Structured Query Language

DI Markus Haslinger MSc

DBI/INSY 3rd year

HTL LEONDING

Agenda

HTL LEONDING

SQL

Definition

Structured Query Language (SQL) is a standard computer language for relational database management and data manipulation. SQL is used to query, insert, update and modify data.[1]

- Despite the name → more than just queries
- Designed with the relational model in mind
- 'Lingua Franca' for RDBMS
  - But with variations in most implementations

[1] https://www.techopedia.com/definition/1245/structured-query-language-sql, 2018-08-16

Syntax

```
SELECT [DISTINCT] expression1 [alias_1] [, expression2 [alias_2], ....]
FROM table_name1 [,table_name2, ....]
[WHERE condition]
[GROUP BY grouping_expr1 [, grouping_expr2, ....]]
[HAVING having_condition]
[ORDER BY order_expr1 [, order_expr2, ....]]
```

- Each SELECT statement has at least two clauses (SELECT & FROM)
- The order in which clauses (may) appear is fixed
- A HAVING clause can only exist together with a GROUP BY clause

HTL LEONDING

Literals

- Numeric
  - Integer (WHERE Quantity = 3)
  - Floating point (WHERE Price > 99.9)
- Alphanumeric
  - WHERE LastName = 'Knuth'
- Dates
  - WHERE DateOfBirth > '1982-04-01'
- Usually available in every RDBMS

HTL LEONDING

System variables

- Similar but with different syntax and availability depending on the RDBMS
- Examples:
  - USER
  - SYSDATE
  - ROWNUM
  - LEVEL

  - SELECT user, sysdate, rownum FROM dual; [Oracle]

Numeric expressions

- Operators (+, -, *, /, mod)
- NULL
- Function NVL [Oracle]
  - SELECT NVL(leagueno, 'no league') FROM Players;
  - Replaces all NULL values with the provided value

HTL LEONDING

Date expressions [Oracle]

- Date differences, e.g. SELECT SYSDATE - PenaltyDate FROM Penalties;
- Date Input and Output:
  - Requires format string
  - TO_DATE('2018-08-16', 'YYYY-MM-DD')
  - TO_CHAR(SYSDATE, 'DD/MM/YYYY HH24:MI')
  - Format string is case sensitive
  - See `https://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements004.htm`

HTL LEONDING

Date expressions [Oracle]

TO_CHAR(PENALTYDATE,'**Dy**,DD-MONTH-YYYY')
Mo, 08-Dezember -1980
Di, 05-Mai -1981
Sa, 10-September-1983
Sa, 08-Dezember -1984
Mo, 08-Dezember -1980
Mo, 08-Dezember -1980
Do, 30-Dezember -1982
Mo, 12-November -1984

TO_CHAR(PENALTYDATE,'**DY**,DD-MONTH-YYYY')
MO, 08-Dezember -1980
DI, 05-Mai -1981
SA, 10-September-1983
SA, 08-Dezember -1984
MO, 08-Dezember -1980
MO, 08-Dezember -1980
DO, 30-Dezember -1982
MO, 12-November -1984

HTL LEONDING

Basic query

- FROM defines the source (tables)
- SELECT defines the projection on the result
  - without condition the source is already the result
- **SELECT** Name, City **FROM** Players;

	NAME	CITY
1	Parmenter	Stratford
2	Baker	Inglewood
3	Hope	Stratford
4	Everett	Stratford
5	Collins	Eltham
6	Moorman	Eltham
7	Wise	Stratford

HTL LEONDING

Schemas and Pseudonyms

- Tables can reside in a schema, e.g. **mycorp.**employees
- Oracle calls this user- or workspace

- Pseudonyms for tables can be used
  - **SELECT** p.Name, p.City **from** Players p
  - Especially useful for joins
- Aliases for columns can be used as well
  - **SELECT** Name "LastName", City **from** Players

- DISTINCT removes duplicate rows from the result set

HTL LEONDING

Statistical functions

- Calculated over the query result ⇒ single row returned
  - Exception: GROUP BY
  - Affected by DISTINCT
- Available functions:
  - COUNT
  - MIN
  - MAX
  - SUM
  - AVG
  - STDDEV
  - VARIANCE

HTL LEONDING

Statistical functions

- **SELECT COUNT**(Amount) **from** Penalties;
- **SELECT COUNT**(**DISTINCT** Amount)**from** Penalties;

	PENALTYNO	PLAYERNO	PENALTYDATE	AMOUNT
1	1	6	08.12.80	100
2	2	44	05.05.81	75
3	3	27	10.09.83	100
4	4	104	08.12.84	50
5	5	44	08.12.80	25
6	6	8	08.12.80	25
7	7	44	30.12.82	30
8	8	27	12.11.84	75

HTL LEONDING

WHERE clause

- Filters/restricts the query result
- Only those rows are returned which fulfill the conditions
- Comparison operators are used
  - =, <, >, <=, >=, <> (or '! =')
- Comparable are chars, numbers and dates [Oracle]
- Alphanumerical comparison based on ANSI value
- Multiple values can be compared using = and <> [Oracle]
  - **WHERE** (c1, c2)= (v1, v2)
- Possible results: TRUE, FALSE, **UNKNOWN** [Oracle]

HTL LEONDING

WHERE clause [Oracle]

and	T	F	?
T	T	F	?
F	F	F	F
?	?	F	?

or	T	F	?
T	T	T	T
F	T	F	?
?	T	?	?

not	T	F	?
	F	T	?

