Chapter 10: Joins

Cartesian product or CROSS JOIN replicates each row from the first table with every row from the second table.
Equality join also known as equijoin, inner join or simple join creates a join by using a commonly named and defined column.
Non-equality join joins tables where there are no equivalent rows in the tables to be joined.
Self-join joins a table to itself.
Outer join includes records of a table when there is no matching record in the other table.
Set operators UNION, UNION ALL, INTERSECT and MINUS combines results from multiple select statements.

WHERE in the traditional approach, the WHERE clause indicates which columns should be joined.
NATURAL JOIN the keywords are used in the FROM clause to join tables containing a common column with the same name.
JOIN … USING the join keyword is used in the FROM clause; combined with the USING clause, it identifies the common column used to join the tables.
JOIN … ON the join keyword is used in the FROM clause. The ON clause identifies the columns used to join the tables.
OUTER JOIN can be used with LEFT, RIGHT, FULL indicates that at least one of the tables does not have a matching row.

Cartesian Product/Cross Join: SELECT * FROM heroes, skills CROSS JOIN table
Inner Join: SELECT name, skill_name FROM heroes h, skills s WHERE h.skill_code = s.skill_code.
- CANNOT USE TABLE NAMES ONCE ALIASES HAVE BEEN ASSIGNED

Non-equi Join: SELECT name, score, grade FROM students, grade_range WHERE score BETWEEN beg_score AND end_score;
- It is used when the related columns can't be joined with an equal sign – meaning no equivalent rows in tables joined

Self Join: SELECT e.name Employee, m.name Manager FROM employee e, employee m WHERE e.manager_ssn=m.ssn;
Outer Join: SELECT * FROM a, b, c where col1 = col2(+) and col2 = col3;
- CANNOT USE TWO +'s

Full Outer Join: SELECT p.fname, d.disease_desc FROM disease d FULL OUTER JOIN patient_disease pd ON d.disease_id=pd.disease_id FULL OUTER JOIN patient p ON p.patient_id=pd.patient_id;
Group By Example: SELECT fname firstname, count(*) DiseaseCount FROM patient p, patient_disease pd WHERE p.patient_id=pd.patient_id GROUP BY firstName Having DiseaseCount > 1;

Chapter 10: Set Operators

Union (Returns the results of both queries and removes duplicates)
- SELECT lname, hrly_wage, 'poor' FROM hourly WHERE hrly_wage <= 15 UNION Select lname, hrly_wage, 'okay' FROM hourly WHERE hrly_wage > 15;
- NEED TO HAVE SAME NUMBER OF COLUMNS, TYPES MUST MATCH

Union All (Returns the results of both queries but includes duplicates)
- SELECT lname FROM hourly UNION ALL SELECT lname FROM salaried;

Intersect (Returns only the rows included in the results of both queries)
- SELECT lname, 'hello' FROM hourly INTERSECT SELECT lname, 'hello' FROM salaried;

Minus (Subtracts the second query's results if they are also returned in the first query's result)
- SELECT lname, 'hello' FROM salaried MINUS SELECT lname, 'hello' FROM hourly;

Exists (Searches for the presence of a single row meeting the stated criteria)
- SELECT patient_id, lname FROM patient p WHERE EXISTS (SELECT * FROM patient_disease pd WHERE p.patient_id=pd.patient_id);

Not Exists (If there is at least a single row meeting the state criteria, nothing is displayed)
- SELECT patient_id, lname FROM patient p WHERE NOT EXISTS (SELECT * FROM patient_disease pd WHERE p.patient_id=pd.patient_id);

Chapter 12: System Tables

user_tables: SELECT table_name FROM user_tables;
- List all of the tables created by the user

user_constraints: SELECT table_name, constraint_name, r_constraint_name, constraint_type, status FROM user_constraints WHERE table_name IN ('PATIENT', 'PATIENT_DISEASE');
user_cons_columns: SELECT table_name, constraint_name, column_name FROM user_cons_columns WHERE table_name IN ('PATIENT', 'PATIENT_DISEASE');
- The top two can be used connected to see what rows are used in references

user_indexes: SELECT index_name, table_name FROM user_indexes WHERE table_name IN ('PATIENT', 'PATIENT_DISEASE');
- List all of the indexes that have been created

user_ind_columns: SELECT index_name, column_name, table_name FROM user_ind_columns WHERE table_name='PATIENT';
- The column information for all of the indexes are in the user_ind_columns table

user_views: SELECT view_name, text FROM user_views;
- View information is in the user_views table

SELECT uc.table_name, column_name, constraint type FROM user_constraints uc, user_cons_columns ucc WHERE (uc.table_name='PATIENT_DISEASE' AND constraint_type='R') (AND uc.r_constraint_name = ucc.r_constraint_name OR uc.constraint_name = ucc.constraint_name);

Give a listing of all the ssns,first names and the class descriptions of all the classes the students are taking. If there are no class _descriptions display 'No description is available yet'. (USE NVL)

```
SELECT ssn, fname, nvl(class_description, 'no description is available yet') class_description
FROM student NATURAL JOIN student_class NATURAL JOIN class;
```

Give a listing of only the lname and the class_code for students who are taking 'Introduction to C programming'. (Inner join)

```
SELECT lname, c.class_code
FROM student s, student_class sc, class c
WHERE s.ssn = sc.ssn AND sc.class_code = c.class_code AND
class_description LIKE 'Introduction to C programming';
```

