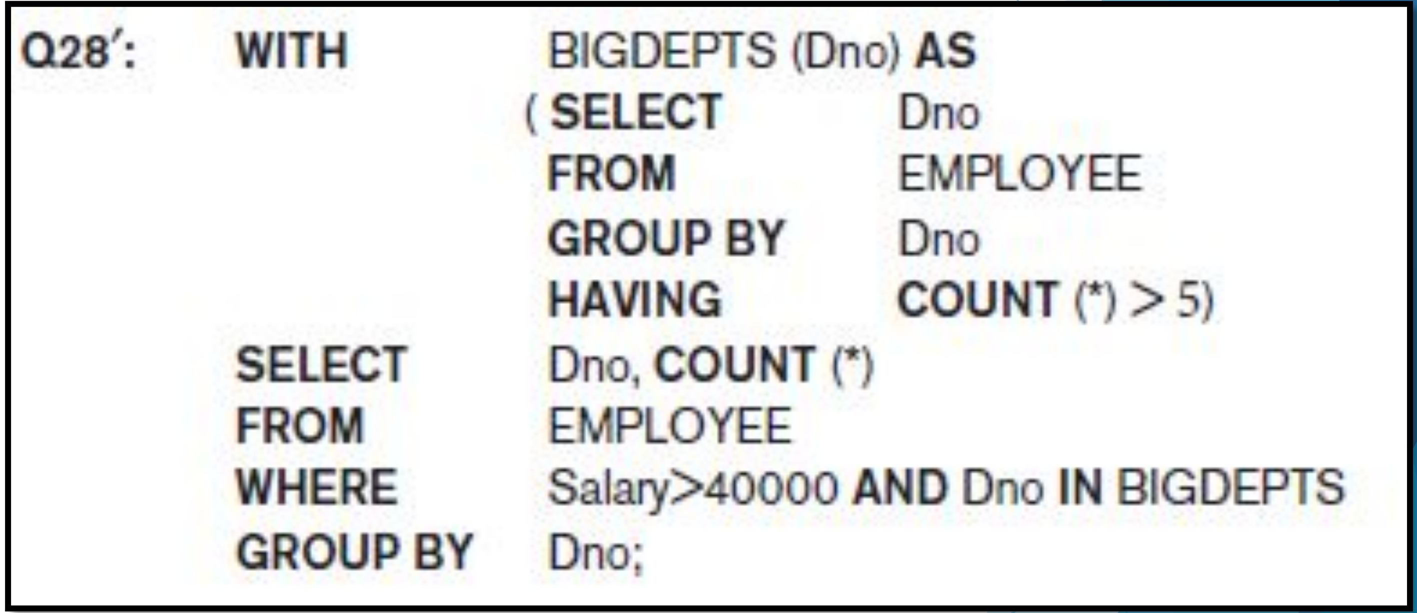
More Complex SQL Retrieval Queries

- ***Other SQL Constructs: WITH and CASE***

- The WITH clause allows a user to define a table that will **only be used in a particular query.**
- Its similar to creating a view used for next query & then dropped
- This construct is not available in all SQL based DBMSs

More Complex SQL Retrieval Queries

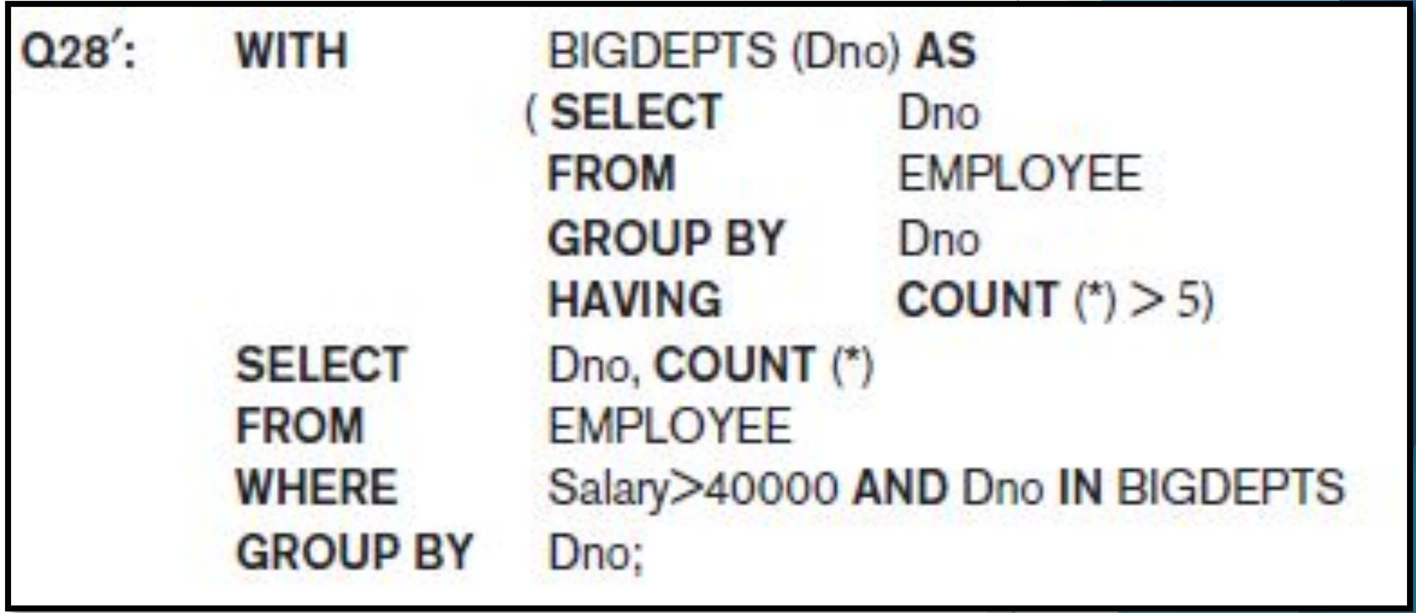
- ***<u>Other SQL Constructs: WITH and CASE</u>***
- In Q28′, we defined in the WITH clause a **temporary table BIG_DEPTS** whose result holds the Dno’s of departments with more than five employees, then used this table in the subsequent query.
- Once this query is executed, the temporary table BIGDEPTS is discarded.



```
Q28′:   WITH        BIGDEPTS (Dno) AS
                  ( SELECT      Dno
                    FROM        EMPLOYEE
                    GROUP BY    Dno
                    HAVING      COUNT (*) > 5)
        SELECT      Dno, COUNT (*)
        FROM        EMPLOYEE
        WHERE       Salary>40000 AND Dno IN BIGDEPTS
        GROUP BY    Dno;
```

More Complex SQL Retrieval Queries

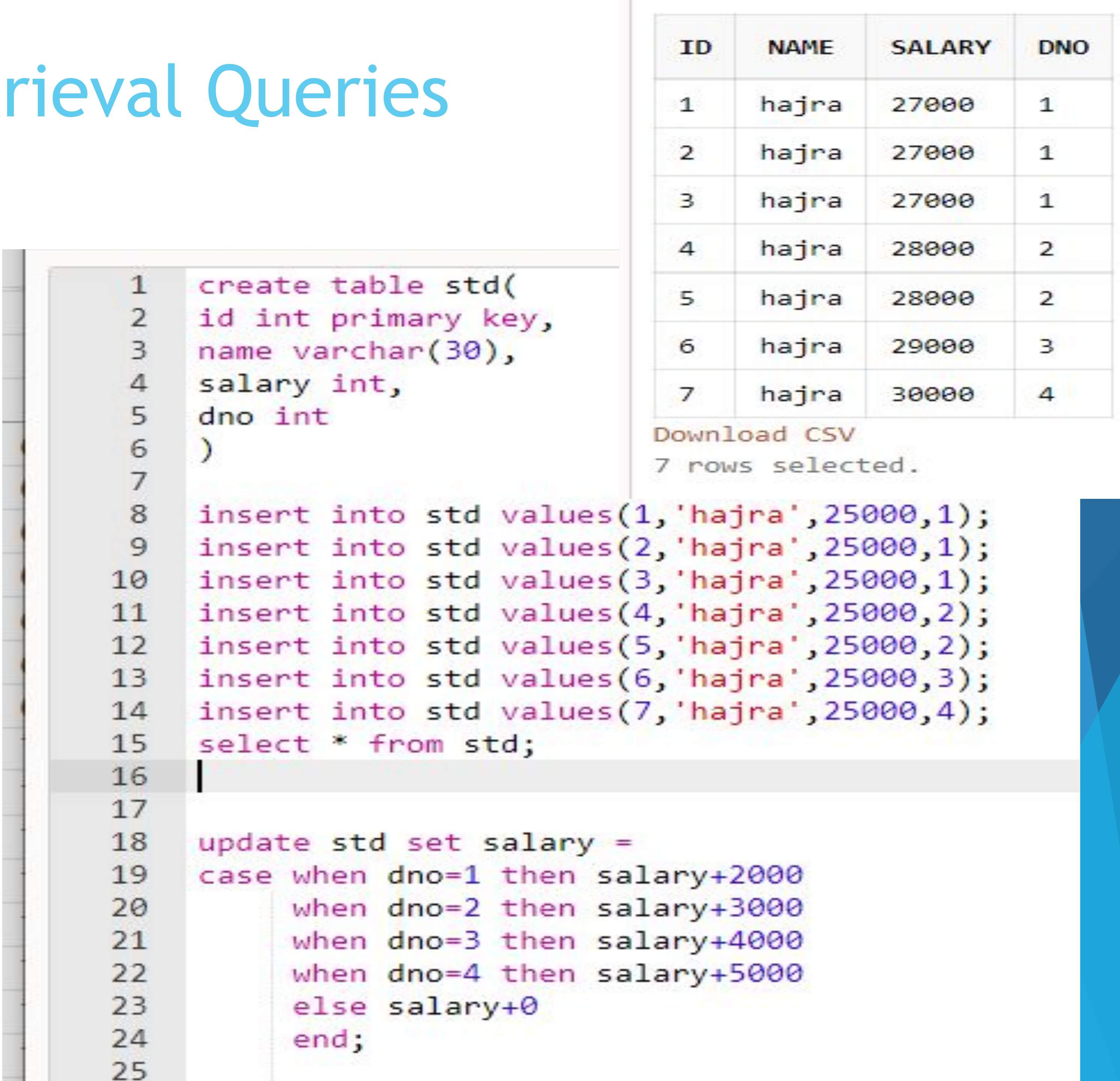
- *Other SQL Constructs: WITH*
- It is **considered “temporary” because the result is not permanently stored anywhere in the database schema.**
- advantage:
  - simplify long and complex hierarchical queries
  - by breaking them down into smaller, more readable chunks.



```
Q28′:   WITH        BIGDEPTS (Dno) AS
                    ( SELECT      Dno
                      FROM        EMPLOYEE
                      GROUP BY    Dno
                      HAVING      COUNT (*) > 5)
        SELECT      Dno, COUNT (*)
        FROM        EMPLOYEE
        WHERE       Salary>40000 AND Dno IN BIGDEPTS
        GROUP BY    Dno;
```

More Complex SQL Retrieval Queries

- ***Other SQL Constructs: CASE***
- **CASE construct** can be used when a value can be different based on certain conditions.
- This can be **used in any part of an SQL query where a value is expected**, including when querying, inserting or updating tuples.
- Suppose we have to **give employees different raise amounts depending on which department they work for**.
- The CASE construct can also be used when **inserting tuples.**