HTL LEONDING

BETWEEN operator

- Syntax: expr1 [**NOT**] **BETWEEN** expr2 **AND** expr3
- Returns TRUE if
  - value $\geq$ lower bound
  - value $\leq$ upper bound
- If any of the expressions (expr1, expr2, expr3) is NULL the result is UNKNOWN [Oracle]
- Example: **SELECT** $*$ **from** Employees **WHERE** EmployeeNo **BETWEEN** 120 **AND** 200;

HTL LEONDING

LIKE operator

- Syntax: expr1 [**NOT**] **LIKE** expr2
- Used for wildcard string comparison
  - '%' ⇒ any 0, 1 or n chars (cf. '.*' Regex)
  - '_' ⇒ any exactly one character
- Example:
  SELECT ∗ **from** Employees **WHERE** Name **LIKE** 'K%';
- Be careful with (6 byte) blank patting of char (vs. varchar2) [Oracle]

HTL LEONDING

IN operator

- Syntax: expr1 [**NOT**] **IN** expr2
- Returns TRUE if value (element) is contained in a set
- Example:
  SELECT ∗ **from** Employees **WHERE** EmployeeNo **IN** (123,321);

HTL LEONDING

Subqueries with the IN operator

- Syntax:
  WHERE colName1 **IN** (**SELECT** colName2 **FROM** table2)
- Avoids copy/pasting value lists for the IN clause
- Updated with every execution
- Especially useful for large lists
- If the subquery does not return a result (0 rows)
  - IN (subquery) evaluates to FALSE
  - NOT IN (subquery) evaluates to TRUE

HTL LEONDING

Subqueries with the IN operator

Example

- Tables:
  - Customer(CustNo, LastName, FirstName, ZIP, City, Street, StreetNo, DateOfBirth)
  - Order(OrderNo, CustNo, Amount, Date)
- Select all orders from customers who have a last name starting with 'Ma'
- **SELECT** * **FROM Order WHERE** CustNo **IN** (**SELECT** CustNo **FROM** Customer **WHERE** LastName **LIKE** 'Ma%')

HTL LEONDING

Comparisons with subqueries

- Syntax: expr operator (subquery))
- Possible operators: $<$, $>$, $=$, $>=$, $<=$, $<>$
- When using a comparison operator the subquery must only return a single value

Comparisons with subqueries

Example

- Select all orders from customers who match these criteria: LastName=Maier, FirstName=Franz, ZIP=4020, City=Linz, DateOfBirth=1980-05-04

```
SELECT * FROM Order WHERE CustNo =
(SELECT CustNo FROM Customer WHERE LastName='Maier'
AND FirstName='Franz'
AND ZIP=4020 AND City='Linz'
AND DateOfBirth='1980-05-04');
```

HTL LEONDING

Subqueries with ALL/ANY operator

- Syntax: expr comparison_operator (**ALL**|**ANY**)(subquery))
- ALL evaluates to TRUE if all rows returned by the subquery fulfill the condition
- ANY evaluates to TRUE if at least one row returned by the subquery fulfills the condition

- Consider expressing the same conditions using the MIN() and MAX() functions

HTL LEONDING

Subqueries with ALL/ANY operator

Example

- Table Customer(CustNo, LastName, FirstName, ZIP, City, Street, StreetNo, DateOfBirth, Revenue)
- Select all customers from Wels who have a higher revenue than any customer from Linz

```
SELECT * FROM Customer
WHERE City='Wels' AND Revenue >= ANY
(SELECT Revenue FROM Customer WHERE City='Linz');
```

HTL LEONDING

Subqueries with ALL/ANY operator

Example

- Table Customer(CustNo, LastName, FirstName, ZIP, City, Street, StreetNo, DateOfBirth, Revenue)
- Select all customers from Wels who have a higher revenue than all customers from Linz

```
SELECT * FROM Customer
WHERE City='Wels' AND Revenue >= ALL
(SELECT Revenue FROM Customer WHERE City='Linz');
```

HTL LEONDING

Subqueries with ALL/ANY operator

Example

- Table Customer(CustNo, LastName, FirstName, ZIP, City, Street, StreetNo, DateOfBirth, Revenue)
- Select all customers who's birthday is at the same day as the one of the oldest customer

```
SELECT * FROM Customer
WHERE DateOfBirth <= ALL
(SELECT DateOfBirth FROM Customer);
```

HTL LEONDING

Subqueries with EXISTS operator

- Syntax: [**NOT**] **EXISTS** (subquery))
- Evaluates to TRUE if at least one row is returned by the subquery, otherwise FALSE
- Never returns UNKNOWN [Oracle]

Subqueries with EXISTS operator

Example

- Select all customers who placed at least one order

```
SELECT * FROM Customer c
WHERE EXISTS
(SELECT * FROM Order o
WHERE o.CustNo = c.CustNo);
```

HTL LEONDING

NULL comparison

- Syntax: expr1 IS [**NOT**] **NULL**
- When comparing with NULL neither = nor <> can be used
- Explicitly evaluating IS NULL or IS NOT NULL prevents getting UNKNOWN values [Oracle]
  - **WHERE** Name <> 'Knuth' vs.
  - **WHERE** Name IS **NOT NULL AND** Name <> 'Knuth'

HTL LEONDING

NULL comparison

Value a	Condition	Evaluates to
10	a IS NULL	FALSE
10	a IS NOT NULL	TRUE
NULL	a IS NULL	TRUE
NULL	a IS NOT NULL	FALSE
10	a = NULL	UNKNOWN
10	a != NULL	UNKNOWN
NULL	a = NULL	UNKNOWN
NULL	a != NULL	UNKNOWN
NULL	a = 10	UNKNOWN
NULL	a != 10	UNKNOWN

HTL LEONDING

ORDER BY clause

- Syntax: **ORDER BY** expr [**ASC**|**DESC**] [, expr [**ASC**|**DESC**], ...]
- Order of rows in a result set is usually not guaranteed (sets vs. (ordered) lists)
- ASC order is the default $\Rightarrow$ **ORDER BY** Quantity $=$ **ORDER BY** Quantity **ASC**

ORDER BY clause

- If multiple ORDER BY expressions are present ordering is applied in order of appearance in the query
- ASC|DESC can differ for each expression
- NULL values are placed at the end when ordering ascending and at the beginning when ordering descending
- Column name, alias and column index can be used to identify the column

HTL LEONDING

ORDER BY clause

Example

- Table Customer(CustNo, LastName, FirstName, ZIP, City, Street, StreetNo, DateOfBirth, Revenue)
- Select Lastname, Firstname, Revenue. Order by Revenue descending, then by Lastname ascending, then by Firstname ascending

```
SELECT LastName, FirstName fn, Revenue r FROM Customer
ORDER BY r DESC, LastName ASC, fn;
```

