Chapter 4: Intermediate SQL
part1- join to views

Chapter 4: Intermediate SQL

- Join Expressions
- Views
- Transactions
- Integrity Constraints
- SQL Data Types and Schemas
- Authorization

Joined Relations

- **Join operations** take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join
- The join operations are typically used as subquery expressions in the **from** clause

```
select name, course_id
from instructor natural join teaches;
```

Joined Relations

- **Join operations** take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the **from** clause
- **Join condition** – defines which tuples in the two relations match, and what attributes are present in the result of the join.
- **Join type** – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

- Suppose we wish to display a list of all students, displaying their *ID*, and *name*, *dept name*, and *tot cred*, along with the courses that they have taken. The following SQL query may appear to retrieve the required information:

select * **from** *student* **natural join** *takes*;

We would thus not see any information about students who have not taken any course.

Join operations – Example

- Relation *course*

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

n Relation *prereq*

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

n Observe that

prereq information is missing for CS-315 and

course information is missing for CS-437

Natural or conditional Joins

- *course* **natural join** *prereq*

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101

Outer Join

- More generally, some tuples in either or both of the relations being joined may be “lost” in this way.
- An extension of the join operation that avoids loss of information.
- The **outer join** operation works in a manner similar to the join operations we have already studied, but preserve those tuples that would be lost in a join, by creating **tuples in the result containing null values.**
- Computes the join and then adds tuples from one relation that **does not match tuples** in the other relation to the result of the join.
- Uses *null* values.

- There are in fact three forms of outer join:
- The **left outer join** preserves tuples only in the relation named before (to the left of) the **left outer join** operation
- The **right outer join** preserves tuples only in the relation named after (to the right of) the **right outer join** operation.
- The **full outer join** preserves tuples in both relations

Join types	*Join Conditions*
inner join	**natural**
left outer join	**on** <predicate>
right outer join	**using** $(A_1, A_1, \ldots, A_n)$
full outer join	

Left Outer Join

- *course* **natural left outer join** *prereq*

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

Right Outer Join

n *course* **natural right outer join** *prereq*

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

Full Outer Join

- *course* **natural full outer join** *prereq*

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

- “Display a list of all students in the Comp. Sci. department, along with the course sections, if any, that they have taken in Spring 2009; all course sections fromSpring 2009 must be displayed, even if no student from the Comp. Sci. department has taken the course section.” This query can be written as:
- **select** * **from** (**select** * **from** *student* **where** *dept name*= ’Comp. Sci’) **natural full outer join** (**select** * **from** *takes* **where** *semester* = ’Spring’ **and** *year* = 2009);

Join…using

- To provide the benefit of natural join while avoiding the danger of equating attributes erroneously, SQL provides a form of the natural join construct that allows you to specify exactly which columns should be equated. This feature is illustrated by the following query:
- **select** *name*, *title* **from** (*instructor* **natural join** *teaches*) **join** *course* **using** (*course id*);
- The operation **join** *. . .* **using** requires a list of attribute names to be specified. Both inputs must have attributes with the specified names.

- Consider the operation *r*1 **join** *r*2 **using**(*A*1*, A*2).

Join…on

- The **on** condition allows a general predicate over the relations being joined.This predicate is written like a **where** clause predicate except for the use of the keyword **on** rather than **where**. Like the **using** condition, the **on** condition appears at the end of the join expression.
- Consider the following query, which has a join expression containing the **on** condition.
- **select** * **from** *student* **join** *takes* **on** *student*.*ID*= *takes*.*ID*;
- The **on** condition above specifies that a tuple from *student* matches a tuple from *takes* if their *ID* values are equal.

