Databases

Cap. 6. SQL. Data projection, selection, ordering, join

Textbook: Ramakrishnan, Gehrke, “Database Management Systems”, McGraw Hill, 2003

2020 UPT

Conf.Dr. Dan Pescaru

Review: RA operators

1. Relational Algebra operators:
   - Projection (Π) – deletes unwanted columns from a relation R (discards duplicates)
   - Selection (σ) – selects a subset of rows from R
   - Cartesian Product ($\times$) – combines two relations R_1 and R_2, includes all pairs ($t_1 \in R_1, t_2 \in R_2$)
   - Join ($\bullet$) – $\sigma_{join\_condition}(R_1 \times R_2)$
   - Union ($\cup$) – tuples from R_1 and from R_2 (discards duplicates)
   - Intersect ($\cap$) – tuples in both R_1 and in R_2
   - Set difference (/) – tuples in R_1, but not in R_2

Example database (harbour)

- **Sailors table**

sid	name	rank	age
22	John	7	45
31	Horace	1	33
58	Andrei	8	54
71	John	9	55

- **Boats Table**

bid	name	color
101	Cleo	Blue
102	Gazelle	Red
103	Poseidon	Green

- **Reserves Table**

sid	bid	date
58	101	2014/10/03
22	102	2014/10/18
58	103	2014/11/23

SQL - Projection

1. Remove unwanted columns from result:

   SELECT [DISTINCT] prj_list
   FROM table;

   - prj_list: comma separated list of fields or * for 1:1 projection
   - DISTINCT: by default duplicates are not removed!
     - Why? When necessary?
     - Performance issues?

2. E.g.: SELECT sid, name FROM Sailors;

   SELECT DISTINCT rank FROM Sailors;

Projection – renaming fields

1. Aliases are useful for avoiding ambiguity:
   - for calculated fields
     - can contain library function call (set of functions is system dependent) + aggregation functions
   - when prj_list includes fields with same name from different tables (e.g. join)

     SELECT [DISTINCT] $field_1$ AS $alias_1$, $field_2$ AS $alias_2$,..., exp_1 AS $alias_{e1}$, ...
     FROM table;

   - E.g.: SELECT bid AS Boat_ID,
     year(date) AS Year_of_Reserve
     FROM Reserves; // -- MySQL syntax

Oracle SQL - Projection

1. Oracle syntax:

   SELECT [hint] [ALL | DISTINCT | UNIQUE]
   field$_1$ AS alias$_1$, …, exp$_1$ AS exp_name$_1$, …
   FROM table;

   - fields list could contains ROWNUM for result records numbering
   - hint – hints for query execution optimizer

2. E.g.: SELECT ROWNUM AS No, bid, name AS Boat
   FROM boats;

MySQL SQL - Projection

1. MySQL syntax:

```
SELECT [ALL | DISTINCT | DISTINCTROW]
        field1 [AS] alias1, exp1 [AS] exp_name1, ...
        FROM table;
```

- For numbering the result records:

```
SET @r=0;
SELECT @r:=@r+1 AS no, bid, name
        FROM boats;
```

SQL - Selection

1. Selects a subset of rows based on a condition:

   SELECT prj_list FROM table

   WHERE condition;

