COMP810 Data Warehousing and Big Data

Semester 2 2024

Dr Victor Miranda

COMP810

Week 6 Data Warehousing

- Complex OLAP operations - II & SQL queries

Week 5 – (summary) JOINS IN SQL

- An SQL ‘join’ is used fetch data from two or more tables, which is set to appear as a single set of data.

- Right outer join:
  - In practice, a ‘join’ combines columns from two or more table by using common identifiers (IDs) to both tables

- Keyword: JOIN … ON

Week 5 – (summary) JOINS IN SQL

- An SQL 'join' is used fetch data from two or more tables, which is set to appear as a single set of data.
- Right outer join:
  - In practice, a 'join' combines columns from two or more table by using common identifiers (IDs) to both tables
- Keyword: JOIN … ON
- NO LAB TODAY (Work on the DW project)

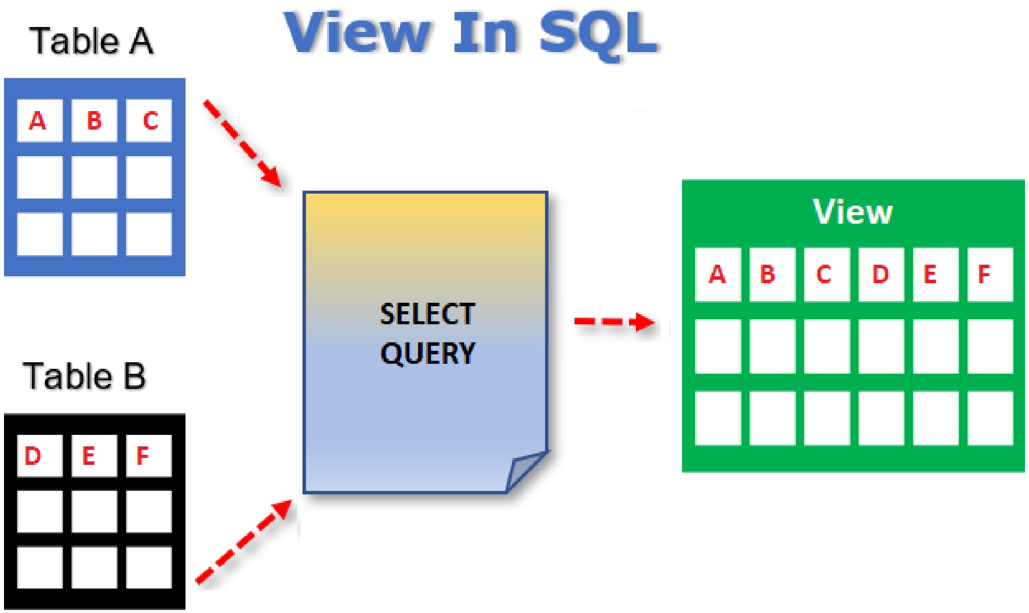
Types of ‘JOIN’

- Inner join:
  - Include all the rows from both tables
- Left outer join:
  - Include the left TABLE even if there’s no match.
- Right outer join:
  - Include the right TABLE even if there’s no match
- Full outer join:
  - Include the both left and right TABLES even if there’s no match
- The CROSS join

- Materialized Views
- Nested Queries
- CUBE OPERATOR
- ROLL UP

SQL Views

- In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table.



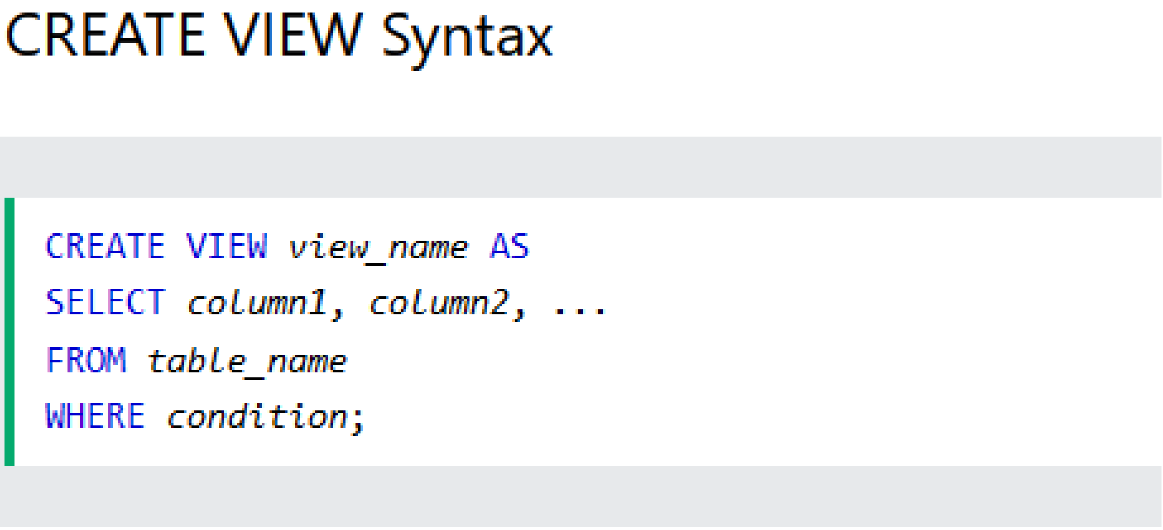
From Week 5 DW - Types of Views

- Virtual views:
  - Used in databases
  - Computed only on-demand – slow*er* at runtime
  - Always up to date