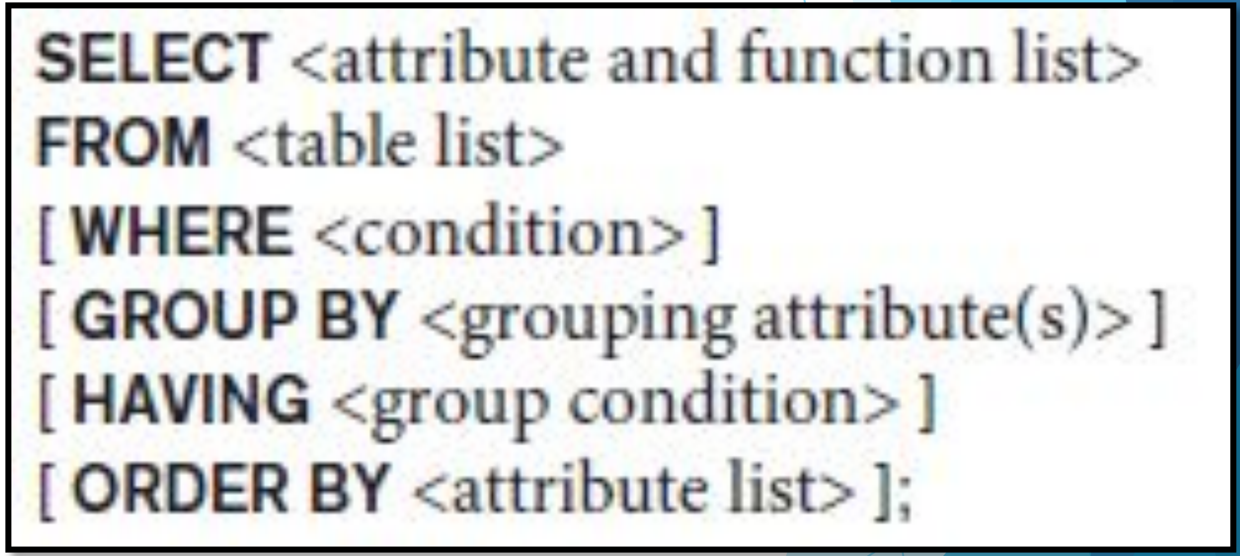
```sql
create table std(
id int primary key,
name varchar(30),
salary int,
dno int
)

insert into std values(1,'hajra',25000,1);
insert into std values(2,'hajra',25000,1);
insert into std values(3,'hajra',25000,1);
insert into std values(4,'hajra',25000,2);
insert into std values(5,'hajra',25000,2);
insert into std values(6,'hajra',25000,3);
insert into std values(7,'hajra',25000,4);
select * from std;


update std set salary =
case when dno=1 then salary+2000
     when dno=2 then salary+3000
     when dno=3 then salary+4000
     when dno=4 then salary+5000
     else salary+0
     end;
```

ID	NAME	SALARY	DNO
1	hajra	27000	1
2	hajra	27000	1
3	hajra	27000	1
4	hajra	28000	2
5	hajra	28000	2
6	hajra	29000	3
7	hajra	30000	4

```
7 rows selected.
```

More Complex SQL Retrieval Queries- repeated overview

- *Discussion and Summary of SQL Queries* A retrieval query in SQL can consist of up to six clauses, **but only the first two— SELECT and FROM—are mandatory.**
- The query can span several lines and is ended by a semicolon.
- Query terms are separated by spaces, and parentheses can be used to group relevant parts of a query in the standard way.
- **The clauses are specified in the following order, with the clauses between square brackets […] being optional**.



```
SELECT <attribute and function list>
FROM <table list>
[ WHERE <condition> ]
[ GROUP BY <grouping attribute(s)> ]
[ HAVING <group condition> ]
[ ORDER BY <attribute list> ];
```

More Complex SQL Retrieval Queries – repeated overview

- *Discussion and Summary of SQL Queries*
- The **SELECT** clause lists the attributes or functions to be retrieved.
- The **FROM** clause specifies all relations (tables) needed in the query, including joined relations, but not those in nested queries.
- The **WHERE** clause specifies the conditions for selecting the tuples from these relations, including join conditions if needed.
- **GROUP BY** specifies grouping attributes, whereas **HAVING** specifies a condition on the groups being selected rather than on the individual tuples.
- The built-in aggregate functions **COUNT, SUM, MIN, MAX, and AVG** are used in conjunction with grouping, but they can also be applied to all the selected tuples in a query without a GROUP BY clause.
- Finally, **ORDER BY** specifies an order for displaying the result of a query.

More Complex SQL Retrieval Queries

- ***Discussion and Summary of SQL Queries***
- A query is evaluated conceptually **by first applying the FROM clause** (to identify all tables involved in the query or to materialize any joined tables), **followed by the WHERE clause to select and join tuples, and then by GROUP BY and HAVING.**
- Conceptually, **ORDER BY is applied at the end to sort the query result**.
- If none of the last three clauses (GROUP BY, HAVING, and ORDER BY) are specified, we can think conceptually of a query as being executed as follows:
- For each combination of tuples—one from each of the relations specified in the FROM clause—evaluate the WHERE clause; if it evaluates to TRUE, place the values of the attributes specified in the SELECT clause from this tuple combination in the result of the query.

Schema Change Statements in SQL

- Schema changes are required when:
  - a relation/constraint is altered or dropped from the schema
- For that schema evolution commands are used
- For example, ALTER and DROP
- These don’t need recompilation of database schema as the changes are done in existing DBMS.
- Just make sure that the DBMS remains in the consistent state even after the changes are done.

Schema Change Statements in SQL

- ***The DROP Command***
- The DROP command can be used to drop named schema elements, **such as tables, domains or constraints**.
- One can also drop a whole schema if it is no longer needed by using the **DROP SCHEMA command.**
- There are two drop behavior options: CASCADE and RESTRICT.
  - For example, to **remove the COMPANY database schema and all its tables, domains, and other elements**, the CASCADE option is used as follows:
    - **DROP SCHEMA COMPANY CASCADE;**
- If the **RESTRICT option** is chosen in place of CASCADE, the **schema is dropped only if it has no elements in it**; **otherwise,** the DROP **command will not be executed.**
- To use the RESTRICT option, the **user must first individually drop each element in the schema**, then drop the schema itself.

Schema Change Statements in SQL

- *The DROP Command*
- If a **base relation** within a schema is no longer needed, the relation and its definition can be deleted by using the **DROP TABLE** command.
- For example, if we no longer wish to keep track of dependents of employees in the COMPANY database, we can get rid of the DEPENDENT relation by issuing the following command:
  - **DROP TABLE DEPENDENT CASCADE;**
- **CASCADE option**,
  - all **constraints, views, and other elements that reference the table being dropped are also dropped automatically** from the schema, along with the table itself.
  - (drop all referential integrity constraints that refer to primary keys).
- **RESTRICT option**
  - **a table is dropped only if it is not referenced in any constraints or views or by any other elements.**
  - (for example, not referenced by foreign key in another relation)

Schema Change Statements in SQL

- ***The DROP Command***

- DROP TABLE = **deletes all the records in the table** + **removes the table definition from the catalog**.
- For only record deletion -> use **DELETE** command.

Schema Change Statements in SQL

- ***The ALTER Command***
- The definition of a base table or of other named schema elements can be changed by using the **ALTER command**.
- For base tables, the possible alter table actions include
  - adding or dropping a column (attribute),
  - changing a column definition,
  - and adding or dropping table constraints.

Schema Change Statements in SQL

- *The ALTER Command : add atttribute*
- For example, **to add an attribute for keeping track of jobs of employees** to the EMPLOYEE base relation in the COMPANY schema we can use the command
  - **ALTER TABLE EMPLOYEE ADD COLUMN job VARCHAR(20);**
- After alteration, we have to **add the value of job attribute for already registered employee.**
  - Solution:
    - **specifying a default clause**
    - **by using the UPDATE command individually on each tuple.**
- **If no default clause is specified, the new attribute will have NULLs in all the tuples of the relation immediately after the command is executed**
  - **hence, the NOT NULL constraint is not allowed in this case.**

Schema Change Statements in SQL

- ***The ALTER Command: remove a column***
- To drop a column, we must choose either CASCADE or RESTRICT for drop behavior.
  - If **CASCADE** is chosen, all constraints and views that reference the column are dropped automatically from the schema, along with the column.
  - If **RESTRICT** is chosen, the command is successful only if no views or constraints (or other schema elements) reference the column.
- For example, the following command removes the attribute Address from the EMPLOYEE base table:
  - **ALTER TABLE EMPLOYEE DROP COLUMN Job CASCADE;**

Schema Change Statements in SQL

- *The ALTER Command: change a column definition*
- It is also possible to alter a column definition by **dropping an existing default clause or by defining a new default clause**. The following examples illustrate this clause:
  - **ALTER TABLE DEPARTMENT ALTER COLUMN job DROP DEFAULT;**
  - **ALTER TABLE DEPARTMENT ALTER COLUMN job SET DEFAULT ‘employed’;**

Schema Change Statements in SQL

- *The ALTER Command: change constraints*
- You can add or drop a constraint.
- **Only the named constraints can be dropped from table.**
- For example, to drop the constraint named EMPSUPERFK in Figure 6.2 from the EMPLOYEE relation, we write:
  - **ALTER TABLE EMPLOYEE**

    DROP CONSTRAINT EMPSUPOERFK CASCADE;
- After dropping a constraint, new constraint can be defined over a column.
- A primary key constraint can be added as :
  - **ALTER TABLE EMPLOYEE**

    ADD CONSTRAINT EMPPK PRIMARY KEY (email);

