SQL as Data Manipulation Language (DML)

Insert and update data

Simple SQL queries

Advanced SQL queries

Views

SQL / DML: Overview

- Insert, update, delete data

- Query data
  - Interactively
  - (Embedded in host language)

- Data presentation to users
  - Output improvements
  - Views

SQL / DML: Insert data

- Complete form:
  - Predefined order of values

```
INSERT INTO Customer
VALUES (001, 'Müller', 'Tina', NULL,NULL);
```

- Incomplete form:
  - Free order of values

```
INSERT INTO Customer
(last_name, mem_no) VALUES ('Müller', 001);
```

SQL / DML: Insert data

- Inserting dates

```
INSERT INTO movie
VALUES (95, 'Psycho', 'suspense',
             TO_DATE('1969', 'yyyy'),
            'Hitchcock', 2.00, NULL);
```

- Conversion functions
  - String to date

```
TO_DATE(<string>[,<format>])  ORACLE/Postgres
```

  - Date to string

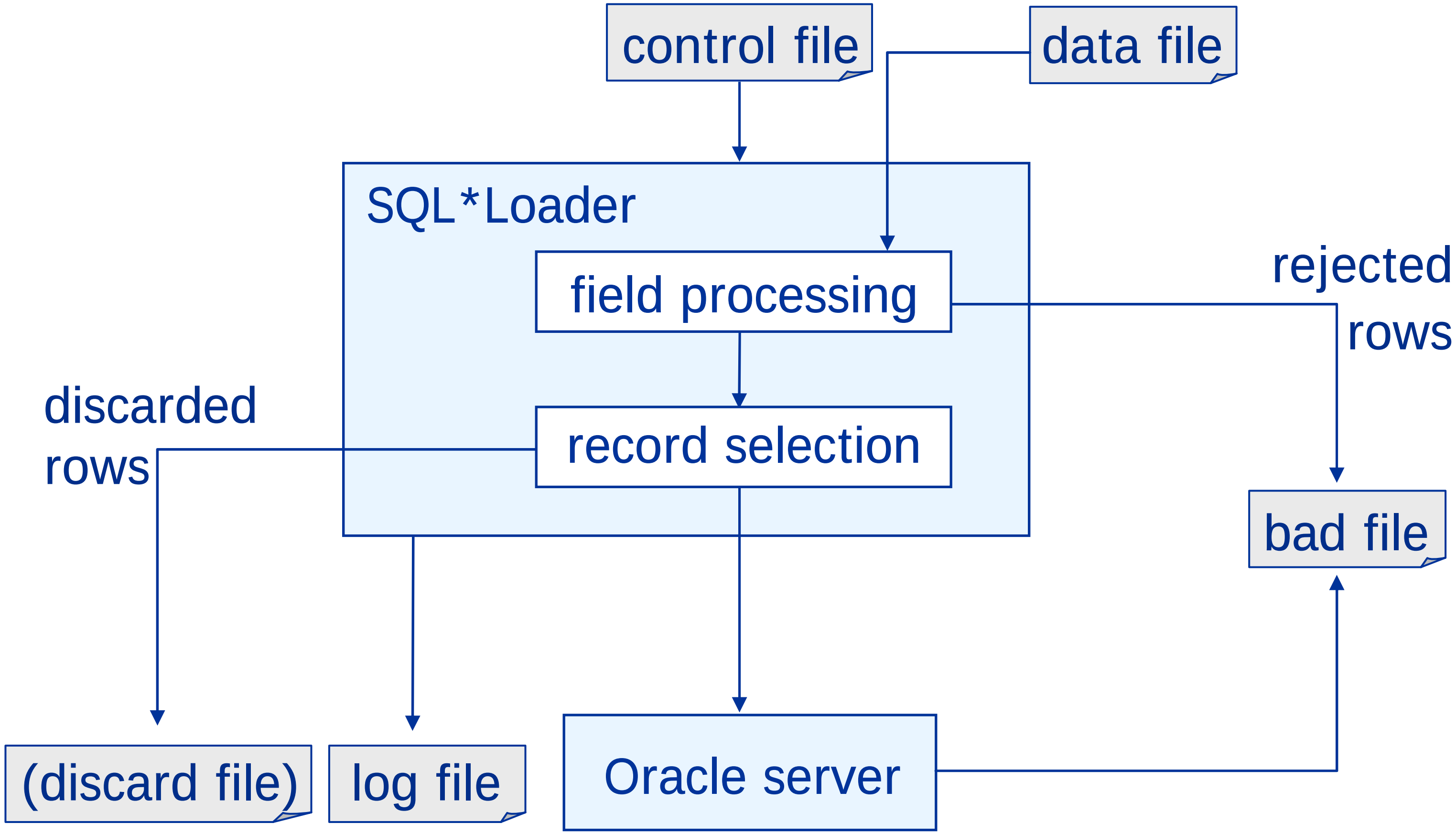
```
TO_CHAR(<date>[,<format>]) ORACLE/Postgres
```

SQL / DML: Insert data

- Loading data from files
  - System dependent
- Oracle:
  - **`INSERT INTO …`**
  - Bulk load from file: SQL loader
  - Bulk load from other database: export / import tool
- MySQL:
  - **`INSERT INTO …`**
  - Bulk load: **`LOAD DATA …`**
  - Bulk load from other database:
    `SELECT … INTO OUTFILE, LOAD DATA…`
- Postgres:
  - **`INSERT INTO …`**
  - Bulk load: **`Copy …`**
  - Bulk load from other database: Copy …

SQL / DML: Implementations of bulk load

- *Oracle SQL loader*



SQL / DML: Implementations of bulk load

- Example:

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

loadtest.dat

```
'four' , 4
'five' , 5
'six'  , 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)
```

SQL / DML: Implementations of bulk load

- MySQL Example:

```
Mysql> LOAD DATA INFILE 'loadtest.dat'
    ->    IGNORE
    ->    INTO TABLE loadtest
    ->    FIELDS
    ->      TERMINATED BY ","
    ->      OPTIONALLY ENCLOSED BY ''
    ->     (name, num );
Query OK, 3 rows affected (0.01 sec)
Records: 3 Deleted:0 Skipped: 0 Warnings: 0
```

