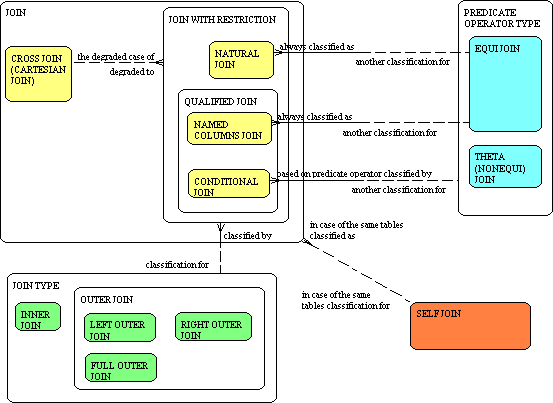
**SQL join types**

**1 Introduction**

Joins are one of the basic constructions of SQL and Databases as such - they combine records from two or more database tables into one row source, one set of rows with the same columns. And these columns can originate from either of the joined tables as well as be formed using expressions and built-in or user-defined functions. Depending on join type and join restrictions returned row count can be from 0 till all possible combinations of involved tables. So if one has two tables containing 10 000 rows each then the maximum number of resultant rows can be 10 000 \* 10 000 = 100 000 000 rows. Databases are built to make joins as efficient as possible. It means almost always joining data in database is more efficient than doing that somewhere else. It also means one has to know the power and possibilities of joins to fully exploit their strength. And it doesn't matter whether one is using Oracle, SQL Server, MySQL, IBM DB2, PostgreSQL or whatever else DBMS.  
**NB 1!** It is worth to mention that throughout all this article it is supposed two tables are joined together. It is used both for theoretical parts of it as well as examples. However one has to remember that **every two row sources can be joined together**, and both or one of them can be table, view, subquery, materialized view or other construction returning rows.  
**NB 2!** All examples are created for Oracle database and written according to Oracle syntax. However it doesn't matter what database management system is used, many of them with (probably) very little modifications or even exactly the same can be used for every other DBMS supporting joins. Exactly why they work or why not are described for [Oracle](http://www.gplivna.eu/papers/sql_join_types.htm#p12.1), [SQL Server](http://www.gplivna.eu/papers/sql_join_types.htm#p12.2) and [MySQL](http://www.gplivna.eu/papers/sql_join_types.htm#p12.3). If you need to use them for other DBMSes then you should check these examples yourself although I would be very pleased if you'd send me information what examples are not working on what DBMSes. I will include this info here along with your name.

**2 Classification schemes**

There are different classification schemes and different criteria according to what joins are classified. As a result there is a bit mess in the process of understanding them. I haven't found a nice scheme or even textual description how each one of various classification schemes and join types relates to other. In the following Meta model there is one possible variant of that, however theoreticians probably would break lances around other possible classification schemes.  


**2.1 Depending on condition existence**

Depending on whether we add any join condition or not there are following join types (yellow in model):

* CROSS JOIN (synonyms also CARTESIAN JOIN, CARTESIAN PRODUCT) - there isn't any join condition or it is always true. All other join types degrade to CROSS JOINS as soon as join condition (-s) is (are) always true.
* JOIN WITH RESTRICTION - there is applied join condition to joined tables. One can write join with restriction using different syntactic notations. As already said above every join with restriction may degrade to cross join.
* NATURAL JOIN - syntactic notation joining source tables on all columns having the same name. This can be quite dangerous as explained below in the chapter for Natural join. Natural joins always are Equi joins.
* QUALIFIED JOIN - user has possibility to define which columns are used in join condition.
* NAMED COLUMNS JOIN - syntactic notation joining source tables on user defined columns having the same name. This is less dangerous than Natural join and just short form of writing Equi joins on some common columns joined together. Named columns joins always are Equi joins.
* CONDITIONAL JOIN - fully controllable syntax by user. This is the most widespread and most useful syntactic convention. Depending on used predicates in join condition it may be Equi join as well as Non-equi join.

**2.2 Depending on row selection**

Depending on whether only rows satisfying join condition are selected or all rows are selected in one or both involved tables, joins are divided into (green in model):

* INNER JOIN - only rows satisfying selection criteria from both joined tables are selected.
* LEFT OUTER JOIN - rows satisfying selection criteria from both joined tables are selected as well as all remaining rows from left joined table are being kept along with Nulls instead of actual right joined table values.
* RIGHT OUTER JOIN - rows satisfying selection criteria from both joined tables are selected as well as all remaining rows from right joined table are being kept along with Nulls instead of actual left joined table values.
* FULL OUTER JOIN - rows satisfying selection criteria from both joined tables are selected as well as all remaining rows both from left joined table and right joined table are being kept along with Nulls instead of values from other table.

**2.3 Depending on comparison operator**

Depending on used comparison operator in join condition there are following join types (blue in model):

* PREDICATE OPERATOR TYPE - based on predicate operator type (i.e. equality and everything other) joins are divided into two parts Equi joins and Nonequi joins.
* EQUI JOIN - join condition uses only equality predicate "=". It can be both explicit for Conditional join and implicit for Natural join and Named columns join.
* THETA (NONEQUI) JOIN - everything other than equality predicate "=", for example ">=", between. This can be used only by Conditional joins.

**2.4 Depending on used tables**

There is one special case for classification based on what tables are involved in join (orange in model).

* SELF JOIN - table is joined to itself.

**2.5 Classification schemes summary**

There are some interesting facts we can observe from the diagram above:

* Every join either is Cartesian join or can degrade to Cartesian join in case the join criteria always are true. Each and every join type, be it inner, outer, self, natural, equi, non-equi, named column or join with explicitly declared join condition can degrade to Cartesian join.
* Classification scheme Depending on row existence (i.e. inner and outer joins) are absolutely independent of other classification schemes, e.g. natural joins can be both inner or outer, as well as self join can be inner or outer.
* Every other join type can be self join if both joined tables are the same.
* Equi join - although most used one - is not the only one. It is the only one if natural join or named columns join syntax is used, but for explicitly specified join condition other comparison operators besides equality ("=") can be used.

**3 Used data model for examples**

Throughout this entire article we will use following tables (table names according to my favourite [naming conventions](http://www.gplivna.eu/papers/naming_conventions.htm)) and table data:

CREATE TABLE addresses (

adr\_id INTEGER NOT NULL PRIMARY KEY,

adr\_city VARCHAR(15),

adr\_country VARCHAR(15) NOT NULL);

CREATE TABLE persons (

prs\_id INTEGER NOT NULL PRIMARY KEY,

prs\_father\_id INTEGER,

prs\_mother\_id INTEGER,

prs\_adr\_id INTEGER,

prs\_first\_name VARCHAR(15),

prs\_surname VARCHAR(15),

CONSTRAINT prs\_prs\_father\_fk FOREIGN KEY (prs\_father\_id)

REFERENCES persons(prs\_id),

CONSTRAINT prs\_prs\_mother\_fk FOREIGN KEY (prs\_mother\_id)

REFERENCES persons(prs\_id),

CONSTRAINT prs\_adr\_fk FOREIGN KEY (prs\_adr\_id)

REFERENCES addresses(adr\_id));

INSERT INTO addresses VALUES (1, 'RIGA', 'LATVIA');

INSERT INTO addresses VALUES (2, 'BERLIN', 'GERMANY');

INSERT INTO addresses VALUES (3, 'NEW YORK', 'USA');