HTL LEONDING

GROUP BY clause

- Syntax: **GROUP BY** col1 [, col2, ...]
- Groups rows in the result set based on similar properties
- Usually used in conjunction with statistical functions $\Rightarrow$ those are evaluated for each group independently
- Only those columns used in the GROUP BY clause may appear in the SELECT clause
  - With the exception of using COUNT(*)
  - Example: **SELECT** LastName, **COUNT**(*)**FROM** Customer **GROUP BY** LastName

HTL LEONDING

GROUP BY clause

Example

- Table Order(OrderNo, CustNo, Amount, Date)
- Select the number of orders, the total revenue and the average order amount (rounded to two decimal places) per calendar year. Sort the result descending based on the calendar year.

```
SELECT TO_CHAR(Date, 'YYYY') Year, COUNT(*) Count,
SUM(Amount) "Total Revenue",
ROUND(AVG(Amount),2) "Avg. Revenue" FROM Order
GROUP BY TO_CHAR(Date, 'YYYY') ORDER BY Year DESC;
```

HTL LEONDING

HAVING clause

- Syntax: **HAVING** condition[s]
- Defines conditions for the filtered and grouped result set ⇒ 'second level WHERE'
  - Contrary to WHERE the conditions can contain statistical function
- Only those columns used in the GROUP BY clause or a statistical function may appear in the HAVING clause
- Example: **SELECT** LastName, **COUNT**(∗)**FROM** Customer **GROUP BY** LastName **HAVING COUNT**(∗)> 10

HTL LEONDING

HAVING clause

Example

- Table Order(OrderNo, CustNo, Amount, Date)
- Select the CustNo and number of orders (for the year 2014) of all customers who had at least two orders in 2014. Sort the result descending based on the order count.

```
SELECT CustNo, COUNT(*) "Order Count",
FROM Order WHERE TO_CHAR(Date, 'YYYY') = '2014'
GROUP BY CustNo HAVING COUNT(*) >= 2
ORDER BY COUNT(*) DESC;
```

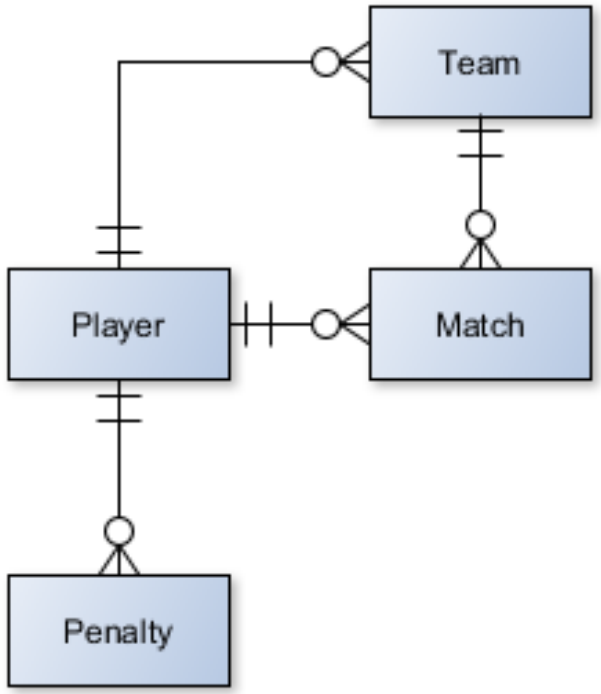
HTL LEONDING

The Tennis Club

- The club has amateur and professional players as members
- Professional players play in teams against other clubs
- Every professional player has a unique league id
- The club has several teams which participate in the cup
- Every team has a captain
- Members of a team change. So for each match it is necessary to log which player started for which team and which score was reached
- A player can get a penalty from the league for unfair behavior

HTL LEONDING

The Tennis Club



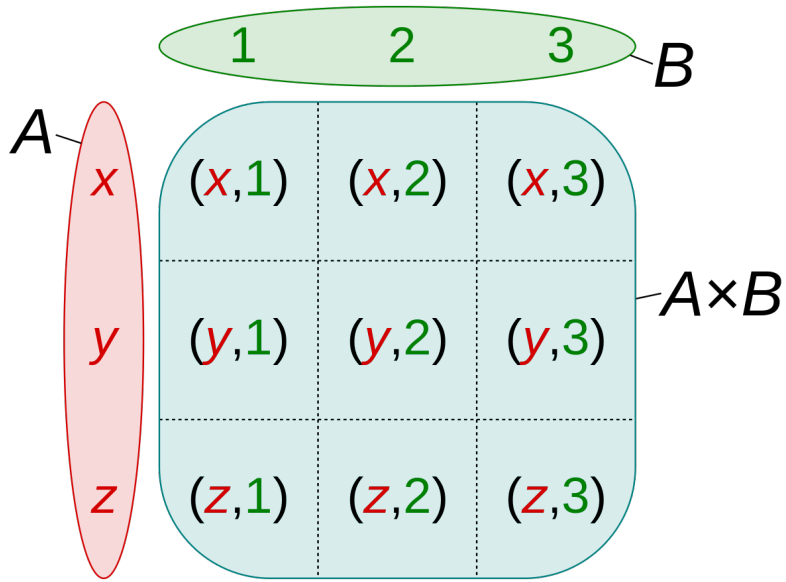
HTL LEONDING

The Tennis Club

- Tables:
  - Players (PlayerNo, Name, LeagueNo, YearOfBirth, . . .)
  - Teams (TeamNo, Name, PlayerNo, Division)
  - Matches (MatchNo, TeamNo, PlayerNo, SetsWon, SetsLost)
  - Penalties (PenaltyNo, PlayerNo, PenaltyDate, Amount)

