**IS 441 SQL Demos November 2, 9, 14**

*Dr. Yue “Jeff” Zhang*

I. INSERT

Syntax:

1. Inserting a COMPLETE row:

INSERT INTO *table\_name* VALUES (*field\_value\_list*);

2. Inserting a PARTIAL row:

INSERT INTO (*partial\_field\_list*) VALUES (*field\_value\_list*);

NOTE: In both of the situations, the value list SEQUECE must be corresponding to the field list; in case 1 they must be corresponding to the original list of fields in the table; in case 2 they must be corresponding to the “partial\_field\_list” as given.

Example:

1. Full list:

INSERT INTO Restaurants

VALUES ("R0021"," 12345 Nordhoff", "NR", "CA", "91324","818-993-1233",500000,

2000,1/1/2001,"Table service", Yes, Yes, Yes, "F001");

VALUES (values of ALL fields, In the order of the original coluimnS)

2. Partial list: If it’s a partial row, say, I only have RID, Annual sales, franchisseID:

INSERT INTO Restaurants (RestaurtID, AnnualSales, FreanchiseeID)

VALUES ("R0022”,51000,”F002”)

II. UPDATE

Syntax:

UPDATE *table\_name* SET *field* = *value* WHERE *condition\_to\_apply\_update*

Example:

UPDATE Restaurants

SET Zipcode = "91326"

WHERE city = "Northridge";

In our demo, one of the records (rows) had the city being “NR” rather than “Northridge;” that one row was NOT updated since it didn’t match the criterion “Northridge” for its value for the column “City” (since it’s “NR” rather than “Northridge”).

III. JOIN

**Not comma**

Syntax 1 (Recommended):

SELECT … FROM Table1, Table2, Table3 WHERE Table1.PK=Table2.FK AND Table2.PK=Table3.FK

Advantage: No matter how many tables are to be joined, one simply needs to include the tables needed to be joined, and add the join condition between the newly brought-in table with the previous table.

Syntax 2 (NOT recommended):

SELECT … FROM

(Table1 JOIN Table2 ON Table1.PK=Table2.FK)

JOIN

Table3 ON Table2.PK=Table3.FK;

Problem: Parentheses are needed, and the more table brought in, the more parentheses (AND RIGHT parentheses, at RIGHT places!!) must be used, which could cause confusion and could be error-prone).

General Notes:

1. To join N tables, N-1 JOIN conditions are needed;
2. In the case of three tables or more, even if you only need, say, the two tables at the two far ends, you still need to include all tables in the JOIN part, because those two at the far ends would not join, without the “bridging” of those tables in between.

Examples:

Syntax 2 (NOT recommended):

SELECT RestaurantID, address, LastName, FirstName, Franchisees.FranchiseeID

FROM Franchisees JOIN Restaurants ON Franchisees.FranchiseeID = Restaurants.FranchiseeID;

* \*\*\* NOT recommended: when there’re three or more tables, parentheses are required

Syntax 1 (Recommended):

SELECT RestaurantID, address, LastName, FirstName, Franchisees.FranchiseeID

FROM Restaurants, Franchisees

WHERE Franchisees.FranchiseeID = Restaurants.FranchiseeID;

General notes:

1. To join N tables, you need N-1 join conditions;
2. In the situation where there are three or more tables, if the tables from which we need to display the fields are the two at the “extreme far ends”, we still need to include the middle table(s) in the JOIN-clause – otherwise the two at the far end would not be able to “connect”: we need the/those table(s) in between to function as “bridge(s)”.

IV. OUTER JOIN

List all franchisees and their restaurants, including those franchisees who do not own a restaurant.

SELECT F.FranchiseeID, LASTnAME, rESTAURANTid, aDDRESS

FROM Franchisees F LEFT OUTER JOIN Restaurants R

on F.FranchiseeID = R.FranchiseeID;

Key things to note:

ON:

1)      Works in OUTER JOIN while WHERE doesn’t;

2)      In INNER JOIN, when there’re M tables, ON requires clear and complex parentheses, while WHERE doesn’t

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

In multi-table join [of course NOT outer join! – outer join don’t work for 3 tables!],

be aware that there can be two TYPES of conditions:

1)      Join conditions (can be multiple of them!)

2)      Logical conditions (can be multiple of them)

V. Self-Join

List salespersons (ID and name) and their supervisors (ID and name).

