Query languages (NDBI049)

SQL language

Jaroslav Pokorný

MFF UK, Praha

jaroslav.pokorny@matfyz.cuni.cz

Content

Content

Start - overview of SQL92

1) data definition language,
2) interactive data manipulation language,
3) data manipulation language in host version,
4) possibility of views definition,
5) possibility of IC definition,
6) possibility of definition přístupových práv,
7) system catalogue
8) module language,
9) transaction management.

Example: relational schema

RENTS(COPY_N, RENTAL_ID, PIN, PRICE, DATE_DB)
{data about rents of copies – rental Id, customer PIN, price, date due back}

CINEMAS(C_NAME, ADDRESS, MANAGER)
{data about cinemas and their managers}

MOVIES(TITLE, DIRECTOR) {data about movies and their directors}

MOVIE_SHOWINGS(C_NAME, TITLE, DATE)
{data about cinemas playing movies}

CUSTOMERS(PIN, NAME, ADDRESS) {data about customers}

EMPLOYEES(E_ID, ADDRESS, NAME, SALARY)
{data about the rental employees}

COPIES(COPY_N, TITLE) {copies of movies}
BOOKING(TITLE, PIN) {booking of movies by customers}

1. Data definition in SQL

- CREATE TABLE

```
CREATE TABLE RENTS
(COPY_N CHAR(3) NOT NULL,
RENTAL_ID CHARACTER(6) NOT NULL,
PIN CHARACTER(10) NOT NULL,
PRICE DECIMAL(5,2),
DATE_DB DATE);
```

Possibilities:

global temporary,

local temporary tables

Also: derived tables ($\supset$ views).

1. Data definition in SQL

Data types in SQL

- numeric (exact and approximate),
- character strings,
- bit strings,
- temporal data,
- time intervals.

NULL (is element of all domain type)

TRUE, FALSE, UNKNOWN

Conversions: automatically, explicitly (function CAST)

1. Data definition in SQL

- column ICs
  - NOT NULL — the column cannot contain the NULL value,
  - DEFAULT — sets column default value for the column when no value is specified,
  - UNIQUE — ensures that all values in the column are different, NULL value nevadí,
  - PRIMARY KEY — column combination of NOT NULL and uniquely identifies each row in column table,
  - FOREIGN KEY — column is a foreign key defining referential integrity with another table
  - CHECK — logical expression defining a specific IC
- table ICs (e.g., composite primary key), named ICs

1. Data definition in SQL

CREATE TABLE table_name(list_of_table_elements)
list_of_table_elements ::= table_element[,table_element]...
table_element ::= column_definition | table_IC_definition

- ALTER TABLE;
- DROP TABLE
- CREATE SCHEMA
  - contains definitions of basic tables, views, domains, integrity constraints, authorization privilege
- DROP SCHEMA

Df.: *Database in SQL* is a collection of tables and views. It can be defined by one or more schemas.