SQL / DML: Insert unique data

- For synthetic keys, e.g. tapeId, …

- Counter-variables in application not sufficient
  - Session dependent
  - Concurrent access problematic

- Counter relation in database
  - Expensive

- Proprietary solutions in existing DBMS
  - MySQL: autoincrement keyword
  - SQL-Server: identity keyword
  - PostgreSQL: serial type and sequence
  - Oracle: sequence-object

SQL / DML: Insert unique data - sequence

- Abstract sequence-object (Oracle)
  - Creates unique integer values

- Syntax:

```
CREATE SEQUENCE <seqName>
[START WITH <integer>]
[INCREMENT BY <integer>]
[MAXVALUE <integer> | NOMINVALUE]
[MINVALUE <integer> | NOMAXVALUE]
[CYCLE | NOCYCLE]
[CACHE <integer> | NOCACHE]
[ORDER | NOORDER];
```

- Example:

```
create sequence tape_sequence;
```

SQL / DML: Insert unique data - sequence

- Value Access
  - **`<seqName>.NEXTVAL =`** value of last call + increment
  - **`<seqName>.CURRVAL =`** value of last call

  - **`SELECT <seqName>.CURRVAL FROM DUAL;`**
  - **`DUAL`** is Oracle pseudo-table

- Example:

```
INSERT INTO tape
VALUES(tape_sequence.nextval, 'DVD', 95);
```

SQL / DML: Delete data

- Syntax:

```
DELETE from <tableName>
[WHERE <predicate>];
```

  - Delete all rows :

```
DELETE from <tableName>;
```

- Example:

```
DELETE from tape
WHERE format= 'Beta';
```

SQL / DML: Update data

- Syntax:

```
UPDATE <tableName>
SET   <attr> = <value>
   {,<attr> = <value> }
WHERE <predicate>
```

- Examples:

```
UPDATE Customer
SET telephone = 456789
WHERE mem_no = 200;
```

```
UPDATE Rental
SET until_date = SYSDATE
WHERE tape_ID = 3
AND mem_no = 200
AND TO_CHAR(from_date,'yyyy-mm-dd')='2002-05-01';
```

SQL / DML: Example database

```
insert into customer values (001, 'Müller', 'Tina', NULL,NULL);
insert into customer values (007, 'Katz', 'Anna', NULL,NULL);
insert into customer values (002, 'Maus', 'Carla', NULL,NULL);
....


insert into movie values (95, 'Psycho', 'suspense',
            to_date('1969', 'yyyy'), 'Hitchcock', 2.00, NULL);
insert into movie values (112, 'ET', 'comedy',
            to_date('1982', 'yyyy'), 'Spielberg', 1.50, NULL);
....


insert into format values('DVD', '2.00');
insert into format values('Beta', '0.00');
insert into format values('VHS', '0.00');


create sequence tape_sequence;
insert into tape values (tape_sequence.nextval, 'DVD', 95);
insert into tape values (tape_sequence.nextval, 'DVD', 112);
insert into tape values (tape_sequence.nextval, 'VHS', 222);
```

SQL / DML: Example database

```
insert into rental values (3, 1,
                      to_date('2002-05-01','yyyy-mm-dd'), NULL);
insert into rental values (4, 1,
                      to_date('2002-05-01','yyyy-mm-dd'), NULL);
insert into rental values (5, 3,
                      to_date('2002-05-01','yyyy-mm-dd'),
                      to_date('2002-05-02','yyyy-mm-dd'));


insert into actor values ('Hitchcock',
                'Hitchcock', to_date(1899-08-13','yyyy-mm-dd'));
insert into actor values ('Harrison Ford',
            'Harrison Ford', to_date('1942-07-13','yyyy-mm-dd'));


insert into play values(290,'Harrison Ford');
insert into play values(98,'Hitchcock');
```

SQL / DML: Querying Language

- SQL is relational complete

- Additional query concepts

  - Advanced search expressions on strings
    e.g., find all movies starting with “star wars”

  - Arithmetic in expressions,
    e.g., number of tapes for each movie

  - Grouping and predicates over sets
    e.g., total receipts of each movie within the last year

SQL / DML: Basics

- Basic query pattern:

```
SELECT   [DISTINCT] A1, A2,...,An
FROM     R1, R2,...Rm
WHERE    predicate P;
```

  - A_1 , A_2 , ..., A_n attribute names,
  - R_1 , R_2 , ..., R_m relation names,
  - P Boolean predicate on attributes and constants

- Equivalent to relational algebra expression:

$$\Pi_{A1,\ A2,\ \ldots,\ An} \left(\sigma_P \left(R_1 \text{ X } R_2 \text{ X } \ldots \text{ X } R_m \right)\right)$$

  - Projection (RA) → **SELECT** (SQL)
  - Cartesian Product (RA) → **FROM** (SQL)
  - Selection (RA) → **WHERE** (SQL)

SQL / DML: Basics

- Query result is relation
- Query evaluation order:
  1. **FROM**-clause
  2. **WHERE**-clause
  3. **SELECT**-clause
- No duplicate removal (performance!)

```
 SQL> SELECT last_name
   2   FROM Customer;

LAST_NAME
------------------------------
Müller
Katz
Maus
Hinz
Kunz
Müller
```

SQL / DML: Basics

- Eliminating duplicates:
  - Targetlist contains KEY attribute
  - Targetlist constrains UNIQUE attribute
  - Targetliste defined with **DISTINCT**

```
SELECT mem_no, last_name
FROM Customer
```

```
 MEM_NO     LAST_NAME
---------- -----------------
         1 Müller
         7 Katz
         2 Maus
        11 Hinz
        23 Kunz
       111 Müller
```

```
SELECT DISTINCT last_name
FROM Customer
```

```
LAST_NAME
-----------------
Hinz
Katz
Kunz
Maus
Müller
```

SQL / DML: Basics

- **WHERE**-clause structure:
  - Simple Boolean predicates similar to RA and Calculus
  - Additional simple predicates:

    ```
    <attribute> BETWEEN <value1>  AND <value2>
    <attribute> IS [NOT] NULL
    <attribute> LIKE <string>
    <attribute> SIMILAR TO <string>
    ```

