The AMOUNT column and totals are calculated fields and are generally not stored. The total value goes to another table under the Accounts Receivable department showing what the customer owes.

Each of these 2 lines is one row in the ORDERLINES table.

The product name and sell price are from the PRODUCTS table

**Types of Joins**

1. **EQUI-JOINS** (also called INNER JOINS)

To get customer id, customer name, order number and order\_date. You need to look at what tables hold this data. … CUSTOMERS hold the name and number, ORDERS hold the order number, date and customer number as an FK to customer. **You will join over matching the PK column of the Parent table (Customer) with the FK column of the Child table (Orders).** Since I don't want to see all the orders from the customer, I am limiting it to order\_id < 6

1. **OLD SCHOOL METHOD (prior to Oracle 9i)**

**SELECT c.customer\_id, name , order\_id, order\_date**

**FROM customers c, orders o**

**WHERE c.customer\_id = o.customer\_id**

**AND order\_id < 6**

**ORDER BY 3**

|  |  |  |  |
| --- | --- | --- | --- |
| **CUSTOMER\_ID** | **NAME** | **ORDER\_ID** | **ORDER\_DATE** |
| 4 | AbbVie | 1 | 15-OCT-17 |
| 4 | AbbVie | 2 | 26-APR-15 |
| 5 | Centene | 3 | 26-APR-17 |
| 8 | International Paper | 4 | 09-APR-15 |
| 5 | Centene | 5 | 09-APR-17 |

**B) NEW METHOD (with Oracle 9i, and after)**

This is the improved way by using JOIN ON

This due date is calcuated by order\_dt + 1 month

**SELECT c.customer\_id, c.name, o.order\_id, o.order\_date**

**FROM customers c JOIN orders o**

Here are the join conditions

**ON (c.customer\_id = o.customer\_id)**

**WHERE o.order\_id < 6**

And this is the WHERE condition that limits the rows to look at.

**ORDER BY 3**

**C) USING clause (only if PK and FK column hold the same name)**

**SELECT c.customer\_id, c.name, o.order\_id, o.order\_date**

**FROM customers c JOIN orders o**

**USING (customer\_id)**

**WHERE o.order\_id < 6**

**ORDER BY 3**

Notice that here you must NOT use a Table Prefix with the Used Column, so cust\_no in the Select line must be without c. prefix (otherwise syntax error)

1. **OUTER JOINS**

You are running a little kids baseball club that every spring needs to register children into teams. To keep it very simple you have 2 tables called TEAMS and KIDS

TEAMS [teamid, temaname]

KIDS [kidid, kidname, teamid] The teamid starts as null until the child is assigned to a team.

PROBLEM to solve:

After 5 days the CEO of the baseball league asks for

Give me the team name and kid name in that team.

ANSWER: Using Equi-joins

SELECT teamname, kidname

FROM kids k, teams t

WHERE k.teamid = t.teamid;

Better method:

SELECT teamname, kidname

FROM kids k, JOIN teams t

ON (k.teamid = t.teamid);

The result is a listing of teams that have kids assigned to them already. However, if there wasn't a match based on the WHERE then some teams were not mentioned and some kids who registered but were not assigned a team and will not appear as output.

Show me ALL teams and any kid assigned to them

OUTER JOINS

SELECT teamname, kidname

FROM kids k, teams t

WHERE k.teamid = t.**teamid (+); 🡸 old method**

BETTER METHOD ….

SELECT teamname, kidname

FROM kids k RIGHT JOIN teams t 🡸 RIGHT because TEAM is on the right

ON k.teamid = t.teamid

Show me teams with kids and any kid not assigned to a team

SELECT teamname, kidname

FROM kids k LEFT JOIN teams t 🡸 LEFT because KIDS is on the left

ON k.teamid = t.teamid

SELECT teamname, kidname

FROM teams t RIGHT JOIN kids k 🡸 it is RIGHT, want all kids. We just changed table order

ON k.teamid = t.teamid

Show me all teams and all kids even if the kids are not yet matched up to a team. You want left and right side.

SELECT teamname, kidname

FROM kids k FULL JOIN teams t

ON k.teamid = t.teamid

1. **SELF JOINS**

Sometimes (usually it occurs with employee tables) you need to join a copy of the table to itself.

I want a list of manager names and the employees that work for them. Because managers and non-managers are all employees there is no need for a separate manager table.

Notice that all employees have the data about their name, job, and also an ID pointing to who their manager is

176 Taylor other info … 149 (manager id) 80(dept id)

To know Taylor’s manager, we look for manger id 149 and match it to employee 149 to get the manager.

**We need to use the same table Twice, but with a different Table Alias 🡪 Self Join.**

SELECT w.last\_name “Worker” , m.last\_name “Manager”

FROM employee w, employee m

WHERE m.employee\_id = w.manager\_id

ORDER BY w.last\_name

|  |  |
| --- | --- |
| **Worker** | **Manager** |
| Abel | Zlotkey |
| Davies | Mourgos |
| De Haan | King |
| Ernst | Hunold |
| Fay | Hartstein |
| Gietz | Higgins |
| Grant | Zlotkey |
| Hartstein | King |
| Higgins | Kochhar |
| Hunold | De Haan |
| Kochhar | King |
| Lorentz | Hunold |
| Matos | Mourgos |
| Mourgos | King |
| Rajs | Mourgos |
| Taylor | Zlotkey |
| Vargas | Mourgos |
| Whalen | Kochhar |
| Zlotkey | King |

19 rows selected

What happens if we rotate table aliases in the WHERE clause?

Try that and make a conclusion.

1. **NON-EQUI JOINS**

READ about it in the oracle NOTES