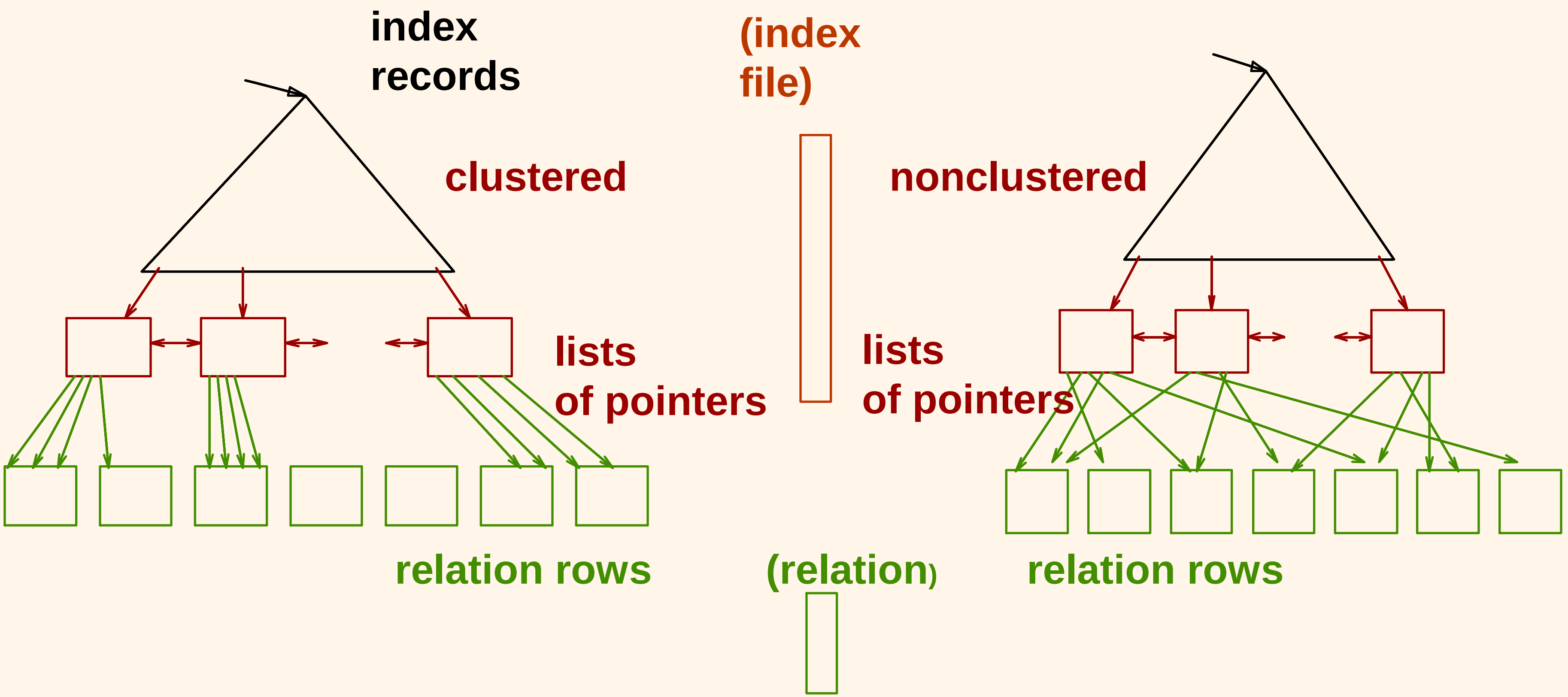
1. Data definition in SQL

```
. . .
CREATE TABLE CINEMAS . . .
. . .
CREATE TABLE MOVIE_SHOWINGS
 (C_NAME   Char_Varying(20)    NOT NULL,
  TITLE  Char_Varying(20)        NOT NULL,
  DATE  Date                     NOT NULL,
  PRIMARY KEY (C_NAME, TITLE),
  FOREIGN KEY (C_NAME) REFERENCES CINEMAS,
  FOREIGN KEY (TITLE) REFERENCES MOVIES);
```

Remark: Tables in SQL may not have a primary key!

1.3 Indexes in SQL - Nonclustered vs. clustered

```
CREATE INDEX  Idx_Cust_name_addr
              ON CUSTOMERS (NAME, ADDRESS)
```



2. Data manipulation in SQL

```
SELECT [{DISTINCT | ALL}] [{* |  name_atr1[, name_atr2]... } ]
FROM name_rel1[, name_rel2]...
[WHERE condition]
[ORDER BY sorting_specification]
```

Simple queries in SQL: Boolean expressions, event. with new predicates, are allowed in the WHERE clause

DATE_DB BETWEEN '2015-04-23' AND '2015-05-23’

Q1. Find customer names with their addresses.

```
SELECT NAME, ADDRESS
FROM CUSTOMERS
```

```
SELECT DISTINCT NAME,
FROM CUSTOMERS;
ORDER BY NAME ASC;
```

2. Data manipulation in SQL

Semantics:

SELECT DISTINCT $A_1,A_2,\ldots,A_j$ FROM $R_1,R_2,\ldots,R_k$ WHERE φ

$\cong$ $(R_1 \times R_2 \times \ldots \times R_k)(\varphi)[A_1,A_2,\ldots,A_j]$

Q2. Find couples of customers, having the same address.

```
SELECT X.PIN AS first, Y.PIN AS second
FROM CUSTOMERS X, CUSTOMERS Y
WHERE X.ADDRESS = Y.ADDRESS AND X.PIN < Y.PIN;
```

From version SQL92: *local renaming* columns

2. Data manipulation in SQL

Q3. Find rows in RENTS with date due back until 23.4.2015.

```
SELECT * FROM RENTS
                WHERE DATE_DB ≤ '2015-04-23';
```

Q4. Find directors, whose some movies are booked.

```
SELECT DISTINCT DIRECTOR
FROM MOVIES, BOOKING
WHERE  MOVIES.TITLE = BOOKING.TITLE;
```

2. Data manipulation in SQL

Evaluation of logical conditions

A	B	A and B	A or B	not A
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN

Semantics of comparisons:

x Θ y = UNKNOWN if and only if at least one from x, y is NULL

So: NULL = NULL is evaluated as UNKNOWN

2.1 Arithmetic

Q5. Find for Heinrich Götz numbers of copies, he borrowed, with the rents prices in EUR.

```
SELECT R.COPY_N, R.PRICE/25.15
FROM RENTS R, CUSTOMERS C
WHERE C.NAME = ' Götz H.' AND R.PIN = C.PIN;
```

- operators /,+, - and *, precedence order from usual practice

  Recommendation: better to use always parentheses
- NULL is propagated into the result, i.e., when one from operands is NULL, the operation result is NULL.

2.2 Aggregate functions

aggregate_function([{ALL|DISTINCT}] columns_names)

COUNT, SUM, MAX, MIN and AVG.

- They are applied on by a query specified column of a table.

Exception: COUNT(*) counts items including their duplicates and *empty rows*

- Aggregate functions applied on columns ignore NULL values.
- inclusion or non-inclusion of duplicates in the result is obeyed by ALL and DISTINCT.
- In the case of $\varnothing$ (empty table) COUNT($\varnothing$) = 0.

2.2 *Aggregate functions*

Q6. How many movies are booked?

```
SELECT  COUNT(DISTINCT TITLE)
FROM  BOOKING;
```

Q7. Find the number of rents with rent prices up to 899 CZK.

```
SELECT COUNT(*)
FROM RENTS
WHERE PRICE ≤ 899.00;
```

2.2 Aggregate functions

- SUM and AVG calculate (DISTINCT is not specified) with duplicate values.
- Inclusion of duplicate values also explicitly with ALL.
- SUM(∅) = NULL and AVG(∅) = NULL.

Q8. What is the total amount of money in rents of H. Götz?

```
SELECT SUM(R.PRICE)
FROM RENTS R, CUSTOMERS C
WHERE C.NAME = ' Götz H.' AND R.PIN = C.PIN;
```

- MIN(∅) = NULL and MAX(∅) = NULL.

2.2 Aggregate functions

```
SELECT [{DISTINCT | ALL}] {*|
value_expression1[,value_expression2] ...}
```

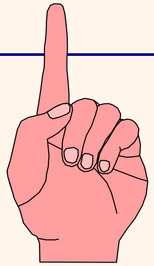
value expression uses arithmetical expressions, applications of aggregate functions, values of scalar subqueries (return just one value).

Rule: The use of aggregate functions in SELECT clause precludes the use of another column.

Q9. Find copie numbers with the highest rent price.

Incorrectly

```
SELECT COPY_N, MAX(PRICE)
FROM RENTS;
```



2.2 Aggregate functions

Q9 with a *scalar subquery*:

```
SELECT COPY_N, PRICE
FROM RENTS
WHERE PRICE = (SELECT MAX(PRICE) FROM RENTS);
```

Q10. Find PINs of customers, having rented more than 2 copies.

```
SELECT PIN, COUNT(COPY_N) AS number_of_copies
FROM RENTS
GROUP BY PIN
HAVING 2 < COUNT(COPY_N);
```

Remark: If we only want PIN, it is not necessary to write COUNT(COPY_N) in SELECT clause. Older SQL implementations require it often.