- Materialized views
  - Used in data warehouses
  - Precomputed offline – fast*er* at runtime
  - May have stale data

Defining Views - Syntax

Views are relations, except that they are not physically stored.



CREATE VIEW Syntax

```sql
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

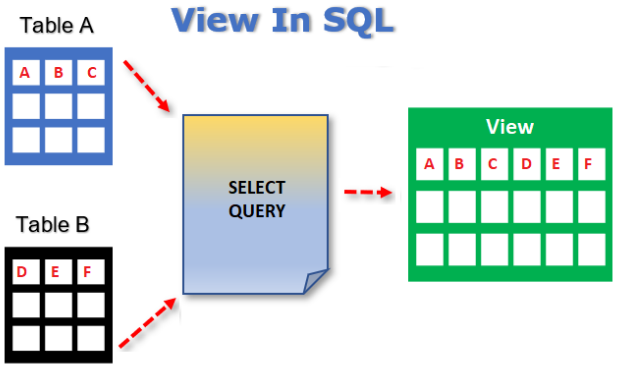
Defining Views - Syntax

Views are relations, except that they are not physically stored.

For presenting different information to different users

Employee(ssn, name, department, project, salary)

```
CREATE VIEW  Developers AS
  SELECT name, project
  FROM  Employee
  WHERE department = “Development”
```



Payroll has access to Employee, others only to Developers

Materialized Views

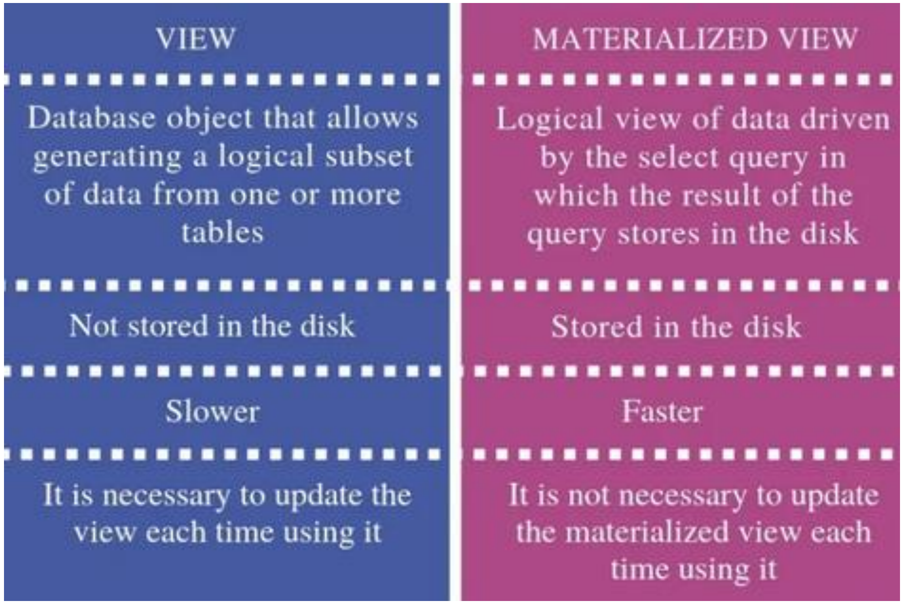
- In SQL, A materialized view are also logical virtual copies of data created by combining data from multiple existing tables for faster data retrieval.

  - Like a view, a materialized view contains rows and columns, just like a real table.

  - Unlike a view, the resulting materialized view will get stored on disk

Materialized Views

- ✓ These automatically get updated as data changes in the underlying tables.
- ✓ It improves the performance of complex queries (typically queries with joins and aggregations) while offering simple maintenance operations.



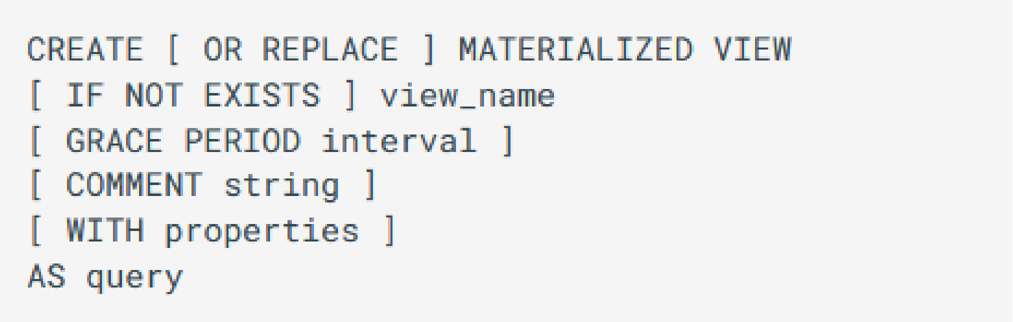
VIEW	MATERIALIZED VIEW
Database object that allows generating a logical subset of data from one or more tables	Logical view of data driven by the select query in which the result of the query stores in the disk
Not stored in the disk	Stored in the disk
Slower	Faster
It is necessary to update the view each time using it	It is not necessary to update the materialized view each time using it

Types of Materialized Views

- ✓ Read – only: This type of MVs cannot send data back to the server Master tables. These server only one way communication i.e. from server to the client.

- ✓ Updatable: This type of MVs can send the data, changed locally, back to the server (depends on the DBMS not on the user).

Syntax:



```
CREATE [ OR REPLACE ] MATERIALIZED VIEW
[ IF NOT EXISTS ] view_name
[ GRACE PERIOD interval ]
[ COMMENT string ]
[ WITH properties ]
AS query
```

Examples

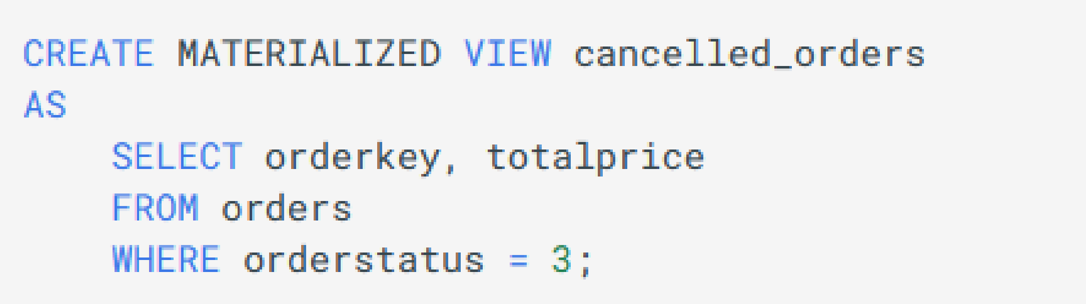
Create a simple materialized view `cancelled_orders` over the `orders` table that only includes cancelled orders. Note that `orderstatus` is a numeric value that is potentially meaningless to a consumer, yet the name of the view clarifies the content.

orders(oderkey, totalprice, orderstatus)

Examples

Create a simple materialized view `cancelled_orders` over the `orders` table that only includes cancelled orders. Note that `orderstatus` is a numeric value that is potentially meaningless to a consumer, yet the name of the view clarifies the content.

orders(oderkey, totalprice, orderstatus)



```
CREATE MATERIALIZED VIEW cancelled_orders
AS
    SELECT orderkey, totalprice
    FROM orders
    WHERE orderstatus = 3;
```

- Materialized Views
- Nested Queries
- CUBE OPERATOR
- ROLL UP

Nested Query

In SQL, a ‘query’ is an operation that retrieves data from a table in a database and always includes a SELECT statement.

A nested query is a complete query embedded within another operation.

Nested Query

In SQL, a ‘query’ is an operation that retrieves data from a table in a database and always includes a SELECT statement.

A nested query is a complete query embedded within another operation.

It can have all the elements used in a regular query, and any valid query can be embedded within another operation to become a nested query

NON – correlated Sub Queries

Non-Correlated Sub Queries:

- Requires data required by outer query before it can be executed
- Inner query does not contain any reference to outer query
- Behaves like a function

NON – correlated Sub Queries

Non-Correlated Sub Queries:

- Requires data required by outer query before it can be executed
- Inner query does not contain any reference to outer query
- Behaves like a function

Example:

People(person_fname, person_lname, person_id, person_state, person_city)

Movies(movie_id, movie_title, director_id, studio_id)

