

# Relational Databases: SQL(Structure Query Language)

ACS 575: Database Systems

Instructor: Dr. Jin Soung Yoo, Professor  
Department of Computer Science  
Purdue University Fort Wayne

# References

---

- R. Elmasri et al., Fundamentals of Database Systems, 7<sup>th</sup> ed.  
Ch 6 and Ch 7
- W. Lemanhieu, et al., Principles of Database Management,  
Ch 7

# Outline

---

- Basic SQL Queries
  - A structure of basic SQL query
  - Sort : Order by
  - Prefix, and Aliases
  - \*, and DISTINCT
  - IN, and IS NULL comparison operators
  - String comparison operators
  - Arithmetic operation
  - Set operation
  - Single-row functions
  - Aggregation functions
  - Grouping query : Group by, Having

# Outline (Conti.)

---

- ❑ Advanced SQL Queries
  - Nested query
  - Correlated query
  - EXISTS operator
- ❑ SQL Query with Multiple Relations
  - Join, Outer Join
- ❑ Other SQL DML Statements
  - Insert, Delete and Update
- ❑ View
- ❑ Other database objects

# Why SQL(Structured Query Language)

---

- ❑ SQL is a very-high-level language, in which the programmer is able to avoid specifying a lot of data-manipulation details that would be necessary in languages like C++.
- ❑ The advantage of limiting the language expressions make it possible effective query processing and optimization.

# SQL(Structured Query Language)

---

- ❑ First version, SQL-86 in 1986, most recent version in 2016 (SQL:2016)
- ❑ Accepted by the American National Standards Institute (ANSI) in 1986 and by the International Organization for Standardization (ISO) in 1987
- ❑ Each vendor provides own implementation (SQL dialect) of SQL

# Key Characteristics of SQL

---

- ❑ Set-oriented and declarative
- ❑ Free form language
- ❑ Case insensitive
- ❑ Can be used interactively from a command prompt or executed by a program

# SQL

---

- SQL can be divided into categories.
  - **DML (Data Manipulation Language)**
    - Commands that maintain and query a database
      - Query (SELECT)
      - Data managements (INSERT, DELETE, UPDATE)
  - **DDL (Data Definition Language)**
    - Commands that define a database, including CREATE, ALTER, and DROP tables or other objects (e.g., view, user, synonym, sequence, role, ...)
  - **DCL (Data Control Language)**
    - Commands that control a database, including administering privileges and committing data (e.g., GRANT, REVOKE, COMMIT, ROLLBACK....)



# SQL DML (Data Manipulation Language)

---

- ❑ SQL SELECT Statement
- ❑ SQL INSERT Statement
- ❑ SQL DELETE Statement
- ❑ SQL UPDATE Statement

# SQL Queries

---

- Basic SQL queries correspond to using the following operations of the relational algebra:
  - **SELECTION** ( $\sigma$ ) selects a subset of tuples from relation
  - **PROJECTION** ( $\pi$ ) exclude unwanted attributes from relation
  - **JOIN** ( $\bowtie$ ) produces a set of all combinations of tuples in two relations R and S that satisfy a given condition (e.g., equal) on their common attribute names.
  - *See the reference slide of the relational algebra query in the course webpage.*
- SQL is the implementation of relational algebra

# A Structure of SQL Query

---

- SQL has one basic statement for retrieving information from a database; the **SELECT** statement
  - This is *not the same as* the SELECTION operation of the relational algebra

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

# SELECT Statement

---

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

- Used for queries on single or multiple tables
  - **SELECT**  
List the columns (and expressions) that should be returned from the query. That is, list attributes in the project operation of relational algebra
  - **FROM**  
Indicate the table(s) or view(s) from which data will be obtained
  - **WHERE**  
Indicate the conditions under which a row will be included in the result. That is, put selection conditions in the selection operation of relational algebra.
  - **GROUP BY**  
Indicate categorization of results
  - **HAVING**  
Indicate the conditions under which a category (group) will be included
  - **ORDER BY**  
Sorts the result according to specified criteria

# Basic Query in SQL

---

- Basic form of the SQL SELECT statement is called a *mapping* or a **SELECT-FROM-WHERE block**

**SELECT** <attribute list>  
**FROM** <table list>  
**WHERE** <condition>

- <attribute list> is a list of attribute names whose values are to be retrieved by the query
- <table list> is a list of the relation names required to process the query
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query

# COMPANY Database Schema

---

## EMPLOYEE

FNAME	MINIT	LNAME	<u>SSN</u>	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
-------	-------	-------	------------	-------	---------	-----	--------	----------	-----

## DEPARTMENT

DNAME	<u>DNUMBER</u>	MGRSSN	MGRSTARTDATE
-------	----------------	--------	--------------

## DEPT\_LOCATIONS

<u>DNUMBER</u>	<u>DLOCATION</u>
----------------	------------------

## PROJECT

PNAME	<u>PNUMBER</u>	PLOCATION	DNUM
-------	----------------	-----------	------

## WORKS\_ON

<u>ESSN</u>	<u>PNO</u>	HOURS
-------------	------------	-------

## DEPENDENT

<u>ESSN</u>	<u>DEPENDENT_NAME</u>	SEX	BDATE	RELATIONSHIP
-------------	-----------------------	-----	-------	--------------

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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	1968-07-19	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

					DEPT_LOCATIONS	DNUMBER	DLOCATION
						1	Houston
						4	Stafford
						5	Bellaire
						5	Sugarland
						5	Houston
DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE			
	Research	5	333445555	1988-05-22			
	Administration	4	987654321	1995-01-01			
	Headquarters	1	888665555	1981-06-19			

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

# Basic SQL Query on One Relation

---

- **Query 0:** Retrieve the last name of male employees.

**Q0:** SELECT     LNAME  
         FROM     EMPLOYEE  
         WHERE    SEX='M'

- Similar to a SELECTION-PROJECTION pair of relational algebra operations
  - Begin with the relation in the FROM clause
  - Apply the selection indicated by the WHERE clause
  - Apply the projection indicated by the SELECT clause



# Result of Query

---

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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	1968-07-19	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

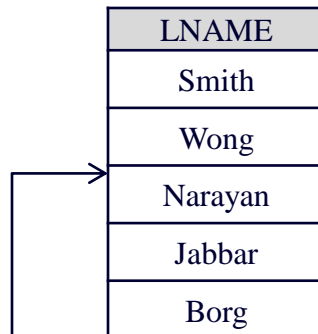
- The answer is a relation with a single attribute, lname, and tuples with the last name of each male employee.

**Q0:** SELECT    LNAME  
         FROM    EMPLOYEE  
         WHERE    SEX='M'

LNAME
Smith
Wong
Narayan
Jabbar
Borg

# Operational Semantics

- Think *a tuple variable* ranging over each tuple of the relation mentioned in FROM
- Check if the “current” tuple satisfied that WHERE clause
- If so, compute the attributes of expressions of the SELECT clause using the components of this tuple.



LNAME
Smith
Wong
Narayan
Jabbar
Borg

**Q0:** SELECT LNAME  
FROM EMPLOYEE  
WHERE SEX='M'

EMPLOYEE	FNAME	MINIT	LNAME	<u>SSN</u>	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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	1968-07-19	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

# Conditions in WHERE-clause

---

- **Query 1:** Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

```
Q1:SELECT BDATE, ADDRESS  
      FROM EMPLOYEE  
      WHERE FNAME='John' AND MINIT='B'  
            AND LNAME='Smith'
```

- Conditions in WHERE clause can use *AND*, *OR*, *NOT(!=)*, and *parentheses* in the usual way **Boolean conditions are built**
- SQL is *case-insensitive*. In general, upper and lower case characters are the same, *except inside quoted strings*.

# UNSPECIFIED WHERE-clause

---

- A *missing WHERE-clause* indicates no selection operation condition; hence, all tuples of the relations in the FROM-clause are selected
  - This is equivalent to the condition WHERE TRUE
- **Query 2:** Retrieve the SSN values for all employees.  
Q2: 

```
SELECT SSN  
FROM EMPLOYEE
```

# ORDER BY

---

## □ Syntax

ORDER BY {column-name | column-position | expression}

[ASC or DESC]

[ NULLS FIRST or NULLS LAST]

{, column-name | column-position | expression}

[ASC or DESC]

[ NULLS FIRST or NULLS LAST]

- NULLS FIRST specifies that NULL values should be returned before non-NULL values.
- NULLS LAST specifies that NULL values should be returned after non-NULL values.

