

SQL Query Slides

Prepared By:

RIPAN BISWAS

Date : 26/03/2009



Retrieval Queries 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

Relational 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 Flice, 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	

Populated Database:

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1968-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1961-06-19

WORKS_ON	ESSN	PNQ	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	1963-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1968-01-04	SON
	123456789	Alice	F	1968-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE



Simple Queries

Query 0: Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

```
Q0:      SELECT BDATE, ADDRESS  
          FROM EMPLOYEE  
          WHERE FNAME='John' AND MINIT='B'  
                AND LNAME='Smith'
```

Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:      SELECT FNAME, LNAME, ADDRESS  
          FROM EMPLOYEE, DEPARTMENT  
          WHERE DNAME='Research' AND  
                DNUMBER=DNO
```




Some Queries Cont.

Query 4: Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

Q4: **(SELECT PNAME**
 FROM PROJECT, DEPARTMENT, EMPLOYEE
 WHERE DNUM=DNUMBER AND MGRSSN=SSN AND
 LNAME='Smith')
UNION (SELECT PNAME
 FROM PROJECT, WORKS_ON, EMPLOYEE
 WHERE PNUMBER=PNO AND ESSN=SSN AND
 LNAME='Smith')



Some Queries Cont. EXISTS

EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not

**Q5B: SELECT FNAME, LNAME
 FROM EMPLOYEE
 WHERE EXISTS (SELECT *
 FROM DEPENDENT
 WHERE SSN=ESSN AND
 FNAME=DEPENDENT_NAME)**

Some Queries Cont.

explicit (enumerated) set of values

It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query

Query 6: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

```
Q6:    SELECT DISTINCT ESSN
        FROM WORKS_ON
        WHERE PNO IN (1, 2, 3)
```

Some Queries Cont.

The **CONTAINS** operator compares two *sets of values* , and returns TRUE if one set contains all values in the other set (reminiscent of the *division* operation of algebra).

Query 7: Retrieve the name of each employee who works on *all* the projects controlled by department number 5.

```
Q7:    SELECT FNAME, LNAME
        FROM EMPLOYEE
        WHERE ( (SELECT PNO
                 FROM WORKS_ON
                 WHERE SSN=ESSN)
              CONTAINS
              (SELECT PNUMBER
               FROM PROJECT
               WHERE DNUM=5) )
```



Some Queries Cont. Null Value

SQL uses **IS** or **IS NOT** to compare NULLs because it considers each NULL value distinct from other NULL

Query 8: Retrieve the names of all employees who do not have supervisors.

Q8: **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



Some Queries Cont. JOIN

QT: **SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME**
 FROM EMPLOYEE E S
 WHERE E.SUPERSSN=S.SSN

Can be written as:

QTA: **SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME**
 FROM (EMPLOYEE E **LEFT OUTER JOIN EMPLOYEES**
 ON E.SUPERSSN=S.SSN)



Some Queries Cont. JOIN

Q9: **SELECT FNAME, LNAME, ADDRESS**
 FROM EMPLOYEE, DEPARTMENT
 WHERE DNAME='Research' AND DNUMBER=DNO

Can be written as:

Q9A: **SELECT FNAME, LNAME, ADDRESS**
 FROM (EMPLOYEE **JOIN DEPARTMENT**
 ****ON** DNUMBER=DNO)**
 WHERE DNAME='Research'

Or as:

Q9B: **SELECT FNAME, LNAME, ADDRESS**
 FROM (EMPLOYEE **NATURAL JOIN**
 DEPARTMENT AS DEPT(DNAME, DNO, MSSN, MSDATE)
 WHERE DNAME='Research'



Joined Relations Feature in SQL2

Query 2: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

```
Q2 B:  SELECT PNUMBER, DNUM,  
        LNAME, BDATE, ADDRESS  
        FROM (PROJECT JOIN  
              DEPARTMENT ON  
              DNUM=DNUMBER) JOIN  
              EMPLOYEE ON  
              MGRSSN=SSN) )  
        WHERE PLOCATION='Stafford'
```



AGGREGATE FUNCTIONS

Include **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**

Query 10: Find the maximum salary, the minimum salary, and the average salary among all employees.

**Q10: SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
 FROM EMPLOYEE**

Query 11: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

**Q11: SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
 FROM EMPLOYEE, DEPARTMENT
 WHERE DNO=DNUMBER AND
 DNAME='Research'**



Group by

SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

Query 12: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q12:  SELECT DNO, COUNT (*), AVG (SALARY)
      FROM EMPLOYEE
      GROUP BY DNO
```

