8. SQL – Data Handling

8.1 Update, Deletion, Insertion and bulk load*

8.2 The query language SQL

8.2.1 Search predicates

8.2.2 Arithmetic expressions and functions in predicates

8.2.3 Different kinds of join

8.2.4 Output improvement

8.3 Advanced SQL

8.3.1 Subselects and Correlated subqueries

8.3.2 Quantified expressions, SOME, ANY

8.3.3 Grouping and Aggregation

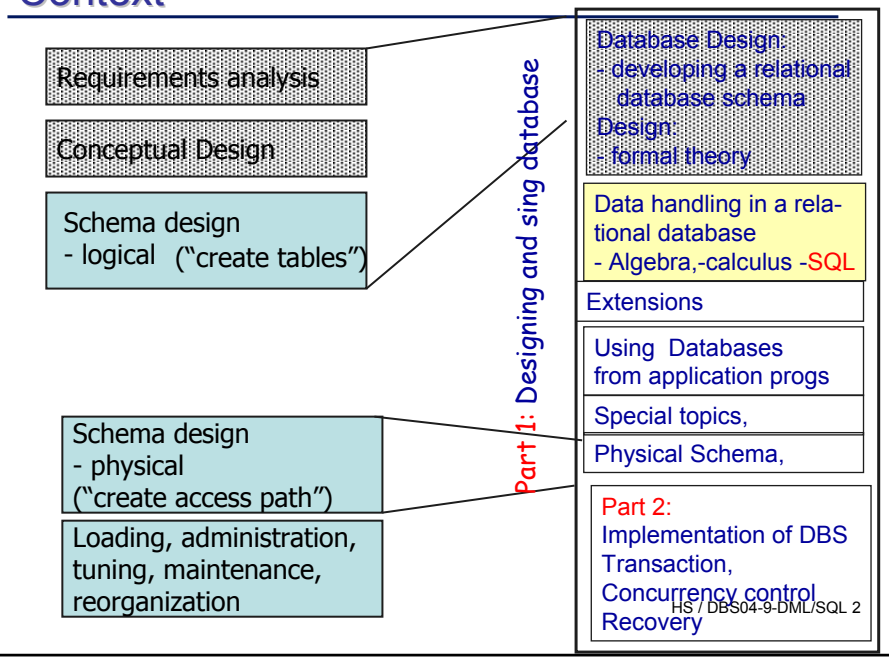
8.3.4 Transitive closure

8.3.5 Final remarks: NULLS, temporary relations and more

Lit.: Melton / Simon, Understanding SQP 1999, chap. 2,5,7; Kemper / Eickler chap 4, SQL chapter in any book on DBS

* useful for the projects

Context



SQL / DML: Overview

- Insert, update, delete data
- Query data
  - Interactively
  - Embedded in host language *important in applications!*
    next chapter
- Operations on views

8.1 SQL / DML: Update operations

- Delete, Insert, Update
  - The easiest way to ruin your company:

```
DELETE FROM Customer;
```

deletes all rows from `Customer` relation

  - In general, the rows to be deleted are specified by a (search) predicate:

```
DELETE FROM <tablename> WHERE <predicate>;
```

  - This is very different from deleting metadata:

```
DROP TABLE Customers;    DROP SCHEMA my_database;
```

SQL / DML: Update

– General form (simplified) for changing values :

```
UPDATE <tableName> SET
  <attr> = <value> [, <attr> = <value> ]0..*
   WHERE <searchPredicate>
```

- Note: without a predicate all rows will be changed
- Primary key predicate for update is very common

```
Update Customer SET email = 'me@acm.org'
   WHERE mem_No = 4711;     /* primary key */

Update Rental SET until_date = SYSDATE
  WHERE t_Id = 23 AND mem_no = 3315
  AND until_date IS NULL;

UPDATE Format SET extraCh = 1 ;
```


SQL / DML: Insertion

Insertion

– Complete form:

- Predefined order of values

```
INSERT INTO Customer
VALUES (011, 'Müller', 'Tina', NULL,NULL,NULL);
```

order of values as attributes in table definition

– Incomplete form with attribute and value list:

- Free order of values

```
INSERT INTO Customer
(name, mem_no) VALUES ('Müller', 012);
```

SQL / DML: Insert data

- Insertion using a query

```
INSERT INTO  Foo (select * FROM Tmp)
```

  Result set of query must have same type signature as table inserted to.