# ORDER BY (contd.)

---

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)
- **Query 12:** Retrieve a list of employees, ordered by the employee's salary.

```
Q12: SELECT LNAME  
      FROM EMPLOYEE  
      ORDER BY SALARY;
```

- **Query13:** Retrieve a list of employees, ordered by the employee's department, and within each department ordered by salary.

```
Q13: SELECT DNO, LNAME, SALRY  
      FROM EMPLOYEE  
      ORDER BY DNO, SALARY;
```

# ORDER BY (contd.)

---

- ❑ The default order is in ascending order of values
- ❑ We can specify the keyword **DESC** if we want a descending order; the keyword **ASC** can be used to explicitly specify ascending order, even though it is the default

```
SELECT LNAME, SALARY  
FROM EMPLOYEE  
ORDER BY SALARY DESC;
```

```
SELECT LNAME, SALARY  
FROM EMPLOYEE  
ORDER BY SALARY ASC;
```

# Outline

---

- Basic SQL Queries
  - A structure of basic SQL query
  - Sort : Order by
  - ☞ Prefix, and Aliases
  - \*, and DISTINCT
  - IN, and IS NULL comparison operators
  - String comparison operators
  - Arithmetic operation
  - Set operation
  - Single-row functions
  - Aggregation functions
  - Grouping query : Group by, Having
- Advanced SQL Queries
- SQL Query with Multiple Relations



# PREFIX Relational Name

---

- ❑ In SQL, we can use the same name for two (or more) attributes as long as the attributes are in *different relations*
- ❑ A query that refers to **two or more attributes with the same name** must *qualify* the attribute name with the relation name **by prefixing** the relation name to the attribute name
- ❑ Example:  
**EMPLOYEE.NAME, DEPARTMENT.NAME**

# ALIASES

---

- ❑ Aliasing can be used in any SQL query for convenience
- ❑ *Aliases* are given to the relation name and columns.
- ❑ Can also use the AS keyword to specify aliases

Q: SELECT E.LNAME AS Name  
FROM EMPLOYEE AS E;

Q: SELECT E.LNAME Name  
FROM EMPLOYEE E;

(Note: In Oracle, 'AS' is not required in FROM clause.)

Q: SELECT E.LNAME AS "Employee Name"  
FROM EMPLOYEE E;

# Outline

---

- Basic SQL Queries
  - A structure of basic SQL query
  - Sort : Order by
  - Prefix, and Aliases
  - ☞ \*, and DISTINCT
  - IN, and IS NULL comparison operators
  - String comparison operators
  - Arithmetic operation
  - Set operation
  - Single-row functions
  - Aggregation functions
  - Grouping query : Group by, Having
- Advanced SQL Queries
- SQL Query with Multiple Relations

# USE OF \*

---

- To retrieve all the attribute values of the selected tuples, a \* is used, which stands for *all the attributes*

Q: SELECT \*  
FROM EMPLOYEE  
WHERE DNO=5

# USE OF DISTINCT

---

- In SQL, duplicate tuples can appear
- To eliminate duplicate tuples in a query result, the keyword **DISTINCT** is used
- For example, the result of QA may have duplicate SALARY values whereas QBA does not have any duplicate values

QA:        SELECT    SALARY  
             FROM     EMPLOYEE

QB:        SELECT    **DISTINCT** SALARY  
             FROM     EMPLOYEE

# Outline

---

- Basic SQL Queries
  - A structure of basic SQL query
  - Sort : Order by
  - Prefix, and Aliases
  - \*, and DISTINCT
  - ☞ IN, and IS NULL comparison operators
  - String comparison operators
  - Arithmetic operation
  - Set operation
  - Single-row functions
  - Aggregation functions
  - Grouping query : Group by, Having
- Advanced SQL Queries
- SQL Query with Multiple Relations

# EXPLICIT SETS with IN

---

- It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause
- **WHERE  $v$  IN  $V$ :** The comparison operator **IN** compares a value  $v$  with a set (or multi-set) of values  $V=(v_1, v_2, \dots, v_n)$  and evaluates to TRUE if  $v$  is one of the elements in  $V$
- **Query 3:** Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

Q3:           SELECT       DISTINCT ESSN  
              FROM       WORKS\_ON  
              WHERE       PNO IN (1, 2, 3)

# BETWEEN AND

---

- Query: Retrieve the name of employees who born in 1960s (i.e., between 01/01/1960 and 12/31/1969)

```
SELECT LNAME
```

```
FROM EMPLOYEE
```

```
WHERE BDATE BETWEEN '01-JAN-60' AND '31-DEC-69'
```

\*\* Note the format of date data. It follows the default format of date of the database system you use. Oracle's default date format is 'DD-MON-YY'.



# NULLS IN SQL QUERIES

---

- ❑ SQL allows queries that check if a value is **NULL** (missing or undefined or not applicable)
- ❑ SQL uses **IS** or **IS NOT** to compare **NULLs** because it considers each NULL value distinct from other NULL values, so *equality comparison is not appropriate*.
- ❑ **Query 4** : Retrieve the names of all employees who do not have supervisors.

**Q4:**

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE       SUPERSSN IS NULL
```

- Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

# SUBSTRING COMPARISON

---

- WHERE clauses can have conditions in which a string can be compared with a pattern, to see if it matches.
- The **LIKE** and **NOT LIKE** comparison operators are used to compare partial strings
  - General form:
    - <Attribute> **LIKE** <pattern> or
    - <Attribute> **NOT LIKE** <pattern>
  - Pattern is a quoted string with % or \_
    - '%' replaces an arbitrary number of characters (any string), and '\_' replaces a single arbitrary character

# SUBSTRING COMPARISON (contd.)

---

- Query 5: Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX' in it.

```
Q5: SELECT  FNAME, LNAME  
      FROM    EMPLOYEE  
      WHERE ADDRESS LIKE '%Houston,TX%'
```

# SUBSTRING COMPARISON (contd.)

---

- Query 6: Retrieve all information (all attribute values) of dependent whose name starts 'A' and consists of 5 characters.

