

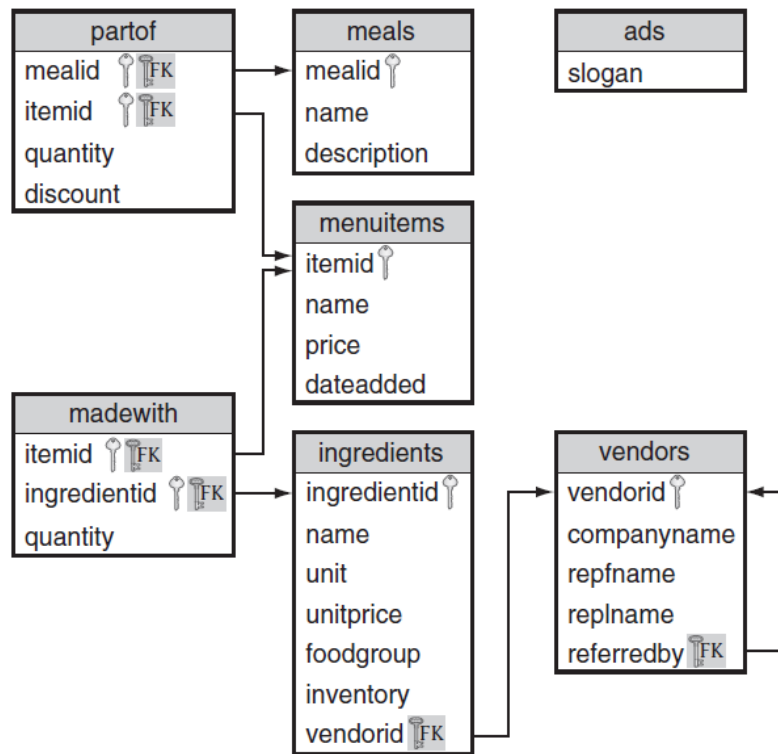
Birla Institute of Technology and Science, Pilani.

Database Systems

Lab No #4

In this lab we will continue practicing SQL queries related to joins etc on a schema of a restaurant chain created and populated in the last lab.

The table schema is given below.



JOIN

- ❖ Until now, all of our queries have used a single table. It is not surprising that many queries require information from more than one table. Suppose that you want a list of ingredients and their vendor (ID and company name).
- ❖ **Combining tables to derive a new table is called a *join*.**

For each ingredient, find its name and the name and ID of the vendor that supplies it

```
SELECT vendors.vendorid, name, companyname
FROM ingredients, vendors
WHERE ingredients.vendorid = vendors.vendorid;
```

Find the names of the ingredients supplied to us by Veggies_R_Us

```
SQL>SELECT name
```

```
FROM ingredients, vendors
WHERE ingredients.vendorid = vendors.vendorid AND companyname =
'Veggies_R_Us';
```

❖ **Designing a join is a two-step process:**

- 1. Find the tables with the information that we need to answer the query.
- 2. Determine how to connect the tables.

SELF JOIN- Joining a Table with Itself:

- ❖ SQL handles this situation by allowing two copies of the same table to appear in the FROM clause. Allowing two copies of the same table to appear in the FROM clause. The only requirement is that each copy of the table must be given a distinct alias to distinguish between the copies of the table

Find all of the vendors referred by Veggies_R_Us

```
SELECT v2.companyname
FROM vendors v1, vendors v2 /*Note the table alias*/
WHERE v1.vendorid = v2.referredby AND v1.companyname = 'Veggies_R_Us';
```

THETA JOIN- Generalizing Join Predicates:

- ❖ **A join where the join predicate uses any of the comparison operators is called a *theta join***

Find all of the items and ingredients where we do not have enough of the ingredient to make three items.

```
SELECT items.name, ing.name
FROM items, madewith mw, ingredients ing
WHERE items.itemid = mw.itemid AND mw.ingredientid = ing.ingredientid
AND
3 * mw.quantity > ing.inventory;
```

Find the name of all items that cost more than the garden salad

```
SQL>SELECT a.name
FROM items a, items q
WHERE a.price > q.price AND q.name = 'Garden Salad';
```

JOIN Operators:

- ❖ Joining tables together is so common that SQL provides a JOIN operator for use in the FROM clause. There are several variants of the JOIN, which we explore here.
- ❖ **INNER JOIN:** **INNER JOIN takes two tables and a join specification describing how the two tables should be joined. The join specification may be specified as a condition. The condition follows the keyword ON**