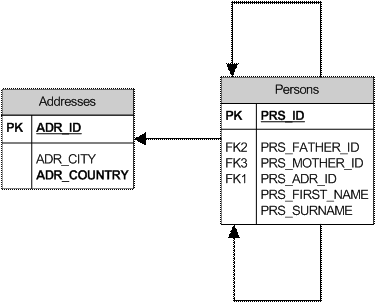
INSERT INTO persons VALUES (1, NULL, NULL, NULL, 'JANIS', 'BERZINS');

INSERT INTO persons VALUES (2, 1, NULL, 2, 'PETER', 'BERZINS');

INSERT INTO persons VALUES (3, NULL, NULL, 2, 'ANN', 'SMYTH');

INSERT INTO persons VALUES (4, 2, 3, 2, 'CHARLES', 'BERZINS');

COMMIT;

Although data model is very simple, it is very important for you to understand the data for examples, because all the examples are as important as theory. So the data model is as follows:  
  
And data looks as follows:

SQL> SELECT \* FROM persons;

PRS\_ID PRS\_FATHER\_ID PRS\_MOTHER\_ID PRS\_ADR\_ID PRS\_FIRST\_NAME PRS\_SURNAME

------ ------------- ------------- ---------- -------------- -----------

1 JANIS BERZINS

2 1 2 PETER BERZINS

3 2 ANN SMYTH

4 2 3 2 CHARLES BERZINS

SQL> SELECT \* FROM addresses;

ADR\_ID ADR\_CITY ADR\_COUNTRY

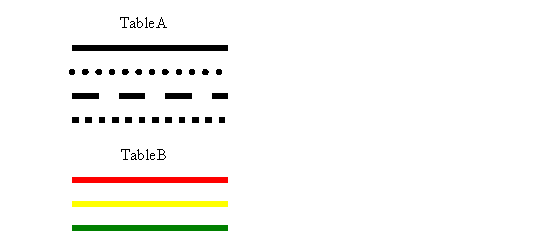
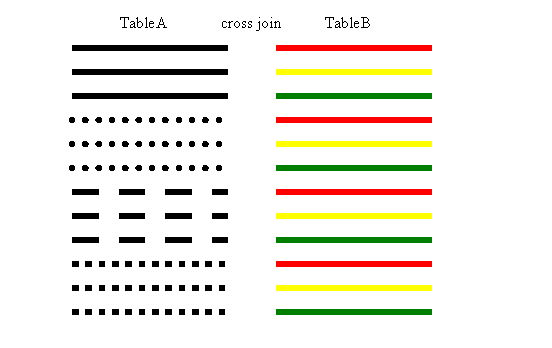
---------- --------------- -----------

1 RIGA LATVIA

2 BERLIN GERMANY

3 NEW YORK USA

**4 Cross joins**

Cross join consists of all possible combinations of two table's rows. Imagine we have two tables TableA and TableB with 4 and 3 rows respectively as follows:  
  
Cross join of both tables is as follows:  
  
It is obvious that cross join of rather small tables may lead to quite disastrous resultant record count, for example, two tables of just 100 thousand rows each results in 100 000 \* 100 000 = 10 000 000 000 (10 billion rows).

**4.1 Syntax**

There are two syntax types how to normally get cross join. One explicitly declares cross join:

**SELECT <column list>**

**FROM <left joined table>**

**CROSS JOIN <right joined table>**

Second syntax type is functionally the same; just both tables are written in FROM clause:

**SELECT <column list>**

**FROM <left joined table>, <right joined table>**

As one can see, second query hasn't WHERE clause, it means as soon as one omits it, and uses more than one table in FROM clause, it results in cross join and possibly very many rows.

**4.2 Facts to remember**

Usually cross joins are used quite rarely; some of the scenarios could be as follows:

* Possibility to generate high amount of rows. As we can see from relatively small tables there is possibility to get quite monstrous numbers.
* Find out all possible row combinations of some tables. Mostly this is useful for reports where one needs to generate all combinations for example all nationalities x genders for persons.
* To join a table with just one row. Most often used to get some configuration parameters.

There are other facts to remember about cross joins:

* **Every other join type can degrade to cross join** in case the join condition is always true for all records. This is true for natural joins, inner joins, outer joins etc. See [Example 3](http://www.gplivna.eu/papers/sql_join_types.htm#e3), [Example 4](http://www.gplivna.eu/papers/sql_join_types.htm#e4) and [Example 5](http://www.gplivna.eu/papers/sql_join_types.htm#e5).
* Making cross join as Self join one must use table aliases to let know database from which table instance get selected columns. See [Example 6](http://www.gplivna.eu/papers/sql_join_types.htm#e6) and [Example 7](http://www.gplivna.eu/papers/sql_join_types.htm#e7).

**4.3 Examples**

**Example 1 Selects first name, surname and city from Cartesian join of persons and addresses using CROSS JOIN syntax. The result is identical as in** [**Example 2**](http://www.gplivna.eu/papers/sql_join_types.htm#e2)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

CROSS JOIN addresses;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

JANIS BERZINS RIGA

PETER BERZINS RIGA

ANN SMYTH RIGA

CHARLES BERZINS RIGA

JANIS BERZINS BERLIN

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

JANIS BERZINS NEW YORK

PETER BERZINS NEW YORK

ANN SMYTH NEW YORK

CHARLES BERZINS NEW YORK

12 rows selected.

**Example 2 Selects first name, surname and city from Cartesian join of persons and addresses using both table names in FROM clause. The result is identical as in** [**Example 1**](http://www.gplivna.eu/papers/sql_join_types.htm#e1)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons, addresses;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

JANIS BERZINS RIGA

PETER BERZINS RIGA

ANN SMYTH RIGA

CHARLES BERZINS RIGA

JANIS BERZINS BERLIN

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

JANIS BERZINS NEW YORK

PETER BERZINS NEW YORK

ANN SMYTH NEW YORK

CHARLES BERZINS NEW YORK

12 rows selected.

**Example 3 Selects first name, surname and city from persons and addresses using NATURAL INNER JOIN syntax. Because both tables haven't column with the same name, result is identical as in** [**Example 1**](http://www.gplivna.eu/papers/sql_join_types.htm#e1) **and** [**Example 2**](http://www.gplivna.eu/papers/sql_join_types.htm#e2)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

NATURAL INNER JOIN addresses;

**Example 4 Selects first name, surname and city from persons and addresses using NATURAL FULL OUTER JOIN syntax. Because both tables haven't column with the same name, result is identical as in** [**Example 1**](http://www.gplivna.eu/papers/sql_join_types.htm#e1)**,** [**Example 2**](http://www.gplivna.eu/papers/sql_join_types.htm#e2) **and** [**Example 3**](http://www.gplivna.eu/papers/sql_join_types.htm#e3)**, just row order is different.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

NATURAL FULL OUTER JOIN addresses;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

JANIS BERZINS RIGA

JANIS BERZINS BERLIN

JANIS BERZINS NEW YORK

PETER BERZINS RIGA

PETER BERZINS BERLIN

PETER BERZINS NEW YORK

ANN SMYTH RIGA

ANN SMYTH BERLIN

ANN SMYTH NEW YORK

CHARLES BERZINS RIGA

CHARLES BERZINS BERLIN

CHARLES BERZINS NEW YORK

**Example 5 Selects first name, surname and city from persons and addresses using simply INNER JOIN. Because join condition 1=1 is always true result is Cartesian join.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

INNNER JOIN addresses ON (1=1);

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

JANIS BERZINS RIGA

PETER BERZINS RIGA

ANN SMYTH RIGA

CHARLES BERZINS RIGA

JANIS BERZINS BERLIN

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

JANIS BERZINS NEW YORK

PETER BERZINS NEW YORK

ANN SMYTH NEW YORK

CHARLES BERZINS NEW YORK

12 rows selected.