```
Select movie_title, studio_id
            From Movies
            Where director_id IN (
                        Select person_id
                        From People
                        Where person_state = 'TX')
```

Steps:

1) Subquery is executed
2) Subquery results are plugged into the outer query
3) The outer query is processed

Correlated Sub Queries

Correlated Sub Queries:

- Contains reference to the outer query
- Behaves like a loop, as it evaluates once for each tuple in the outer query

Common SQL functions for correlated queries

- `EXISTS` function
  - Check whether the result of a correlated nested query is empty or not. They are Boolean functions that return a TRUE or FALSE result.
- `EXISTS` and `NOT EXISTS`
  - Typically used in conjunction with a correlated nested query
- SQL function `UNIQUE(Q)`
  - Returns `TRUE` if there are no duplicate tuples in the result of query Q

Correlated Sub Queries

Correlated Sub Queries:

- Contains reference to the outer query
- Behaves like a loop, as it evaluates once for each tuple in the outer query

Example:

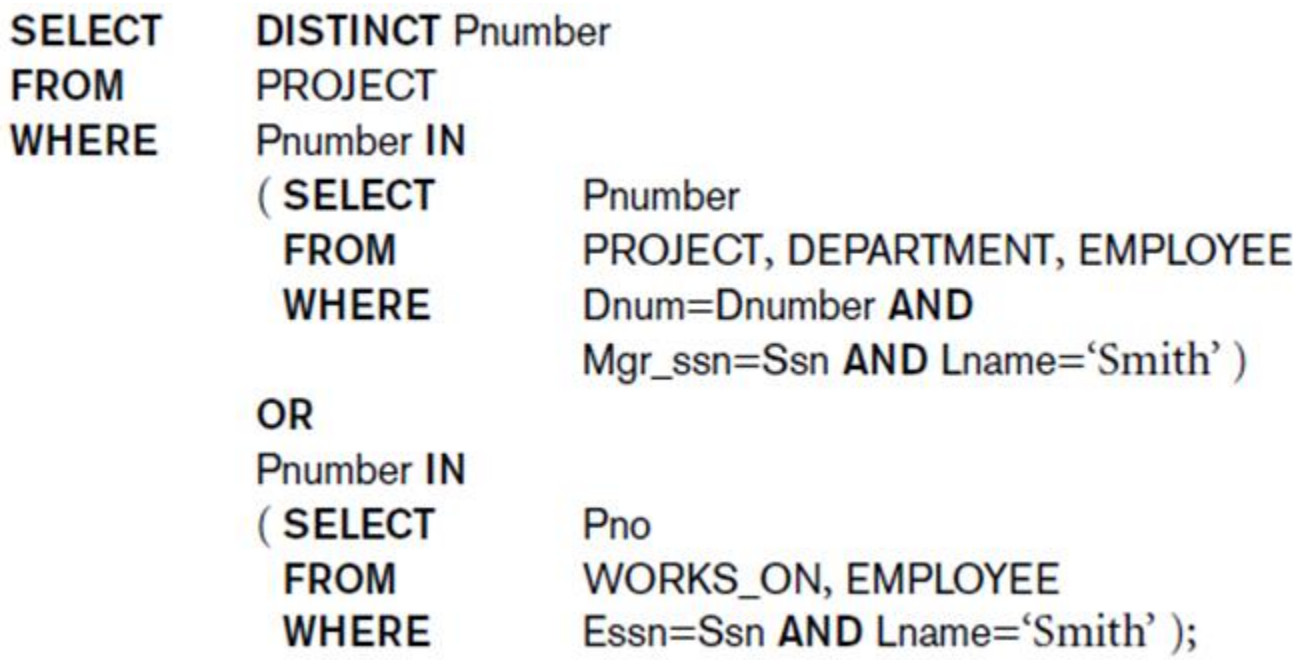
```
SELECT name, street, city, state FROM addresses