Find the names of the ingredients supplied to us by Veggies_R_Us

```
SELECT name
FROM ingredients i INNER JOIN vendors v ON i.vendorid = v.vendorid
WHERE v.companyname = 'Veggies_R_Us';
```

Find the name of all items that cost more than the garden salad

```
SELECT i1.name
FROM items i1 INNER JOIN items i2 ON i1.price > i2.price
WHERE i2.name = 'Garden Salad';
```

- ❖ It is common to join tables over attributes with the same name. Thus, **SQL provides a shorthand for this type of join. The USING clause lists those attributes common to both tables that must have the same value to be in the result. Here USING is identical to the equijoin. Note that INNER may be omitted because it is the default type of JOIN.**

Find the names of the ingredients supplied to us by Veggies_R_Us

```
SELECT companyname, name, vendorid
FROM ingredients JOIN vendors v USING (vendorid)
WHERE v.companyname = 'Veggies_R_Us';
```

Find the names of items that are made from ingredients supplied by the company Veggies_R_Us

```
SELECT DISTINCT(i.name)
FROM vendors JOIN ingredients USING (vendorid) JOIN
madewith USING(ingredientid) JOIN items i USING (itemid)
WHERE companyname = 'Veggies_R_Us';
```

- ❖ **OUTER JOIN:** Outer joins are useful where we want not only the rows that satisfy the join predicate, but also the rows that do not satisfy it. There are three types of OUTER JOIN: FULL, LEFT, and RIGHT. FULL OUTER JOIN includes three kinds of rows:
 - All rows that satisfy the join predicate (same as INNER JOIN)
 - All rows from the first table that don't satisfy the join predicate for any row in the second table
 - All rows from the second table that don't satisfy the join predicate for any row in the first table.

For each ingredient, find its name and the name and ID of the vendor that supplies them. Include vendors who supply no ingredients and ingredients supplied by no vendors

```
SELECT companyname, i.vendorid, i.name
FROM vendors v FULL JOIN ingredients i ON v.vendorid = i.vendorid;
```

Find the vendors who do not provide us with any ingredients

```
SELECT companyname
FROM vendors v LEFT JOIN ingredients i on v.vendorid=i.vendorid
WHERE ingredientid IS NULL;
```

- ❖ **CROSS JOIN:** SQL also provides a CROSS JOIN. It computes the cross product of two tables. We cannot use OUTER, NATURAL, USING, or ON with CROSS JOINS. Note that this is the same behavior as using a comma-delimited list of tables in the FROM clause.

- The size of a Cartesian product result set is the number of rows in the first table multiplied by the number of rows in the second table.

```
SELECT i.name, v.companyname AS Vendor
FROM ingredients i
CROSS JOIN vendors v
ORDER BY v.vendorid
```

- ❖ **NATURAL JOIN:** This particular operator is not supported in MS SQL Server. It is supported in Oracle.
 - By now you may have noticed we often use equality predicates where both tables contain attribute(s) using the same name. This is a common join query. In fact, it is so common that it is often called the natural join. Again, to simplify the syntax, SQL has a NATURAL modifier for both the inner and outer joins. With this modifier present, we don't need a join specification, so the ON .
 - Note that if the tables contain no matching attribute names, the NATURAL JOIN performs a Cartesian product.

Find the names of the ingredients supplied to us by Veggies_R_Us


```
SELECT name
FROM ingredients NATURAL JOIN vendors
WHERE companyname = 'Veggies_R_Us';
```

Find the vendors who do not provide us with any ingredients

```
SELECT companyname
FROM vendors v NATURAL LEFT JOIN ingredients i
WHERE ingredientid IS NULL;
```

