orders
15. Disply orderno, no.of items in an order and avg rate of orders

Nested Queries

1. Display orderno,orderdate,custno,name for all the orders where the order contains order for itemno 5.
2. Display details of customers who placed any orders worth more than 30000
3. Display details of order in which we sold item 3 for max price
4. Display details of items for which there is an order in the last 7 days or total no.of units ordered is more than 10.
5. Display all the lineitems in which the rate of the item is more than avg rate of the items
6. Display details of orders in which atleast one item is sold for higher rate than actual rate
7. Display details of customer who has placed max no of orders
8. Details of customers who have not placed any order for the last 15 days
9. Display details of items that are purchased by customer 102

Nested Queries

10. Change shipdate of order 1004 to the order date of most recent order
11. Display the details of item that has highest price.
12. Display details of customers who placed more than 5 orders.
13. Display details of cutomers who have not placed any order.
14. Display details of cutomers who have placed an order in the last 6 months.
15. Display the items for which we have sold more than 50 units by taking into orders where the price is more than 5000.
16. Display the details of orders that were placed by a customer with phone number starting with 541 or the orders in which we have more than 5 items.
17. Change the rate of itemno 1 in items table to the highest rate of lineitems table of that item.
18. Delete customers who have not placed any order.