Basic Join

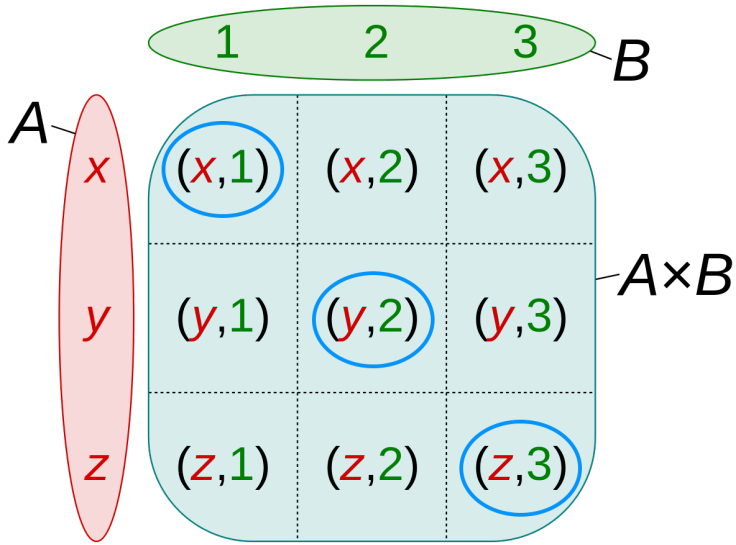
- FROM clause contains (at least) two tables
- **SELECT** ∗ **FROM** A, B;
- The result is a cartesian product



HTL LEONDING

Inner Equijoin

- Adding Join Columns creates what is usually called an 'inner join'
- **SELECT** ∗ **FROM** A, B **WHERE** A.FK = B.PK;
- This is the most common join for tables with foreign key relationships



HTL LEONDING

Inner Equijoin

Example

Players	
PlayerNo	**Name**
2	Everett
6	Parmenter
44	Baker

Penalties			
PenaltyNo	**PlayerNo**	**Date**	**Amount**
1	6	2000	100
2	44	2001	75
5	44	2003	25

Inner Equijoin

Example

```
SELECT pl.PlayerNo, pl.Name, pe.Amount
FROM Players pl, Penalties pe
WHERE pl.PlayerNo = pe.PlayerNo;
```

PlayerNo	Name	Amount
6	Parmenter	100
44	Baker	25
44	Baker	75

Equijoin

- Two types:
  - inner equijoin
  - outer equijoin
- Previous example was an inner equijoin $\Rightarrow$ only those players with at least one penalty are in the result set
- An outer equijoin is requested by adding '(+)' to the condition

Outer Equijoin

Example

```
SELECT pl.PlayerNo, pl.Name, pe.Amount FROM Players pl,
Penalties pe WHERE pl.PlayerNo = pe.PlayerNo (+);
```

PlayerNo	Name	Amount
6	Parmenter	100
44	Baker	25
44	Baker	75
2	Everett	

HTL LEONDING

Equijoin

Example

- Tables:
  - Players (PlayerNo, Name, Initials, ...)
  - Teams (TeamNo, PlayerNo, Division)
- Select for each team the TeamNo, the Name of the captain and the Division

```
SELECT t.TeamNo, p.Name, t.Division
FROM Players p, Teams t
WHERE p.PlayerNo = t.PlayerNo;
```

HTL LEONDING

Equijoin

Example

- Tables:
  - Players (PlayerNo, Name, Initials, ...)
  - Matches (MatchNo, PlayerNo, SetsWon, SetsLost)
- Select for each match the MatchNo, the name of the player and the won and lost sets. Sort the result by player name.

```
SELECT m.MatchNo, p.Name, m.SetsWon Won, m.SetsLost Lost
FROM Players p, Matches m WHERE p.PlayerNo = m.PlayerNo
ORDER BY p.Name;
```

HTL LEONDING

Equijoin

Example

- Table Penalties (PenaltyNo, PlayerNo, PenaltyDate, Amount)
- Select all PlayerNo and Names who ever got a penalty together with their average penalty (round to two decimal places).

```
SELECT pl.PlayerNo, pl.Name, ROUND(AVG(pe.Amount),2)
FROM Players pl, Penalties pe WHERE pl.PlayerNo = pe.PlayerNo
GROUP BY pl.PlayerNo, pl.Name;
```

HTL LEONDING

Equijoin

Example

- Select all PlayerNo and Names together with their average penalty (round to two decimal places). If a player hasn't ever got a penalty use 0 for the amount.

```
SELECT pl.PlayerNo, pl.Name, ROUND(AVG(NVL(pe.Amount,0)),2)
FROM Players pl, Penalties pe
WHERE pl.PlayerNo = pe.PlayerNo (+)
GROUP BY pl.PlayerNo, pl.Name;
```

HTL LEONDING

SQL Standard Joins

- Starting with version 9i Oracle supports joins according to the SQL:1999 Standard:
  - Cross Join
  - Natural Join
  - Equijoins with USING clause
  - Outer Joins (full, left, right)
- Those have an easier syntax with a separate join condition

Cross Join

- Syntax: **FROM** table1 **CROSS JOIN** table2
- Returns the cartesian product (cf. Oracle Equijoin without join columns)
- Example: **SELECT** pl.PlayerNo, pl.Name, pe.Amount **FROM** Players pl **CROSS JOIN** Penalties pe;

Natural Join

- Syntax: **FROM** table1 **NATURAL JOIN** table2
- Based on columns with the same name
  - Join columns (= those with the same name) have to have the same data types
  - Shared (join) columns are returned only once in the result set
    ⇒ Aliases cannot be used
- Example: **SELECT** PlayerNo, Name, Amount **FROM** Players **NATURAL JOIN** Penalties;

HTL LEONDING

Natural Join

Example

- Tables:
  - Players (PlayerNo, Name, Initials, ...)
  - Matches (MatchNo, PlayerNo, SetsWon, SetsLost)
- Select for each match the MatchNo, the player's Name and the won and lost sets. Sort the result descending based on the difference between won and lost sets.

```
SELECT PlayerNo, pl.Name, SetsWon, SetsLost FROM Players pl
NATURAL JOIN Matches ORDER BY (SetsWon−SetsLost) DESC;
```

HTL LEONDING

Equijoin with USING clause

- Syntax: **FROM** table1 **JOIN** table2 **USING** (col)
- Based on columns with the same name
- Similar to NATURAL JOIN but the join columns can be chosen
- Example: **SELECT** PlayerNo, Name, Amount **FROM** Players **JOIN** Penalties **USING**(PlayerNo);

Equijoin with USING clause

Example

- Tables:
  - Players (PlayerNo, Name, Initials, ...)
  - Teams (TeamNo, PlayerNo, Division)
- Select for each team the TeamNo, the captain's Name and the Division.

```
SELECT TeamNo, pl.Name, Division
FROM Teams JOIN Players pl USING(PlayerNo);
```