```
Q6: SELECT *  
      FROM DEPENDENT  
      WHERE DEPENDENT_NAME LIKE 'A _ _ _ _'
```

# Summary of SQL Comparison Operators

## ❑ Comparison Operators

Operator	Description	Example
=	Equality test.	<pre>SELECT ENAME "Employee" FROM EMP WHERE SAL = 1500;</pre>
!=, ^=, <>	Inequality test.	<pre>SELECT ENAME FROM EMP WHERE SAL ^= 5000;</pre>
>	Greater than test.	<pre>SELECT ENAME "Employee", JOB "Title" FROM EMP WHERE SAL &gt; 3000;</pre>
<	Less than test.	<pre>SELECT * FROM PRICE WHERE MINPRICE &lt; 30;</pre>
>=	Greater than or equal to test.	<pre>SELECT * FROM PRICE WHERE MINPRICE &gt;= 20;</pre>
<=	Less than or equal to test.	<pre>SELECT ENAME FROM EMP WHERE SAL &lt;= 1500;</pre>
IN	"Equivalent to any member of" test. Equivalent to "= <a href="#">ANY</a> ".	<pre>SELECT * FROM EMP WHERE ENAME IN ('SMITH', 'WARD');</pre>
ANY/ SOME	Compares a value to each value in a list or returned by a query. Must be preceded by =, !=, >, <, <= or >=. Evaluates to <a href="#">FALSE</a> if the query returns no rows.	<pre>SELECT * FROM DEPT WHERE LOC = SOME ('NEW YORK', 'DALLAS');</pre>
NOT IN	Equivalent to "!= <a href="#">ANY</a> ". Evaluates to <a href="#">FALSE</a> if any member of the set is <a href="#">NULL</a> .	<pre>SELECT * FROM DEPT WHERE LOC NOT IN ('NEW YORK', 'DALLAS');</pre>
ALL	Compares a value with every value in a list or returned by a query. Must be preceded by =, !=, >, <, <= or >=. Evaluates to <a href="#">TRUE</a> if the query returns no rows.	<pre>SELECT * FROM emp WHERE sal &gt;= ALL (1400, 3000);</pre>
[NOT] BETWEEN x and y	[Not] greater than or equal to x and less than or equal to y.	<pre>SELECT ENAME, JOB FROM EMP WHERE SAL BETWEEN 3000 AND 5000;</pre>
EXISTS	<a href="#">TRUE</a> if a sub-query returns at least one row.	<pre>SELECT * FROM EMP WHERE EXISTS (SELECT ENAME FROM EMP WHERE MGR IS NULL);</pre>
x [NOT] LIKE y [ESCAPE z]	<a href="#">TRUE</a> if x does [not] match the pattern y. Within y, the character "%" matches any string of zero or more characters except null. The character "_" matches any single character. Any character following <a href="#">ESCAPE</a> is interpreted literally, useful when y contains a percent (%) or underscore (_).	<pre>SELECT * FROM EMP WHERE ENAME LIKE '%E%';</pre>
IS [NOT] NULL	Tests for nulls. This is the only operator that should be used to test for nulls.	<pre>SELECT * FROM EMP WHERE COMM IS NOT NULL AND SAL &gt; 1500;</pre>

# Summary of SQL Logical Operators

---

## □ Logical Operators

Operator	Description	Example
NOT	Returns <b>TRUE</b> if the following condition is <b>FALSE</b> . Returns <b>FALSE</b> if it is <b>TRUE</b> . If it is <b>UNKNOWN</b> , it remains <b>UNKNOWN</b> .	<code>SELECT * FROM EMP WHERE NOT (job IS NULL)</code> <code>SELECT * FROM EMP WHERE NOT (sal BETWEEN 1000 AND 2000)</code>
AND	Returns <b>TRUE</b> if both component conditions are <b>TRUE</b> . Returns <b>FALSE</b> if either is <b>FALSE</b> ; otherwise returns <b>UNKNOWN</b> .	<code>SELECT * FROM EMP WHERE job='CLERK' AND deptno=10</code>
OR	Returns <b>TRUE</b> if either component condition is <b>TRUE</b> . Returns <b>FALSE</b> if both are <b>FALSE</b> . Otherwise, returns <b>UNKNOWN</b> .	<code>SELECT * FROM emp WHERE job='CLERK' OR deptno=10</code> - - -

□ Reference: [https://docs.oracle.com/cd/B19188\\_01/doc/B15917/sqopr.htm](https://docs.oracle.com/cd/B19188_01/doc/B15917/sqopr.htm)

# Outline

---

- Basic SQL Queries
  - A structure of basic SQL query
  - Prefix, and Aliases
  - Sort : Order by
  - \*, and DISTINCT
  - IN, and IS NULL comparison operators
  - String comparison operators
  - ☞ Arithmetic operation
  - Set operation
  - Single-row functions
  - Aggregation functions
  - Grouping query : Group by, Having
- Advanced SQL Queries
- SQL Query with Multiple Relations

# ARITHMETIC OPERATIONS

---

- The standard arithmetic operators '+', '-', '\*', and '/' (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result
- Query 7 : Show the effect of giving all employees a 10% raise on salary.

**Q7:**   SELECT    FNAME, LNAME, 1.1\*SALARY  
          FROM     EMPLOYEE;



# Constant Expressions

---

- Constant value is presented with a single quotation.
- Example

**Q7-2:** SELECT 'The 10% increased salary of ', FNAME, 'is',  
          1.1\*SALARY, '. '  
FROM EMPLOYEE;

# Concatenation Operator

---

- || concatenates character strings and CLOB data
- Example

**Q7-3:** `SELECT FNAME||' '||LNAME||' '`

`'likes the 10% increased salary, $'||' ||1.1*SALARY||'. '`

`AS "Survey result"`

`FROM EMPLOYEE;`

# Summary of SQL Arithmetic and Character Operators

---

## □ Arithmetic Operators

Operator	Description	Example
+ (unary)	Makes operand positive	<code>SELECT +3 FROM DUAL;</code>
- (unary)	Negates operand	<code>SELECT -4 FROM DUAL;</code>
/	Division (numbers and dates)	<code>SELECT SAL / 10 FROM EMP;</code>
*	Multiplication	<code>SELECT SAL * 5 FROM EMP;</code>
+	Addition (numbers and dates)	<code>SELECT SAL + 200 FROM EMP;</code>
-	Subtraction (numbers and dates)	<code>SELECT SAL - 100 FROM EMP;</code>

## □ Character Operators

Operator	Description	Example
	Concatenates character strings	<code>SELECT 'The Name of the employee is: '    ENAME FROM EMP;</code>

# Outline

---

- Basic SQL Queries
  - A structure of basic SQL query
  - Prefix, and Aliases
  - Sort : Order by
  - \*, and DISTINCT
  - IN, and IS NULL comparison operators
  - String comparison operators
  - Arithmetic operation
  - ☞ Set operation
  - Single-row functions
  - Aggregation functions
  - Grouping query : Group by, Having
- Advanced SQL Queries
- SQL Query with Multiple Relations

# SET OPERATIONS

---

- ❑ SQL has directly incorporated some set operations
- ❑ There is a union operation (UNION or UNION ALL), and in *some versions* of SQL there are set difference (MINUS) and intersection (INTERSECT) operations
- ❑ The set operations apply only to *union compatible relations*; the two query result sets must have the same number of attributes and the attribute characteristics (e.g., data type) must be the same.
- ❑ The resulting relations of these set operations are sets of tuples; *duplicate tuples are eliminated from the result* except UNION ALL

# SET OPERATION: UNION, UNION ALL

---

Q: SELECT FNAME  
FROM EMPLOYEE;  
**UNION**  
SELECT DEPENDENT\_NAME  
FROM DEPENDENT;

Q: SELECT FNAME, BDATE  
FROM EMPLOYEE  
**UNION**  
SELECT DEPENDENT\_NAME, BDATE  
FROM DEPENDENT;

Q: SELECT FNAME, BDATE  
FROM EMPLOYEE  
**UNION ALL**  
SELECT DEPENDENT\_NAME, BDATE  
FROM DEPENDENT;

# SET OPERATION: INTERSECT

---

- Query : List the SSN of employee who work for both project no 1 and project no 2.

Q: SELECT ESSN  
FROM WORKS\_ON  
WHERE PNO = 1;

Q: SELECT ESSN  
FROM WORKS\_ON  
WHERE PNO = 2;

Q: SELECT ESSN  
FROM WORKS\_ON  
WHERE PNO = 1  
**INTERSECT**  
SELECT ESSN  
FROM WORKS\_ON  
WHERE PNO = 2;

# SET OPERATION: MINUS (DIFFERENCE)

---

- Query : List the SSN of employees who work for project no 61 but not for project no 62.

Q: `SELECT ESSN  
FROM WORKS_ON  
WHERE PNO = 61  
MINUS  
SELECT ESSN  
FROM WORKS_ON  
WHERE PNO = 62;`



# Outline

---

- Basic SQL Queries
  - A structure of basic SQL query
  - Sort : Order by
  - Prefix, and Aliases
  - \*, and DISTINCT
  - IN, and IS NULL comparison operators
  - String comparison operators
  - Arithmetic operation
  - Set operation
  - ☞ Single-row functions
  - Aggregation functions
  - Grouping query : Group by, Having
- Advanced SQL Queries
- SQL Query with Multiple Relations

# SQL Functions

---

- ❑ SQL functions are built into relational database systems and available for use in various appropriate SQL statements.
- ❑ SQL functions
  - Single-Row Functions
    - : return a single result row for every row of a queried table or view.
  - Aggregation Functions