  - Advanced predicated with sub-queries

    Set-operators (**IN, NOT IN, SOME, ALL, EXISTS)**

SQL / DML: Simple queries

Example: All customers named Anna

```
SQL>  select mem_no, last_name, first_name
  2   from customer
  3   where first_name='Anna';

    MEM_NO LAST_NAME                      FIRST_NAME
---------- ------------------------------ --------------
         7 Katz                           Anna
        23 Kunz                           Anna
```

Example: All movies by Lucas from 1999 or later

```
SQL> select id, title
  2  from movie
  3  where director='Lucas'
  4  and to_char(year,'yyyy')>='1999';

        ID TITLE
---------- ---------------------------------------
       345 Star Wars I
```

SQL / DML: Simple queries

- More examples:

All formats with extra charge between 1 and 2 Euro

```
SELECT *
FROM Format
WHERE charge BETWEEN 1.00 and 2.00;
```

All tapes currently on loan

```
SELECT tape_id
FROM Rental
WHERE until_date IS NULL;
```

SQL / DML: Simple queries - expressions

Core
SQL:1999

- **LIKE** - expression
  - Simple form of regular expression
  - % : any sequence of characters
  - _ : exactly one character

- Example:

  All 'star wars' movies

```
SQL> select id, title, director
  2  from movie
  3  where title like 'Star Wars %';

        ID TITLE        DIRECTOR
---------- ------------ ---------------
       345 Star Wars I  Lucas
       290 Star Wars IV Lucas
```

SQL / DML: Simple queries - expressions

enhanced
SQL:1999

- **SIMILAR** - expression
  - Advanced form of regular expression

- Example:

  All 'star wars' movies

```
SELECT id, title, director
FROM movie
WHERE title SIMILAR TO
      'Star Wars (I | IV | V | VI | 1 | [4-6])';
```

SQL / DML: Simple queries

- Member in set: **IN**

All movies from Spielberg or Lukas

Core
SQL:1999

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

All movies from Lucas in 1999

enhanced
SQL:1999

```
SELECT title, director
FROM Movie
WHERE (director, year)
  IN (('Lucas',to_date(1999,'yyyy')));
```

SQL / DML: Simple queries - expressions

- Functions
  - Expressions may contain functions
  - Arithmetical and string built-in functions
  - User defined functions on user defined types

- String function examples:
  - **`SOUNDEX (<string>)`**, **`UPPER (<string>)`**
  - **`SUBSTRING(<string> FROM <integer> FOR <integer>)`**

```
SQL> SELECT title, director
  2  FROM Movie
  3  WHERE SOUNDEX(director)  =  SOUNDEX('Spilbak');

TITLE                DIRECTOR
----------------- ----------------
ET                   Spielberg
Psycho               Spielberg
Jaws                 Spielberg
```

SQL / DML: Simple queries - expressions

- Arithmetic function examples
  - **SQRT(<number>)**
  - Basic arithmetic expressions

All tapes, their price and tax

```
 SQL> SELECT id, pricepday,
  2          0.16*pricepday as tax
  3   FROM Movie;

        ID  PRICEPDAY        TAX
---------- ---------- ----------
        95          2        .32
       112        1.5        .24
       345          2        .32
       222        2.2       .352
       290          2        .32
       100        1.5        .24
```

SQL / DML: Simple queries - expressions

- Date function examples
  - differ heavily between systems
  - Oracle: **SYSDATE, MONTHS_BETWEEN, ADD_MONTHS**

```
SQL> SELECT title, to_char(year,'yyyy') as year
  2  FROM movie
  3  WHERE months_BETWEEN(SYSDATE,year)> 120 ;

TITLE                     YEAR
------------------------- ----
Psycho                    1969
ET                        1982
Jaws                      1975
```

SQL / DML: Simple queries

Core
SQL:1999

- Combination of relations
  - Schema compatible relations
  - **UNION, INTERSECT, EXCEPT**

- Syntax:

```
UNION | INTERSECT | EXCEPT
          [DISTINCT | ALL]
          [CORRESPONDING [BY <attributes>]]
```

  - Default **DISTINCT**

  - Default: all attributes used
  - **CORRESPONDING BY**: defines used common attributes
  - **CORRESPONDING**: uses all common attributes

SQL / DML: Simple queries

- Example:

All movies from Spielberg or Lukas

```
(SELECT title, director
 FROM Movie
 WHERE director like 'Spielberg')
UNION
(SELECT title, director
 FROM Movie
 WHERE director like 'Lucas');
```

SQL / DML: Simple queries

- More examples:

  All movies not by Lucas

```
(SELECT *
 FROM Movie)
EXCEPT
(SELECT * from Movie
 WHERE director='Lucas');
```

  All directors and actors in our database

```
(SELECT director as celebrity
 FROM Movie)
UNION DISTINCT
(SELECT stage_name as celebrity
 FROM Actor);
```

SQL / DML: Implementations combinations

- Oracle:
  - **UNION**
  - **MINUS** implements **EXCEPT**
  - **INTERSECT**
  - **CORRESPONDING [BY]** not implemented

- PostgreSQL:
  - **UNION**
  - **EXCEPT**
  - **INTERSECT**
  - **CORRESPONDING [BY]** not implemented

- MySQL:
  - **UNION, EXCEPT, INTERSECT** not implemented

SQL / DML: Simple queries with joins

Core
SQL:1999

- Simple joins
  - Search predicates and join conditions mixed

Example: All Tapes and their corresponding movie

```
SQL> SELECT t.id, t.format, m.id, m.title
  2  FROM Tape t, Movie m
  3  WHERE m.id = t.movie_id;

        ID FORMAT         ID TITLE

---------- ----- ---------- ----------------------------
         1 DVD            95 Psycho
         2 DVD           112 ET
         3 VHS           222 Psycho
         4 DVD           345 Star Wars I
         5 VHS           345 Star Wars I
         9 VHS           345 Star Wars I
```

SQL / DML: Simple queries with joins

enhanced SQL:1999

- Cross join (cross product)

```
<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

enhanced SQL:1999

- Inner join with attribute list