**Example 6 Selects all columns from persons using SELF JOIN to the same table. Because there are the same column name aliases must be used otherwise error is raised as in this example.**

SELECT \*

FROM persons

CROSS JOIN persons;

SELECT \*

\*

ERROR at line 1:

ORA-00918: column ambiguously defined

**Example 7 Selects first name and surname from persons using SELF JOIN to the same table. Because there are the same column name aliases must be used. There are 4 rows in persons table so we got 4\*4 = 16 rows from Cartesian product.**

SELECT p1.prs\_first\_name, p2.prs\_surname

FROM persons p1

CROSS JOIN persons p2;

PRS\_FIRST\_NAME PRS\_SURNAME

--------------- ---------------

JANIS BERZINS

PETER BERZINS

ANN BERZINS

CHARLES BERZINS

JANIS BERZINS

PETER BERZINS

ANN BERZINS

CHARLES BERZINS

JANIS SMYTH

PETER SMYTH

ANN SMYTH

CHARLES SMYTH

JANIS BERZINS

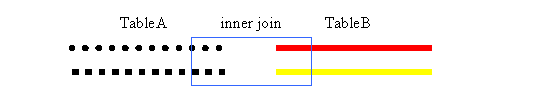
PETER BERZINS

ANN BERZINS

CHARLES BERZINS

16 rows selected.

**5 Inner joins**

Inner join is the most widely used join type. It returns those and only those rows from both joined tables satisfying join condition. Usually join condition is equality of two columns one from table A and other from table B. Sometimes it is expanded so that more than one column of table A must be equal to table's B column. Although these are most common join conditions, and in fact equality is not the only one.  
Assuming we have tables A and B as defined in [chapter about Cross joins](http://www.gplivna.eu/papers/sql_join_types.htm#p4) inner join with equality can be visually interpreted as follows:   
Blue rectangle shows common columns from both tables satisfying join criteria. As one can see generally **not all** rows are returned.

**5.1 Syntax**

Inner join syntax is as follows:

**SELECT <column list>**

**FROM <left joined table>**

**[INNER] JOIN <right joined table>**

**ON <join condition>**

Keyword INNER sometimes is avoided, but anyway it remains inner join. After the keyword ON join condition is written, generally it can contain many predicates connected with Boolean AND, OR, NOT.

**5.2 Facts to remember**

All following facts are relevant to almost every join type, however I'd like to stress them here because inner join is the most common join type and as a result more impact will be done taking them into account.

* Databases are designed to do joins efficiently. Not client, not middle tier, but databases. Creating data model usually normalization is being done mostly to avoid data duplication. To show data in person-readable form, joining is one of the necessary prerequisites. Inner join is one of the most common join types and it should be done in database. Database means efficiently in the place where data resides and not in client/middle tier cycling through (possibly) many cycles.
* It should be noted that other clauses for SQL statements e.g. GROUP BY, HAVING, ORDER BY are of course usable for SQL statements with joins.
* One must be cautious using join conditions with columns without NOT NULL constraint. Comparing NULL values with different explicit values or even Nulls is probably a bit counter-intuitive on the first sight.
* The position of additional conditions is irrelevant for INNER joins. These can be written both as join condition and in WHERE clause, result is the same. See [Example 11](http://www.gplivna.eu/papers/sql_join_types.htm#e11) and [Example 12](http://www.gplivna.eu/papers/sql_join_types.htm#e12).
* For Inner join it is not relevant, which table is first one and which second one. See [Example 10](http://www.gplivna.eu/papers/sql_join_types.htm#e10).

**5.3 Examples**

**Example 8 Selects first name, surname and city for every person.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

INNER JOIN addresses ON prs\_adr\_id = adr\_id;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

**Example 9 Selects first name, surname and city for every person. Keyword Inner is not present, but functionally result is the same.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

JOIN addresses ON prs\_adr\_id = adr\_id;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

**Example 10 Inner join doesn't care, which table is the first, which the second one. Compare with** [**Example 8**](http://www.gplivna.eu/papers/sql_join_types.htm#e8)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM addresses

JOIN persons ON prs\_adr\_id = adr\_id;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

The position of additional conditions is irrelevant for INNER joins. These can be written both as join condition and in WHERE clause, result is the same.

**Example 11 Selects first name, surname and city for every person, which first name starts with letter P. Additional condition is placed as join condition. Result is the same as in** [**Example 12**](http://www.gplivna.eu/papers/sql_join_types.htm#e12)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

INNER JOIN addresses ON prs\_adr\_id = adr\_id

AND prs\_first\_name LIKE 'P%';

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

PETER BERZINS BERLIN

**Example 12 Selects first name, surname and city for every person, which first name starts with letter P. Additional condition is placed in WHERE clause. Result is the same as in** [**Example 11**](http://www.gplivna.eu/papers/sql_join_types.htm#e11)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

INNER JOIN addresses ON prs\_adr\_id = adr\_id

WHERE prs\_first\_name LIKE 'P%';

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

PETER BERZINS BERLIN

Following example illustrate the use of OR operator in join condition. From this example is obvious that inner joins are subset of Cartesian joins - for person with name starting with P Cartesian join is performed.

**Example 13 One should be cautious using logical operator OR in join conditions, this can lead to quite interesting results. Compare the result with** [**Example 11**](http://www.gplivna.eu/papers/sql_join_types.htm#e11) **and** [**Example 1**](http://www.gplivna.eu/papers/sql_join_types.htm#e1)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

INNER JOIN addresses ON prs\_adr\_id = adr\_id

OR prs\_first\_name LIKE 'P%';

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

PETER BERZINS RIGA

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

PETER BERZINS NEW YORK

**Example 14 If join condition is always true inner join degrades back to cross join. This is the same** [**Example 5**](http://www.gplivna.eu/papers/sql_join_types.htm#e5)**.**

Next example shows that not always join condition must be equality (=). One can create another join conditions for example join between persons and addresses where identifiers are different. See also the fact that rows from persons where prs\_adr\_id is NULL is not selected, because every compare with NULL always gives false.

**Example 15 Example of inner join which is not equi join, i.e. join condition is not equality.**

SELECT prs\_first\_name, prs\_surname, adr\_city, prs\_adr\_id, adr\_id

FROM persons

INNER JOIN addresses ON prs\_adr\_id <> adr\_id;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY PRS\_ADR\_ID ADR\_ID

--------------- --------------- --------------- ---------- ----------

PETER BERZINS RIGA 2 1

ANN SMYTH RIGA 2 1

CHARLES BERZINS RIGA 2 1

PETER BERZINS NEW YORK 2 3

ANN SMYTH NEW YORK 2 3

CHARLES BERZINS NEW YORK 2 3

Of course tables involved in inner join can be the same, i.e. inner join can be also self join.