     WHERE EXISTS

             (SELECT * FROM states WHERE

                  states.state = addresses.state);
```

Steps:

- The query extracts and evaluates each `addresses.state` value (outer sq)
- Then the query – using EXISTS – checks the addresses in the inner(correlated) query
- The outer query is executed.

Example II

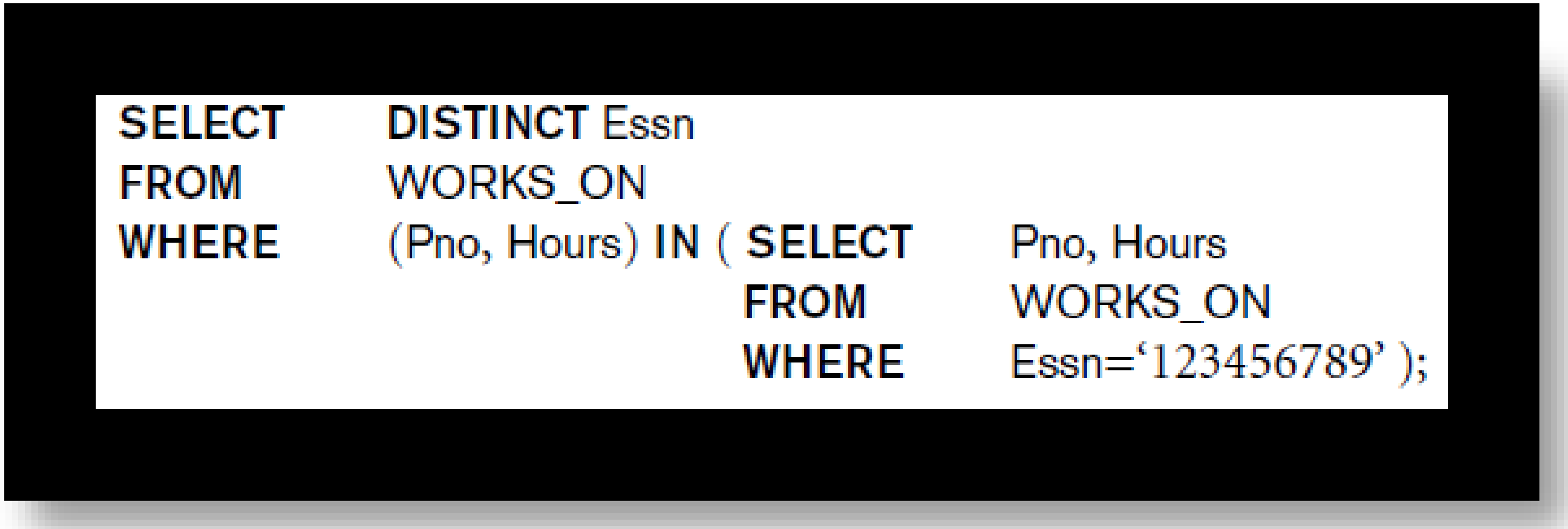
Make a list of all project numbers for projects that involve employee Smith either as worker or as a manager of the department that controls the project:



```
SELECT      DISTINCT Pnumber
FROM        PROJECT
WHERE       Pnumber IN
            ( SELECT        Pnumber
              FROM          PROJECT, DEPARTMENT, EMPLOYEE
              WHERE         Dnum=Dnumber AND
                            Mgr_ssn=Ssn AND Lname='Smith' )
            OR
            Pnumber IN
            ( SELECT        Pno
              FROM          WORKS_ON, EMPLOYEE
              WHERE         Essn=Ssn AND Lname='Smith' );
```

Example II

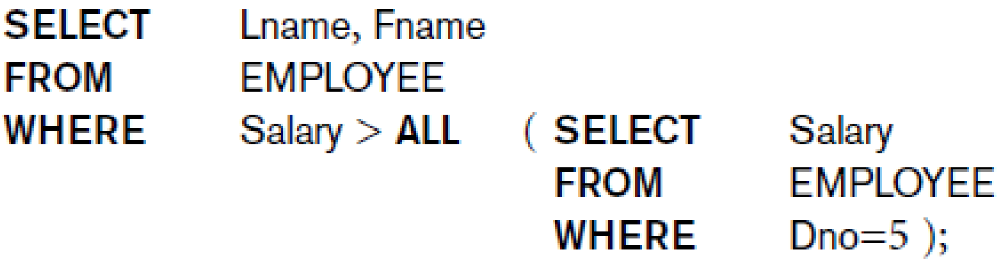
- Use tuples of values in comparisons
  - Place them within parentheses



```
SELECT     DISTINCT Essn
FROM       WORKS_ON
WHERE      (Pno, Hours) IN ( SELECT     Pno, Hours
                             FROM       WORKS_ON
                             WHERE      Essn='123456789' );
```

Operators – Nested Queries

- Use other comparison operators to compare values
  - = ANY (or = SOME) operator [equivalent to IN]
    - Returns TRUE if the value *v* is equal to some value in the set
  - Other operators that can be combined with ANY (or SOME): >, >=, <, <=, and <>
  - ALL: value must exceed all values from nested query