    - : return a single result row based on groups of rows, rather than on single rows.

# SINGLE-ROW FUNCIONTS

---

- ❑ Single-row functions return a single result row for every row of a queried table or view.
- ❑ Single-row functions can appear in SELECT lists, WHERE clauses, HAVING clauses, etc.
- ❑ Single-row functions
  - Numeric functions
  - Char functions
  - Date functions
  - Conversion functions
- ❑ Reference: for Oracle,  
<https://docs.oracle.com/database/121/SQLRF/functions002.htm#SQLRF51178>

# Example: Numeric Functions

---

Most numeric functions return number values.

## ❑ SIGN

- SELECT SIGN(salary) FROM employee;
- SELECT SIGN(10), SIGN (0), SIGN(-10) FROM dual;

## ❑ ABS

- SELECT ABS(-15) “Absolute” FROM dual;

## ❑ ROUND

- SELECT ROUND(15.193, 1) “Round” FROM dual;

## ❑ TRUNC

- SELECT TRUNC(15.193, 1) “Truncate” FROM dual;

## ❑ POWER

- SELECT POWER(3, 2) “Raised” FROM dual;

# Example: Character Functions

---

Character functions to return character values:

- **UPPER / LOWER**

- SELECT UPPER(lname), LOWER(fname) FROM employee;

- **SUBSTR** (*string, start\_position, substring\_length*)

- SELECT SUBSTR('ABCDEFGFG', 3, 4) FROM dual;

- SELECT address, SUBSTR(address, -2, 2) "State" FROM employee;

- **LTRIM / RTRIM** (*string, trim\_string*)

- SELECT RTRIM('Tech123123', '123') FROM dual;

- **LPAD / RPAD** (*string, length [, pad-exp]*)

- SELECT RPAD('Morrison', 12, '>' ) FROM dual;

- SELECT lname, salary, rpad(' ', salary/1000, '\*') "salary bar"  
FROM employee

# Example: Character Functions

---

Character functions to return number values:

- **LENGTH**

- `SELECT LENGTH(lname) FROM employee;`

- **INSTR** (*string, sub\_string, start\_position, occurrence*)

- `SELECT INSTR('Tech on the net', 'e') FROM DUAL;`

- `SELECT INSTR('Tech on the net', 'e', 1,1 ) FROM DUAL;`

- `SELECT INSTR('Tech on the net', 'e', 1,2 ) FROM DUAL; -- the second occurrence of e`

- `SELECT INSTR('Tech on the net', 'e', -3,2 ) FROM DUAL;`

# Example: DateTime Functions

---

Datetime functions operates on date(DATE), timestamp (TIMESTAMP) and interval values.

- **ADD\_MONTHS**

- `SELECT ADD_MONTHS(bdate, 12) FROM employee;`

- **SYSDATE**

- `SELECT SYSDATE FROM dual;`

- **MONTHS\_BETWEEN**

- `SELECT MONTHS_BETWEEN (bdate, SYSDATE) "Months"`  
`FROM employee;`

- `SELECT TRUNC(MONTHS_BETWEEN (bdate, SYSDATE)/12)`  
`"Age" FROM employee;`

# Example: Conversion Functions

---

- ❑ **TO\_CHAR** (*datetime*)
  - SELECT TO\_CHAR(bdate, 'MM/DD/YYYY HH:24HI:MI' ) FROM employee;
- ❑ **TO\_CHAR**(*number*)
  - SELECT TO\_CHAR(10) FROM dual;
- ❑ **TO\_NUMBER**(*char*)
  - SELECT TO\_NUMBER('123') FROM dual;
- ❑ **TO\_DATE**
  - SELECT TO\_CHAR(bdate, 'MM/DD/YYYY HH:24HI:MI' ) FROM employee;
  - For date format, refer to [https://docs.oracle.com/cd/B28359\\_01/server.111/b28286/sql\\_elements004.htm#SQLRF00210](https://docs.oracle.com/cd/B28359_01/server.111/b28286/sql_elements004.htm#SQLRF00210)
- ❑ And so on.



# Outline

---

- Basic SQL Queries
  - A structure of basic SQL query
  - Sort : Order by
  - Prefix, and Aliases
  - \*, and DISTINCT
  - IN, and IS NULL comparison operators
  - String comparison operators
  - Arithmetic operation
  - Set operation
  - Single-row functions
  - ☞ Aggregation functions
  - Grouping query : Group by, Having
- Advanced SQL Queries
- SQL Query with Multiple Relations

# AGGREGATE FUNCTIONS

---

- ❑ Aggregate functions return a single result row based on groups of rows, rather than on single rows.
- ❑ Aggregate functions can appear in SELECT lists, in ORDER BY and HAVING clauses.
- ❑ Aggregate functions include **COUNT, SUM, MAX, MIN, AVG, MEDIAN**, etc.
- ❑ Some SQL implementations *may not allow more than one function* in the SELECT-clause

# AGGREGATE FUNCTIONS (contd.)

---

- Query 8: Find the maximum salary, the minimum salary, and the average salary among all employees.

Q8: `SELECT MAX(SALARY),  
MIN(SALARY), AVG(SALARY)  
FROM EMPLOYEE  
WHERE DNO =8;`

# AGGREGATE FUNCTIONS (contd.)

---

- Query 9: Retrieve the total number of employees in the company.

**Q9:** SELECT COUNT (\*)  
FROM EMPLOYEE;

SELECT COUNT (\*), COUNT(ssn), COUNT(superssn)  
FROM EMPLOYEE;

- Query: Retrieve the total number of employees working in department 5

**Q:** SELECT COUNT (\*)  
FROM EMPLOYEE  
WHERE DNO=5;

# AGGREGATE FUNCTIONS (contd.)

---

- Many other aggregation functions for analytical analysis
  - <https://docs.oracle.com/database/121/SQLRF/functions003.htm#SQLRF20035>

# Outline

---

- Basic SQL Queries
  - A structure of basic SQL query
  - Sort : Order by
  - Prefix, and Aliases
  - \*, and DISTINCT
  - IN, and IS NULL comparison operators
  - String comparison operators
  - Arithmetic operation
  - Set operation
  - Single-row functions
  - Aggregation functions
  - ☞ Grouping query : Group by, Having
- Advanced SQL Queries
- SQL Query with Multiple Relations

# GROUPING

---

- ❑ In many cases, we want to apply the aggregate functions to *subgroups of tuples* in a relation
- ❑ SQL has a **GROUP BY**-clause for specifying the grouping attributes.
- ❑ Each subgroup of tuples consists of the set of tuples that have the *same value* for the *grouping attribute(s)*
- ❑ The function is applied to each subgroup independently
- ❑ **SELECT**-clause has only attributes specifying in **GROUP BY** and/or aggregate functions. No other attributes can not be listed in **SELECT**.

# GROUPING (contd.)

---

- Query 10: For each department, retrieve the department number, the number of employees in the department, and their average salary.

**Q10:** `SELECT DNO, COUNT (*), AVG (SALARY)`  
`FROM EMPLOYEE`  
`GROUP BY DNO`

- In Q10, the EMPLOYEE tuples are divided into groups-
  - Each group having the same value for the grouping attribute DNO
- The COUNT and AVG functions are applied to each such group of tuples separately
- The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples
- A join condition can be used in conjunction with grouping



# THE HAVING-CLAUSE

---

- Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*
- The **HAVING**-clause is used for specifying a selection condition on groups (rather than on individual tuples)

# THE HAVING-CLAUSE (contd.)

---

- Query 11: For each department *on which more than three employee work*, retrieve the department number, the number of employees in the department, and their average salary.

**Q11:** SELECT DNO, COUNT (\*), AVG (SALARY)  
FROM EMPLOYEE  
GROUP BY DNO  
HAVING COUNT (\*) > 3

# Outline

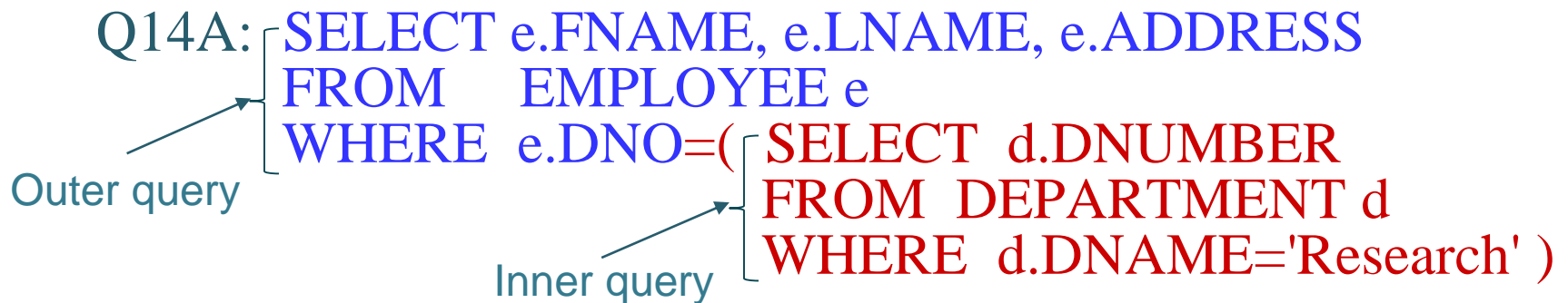
---

- Basic SQL Queries
- ☞ Advanced SQL Queries
  - Nested query
  - Correlated query
  - EXISTS operator
- SQL Query with Multiple Relations
- Other DML
- View
- Other database objects

# NESTING OF QUERIES

---

- A complete SELECT query, called a *nested query*, can be specified within the WHERE-clause of another query, called the *outer query*
- **Query 14:** Retrieve the name and address of all employees who work for the 'Research' department.

Q14A:   
Outer query      Inner query

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- In general, we can have several levels of nested queries

# NESTING OF QUERIES

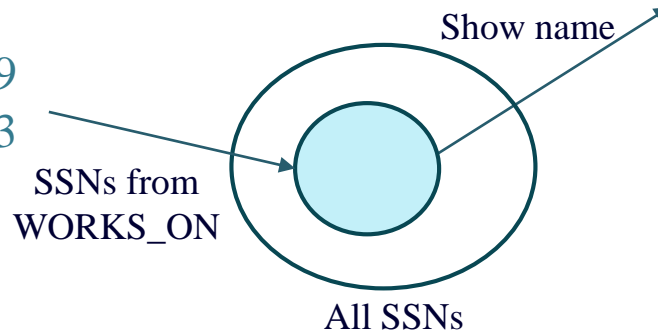
Q14-2: `SELECT e.LNAME`  
`FROM EMPLOYEE e`  
`WHERE e.SSN IN` ( `SELECT w.ESSN`  
`FROM WORKS_ON w`  
`WHERE w.PNO =1`);

Outer query

Sub query

1. The subquery is processed first and an intermediate results table created:

ESSN  
123456789  
453453453



2. The outer query returns the requested employee information (i.e., lname) of employee who work in projects included in the intermediate results table.

LNAME  
English  
Smith

# NESTING OF QUERIES (contd.)

---

- If the nested query result is expected to have ONE value, equality (=) operator can be used. Otherwise, IN operator is safe.