**Example 16 Example of table join to itself (self join) - two times table is joined to itself, as a result there are three instances of the same table in this SQL query.**

SELECT father.prs\_first\_name "Father Name",

mother.prs\_first\_name "Mother Name",

child.prs\_first\_name "Child Name"

FROM persons child

INNER JOIN persons father ON child.prs\_father\_id = father.prs\_id

INNER JOIN persons mother ON child.prs\_mother\_id = mother.prs\_id;

Father Name Mother Name Child Name

--------------- --------------- ---------------

PETER ANN CHARLES

**5.4 Alternative syntax**

For this join type one can use also different syntax:

**SELECT <column list>**

**FROM <left joined table>, <right joined table>**

**WHERE <join condition>**

In this syntax all join conditions are written together with other conditions in WHERE clause. Every join written in INNER JOIN syntax, can be rewritten with this syntax and vice versa. Functionally result is the same, i.e. returned rows are the same. Time after time people have quite hot discussion which syntax is better for example "should one use ANSI join syntax when writing an Oracle application? (http://www.freelists.org/archives/oracle-l/10-2006/threads.html#00594)". On the one hand for this syntax it is easier to get Cartesian join on the other hand for "explicit inner join syntax". I personally can write both syntaxes although slightly prefer this alternative syntax simply because more than 10 years ago started to write joins in this manner :

**Example 17 This is equivalent to** [**Example 8**](http://www.gplivna.eu/papers/sql_join_types.htm#e8)**. Both tables are mentioned in FROM clause and join conditions are in WHERE clause.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons, addresses

WHERE prs\_adr\_id = adr\_id;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

**Example 18 This is equivalent to** [**Example 16**](http://www.gplivna.eu/papers/sql_join_types.htm#e16)**. All three tables are mentioned in FROM clause and both join conditions are in WHERE clause.**

SELECT father.prs\_first\_name "Father Name",

mother.prs\_first\_name "Mother Name",

child.prs\_first\_name "Child Name"

FROM persons child, persons father, persons mother

WHERE child.prs\_father\_id = father.prs\_id

AND child.prs\_mother\_id = mother.prs\_id;

Father Name Mother Name Child Name

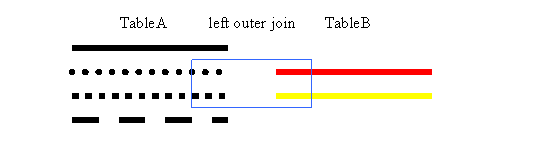
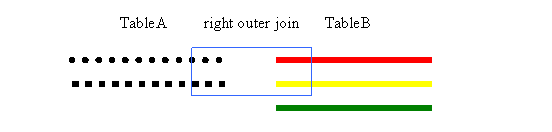
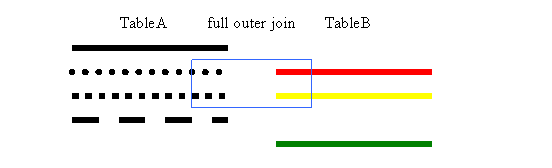
--------------- --------------- ---------------

PETER ANN CHARLES

**6 Outer joins**

Outer join extends the functionality of inner join. It returns following rows:

* the same rows as inner join i.e. rows from both tables, which matches join condition and
* rows from one or both tables, which do not match join condition along with NULL values in place of other table's columns.

There are three types of outer join - Left, Right and Full outer join. Left join keeps rows in left joined table. Right join keeps rows in right joined table. Full join keeps rows in both tables. Let's imagine we have the same tables A and B as defined in [chapter about Cross joins](http://www.gplivna.eu/papers/sql_join_types.htm#p4). Then each of left, right and full join can be graphically shown as follows:  
  
  
  
Rectangle in the pictures above represents result of [inner join](http://www.gplivna.eu/papers/sql_join_types.htm#p5). So we can see that left outer join keeps all rows from left joined table, as much as possible combines with the rows from right table, and those, which cannot be joined, leaves with Nulls on the right side. Right outer join does the same, just table A and B are switched. Full outer join tries to combine as much rows as possible and then rows from left joined table, which cannot be joined, leaves with Nulls on the right side and rows from right joined table, which cannot be joined, leaves with Nulls on the left side.

**6.1 Syntax**

Outer join syntax is as follows:

**SELECT <column list>**

**FROM <left joined table>**

**LEFT|RIGHT|FULL [OUTER] JOIN <right joined table>**

**ON <join condition>**

Only one of the keywords left, right, full can be provided but exactly one is required. Keyword OUTER sometimes is avoided, but anyway keywords left, right or full indicate it is outer join. After the keyword ON join condition is written, generally it can contain many predicates connected with Boolean AND, OR, NOT.

**6.2 Facts to remember**

* Outer joins should be used only when necessary. If it is possible (data model and business data allows) inner join should be used. Inner join offers greater flexibility for optimizer and doesn't mislead people to the thoughts that some rows of one cannot be joined to other table.
* Unlike Inner joins it is important where the condition is written - either as join condition or in WHERE clause. See chapter [Difference between predicates in ON clause vs. predicates in WHERE clause](http://www.gplivna.eu/papers/sql_join_types.htm#p6.4).
* Not all Database management systems support Full outer joins. But one can use [set operator Union](http://www.gplivna.eu/papers/sql_set_operators.htm#p2.3) of Left and Right joins to simulate Full outer join. See [Example 22a](http://www.gplivna.eu/papers/sql_join_types.htm#e22a).

**6.3 Examples**

**Example 19 Selects first name, surname and city for every person keeping all persons, even without address. Compare with** [**Example 8**](http://www.gplivna.eu/papers/sql_join_types.htm#e8)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

LEFT OUTER JOIN addresses ON prs\_adr\_id = adr\_id;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ---------------

JANIS BERZINS

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

**Example 20 Selects first name, surname and city for every person. Keyword outer is not present, but functionally result is the same as in** [**Example 19**](http://www.gplivna.eu/papers/sql_join_types.htm#e19)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

LEFT JOIN addresses ON prs\_adr\_id = adr\_id;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

JANIS BERZINS

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

**Example 21 Selects first name, surname and city for every person keeping all addresses, even without any associated person. Compare with** [**Example 19**](http://www.gplivna.eu/papers/sql_join_types.htm#e19)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

RIGHT OUTER JOIN addresses ON prs\_adr\_id = adr\_id;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ---------------

RIGA

PETER BERZINS BERLIN

CHARLES BERZINS BERLIN

ANN SMYTH BERLIN

NEW YORK

**Example 22 Selects first name, surname and city for every person keeping all personas and addresses. Compare with** [**Example 19**](http://www.gplivna.eu/papers/sql_join_types.htm#e19) **and** [**Example 21**](http://www.gplivna.eu/papers/sql_join_types.htm#e21)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

FULL OUTER JOIN addresses ON prs\_adr\_id = adr\_id;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ---------------

JANIS BERZINS

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

RIGA

NEW YORK

Some databases for example MySQL doesn't support full outer join syntax directly. Then one must use workaround to get desired output. The same effect can be achieved with

* left join unioned with
* right join without rows resulting from inner join.

Of course one can combine left and right joins as well as change the order of joined tables to create more substitutes, but the main idea is the same.