```
<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

enhanced SQL:1999

- Inner join with condition

```
<tableName> [INNER] JOIN <tableName>
ON <condition>
```

- Examples:

```
SELECT *
FROM Tape t JOIN Movie m
ON t.movie_id = m.id;
```

```
SELECT *
FROM Rental r JOIN Customer c
ON r.mem_no = c.mem_no;
```

SQL / DML: Simple queries with joins

All Customers who have rented at least one science fiction film

```
SELECT c.mem_no, c.last_name, c.first_name
FROM ((Customer c
          JOIN Rental r ON c.mem_no = r.mem_no)
          JOIN Tape t ON t.id = r.tape_id )
          JOIN Movie m ON t.movie_id = m.id
WHERE m.category='Scifi';
```

```
SELECT c.mem_no, c.last_name, c.first_name
FROM Customer c, Rental r, Tape t, Movie m
WHERE c.mem_no=r.mem_no
AND t.id = r.tape_id
AND t.movie_id = m.id
AND m.category='Scifi';
```

SQL / DML: Simple queries with joins

enhanced SQL:1999

- Natural outer join

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

- Outer join with condition

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

- Example:

```
SELECT *
FROM Rental r RIGHT OUTER JOIN Customer c
ON r.mem_no = c.mem_no;
```

SQL / DML: Simple queries with joins

- Example (extended)

```
SQL> SELECT r.tape_id, r.from_date, c.mem_no,c.first_name
  2  FROM Rental r RIGHT OUTER JOIN Customer c
  3  ON r.mem_no = c.mem_no;

   TAPE_ID FROM_DATE     MEM_NO FIRST_NAME
---------- --------- ---------- --------------------
         3 01-MAY-02          1 Tina
         4 01-MAY-02          1 Tina
         5 01-MAY-02          2 Carla
                             23 Anna
                            111 Bert
                             11 Fritz
                              7 Anna
```

SQL / DML: Implementations of joins

- Oracle/Postgres:
  - Simple join
  - Cross join
  - (natural) inner join with attribute list, with condition
  - (natural) Right, left, full outer join with condition

  - recommends ANSI-syntax for compatibility

- MySQL:
  - Simple join
  - Cross join
  - Straight join (left table always read before right one)
  - Inner join with condition
  - (natural) left, right outer join with condition

SQL / DML: Improving the output

- Not feature of relational algebra
- Example:
  - Rename column title for *this* query
  - Order tuples

```
SQL> SELECT m.title as Movies, t.id, t.format
  2  FROM Movie m, Tape t
  3  WHERE m.id = t.movie_id
  4  ORDER BY title;

MOVIES                                       ID FORMAT
------------------------------------ -------- ------
ET                                            2 DVD
Psycho                                        1 DVD
Psycho                                        3 VHS
Star Wars I                                   4 DVD
Star Wars I                                   5 VHS
Star Wars I                                   9 VHS
```

SQL / DML: Improving the output

- Syntax:

```
ORDER BY <orderexpression> ASC|DESC
```

- Ordering expression
  - No advanced expressions (no sub-query, no grouping)
  - At least one attribute reference
  - References in order expression $\subseteq$ result attributes

- Multiple sort attributes:
  - Primary ordering by first attribute
  - Secondary ordering by second attribute, …

```
SELECT m.title as Movies, t.id, t.format
FROM Movie m, Tape t
WHERE m.id = t.movie_id
ORDER BY title, format;
```

SQL / DML: Improving the output

- Advanced features system dependent
- Oracle SQL+ Example:
  - Format column title for *all* queries
  - Don't repeat identical titles

```
SQL> BREAK ON title
SQL> COLUMN title HEADING "Movies" FORMAT A15

SQL> SELECT m.title, t.id, t.format
  2  FROM Movie m, Tape t
  3  WHERE m.id = t.movie_id;

Movies                  ID FORMAT
--------------- ---------- ------
Psycho                   1 DVD
ET                       2 DVD
Psycho                   3 VHS
Star Wars I              4 DVD
                         5 VHS
                         9 VHS
```

SQL / DML: Sub-queries

Core
SQL:1999

- Sub-queries with single results
  - Operators {=, ≤, ≥, ≠, <, >}
  - Expressible without sub-query

- Example:

Movies shorter than 'Star Wars I' (id 345)

```
SELECT m.id
FROM Movie m
WHERE m.length <
              (SELECT m1.length
               FROM Movie m1
               WHERE m1.id = 345);
```

SQL / DML: Sub-queries

Core
SQL:1999

- Set Operator **IN**

- Independent sub-query example:

All Tapes for movie 'Star Wars '

```
SELECT t.id, t.format
FROM Tape t
WHERE t.movie_id
      IN (SELECT m.id
          FROM Movie m
          WHERE m.title like 'Star Wars %');
```

SQL / DML: Sub-queries

Core
SQL:1999

- Set Operator **IN**

- Correlated sub-query example:

Directors playing in their own movies

```
SELECT m.director
FROM Movie m
WHERE m.director IN
      (SELECT p.actor_name
       FROM Play p
       WHERE p.movie_id = m.id);
```

SQL / DML: Sub-queries

Core
SQL:1999

- Alternative syntax: **EXISTS**

- Example:

```
SELECT t.id, t.format
FROM Tape t
WHERE EXISTS (SELECT *
              FROM Movie m
              WHERE t.movie_id = m.id
              AND m.title like 'Star Wars %');


SELECT m.director
FROM Movie m
WHERE EXISTS (SELECT *
              FROM Play p
              WHERE p.movie_id = m.id
              AND m.director like p.actor_name);
```

SQL / DML: Query rewriting

- Rewriting possible for **IN, EXISTS**
- Examples:

```
SELECT t.id, t.format
FROM Tape t, Movie m
WHERE m.id = t.movie_id
AND m.title like 'Star Wars %';
```

```
SELECT m.director
FROM Movie m, Play p
WHERE p.movie_id = m.id
AND m.director like p.actor_name;
```

