SQL Course

An Introduction to Joins

- Introduce the concept of a multi-table query in the form of a join.
- Explain why joins are necessary and how the DBMS handles them.
- Note some important points about joins.

- So far we have looked at single table queries (i.e. queries involving only one table). This is not sufficient as we shall on many occasions need to retrieve information from several tables at the same time.

- We call the combining of information from different tables a **JOIN**.

- Values can be displayed from more than one table by "joining" the rows using columns of the same type and size.

- Tables can be joined by adding a condition to the **WHERE** clause.

- Joins complete the triad of operations that a relational query language must provide: selection, projection, and join. The join operation lets you retrieve and manipulate data from more than one table in a single **SELECT** statement. Joining two tables is a little like making a seam to join two pieces of material. Once you have made the seam, you can work with the whole cloth. You specify joins in the **WHERE** clause of a **SELECT** statement. Like projections, which are specified in the select list of the **SELECT** statement, joins are expressed implicitly rather than explicitly.

- You specify each join on two tables at a time, using a column from each table as a connecting column or join column. A connecting column should have values that match or compare easily, representing the same or similar data in each of the tables participating in the join.

- Connecting columns almost always have the same data type. The values in connecting columns are join compatible: their values come from the same general class of data.

- A skeleton version of join syntax is:

  SELECT *select_list* **FROM** *table_1, table_2* [,*table_3* ...] **WHERE** *[table_1.]column* **JOIN_OPERATOR** *[table_2.]column*;

- The **FROM** clause table list must contain at least two tables, and the columns specified in the **WHERE** clause must be join compatible.

- When the join columns have identical names you must qualify the columns with their table names in the select list and in the **WHERE** clause.

- You can use any of the relational operators to express the relationships between the join columns as the join operator but equality is the most common. However it may be one of the following :

  < > < >= <> or **!=**

- You don't have to name the join column twice in the select list, or even include it in the results at all, in order for the join to succeed. However you may need to qualify the name of a join column with its table name in the select list or in join specification of the **WHERE** clause.
- As many tables as desired may be joined together.
- The matching criteria for the tables is called the **join predicate** or **join criteria**.
- More than one pair of columns may be used to specify the join condition between any two tables.
- Columns from any tables may be named in the **SELECT** clause.
- Columns which have the same name in the multiple tables named in the **FROM** clause must be uniquely identified by specifying **tablename.columnname**.
- If the columns are uniquely named across the tables, it is not necessary to specify **tablename.columnname**. However it improves readability of the **SELECT** statement.
- When n tables are joined, it is necessary to have at least n-1 table join conditions to avoid the Cartesian product. (Four table joins requires specification of a join criteria for three pairs of tables).
- When an = is used as the join operator the join is called an **Equi-Join** or **Equal-Join** and is used to retrieve rows which have a matching values in each column named in the join clause.
- The joining of tables can be based on any criteria you choose
  (not only equality comparison).
- The high level join construct hides all asymmetry of the implementation. A join is commutative. This means A join B = B join A. Therefore, the order in which the names appear in the **FROM** clause is irrelevant.
- The notion of a join generalises to more than two tables. To perform a join between A, B and C, two tables are first joined and then the result is joined with the third table. The join operation is also associative : (A join B) join C = A join (B join C).

Joins are necessary because:

- In a database that has been designed according to the normalisation rules, one table may not give you all the information you need about a particular entity. For comprehensive data analysis, you must assemble data from several of the tables you created when you normalised your database.
- The relational model - having led you to partition your data into several single subject tables following the rules of normalisation and good clean database design - relies on the join operation to enable you to perform ad hoc queries and produce comprehensible reports.
- Joins are possible in a relational database management system because the relational model's data independence permits you to bring data from separate tables into new and

unanticipated relationships. Relationships among data become explicit when you manipulate the data, not when you create the database.

- Joins are necessary because during the process of analysis that distributes data over the relational database landscape, you cleanly separate information on separate entities. You can get a comprehensive view of your data only if you can reconnect the tables. A corollary of the join is that it gives you unlimited flexibility in adding new kinds of data to your database.