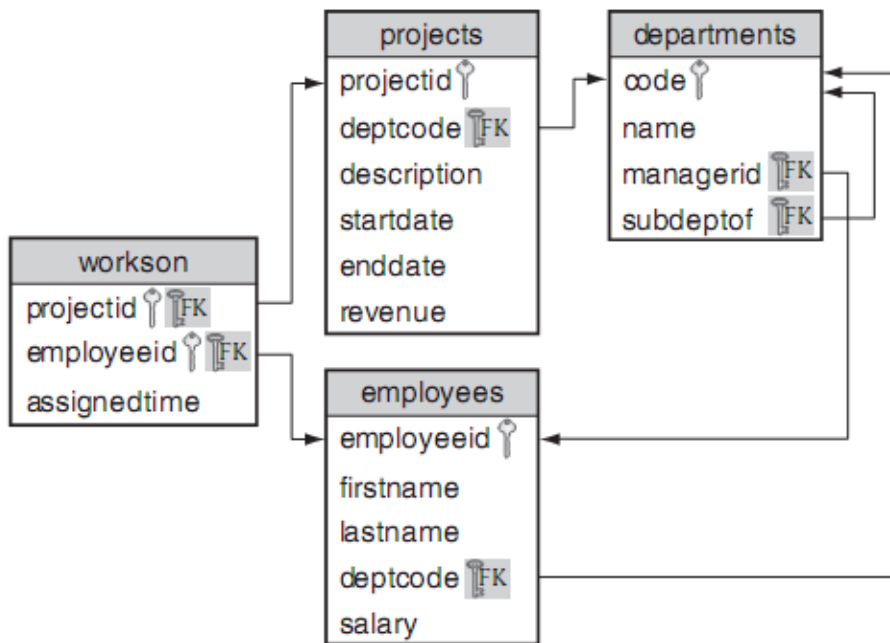
Find the names of items that are made from ingredients supplied by Veggies_R_Us

```
SELECT name
FROM vendors NATURAL JOIN ingredients NATURAL JOIN madewith NATURAL
JOIN items
WHERE companyname = 'Veggies_R_Us';
```

- ❖ What is wrong with above query? 

EXERCISES:

- ❖ Write a single SQL query for each of the following based on employee database created in the last lab. The scheme for employee database is shown below.



- Find the names of all people who work in the Consulting department. Solve it two ways:
 - using only WHERE-based join (i.e., no INNER/OUTER/CROSS JOIN)
 - with CROSS JOIN.
- Find the names of all people who work in the Consulting department and who spend more than 20% of their time on the project with ID ADT4MFIA. Solve three ways:
 - using only WHERE-based join (i.e., no INNER/OUTER/CROSS JOIN),\
 - using JOIN ON
- Find the total percentage of time assigned to employee Abe Advice. Solve it two ways:
 - using only WHERE-based join (i.e., no INNER/OUTER/CROSS JOIN) and
 - using some form of JOIN.
- Find the descriptions of all projects that require more than 70% of an employee's time. Solve it two ways:
 - using only WHERE-based join (i.e., no INNER/OUTER/CROSS JOIN) and
 - using some form of JOIN.
- For each employee, list the employee ID, number of projects, and the total percentage of time for the current projects to which she is assigned. Include employees not assigned to any project.
- Find the description of all projects with no employees assigned to them.
- For each project, find the greatest percentage of time assigned to one employee. Solve it two ways:
 - using only WHERE-based join (i.e., no INNER/OUTER/CROSS JOIN) and
 - using some form of JOIN.
- For each employee ID, find the last name of all employees making more money than that employee. Solve it two ways: 1) using only WHERE-based join (i.e., no INNER/OUTER/CROSS JOIN) and 2) using some form of JOIN.

9. Rank the projects by revenue. Solve it two ways: 1) using only WHERE-based join (i.e., no INNER/OUTER/CROSS JOIN) and 2) using some form of JOIN.

UNION, INTERSECTION and MINUS

- ❖ **Union:** UNION combines two query results. The syntax for UNION is as follows:

```
<left SELECT> UNION [{ALL | DISTINCT}] <right SELECT>.
```

- The <left SELECT> and <right SELECT> can be almost any SQL query, provided that the result sets from the left and right SELECT are compatible. Two result sets are compatible if they have the same number of attributes and each corresponding attribute is compatible. Two attributes are compatible if SQL can implicitly cast them to the same type.
- Out of ALL or DISTINCT options DISTINCT is default. that means query will eliminate duplicates by default.
- if we want the sets to be treated as multi sets the we should use ALL option instead.