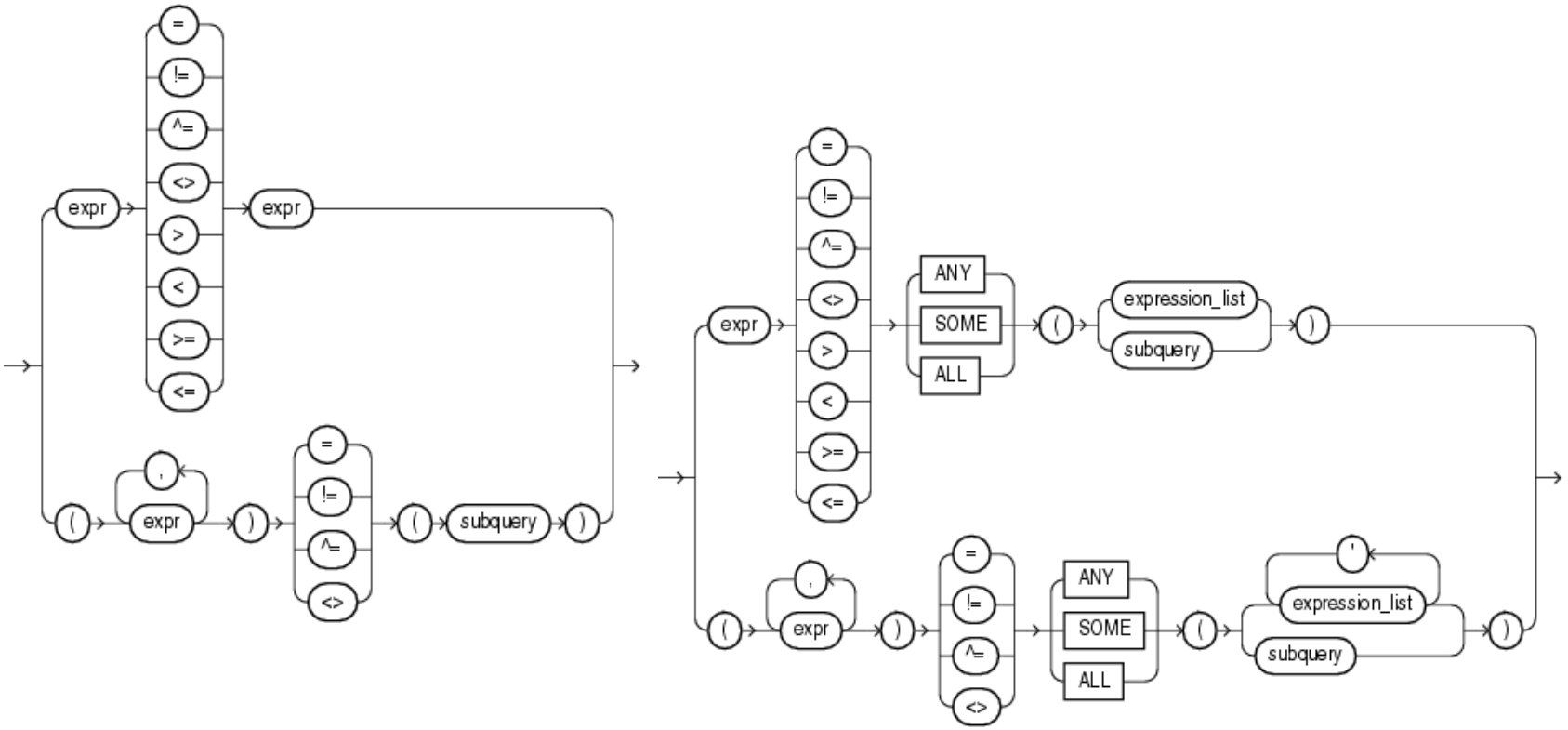
   - Set prj_list to * for *pure* selection (no projection)
   - What about duplicates?
   - Selection condition:
     - Logical condition includes AND, OR, NOT operators and SQL functions (system specific)

2. E.g.: SELECT sid, name FROM Sailors

   WHERE age>30;

Oracle SQL - Selection

1. Comparison conditions



Oracle SQL – selection conditions

1. E.g.

SELECT * FROM Sailors

- WHERE rank=7;
- WHERE rank=7 AND age>=25;
- WHERE rank IN (1,2,7);
- WHERE rank = ANY (2,5,6);
- WHERE rank NOT IN (1,3);
- WHERE age > ANY (18, 13, 42, 27);
- WHERE age > ALL (18, 13, 42, 27);
- WHERE name LIKE '_o%'

Oracle SQL – functions (I)

1. Functions
   - ABS(*n*) - return the absolute value
   - CONCAT(*str1*,*str2*,…) - concatenated string
   - ROUND(*n*) - rounds a number
   - TRUNC(*n*) - truncates a number
   - LOWER(*str*) - the argument in lowercase
   - SUBSTR(*substr*, *str*,[*s_pos*]) – position of the first occurrence of substring
   - TRIM(*str*) - remove leading and trailing spaces
   - LENGTH(*str*) - the length of a string

Oracle SQL – functions (II)