```
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     Salary > ALL   ( SELECT    Salary
                           FROM      EMPLOYEE
                           WHERE     Dno=5 );
```

General Form

- `= ANY` (or `= SOME`) operator [equivalent to IN]
  - Returns `TRUE` if the value *v* is equal to some value in the set

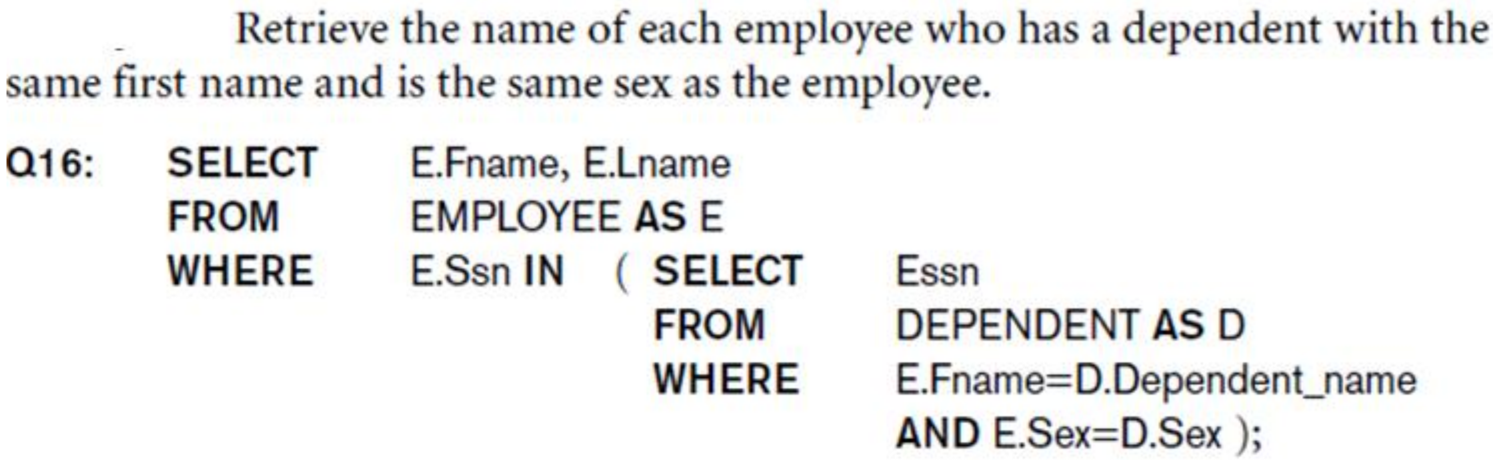
- Other operators that can be combined with `ANY` (or `SOME`): >, >=, <, <=, and <>

- `ALL:` value must exceed all values from nested query

```
SELECT [column_name ]
FROM [table_name]
WHERE expression operator
        {ALL | ANY | SOME} ( subquery )
```

Example - III

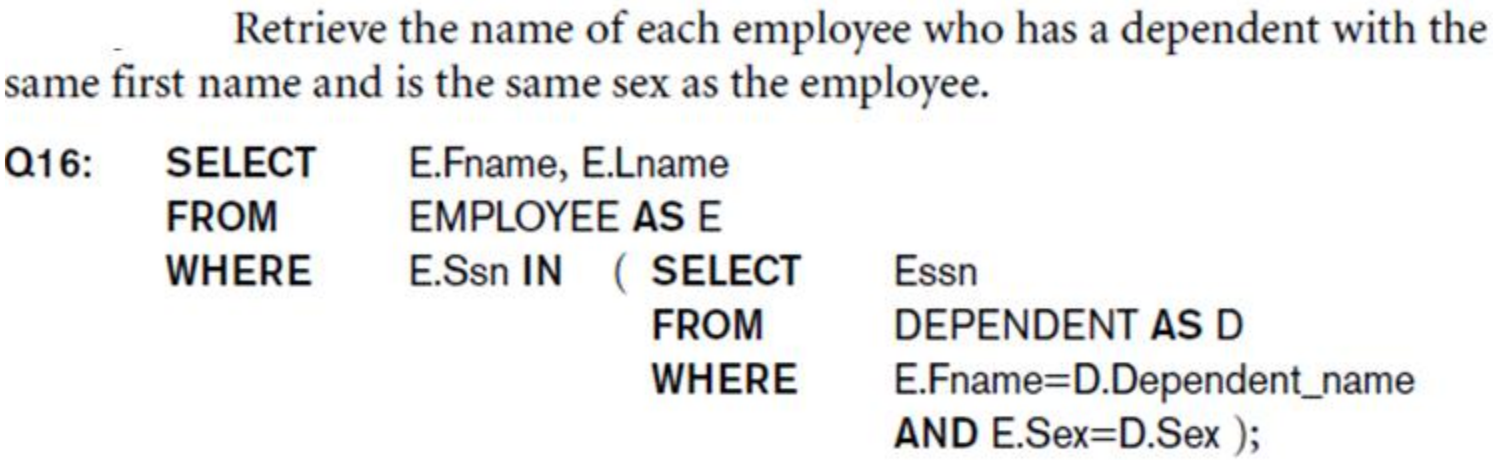
- Avoid potential errors and ambiguities
  - Create tuple variables (aliases) for all tables referenced in SQL query



Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
Q16:    SELECT   E.Fname, E.Lname
        FROM     EMPLOYEE AS E
        WHERE    E.Ssn IN  ( SELECT   Essn
                             FROM     DEPENDENT AS D
                             WHERE    E.Fname=D.Dependent_name
                                      AND E.Sex=D.Sex );
```

Example - III



Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
Q16:   SELECT   E.Fname, E.Lname
       FROM     EMPLOYEE AS E
       WHERE    E.Ssn IN  ( SELECT    Essn
                            FROM      DEPENDENT AS D
                            WHERE     E.Fname=D.Dependent_name
                                      AND E.Sex=D.Sex );
```

```
For each E tuple,
  Evaluate the nested query
     which retrieves the Essn values of all D tuples
     with the same sex and name as E tuple
  If the Ssn value of E tuple is in the result,
      then select the E tuple
```

Use of EXISTS

```
SELECT Fname, Lname
FROM Employee
WHERE EXISTS (SELECT *
                         FROM DEPENDENT
                         WHERE Ssn= Essn)

              AND EXISTS (SELECT   *
                              FROM Department
                              WHERE Ssn= Mgr_Ssn)
```

List the managers who have at least one dependent

Use of NOT EXISTS and EXCEPT

Query: List first and last name of employees who work on ALL projects controlled by Dno=5.