Summary of SQL

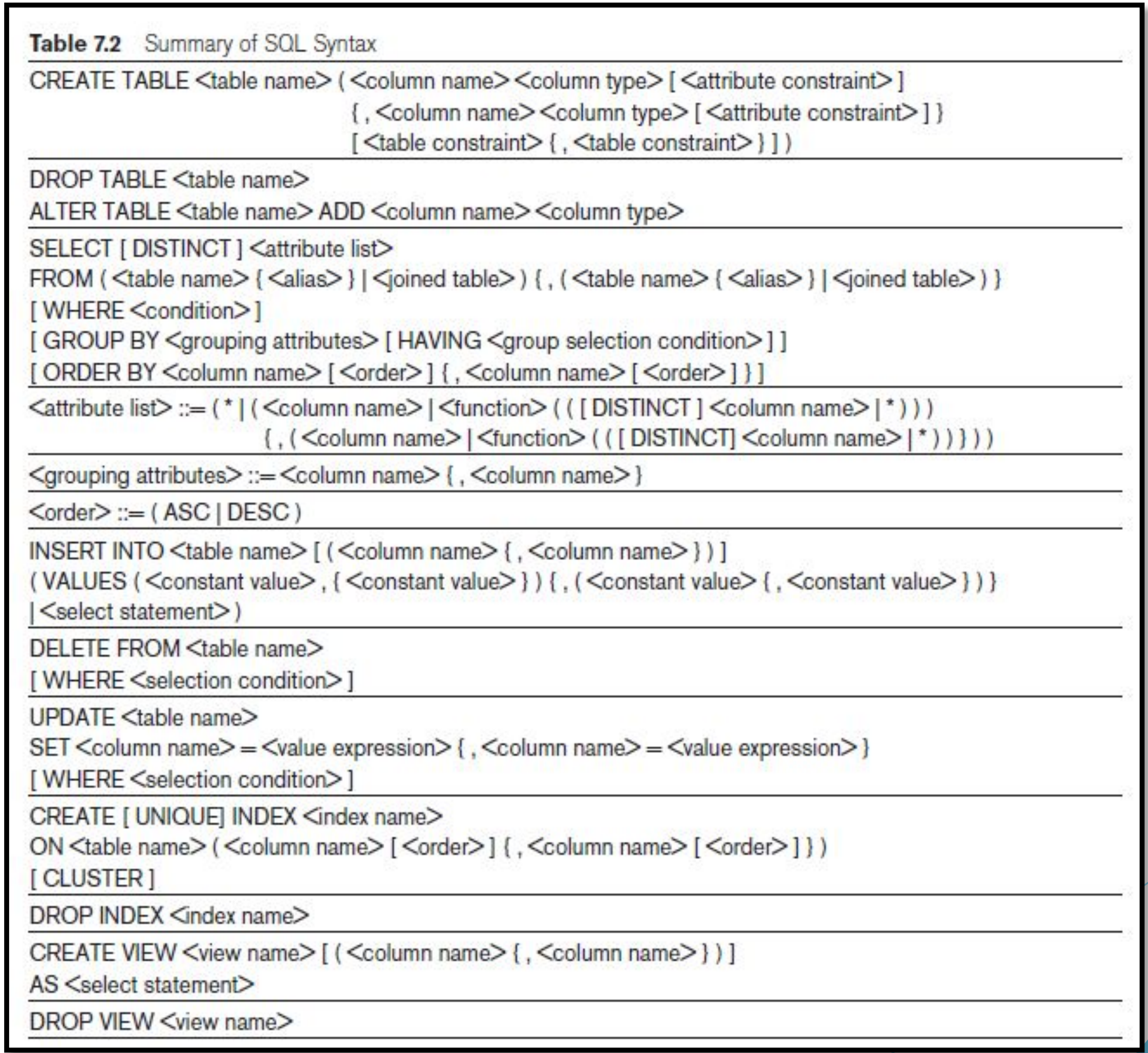


Table 7.2 Summary of SQL Syntax

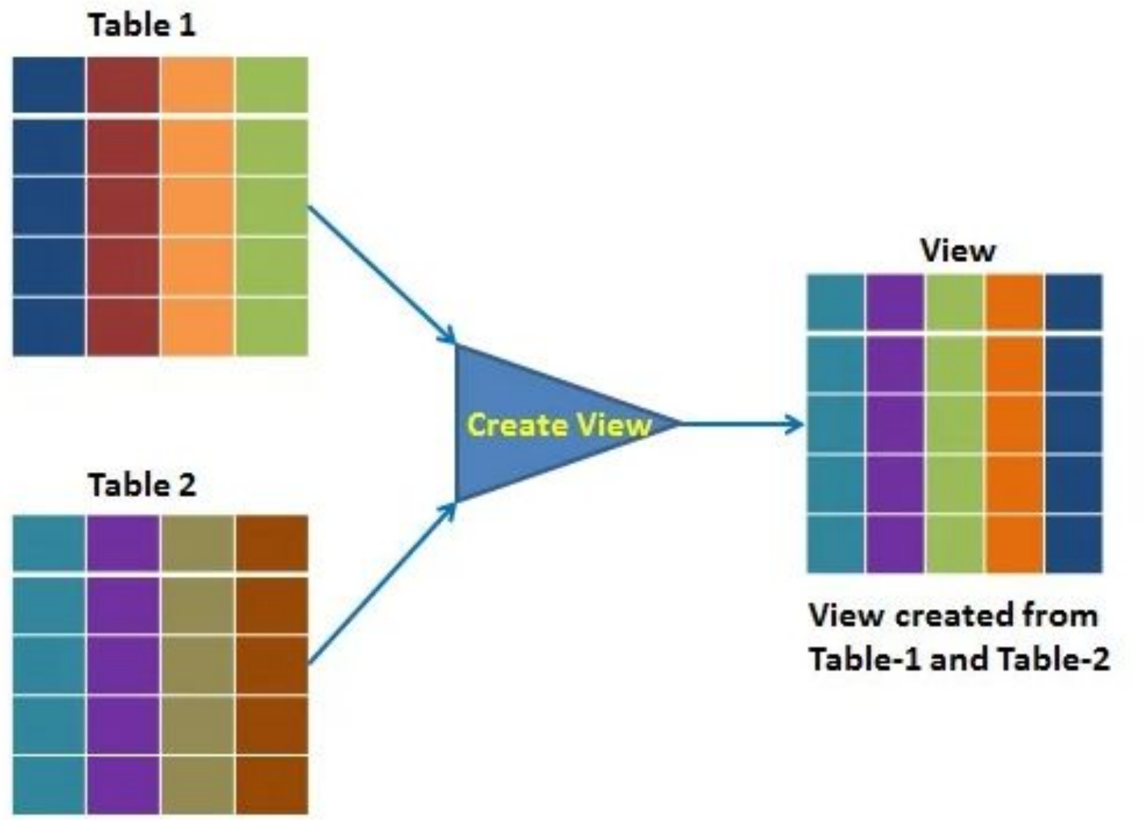
```
CREATE TABLE <table name> ( <column name> <column type> [ <attribute constraint> ]
                            { , <column name> <column type> [ <attribute constraint> ] }
                            [ <table constraint> { , <table constraint> } ] )

DROP TABLE <table name>
ALTER TABLE <table name> ADD <column name> <column type>

SELECT [ DISTINCT ] <attribute list>
FROM ( <table name> { <alias> } | <joined table> ) { , ( <table name> { <alias> } | <joined table> ) }
[ WHERE <condition> ]
[ GROUP BY <grouping attributes> [ HAVING <group selection condition> ] ]
[ ORDER BY <column name> [ <order> ] { , <column name> [ <order> ] } ]

<attribute list> ::= ( * | ( <column name> | <function> ( ( [ DISTINCT ] <column name> | * ) ) )
                     { , ( <column name> | <function> ( ( [ DISTINCT] <column name> | * ) ) } ) )

<grouping attributes> ::= <column name> { , <column name> }

<order> ::= ( ASC | DESC )

INSERT INTO <table name> [ ( <column name> { , <column name> } ) ]
( VALUES ( <constant value> , { <constant value> } ) { , ( <constant value> { , <constant value> } ) }
| <select statement> )

DELETE FROM <table name>
[ WHERE <selection condition> ]

UPDATE <table name>
SET <column name> = <value expression> { , <column name> = <value expression> }
[ WHERE <selection condition> ]

CREATE [ UNIQUE] INDEX <index name>
ON <table name> ( <column name> [ <order> ] { , <column name> [ <order> ] } )
[ CLUSTER ]

DROP INDEX <index name>

CREATE VIEW <view name> [ ( <column name> { , <column name> } ) ]
AS <select statement>

DROP VIEW <view name>
```

Views (Virtual Tables) in SQL

- *Concept of a View in SQL*
- A **view** is a single table that is derived from **other tables** (that could be base tables or previously defined views).
- A view are a **virtual table** as it is not stored physically in a database.
  - Can’t use DDL queries on views, however, DML queries can be used
  - you can query from a view but can’t do all update operations on that.
- Used when we have to create reference for frequently accessed data from multiple tables
  - You can select data from multiple tables,
  - or you can select specific data based on certain criteria in views.

Views (Virtual Tables) in SQL

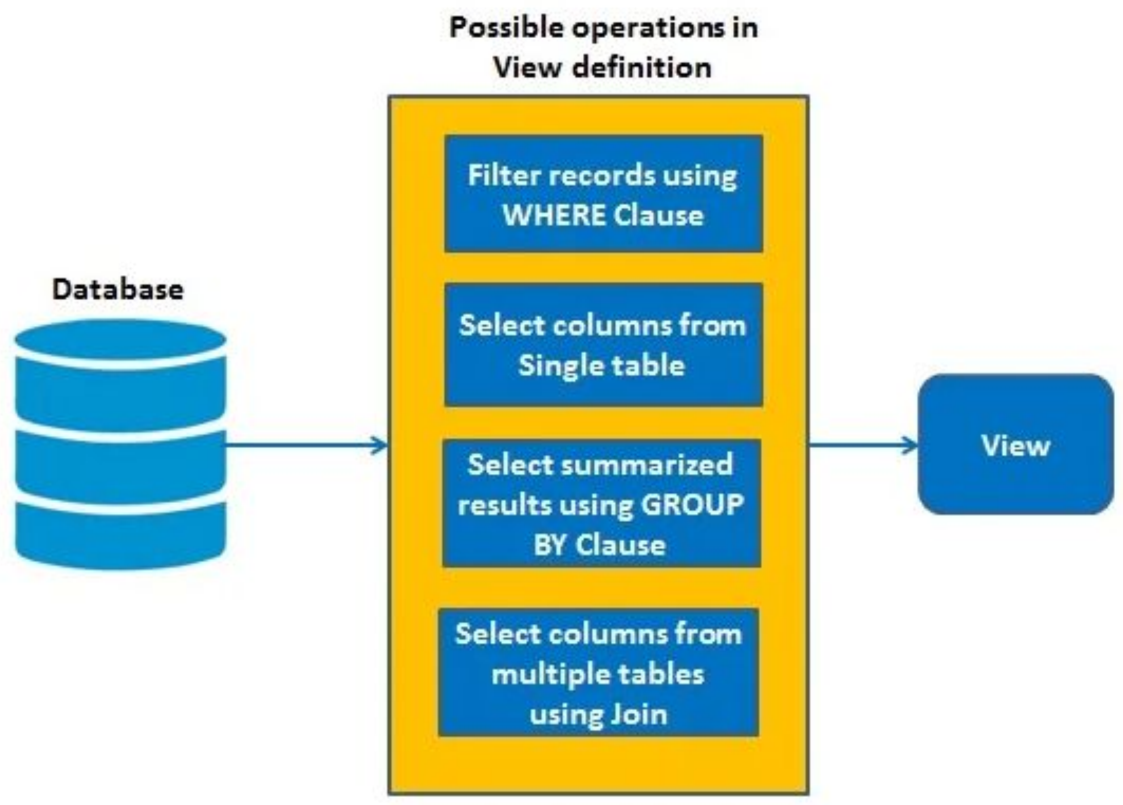
- ***Concept of a View in SQL***
- The view is a **query stored in the data dictionary, on which the user can query just like they do on tables.**
- It does not use the physical memory, **only the query is stored in the data dictionary.**
- It is **computed dynamically**, whenever the user performs any query on it.
- **Changes made at any point in view are reflected in the actual base table.**



View created from Table-1 and Table-2

Views (Virtual Tables) in SQL

- ***Concept of a View in SQL***
- The view has primarily two purposes:
  - Simplify the complex SQL queries.
  - Provide restriction to users from accessing sensitive data.