HTL LEONDING

Join with ON clause

- Syntax: **FROM** table1 **JOIN** table2 **ON** (condition)
- Flexible and easy to control:
  - Aliases for columns can be used
  - Subqueries can be used
  - Logical operators can be used
- Example: **SELECT** pl.PlayerNo, pl.Name, pe.Amount **FROM** Players pl **JOIN** Penalties pe **ON**(pl.PlayerNo=pe.PlayerNo) **WHERE** pl.PlayerNo=44;

HTL LEONDING

Join with ON clause

- It is even possible to join multiple tables:

```
SELECT d.DepartmentName, l.City, c.CountryName
FROM Departments d
JOIN Locations l ON (d.LocationId=l.LocationId)
JOIN Countries c ON (l.CountryId=c.CountryId)
WHERE c.RegionId=1;
```

Join with ON clause

Example

- Tables: Players (PlayerNo, Name, Initials, ...), Teams (TeamNo, PlayerNo, Division), Matches (MatchNo, TeamNo, SetsWon, SetsLost)
- Select for each match the MatchNo, the team's Division, the player's Name and the won and lost sets.

```
SELECT m.MatchNo, t.Division, p.Name, m.SetsWon, m.SetsLost
FROM Matches m JOIN Teams t ON (m.TeamNo=t.TeamNo)
JOIN Players p ON (p.PlayerNo=t.PlayerNo);
```

HTL LEONDING

Outer Join

- Syntax: **FROM** table1 (**LEFT**|**RIGHT**|**FULL**)**OUTER JOIN** table2 **ON** (condition)
- Comparable to the '(+)' Syntax in Oracle, but allows for better control
- Three Variants:
  - LEFT: Include rows from the left hand side table even if no match
  - RIGHT: Include rows from the right hand side table even if no match
  - FULL: Include rows from both tables even if no match

Left Outer Join

```
SELECT p.PlayerNo, p.Name, t.PlayerNo, t.TeamNo FROM
Players p LEFT OUTER JOIN Teams t ON (p.PlayerNo=t.PlayerNo)
```

Players		Teams	
PlayerNo	**Name**	**PlayerNo**	**TeamNo**
2	Everett		
6	Parmenter	6	1
27	Collins	27	2
44	Baker		
		200	3

HTL LEONDING

Right Outer Join

```
SELECT p.PlayerNo, p.Name, t.PlayerNo, t.TeamNo FROM
Players p RIGHT OUTER JOIN Teams t ON (p.PlayerNo=t.PlayerNo)
```

Players		**Teams**	
PlayerNo	**Name**	**PlayerNo**	**TeamNo**
2	Everett		
6	Parmenter	6	1
27	Collins	27	2
44	Baker		
		200	3

HTL LEONDING

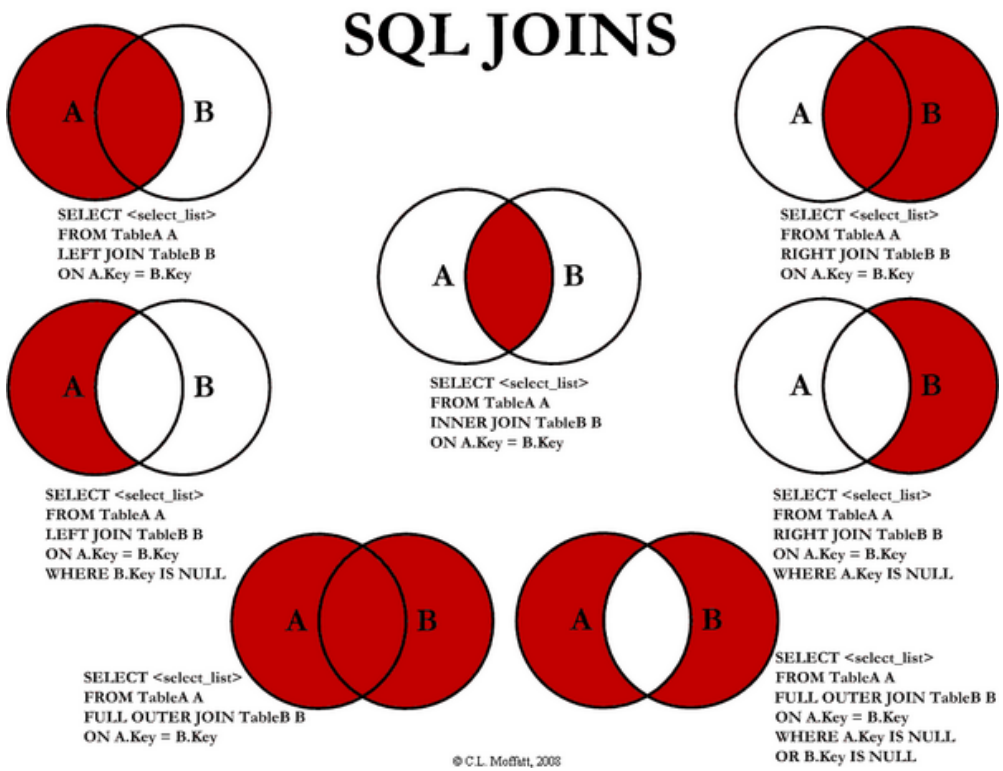
Full Outer Join

```
SELECT p.PlayerNo, p.Name, t.PlayerNo, t.TeamNo FROM
Players p FULL OUTER JOIN Teams t ON (p.PlayerNo=t.PlayerNo)
```

Players		**Teams**	
PlayerNo	**Name**	**PlayerNo**	**TeamNo**
2	Everett		
6	Parmenter	6	1
27	Collins	27	2
44	Baker		
		200	3