```
SELECT * FROM EMPLOYEE e
WHERE e.DNO IN ( SELECT d.DNUMBER
                  FROM DEPARTMENT d
                  WHERE d.DNAME like 'H%');
```

- A reference to an *unqualified attribute* refers to the relation declared in the *innermost nested query*. It is recommended to use table alias.

```
SELECT PNO FROM WORKS_ON
WHERE ESSN IN ( SELECT ESSN
                  FROM DEPENDENT
                  WHERE DEPENDENT_NAME ='Sam');
```

# ALL Operator

---

- **ALL** is used to compare a value to a list or subquery.

"x = ALL (...)": The value must match all the values in the list to evaluate to TRUE.

"x != ALL (...)": The value must not match any values in the list to evaluate to TRUE.

"x > ALL (...)": The value must be greater than the biggest value in the list to evaluate to TRUE.

"x < ALL (...)": The value must be smaller than the smallest value in the list to evaluate to TRUE.

```
SELECT e.FNAME, e.SALARY
FROM EMPLOYEE e
WHERE e.SALARY > ALL (50000, 60000, 70000);
```

```
SELECT e.FNAME, e.SALARY
FROM EMPLOYEE e
WHERE e.SALARY >= ALL (SELECT e2.SALARY
                       FROM EMPLOYEE e2
                       WHERE e2.dno=1);
```

# ANY Operator

---

- **ANY** is also used to compare a value to a list or subquery.

"x = ANY (...)": The value must match one or more values in the list to evaluate to TRUE.

"x != ANY (...)": The value must not match one or more values in the list to evaluate to TRUE.

"x > ANY (...)": The value must be greater than the smallest value in the list to evaluate to TRUE.

"x < ANY (...)": The value must be smaller than the biggest value in the list to evaluate to TRUE.

```
SELECT e.FNAME, e.SALARY
FROM EMPLOYEE e
WHERE e.SALARY < ANY (50000, 60000, 70000);
```

```
SELECT e.FNAME, e.SALARY
FROM EMPLOYEE e
WHERE e.SALARY <= ANY (SELECT e2.SALARY
                        FROM EMPLOYEE e2
                        WHERE e2.dno=1);
```



# NESTING OF QUERIES (Again)

---

- **Query 14:** Retrieve the name and address of all employees who work for the 'Research' department.

Q14A: SELECT e.FNAME, e.LNAME, e.ADDRESS  
FROM EMPLOYEE e  
WHERE e.DNO=( SELECT d.DNUMBER  
FROM DEPARTMENT d  
WHERE d.DNAME='Research' )

- This nested query is *not correlated* with the outer query

# CORRELATED NESTED QUERIES

---

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query*, the two queries are said to be *correlated*
  - The result of a correlated nested query is different for each tuple (or combination of tuples) of the relation(s) in the outer query
- **Query 15:** Retrieve the name of each employee who has a dependent with the same first name as the employee.

**Q15:** SELECT e.FNAME, e.LNAME  
FROM EMPLOYEE e  
WHERE e.SSN IN (SELECT d.ESSN  
FROM DEPENDENT d  
WHERE d.ESSN=e.SSN  
AND d.DEPENDENT\_NAME=e.FNAME)

# CORRELATED NESTED QUERIES

---

- **Query 15-1:** Retrieve the name of male employee who has a son.

```
SELECT e.FNAME, e.LNAME
FROM EMPLOYEE e
WHERE e.SEX='M'
AND e.SSN IN (SELECT d.ESSN
              FROM DEPENDENT d
              WHERE d.ESSN=e.SSN
              AND d.SEX='M');
```

# EXISTS Operator

---

- ❑ **EXISTS** is used to check whether the result of a correlated nested query is empty (contains no tuples) or not
- ❑ We can formulate Query 15 in an alternative form that uses EXISTS as Q15B

**Query 15:** Retrieve the name of each employee who has a dependent with the same first name as the employee.

## Q15A:

```
SELECT e.FNAME, e.LNAME
FROM EMPLOYEE e
WHERE e.SSN IN
    (SELECT d.ESSN
     FROM DEPENDENT d
     WHERE d.ESSN=e.SSN
     AND
     d.DEPENDENT_NAME=e.FNAME)
```

## Q15B:

```
SELECT e.FNAME, e.LNAME
FROM EMPLOYEE e
WHERE EXISTS (SELECT ''
              FROM DEPENDENT d
              WHERE e.SSN=d.ESSN
              AND
              e.FNAME=d.DEPENDENT_NAME)
```

# EXISTS (contd.)

---

- **Query 15-2:** Retrieve the names of employees who have any dependent.

```
SELECT e.FNAME, e.LNAME
      FROM EMPLOYEE e
      WHERE EXISTS (SELECT *
                    FROM DEPENDENT d
                    WHERE e.SSN=d.ESSN)
```

# NOT EXISTS Operator

---

- **Query 17:** Retrieve the names of employees who have no dependents. (Use NOT EXISTS)

```
Q17: SELECT e.FNAME, e.LNAME
      FROM   EMPLOYEE e
      WHERE  NOT EXISTS (SELECT*
                        FROM DEPENDENT d
                        WHERE  s.SSN=d.ESSN)
```

- In Q17, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist*, the EMPLOYEE tuple is selected
- EXISTS and NOT EXISTS operators are necessary for the expressive power of SQL. These operators shows better query performance than equivalent operation with =, IN, !=, or NOT IN.

# Derived Table

---

```
SELECT *  
FROM (SELECT fname FROM employee);
```

```
SELECT a.fname, d.dname  
FROM (SELECT fname, dno FROM employee) a, department d  
WHERE a.fname LIKE 'S%'  
AND a.dno=d.dnumber;
```

# Outline

---

- ❑ Basic SQL Queries
- ❑ Advanced SQL Queries
- ☞ SQL Query with Multiple Relations
  - Join, Outer Join
- ❑ Other SQL DML Statements
- ❑ View
- ❑ Other database objects



# Query with Multiple Relations

---

- UNSPECIFIED WHERE-clause with multiple relations

Q: `SELECT e.SSN, d.DNAME  
FROM EMPLOYEE e, DEPARTMENT d`

- If more than one relation is specified in the FROM-clause *and* there is **no join condition**, then **the *CARTESIAN PRODUCT*** of tuples is selected

# UNSPECIFIED WHERE-clause with Multiple tables

```
SELECT *  
FROM EMPLOYEE, DEPARTMENT
```

EMPLOYEE

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
Andy	C		Vile	22222...	21-JUN-44	1967 Jorda...	M	53000	222222200	7
Brad	C		Knight	11111...	13-FEB-68	176 Main St...	M	44000	111111100	6
Evan	E		Wallis	22222...	16-JAN-58	134 Pelham...	M	92000	(null)	7
Josh	U		Zell	22222...	22-MAY-54	266 McGrad...	M	56000	222222200	7
Jared	D		James	11111...	10-OCT-66	123 Peachtr...	M	85000	(null)	6
Justin	n		Mark	11111...	12-JAN-66	2342 May, ...	M	40000	111111100	6
Jon	C		Jones	11111...	14-NOV-67	111 Allgood...	M	45000	111111100	6
John	C		James	55555...	30-JUN-75	7676 Bloomi...	M	81000	(null)	6
Alex	D		Freed	44444...	09-OCT-50	4333 Pillsbu...	M	89000	(null)	7
Ahmad	V		Jabbar	98798...	29-MAR-59	980 Dallas, ...	M	25000	987654321	4
Joyce	A		English	45345...	31-JUL-62	5631 Rice, ...	F	25000	333445555	5
Ramesh	K		Narayan	66688...	15-SEP-52	971 Fire Oa...	M	38000	333445555	5
Alicia	J		Zelaya	99988...	19-JUL-58	3321 Castle...	F	25000	987654321	4
John	B		Smith	12345...	09-JAN-55	731 Fondre...	M	30000	333445555	5
Jennifer	S		Wallace	98765...	20-JUN-31	291 Berry, ...	F	43000	888665555	4
Franklin	T		Wong	33344...	08-DEC-45	638 Voss, H...	M	40000	888665555	5
James	E		Borg	88866...	10-NOV-27	450 Stone, ...	M	55000	(null)	1
Tom	G		Brand	22222...	16-DEC-66	112 Third St...	M	62500	222222200	7
Jenny	F		Vos	22222...	11-NOV-67	263 Mayber...	F	61000	222222201	7
Chris	A		Carter	22222...	21-MAR-60	565 Jordan,...	F	43000	222222201	7

DEPARTMENT

DEPARTMENT	DNUMBER	DNAME	MGRSSN	MGRSTARTDATE
	5	Research	333445555	22-MAY-78
	4	Administration	987654321	01-JAN-85
	1	Headquarters	888665555	19-JUN-71
	6	Software	111111100	15-MAY-99
	7	Hardware	444444400	15-MAY-98
	8	Sales	555555500	01-JAN-97

# SPECIFIED WHERE-clause with Multiple tables