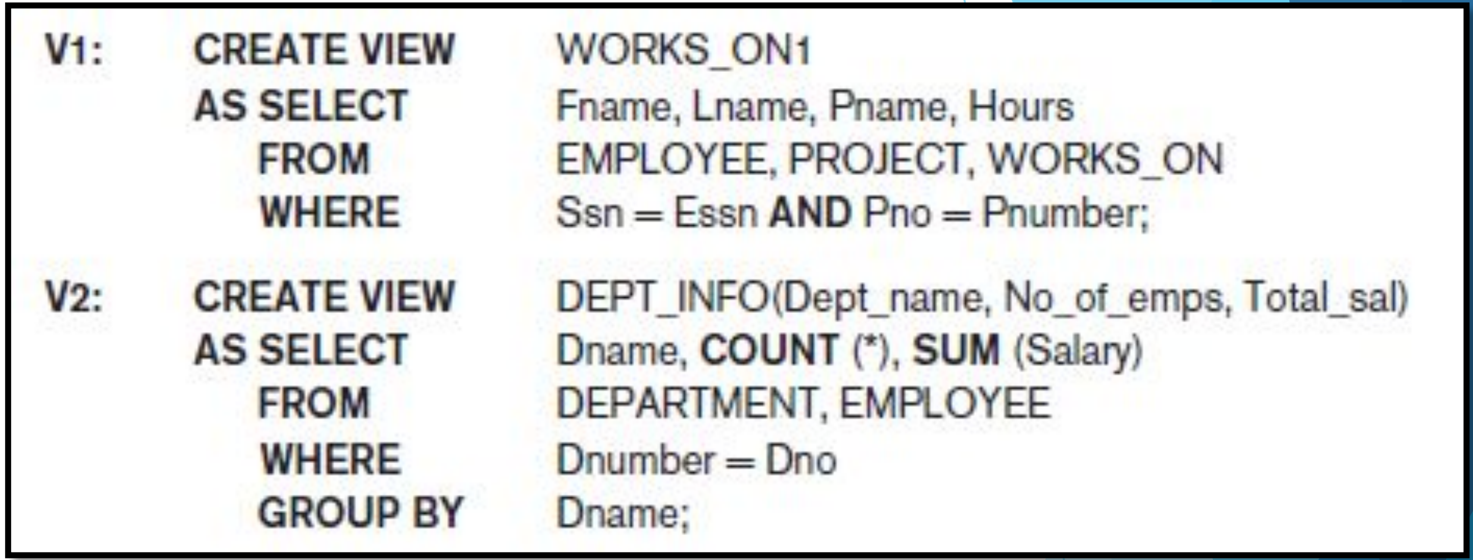

Possible operations in View definition
Database
Filter records using WHERE Clause
Select columns from Single table
Select summarized results using GROUP BY Clause
Select columns from multiple tables using Join
View

Views (Virtual Tables) in SQL

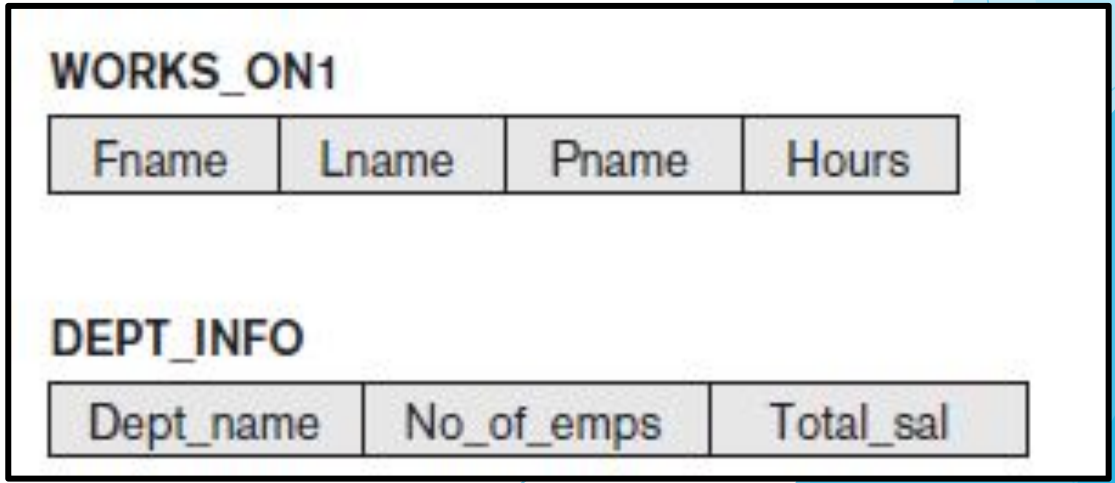
- *Concept of a View in SQL*
- For example, referring to the COMPANY database, we may frequently issue **queries that retrieve the employee name and the project names that the employee works on.**
- **STEPS TO BE FOLLOWED:**
  - **specify the join of the three tables EMPLOYEE, WORKS_ON, and PROJECT**
  - **define a view that is specified as the result of these joins.**
  - issue queries on the view, which are specified as single table retrievals rather than as retrievals involving two joins on three tables.
- **DEFINING TABLES OF VIEW:** tables used in view e.g, EMPLOYEE,PROJECT,WORKS_ON .

Views (Virtual Tables) in SQL

- ***Specification of Views in SQL***
- In SQL, the command to specify a view is **CREATE VIEW**.
- Create view contains:
  - **table name** (or **view name**),
  - a list of **attribute names**,
  - and **a query to specify the contents of the view.**
- We could **give names for attributes in view.** If not, then by default the names mentioned in base tables are used
  - See the difference in V1 and V2 query.



```
V1:   CREATE VIEW   WORKS_ON1
      AS SELECT     Fname, Lname, Pname, Hours
         FROM       EMPLOYEE, PROJECT, WORKS_ON
         WHERE      Ssn = Essn AND Pno = Pnumber;

V2:   CREATE VIEW   DEPT_INFO(Dept_name, No_of_emps, Total_sal)
      AS SELECT     Dname, COUNT (*), SUM (Salary)
         FROM       DEPARTMENT, EMPLOYEE
         WHERE      Dnumber = Dno
         GROUP BY   Dname;
```



WORKS_ON1

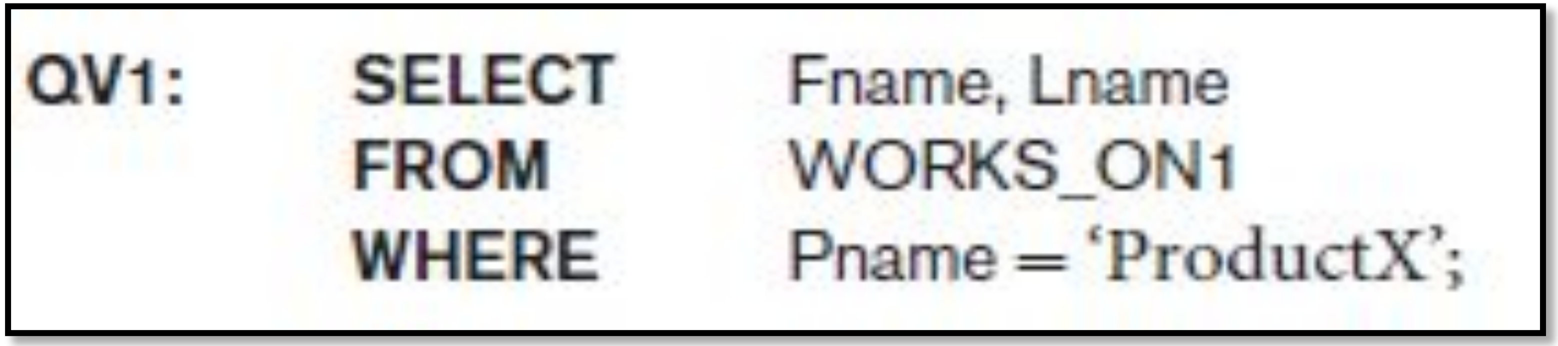
Fname	Lname	Pname	Hours

DEPT_INFO

Dept_name	No_of_emps	Total_sal

Views (Virtual Tables) in SQL

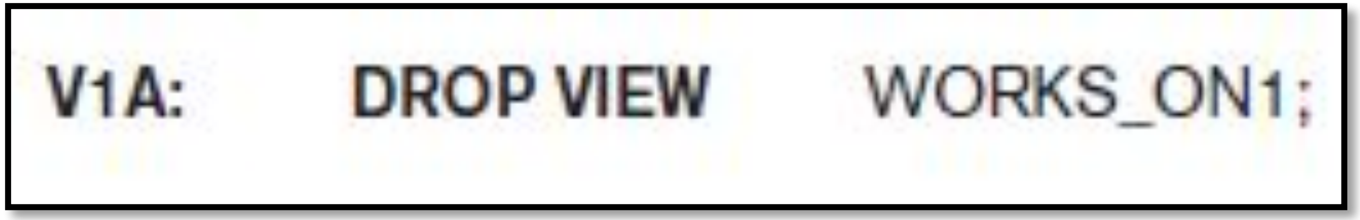
- ***Specification of Views in SQL - getting data from views***
- For example, **to retrieve the last name and first name of all employees who work on the ‘ProductX’ project**, we can utilize the WORKS_ON1 view and specify the query as in QV1:
- If this query is specified on base table, then join between 3 tables is needed for fulfilling the query.
  - Simplifies complex queries
  - Ensure security



```
QV1:    SELECT    Fname, Lname
        FROM      WORKS_ON1
        WHERE     Pname = ‘ProductX’;
```

Views (Virtual Tables) in SQL

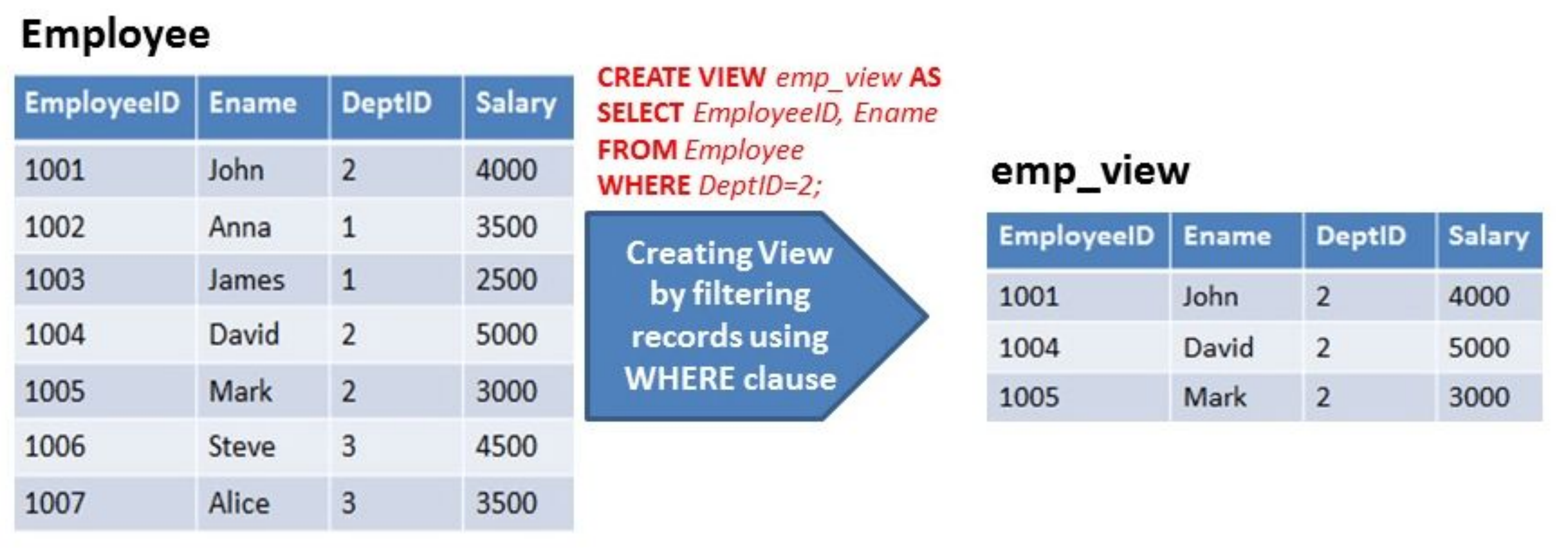
- ***Specification of Views in SQL***
- A view is supposed to be always up-to-date; **if we modify the tuples in the base** tables on which the view is defined, **the view must automatically reflect these changes**.
- Hence, the **view does not have to be realized or materialized at the time of view definition but rather at the time when we specify a query on the view.**
- If we do not need a view anymore, we can use the **DROP VIEW command** to dispose of it.
- For example, to get rid of the view V1, we can use the SQL statement in V1A:



```
V1A:     DROP VIEW     WORKS_ON1;
```

Views (Virtual Tables) in SQL

- ***Types of Views***
- Simple View: A view based on only a single table, which doesn't contain GROUP BY clause and any functions.