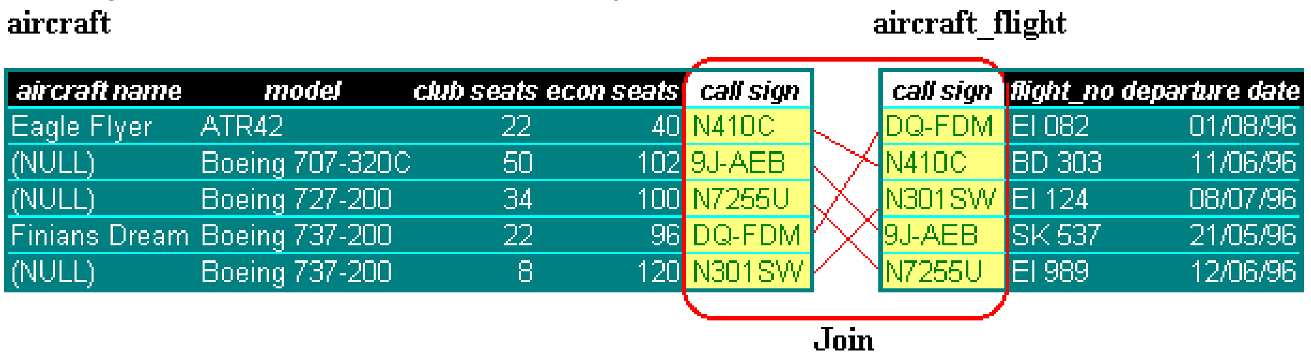
- In relational theory, a join is the projection and restriction of the product. The product or cartesian product, is the set of all possible combinations of the rows from the two tables.

- The system first examines all possible combinations of the rows from two tables, then eliminates all the rows that do not meet the conditions of the projection (columns) and restriction (rows). The actual procedure that the system follows is more sophisticated.

- Conceptually speaking, the first step in processing a join is to form the cartesian product of the tables - all the possible combinations of the rows from each of the tables. Once the system obtains the cartesian product, it uses the columns in the select list for the projection, and the conditions in the **WHERE** clause for the selection, to eliminate the rows that do not satisfy the join.

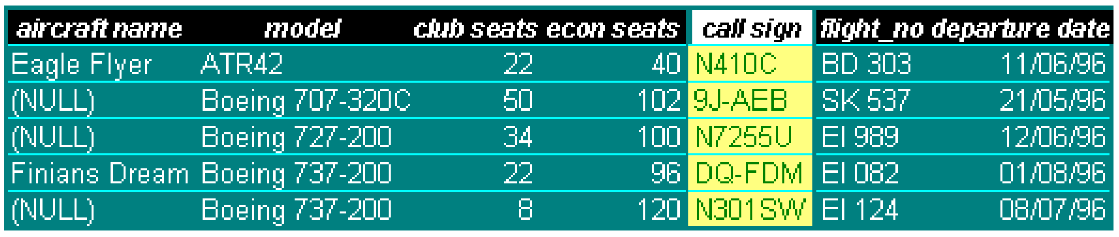
- The cartesian product is the matrix of all possible combinations that could satisfy the join condition. If there is only one row in each table, then there is only one possible combination. The number of rows in the cartesian product equals the number of rows in the first table multiplied by the number of rows in the second table.

The diagram below shos how 2 tables are joined -



aircraft name	model	club seats	econ seats	call sign
Eagle Flyer	ATR42	22	40	N410C
(NULL)	Boeing 707-320C	50	102	9J-AEB
(NULL)	Boeing 727-200	34	100	N7255U
Finians Dream	Boeing 737-200	22	96	DQ-FDM
(NULL)	Boeing 737-200	8	120	N301SW

call sign	flight_no	departure date
DQ-FDM	EI 082	01/08/96
N410C	BD 303	11/06/96
N301SW	EI 124	08/07/96
9J-AEB	SK 537	21/05/96
N7255U	EI 989	12/06/96



aircraft name	model	club seats	econ seats	call sign	flight_no	departure date
Eagle Flyer	ATR42	22	40	N410C	BD 303	11/06/96
(NULL)	Boeing 707-320C	50	102	9J-AEB	SK 537	21/05/96
(NULL)	Boeing 727-200	34	100	N7255U	EI 989	12/06/96
Finians Dream	Boeing 737-200	22	96	DQ-FDM	EI 082	01/08/96
(NULL)	Boeing 737-200	8	120	N301SW	EI 124	08/07/96

Table returned

The bottom table is what results after the 2 tables are joined.