Query 13: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
Q13:  SELECT PNUMBER, PNAME, COUNT (*)
      FROM PROJECT, WORKS_ON
      WHERE PNUMBER=PNO
      GROUP BY PNUMBER, PNAME
```



Group by cont. Having

The HAVING-clause is used for specifying a selection condition on groups (rather than on individual tuples)

Query 14: For each project *on which more than two employees work*, retrieve the project number, project name, and the number of employees who work on that project.

Q14: **SELECT PNUMBER, PNAME, COUNT (*)**
 FROM PROJECT, WORKS_ON
 WHERE PNUMBER=PNO
 GROUP BY PNUMBER, PNAME
 HAVING COUNT (*) > 2



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:

SELECT <attribute list>

FROM <table list>

[WHERE <condition>]

[GROUP BY <grouping attribute(s)>]

[HAVING <group condition>]

[ORDER BY <attribute list>]



Summary of SQL Queries (cont.)

- 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-

More complex Select “SQL Server”

```
SELECT select_list
[ INTO new_table ]
FROM table_source
[ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC |
DESC ] ]
```

Select Clause:

```
SELECT [ ALL | DISTINCT ]
[ TOP n [ PERCENT ] [ WITH TIES ] ]
< select_list >
< select_list > ::=
{ *
| { table_name | view_name | table_alias }. *
| { column_name | expression | IDENTITYCOL |
ROWGUIDCOL }
[ [ AS ] column_alias ]
| column_alias = expression
} [ ,...n ]
```

From Clause:

```
[ FROM { < table_source > } [ ,...n ] ]
< table_source > ::=
table_name [ [ AS ] table_alias ] [ WITH ( < table_hint > [ ,...n ]
) ]
| view_name [ [ AS ] table_alias ]
| rowset_function [ [ AS ] table_alias ]
| OPENXML
| derived_table [ AS ] table_alias [ ( column_alias [ ,...n ] ) ]
| < joined_table >
< joined_table > ::=
< table_source > < join_type > < table_source > ON <
search_condition >
| < table_source > CROSS JOIN < table_source >
| < joined_table >
< join_type > ::=
[ INNER | { { LEFT | RIGHT | FULL } [ OUTER ] } ]
[ < join_hint > ]
JOIN
Arguments
< table_source >
```

More complex Select “SQL Server”

Cont.

Where Clause:

```
[ WHERE < search_condition > | <
old_outer_join > ]

< old_outer_join > ::=
column_name { * = | = * } column_name
```

Group by clause:

```
[ GROUP BY [ ALL ] group_by_expression [
,...n ]
[ WITH { CUBE | ROLLUP } ]
]
```

Having:

```
[ HAVING < search_condition > ]
```

Order By Clause:

```
[ ORDER BY { order_by_expression [ ASC |
DESC ] } [ ,...n ] ]
```

Compute Clause:

```
[ COMPUTE
{ { AVG | COUNT | MAX | MIN | STDEV | STDEVP
| VAR | VARP | SUM }
( expression ) } [ ,...n ]
[ BY expression [ ,...n ] ]
]
```



Compute

Row aggregate function	Result
AVG	Average of the values in the numeric expression
COUNT	Number of selected rows
MAX	Highest value in the expression
MIN	Lowest value in the expression
STDEV	Statistical standard deviation for all values in the expression
STDEVP	Statistical standard deviation for the population for all values in the expression
SUM	Total of the values in the numeric expression
VAR	Statistical variance for all values in the expression
VARP	Statistical variance for the population for all values in the expression



INSERT QUERY

An SQL **INSERT** statement adds one or more records to any single table in a relational database.')

```
INSERT INTO table (column1, [column2, ... ]) VALUES (value1,  
[value2, ...])
```

The number of columns and values must be the same. If a column is not specified, the default value for the column is used. The values specified (or implied) by the INSERT statement must satisfy all the applicable constraints (such as primary keys, CHECK constraints, and NOT NULL constraints). If a syntax error occurs or if any constraints are violated, the new row is not added to the table and an error returned instead.

Example:


```
INSERT INTO phone_book (name, number) VALUES ('John Doe',  
'555-1212');
```



Multirow inserts

An SQL feature (since SQL-92) is the use of *row value constructors* to insert multiple rows at a time in a single SQL statement:

```
INSERT INTO "TABLE" ("column1", ["column2, ...  
"]) VALUES ("value1a", ["value1b, ..."]),  
("value2a", ["value2b, ..."]), ...
```



This feature is supported by DB2, SQL Server (since version 10.0), PostgreSQL (since version 8.2), MySQL, and H2.

Example (assuming that 'name' and 'number' are the only columns in the 'phone_book' table):

```
INSERT INTO phone_book VALUES ('John Doe',  
'555-1212'), ('Peter Doe', '555-2323');
```

which may be seen as a shorthand for the two statements

```
INSERT INTO phone_book VALUES ('John Doe',  
'555-1212'); INSERT INTO phone_book  
VALUES ('Peter Doe', '555-2323');
```



Copying rows from other tables

An INSERT statement can also be used to retrieve data from other, modify it if necessary and insert it directly into the table. All this is done in a single SQL statement that does not involve any intermediary processing in the client application. A subselect is used instead of the VALUES clause. The subselect can contain joins, function calls, and it can even query the same table into which the data is inserted. Logically, the select is evaluated before the actual insert operation is started. An example is given below.