2.2 *Aggregate functions*

Q11. Find cinemas and their addresses, where they have more than 8 movies in the programme.

```
SELECT DISTINCT C.NAME, C.ADDRESS
FROM CINEMAS C
WHERE 8 < (SELECT COUNT(TITLE)
           FROM MOVIE_SHOWINGS M
           WHERE M.NAME = K. NAME);
```

Remark: placing a scalar subquery on both sides of the comparison operator Θ is possible.

Q12. Find average price from minimum prices of rented copies for each customer.

In SQL89 it is not possible to formulate this query by one SQL statement.

2.2 Aggregate functions

- Multi-level aggregation

```
SELECT PIN, PRICE, COUNT(COPY_N) AS počet_kopií,
                    (SELECT SUM(R.PRICE) FROM RENTS R
                    WHERE R.PIN = PIN) AS TOTAL_PRICE
FROM RENTS
GROUP BY PIN, PRICE;
```

Q13. Find for each customer and the given price the number of his/her rents (with this price) and total amount of money for *all* his/her rents.

2.3 Value expressions

- CASE expressions

```
CASE
        WHEN GENDER = 'M' THEN 1
        WHEN GENDER = ‚W'  THEN 2
END
```

ELSE is also possible. In example, we suppose implicitly ELSE NULL, i.e., if GENDER value is not given, then NULL is inserted in the row on place of the value of the column.

2.3 Value expressions

- function COALESCE

  COALESCE(RENTS.PRICE, "PRICE IS NOT GIVEN")

  returns in the case, when price of the copy is NULL, "PRICE IS NOT GIVEN", otherwise, value RENTS.PRICE.

Generally:

$$COALESCE(V_1,V_2,...,V_n)$$

evaluates from left to right and returns the first value that is not NULL. If it does not exist, the result is NULL.

2.3 Value expressions

- function NULLIF

  NULLIF(V1, V2), is equivalent to expression

  CASE WHEN V1 = V2 THEN NULL ELSE V1 END

Q14.(SQL92)

```
SELECT DISTINCT MANAGER
FROM CINEMAS C, CUSTOMERS CU
WHERE C.MANAGER = CU.NAME AND
        2000 > COALESCE((SELECT SUM(R.PRICE)
                        FROM RENTS R
                        WHERE R.PIN = CU.PIN),0);
```

2.4 Predicate LIKE

Q16. Find salaries of employees, who are from Kolín.

The problem is we do not know whether the database contains 'Kolin', or 'Kolín'.

```
SELECT E.SALARY
FROM EMPLOYEES E
WHERE E.ADDRESS LIKE '%Kol_n%';
```

_ the underscore represents a single character,

% the percent sign represents zero, one, or multiple characters.

2.5 Other predicates in SQL92

Q18. Find rental IDs of rents, that are rented indefinitely (DATE_DB is missing).

possibilities: IS NOT NULL,

comparisons with TRUE, FALSE and UNKNOWN.

```
SELECT RENTAL_ID
FROM RENTS
WHERE DATE_DB IS NULL;
```

2.6 Set predicates

- Predicate IN

```
column_name [NOT] IN subquery
or
column_name [NOT] IN (list_hodnot)
```

Q19. Find the addresses of the cinemas where they play the movie Aquaman.

```
SELECT ADDRESS FROM CINEMAS
WHERE NAME  IN        (SELECT  NAME
                       FROM MOVIE_SHOWINGS
                       WHERE TITLE = ‚Aquaman');
```

- column_name IN (∅) returns FALSE
- column_name IN (ℵ) returns UNKNOWN

2.6 Set predicates

Q20. Find movies with given directors.

```
SELECT TITLE  FROM  MOVIES
 WHERE DIRECTOR IN (' Menzel ',' Chytilová ',  ‘Kachyňa');
```