Employee

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

```
CREATE VIEW emp_view AS
SELECT EmployeeID, Ename
FROM Employee
WHERE DeptID=2;
```

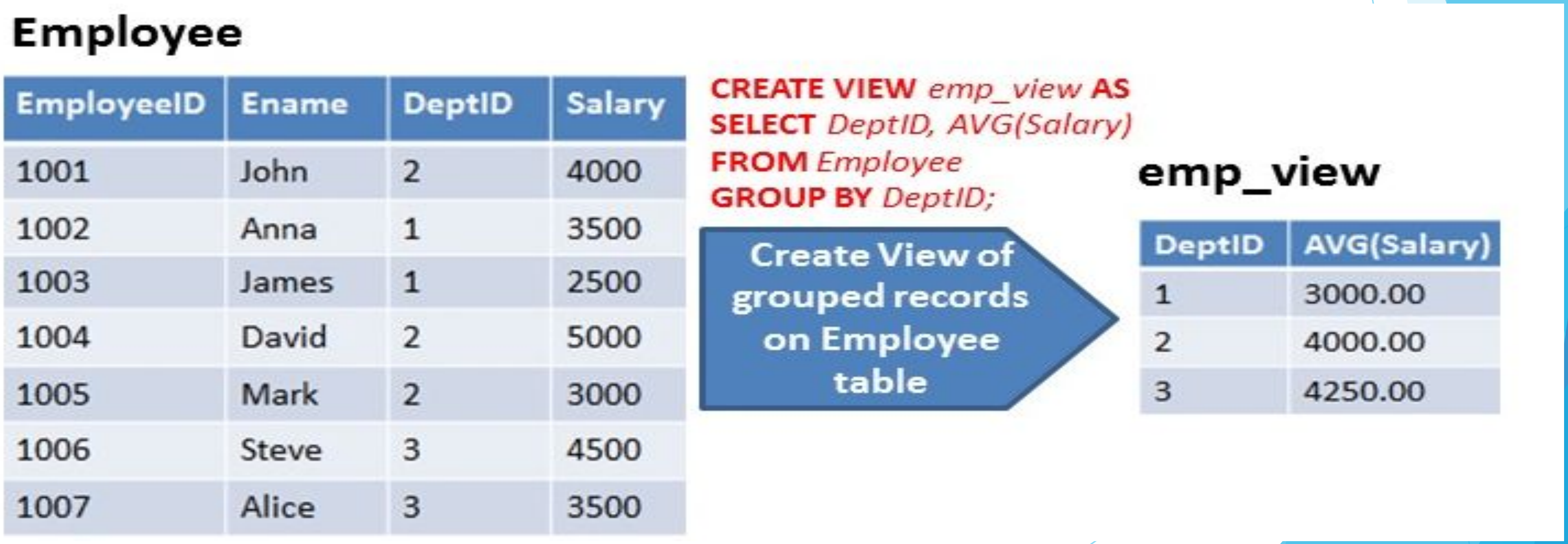
Creating View by filtering records using WHERE clause

emp_view

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1004	David	2	5000
1005	Mark	2	3000

Views (Virtual Tables) in SQL

- ***Types of Views***
- Complex View: A view based on data drawn from multiple tables, which may also contain GROUP BY clause and functions.



Employee

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

```
CREATE VIEW emp_view AS
SELECT DeptID, AVG(Salary)
FROM Employee
GROUP BY DeptID;
```

Create View of grouped records on Employee table

emp_view

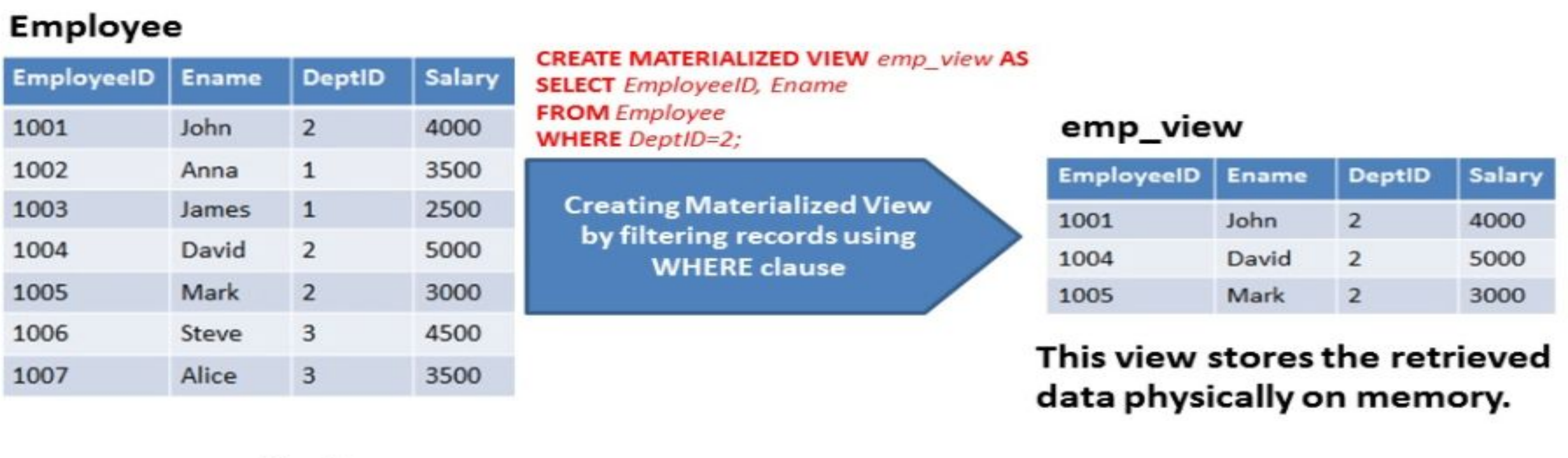
DeptID	AVG(Salary)
1	3000.00
2	4000.00
3	4250.00

Views (Virtual Tables) in SQL

- ***Types of Views***
- Materialized View
  - Materialized view **replicates the retrieved data physically**.
  - This replicated data can be reused without executing the view again.
  - **also known as "SNAPSHOTS".**
  - **For data warehousing**
- **Advantage: reduce the processing time to regenerate the whole data.**
- **Challenge: synchronize the changes done by views in underlying table**. Data consistency between view copy and physical table copy.

Views (Virtual Tables) in SQL

- ***Types of Views -*** Materialized View



Employee

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

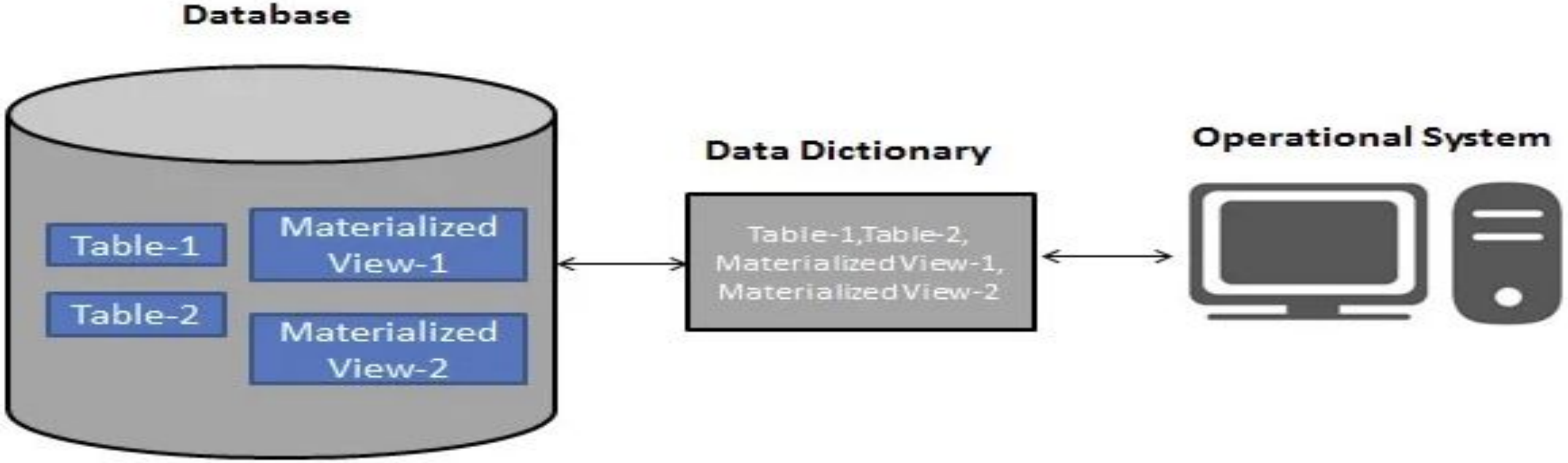
```
CREATE MATERIALIZED VIEW emp_view AS
SELECT EmployeeID, Ename
FROM Employee
WHERE DeptID=2;
```

Creating Materialized View by filtering records using WHERE clause

emp_view

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1004	David	2	5000
1005	Mark	2	3000

This view stores the retrieved data physically on memory.