```
SELECT *
FROM employee e , department d
WHERE e.dno= d.dnumber
```

EMPNO	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
7369	Andy	C	Vile	22222...	21-JUN-44	1967 Jorda...	M	53000	222222200	7
7372	Brad	C	Knight	11111...	13-FEB-68	176 Main St...	M	44000	111111100	6
7373	Evan	E	Wallis	22222...	16-JAN-58	134 Pelham...	M	92000	(null)	7
7374	Josh	U	Zell	22222...	22-MAY-54	266 McGrad...	M	56000	222222200	7
7375	Jared	D	James	11111...	10-OCT-66	123 Peachtr...	M	85000	(null)	6
7376	Justin	n	Mark	11111...	12-JAN-66	2342 May, ...	M	40000	111111100	6
7377	Jon	C	Jones	11111...	14-NOV-67	111 Allgood...	M	45000	111111100	6
7378	John	C	James	55555...	30-JUN-75	7676 Bloomi...	M	81000	(null)	6
7379	Alex	D	Freed	44444...	09-OCT-50	4333 Pillsbu...	M	89000	(null)	7
7380	Ahmad	V	Jabbar	98798...	29-MAR-59	980 Dallas, ...	M	25000	987654321	4
7381	Joyce	A	English	45345...	31-JUL-62	5631 Rice, ...	F	25000	333445555	5
7382	Ramesh	K	Narayan	66688...	15-SEP-52	971 Fire Oa...	M	38000	333445555	5
7383	Alicia	J	Zelaya	99988...	19-JUL-58	3321 Castle...	F	25000	987654321	4
7384	John	B	Smith	12345...	09-JAN-55	731 Fondre...	M	30000	333445555	5
7385	Jennifer	S	Wallace	98765...	20-JUN-31	291 Berry, ...	F	43000	888665555	4
7386	Franklin	T	Wong	33344...	08-DEC-45	638 Voss, H...	M	40000	888665555	5
7387	James	E	Borg	88866...	10-NOV-27	450 Stone, ...	M	55000	(null)	1
7388	Tom	G	Brand	22222...	16-DEC-66	112 Third St...	M	62500	222222200	7
7389	Jenny	F	Vos	22222...	11-NOV-67	263 Mayber...	F	61000	222222201	7
7390	Chris	A	Carter	22222...	21-MAR-60	565 Jordan,...	F	43000	222222201	7

DEPARTMENT

DNUMBER	DNAME	MGRSSN	MGRSTARTDATE
5	Research	333445555	22-MAY-78
4	Administration	987654321	01-JAN-85
1	Headquarters	888665555	19-JUN-71
6	Software	111111100	15-MAY-99
7	Hardware	444444400	15-MAY-98
8	Sales	555555500	01-JAN-97

(Alternative)

```
SELECT *
FROM employee e JOIN department d ON e.dno= d.dnumber
```

# Joined Relations

---

- ❑ Specify **join conditions** for querying from multiple relations.
- ❑ Natural JOIN, inner JOIN
- ❑ Allows the user to specify different types of joins, e.g., LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN
- ❑ It is extremely important not to overlook specifying any selection and join conditions; otherwise, incorrect and very large relations may result

# SQL Query on Two Relations

---

- **Query 14:** Retrieve the name and address of all employees who work for the 'Research' department.

**Q14B:** SELECT e.FNAME, e.LNAME, e.ADDRESS  
FROM EMPLOYEE e, DEPARTMENT d  
WHERE d.DNAME='Research' AND  
d.DNUMBER=e.DNO

- Similar to a SELECTION-PROJECTION-JOIN sequence of relational algebra operations
  - (DNAME='Research') is a selection condition (corresponds to a SELECTION operation in relational algebra)
  - (DNUMBER=DNO) is a join condition (corresponds to a JOIN operation in relational algebra)

## Join Approach

```
SELECT *
FROM EMPLOYEE e, DEPARTMENT d
WHERE d.DNAME = 'Research'
      AND d.DNUMBER = e.DNO
```

WHERE d.DNAME = 'Research'

### DEPARTMENT

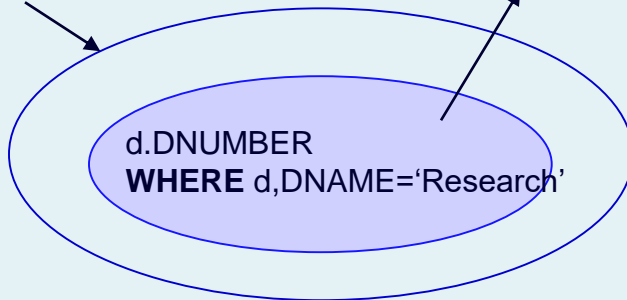
DNUMBER	DNAME	MGRSSN	MGRSTARTDATE
5	Research	333445555	22-MAY-78
4	Administration	987654321	01-JAN-85
1	Headquarters	888665555	19-JUN-71
6	Software	111111100	15-MAY-99
7	Hardware	444444400	15-MAY-98
8	Sales	555555500	01-JAN-97

WHERE d.DNUMBER = e.DNO

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
Joyce	A	English	45345...	31-JUL-62	5631 Rice, ...	F	25000	333445555	5
Ramesh	K	Narayan	66688...	15-SEP-52	971 Fire Oa...	M	38000	333445555	5
Alicia	J	Zelaya	99988...	19-JUL-58	3321 Castle...	F	25000	987654321	4
John	B	Smith	12345...	09-JAN-55	731 Fondre...	M	30000	333445555	5
Jennifer	S	Wallace	98765...	20-JUN-31	291 Berry, ...	F	43000	888665555	4
Franklin	T	Wong	33344...	08-DEC-45	638 Voss, H...	M	40000	888665555	5
James	E	Borg	88866...	10-NOV-27	450 Stone, ...	M	55000 (null)		1

SELECT

All DNUMBERS



Show employee data for employees with this department number

## Nested Query Approach

```
SELECT *
FROM EMPLOYEE e
WHERE e.DNO = (SELECT d.DNUMBER
                FROM DEPARTMENT d
                WHERE d.DNAME='Research' )
```

# Alternative SQL JOIN Expression

---

- Can specify a "joined relation" in the FROM-clause

- Examples:

```
Q14B: SELECT  e.FNAME, e.LNAME, e.ADDRESS
        FROM    EMPLOYEE e, DEPARTMENT d
        WHERE   d.DNAME='Research'
        AND     d.DNUMBER=e.DNO
```

- could be written as:

```
Q14B: SELECT  e.FNAME, e.LNAME, e.ADDRESS
        FROM    (EMPLOYEE e JOIN DEPARTMENT d
                  ON d.DNUMBER=e.DNO)
        WHERE   d.DNAME='Research'
```

# SQL Query on Multiple Relations

---

- **Query 20:** For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

Q20A: SELECT p.PNUMBER, p.DNUM,  
          e.LNAME, e.BDATE, e.ADDRESS  
FROM PROJECT p, DEPARTMENT d, EMPLOYEE e  
WHERE p.DNUM=d.DNUMBER AND d.MGRSSN=e.SSN  
      AND p.PLOCATION='Stafford'

- In Q20, there are two join conditions
- The join condition DNUM=DNUMBER relates a project to its controlling department
- The join condition MGRSSN=SSN relates the controlling department to the employee who manages that department



# Alternative SQL JOIN with Multiple Relations

---

- Q20A could be written as follows;

Q20A: SELECT PNUMBER, DNUM, LNAME, BDATE, ADDRESS  
FROM PROJECT, DEPARTMENT, EMPLOYEE  
WHERE DNUM=DNUMBER AND MGRSSN=SSN  
AND PLOCATION='Stafford'

Q20B: SELECT PNUMBER, DNUM, LNAME, BDATE, ADDRESS  
FROM ((PROJECT JOIN DEPARTMENT ON  
DNUM=DNUMBER) JOIN  
EMPLOYEE ON MGRSSN=SSN)  
WHERE PLOCATION='Stafford'

# Self Join

---

- **Self join:** An operation to join a table to itself
- **Query 21:** For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

**Q21:** SELECT e.FNAME, e.LNAME, s.FNAME, s.LNAME  
FROM EMPLOYEE e, EMPLOYEE s  
WHERE e.SUPERSSN=s.SSN

- Need the alternate relation names E and S which are called *aliases* or *tuple variables* for the EMPLOYEE relation
- We can think of E and S as two different *copies* of EMPLOYEE; E represents employees in role of *supervisees* and S represents employees in role of *supervisors*

# Outlier Join

---

- ❑ An **outer join** retrieves all rows that satisfy the join condition, plus unmatched rows in one or both tables.
- ❑ When a row with unmatched columns is retrieved, any columns from the other table that are included in the result set are given null values.

Joins of this type	Keep unmatched rows from
Left outer join	The first (left) table
Right outer join	The second (right) table
Full outer join	Both tables

- ❑ The explicit syntax for an outer join