Q21. Find names of customers booking a movie directed by Spielberg.

```
SELECT NAME  FROM CUSTOMERS  WHERE PIN IN
  (SELECT PIN FROM BOOKING B
   WHERE B. TITLE = (SELECT M.TITLE FROM MOVIES M
                     WHERE M.DIRECTOR = ‘Spielberg’));
```

2.7. Predicates ANY, ALL, SOME

- >SOME, <SOME, <>SOME ($\Leftrightarrow$ NOT IN), =SOME ($\Leftrightarrow$ IN)). ANY is synonym for SOME.
- > ALL expresses: "greater than all items from the specified set" (+ another comparisons)
  - column_name Θ ALL($\emptyset$) returns TRUE,
  - column_name Θ ALL($\aleph$) returns UNKNOWN,
  - column_name Θ ANY($\emptyset$) returns FALSE,
  - column_name Θ ANY($\aleph$) returns UNKNOWN.

2.7. Predicates ANY, ALL, SOME

Q22. Find employees having salary higher that all employees from Praha.

```
SELECT E_ID, NAME
FROM EMPLOYEES
WHERE SALARY >  ALL(SELECT Z.SALARY
                    FROM EMPLOYEES E
                   WHERE E.ADDRESS LIKE '%Praha%');
```

2.8 Quantification in SQL

Ex. "For all movies holds, they have a director".

Logic: universal (∀) and existential (∃) quantifier are related by transformation:

$\forall x\ (p(x)) \cong \neg \exists x\ (\neg p(x))$ /* is equivalent to*/

Equivalent expression: „There is no movie such that it is not true, that this movie has a director".

More simply: "Each movie has a director " is equivalent to „There is no movie without director".

- EXISTS

simulates ∃ (test of non-emptiness of a set)

[NOT] EXISTS subquery

2.8 Quantification in SQL

```
SELECT NAME
FROM CUSTOMERS C
WHERE EXISTS (SELECT *  FROM BOOKING
                        WHERE PIN = C. PIN);
```

Q23. Find names of customers having booked a movie.

Q23'. Find names of customers such that there is a movie, they have booked.

Semantics:

- The expression is evaluated as TRUE, if the set given by the subquery is non-empty. Otherwise, it gets the value FALSE.
- The evaluation goes according to the Boolean logic.

2.9 Set operations

query_expression UNION [ALL] query_expression [ORDER BY ordering_specification]

- UNION,
- INTERSECT,
- EXCEPT.
  - + more complex expressions, e.g., (set-like) $(X \cap Y) \cup Z$, where X, Y, Z are given by subqueries or as TABLE T
  - eliminate duplicates
  - can be simulated using LEFT OUTER JOIN and test IS NULL

```
Q24.    (SELECT NAME FROM CINEMAS)
        EXCEPT
        (SELECT NAME FROM MOVIE_SHOWINGS);
```

2.9 Set operations

CORRESPONDING [BY (column_list)]

- CORRESPONDING
  - It is possible to specify over which common columns the set operation is performed
  - without columns specification, only columns common for both operands appear.
  - adding BY (column_list) it is possible to chose only some common columns.

TABLE CUSTOMERS UNION CORRESPONDING
TABLE EMPLOYEES

⇔ CUSTOMERS[JM, ADDRESS] ∪ EMPLOYEES[JM, ADDRESS]

2.10 Join of tables

- natural join,
- cross join,
- join with condition,
- join on listed columns,
- inner join,
- outer join,
- union join

2.10 Join of tables

- *Natural join*

```
SELECT *
FROM MOVIES NATURAL JOIN
MOVIE_SHOWINGS;
```

- *Cross join*

```
SELECT *
FROM R CROSS JOIN S;
```

- *Join with condition*

```
SELECT *
FROM R JOIN S ON A≤B;
```

- *Join on listed columns*

```
SELECT *
FROM U JOIN V USING (Z, Y);
```

2.10 Join of tables

- *inner join*
- *outer join* (LEFT, RIGHT and FULL)
  - again naturally or with ON.

```
SELECT *
FROM MOVIE_SHOWINGS NATURAL RIGHT OUTER JOIN
        MOVIES;
```

We obtain a table containing also the movies they do not give anywhere.