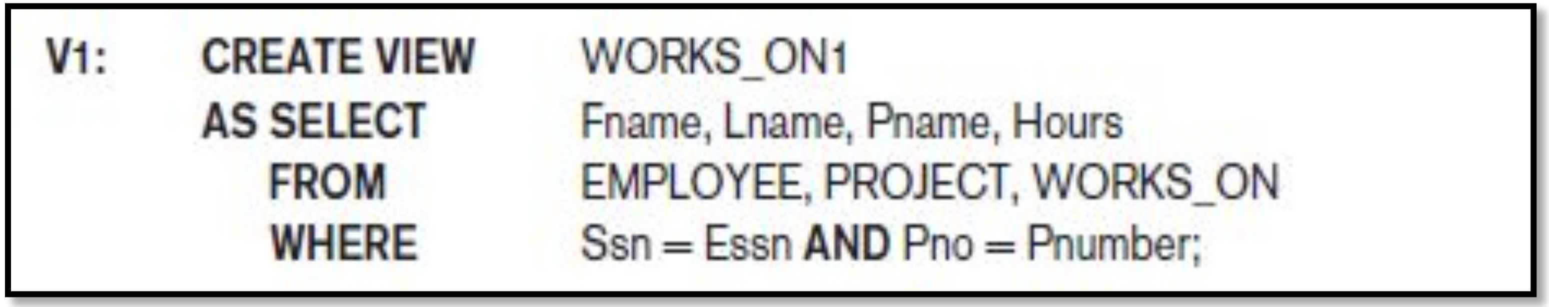


Views (Virtual Tables) in SQL

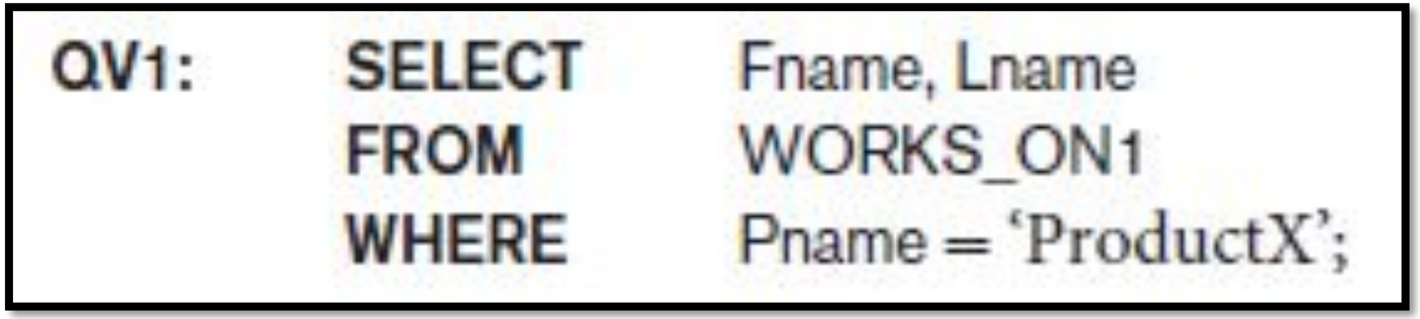
- *View Implementation, View Update, and Inline Views*
- **Problem:** How a DBMS can implement a view for efficient querying?
- **Two solutions**:
  - Query modification
  - View materialization
- **query modification -** modify the view query (submitted by the user) into a query on the underlying base tables.
- **Disadvantage:**
  - inefficient for views defined via complex queries that are time-consuming to execute, especially if multiple view queries are going to be applied to the same view within a short period of time.

Views (Virtual Tables) in SQL

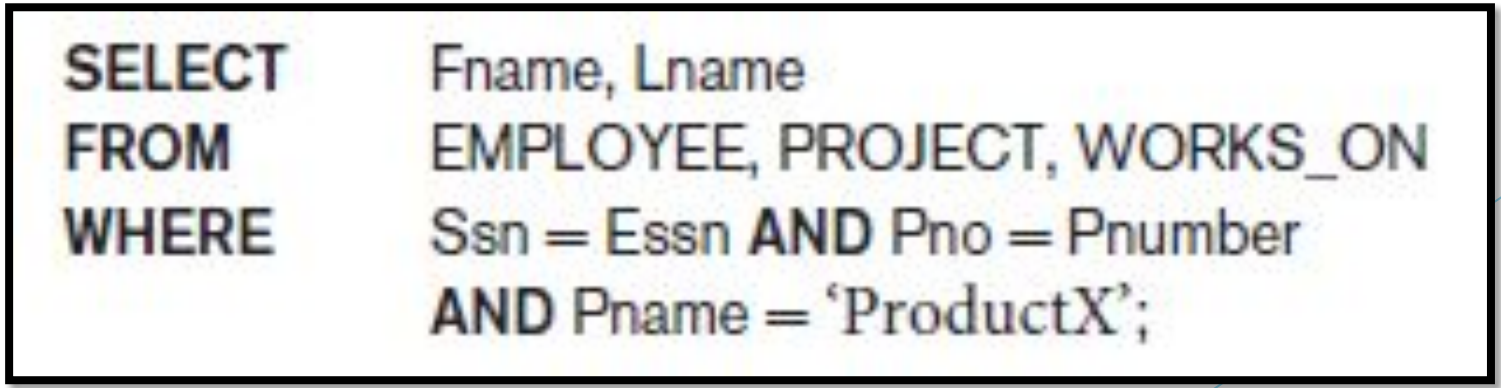
- ***View Implementation, View Update, and Inline Views***
- For example, the query QV1 would be automatically modified to the following query by the DBMS:



```
V1:     CREATE VIEW     WORKS_ON1
        AS SELECT       Fname, Lname, Pname, Hours
           FROM         EMPLOYEE, PROJECT, WORKS_ON
           WHERE        Ssn = Essn AND Pno = Pnumber;
```



```
QV1:    SELECT     Fname, Lname
        FROM       WORKS_ON1
        WHERE      Pname = 'ProductX';
```



```
SELECT     Fname, Lname
FROM       EMPLOYEE, PROJECT, WORKS_ON
WHERE      Ssn = Essn AND Pno = Pnumber
           AND Pname = 'ProductX';
```

Views (Virtual Tables) in SQL

- ***View Implementation, View Update, and Inline Views***
- **view materialization:**
  - physically **creates a permanent view table when the view is first created**
  - and keeps that **table on the assumption that other queries on the view will follow.**
- Since a copy of base table is created, so **for data consistency, automatically updating the base tables is much needed** because the view should contain the up to date data
  - Techniques using the concept of **incremental update** have been developed for this purpose,
  - where the **DBMS can determine what new tuples must be inserted, deleted, or modified in a materialized view table when a database update is applied to one of the defining base tables.**

Views (Virtual Tables) in SQL

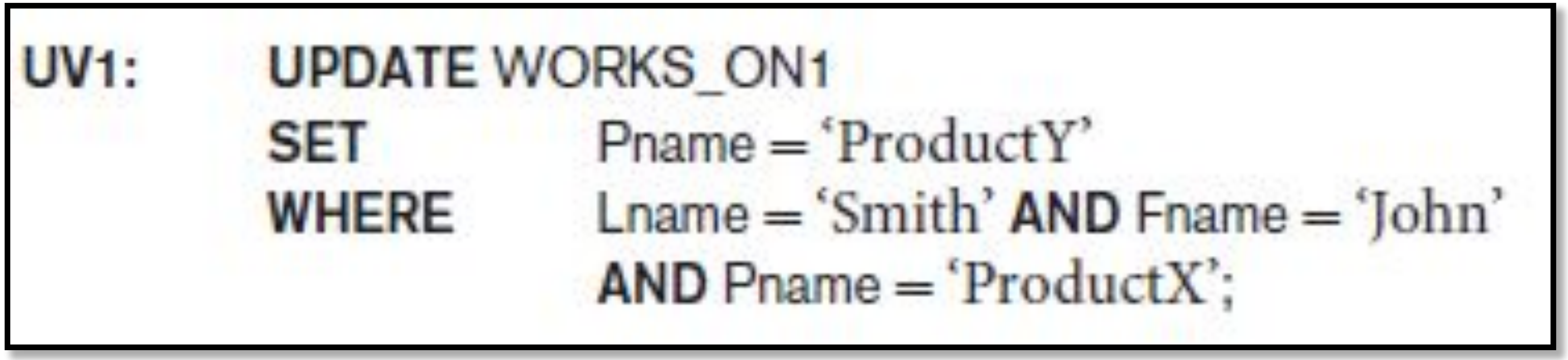
- ***View Implementation, View Update, and Inline Views***
- The view is generally kept as a materialized (physically stored) table as long as it is being queried.
- If the view is not queried for a certain period of time, the system may then automatically remove the physical table and re compute it from scratch when future queries reference the view.
- Strategies to update the materialized view:
- 1. **Immediate update strategy:** updates a view as soon as the base tables are changed.
- 2. **Lazy update strategy:** updates the view when needed by a view query.
- 3. **Periodic update strategy:** updates the view periodically (it is possible that a view query may get a result that is not up-to-date).

Views (Virtual Tables) in SQL

- ***View Implementation, View Update, and Inline Views***
- **No restriction for retrieval query** against any view.
- But **update queries are in many cases not possible**.
- **An update on a simple view ≈ multiple updates on the underlying base table under certain conditions.**
- **An update on a complex view ≈ multiple updates on the multiple underlying base tables under certain conditions.**

Views (Virtual Tables) in SQL

- ***View Implementation, View Update, and Inline Views***
- To illustrate potential problems with updating a view defined on multiple tables, consider the WORKS_ON1 view, and suppose that we issue the command to **update the PNAME attribute of ‘John Smith’ from ‘ProductX’ to ‘ProductY’.**
- This view update is shown in UV1:

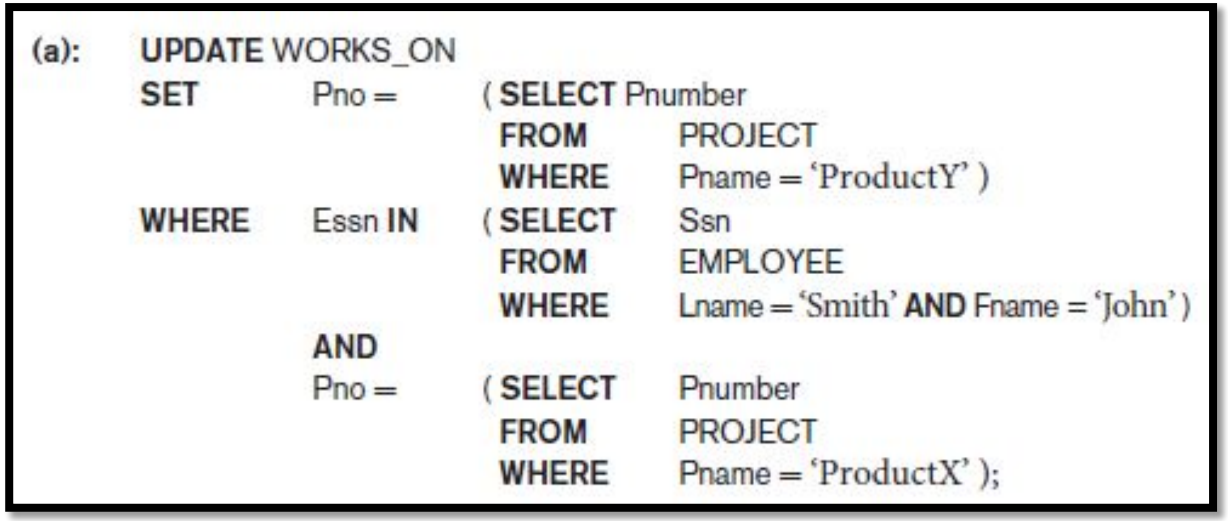


```
UV1:    UPDATE WORKS_ON1
        SET       Pname = ‘ProductY’
        WHERE     Lname = ‘Smith’ AND Fname = ‘John’
                  AND Pname = ‘ProductX’;
```

- **This query can be mapped into several updates on the base relations to give the desired update effect on the view.**

Views (Virtual Tables) in SQL

- *View Implementation, View Update, and Inline Views*
- *METHOD A:*



```
(a):    UPDATE WORKS_ON
        SET       Pno =      ( SELECT Pnumber
                               FROM     PROJECT
                               WHERE    Pname = 'ProductY' )
        WHERE     Essn IN    ( SELECT   Ssn
                               FROM     EMPLOYEE
                               WHERE    Lname = 'Smith' AND Fname = 'John' )
                  AND
                  Pno =      ( SELECT   Pnumber
                               FROM     PROJECT
                               WHERE    Pname = 'ProductX' );
```

- Update (a) relates 'John Smith' to the 'ProductY' and is the most likely desired update.
- Firstly get the ESSN number of employee and the Pno of product, then update the value t the value of productY