**Example 22a Emulates FULL outer join using left and right join with Union. Compare with** [**Example 22**](http://www.gplivna.eu/papers/sql_join_types.htm#e22)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

LEFT OUTER JOIN addresses ON prs\_adr\_id = adr\_id

UNION ALL

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

RIGHT OUTER JOIN addresses ON prs\_adr\_id = adr\_id

WHERE prs\_adr\_id IS NULL;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ---------------

JANIS BERZINS

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

RIGA

NEW YORK

**Example 23 If the condition is always true outer join degrades to cross join.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

RIGHT OUTER JOIN addresses ON 1 = 1;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

JANIS BERZINS RIGA

PETER BERZINS RIGA

ANN SMYTH RIGA

CHARLES BERZINS RIGA

JANIS BERZINS BERLIN

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

JANIS BERZINS NEW YORK

PETER BERZINS NEW YORK

ANN SMYTH NEW YORK

CHARLES BERZINS NEW YORK

12 rows selected.

Next example shows that not always join condition must be equality (=). One can create another join conditions for example join between persons and addresses where identifiers are different. There is one additional row which (in bold) is the result of outer join.

**Example 24 Example of outer join which is not equi join, i.e. join condition is not equality. Compare with** [**Example 15**](http://www.gplivna.eu/papers/sql_join_types.htm#e15)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city, prs\_adr\_id, adr\_id

FROM persons

LEFT JOIN addresses ON prs\_adr\_id <> adr\_id;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY PRS\_ADR\_ID ADR\_ID

--------------- --------------- --------------- ---------- ----------

**JANIS BERZINS**

PETER BERZINS RIGA 2 1

PETER BERZINS NEW YORK 2 3

ANN SMYTH RIGA 2 1

ANN SMYTH NEW YORK 2 3

CHARLES BERZINS RIGA 2 1

CHARLES BERZINS NEW YORK 2 3

In inner join we had Example 16 of three way self join to get child, his mother and father. It returned only one row. With outer joins we can get more information, also partial information. Look at the rows in bold which are added because of inner join replacement by left join.

**Example 25 Example of table outer join to itself (self join) - two times table is joined to itself, as a result there are three instances of the same table in this SQL query. Compare with** [**Example 16**](http://www.gplivna.eu/papers/sql_join_types.htm#e16)**.**

SELECT father.prs\_first\_name "Father Name",

mother.prs\_first\_name "Mother Name",

child.prs\_first\_name "Child Name"

FROM persons child

LEFT JOIN persons father ON child.prs\_father\_id = father.prs\_id

LEFT JOIN persons mother ON child.prs\_mother\_id = mother.prs\_id;

Father Name Mother Name Child Name

--------------- --------------- ---------------

**JANIS**

**JANIS PETER**

**ANN**

PETER ANN CHARLES

**6.4 Difference between predicates in ON clause vs. predicates in WHERE clause**

For inner joins it was irrelevant where the additional conditions are written - both join condition and conditions in WHERE clause worked the same way - see [Example 11](http://www.gplivna.eu/papers/sql_join_types.htm#e11) and [Example 12](http://www.gplivna.eu/papers/sql_join_types.htm#e12). For outer joins it is important where one places additional conditions.  
Let's imagine we'd like to get only persons having surname "Berzins" along with cities where they live. One has to add another condition in WHERE clause in [Example 19](http://www.gplivna.eu/papers/sql_join_types.htm#e19).

**Example 26 Selects first name, surname and city for every person keeping all persons, even without address. There is additional condition in WHERE clause for table persons.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

LEFT JOIN addresses ON prs\_adr\_id = adr\_id

WHERE prs\_surname = 'BERZINS';

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

JANIS BERZINS

PETER BERZINS BERLIN

CHARLES BERZINS BERLIN

As expected Ann Smyth is lost from result, because she has different surname. Now let's put condition as join condition.

**Example 27 Selects first name, surname and city for every person keeping all persons, even without address. There is additional condition in join clause for table persons.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

LEFT JOIN addresses ON prs\_adr\_id = adr\_id

AND prs\_surname = 'BERZINS';

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

JANIS BERZINS

PETER BERZINS BERLIN

ANN SMYTH

CHARLES BERZINS BERLIN

Ann is back! Just she hasn't address. Why?  
Let's look closer at mechanism how outer join result theoretically is created!  
1. Firstly Cartesian join of involved table's rows is created. This is another reason why Cartesian join concept is important.  
2. For each row in Cartesian join result join conditions are being checked.  
2.1. If join condition is evaluated to TRUE, then this row is preserved.  
2.2. If join condition is evaluated to FALSE, then this row also is preserved, with following changes:  
2.2.1. Columns from the table, for which rows **should be** kept (left joined table persons in the example above), are preserved without changes.  
2.2.2. Columns from the table, for which rows **should not be** kept (table addresses in the example above), are changed to NULL.  
3. Only unique rows are preserved from all the rows created by step 2.2.  
4. Finally WHERE clause is evaluated and preserved only these rows, which satisfy WHERE condition (-s).  
So the answer why Ann was back in result set can be found in step 2.2 and its sub steps. Row is preserved along with columns from table persons, but columns from table addresses are set to NULL.  
So with already known experience let's look in two more examples where additional conditions will be put on table addresses. In the following example none of the rows satisfies condition in WHERE clause, and therefore all of them are being discarded from result set according to step 4 in the algorithm above.

**Example 28 Selects first name, surname and city for every person keeping all persons, even without address. There is additional condition in WHERE clause for table addresses.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

LEFT JOIN addresses ON prs\_adr\_id = adr\_id

WHERE adr\_city = 'RIGA';

no rows selected

In the next example none of the rows from Cartesian join satisfies join condition but rows are preserved anyway, just columns from tables addresses are set to NULL according to step 2.2 in the algorithm above.

**Example 29 Selects first name, surname and city for every person keeping all persons, even without address. There is additional condition in join clause for table addresses.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

LEFT JOIN addresses ON prs\_adr\_id = adr\_id

AND adr\_city = 'RIGA';

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

JANIS BERZINS

PETER BERZINS

ANN SMYTH

CHARLES BERZINS

**6.5 Old style Oracle syntax for Outer joins**

Even before ANSI standardized outer joins Oracle had a way to express them. Of course they have to pay for being one of the first ones, old syntax is not according to standard and has less features. However it is working and one can use it. The idea is the same as for alternative inner join syntax, just a special outer join operator (+) have to be applied for conditions for outer joined table. So the overall syntax is:

**SELECT <column list>**

**FROM <left joined table>, <right joined table>**

**WHERE <join condition>**

Let's look at example which is functionally equivalent to left outer join in [Example 19](http://www.gplivna.eu/papers/sql_join_types.htm#e19).

**Example 30 This is equivalent to** [**Example 19**](http://www.gplivna.eu/papers/sql_join_types.htm#e19)**, table addresses is outer joined to table persons using Oracle outer join operator (+).**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons, addresses

WHERE prs\_adr\_id = adr\_id (+);

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

JANIS BERZINS

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

In order to get right outer join we just move outer join operator to other side of equivalence

**Example 31 This is equivalent to** [**Example 20**](http://www.gplivna.eu/papers/sql_join_types.htm#e20)**, table persons is outer joined to table addresses using Oracle outer join operator (+).**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons, addresses

WHERE prs\_adr\_id (+) = adr\_id;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

RIGA

PETER BERZINS BERLIN

CHARLES BERZINS BERLIN

ANN SMYTH BERLIN

NEW YORK

Oracle outer join operator has more restrictions than more modern ANSI syntax, for example:

* It doesn't support Full Outer join;
* It cannot be used together with modern syntax;
* Join condition cannot contain logical OR and IN operators.