- *union join*

```
SELECT *
FROM U UNION JOIN V;
```

Each row of the left and right operand is completed from the right and from the left, respectively, with NULL values in the result .
UNION JOIN is absent from SQL:2003!

2.10 Join of tables

The FROM clause can contain derived tables specified by SELECT (⇔ CROSS JOIN)

```
Q12. (SQL)
        SELECT AVG(T.minim_c)
        FROM (SELECT  MIN(PRICE)
                FROM RENTS
                GROUP BY PIN) AS T(minim_c);
```

Query expression is a collection of terms connected with UNION, INTERSECT, EXCEPT. Each term is either a query specification (SELECT) or constant row or a table given by respective constructors.

3. Updating in SQL

```
DELETE FROM MOVIES
WHERE TITLE = 'Gun';
```

What will be done, when the movie Gun has copies, or it is booked?

```
UPDATE CUSTOMERS  SET NAME = 'Götz'
                           WHERE PIN = '4655292130';
```

```
UPDATE CUSTOMERS SET NAME = 'Müller'
                          WHERE NAME = 'Muller';
```

```
ALTER TABLE CUSTOMERS
Add NUMBER_OF_RENTS Number;

UPDATE CUSTOMERS C
SET NUMBER_OF_RENTS = (SELECT count(*) from
                          RENTS R WHERE R.PIN = C. PIN);
```

4. Views

for updatable views

```
CREATE VIEW view_name [(v_atr1_name[,v_atr2_name]...)]
AS query_specification
[WITH [{CASCADE | LOCAL} CHECK OPTION]
```

```
CREATE VIEW Praguers AS
SELECT RENTAL_ID, NAME,  ADDRESS
FROM CUSTOMERS WHERE  ADDRESS LIKE '%PRAHA%';
```

```
DROP VIEW Praguers;
```

```
CREATE VIEW HOW_MANY_COPIES (PIN,
        NUMBER_OF_RENTS) AS
                SELECT PIN, COUNT(COPY_N) FROM RENTS
                GROUP BY PIN;
```

4. Views

- view can not be indexed

Updating view leads to updating the basic table underlining the view,

- a view given by a join of more tables is not (usually) updatable,
- a view based on one table is not updatable, if it
  - contains a column with a derived value,
  - separates by a projection a column restricted by NOT NULL constraint (mainly PRIMARY KEY)

4. Views

Usage of views

- data confidentiality (it is possible to submit only some columns and rows),
- hiding complexity (complex query hidden in the view definition is designed only once),
- optimization (e.g., hiding complexity when searching for common subexpressions).

4. Views

Materialization of views

- Materialized views are not virtual, but real tables.
- They can be automatically maintained (incrementally or by recalculating the whole table of the view).
- Support: Oracle, DB2

5. Integrity constraints

- CREATE DOMAIN

```
CREATE DOMAIN
    THIS_YEAR IS DATE DEFAULT ‘2001-12-31’
    CHECK (VALUE >= ‘2010-01-01’ AND VALUE <= ‘2010-12-31’)
    NOT NULL;

CREATE TABLE RENTS
(COPY_N CHAR(3) UNIQUE NOT NULL,
RENTAL_ID CHARACTER(6) NOT NULL,
PRICE DECIMAL(5,2) CHECK (PRICE >= 100),
PIN CHARACTER(10) NOT NULL,
DATE_DB THIS_YEAR)
PRIMARY KEY (RENTAL_ID);
```

5. Integrity constraints

```
PRICE DECIMAL(5,2)
        CONSTRAINT GREATER100 CHECK (PRICE >= 100)
```

- named IC, references to other columns, tables

IC: „No movie directed by Woody Allen is played at cinemas" for the column TITLE in MOVIE_SHOWINGS.

```
CHECK (TITLE <> ANY (SELECT TITLE FROM MOVIES
                     WHERE DIRECTOR = 'Woody Allen') )
```

- table ICs

```
CONSTRAINT Allen_no ...
```

5. Integrity constraints

Problem: Table ICs are satisfied in ∅ as well.