1. Functions
   - CURRENT_DATE - the current date

```
SELECT CURRENT_DATE FROM DUAL;
```

   - EXTRACT(*spec*, *d*) – extract a part of date (spec ∈{year, month, day, hour, minute, second, timezone_hour, timezone_minute})
   - NVL(*expr1*,*expr2*) returns *expr2* if *expr1* null

```
SELECT NVL(name, 'unknown') AS sn FROM Sailors;
```

   - CASE WHEN *expr1* THEN *expr2* ELSE *expr3* END similar to C: *expr1* ? *expr2* : *expr3*

MySQL SQL – selection conditions

1. E.g.

```
SELECT * FROM Sailors
    - WHERE rank=7;
    - WHERE rank=7 AND age>=25;
    - WHERE rank IN (1,2,7);
    - WHERE rank BETWEEN 3 AND 7;
    - WHERE age IS NULL;
    - WHERE age <=> NULL; // NULL-safe equal
SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
SELECT 1 = 1, NULL = NULL, 1 = NULL;
```

MySQL SQL – functions (I)

1. Functions
   - ABS(*n*) - return the absolute value
   - CONCAT(*str1*,*str2*,…) - concatenated string
   - CURTIME() - return the current system time
   - DAY() - return the day of the month (0-31)
   - DAYOFYEAR() - the day of the year (1-366)
   - FORMAT(*n*, *decPlaces*) - format a number
   - IF(*exp1*,*exp2*,*exp3*) - similar to exp1?exp2:exp3
   - IFNULL(*exp1*,*exp2*) – if *exp1*=NULL return *exp2*
   - LEFT(*str*,*len*) - the leftmost characters
   - LENGTH(*str*) - the length of a string

MySQL SQL – functions (II)

1. Functions
   - LOCATE(*substr*, *str*,[*s_pos*]) – position of the first occurrence of substring
   - LOWER(*str*) - the argument in lowercase
   - MID(*str*,*pos*,*len*) - a substring starting from the specified position
   - MONTH(*d*) - the month from the date passed
   - NOW() - the current date and time
   - PASSWORD(*str*) - return not reversible password
   - RAND() - a random floating-point $\in$[0.0…1.0]
   - TRIM() - remove leading and trailing spaces

SQL selection example

- Retrieve the rank of all sailors that have the name in the first half of the alphabet. Classify the sailors in *juniors* and *seniors* if the age is under/over 31
  - MySQL solution:

```
SELECT CONCAT('The ', IF(age<31, 'junior',
    'senior'), ' sailor ', name, ' has the rank ',
    rank, '.') AS sailor, IFNULL(age,
    'unknown') AS age
    FROM Sailors
    WHERE LEFT(name,1)<'L';
```

SQL – Ordering the result

1. ORDER BY is used in SQL to sort the result set by one or more attributes
2. Ascending order (ASC) is default

   SELECT *column1*,*column2*,…

   FROM *table*

   ORDER BY $column_i$,$column_j$ [ASC|DESC];

3. E.g.: SELECT * FROM Sailors

   ORDER BY rank, age DESC;

   – what if NULL? (in Oracle: [NULLS FIRST|NULLS LAST])

SQL – Cartesian product

1. Not very common in real applications
2. Syntax:
   - Multiple tables in FROM (useful: aliases)

     SELECT $t_1.*$, $t_2.*$

     FROM *table*$_1$ t_1, *table*$_2$ t_2;
3. E.g.

   SELECT s.*, r.*

   FROM Sailors s, Reserves r;

SQL – JOIN

1. The SQL JOIN clause is used to combine rows from two or more tables, based on logical links (e.g. common fields) between them. Usually: t_1.PK=t_2.FK
2. Two syntax:
   - Implicit JOIN: multiple tables in FROM; the join condition embedded in WHERE

     SELECT t_1.*column*$_1$,…,t_2.*column*$_1$,…

     FROM *table*$_1$ t_1, *table*$_2$ t_2 …

     WHERE t_1.*column*$_i$=t_2.*column*$_j$ [AND select_conditions];
   - Explicit JOIN: tables and conditions in FROM