Other common mistakes used in old style Oracle outer joins one can see in article [Common errors seen when using OUTER-JOIN](http://www.orafaq.com/node/855).

**6.6 Old style SQL Server syntax for Outer joins**

SQL Server has created his own syntax for outer joins just as Oracle. And it similarly has special operator. For SQL Server it was \*= and =\* instead of simply = in join conditions.

**Example 32 This is equivalent to Example 19, table addresses is outer joined to table persons using SQL Server outer join operator \*=.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons, addresses

WHERE prs\_adr\_id \*= adr\_id;

JANIS BERZINS NULL

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

In order to get right outer join we just put the \* sign on the other side of equivalence =

**Example 33 This is equivalent to Example 20, table persons is outer joined to table addresses using SQL Server outer join operator =\***

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons, addresses

WHERE prs\_adr\_id =\*adr\_id;

NULL NULL RIGA

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

NULL NULL NEW YORK

Unlike in Oracle where one can use both syntaxes without any restriction, SQL Server 2005 and 2008 by default cannot use it, one has to set compatibility level 80 (SQL Server 2000).

**7 Equi join**

To be sure that particular join is equi join one has to look at join condition. If it contains only equality operator in join conditions then it is equi join. Usually this is the most used join type. Even more - it is so often used that probably some people even cannot imagine the fact that something more than equality operator can be used.  
There isn't special kind of syntax just for equi joins. If one looks at the scheme above it finds out following characteristics:

* Natural join is always equi join because it has hidden join condition - equivalence of commonly named columns. Except of course when Natural join has degraded back to Cross join, because there isn't any common column;
* Named columns joins are always equi joins because it has hidden join condition - equivalence of columns listed in USING clause;
* Inner, outer and self joins can be or not be equi joins. It is dependant on operators in join condition;
* It is not applicable to speak about cross join as equi or not as it haven't any join condition at all.

**7.1 Examples**

Most of until now used examples in this article were equi joins. Let's take this one:

**Example 34 Selects first name, surname and city for every person. The same as** [**Example 8**](http://www.gplivna.eu/papers/sql_join_types.htm#e8)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

INNER JOIN addresses ON prs\_adr\_id = adr\_id;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ------------

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

**Example 35 Selects first name, surname and city for every person keeping all persons, even without address. The same as** [**Example 19**](http://www.gplivna.eu/papers/sql_join_types.htm#e19)**.**

SELECT prs\_first\_name, prs\_surname, adr\_city

FROM persons

LEFT OUTER JOIN addresses ON prs\_adr\_id = adr\_id;

PRS\_FIRST\_NAME PRS\_SURNAME ADR\_CITY

--------------- --------------- ---------------

JANIS BERZINS

PETER BERZINS BERLIN

ANN SMYTH BERLIN

CHARLES BERZINS BERLIN

**8 Non-Equi (Theta) join**

Probably it is something new for you but join conditions can have operators other than equivalence. It is quite rarely used in practice, more just like a scientific possibility. On the other hand I'm quite sure one of the reasons why it is not used - is the fact that many people cannot imagine such possibility and overlook cases when it would be possible.  
There isn't special syntax for these joins, except the fact that join condition must be explicitly used.

**8.1 Facts to remember**

* Unless You my dear reader think You are real join virtuoso - do test non-equi joins on real data and with even more care than You test other SQL statements. They may produce surprising results if not used cautiously.

**8.2 Examples**

**Example 36 Selects all pairs of first names where first one is longer than second one.**

SELECT p1.prs\_first\_name, p2. prs\_first\_name

FROM persons p1

INNER JOIN persons p2

ON length(p1.prs\_first\_name) > length(p2.prs\_first\_name);

PRS\_FIRST\_NAME PRS\_FIRST\_NAME

--------------- ---------------

CHARLES JANIS

CHARLES PETER

CHARLES ANN

JANIS ANN

PETER ANN

Probably not much real value in previous example. More real value can be in cases when one has somehow group people or whatever else. For example we have groups with dynamic upper and lower bounds stored in database. Let's imagine our persons are grouped according to their first name to different groups. We will create following table and add following data for that:

CREATE TABLE groups (

grp\_name VARCHAR(15) NOT NULL,

grp\_min\_value VARCHAR(1) NOT NULL,

grp\_max\_value VARCHAR(1) NOT NULL);

INSERT INTO groups VALUES ('First group', 'A', 'G');

INSERT INTO groups VALUES ('Second group', 'H', 'N');

INSERT INTO groups VALUES ('Third group', 'O', 'Z');

COMMIT;

Now we can split people to different groups according to their first name using operator BETWEEN. Of course on has to remember that only first symbol of each name should be used because otherwise for example name Gints won't be in any group because Gints is after lower bound of first group (G) but before than upper bound of second group (H).

**Example 37 Selects each first name, surname and their group for each person.**

SELECT prs\_first\_name, prs\_surname, grp\_name

FROM persons

INNER JOIN groups

ON substr(prs\_first\_name, 1, 1)

BETWEEN grp\_min\_value AND grp\_max\_value;

PRS\_FIRST\_NAME PRS\_SURNAME GRP\_NAME

--------------- --------------- ------------

ANN SMYTH First group

CHARLES BERZINS First group

JANIS BERZINS Second group

PETER BERZINS Third group

**9 Self join**

With self joins it is quite simple. Self join is every other join type if both tables are the same. On the first spot probably it is not so obvious how to write such construction, but in this case I suggest following scenario:

* Let's imagine we have two different tables with the same column names and the same data;
* Let's create necessary SQL statement with join using two different tables and using aliases for their names. This time everything is quite clear?
* Then just replace another table name with the same one. SQL statement is yet valid because everywhere aliases are used. And voila! That's it!

There isn't any special syntax for self joins, just used tables in them does matter.

**9.1 Examples**

Let's remember some of already used self joins throughout this article. All the following examples are self joins [Example 7](http://www.gplivna.eu/papers/sql_join_types.htm#e7), [Example 16](http://www.gplivna.eu/papers/sql_join_types.htm#e16), [Example 18](http://www.gplivna.eu/papers/sql_join_types.htm#e18) and [Example 25](http://www.gplivna.eu/papers/sql_join_types.htm#e25). The most interesting probably is this one:

**Example 38 Example of table outer join to itself (self join) - two times table is joined to itself, as a result there are three instances of the same table in this SQL query. The same as** [**Example 25**](http://www.gplivna.eu/papers/sql_join_types.htm#e25)**.**

SELECT father.prs\_first\_name "Father Name",

mother.prs\_first\_name "Mother Name",

child.prs\_first\_name "Child Name"

FROM persons child

LEFT JOIN persons father ON child.prs\_father\_id = father.prs\_id

LEFT JOIN persons mother ON child.prs\_mother\_id = mother.prs\_id;

Father Name Mother Name Child Name

--------------- --------------- ---------------

JANIS

JANIS PETER

ANN

PETER ANN CHARLES

As we can observe there are 3 instances of the same table in one row: child, father and mother. And this usually is true - self joins are used in some kind recursive relationships. However one has to understand that self joins have limited count of recursions this can be 1, 2, ..., n but for unknown number of recursions one has to use recursive queries - START WITH .. CONNECT BY clause in Oracle and Common table expressions in SQL Server.