HTL LEONDING

Overview Joins



HTL LEONDING

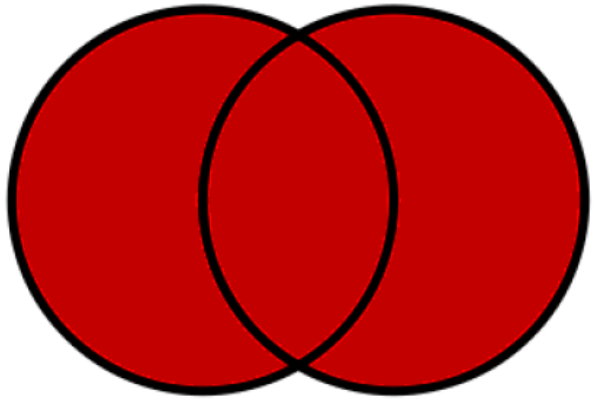
Set Theory

- SQL queries return sets ⇒ sets can be merged
- Conditions:
  - Number of columns in the result has to be equal
  - Data types of the columns in the result have to be equal (keep column order in mind)
- Three variants:
  - UNION
  - INTERSECT
  - MINUS

HTL LEONDING

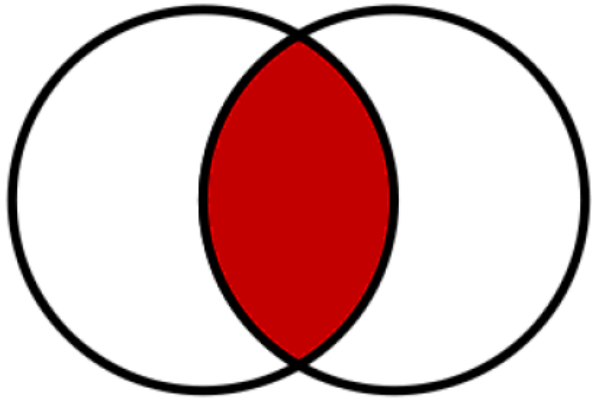
UNION

- Syntax: stmt1 **UNION** [**ALL**] stmt2
- The result set contains all rows of both statements
  - If the modifier ALL is added duplicate (=identical) rows are not removed



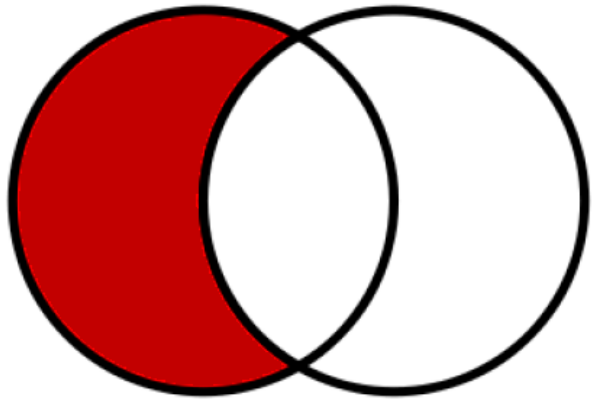
INTERSECT

- Syntax: stmt1 **INTERSECT** stmt2
- The result set contains the intersection of both statements
- Alternatives: EXISTS, IN



MINUS

- Syntax: stmt1 MINUS stmt2
- The result set contains the difference of both statements (rows are contained in the first but not in the second set)
- Alternatives: NOT EXISTS, NOT IN



HTL LEONDING

UNION

Example

- Select all Players and their penalties.

```
SELECT Name, Initials, Amount FROM Players
NATURAL JOIN Penalties
UNION
SELECT Name, Initials, 0 FROM Players
WHERE NOT EXISTS
(SELECT * FROM Penalties WHERE PlayerNo=Players.PlayerNo)
ORDER BY Amount DESC;
```

UNION

Example Result

	NAME	INITIALS	AMOUNT
1	Collins	DD	100
2	Parmenter	R	100
3	Baker	E	75
4	Collins	DD	75
5	Moorman	D	50
6	Baker	E	30
7	Baker	E	25
8	Newcastle	B	25
9	Bailey	IP	0
10	Bishop	D	0
11	Brown	M	0
12	Collins	C	0
13	Everett	R	0
14	Hope	PK	0
15	Miller	P	0
16	Parmenter	P	0
17	Wise	GWS	0

HTL LEONDING

INTERSECT

Example

- Select all PlayerNos of captains.

```
SELECT PlayerNo FROM Penalties
INTERSECT
SELECT PlayerNo From Teams;
```

INTERSECT

Example Result

	PLAYERNO
1	6
2	27

MINUS

Example

- Select all PlayerNos of players who have never competed in a league match.

```
SELECT PlayerNo FROM Players
MINUS
SELECT PlayerNo From Matches;
```

