Structured Query Language
SQL I

History

- IBM – after 1970 - relational DBMS prototype - System R
- After 1908 – basis of 2 commercial DBMSs: SQL/DS, DB2

SQL standard

- ***Standardization institutions***
  - ANSI: American National Standards Institute
  - ISO: International Organization for Standardization
  - IEC: International Electrotechnical Commission

- ***SQL86 (sometimes called SQL87) - SQL 1***
  - 1986 ANSI X3.135-1986 *Database language SQL*
  - 1987 ISO 9075-1987 *Database language SQL*

  Referential integrity not standardized before 1989.
  - 1989 ANSI X3.135-1989 *Database Language SQL With Integrity Enhancement*
  - 1989 ISO 9075-1989 *Database language SQL*

History II

- ***Embedded SQL***
  - 1989 ANSI X3.168-1989 Database Language Embedded SQL
  - neexistuje ISO standard pro embedde SQL

  Embedded SQL allows for asking SQL queries from a program written in classical host programming language, typically C language

  Queries written directly to C source code.

  Special preprocessor replaces these queries with invoking respective functions that are part of DBMS vendor's libraries.

  Around 1990 frequently used when developing database applications in C languege.

History

- ***SQL92 – aka SQL2:***
  - 1992 ANSI X3.135-1992 *Database language SQL*
  - 1992 ISO/IEC 9075-1992 *Database language SQL*

  - The most frequently used SQL standard.
  - These lectures will focus on SQL 92

Further development

- ***SQL99 – aka SQL3***
  - Regular expressions
  - Recursive queries
  - Non-scalar data types
  - Etc.
- ***2003: SQL2003***
  - XML support
  - Standardized concept of SEQUENCE and automatically generated values
    http://www.wiscorp.com/sql_2003_standard.zip
- ***2006: SQL2006***
  - Extended support of XML (XQuery)
  - Conversions XML ⇔ SQL
  - Export/import XML

SQL – data types I

Numerical data types stored precisely (fixed decimal point)

INTEGER	integers	Range implementationally dependent, typically between -2147483648 and 214783647
SMALLINT	integers	Range implementationally dependent, typicallye -32768 až 32767. Definition: range not bigger than INTEGER.
NUMERIC **NUMERIC(p)** **NUMERIC(p,s)**	Number – possibly with fractional part. Decadic number with *p* digits, *s* of them behind decimal point. E.g. DECIMAL(5,2) has 3 positions on the left of the decimal point and 2 positions on the right of the decimal point. If *p* or *s* not expressed explicitly, a default is use that is dependent on particular implementation. **The number is always stored with the respective precissi**on (unless a „physical“ limit reached)	
DECIMAL **DECIMAL(p)** **DECIMAL(p,s)**	Similar to NUMERIC. However, the number may be stored more precisely that prescribed (unless „physical“ limit reached).	

Remark: NUMERIC – may be stored as string of digits, while DECIMAL in interbal (binary) representation of the respective processor type.

SQL – data types II

Numeric data types stored imprecisely (floating decimal point)

REAL	Floating decimal point – **simple** precission	Precision implementationally dependent Usually **default** precission for data with floating decimal point on given HW platform
DOUBLE PRECISON	Floating decimal point – **double** precission	Precision implementationally dependent Usually **double** precission for data with floating decimal point on given HW platform Definition: precision **greater** than REAL.
FLOAT **FLOAT(p)**	Allows for defining the requested precission. The number **may** be stored with **higher** precision than prescribed. The requested precision *p* may be achieved with simple (default) precision on one platform, whereas with double precision on another platform.	