**10 Natural join**

Natural join is special kind of syntax. Unlike explicit inner and outer join syntax as described above natural join syntax has implicit join condition. It is both its strength and weakness - strength because one can write a bit less words, weakness - because it is quite dangerous and unprotected to structure changes of tables. What is this implicit join condition? Natural join combines two tables based on their common columns i.e. columns with the same name. Therefore join condition is hidden and dependant on table structures at runtime. This obviously creates potential future danger - as soon as table structure changes, result can become unpredictable yet syntactically correct.

**10.1 Syntax**

Syntax for Natural joins is as follows:

**SELECT <column list>**

**FROM <left joined table>**

**NATURAL <join type> JOIN <right joined table>**

Join type can be one of the following:

* empty - this means Inner join;
* INNER;
* LEFT [OUTER] - keyword outer is optional;
* RIGHT [OUTER] - keyword outer is optional;
* FULL [OUTER] - keyword outer is optional.

**10.2 Facts to remember**

* Natural join actually performs the same inner or one of the outer joins. It is just another syntactic convention.
* Natural join always is equi-join; join condition always is equivalence between common columns.
* If joined tables haven't common column then natural joins degrades back to Cartesian join. See [Example 41](http://www.gplivna.eu/papers/sql_join_types.htm#e41).
* Natural join syntax is dangerous and may lead to unpleasant surprises because of zero common columns between joined tables or too much common columns between joined tables. Therefore better never use it in real production program code. For more possible risks connected with natural join syntax see blog post [Natural joins are evil](http://gplivna.blogspot.com/2007/10/natural-joins-are-evil-motto-if-you.html).

**10.3 Examples**

Because natural joins need tables with the same column names example tables used earlier in this article are not usable. So we create new tables.

CREATE TABLE countries (

cntr\_id INTEGER NOT NULL PRIMARY KEY,

country VARCHAR(15) NOT NULL);

CREATE TABLE cities (

cty\_id INTEGER NOT NULL PRIMARY KEY,

cntr\_id INTEGER,

city VARCHAR(15) NOT NULL);

CREATE TABLE places (

plc\_id INTEGER NOT NULL PRIMARY KEY,

cty\_id INTEGER,

text VARCHAR(15) NOT NULL);

INSERT INTO countries VALUES (1, 'LATVIA');

INSERT INTO countries VALUES (2, 'ESTONIA');

INSERT INTO cities VALUES (1, 1, 'RIGA');

INSERT INTO cities VALUES (2, 1, 'LIEPAJA');

INSERT INTO cities VALUES (3, 2, 'TALLINN');

INSERT INTO places VALUES (1, 1, 'OLD RIGA');

INSERT INTO places VALUES (2, 2, 'KARA OSTA');

COMMIT;

**Example 39 Selects countries and cities using natural inner join syntax.**

SELECT country, city

FROM countries

NATURAL INNER JOIN cities;

COUNTRY CITY

--------------- ----------

LATVIA RIGA

LATVIA LIEPAJA

ESTONIA TALLINN

**Example 40 Selects cities and places using natural left outer join syntax and keeping all cities.**

SELECT city, text

FROM cities

NATURAL LEFT OUTER JOIN places;

CITY TEXT

--------------- ---------------

RIGA OLD RIGA

LIEPAJA KARA OSTA

TALLINN

**Example 41 In case tables haven't common column name then natural join degrades back to Cartesian join.**

SELECT country, text

FROM countries

NATURAL JOIN places;

COUNTRY TEXT

--------------- ---------

LATVIA OLD RIGA

LATVIA KARA OSTA

ESTONIA OLD RIGA

ESTONIA KARA OSTA

As soon as there are more than one common column in tables join condition is placed on all of them. Usually it is not what we really want. For example imagine we are adding both countries and cities a new column "notes". This column practically never will have the same values and it shouldn't be used in join condition. But natural join does that, and this time as all values are NULL, we get zero rows back.

**Example 42 Selects countries and cities using natural inner join syntax. But another column is added for both tables' countries and cities. Compare with** [**Example 39**](http://www.gplivna.eu/papers/sql_join_types.htm#e39)**.**

ALTER TABLE countries ADD notes VARCHAR(100);

ALTER TABLE cities ADD notes VARCHAR(100);

SELECT country, city

FROM countries

NATURAL INNER JOIN cities;

no rows selected

**11 Named columns join**

Named columns join is some kind of enhanced version of natural join. One can explicitly name columns, which must be used in join condition. Therefore it isn't as dangerous as natural join but simply is just for limited use because not always column names to be joined are the same.

**11.1 Syntax**

**SELECT <column list>**

**FROM <left joined table>**

**<join type> JOIN <right joined table>**

**USING (<join column list>)**

Join type can be one of the following:

* empty - this means Inner join;
* INNER;
* LEFT [OUTER] - keyword outer is optional;
* RIGHT [OUTER] - keyword outer is optional;
* FULL [OUTER] - keyword outer is optional.

In join column list one have to write comma delimited column list which are in both joined tables and are used to create equivalence condition. Named columns joins always are equi joins.

**11.2 Facts to remember**

* This syntax type can be used only in case the joined columns have the same name in both involved tables.

**11.3 Examples**

We will use the same tables as defined for natural joins.

**Example 43 Selects countries and cities using named columns join syntax. Compare with** [**Example 42**](http://www.gplivna.eu/papers/sql_join_types.htm#e42) **and see how this time example is working again as expected.**

SELECT country, city FROM countries

JOIN cities USING (cntr\_id);

COUNTRY CITY

--------------- ---------------

LATVIA RIGA

LATVIA LIEPAJA

ESTONIA TALLINN

**Example 44 Selects countries and cities using named columns join full outer join syntax and using column notes.**

SELECT country, city FROM countries

FULL JOIN cities USING (notes);

COUNTRY CITY

--------------- ---------------

ESTONIA

LATVIA

TALLINN

LIEPAJA

RIGA

**12 Example usage for various DBMSes**

**12.1 Oracle**

All examples except [Example 32](http://www.gplivna.eu/papers/sql_join_types.htm#e32), [Example 33](http://www.gplivna.eu/papers/sql_join_types.htm#e33) (which were specifically created for SQL Server) were created and tested on Oracle 10g. They should work on Oracle versions since 9i, except probably on very early versions of 9i due to bugs in implementation of ANSI join syntax. But these versions you shouldn't use anyway.

**12.2 Microsoft SQL Server**

Examples were tested on SQL Server 2008. SQL Server doesn't support Natural join and Using clause syntax and therefore following examples are not valid for it [Example 3](http://www.gplivna.eu/papers/sql_join_types.htm#e3), [Example 4](http://www.gplivna.eu/papers/sql_join_types.htm#e4), and all examples starting with [Example 39](http://www.gplivna.eu/papers/sql_join_types.htm#e39) till [Example 44](http://www.gplivna.eu/papers/sql_join_types.htm#e44).  
Following examples uses Oracle specific outer join operator and therefore isn't usable in SQL Server [Example 30](http://www.gplivna.eu/papers/sql_join_types.htm#e30), [Example 31](http://www.gplivna.eu/papers/sql_join_types.htm#e31).  
In [Example 36](http://www.gplivna.eu/papers/sql_join_types.htm#e36) function length should be replaced by function len.  
In [Example 37](http://www.gplivna.eu/papers/sql_join_types.htm#e37) function substr should be replaced by function substring.