SQL / DML: Sub-queries

Core
SQL:1999

- Negation **NOT EXISTS, NOT IN**

All tapes that never have been on loan

```
SELECT t.id
FROM Tape t
WHERE NOT EXISTS (SELECT *
                  FROM Rental r
                  WHERE r.tape_id = t.id);
```

All movies no copy of which are currently on loan

```
SELECT distinct m.id
FROM Movie m
Where NOT EXISTS (SELECT *
                  FROM Rental r, Tape t
                  WHERE r.tape_id = t.id
                  AND m.id = t.movie_id
                  AND r.until_date IS NULL);
```

SQL / DML: Quantified sub-queries

enhanced
SQL:1999

- Quantified comparison operators
  - No re-writing without sub-query
  - Quantification: **ALL**, **SOME** (synonym **ANY**)
  - Operators $\in \{=, \leq, \geq, \neq, <, >\}$

- Quantification **All**
  - Sub-query true if true for all tuples

Most expensive movies

```
SELECT m.id, m.pricepday
FROM Movie m
WHERE m.pricepday >= ALL
                    (SELECT m1.pricepday
                     FROM MOVIE m1);
```

SQL / DML: Quantified sub-queries

enhanced SQL:1999

- Quantification: **SOME**
  - Sub-query true if true for at least one tuple

```
SELECT title, pricePDay
FROM Movie m
WHERE pricePday  < SOME
                   (SELECT m1.pricepday
                    FROM Movie m1);
```

  - **= SOME** *equivalent* **IN**

```
SELECT m.director
FROM Movie m
WHERE m.director = SOME
      (SELECT p.actor_name
       FROM Play p
       WHERE p.movie_id = m.id);
```

SQL / DML: Implementations of sub-queries

- Oracle:
  - Sub-queries with single results
  - Sub-queries with **`[NOT] IN, [NOT] EXISTS`**
  - Quantified comparison **`ALL, SOME, ANY`**
- PostgreSQL
  - Similar to Oracle
- MySQL:
  - No sub-queries supported

SQL / DML: Universal quantifiers

- ∀ - Quantification
- ∃ - Quantification (exactly one)
  - Describe counterexample
  - Combine with **NOT EXISTS**

Movies with only one tape

```
    SELECT m.id
    FROM Movie m
    WHERE NOT EXISTS
          (SELECT *
           FROM Tape t1, Tape t2
           WHERE t1.movie_id = t2.movie_id
           AND t1.id <> t2.id
            AND t2.movie_id = m.id );
```

SQL / DML: Universal quantifiers

All Customers whose rented movies all have category “suspense“

```
SELECT c.mem_no
FROM Customer c
WHERE NOT EXISTS
          (SELECT m.id
           FROM Movie m, Rental r, Tape t
           WHERE m.id = t.movie_id
           AND r.tape_id = t.id
           AND c.mem_no = r.mem_no
           AND m.category <> 'suspense');
```

SQL / DML: Universal quantifiers

Customers that had rented all movies

```
SELECT c.mem_no
FROM Customer c
WHERE NOT EXISTS
          (SELECT m.id
           FROM Movie m
           WHERE NOT EXISTS
                   (SELECT *
                    FROM Rental r, Tape t
                    WHERE m.id = t.movie_id
                    AND r.tape_id = t.id
                    AND c.mem_no = r.mem_no));
```

SQL / DML: Universal quantifiers

Customers that rented only one movie

```
SELECT c.mem_no
FROM Customer c, Rental r, Tape t, Movie m
WHERE c.mem_no = r.mem_no
AND r.tape_id = t.id
AND t.movie_id = m.id
AND NOT EXISTS
          (SELECT m1.id
           FROM Rental r1, Tape t1, Movie m1
           WHERE r1.tape_id = t1.id
           AND t1.movie_id = m1.id
           AND c.mem_no = r1.mem_no
           AND m1.id <> m.id);
```

SQL / DML: Aggregate functions

- Mathematical aggregate functions on data sets
- Example: SUM, AVG, MIN, MAX, COUNT
- Not in relational algebra

```
SQL> SELECT MIN(pricePDay) as MIN,
  2  MAX(pricePDay) as MAX, AVG(pricePDay)
  3  FROM Movie;

       MIN        MAX AVG(PRICEPDAY)
---------- ---------- --------------
       1.5        2.2     1.86666667
```

- Target list: only aggregate functions or none
  - Exception: **`GROUP BY`**

SQL / DML: Aggregate functions

- Comparison using aggregates: sub-queries

Movies with price above average

```
SQL> SELECT m.id, m.Title, m.pricepday
  2  FROM Movie m
  3  WHERE pricePDay >
  4                 (SELECT AVG(pricePDay)
  5                  FROM Movie);

    ID TITLE                                      PRICEPDAY
------ ---------------------------------------- ---------
    95 Psycho                                          2
   345 Star Wars I                                     2
   222 Psycho                                        2.2
   290 Star Wars IV                                    2
```

SQL / DML: Aggregate functions

- Examples:

Movies with minimal price

```
SELECT m.Title, m.pricepday
FROM Movie m
WHERE pricePDay =
                 (SELECT MIN(pricePDay)
                  FROM Movie);
```

Movie with more than 2 tapes

```
SELECT m.id, m.title
FROM Movie m
WHERE 2 <  (SELECT count(t.id)
            FROM tape t
            WHERE t.movie_id = m.id)
```

SQL / DML: Aggregate functions

- More examples:

Movies having tapes in one format only

```
SELECT m.id, m.title
FROM Movie m, Tape t1
WHERE m.id = t1.movie_id
AND 0 =
       (SELECT COUNT(*)
        FROM Tape t2
        WHERE t1.id <> t2.id
        AND t1.format <> t2.format
        AND t2.movie_id = m.id);
```

SQL / DML: Aggregate functions

- Additional qualification with **`DISTINCT | ALL`**

- Example:

  Movies that are available in all formats

```
SELECT DISTINCT t1.movie_id
FROM Tape t1
WHERE
    (SELECT COUNT(DISTINCT format)
     FROM   Tape t2
     WHERE  t2.movie_id = t1.movie_id)
     =
     (SELECT COUNT(*)
      FROM Format);
```

SQL / DML: Grouping

- Syntax:

```
SELECT <targetlist>
FROM <tablelist>
[WHERE <predicate>]
GROUP BY <attributelist>
```