IC: „They are always playing a movie“.

```
CONSTRAINT MOVIE_SHOWINGS_ALWAYS
        CHECK (SELECT COUNT(*) FROM MOVIE_SHOWINGS) > 0
```

Solution:

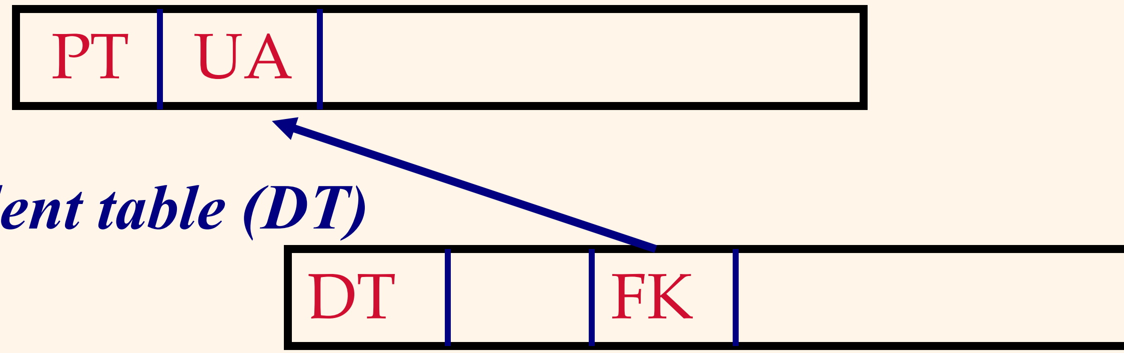
assertions - are defined out of tables

- CREATE ASSERTION

named IC formulated using CHECK. IC test is not automatically TRUE if the associated table is empty!

5.1 Referential integrity

parent table (PT)
master



dependent table (DT)
detail
child

FK foreign key, its value can be NULL,
its domain is given by the actual domain of the unique attribute UA (e.g., primary key or UNIQUE NOT NULL)

Remarks:

- null values are associated with cardinalities 1:M in E-R model.
- an attempt to break the referential integrity, only an error message was raised by SQL89.

5.1 Referential integrity

- Referential integrity can be defined
  - in definition of a column IC
  - in definition of a table IC

```
FOREIGN KEY (COPY_N) REFERENCES Copies,
FOREIGN KEY (PIN) REFERENCES CUSTOMERS)
```

- Operational behaviour

DELETE (row from parent table)

- cascade delete of rows (ON DELETE CASCADE)
- replacing foreign key by null value (SET NULL)
- replacing foreign key by implicit value (SET DEFAULT)
- Non-deleting row with a notice (NO ACTION)

Syntax: ON DELETE action, or ON UPDATE action

5.1 Example

```
. . .
DROP TABLE CINEMAS CASCADE CONSTRAINTS;
CREATE TABLE CINEMAS . . .
ON DELETE CASCADE

CREATE TABLE MOVIE_SHOWINGS
 (C_NAME   Char_Varying(20)    NOT NULL,
  TITLE   Char_Varying(20)        NOT NULL,
  DATE  Date                      NOT NULL,
  PRIMARY KEY (C_NAME, TITLE),
  FOREIGN KEY (C_NAME) REFERENCES CINEMAS,
  FOREIGN KEY (TITLE) REFERENCES MOVIES);
```

5.1 Table definition - summary

```
CREATE TABLE table_name (
    {column_name data_type [ NOT NULL ] [ UNIQUE ]
       [ DEFAULT value ] [ CHECK (selection_condition)
       [, column_name … ]}
    [ PRIMARY KEY (list_of_column_names), ]
    { [ FOREIGN KEY (list_of_column_names_creating_foreign_key)
      REFERENCES parent_table_name [(list_of_column_names)] ,
      [ MATCH { PARTIAL | FULL }]
      [ ON UPDATE referential_action ]
      [ ON DELETE referential_action ] ]
      [ , … ] }
    { [ CHECK  (selection_condition) [ , …] }
  )
```

5.2 *Other possibilities of IC*

WITH CHECK OPTION provides another possibility for expressing an IC over a basic table of a view.

```
CREATE VIEW COPIES_V
AS
SELECT * FROM Copies C
WHERE C.TITLE IN (SELECT TITLE FROM MOVIES)
WITH CHECK OPTION
```

View expresses referential integrity and can be an alternative to its declarative expressing for SQL machines, where it is not supported.