- Bulk insertion
  - large file of INSERT statements may be inefficient
  - insertion of large data sets by specific DB tools
    Postgres: COPY command to and from files (e.g. cvs)
    Oracle and others: bulk loader
  - not standardized

SQL / DML: bulk load

- Bulk load: inserting many data
- Example:

```
CREATE TABLE loadtest(
  name varchar(20),
  num number(10,2));
```

loadtest.dat

```
'XYZ' , 4
'YZX' , 5
'ZXY  , 6
```

- Oracle Syntax:

```
sqlldr <user>/<password> <controlfile>
       <logfile> <badfile> <datafile>
```

loadtest.ctl

```
load data
infile 'loadtest.dat'
badfile 'loadtest.bad'
discardfile 'loadtest.dis'
APPEND INTO table loadtest
 fields terminated by " , "
 optionally enclosed by " ' "
(name char, num integer external)
```

8.2 The Query Language SQL

SQL is relational complete

- Many additional query concepts
  - Advanced search predicates on strings
    e.g., find all movies starting with “star wars”
  - Arithmetic in expressions,
    e.g., number and percentage of tapes for each movie
  - Grouping and predicates over sets
    e.g., total revenue for each movie within the last year

SQL and RA (recap)

- Basic SQL ... FROM ... WHERE
  - Transformation rule: for every relational algebra expression with join, project, cartesian product and select operations there is an equivalent expression of the form:

$$\pi \ldots (\sigma_P (R_1 \times R_2 \times \ldots \times R_n))$$

projection ←
cartesion product ←
predicate ←

  - Simple SQL (Sequel) block:

```
SELECT  a,b,.....
FROM    R1 ,..., Rn
WHERE   < predicate P >
```

  - Not exactly equivalent to Relational Algebra
    Duplicates only eliminated explicitly by …

8.2.1 Simple SQL Search predicates

Simple search predicates

- Defined like Boolean predicates in RA and Calculus
- Some syntax extensions
  - <attribute> **BETWEEN <value1> AND <value2>**
  - <attribute> **IS [NOT] NULL**

```
SQL> SELECT *
  2  FROM Format
  3  WHERE extraCh BETWEEN 0 AND 0.5
  4   AND extraCh IS NOT NULL;

FORMA    EXTRACH
----- ----------
Beta           ,5
```


Simple SQL Search expressions

- Boolean Predicates

  Be careful with negation!

```
SQL>  SELECT title
  2   FROM Movie m
  3   WHERE NOT(title <> 'Casablanca'
  4         OR title <> 'A.I.');

 Result?
```

  - Predicates with conjunctive primitives

    a = <value> AND a = <differentValue>

    will never have a non-zero result

Remember…

All logical differences are big differences
(Wittgenstein)

Corollar:
All logic mistakes are big mistakes

not so big

Simple SQL Search expressions

- Simplified OR

```
SELECT title, director
FROM Movie
WHERE director IN ('Spielberg','Lubitsch',
                   'Kubrick');
```

  - Equivalent to <attr> = <val1> OR <attr> = <val2> OR ...
  - Special case of a relational constant:

```
SELECT title, director
FROM Movie
WHERE (cat, director)
   IN(('Anti-War','Spielberg'),
      ('Satire','Kubrick') );
```

8.2.2 Arithmetic and functions in search predicates

Extensions of RA - Arithmetic expressions

– May occur in simple predicates and target list

```
SELECT title, price_Day + 0.16* price_Day AS
                          "Price incl. Tax"
FROM Movie
WHERE 1.16*price_Day < 2;

Movie Title          Price incl. Tax
-------------------- ---------------
Matrix                            ,8
To be or not to be                ,8
```


Simple SQL String search expressions

- String expressions
  - Simple form of regular expressions: LIKE

LIKE patterns:
% : any sequence of characters
_ : exactly one character

```
SELECT title
FROM Movie
WHERE title LIKE '%be%'
  AND title LIKE '%not%';

Movie Title
--------------------
To be or not to be
```

- Regular expressions defined in SQL99*

  <string> SIMILAR TO <pattern>

* implemented in Postgres and others

Simple SQL Functions in Search expressions

- Functions
  - Expressions may contain functions
  - Many arithmetical and string built-in functions
  - User defined functions on user defined types (see below)

```
SELECT title
FROM Movie
WHERE SOUNDEX(director)
   =  SOUNDEX('Spilbak');

Movie Title
--------------------
Amistad
A.I.
Private Ryan
```


Simple SQL Search expressions

More expressions

Arithmetic expressions example

Constant	7.5, 3.
[qualifier.]columnname	T.format, fname
arithExpr op arithExp	3 + 4, price - 2*discount
function (aexpr)	Sqrt(xCoord*xCoord + yCoord * yCoord)

Character and Dateexpressions

cexpr '+' cexpr	'The winner is: ' || fname
function (cexpr)	SOUNDEX ('Meier')
	UPPER (fname)
	TRIM (TRAILING ' ' FROM ' Hello ')
	SUBSTRING(fname FROM 0 FOR 4)
Date value express.	SYSDATE – date_of_birth

Simple SQL Search expressions

- Date-Functions
  - Notoriously low level of standardization, differs heavily between systems

```
SELECT title, year
FROM Movie
WHERE MONTHS_BETWEEN(SYSDATE,year)> 120 ;

Movie Title          YEAR
-------------------- --------
To be or not to be   01.05.42
Der Förster vom Silb 01.05.54
erwald
```

  - Oracle's Date Model is flexible, but clumsy
  - Problem in general: Casting a time INTERVAL
    e.g. one year and five month to – say – seconds does not result in a well defined value (how many month with 30 | 31 | 28 days?)