JOIN Types

1. INNER JOIN - returns all rows when there is at least one match in both tables. Implicit syntax imply inner join
2. LEFT (OUTER) JOIN - return all rows from the left table, and the matched rows from the right table. NULL for missing corresponding fields
3. RIGHT (OUTER) JOIN - return all rows from the right table, and the matched rows from the left table
4. FULL (OUTER) JOIN - return all rows when there is a match in at least one of the tables

INNER JOIN

1. INNER JOIN – explicit syntax

```
SELECT column_name(s)
    FROM table1 [INNER] JOIN table2
    ON table1.PK=table2.FK …
    [WHERE selection_condition…];
```

2. E.g.

```
SELECT s.name, s.rank, r.date, b.name AS boat
    FROM ((Sailors s INNER JOIN Reserves r ON
    s.sid=r.sid) INNER JOIN Boats b ON r.bid=b.bid)
    WHERE s.age>25;
```

LEFT JOIN

1. LEFT [OUTER] JOIN – syntax

```
SELECT column_name(s)
    FROM table1 LEFT [OUTER] JOIN table2
    ON table1.PK=table2.FK …
    [WHERE selection_condition…];
```

2. E.g.

```
SELECT s.name, s.rank, r.date, b.name AS boat
    FROM ((Sailors s LEFT JOIN Reserves r ON
    s.sid=r.sid) LEFT JOIN Boats b ON r.bid=b.bid)
    WHERE s.age>25;
```

RIGHT JOIN

1. RIGHT [OUTER] JOIN – syntax

```
SELECT column_name(s)
    FROM table1 RIGHT [OUTER] JOIN table2
    ON table1.PK=table2.FK …
    [WHERE selection_condition…];
```

2. E.g.

```
SELECT s.name, s.rank, r.date, b.name AS boat
    FROM ((Sailors s RIGHT JOIN Reserves r ON
    s.sid=r.sid) INNER JOIN Boats b ON
    r.bid=b.bid);
    // WHERE s.age>25; //contradiction - why ?
```

FULL JOIN

1. FULL [OUTER] JOIN – syntax

```
SELECT column_name(s)
    FROM table1 FULL [OUTER] JOIN table2
    ON table1.PK=table2.FK …
    [WHERE selection_condition…];
```

2. Note: not supported by all DBMS (e.g. MySQL)
3. E.g.

```
SELECT s.name, s.rank, r.date, b.name AS boat
    FROM ((Sailors s FULL JOIN Reserves r ON
    s.sid=r.sid) FULL JOIN Boats b ON r.bid=b.bid);
    // WHERE s.age>25; same?
```

SELF JOIN

1. Just one table involved (recursive link). Could by INNER, LEFT, RIGHT or FULL JOIN, but…
2. E.g.

Persons

ssn	name	ssn_father	ssn_mother
B012	John	B026	NULL
B026	Horace	NULL	G018
G114	Anna	B026	NULL
G018	Sara	NULL	NULL

```
SELECT p.name, f.name AS father, m.name AS mother
    FROM ((Persons p LEFT JOIN Persons f
        ON p.ssn_father=f.ssn) LEFT JOIN Persons m
        ON p.ssn_mother=m.ssn);
// Note:how about INNER+INNER and LEFT+RIGHT?
```

Oracle JOIN

1. Implements FULL OUTER JOIN
2. Defines an outer join operator (+) that can be included in WHERE condition to specify an LEFT/RIGHT JOIN instead an default INNER one. It can not be used for FULL JOIN.
3. E.g. For an LEFT JOIN

```
SELECT s.name, s.rank, r.date
    FROM Sailors s, Reserves r
    WHERE s.sid = r.sid(+);
```

Note: for RIGHT JOIN: s.sid(+) = r.sid

MySQL JOIN

1. Do not implement FULL OUTER JOIN. It can be simulated by an UNION between a LEFT JOIN and a RIGHT JOIN
2. USING can replace ON if PK and FK has same names
   - E.g. SELECT s.*,r.*
     FROM Sailors s INNER JOIN Reserves r
     USING (sid);
3. STRAIGHT_JOIN is similar to JOIN, except that the left table is always read before the right table. This can be used for some (few) cases for which the join optimizer puts the tables in the wrong order