```
SELECT Fname, Lname
FROM Employee
WHERE NOT EXISTS (SELECT  Pnumber
                          FROM PROJECT
                         WHERE Dno=5)

                 EXCEPT (SELECT   Pno
                          FROM WORKS_ON
                          WHERE Ssn= ESsn)
```

CUBE and ROLL UP operators

- ROLLUP operators let you extend the functionality of GROUP BY clauses by calculating subtotals and grand totals for a set of columns.

- The CUBE operator is similar in functionality to the ROLLUP operator; however, the CUBE operator can calculate subtotals and grand totals for all permutations of the columns specified in it.

From tables to OLAP cubes

From a table to a cube:

name	classid	Semester	Grade	Units
Jones	History105	F13	3.3	4.0
Jones	DataScience194	S12	4.0	3.0
Jones	French150	F14	3.7	4.0
Smith	History105	S15	2.3	3.0
Smith	DataScience194	F14	2.7	3.0
Smith	French150	F13	3.0	4.0

Variables used as qualifiers
(In where, GroupBy clauses)
Normally discrete

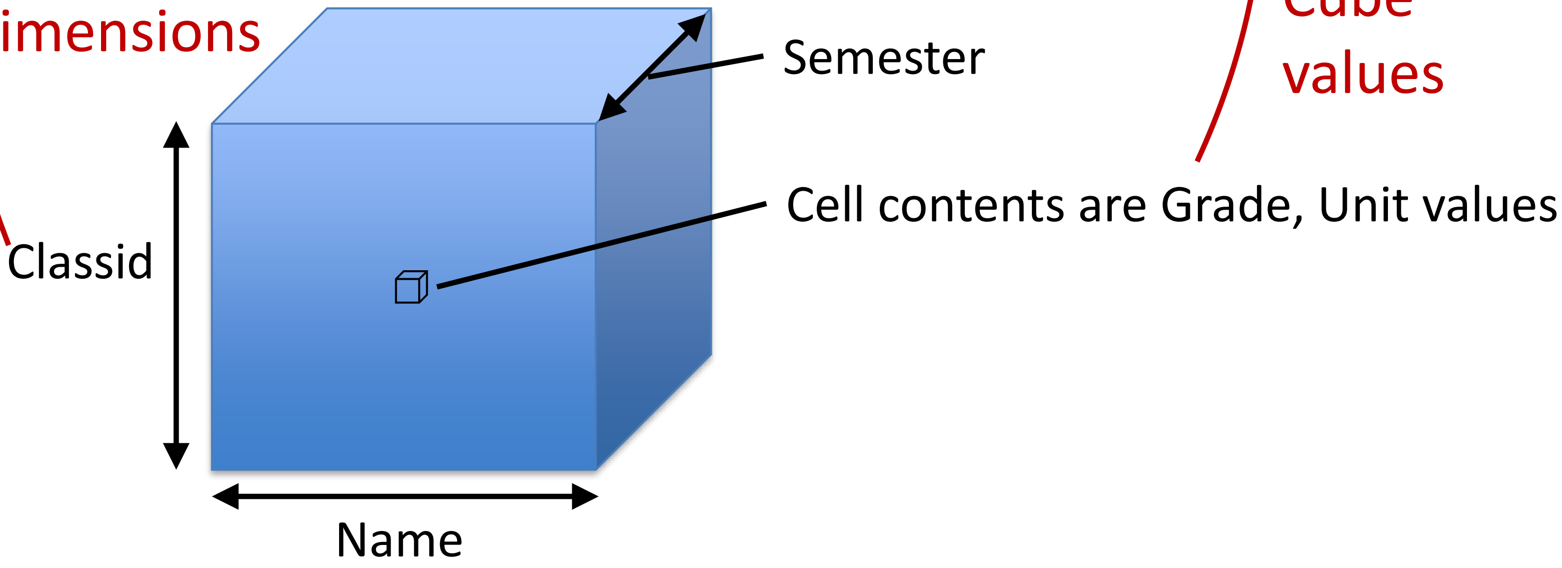
Variables we want to measure
Normally numeric

Constructing OLAP cubes

name	classid	Semester	Grade	Units
Jones	History105	F13	3.3	4.0
Jones	DataScience194	S12	4.0	3.0
…	…	…	…	…

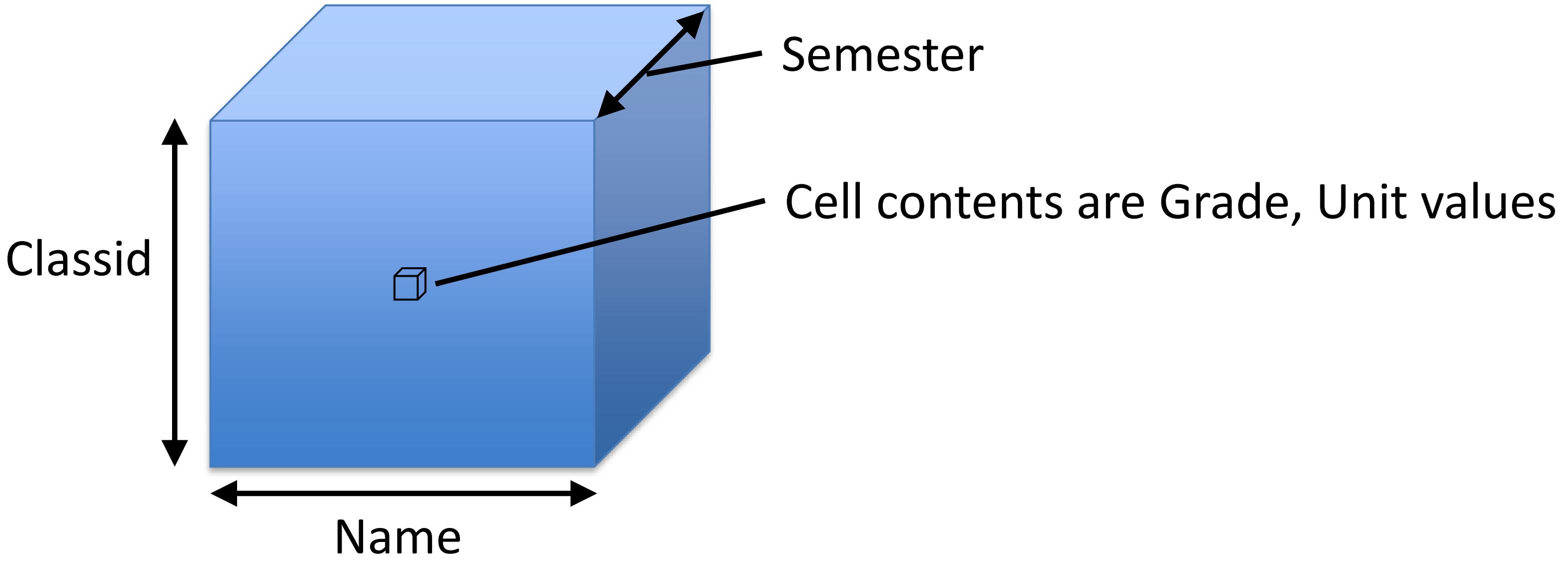
Cube dimensions

Cube values



Queries on OLAP cubes

- Once the cube is defined, its easy to do aggregate queries by projecting along one or more axes.