**12.3 MySQL**

Examples were tested on MySQL 6.0. However there are some problems with them. In MySQL Cross Join is equivalent to Inner Join therefore [Example 1](http://www.gplivna.eu/papers/sql_join_types.htm#e1) works as expected only because it lacks join condition.  
[Example 3](http://www.gplivna.eu/papers/sql_join_types.htm#e3), [Example 4](http://www.gplivna.eu/papers/sql_join_types.htm#e4), [Example 39](http://www.gplivna.eu/papers/sql_join_types.htm#e39) and [Example 42](http://www.gplivna.eu/papers/sql_join_types.htm#e42) do not work because MySQL only partially supports Natural join syntax.  
MySQL doesn't support Full Outer join therefore [Example 22](http://www.gplivna.eu/papers/sql_join_types.htm#e22) and [Example 44](http://www.gplivna.eu/papers/sql_join_types.htm#e44) do not work ([Example 44](http://www.gplivna.eu/papers/sql_join_types.htm#e44) doesn't give an error because keyword FULL is considered as alias).  
Following examples do not work because are specific to Oracle or SQL Server [Example 30](http://www.gplivna.eu/papers/sql_join_types.htm#e30), [Example 31](http://www.gplivna.eu/papers/sql_join_types.htm#e31), [Example 32](http://www.gplivna.eu/papers/sql_join_types.htm#e32), [Example 33](http://www.gplivna.eu/papers/sql_join_types.htm#e33).

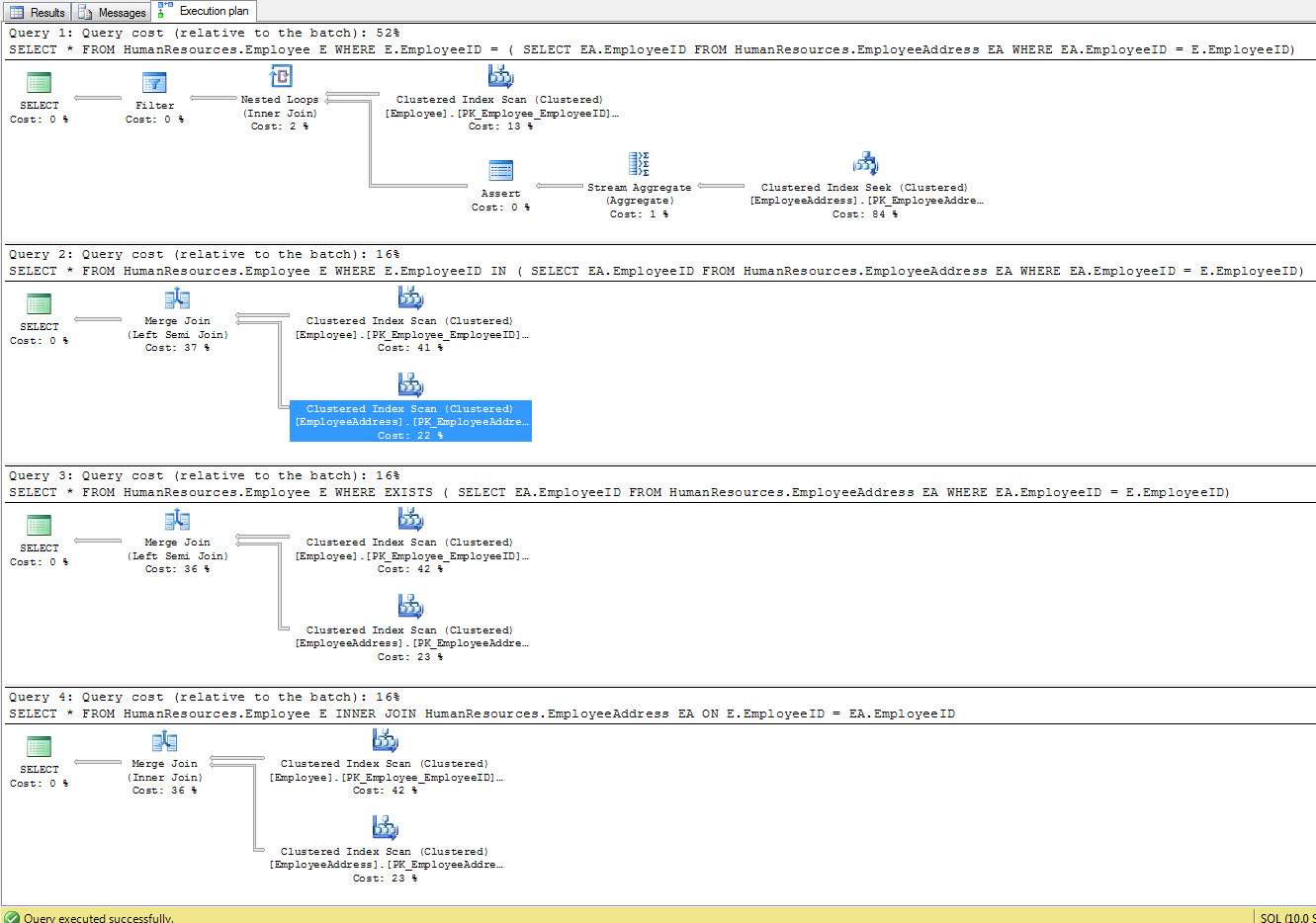
## SQL SERVER – Subquery or Join – Various Options – SQL Server Engine knows the Best

This is followup post of my earlier article SQL SERVER – Convert IN to EXISTS – Performance Talk, after reading all the comments I have received I felt that I could write more on the same subject to clear few things out.

First let us run following four queries, all of them are giving exactly same resultset.

USE AdventureWorks  
GO  
-- use of =  
SELECT \*  
FROM HumanResources.Employee E  
WHERE E.EmployeeID = ( SELECT EA.EmployeeID  
FROM HumanResources.EmployeeAddress EA  
WHERE EA.EmployeeID = E.EmployeeID)  
GO  
-- use of in  
SELECT \*  
FROM HumanResources.Employee E  
WHERE E.EmployeeID IN ( SELECT EA.EmployeeID  
FROM HumanResources.EmployeeAddress EA  
WHERE EA.EmployeeID = E.EmployeeID)  
GO  
-- use of exists  
SELECT \*  
FROM HumanResources.Employee E  
WHERE EXISTS ( SELECT EA.EmployeeID  
FROM HumanResources.EmployeeAddress EA  
WHERE EA.EmployeeID = E.EmployeeID)  
GO  
-- Use of Join  
SELECT \*  
FROM HumanResources.Employee E  
INNER JOIN HumanResources.EmployeeAddress EA ON E.EmployeeID = EA.EmployeeID  
GO

Let us compare the execution plan of the queries listed above. Click on image to see larger image.



It is quite clear from the execution plan that in case of IN, EXISTS and JOIN SQL Server Engines is smart enough to figure out what is the best optimal plan of Merge Join for the same query and execute the same. However, in the case of use of Equal (=) Operator, SQL Server is forced to use Nested Loop and test each result of the inner query and compare to outer query, leading to cut the performance. Please note that here I no mean suggesting that Nested Loop is bad or Merge Join is better. This can very well vary on your machine and amount of resources available on your computer.

When I see Equal (=) operator used in query like above, I usually recommend to see if user can use IN or EXISTS or JOIN. As I said, this can very much vary on different system. ***What is your take in above query?*** I believe SQL Server Engines is usually pretty smart to figure out what is ideal execution plan and use it.