- The following SQL statement will select all customers, and any orders they might have:

- SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  LEFT OUTER JOIN Orders ON Customers.CustomerID =
  Orders.CustomerID;

Inner join

- To distinguish normal joins from outer joins, normal joins are called **inner joins** in SQL. A join clause can thus specify **inner join** instead of **outer join** to specify that a normal join is to be used. The keyword **inner** is, however, optional.
- The default join type, when the **join** clause is used without the **outer** prefix is the **inner join**.

select * **from** *student* **join** *takes* **using** (*ID*);

is equivalent to:

select * **from** *student* **inner join** *takes* **using** (*ID*);

- The INNER JOIN keyword selects records that have matching values in both tables.
- SELECT *column_name(s)*
  FROM *table1*
  INNER JOIN *table2*
  ON *table1.column_name = table2.column_name*;

Joined Relations – Examples

- *course* **inner join** *prereq* **on** *course.course_id = prereq.course_id*

course_id	*title*	*dept_name*	*credits*	*prereq_id*	*course_id*
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

- What is the difference between the above, and a natural join?
- Notice that we do not repeat those attributes that appear in the schemas of both relations; rather they appear only once.
- Notice also the order in which the attributes are listed: first the attributes common to the schemas of both relations, second those attributes unique to the schema of the first relation, and finally, those attributes unique to the schema of the second relation.

Joined Relations – Examples

- *course* **natural right outer join** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

- *course* **full outer join** *prereq* **using** (*course_id*)

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

Views

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
- Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

```
select  ID, name, dept_name
from instructor
```

- A **view** provides a mechanism to hide certain data from the view of certain users.
- Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a **view**.

Does view occupy memory????

- Actually view is a stored select statement and its code is stored in data dictionary.
- It doesn't occupy space in the memory.
- whenever view is called, the stored select statement executes and fetches the data from the base table.
- If base table is deleted, view becomes invalid.

Insertion
Updation

View Definition

- A view is defined using the **create view** statement which has the form

create view *v* **as** < query expression >

where <query expression> is any legal SQL expression. The view name is represented by *v.*

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

Example Views

- A view of instructors without their salary

  create view *faculty* **as**
  select *ID*, *name*, *dept_name*
  from *instructor*

- Find all instructors in the Biology department

  select *name*
  from *faculty*
  where *dept_name* = 'Biology'

- **The attribute names of a view can be specified explicitly as follows:**

- To Create a view of department salary totals

  create view *departments_total_salary*(*dept_name*, *total_salary*) **as**
  select *dept_name*, **sum** (*salary*)
  from *instructor*
  group by *dept_name*;

Views Defined Using Other Views

n **create view** *physics_fall_2009* **as**
select *course.course_id*, *sec_id*, *building*, *room_number*
from *course*, *section*
where *course.course_id* = *section.course_id*
and *course.dept_name* = 'Physics'
and *section.semester* = 'Fall'
and *section.year* = '2009';

n **create view** *physics_fall_2009_watson* **as**
select *course_id*, *room_number*
from *physics_fall_2009*
where *building*= 'Watson';

View Expansion

- Expand use of a view in a query/another view

```
create view physics_fall_2009_watson as
    (select course_id, room_number
    from (select course.course_id, building, room_number
          from course, section
          where course.course_id = section.course_id
              and course.dept_name = 'Physics'
              and section.semester = 'Fall'
              and section.year = '2009')
    where building= 'Watson';
```

Updating a view

- The difficulty is that a modification to the database expressed in terms of a view must be translated to a modification to the actual relations in the logical model of the database.
- **create view** *faculty* **as**
  select *ID*, *name*, *dept_name*
  from *instructor*
- If a tuple is added as follows:
- **insert into** *faculty* **values** ('30765', 'Green', 'Music');
- This insertion must be represented by an **insertion into the relation *instructor*,** since *instructor* is the actual relation from which the database system constructs the view *faculty*. However, to insert a tuple into *instructor*, we must have some value for ***salary*.**

- There are two reasonable approaches to dealing with this insertion:
- • Reject the insertion, and return an error message to the user.
- • Insert a tuple (’30765’, ’Green’, ’Music’, *null*) into the *instructor* relation.