8.2.3 Simple SQL: Joins

All SQL systems support mixture of search predicates and join conditions

```
SELECT t_Id,format, m.title, c.name
FROM Tape t, Movie m, Rental r, Customer c
WHERE m.director = 'Spielberg'
  AND m.m_Id = t.m_Id
  AND t.format = 'VHS'
  AND t.t_ID = r.tape_Id
  AND r.mem_No = c.mem_No ;
```

  - Note: use of tuple variables here used as aliases
  - Idea: separation of join condition from search predicate

Separation of Join and Selection clauses

```
SELECT t_Id,format, m.title, c.name
FROM Movie m JOIN Tape t  ON t.m_Id = m.m_ID
             JOIN Rental  ON t.t_id = Rental.tape_Id
             NATURAL JOIN customer
WHERE m.director = 'Spielberg'
  AND t.format = 'VHS';

t_Id FORMA     Movie Title          NAME
---------- ----- ------------    ----------
11           VHS Jurassic Park        Abel
```

SQL 99

– More syntactical differences ("OUTER JOIN", "NATURAL JOIN"), not supported by all systems

Evaluation of predicates

7 customers in DB, 1 rental row

```
SELECT t_Id,format, m.title, c.name
FROM   Movie m
       JOIN Tape t ON t.m_Id = m.m_ID
       JOIN Rental ON t.t_id = Rental.tape_Id
       NATURAL Right  OUTER JOIN Customer c
WHERE m.director = 'Spielberg'
 AND t.format = 'VHS';
```

why only one result row?

```
  T_ID FORMAT   TITLE              NAME
 ----- -----  ----------------   ------
    2   VHS     Jurassic Park       Abel
```

SQL / DML: Simple queries with joins

- Cross join (cross product)

enhanced SQL:1999

```
<tableName> CROSS JOIN <tableName>
```

- Natural inner join

```
<tableName> NATURAL [INNER] JOIN <tableName>
```

- Example:

```
SELECT *
FROM Rental NATURAL INNER JOIN Customer;
```


SQL / DML: Simple queries with joins

Inner equi-join with attribute list

enhanced SQL:1999

```
<tableName> [INNER] JOIN <tableName>
USING <attributList>
```

Subset of attributes in common

Example:

```
SELECT *
FROM Rental r JOIN Customer c
USING (mem_no);
```

SQL / DML: Simple queries with joins

All Customers who have rented at least one 'Entertainment' film

```
SELECT c.mem_no, c.name, c.first_name
FROM ((Customer c
          JOIN Rental r ON c.mem_no = r.mem_no)
          JOIN Tape t ON t.t_id = r.tape_id )
          JOIN Movie m ON t.m_id = m.m_id
WHERE m.cat='Entertainment';
```

```
SELECT c.mem_no, c. name, c.first_name
FROM Customer c, Rental r, Tape t, Movie m
WHERE c.mem_no=r.mem_no
AND t.t_id = r.tape_id
AND t.m_id = m.m_id
AND m.cat='Entertainment';
```


SQL / DML: Simple queries with joins

- Natural outer join

enhanced SQL:1999

```
<tableName> LEFT|RIGHT|FULL
NATURAL [OUTER] JOIN <tableName>
```

- Outer join with condition

```
<tableName> LEFT|RIGHT|FULL [OUTER] JOIN <tableName>
ON <condition>
```

- Example:

```
SELECT name, r.tape_id
FROM Rental r RIGHT OUTER JOIN Customer c
ON r.mem_no = c.mem_no;
```

8.2.4 Simple SQL: Output

Improving the output

- Not a feature of relational algebra :)
- Different format, even HTML or other markup can be generated in some systems

"Find title, t_Id and format for all movies"

```
BREAK ON title
COLUMN title HEADING "Movies" FORMAT A15
COLUMN id HEADING "TapeNO"
SELECT m.title, t.m_id, t.format
FROM Movie m, Tape t
WHERE m.m_Id = t.m_Id
ORDER BY title ASC;
```

Don't repeat identical titles

Column formating

Aliases for columns

Order (sort) result set

```
Movies                TapeNO FORMA
--------------- ---------- -----
Amistad                   11 VHS
Matrix                    23 DVD
```

System dependent
This kind holds for
Oracle/SQL+