- E.g. to get student GPAs, we project the Grade field onto the student (Name) axis.
- In fact, such aggregates are precomputed and maintained automatically in an OLAP cube, so queries are instantaneous.



- EXAMPLE - SQL

Summary – COMP810

- Complex SQL:
  - Nested queries, joined tables (in the FROM clause), outer joins, aggregate functions, grouping

- `CREATE VIEW` statement and materialization strategies

- Schema Modification for the DBAs using `ALTER TABLE`, `ADD` `and` `DROP COLUMN`, `ALTER CONSTRAINT` etc.

ALL THE BEST IN YOUR FUTURE ENDEAVOURS

References:

(a) A Conceptual Poverty Mapping Data Model
Link: https://www.researchgate.net/figure/Key-thematic-layers-for-poverty-spatial-data-modeling_fig2_229724703

(b) Relational Database relationships
https://www.youtube.com/watch?v=C3icLzBtg8I

(c) https://courses.ischool.berkeley.edu/i202/f97/Lecture13/DatabaseDesign/sld002.htm

(d) https://nexwebsites.com/database/database-management-systems/

(e) Acknowledgement – Thanks to http://courses.cs.washington.edu/courses/cse544/ for providing part of this presentation.

(f) Acknowledgement – Thanks to © Silberchatz, Korth and Surdashan for providing part of this presentation.

(e) Malinowski, Elzbieta, Zimányi, Esteban (2008) *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications .* Springer Berlin Heidelberg. Copyright © 2008 Elzbieta Malinowski & Esteban Zimányi