HTL LEONDING

MINUS

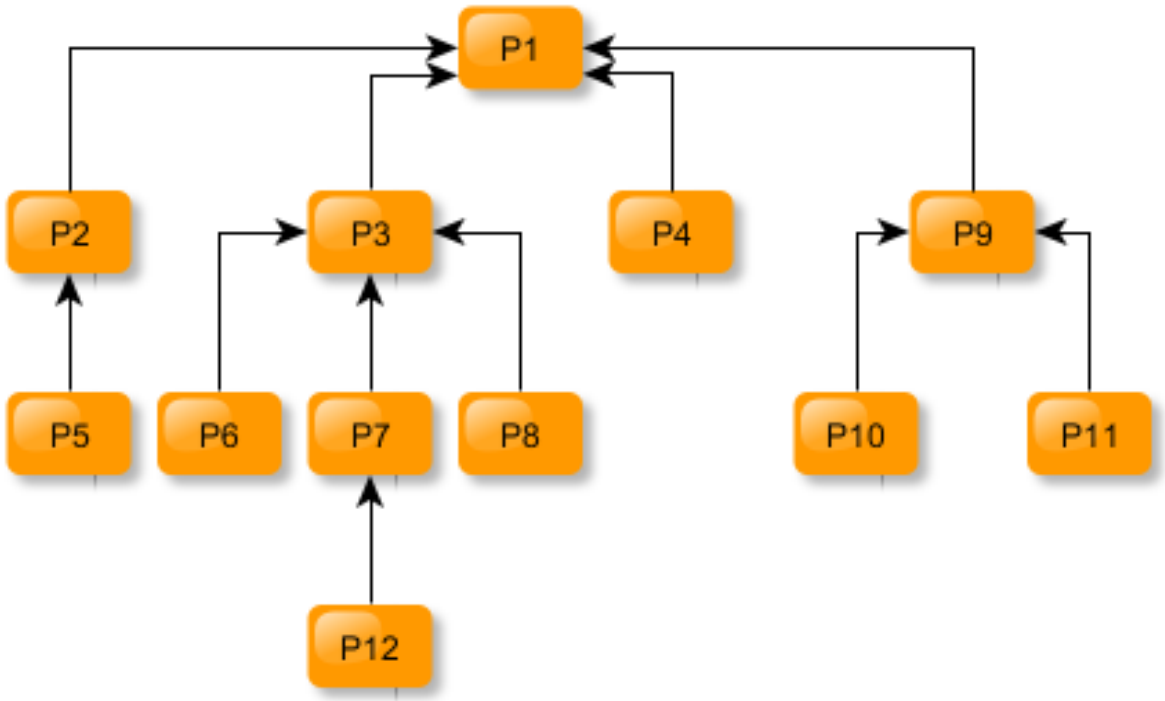
Example Result

	PLAY...
1	7
2	28
3	39
4	95
5	100

HTL LEONDING

Sample Data

- Parts list of a production company



HTL LEONDING

Sample Data

```
CREATE TABLE Parts(
        Sub varchar2(3) NOT NULL,
        Super varchar2(3),
        Price number(7,2) DEFAULT 0
);
```

Sub	Super	Price
P1		130
P2	P1	15
P3	P1	65
P4	P1	20
P9	P1	45
P5	P2	10
P6	P3	10
P7	P3	20
P8	P3	25
P12	P7	10
P10	P9	12
P11	P9	21

HTL LEONDING

CONNECT BY

- Syntax: **CONNECT BY** [**PRIOR**] condition [START WITH condition] [**ORDER** SIBLINGS **BY column**]
- Exists only in Oracle (SQL Standard: WITH clause)
- Clauses:
  - PRIOR: placed left or right of the equality sign denotes the subordinated column
  - START WITH: defines the starting point of the recursive query
  - ORDER BY SIBLINGS: enforces ordering on the tuple of the same level
  - LEVEL: pseudo column indicating the hierarchy level

CONNECT BY

Example

- Select all parts (down to the lowest level) component P3 consists of.

```
SELECT Sub, Super FROM Parts
CONNECT BY PRIOR Sub=Super
START WITH Sub='P3';
```

HTL LEONDING

CONNECT BY – Condition Placement

```
SELECT Sub, Super FROM Parts
WHERE sub != 'P7'
CONNECT BY PRIOR Sub=Super
START WITH Sub='P3';
```

Sub	Super
P3	P1
P6	P3
P12	P7
P8	P3

HTL LEONDING

CONNECT BY – Condition Placement

```
SELECT Sub, Super FROM Parts
CONNECT BY PRIOR Sub=Super
        AND sub != 'P7'
START WITH Sub='P3';
```

Sub	Super
P3	P1
P6	P3
P8	P3

CONNECT BY – LPAD

- Syntax: LPAD(str1, paddedLength [, padStr])
- Can be used to display hierarchy levels with indentation

```
SELECT LEVEL, LPAD(' ', 2*LEVEL-1)
       || Sub
FROM Parts
CONNECT BY PRIOR Sub=Super
START WITH Sub='P3';
```

LEVEL	Part
1	P3
2	P6
2	P7
3	P12
2	P8

HTL LEONDING

CONNECT BY – CONNECT_BY_ROOT

- Modifier for resolving the root element (pay attention to the starting point)

```
SELECT Sub, CONNECT_BY_ROOT
      Sub "Root"
FROM Parts START WITH Super='P1'
CONNECT BY PRIOR Sub=Super;
```

SUB	Root
P2	P2
P5	P2
P3	P3
P6	P3
P7	P3
P12	P3
P8	P3
P4	P4
P9	P9
P10	P9
P11	P9

HTL LEONDING

CONNECT BY – NOCYCLE

- Hierarchies are basically trees
- Recursive cycles in trees are not allowed ⇒ leads to a DB error
- The NOCYCLE clause can be used to suppress the recursion and prevent the issue

HTL LEONDING

CONNECT BY – NOCYCLE

- Precondition: P12 is set as super element of P1

```
SELECT LPAD(' ', 2*LEVEL-1) || Sub,
       Super
FROM Parts START WITH Sub='P3'
CONNECT BY NOCYCLE PRIOR Sub=
       Super;
```

LPAD('',2*LEVEL-1)||SUB	SUPER
P3	P1
P6	P3
P7	P3
P12	P7
P1	P12
P2	P1
P5	P2
P4	P1
P9	P1
P10	P9
P11	P9
P8	P3

HTL LEONDING

CONNECT BY – CONNECT_BY_ISCYCLE

- Shows if and where a cycle exists

```
SELECT Sub, Super,
    CONNECT_BY_ISCYCLE
FROM Parts START WITH Sub='P1'
CONNECT BY NOCYCLE PRIOR Sub=
    Super;
```

SUB	SUPER	CONNECT_BY_IS...
P1	P12	0
P2	P1	0
P5	P2	0
P3	P1	0
P6	P3	0
P7	P3	0
P12	P7	1
P8	P3	0
P4	P1	0
P9	P1	0
P10	P9	0
P11	P9	0

HTL LEONDING

CONNECT BY – CONNECT_BY_ISLEAF

- Shows if an element is a leaf node in the tree

```
SELECT Sub, Super,
       CONNECT_BY_ISLEAF
FROM Parts START WITH Sub='P1'
CONNECT BY NOCYCLE PRIOR Sub=
       Super;
```

SUB	SUPER	Leaf
P1	P12	0
P2	P1	0
P5	P2	1
P3	P1	0
P6	P3	1
P7	P3	0
P12	P7	1
P8	P3	1
P4	P1	1
P9	P1	0
P10	P9	1
P11	P9	1

HTL LEONDING

CONNECT BY – SYS_CONNECT_BY_PATH

- Shows the path from the root to the sub node
- Precondition: P12 is no longer super for P1

```
SELECT SYS_CONNECT_BY_PATH(
      Sub, ' > ') "Path"
FROM Parts START WITH Super IS
      NULL
CONNECT BY PRIOR Sub=Super;
```