- Groups all rows with same values in <attributelist>
- Target list: grouping attributes and aggregates

- Example:

Number of tapes in each format

```
SELECT t.format, count(t.id)
FROM Tape t
GROUP BY t.format;
```

SQL / DML: Grouping

- *Aggregates evaluated over groups*

Number of tapes for each movie

```
SQL> SELECT t.movie_id, count(*)
  2  FROM Tape t
  3  GROUP BY t.movie_id;

  MOVIE_ID   COUNT(*)
---------- ----------
        95          1
       100          1
       112          1
       222          1
       290          1
       345          4
```

SQL / DML: Grouping

Movie						
id	title	cat.	year	director	price	leng.
095	Psycho	…	⋯	Hitchcock	2.00	…
112	ET	…	⋯	Spielberg	1.50	…
345	Star Wars I	…	⋯	Lucas	2.00	…
222	Psycho	…	⋯	Van Sant	2.20	…
290	Star Wars IV	…	⋯	Lucas	2.00	…
100	Jaws	…	⋯	Spielberg	1.50	…
…	…	…	⋯	…	…	…

SELECT sum(price)
FROM movie;

11.2

Implicit group: all tuples in table

SQL / DML: Grouping

Movie

id	title	cat.	year	director	price	leng.
095	Psycho	…	⋯	Hitchcock	2.00	…
112	ET	…	⋯	Spielberg	1.50	…
345	Star Wars I	…	⋯	Lucas	2.00	…
222	Psycho	…	⋯	Van Sant	2.20	…
290	Star Wars IV	…	⋯	Lucas	2.00	…
100	Jaws	…	⋯	Spielberg	1.50	…
…	…	…	⋯	…	…	…

```
SELECT director, sum(price)
FROM movie
Group by director;
```

Hitchcock	2.0
Spielberg	3.0
Lucas	4.0
Van Sant	2.2

SQL / DML: Grouping

Total receipts of each tape within the last year

```
SQL> SELECT t.id, count(*)
  2  FROM Tape t, Rental r
  3  WHERE t.id = r.tape_id
  4  AND to_char(r.from_date,'yyyy') >= 2005
  5  GROUP BY t.id;

        ID   COUNT(*)
---------- ----------
         1          1
         2          1
         3          2
         4          2
         5          1
        11          1
        12          1
```

SQL / DML: Grouping

Total receipts of each movie within the last year

```
SQL> SELECT t.movie_id, count(*)
  2  FROM Tape t, Rental r
  3  WHERE t.id = r.tape_id
  4  AND to_char(r.from_date,'yyyy') >= 2005
  5  GROUP BY t.movie_id;

  MOVIE_ID   COUNT(*)
---------- ----------
        95          1
       100          1
       112          1
       222          2
       290          1
       345          3
```

SQL / DML: Grouping + Having

- Qualifying predicate for groups

```
SQL> SELECT f.name, sum(charge)
  2     FROM Rental r, Tape t,  format f
  3     WHERE t.id = r.tape_id
  4     AND t.format=f.name
  5     GROUP BY f.name;

NAME  SUM(CHA
----- -------
Beta
DVD
VHS
```

```
SQL> SELECT f.name, sum(charge)
  2     FROM Rental r, Tape t,  format f
  3     WHERE t.id = r.tape_id
  4     AND t.format=f.name
  5     GROUP BY f.name
  6     having count(f.name)>2;

NAME  SUM(CHARGE)
----- -----------
DVD             8
VHS             0
```

SQL / DML: Grouping + Having

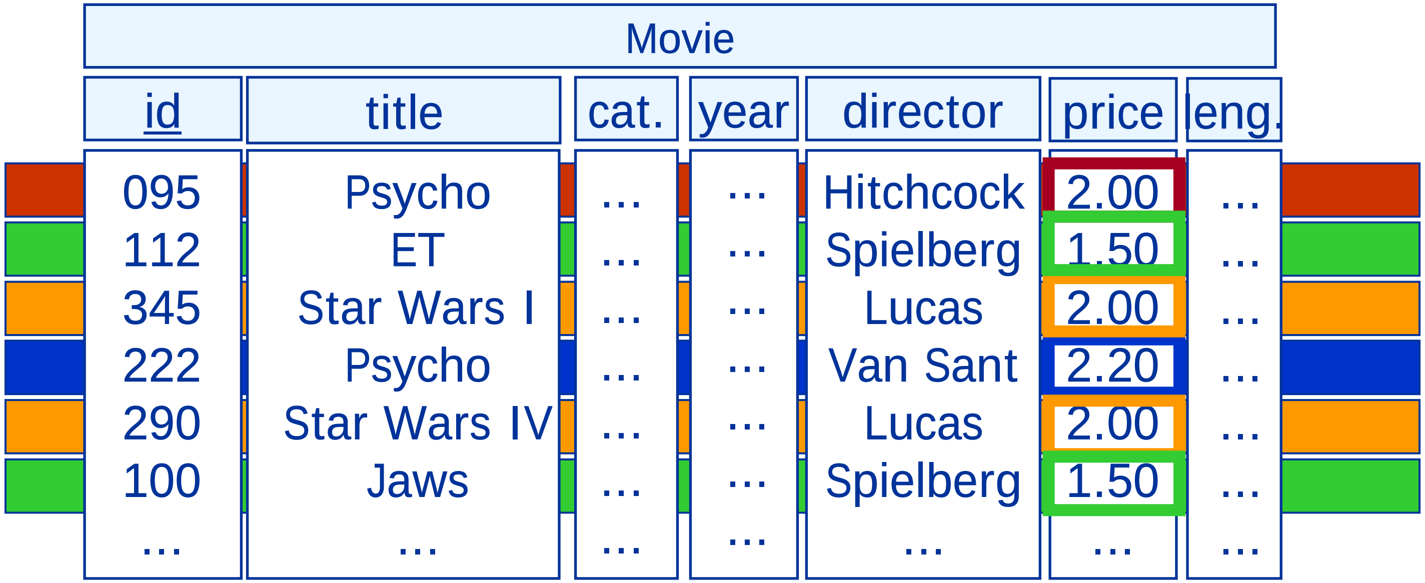
Movie

	id	title	cat.	year	director	price	leng.	
🟥	095	Psycho	…	…	Hitchcock	2.00	…	🟥
🟩	112	ET	…	…	Spielberg	1.50	…	🟩
🟧	345	Star Wars I	…	…	Lucas	2.00	…	🟧
🟦	222	Psycho	…	…	Van Sant	2.20	…	🟦
🟧	290	Star Wars IV	…	…	Lucas	2.00	…	🟧
🟩	100	Jaws	…	…	Spielberg	1.50	…	🟩
	…	…	…	…	…	…	…	

```
SELECT director, sum(price)
FROM movie
Group by director
HAVING sum(price)>2.00;
```