6. System catalogue

SysIndexes(IName, ICreator, TName, Creator, .)

SysViews(ViewName, VCreator, …)

- queries over the catalogue using SQL

SELECT * FROM Tab

7. Data protection

Examples:

- ALTER
- DELETE
- EXECUTE
- INDEX
- INSERT
- REFERENCES
- SELECT
- UPDATE

It is possible to assign a user /user role the right to perform the given actions over a given object

REVOKE ALL PRIVILEGES ON MOVIES FROM PUBLIC;

- remove access privileges
- PUBLIC refers to the implicitly defined group of roles

GRANT ALL PRIVILEGES ON MOVIES TO PUBLIC;

8. Standardization of SQL

SEQUEL: development by IBM in 70ties

Standardizing organizations:

ANSI and ISO (International Organization of Standardization, but also from Greek „the same“ (isos - ίδιος))

SQL standards:

- SQL86
- SQL89 (minor revision of SQL86)
- SQL92
  - entry (minor revision of SQL89)
  - intermediate (appr. a half of all functionality)
  - full

8. Standardization of SQL

- SQL99 (object extension, recursion, triggers, …)
  - all features are enumerated and either flagged mandatory or optional
  - conforming systems must comply with all mandatory features, which are called Core SQL"
- SQL:2003
  - something from XML
  - five parts of SQL/MM (Multimedia and Application Packages) have been completed

8. Standardization of SQL

- SQL:2006
  - full integration of XML into SQL including XQuery
- SQL/MM (Multimedia and Application Packages)
  - Part 1: Framework,
  - Part 2: Full Text,
  - Part 3: Spatial objects,
  - Part 5: Still Images
  - Part 6: Data mining
  - Part 7: History (draft from 2011), now ISO/IEC TS 13249-7
  - Part 8: Metadata registry (draft from 2011), now ISO/IEC 11179

8. Standardization of SQL

- SQL:2008
  - part 1: Framework (SQL/Framework)
  - part 2: Foundation (SQL/Foundation) 1100 p.
  - part 3: Call-Level Interface (SQL/CLI*)
  - part 4: Persistent Stored Modules (SQL/PSM**)
  - part 9: Management of External Data (SQL/MED)
  - part 10: Object Language Bindings (SQL/OLB)
  - part 11: Information and Definition Schemas (SQL/Schemata)
  - part 13: SQL Routines and Types Using the Java TM PL (SQL/JRT)
  - part 14: XML-Related Specifications (SQL/XML)

* alternative to calling SQL from application programs (implementation: ODBC)

** procedural language for transaction management (alternatives: IBM: SQL PL, Microsoft/Sybase: T- SQL, MySQL: MySQL, Oracle: PL/SQL, PostgreSQL: PL/pgSQL

8. Standardization of SQL

– Parts 5, 6, 8 do not exist

Temporally suspended:

– part 7 – SQL/Temporal (partially implemented in ORACLE 11g, IBM DB2 for z/OS, Teradata 13.10),

Canceled:

– part 12 – SQL/Replication

- SQL:2011
  - a statement for disabling validation of ICs
  - contains a support of temporal databases – it distinguishes from the approach of the canceled part 7

8. Standardization of SQL

- SQL:2016 (has more than 4300 pages)
  - recognition of rows patterns – a pattern is given by a regular expression (appropriate for searching patterns in time series)
  - support of JSON type (not natively – see XML, but it uses character strings)
- SQL:2019
  - multimedial arrays (type MDarray + operators)
- SQL:2022
  - new integration object memory
- SQL:2023
  - graph queries, more JSON,
  - polymorphic functions

9. Conclusion

- SQL is primarily the communication language
- aplicability vs. monstrous size