* Logic:

Treat the physical table Salesperson as two logical tables: “employee” E, and “manager” M.

* Join condition:

Employee’s managerID, = their manager’s employee ID.

SELECT E.salespersonID, E.salespersonname, M.salespersonID, M.salespersonname AS Manager

FROM Salesperson\_T E, Salesperson\_T  M

WHERE E.supervisorID = M.salespersonID;

**November 14 class (review and new) begins here**

VI. SUBQuery – Non-Correlated

6.1 Using subquery to get around certain prohibitions

List restaurants and their sales, side-by-side compared with the average sales of all restaurants

|  |  |
| --- | --- |
| SELECT RestaurantID, annualsales, (SELECT AVG(annualsales) FROM Restaurants)FROM Restaurants; | Use of subquery here helps to get around the rule “no row values and set values together in the SELECT-clause.” |

List restaurants and their sales, display only those whose sales are higher than the average sales of all restaurants

|  |  |
| --- | --- |
| SELECT RestaurantID, annualsales,  |  |
| (SELECT AVG(annualsales) FROM Restaurants) AS AVerage | By pass/get around “no mixture of set and row values” |
| FROM Restaurants |  |
| WHERE annualsales > (SELECT AVG(annualsales) FROM Restaurants) ; | By pass “no aggregate function in WHERE-clause”(-- commonly seen mistakes in Exam 2) |

6.2 Subquery with GROUP BY

List the average sales by city, for those cities whose averages are higher than the average of the whole table (Blue: GROUP BY; Red: Subquery)

|  |  |
| --- | --- |
| SELECT AVG(AnnualSales) As CityAVG, City, (SELECT AVG(AnnualSales) FROM Restaurants) AS OverallAvgFROM Restaurants GROUP BY CityHAVING AVG(AnnualSales)>(SELECT AVG(AnnualSales) FROM Restaurants); |  |

VII. SUBQuery –Correlated

Need subquery to deliver logics otherwise impossible

* passing parameters INTO subquery

List the restaurants whose sales are higher than THE AVERAGE of restaurants IN THEIR CITY.

Analysis:

|  |  |
| --- | --- |
| SELECT RestaurantID, AnnualSales, City, (SELECT AVG(Annualsales) FROM Restaurants R\_InsideWHERE R\_Inside.City = R\_Outside.City GROUP BY City) AS CityAVGFROM Restaurants R\_OutsideWHERE AnnualSales >(SELECT AVG(Annualsales) FROM Restaurants R\_InsideWHERE R\_Inside.City = R\_Outside.City GROUP BY City) | Display restaurants and their corresponding sales and cities, for those ……whose sales are more than……the city average of ITS OWN city, accomplished by the passing of THE cityR\_Inside.City = R\_Outside.City |

|  |  |  |
| --- | --- | --- |
| Final example:**Here**SELECT RestaurantID, City,(SELECT COUNT(RestaurantID) FROM Restaurants R\_InsideWHERE R\_Inside.city= R\_Outside.cityGROUP BY R\_Inside.city) AS CountByCity(SELECT COUNT(RestaurantID) FROM Restaurants R\_InsideGROUP BY R\_Inside.cityHAVING R\_Inside.city= R\_Outside.city) AS CountByCityFROM Restaurants R\_OutsideWHERE City IN(SELECT City FROM Restaurants GROUP BY CityHAVING COUNT(RestaurantID)>2); | Ver 1: WHERE … GROUP BY – we filter out other restaurants that are not in THE city that was passed in, and only the rest in THE city will be grouped: filter 1st, group 2ndVer 2: GROUP BY… HAVING – we group ALL rest according to THEIR citieS, and then filter out other GROUPs that are not in THE city that was passed in | **New task for at-home exercise:**What if we want to display the count of the city of the restaurant side by side with the city?Hint: “THAT city” needs the passing of parameter into subquery for the COUNT |

Summary:

|  |  |  |
| --- | --- | --- |
| In clause  | Use of subqueries  | Correlated subquery |
| SELECT | Yes: Function as a filed – providing value(s) for display |  |
| FROM | Yes: Function as a “table” – providing data to query from | Pass parameter |
| WHERE | Yes: Provide value(s) for the comparison | * receive prmtr
 |
| GROUP BY | No |  |
| HAVING | Yes: Similar to WHERE-clause, but for groups only | * receive prmtr
 |
| ORDER BY | No |  |