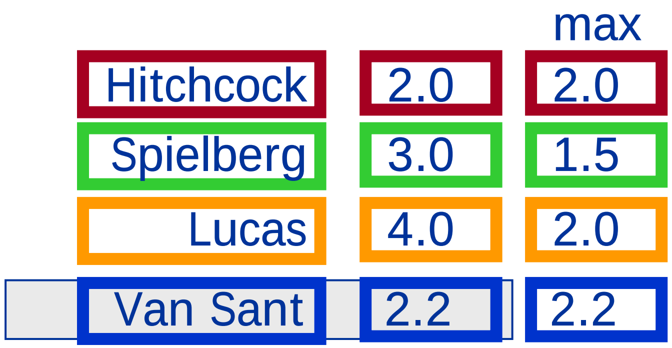
Hitchcock	2.0
Spielberg	3.0
Lucas	4.0
Van Sant	2.2

SQL / DML: Grouping + Having



	Movie							
	id	title	cat.	year	director	price	leng.	
🟥	095	Psycho	…	…	Hitchcock	2.00	…	🟥
🟩	112	ET	…	…	Spielberg	1.50	…	🟩
🟧	345	Star Wars I	…	…	Lucas	2.00	…	🟧
🟦	222	Psycho	…	…	Van Sant	2.20	…	🟦
🟧	290	Star Wars IV	…	…	Lucas	2.00	…	🟧
🟩	100	Jaws	…	…	Spielberg	1.50	…	🟩
	…	…	…	…	…	…	…	

```
SELECT director, sum(price)
FROM movie
Group by director
HAVING max(price)>2.00;
```



		max
Hitchcock	2.0	2.0
Spielberg	3.0	1.5
Lucas	4.0	2.0
Van Sant	2.2	2.2

SQL / DML: Grouping + Having

Movie						
id	title	cat.	year	director	price	leng.
095	Psycho	…	…	Hitchcock	2.00	…
112	ET	…	…	Spielberg	1.50	…
345	Star Wars I	…	…	Lucas	2.00	…
222	Psycho	…	…	Van Sant	2.20	…
290	Star Wars IV	…	…	Lucas	2.00	…
100	Jaws	…	…	Spielberg	1.50	…
…	…	…	…	…	…	…

```
SELECT director,title, sum(price)
FROM movie
Group by director, title
HAVING max(price)>2.00;
```

		max
Hitchcock, Psycho	2.0	2.0
Spielberg, ET	1.5	1.5
Lucas, SW I	2.0	2.0
Van Sant, Psycho	2.2	2.2
Lucas, SW IV	2.0	2.0
Spielberg, Jaws	1.5	1.5

SQL / DML: Grouping + Having

- **HAVING** without **GROUP BY**
  - Implicit single group contains all tuples

```
SQL> SELECT sum(charge)
  2     FROM Rental r, Tape t,  format f
  3     WHERE t.id = r.tape_id
  4     AND t.format=f.name
  5     having count(f.name)>2;

SUM(CHARGE)
-----------
          8
```

SQL / DML: Grouping + Having

- Query evaluation order:
  1. **FROM**-clause
  2. **WHERE**-clause
  3. **GROUP BY**-clause
  4. **HAVING**-clause
  5. **SELECT**-clause

Number of rentals for all customers named Anna or Tina, which rented some tapes more than once

```
SELECT c.mem_no, count(*)
FROM Rental r, Customer c
WHERE r.mem_no= c.mem_no
AND c.first_name ='Anna'
OR c.first_name='Tina'
GROUP BY c.mem_no
HAVING count(DISTINCT r.tape_id)<count(*);
```

SQL / DML: Nested aggregation with groups

- Nested aggregation using groups

Most loaned movie

```
SELECT t.movie_id, count(t.movie_id)
FROM Rental r, Tape t
WHERE r.tape_id = t.id
GROUP BY t.movie_id
HAVING COUNT(t.movie_id) > = ALL
                  (SELECT count(t1.movie_id)
                   FROM Rental r1, Tape t1
                   WHERE r1.tape_id = t1.id
                   Group BY t1.movie_id);
```

SQL / DML: Nested aggregation with groups

Movie with maximal number of tapes, show number of tapes

```
SELECT m.id, m.title, t1.t_no
FROM (SELECT t.movie_id, count(*) as t_no
      FROM tape t
      GROUP BY t.movie_id) t1, movie m
WHERE m.id=t1.movie_id
AND t1.t_no = (SELECT max(count(*))
               FROM tape t
               GROUP by t.movie_id);
```

SQL / DML: Output improvement

Core
SQL:1999

- Select values depending an condition
- Complete **CASE** form:

```
CASE
  WHEN <condition1> THEN <result1>
  [ WHEN <condition2> THEN <result2>
  [ WHEN <condition3> THEN <result3> ]]
  [ ELSE <elseresult>]
END
```

- Example:

```
SELECT length,
CASE WHEN length is NULL then 'not defined'
     WHEN length < 90 THEN 'short'
     ELSE 'long'
END
FROM Movie;
```

SQL / DML: Output improvement

Core
SQL:1999

- Simple **CASE** form

```
CASE <operand>
  WHEN <value1> THEN <result1>
  [ WHEN <value2>THEN <result2>
  [ WHEN <value3> THEN <result3> ]]
  [ ELSE <elseresult>]
END
```