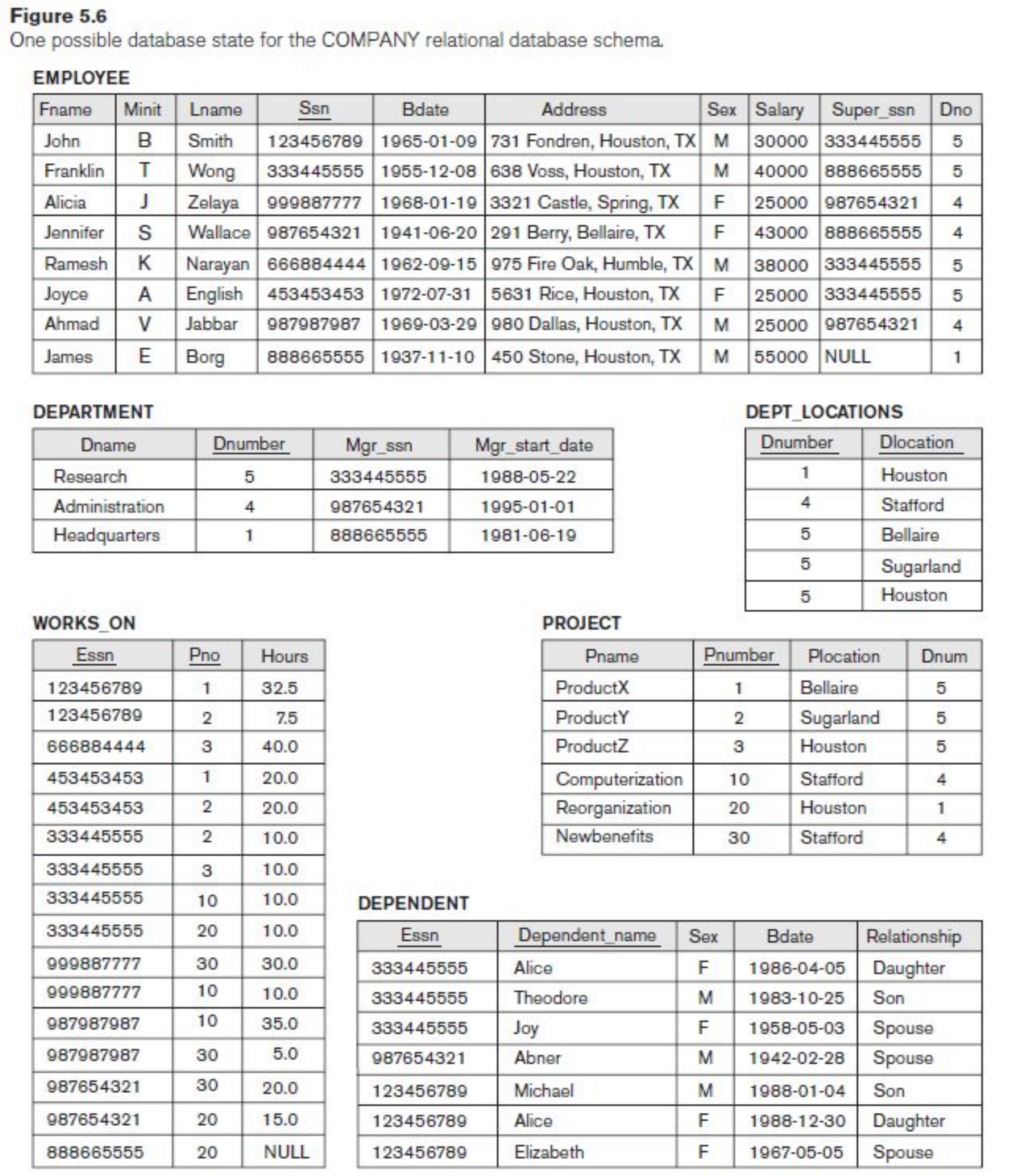


Figure 5.6
One possible database state for the COMPANY relational database schema.

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

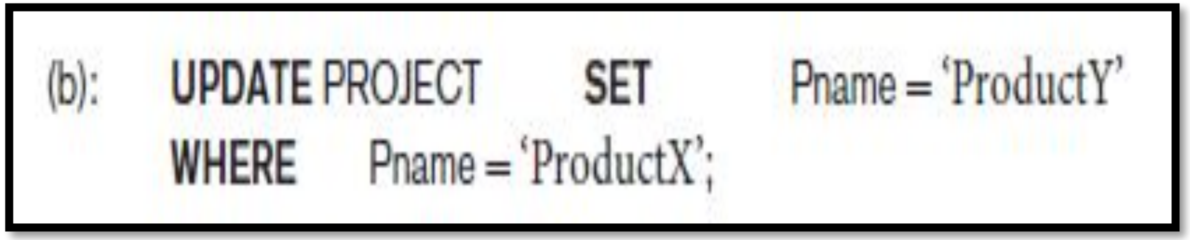
Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

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

Views (Virtual Tables) in SQL

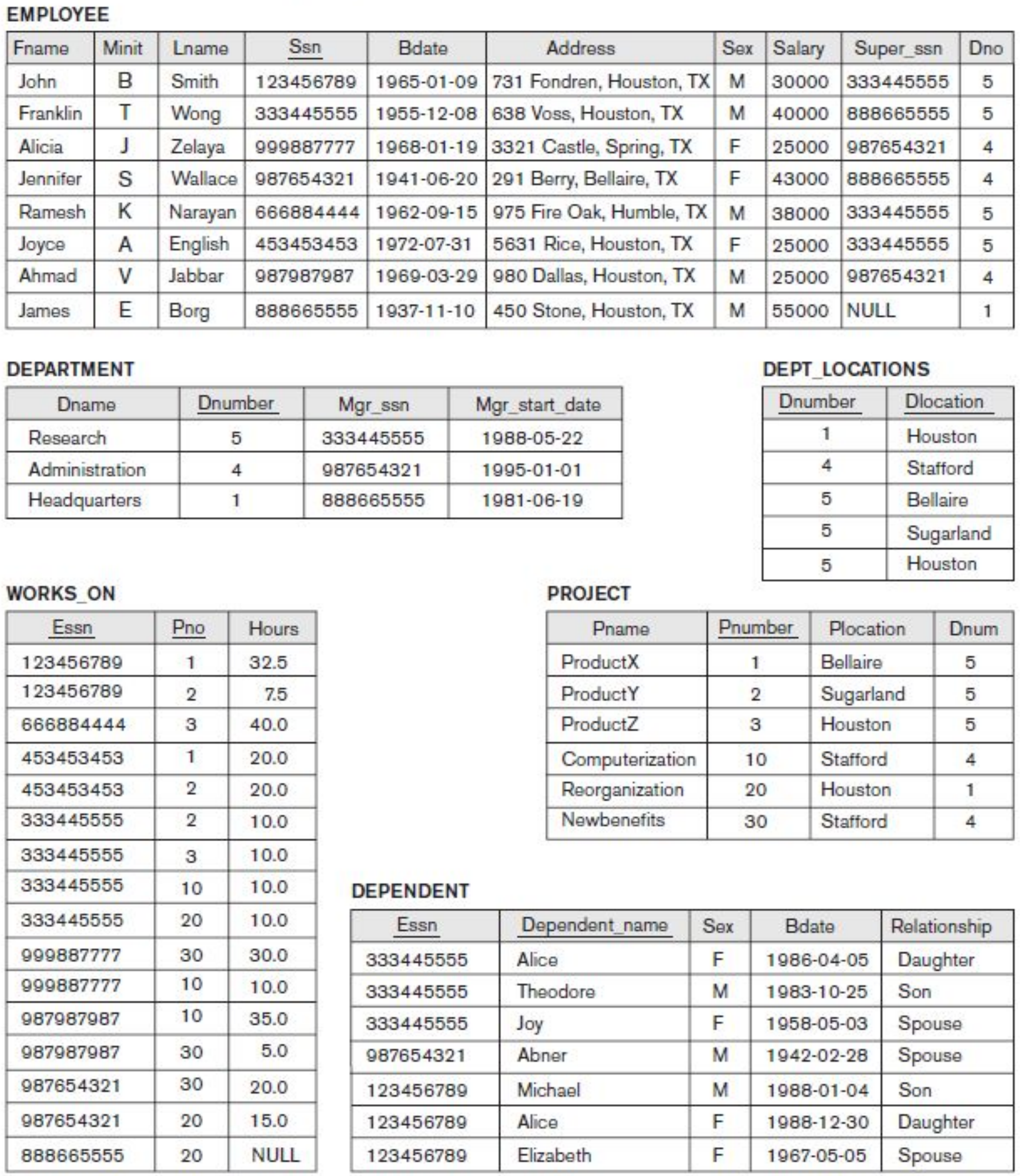
- ***View Implementation, View Update, and Inline Views***
- ***METHOD B:***



```
(b):    UPDATE PROJECT    SET      Pname = 'ProductY'
        WHERE    Pname = 'ProductX';
```

- However, **(b) would also give the desired update effect on the view**, but it **accomplishes this by changing the name of the 'ProductX' tuple in the PROJECT relation to 'ProductY'.**
- It is an **incorrect approach to user query** because it also has the side effect of changing all the view tuples with Pname = 'ProductX'.

Figure 5.6
One possible database state for the COMPANY relational 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

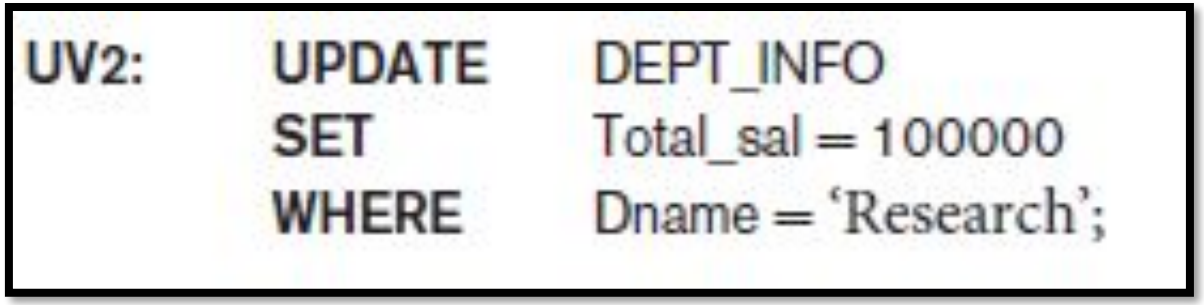
Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

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

Views (Virtual Tables) in SQL

- ***View Implementation, View Update, and Inline Views***
- **Some view updates may not make much sense;**
- for example, **modifying the Total_sal attribute of the DEPT_INFO** view does not make sense because Total_sal is defined to be the sum of the individual employee salaries.
- This **incorrect request** is shown as UV2:



```
UV2:    UPDATE    DEPT_INFO
        SET       Total_sal = 100000
        WHERE     Dname = 'Research';
```