```
INSERT INTO phone_book2 SELECT * FROM  
phone_book WHERE name IN ('John Doe', 'Peter Doe')
```



Update (SQL)

A SQL **UPDATE** statement that changes the data of one or more records in a table. Either all the rows can be updated, or a subset may be chosen using a condition.

The UPDATE statement has the following form[1]:

```
UPDATE table_name SET column_name = value [, column_name = value ...] [WHERE condition]
```

For the UPDATE to be successful, the user must have data manipulation privileges (UPDATE privilege) on the table or column, the updated value must not conflict with all the applicable constraints (such as primary keys, unique indexes, CHECK constraints, and NOT NULL constraints).



Set the value of column *C1* in table *T* to 1, only in those rows where the value of column *C2* is "a".

UPDATE T SET C1 = 1 WHERE C2 = 'a'

Increase value of column *C1* by 1 if the value in column *C2* is "a".

UPDATE T SET C1 = C1 + 1 WHERE C2 = 'a'

Prepend the value in column *C1* with the string "text" if the value in column *C2* is "a".

UPDATE T SET C1 = 'text' || C1 WHERE C2 = 'a'

Set the value of column *C1* in table *T1* to 2, only if the value of column *C2* is found in the sub list of values in column *C3* in table *T2* having the column *C4* equal to 0.

UPDATE T1 SET C1 = 2 WHERE C2 IN (SELECT C3 FROM T2 WHERE C4 = 0)




UPDATE T SET C1 = 1, C2 = 2

Complex conditions are also possible:

UPDATE T SET A = 1 WHERE C1 = 1 AND C2 = 2

The SQL:2003 standard does not support updates of a joined table. Therefore, the following method needs to be used. Note that the subselect in the SET clause must be a scalar subselect, i.e., it can return at most a single row.

**UPDATE T1 SET C1 = (SELECT T2.C2 FROM T2
WHERE T1.ID = T2.ID) WHERE EXISTS (SELECT 1
FROM T2 WHERE T1.ID = T2.ID)**



Delete (SQL)

An SQL **DELETE** statement removes one or more records from a table. A subset may be defined for deletion using a condition, otherwise all records are removed.

The DELETE statement has this syntax:

```
DELETE FROM table_name [WHERE condition]
```

Any rows that match the WHERE condition will be removed from the table. If the WHERE clause is omitted, all rows in the table are removed. The DELETE statement should thus be used with caution!

The DELETE statement does not return any rows; that is, it will not generate a result set.

Executing a DELETE statement may cause triggers to run that may cause deletes in other tables. For example, if two tables are linked by a foreign key and rows in one table were deleted, then it is common that rows in the second table would also have to be deleted to maintain referential integrity. For a more general discussion of how cascading deletes are handled, see propagation constraint.



UNION operator

In SQL the **UNION** clause combines the results of two SQL queries into a single table of all matching rows. The two queries must have the same number of columns and compatible data types to unite. Any duplicate records are automatically removed unless UNION ALL is used.



sales2005

person	amount
Joe	1000
Alex	2000
Bob	5000

sales2006

person	amount
Joe	2000
Alex	2000
Zach	35000

Executing this statement:

```
SELECT * FROM sales2005 UNION SELECT * FROM sales2006;
```



yields this result set, though the order of the rows can vary because no ORDER BY clause was supplied:

person	amount
Joe	1000
Alex	2000
Bob	5000
Joe	2000
Zach	35000



Delete rows from table *pies* where the column *flavour* equals *Lemon Meringue*:

```
DELETE FROM pies WHERE flavour='Lemon Meringue';
```

Delete rows in *trees*, if the value of *height* is smaller than 80.

```
DELETE FROM trees WHERE height < 80;
```

Delete all rows from *mytable*:

```
DELETE FROM mytable;
```

Delete rows from *mytable* using a subquery in the where condition:

```
DELETE FROM mytable WHERE id IN (SELECT id FROM mytable2)
```

Delete rows from *mytable* using a list of values:

```
DELETE FROM mytable WHERE id IN (value1, value2, value3, value4, value5)
```



THANK YOU