Path
> P1
> P1 > P2
> P1 > P2 > P5
> P1 > P3
> P1 > P3 > P6
> P1 > P3 > P7
> P1 > P3 > P7 > P12
> P1 > P3 > P8
> P1 > P4
> P1 > P9
> P1 > P9 > P10
> P1 > P9 > P11

HTL LEONDING

CONNECT BY – ORDER SIBLINGS BY

```
SELECT LPAD(' ', LEVEL)||Sub "Part",
       Super
FROM Parts CONNECT BY PRIOR Sub
       =Super
START WITH Super IS NULL
ORDER BY Sub DESC;
```

Part	SUPER
P9	P1
P8	P3
P7	P3
P6	P3
P5	P2
P4	P1
P3	P1
P2	P1
P12	P7
P11	P9
P10	P9
P1	(null)

HTL LEONDING

CONNECT BY – ORDER SIBLINGS BY

```
SELECT LPAD(' ', LEVEL)||Sub "Part",
       Super
FROM Parts CONNECT BY PRIOR Sub
       =Super
START WITH Super IS NULL
ORDER SIBLINGS BY Sub DESC;
```

Part	SUPER
P1	(null)
P9	P1
P11	P9
P10	P9
P4	P1
P3	P1
P8	P3
P7	P3
P12	P7
P6	P3
P2	P1
P5	P2

HTL LEONDING

WITH clause

- SQL Standard way of performing hierarchy queries
- Supported in Oracle since version 11g
- Utilizes recursive calls of inline views

WITH clause

```
WITH PartsRec(Sub, Super) AS
(
SELECT Sub, Super FROM Parts
WHERE Super IS NULL
UNION ALL
        (
        SELECT p.Sub, p.Super
        FROM PartsRec pr, Parts p
        WHERE pr.Sub = p.Super
        )
)
SELECT Sub, Super FROM PartsRec;
```

SUB	SUPER
P1	(null)
P2	P1
P3	P1
P4	P1
P9	P1
P5	P2
P6	P3
P7	P3
P8	P3
P10	P9
P11	P9
P12	P7

HTL LEONDING

WITH clause – Ordering

- By default sorts on the same hierarchy level first (BREADTH)
- Can be overwritten to sort down (DEPTH) first

WITH clause – Ordering

```
WITH PartsRec(Sub, Super) AS
(
SELECT Sub, Super FROM Parts
WHERE Super IS NULL
UNION ALL
        (
        SELECT p.Sub, p.Super
        FROM PartsRec pr, Parts p
        WHERE pr.Sub = p.Super
        )
)
SEARCH DEPTH FIRST BY Sub SET sort1
SELECT Sub, Super FROM PartsRec
ORDER BY sort1;
```

SUB	SUPER
P1	(null)
P2	P1
P5	P2
P3	P1
P6	P3
P7	P3
P12	P7
P8	P3
P4	P1
P9	P1
P10	P9
P11	P9

HTL LEONDING

WITH clause – Advanced features

- Several features provided by the Oracle approach are not directly available using the WITH clause
  - LEVEL
  - SYS_CONNECT_BY_PATH
  - CONNECT_BY_ISCYCLE
- Can be substituted by other constructs

WITH clause – 'LEVEL'

- Using a counter (pseudo column LEVEL not available)

```
WITH PartsRec(Lvl, Sub, Super) AS
(
SELECT 1 "Lvl", Sub, Super FROM Parts
WHERE Super IS NULL
UNION ALL
        (
        SELECT Lvl+1 "Lvl", p.Sub, p.Super
        FROM PartsRec pr, Parts p
        WHERE pr.Sub = p.Super
        )
)
SELECT Lvl, Sub, Super FROM PartsRec;
```

LVL	SUB	SUPER
1	P1	(null)
2	P2	P1
2	P3	P1
2	P4	P1
2	P9	P1
3	P5	P2
3	P6	P3
3	P7	P3
3	P8	P3
3	P10	P9
3	P11	P9
4	P12	P7

HTL LEONDING

WITH clause – 'SYS_CONNECT_BY_PATH'

```
WITH PartsRec(Lvl, Path, Sub, Super) AS
(
SELECT 1 "Lvl", '/'||Sub "Path", Sub, Super
      FROM Parts
WHERE Super IS NULL
UNION ALL
         (
         SELECT Lvl+1 "Lvl", SUBSTR(Path||'/'||
               p.Sub,0.100) "Path", p.Sub, p.Super
         FROM PartsRec pr, Parts p
         WHERE pr.Sub = p.Super
         )
)
SELECT Lvl, Path, Sub, Super FROM PartsRec;
```

LVL	PATH	SUB	SUPER
1	/P1	P1	(null)
2	/P1/P2	P2	P1
2	/P1/P3	P3	P1
2	/P1/P4	P4	P1
2	/P1/P9	P9	P1
3	/P1/P2/P5	P5	P2
3	/P1/P3/P6	P6	P3
3	/P1/P3/P7	P7	P3
3	/P1/P3/P8	P8	P3
3	/P1/P9/P10	P10	P9
3	/P1/P9/P11	P11	P9
4	/P1/P3/P7/P12	P12	P7

HTL LEONDING

WITH clause – 'SYS_CONNECT_BY_ISCYCLE'

```
WITH PartsRec(Lvl, Sub, Super) AS
(
SELECT 1 "Lvl", Sub, Super FROM Parts
WHERE Super = 'P12'
UNION ALL
        (
        SELECT Lvl+1 "Lvl", p.Sub, p.Super
        FROM PartsRec pr, Parts p
        WHERE pr.Sub = p.Super
        )
)
CYCLE Sub SET IsCycle TO 1 DEFAULT 0
SELECT Lvl, Sub, Super, IsCycle
FROM PartsRec;
```

LVL	.	SUPER	ISCYCLE
1	P1	P12	0
2	P2	P1	0
2	P3	P1	0
2	P4	P1	0
2	P9	P1	0
3	P5	P2	0
3	P6	P3	0
3	P7	P3	0
3	P8	P3	0
3	P10	P9	0
3	P11	P9	0
4	P12	P7	0
5	P1	P12	1

HTL LEONDING