Find the list of item prices and ingredient unit prices

```
SELECT price
FROM items
UNION
SELECT unitprice
FROM ingredients;
```

Find the names and prices of meals and items

```
SELECT name, price
FROM items
UNION
SELECT m.name, SUM(quantity * price * (1.0 - discount))
FROM meals m, partof p, items i
WHERE m.mealid = p.mealid AND p.itemid = i.itemid
GROUP BY m.mealid, m.name;
```

- ❖ **Intersection:** The intersection of two sets is all the elements that are common to both sets. In SQL, the INTERSECT operator returns all rows that are the same in both results. The syntax for INTERSECT is as follows:

```
<left SELECT> INTERSECT [{ALL | DISTINCT}] <right SELECT>
```

- The <left SELECT> and <right SELECT> must be compatible, as with UNION . By default, INTERSECT eliminates duplicates.
- If the left and right tables have l and r duplicates of a value, v ; the number of duplicate values resulting from an INTERSECT ALL is the minimum of l and r .

Find all item IDs with ingredients in both the Fruit and Vegetable food groups

```
SELECT itemid
FROM madewith mw, ingredients ing
WHERE mw.ingredientid = ing.ingredientid and foodgroup = 'Vegetable'
INTERSECT
SELECT itemid
FROM madewith mw, ingredients ing
WHERE mw.ingredientid = ing.ingredientid and foodgroup = 'Fruit';
```

- ❖ **Difference:** Another common set operation is set difference. If R and S are sets, then $R - S$ contains all of the elements in R that are not in S. The EXCEPT (MINUS in Oracle 11g) operator in SQL is similar, in that it returns the rows in the first result that are not in the second one. The syntax for EXCEPT is as follows:

```
<left SELECT> EXCEPT [ALL | DISTINCT] <right SELECT>
```

- The <left SELECT> and <right SELECT> must be compatible, as with UNION and INTERSECT. Like UNION and INTERSECT, EXCEPT eliminates duplicates. To keep duplicates, use EXCEPT ALL. If the left and right tables have l and r duplicates of a value, v; the number of duplicate values resulting from an EXCEPT ALL is the minimum of l-r and 0.

Find all item IDs of items not made with Cheese

```
SELECT itemid
FROM items
MINUS
SELECT itemid
FROM madewith mw, ingredients ing
WHERE mw.ingredientid = ing.ingredientid AND ing.name =
'Cheese';
```

Find all item IDs of items not made with Cheese

```
SELECT DISTINCT(itemid)
FROM madewith mw, ingredients ing
WHERE mw.ingredientid = ing.ingredientid AND ing.name != 'Cheese';
```

- ❖ Did you get the result? What is wrong with above query?

List all the food groups provided by some ingredient that is in the Fruit Plate but not the Fruit Salad?

```
SELECT foodgroup
FROM madewith m, ingredients i
WHERE m.ingredientid = i.ingredientid AND m.itemid = 'FRPLT'
MINUS
```

```

SELECT foodgroup
FROM madewith m, ingredients i
WHERE m.ingredientid = i.ingredientid AND m.itemid = 'FRTSD';

```

Find the vendors who do not provide us with any ingredients

```

SELECT companyname
FROM vendors v LEFT JOIN ingredients i on v.vendorid=i.vendorid
WHERE ingredientid IS NULL;

```

- ❖ Rewrite the above query to use EXCEPT instead of an OUTER JOIN
- ❖ From set theory, $R \cap S = R - (R - S)$. Rewrite the following query using EXCEPT instead of INTERSECT.

```

SELECT itemid
FROM madewith m, ingredients i
WHERE m.ingredientid = i.ingredientid AND foodgroup='Milk'
INTERSECT
SELECT itemid
FROM madewith m, ingredients i
WHERE m.ingredientid = i.ingredientid AND foodgroup='Fruit';

```

EXERCISES

- ❖ Write a single SQL query for each of the following based on employee database.
1. Find all dates on which projects either started or ended. Eliminate any duplicate or *NULL* dates. Sort your results in descending order.
 2. Use INTERSECT to find the first and last name of all employees who both work on the Robotic Spouse and for the Hardware department.
 3. Use EXCEPT to find the first and last name of all employees who work on the Robotic Spouse but not for the Hardware department.
 4. Find the first and last name of all employees who work on the Download Client project but not the Robotic Spouse project.
 5. Find the first and last name of all employees who work on the Download Client project and the Robotic Spouse project.
 6. Find the first and last name of all employees who work on either the Download Client project or the Robotic Spouse project.
 7. Find the first and last name of all employees who work on either the Download Client project or the Robotic Spouse project but not both.
 8. Using EXCEPT, find all of the departments without any projects.

Reference:

- ❖ This lab sheet is prepared from the book: SQL: Practical Guide for Developers. Michael J. Donahoo and Gregory D. Speegle.