```
SELECT select_list FROM table_1
    {LEFT|RIGHT|FULL} [OUTER] JOIN table_2
    ON join_condition_1
    [{LEFT|RIGHT|FULL} [OUTER] JOIN table_3
    ON join_condition_2]...
```

# Inner Join Query

---

- **Query 21:** For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

**Q21:** SELECT e.FNAME, e.LNAME, s.FNAME, s.LNAME  
FROM EMPLOYEE e JOIN EMPLOYEE s  
ON e.SUPERSSN=s.SSN

# Left Outer Join Query

---

**Q21:** SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME  
FROM EMPLOYEE E JOIN EMPLOYEE S  
ON E.SUPERSSN=S.SSN

- To include to include the unmatched rows from the first (left) table

**Q21\_A:** SELECT e.FNAME, e.LNAME, s.FNAME, s.LNAME  
FROM (EMPLOYEE e **LEFT OUTER JOIN**  
EMPLOYEE s **ON** e.SUPERSSN=s.SSN)

In Oracle, (+) is also used for outer join.

**Q21\_A:** SELECT e.FNAME, e.LNAME, s.FNAME, s.LNAME  
FROM EMPLOYEE e, EMPLOYEE s  
WHERE **e.SUPERSSN=s.SSN(+)**

# Right Outer Join Query

---

- What means the following query statement?

**Q21\_B:** SELECT e.FNAME, e.LNAME, s.FNAME, s.LNAME  
FROM (EMPLOYEE e RIGHT OUTER JOIN  
EMPLOYEE s ON e.SUPERSSN=s.SSN)

- In Oracle, (+) is also used for outer join.

**Q21\_B:** SELECT e.FNAME, e.LNAME, s.FNAME, s.LNAME  
FROM EMPLOYEE e, EMPLOYEE s  
WHERE e.SUPERSSN(+)=s.SSN

# Full Outer Join Query

---

- What means the following query statement?

**Q21\_C:** SELECT e.FNAME, e.LNAME, s.FNAME, s.LNAME  
FROM (EMPLOYEE e FULL OUTER JOIN  
EMPLOYEE s ON e.SUPERSSN=s.SSN)

# Summary of SQL Queries

---

- A query in SQL can consist of up to six clauses, but only the first two, **SELECT** and **FROM**, are mandatory. The clauses are specified in the following order:

<b>SELECT</b>	<attribute list>
<b>FROM</b>	<table list>
<b>[WHERE</b>	<condition>]
<b>[GROUP BY</b>	<grouping attribute(s)>]
<b>[HAVING</b>	<group condition>]
<b>[ORDER BY</b>	<attribute list>]



# Summary of SQL Queries (contd.)

---

- ❑ The SELECT-clause lists the attributes or functions to be retrieved
- ❑ The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- ❑ The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- ❑ GROUP BY specifies grouping attributes
- ❑ HAVING specifies a condition for selection of groups
- ❑ ORDER BY specifies an order for displaying the result of a query
  - A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause

# Outline

---

- ❑ Basic SQL Queries
- ❑ Advanced SQL Queries
- ❑ SQL Query with Multiple Relations
- ☞ Other SQL DML Statements
  - Insert, Delete and Update
- ❑ View
- ❑ Other database objects

# Specifying Updates in SQL

---

- There are three SQL commands to modify the database: **INSERT**, **DELETE**, and **UPDATE**

# INSERT

---

- In its simplest form, it is used to add one tuple to a relation

- Example U1:

**INSERT INTO** EMPLOYEE

**VALUES** ('Richard','K','Marini', '653298653', '30-DEC-52',  
'98 Oak Forest,Katy,TX', 'M', 37000,'987654321', 4 )

- If attributes are not explicitly specified, attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command

# INSERT (contd.)

---

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple
  - Attributes with NULL values can be left out
- Example: Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

```
U1A: INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)  
      VALUES ('Richard', 'Marini', '653298653')
```

# INSERT Multiple Tuples

---

- Another variation of INSERT allows insertion of *multiple tuples* resulting from a query into a relation
- Example: Suppose we want to create a temporary table that has the department name, number of employees, and total salaries for each department.

```
U3A: CREATE TABLE DEPTS_INFO
      (DEPT_NAME   VARCHAR(10),
       NO_OF_EMPS  INTEGER,
       TOTAL_SAL   INTEGER);
```

```
U3B: INSERT INTO DEPTS_INFO (DEPT_NAME,
                             NO_OF_EMPS, TOTAL_SAL)
      SELECT DNAME, COUNT (*), SUM (SALARY)
      FROM   DEPARTMENT, EMPLOYEE
      WHERE DNUMBER=DNO
      GROUP BY DNAME ;
```

# DELETE

---

- Removes tuples from a relation

DELETE FROM table\_name ...

- Includes a WHERE-clause to select the tuples to be deleted
- A missing WHERE-clause specifies that *all tuples* in the relation are to be deleted; the table then becomes an empty table
- Tuples are deleted from only *one table* at a time (unless CASCADE is specified on a referential integrity constraint)

# DELETE (contd.)

---

□ Examples:

U4A:        **DELETE FROM** EMPLOYEE  
             WHERE LNAME='Brown'

U4B:        DELETE FROM EMPLOYEE  
             WHERE SSN='123456789'

U4C:        DELETE FROM EMPLOYEE  
             WHERE DNO IN  
                      (SELECT DNUMBER  
                      FROM DEPARTMENT  
                      WHERE DNAME='Research')

U4D:        DELETE FROM EMPLOYEE



# UPDATE

---

- Used to modify attribute values of one or more selected tuples

UPDATE table\_name SET ...

- A SET-clause specifies the attributes to be modified and their new values
- A WHERE-clause selects the tuples to be modified

# UPDATE (contd.)

---

- Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
U5:      UPDATE  PROJECT
          SET    PLOCATION = 'Bellaire',
                DNUM = 5
          WHERE   PNUMBER=10
```

# UPDATE (contd.)

---

- Example: Give all employees in the 'Research' department a 10% raise in salary.

```
U6:UPDATE      EMPLOYEE
      SET       SALARY = SALARY * 1.1
      WHERE     DNO IN (SELECT  DNUMBER
                        FROM DEPARTMENT
                        WHERE DNAME='Research')
```

- In this request, the modified SALARY value depends on the original SALARY value in each tuple
  - The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification
  - The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification

# Enforcement of Constraints

---

- Important Note: Only the constraints specified in the DDL commands (e.g., create table... ) are automatically enforced by the DBMS when updates(insert, delete, update) are applied to the database
  - E.g., When user tries to insert a duplicate primary key value, DBMS send an error message.

# Transaction

---

- A database **transaction** is a single unit of work performed within a database management system, sometimes made up of multiple operations.
- A transaction generally represents any change in a database, that is, a collection of SQL statements modifying the database
- Transactions in a database environment have two main purposes:
  - To provide reliable units of work that allow recovery from failures and keep a database consistent even in a cases of system failure
    - All changes in a transactions must be successfully reflected to the database or none of the changed should be conducted.
  - To provide isolation between programs accessing a database concurrently.

# Statements for Transaction Control

---

## □ COMMIT

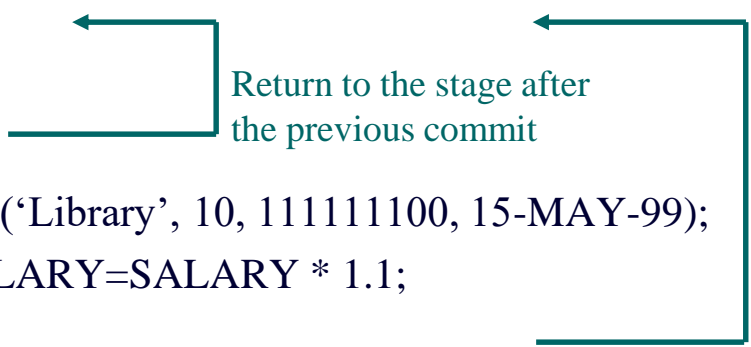
- Save any updates (insert, delete, update) permanently to database (disk) and end the transaction.

## □ ROLLBACK

- Discard any changes made within the transaction
- Cancel all updates after the previous commit. It is not possible to roll back any further than the commit command.

## □ Example

```
INSERT INTO DEPT VALUES ('Marketing', 9, 111111100, 15-MAY-99);  
COMMIT;  
DELETE FROM EMPLOYEE;  
ROLLBACK;  
INSERT INTO DEPT VALUES ('Library', 10, 111111100, 15-MAY-99);  
UPDATE EMPLOYEE SET SALARY=SALARY * 1.1;  
COMMIT;
```



Return to the stage after  
the previous commit

A transaction  
unit -  
Save any  
updates in the  
transaction  
permanently

# Transaction Control in Oracle

---

- ❑ Within Oracle, each new database connection is treated as new transaction.
- ❑ As queries are executed and commands are issued, changes are made only in memory and nothing is committed until an explicit COMMIT statement is given (with a few exceptions related to DDL commands, which include “implicit” commits, and are committed immediately).
- ❑ After the COMMIT, the next command issued essentially initiates a new transaction, and the process begins again.
- ❑ This provides greater flexibility and helps for error control as well, as no changes are committed to disk until the DBA explicitly issues the command to do so.