- Example:

```
select f.name,
case f.name
when 'DVD' then 'DISC'
when 'Beta' then 'TAPE'
when 'VHS' then 'TAPE'
else NULL
end
from Format f;
```

SQL / DML: Transitive closure

enhanced
SQL:1999

- Recursive queries
  - Name recursion expression
  - Use name in associated query expression

- SYNTAX:

```
WITH RECURSIVE
  <queryname1> AS <query1>[,
  <queryname2> AS <query2>,…]
SELECT ...
FROM <queryname1>[,<queryname2>...]
WHERE...
```

SQL / DML: Transitive closure

enhanced SQL:1999

- Example: All lectures required for lecture XYZ

```
create table lecture(
lnr integer primary key,
name varchar(20));

        create table requires(
        pre integer references lecture(lnr),
        suc integer references lecture(lnr),
        constraint req_pk primary key(pre, suc));


WITH RECURSIVE preLecture(pre, suc)
   AS (SELECT pre,suc  FROM requires)
SELECT l.lnr as prerequisite
FROM preLecture p1, preLecture p2, lecture l
WHERE p2.suc = l.lnr
AND   l.name = ‘databases’
AND   p1.suc = p2.pre;
```

SQL / DML: Transitive closure

- Different implementation in oracle:

Lecture:

```
LNR NAME
--- ---------------
  1 databases
  2 algorithms I
  3 algorithms II
```

Requires:

```
PRE SUC
--- ---
  2   3
  3   1
```

```
SQL> SELECT  r.pre
  2  FROM    requires r, lecture l
  3  WHERE   l.name= 'databases'
  4  START WITH r.suc = l.lnr
  5  CONNECT BY PRIOR pre = suc;

       PRE
----------
         3
         2
```

SQL / DML: Structuring

enhanced SQL:1999

- Difficult structuring of complex queries
- No naming of commands / relations for re-use

- Temporary table

```
CREATE TABLE <tablename>
    {global temporary | local temporary }
    <table structure>
[ON COMMIT {PRESERVE ROWS | DELETE ROWS}]
```

  - Stores temporal query result
  - **LOCAL**: Only visible to owner
  - Dropped at end of session

SQL / DML: Temporary table

- Example:

	Test1:

CREATE TABLE testsource(id integer);
CREATE GLOBAL TEMPORARY TABLE
test1(id integer)
ON COMMIT PRESERVE ROWS;
	No rows

INSERT INTO testsource values(1);	No rows

INSERT INTO test1
SELECT *
FROM testsource;
	1

INSERT INTO testsource values(2);	1
COMMIT;	1

SQL / DML: Temporary table

- Example:

	Test2:

`CREATE TABLE testsource(id integer);`
`CREATE GLOBAL TEMPORARY TABLE`
`test2(id integer)`
`ON COMMIT DELETE ROWS;`
	No rows

`INSERT INTO testsource values(1);`	No rows

`INSERT INTO test2`
`SELECT *`
`FROM testsource;`
	1

`INSERT INTO testsource values(2);`	1
`COMMIT;`	No rows

SQL / DML: View

Important concept

- SQL-object (virtual relation)
- Important for
  - Tailoring database schema for different applications
  - Access protection, Privacy
  - Structuring complex queries
- Relational concept for external schema
- Syntax:

```
CREATE VIEW <viewname> AS <query>;
```

- Example:

```
CREATE VIEW rental_overview AS
  SELECT r.mem_no, c.last_name, r.tape_id
  FROM rental r, customer c
  WHERE r.mem_no = c.mem_no;
```

SQL / DML: View updates

Core
SQL:1999

- Updateable views:
  1. No **SELECT DISTINCT**
  2. No duplicate attribute in target list
  3. Only one table reference
  4. No **GROUP BY**
  5. No aggregates
  6. No set operators (**INTERSECT, EXCEPT, UNION**)

- Formal:

$$u\,(V(B)) \;=\; V\,(c_u\,(B)\,)$$

  - V(B) view on base tables B
  - u(V(B)) update on view
  - c_u : equivalent update(s) on base relations must exist

SQL / DML: View updates

Core
SQL:1999

- Examples:
  - Not updateable (distinct, group by):

```
CREATE VIEW movieformats (movie, numFormats)
  AS SELECT  movie_id, count(distinct format)
  FROM  Tape t
  GROUP BY movie_id;
```

  - Updateable:

```
CREATE VIEW movies (movie, name)
  AS SELECT  id, title
  FROM Movie
  WHERE id > 100;
```

u: `INSERT INTO movies VALUES (47,‘The Fugitive‘);`

c_u: `INSERT INTO movie (id, title)`
`VALUES (47,‘The Fugitive‘);`

SQL / DML: View updates

- Additional conditions:
  - If u does not have an effect, then c_u should not
  - No side effects: c should only effect tuples in B which are represented in V
  - Inverse update: For a view update u there should be an inverse update w such that w (u (V(B))) = V (B)
  - No constraint on base tables must be violated by u

SQL / DML: View updates

- Views with check option
  - CHECK OPTION prevent side effects on base tables

- Syntax:

```
CREATE VIEW <viewname>
AS <query>
WITH CHECK OPTION;
```

- Theoretically more views updateable
  - e.g. UNIQUE columns in joins

- Enhanced SQL:
  - 1999 additional complex conditions for view update

SQL / DML: Remarks about NULL

- NULL treated as "unknown"
- Predicates:
  - NULL AND TRUE = NULL
  - NULL OR TRUE = TRUE
  - predicate evaluates to NULL for row r → r not returned
- Arithmetical expression:
  - If NULL involved → expression evaluates to NULL
- Aggregates:
  - Count(*) counts also NULL
  - avg(*) ≠ sum(*)/count(*)

SQL / DML: Summary

- SQL as Data manipulation language
  - Declarative language
  - Relational complete query language

- Important terms and concepts:
  - Insert, update, delete data
  - Basic query pattern
  - **DISTINCT**, **ALL**
  - Set combination (**UNION, INTERSECT, EXCEPT**)
  - Joins
  - Sub-queries (**IN, EXISTSSOME, ALL**)
  - Aggregate functions
  - **GROUP BY, HAVING**
  - View, view updates