Update of a View

- Add a new tuple to *faculty* view which we defined earlier

insert into *faculty* **values** (’30765’, ’Green’, ’Music’);

This insertion must be represented by the insertion of the tuple

insert into *instructor* **values** (’30765’, ’Green’, ’Music’, null)

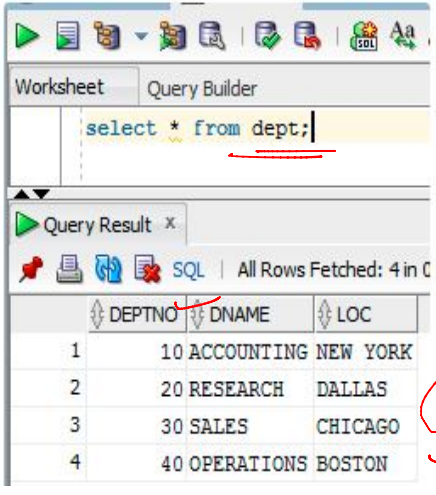
into the *instructor* relation

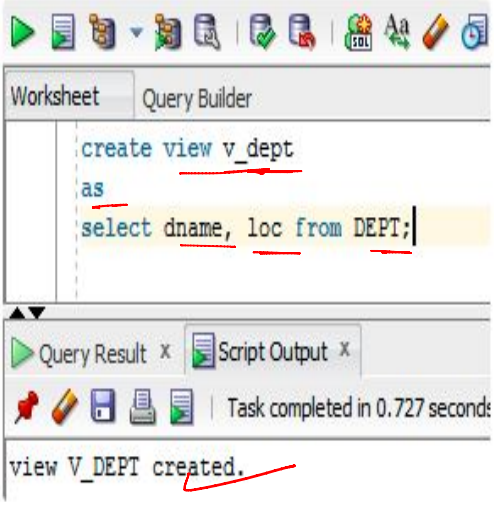
Some Updates cannot be Translated Uniquely

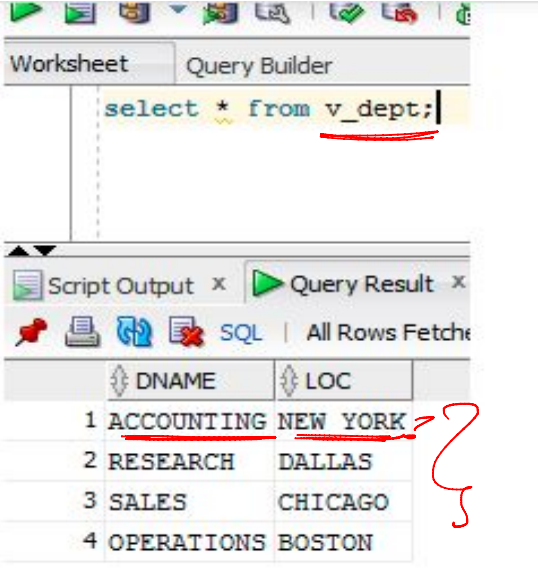
- **create view** *instructor_info* **as**
  select *ID*, *name*, *building*
  from *instructor*, *department*
  where *instructor.dept_name*= *department.dept_name*;

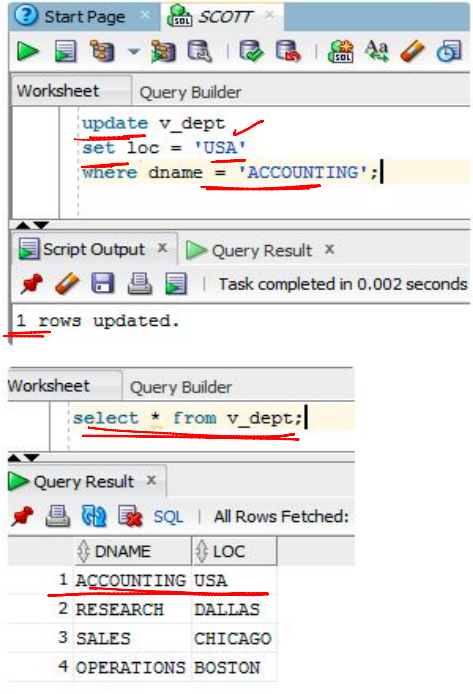
- **insert into** *instructor_info* **values** ('69987', 'White', 'AB5');
  - If id does not exists
  - what if no department is in AB5?
- **Suppose we Insert into instructor the (** 69987', 'White', *null*, *null)*
- *Insert in department(null, AB5, null)*
- *In the absence of common dept name this view update will fail*