# Transaction Control in MS SQL

---

- ❑ One of the biggest differences between Oracle and MS SQL Server is transaction control.
- ❑ By default, MS SQL Server will execute and commit each command/task individually, and it will be difficult or impossible to roll back changes if any errors are encountered along the way.
- ❑ To properly group statements, the “BEGIN TRANSACTION” command is used to declare the beginning of a transaction, and either a COMMIT statement is used at the end.



# Outline

---

- ❑ Basic SQL Queries
- ❑ Advanced SQL Queries
- ❑ SQL Query with Multiple Relations
- ❑ Other DML
- ☞ View
- ❑ Other database objects

# Views

---

- ❑ A *view* is a “virtual table,” a relation that is defined in terms of the contents of other tables and views.
- ❑ Declare by:  
`CREATE VIEW <name> AS <query>;`
- ❑ In contrast, a relation whose value is really stored in the database is called a *base table*.
- ❑ Views can be used to present only necessary information (or a summary), while hiding details in underlying relation(s).
- ❑ VIEW in Oracle at [https://docs.oracle.com/cd/B28359\\_01/server.111/b28310/views001.htm#ADMIN11782](https://docs.oracle.com/cd/B28359_01/server.111/b28310/views001.htm#ADMIN11782)

# Creating View, Views in Queries

---

- Example: Create ResearchEmp view showing the employees of Research department :

```
CREATE VIEW ResearchEmp
```

```
AS
```

```
SELECT *
```

```
FROM Employee
```

```
WHERE dno = (SELECT dnumber
```

```
FROM department WHERE dname='Research');
```

- After creation, you can query a view as a table.

```
SELECT lname FROM ResearchEmp
```

```
WHERE sex= 'F';
```

# View Created with Joined Tables

---

- Example: Create a view which shows project information including the project manager.

```
CREATE VIEW ProjectManager  
AS
```

```
SELECT  p.pnumber, p.plocation, p.dnum AS controlDeptNo,  
        e.fname|| ' ' || e.lname AS managerName  
FROM    project p, department d, employee e  
WHERE   p.dnum=d.dnumber AND d.mgrssn=e.ssn;
```

- Query with the view.

```
SELECT *  
FROM ProjectManager  
WHERE plocation= ' Stafford' '
```

# View with Derived Columns

---

## □ View with derived columns

```
CREATE VIEW OutstandingInvoices (InvoiceNumber, InvoiceDate,  
InvoiceTotal, BalanceDue)  
AS  
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal,  
InvoiceTotal-PaymentTotal-CreditTotal  
FROM Invoices  
WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0
```

## □ View with aggregation

```
CREATE VIEW dept_salary  
AS  
SELECT dno, sum(salary) AS total_salary  
FROM employees  
GROUP BY dno;
```

# Main Advantages of Views

---

- ❑ To protect the data.
  - When you have a table containing sensitive data in certain columns and you might wish to hide those columns from certain groups of users
- ❑ To hide query complexity.
  - You can provide a view that runs a commonly requested query so that users can simply query the view.

# Insert, Update and Delete via Simple View

---

**CREATE VIEW ResearchEmp**

**AS SELECT \***

**FROM Employee**

**WHERE dno = 5;**

**INSERT INTO ResearchEmp (SSN, Fname, Lname, Dno)**

**VALUES (77777, 'Jackson', 'Kim', 5);**

**UPDATE ResearchEmp**

**SET salary = 100000**

**WHERE Fname='Jackson' AND Lname='Kim';**

**DELETE ResearchEmp**

**WHERE SSN=77777;**

# Restrictions on DML on View

---

- The SQL to update rows through a view is the same as for updating a table directly, however there are certain conditions that have to be met before updates are possible:
  - Data cannot be updated in a view that contains grouping, DISTINCT (or UNIQUE) operators or a set operator (UNION, UNION ALL, MINUS, INTERSECT).
  - Data cannot be inserted through a view if the base table has NOT NULL columns that i) are not included in the view and ii) do not have a default value
  - Virtual (derived) columns in a view (created with a DECODE or CASE statement for example) cannot be updated.
  - If a view is defined using the WITH CHECK OPTION constraint then you can only insert rows into the base table that can subsequently be selected through the view.



# View with the WITH CHECK OPTION

---

- The **WITH CHECK OPTION** clause can be given for a view to prevent any operation for which the WHERE clause condition in the view definition is not true

```
CREATE VIEW clerk
```

```
AS
```

```
SELECT employee_id, last_name, department_id, job_id
```

```
FROM employees
```

```
WHERE job_id = 'PU_CLERK' OR job_id = 'SH_CLERK'
```

```
      OR job_id = 'ST_CLERK'
```

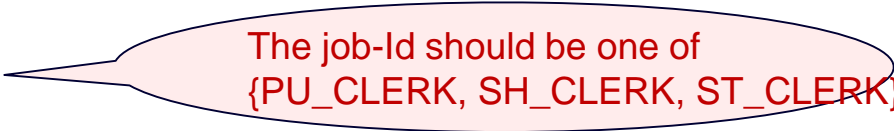
```
WITH CHECK OPTION;
```

- The update below is not allowed...

```
UPDATE clerk
```

```
SET job_id = 'PU_MAN'
```

```
WHERE employee_id = 118;
```



The job-id should be one of  
{PU\_CLERK, SH\_CLERK, ST\_CLERK}

# Updating a Join View in Oracle

---

Rule	Description
General Rule	Any <code>INSERT</code> , <code>UPDATE</code> , or <code>DELETE</code> operation on a join view can modify only one underlying base table at a time.
<code>UPDATE</code> Rule	All updatable columns of a join view must map to columns of a <b>key-preserved table</b> . See " <a href="#">Key-Preserved Tables</a> " for a discussion of key-preserved tables. If the view is defined with the <code>WITH CHECK OPTION</code> clause, then all join columns and all columns of repeated tables are not updatable.
<code>DELETE</code> Rule	Rows from a join view can be deleted as long as there is exactly one key-preserved table in the join. The key preserved table can be repeated in the <code>FROM</code> clause. If the view is defined with the <code>WITH CHECK OPTION</code> clause and the key preserved table is repeated, then the rows cannot be deleted from the view.
<code>INSERT</code> Rule	An <code>INSERT</code> statement must not explicitly or implicitly refer to the columns of a <b>non-key-preserved table</b> . If the join view is defined with the <code>WITH CHECK OPTION</code> clause, <code>INSERT</code> statements are not permitted.

\* A table is key-preserved if every key of the table can also be a key of the result of the join.

# Replacing or Altering/ Dropping Views

---

- ❑ To change the definition of an existing view

```
CREATE OR REPLACE VIEW ResearchEmp
```

```
AS
```

```
SELECT lname, fname, dno
```

```
FROM Employee
```

```
WHERE dno = 5;
```

- ❑ To drop a view

```
DROP VIEW ResearchEmp;
```

# Many Other Database Objects

---

- ❑ Index (Ch5)
- ❑ Trigger (Next class note)
- ❑ Stored Procedure, Function, Package (Next class note)
- ❑ Many other database objects such as sequence, synonym, types, database links, and so on.

# Creation of SEQUENCE object

---

- A database object from which multiple users may generate unique integers.

```
CREATE SEQUENCE employees_seq  
START WITH 1000  
INCREMENT BY 1  
NOCYCLE;
```

```
CREATE SEQUENCE orders_seq  
START WITH 1  
INCREMENT BY 1  
NOCYCLE;
```

```
DROP SEQUENCE custom_seq;
```

# Usage of SEQUENCE object

---

- ❑ `SELECT employees_seq.nextval FROM DUAL;`
- ❑ `SELECT employees_seq.currval FROM DUAL;`
- ❑ `INSERT INTO employees VALUES (employees_seq.nextval, 'John', 'Doe', 'jdoe', '555-1212', SYSDATE, 'PU_CLERK', 2500, null, null, 30);`
- ❑ `INSERT INTO orders (order_id, order_date, customer_id) VALUES ('ORD' || orders_seq.nextval, SYSDATE, 106);`
- ❑ `INSERT INTO order_items (order_id, line_item_id, product_id) VALUES ('ORD' || orders_seq.currval, 1, 2359);`
- ❑ `INSERT INTO order_items (order_id, line_item_id, product_id) VALUES ('ORD' || orders_seq.currval, 2, 3290);`

# [Appendix] Comparison of different SQL Implementations

---

- The website below shows comparisons of different SQL implements in major DBMSs such as PostgreSQL, DB2, MS SQL Server, MySQL, Oracle, and Information

<http://troels.arvin.dk/db/rdbms/>