Give a listing of all the class_descriptions and the number of students enrolled in each class for all students who are older than the average age where the total number of students for the class is more than 1 student. Order by the number of students. If there is no class description replace it with 'Other Classes' (Note: Take it in steps. First do all those who are older than the average age, then do the group by, then add the having c

Create a new table that contains the list of all the students and class_descriptions. Include in this table the list of all students who are not enrolled in any classes (display no classes). If there are no class descriptions then display 'no description' (Use combination of inner join, union and minus) (Note: minus will deal with the students who are not enrolled in any classes)

```
CREATE TABLE new_table AS
SELECT fname, lname, nvl(class_description, 'no description') class_description
    FROM student s INNER JOIN student_class sc ON s.ssn = sc.ssn
    INNER JOIN class c ON sc.class_code = c.class_code
UNION
(
```

Give a listing of all the classes for which no students are enrolled in (use in or not in clause) (subquery)

Repeat question 6 using a combination of inner join, union and not exists (Note: Not exists will deal with the students who are not enrolled in any classes)

```
CREATE TABLE new_table AS
SELECT fname, lname, nvl(class_description, 'no description') class_description
    FROM student s INNER JOIN student_class sc ON s.ssn = sc.ssn
    INNER JOIN class c ON sc.class_code = c.class_code
UNION
(
    SELECT fname, lname, 'no classes' FROM student s
    WHERE NOT EXISTS (SELECT * FROM student_class sc
    WHERE s.ssn = sc.ssn)
)
```

Create a view. We want to find out which courses are being taken by the different students for all those whose age is greater than the average age. Give a listing of the

Relational Algebra

Operation	My HTML	Symbol	Operation	My HTML	Symbol
Projection	PROJECT	π	Cartesian product	X	$\times$
Selection	SELECT	σ	Join	JOIN	$\bowtie$
Renaming	RENAME	ρ	Left outer join	LEFT OUTER JOIN	$\overset{\text{L}}{\bowtie}$
Union	UNION	$\cup$	Right outer join	RIGHT OUTER JOIN	$\overset{\text{R}}{\bowtie}$
Intersection	INTERSECTION	$\cap$	Full outer join	FULL OUTER JOIN	$\overset{\text{F}}{\bowtie}$
Assignment	<-	$\leftarrow$	Semijoin	SEMIJOIN	$\ltimes$

not enrolled in. Give a listing of the e not taking that specific course

```
RAL JOIN class
```

b) Unique key name and the columns that make up the unique key for the **student** table

```
SELECT constraint_name, column_name
FROM user_constraints NATURAL JOIN all_cons_columns
WHERE table_name = 'STUDENT'
AND constraint_type = 'U';
```

c) Foreign key name, the columns that make up the foreign key and the columns it references in the parent table for **student_class** table

```
SELECT constraint_name, column_name, r_constraint_name
FROM user_constraints NATURAL JOIN all_cons_columns
WHERE table_name = 'STUDENT_CLASS'
AND constraint type = 'R';
```

d) Name of all the check constraints and their conditions for the **student** table

```
SELECT constraint_name, search_condition
FROM user_constraints
WHERE table_name = 'STUDENT'
AND constraint_type = 'C';
```

```
-- Which books have both keywords 'database' and 'Programming?

π title
(
   (π isbn, title
     (σ keyword='programming' (books ⋈ describes)))
    ∩
    (π isbn, title
     (σ keyword='database' (books ⋈ describes)))
)
```

```
-- ID and name of students who did not apply anywhere
π SID, SNAME
((π SID student
-
π SID apply) ⋈ student)
```

```
-- List the title of books written by the author 'Expert'  but not
-- books that have the keyword database

π title
(σ aname = 'expert'
  (authors ⋈ (aid = aaid)
    (haswritten ⋈
      (σ keyword ≠ 'database' (books ⋈ describes))
    )
  )
)
```

- Derivation: $r \cap s = r - (r - s)$
- Example: given the relations r and s

r

A	B
α	1
α	2
β	1

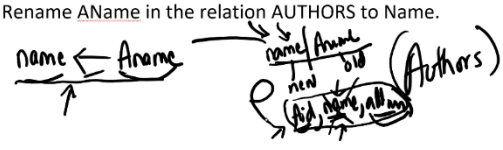
s

A	B
α	2
β	3

$r \cap s$

A	B
α	2

Rename AName in the relation AUTHORS to Name.



$$\pi\left(\rho_{(sid,\ supplierName,\ address)}\left(\left(\pi_{sid}\left(\sigma_{carModel='Toyota'}(Parts \bowtie Suppliers\_parts)\right)\right) \cap \left(\pi_{sid}\left(\sigma_{carModel='Ford'}(Parts \bowtie Suppliers\_parts)\right)\right)\right)\right)$$

$$\pi_{address}\left(Suppliers \bowtie \left(\sigma_{carModel='Ford'}\left(\sigma_{pName='bumper'}\left(\sigma_{cost>100}(Parts \bowtie Supplier\_parts)\right)\right)\right)\right)$$

$$\pi_{sName}\left(\left(\sigma_{carModel='Toyota' \vee carModel='Ford}(Parts \bowtie Supplier\_parts)\right)\right)$$