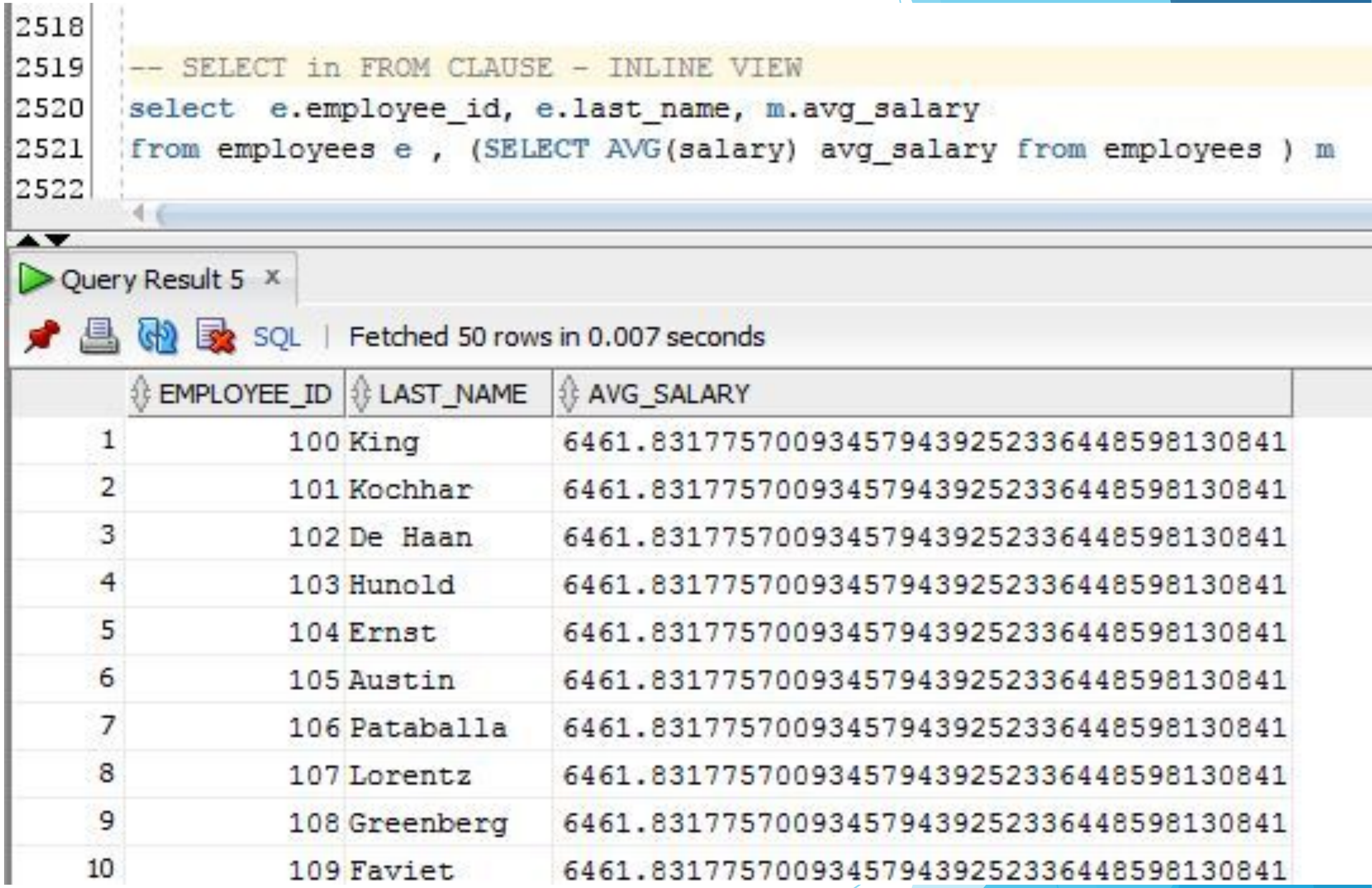
- **Update operations are restricted whenever an update on the view can be mapped to more than one update on the underlying base relations**.

Views (Virtual Tables) in SQL

- *View Implementation, View Update, and Inline Views*
- *When can views be updated or not?*
- *Updatable:*
- A view with a **single defining table** **if** the view attributes contain the primary key of the base relation, as well as **all attributes with the NOT NULL constraint with no default values specified.**
- *Not updatable:*
  - Views **defined on multiple tables using joins.**
  - Views **defined using grouping and aggregate functions.**
  - The view **having any DISTINCT clause** in its definition.

Views (Virtual Tables) in SQL

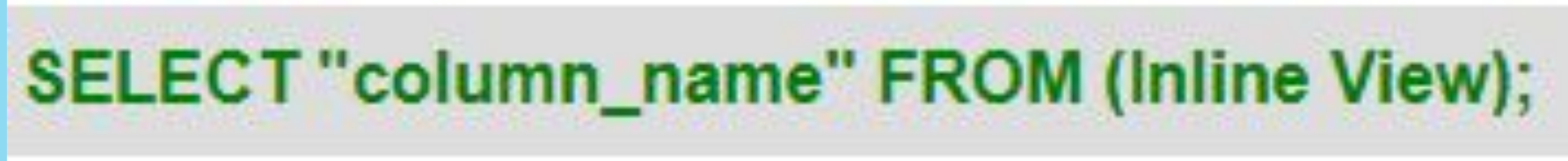
- ***View Implementation, View Update, and Inline Views***
- **Inline Views:**
- A view table in the FROM clause of an SQL query is known as an **in-line view.** Hence, the view is defined within the query itself.
- utilized for writing complex SQL queries without join and subqueries operations



```
-- SELECT in FROM CLAUSE - INLINE VIEW
select  e.employee_id, e.last_name, m.avg_salary
from employees e , (SELECT AVG(salary) avg_salary from employees ) m
```

Fetched 50 rows in 0.007 seconds

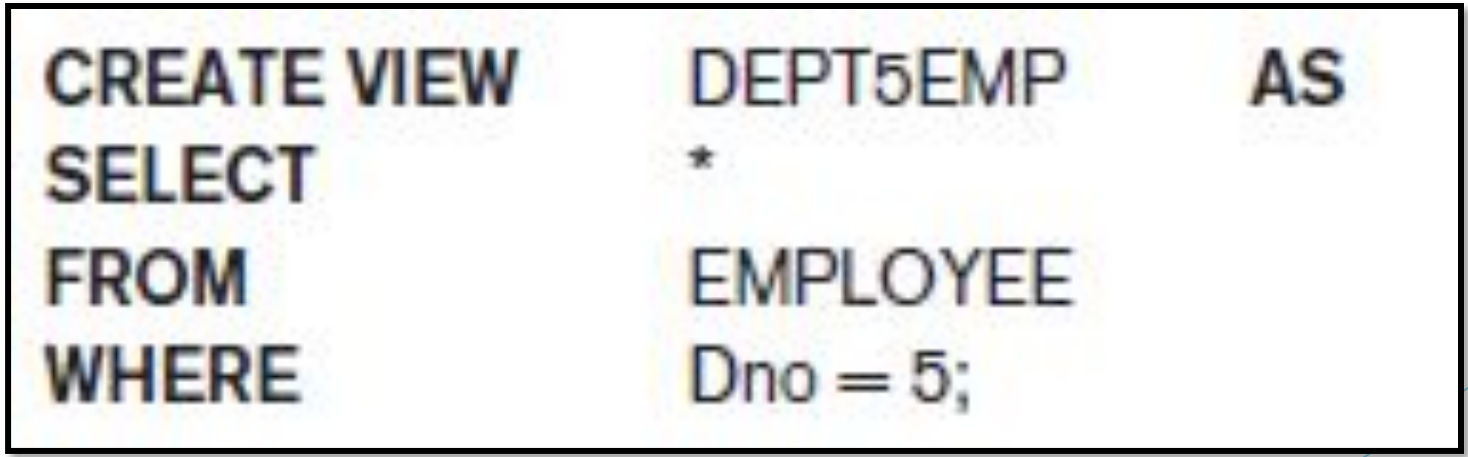
```
    EMPLOYEE_ID LAST_NAME  AVG_SALARY
 1          100 King       6461.83177570093457943925233644859813O841
 2          101 Kochhar    6461.831775700934579439252336448598130841
 3          102 De Haan    6461.831775700934579439252336448598130841
 4          103 Hunold     6461.831775700934579439252336448598130841
 5          104 Ernst      6461.831775700934579439252336448598130841
 6          105 Austin     6461.831775700934579439252336448598130841
 7          106 Pataballa  6461.831775700934579439252336448598130841
 8          107 Lorentz    6461.831775700934579439252336448598130841
 9          108 Greenberg  6461.831775700934579439252336448598130841
10          109 Faviet     6461.831775700934579439252336448598130841
```



SELECT "column_name" FROM (Inline View);

Views (Virtual Tables) in SQL

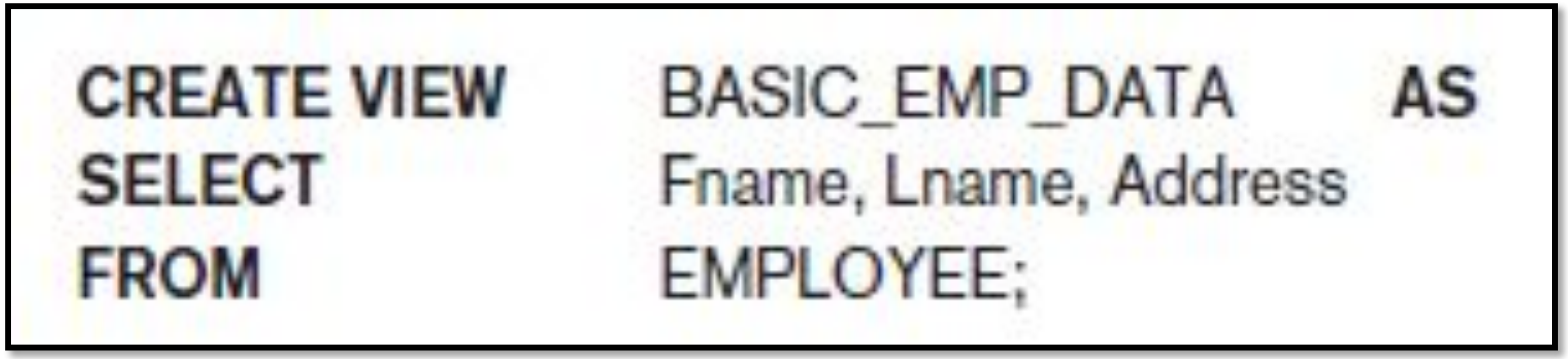
- ***Views as Authorization Mechanisms***
- Views could be defined for authorization.
- **Suppose a certain user is only allowed to see employee information for employees who work for department 5.**
- This user will only be able to retrieve employee information for employee tuples whose Dno = 5 and will not be able to see other employee tuples when the view is queried.



```
CREATE VIEW     DEPT5EMP     AS
SELECT          *
FROM            EMPLOYEE
WHERE           Dno = 5;
```

Views (Virtual Tables) in SQL

- ***Views as Authorization Mechanisms***
- In a similar manner, **a view can restrict a user to only see certain columns;** for example, only the first name, last name, and address of an employee may be visible as follows:



```
CREATE VIEW    BASIC_EMP_DATA        AS
SELECT         Fname, Lname, Address
FROM           EMPLOYEE;
```

- Thus by creating an appropriate view and granting certain users access to the view and not the base tables, they would be restricted to retrieving only the data specified in the view.

Views (Virtual Tables) in SQL

- **Advantages of Views**
- provide an abstraction to various users
- hide the complexity for users who are accessing data from multiple tables.
- simplify complex queries into a simpler one.
- Views can be used for security purposes
- does not require disk space
  - because only the definition is in the data dictionary, not the copy of actual data.

Views (Virtual Tables) in SQL

- **Disadvantages of Views**
  - More computation time.
  - Views have a dependency on the table structure.