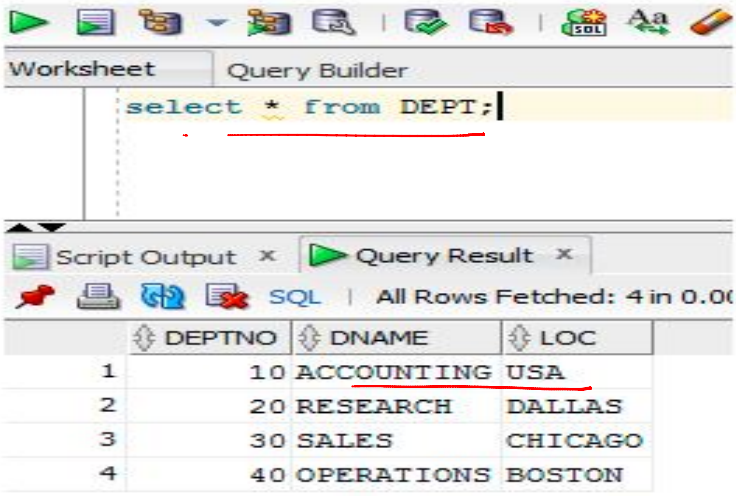
Updating main table through views






Worksheet
Query Builder
select * from v_dept;
Script Output
Query Result
SQL
All Rows Fetche
DNAME
LOC
1 ACCOUNTING NEW YORK
2 RESEARCH DALLAS
3 SALES CHICAGO
4 OPERATIONS BOSTON


Start Page
SCOTT
Worksheet
Query Builder
update v_dept
set loc = 'USA'
where dname = 'ACCOUNTING';
Script Output
Query Result
Task completed in 0.002 seconds
1 rows updated.
Worksheet
Query Builder
select * from v_dept;
Query Result
SQL
All Rows Fetched:
DNAME
LOC
1 ACCOUNTING USA
2 RESEARCH DALLAS
3 SALES CHICAGO
4 OPERATIONS BOSTON



Delete a view;

Drop viewname;

- an SQL view is said to be **updatable** (that is, inserts, updates or deletes can be applied on the view) if the following conditions are all satisfied by the query defining the view:
  - The **from** clause has only one database relation.
  - The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification.
  - Any attribute not listed in the **select** clause can be set to null
  - The query does not have a **group** by or **having** clause.

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
69987	White	*null*	*null*

instructor

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000
null	Taylor	*null*

department

with check option

- **create view** *history_instructors* **as**
  select *
  from *instructor*
  where *dept_name*= 'History';

- What happens if we insert ('25566', 'Brown', 'Biology', 100000) into *history_instructors?*

- By default, SQL would allow the above update to proceed.
- However, views can be defined with a **with check option** clause at the end of the view definition; then, if a tuple inserted into the view does not satisfy the view's **where** clause condition, the insertion is rejected by the database system.
- Updates are similarly rejected if the new value does not satisfy the **where** clause conditions.

```sql
CREATE [redacted] VIEW vps AS
    SELECT
        employeeNumber,
        lastName,
        firstName,
        jobTitle,
        extension,
        email,
        officeCode,
        reportsTo
    FROM
        employees
    WHERE
        jobTitle LIKE '%VP%'
WITH CHECK OPTION;
```

```
INSERT INTO vps(employeeNumber,firstname,lastname,jobtitle,extension,email,office
Code,reportsTo)
VALUES(1704,'John','Smith','IT Staff','x9112','johnsmith@classicmodelcars.com',1,
1703);
```

This time, MySQL rejected the insert and issued the following error message:

```
Error Code: 1369. CHECK OPTION failed 'classicmodels.vps'
```

Materialized Views

- **Materializing a view**: create a physical table containing all the tuples in the result of the query defining the view

- If relations used in the query are updated, the materialized view result becomes out of date
  - Need to **maintain** the view, by updating the view whenever the underlying relations are updated.