SQL – data types III

Strings of characters

CHARACTER **CHAR**ACTER**(x)**	String of characters with specified length. If x not expressed, equiv. To CHAR(1).
CHARACTER **VAR**YING **VARCHAR** **CHAR**ACTER **VAR**YING**(x)** **VARCHAR(x)**	Strings wit variable length (alocated so many bytes as needed for particular string). The maximal length limited by *x*. Maximal length depends on particular implementation.
NATIONAL **CHAR**ACTER **NCHAR** **NVARCHAR**	Support for national alphabets. UNICODE

SQL – data types IV

Date and time

DATE	Date: Length 10 chars incl. separators **YYYY-MM-DD**
TIME **TIME(p)**	Time: *p* number of decimal positions (fraction of a second) Length 8 characters if p=0, else 9+p Default 0 decimal positions **HH:MM:SS** **HH:MM:SS.PPP**
TIMESTAMP **TIMESTAMP(p)**	Date + time. Length 19 positions if p=0, else 20+p Default 6 decimal positions **YYYY-MM-DD HH:MM:SS**

CREATE TABLE I

```
CREATE TABLE PACKAGE
    (   PACKID      CHAR(4),
        PACKNAME    CHAR(20),
        PACKVER     DECIMAL(3,2),
        PACKTYPE    CHAR(15),
        PACKCOST    DECIMAL(5,2)    )
```

Name of the table to be created

Column name

Column data type

Creates an empty table with 5 columns as defined.

```
DROP TABLE Computer
```

Name of the table to be removed

Removes an existing table of given nam

CREATE TABLE II (attribute based integrity constraint)

```
CREATE TABLE COMPUTER
    (   COMPID      DECIMAL(2)  NOT NULL,
        MFGNAME     CHAR(15)    NOT NULL,
        MFGMODEL    CHAR(25),
        PROCTYPE    DECIMAL(7,2)    )
```

Integrity constraint (of a column atributu). In this case it says that the value of the respective column is mandatory

When inserting a new row to the table, we need not specify values of all columns. Cells with missing values will get assigned a special value NULL.

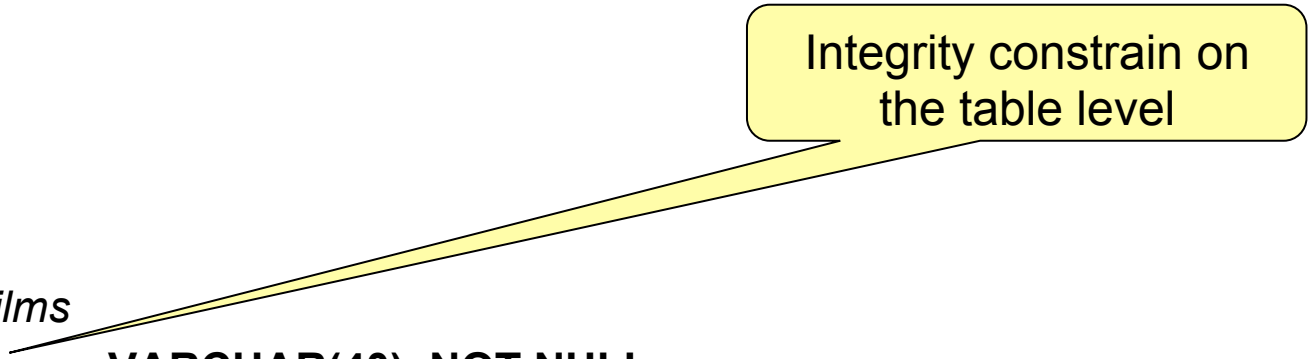
However, if integrity constrain NOT NULL, the value has to be specified mandatorily. If not, DBMS rejects execution of the command and an exception will be raised in the client program.

CREATE TABLE II (attribute based integrity constraint)

```
CREATE TABLE Films
    (   CODE CHAR(5)   CONSTRAINT Firstkey  PRIMARY KEY
        TITLE          VARCHAR(40)  NOT NULL,
        DateProd       DATE,
        KIND           VARCHAR(10),
        LEN            INTERVAL HOUR TO MINUTE  )
```

Integrity constrain may be have got a name

CREATE TABLE IV (table based integrity constraint)




```
CREATE TABLE Films
    (   TITLE           VARCHAR(40)  NOT NULL,
        DateProd        DATE,
        KIND            VARCHAR(10),
        LEN             INTERVAL HOUR TO MINUTE,
        CONSTRAINT pk_const PRIMARY KEY (TITLE, DateProd)
    )
```

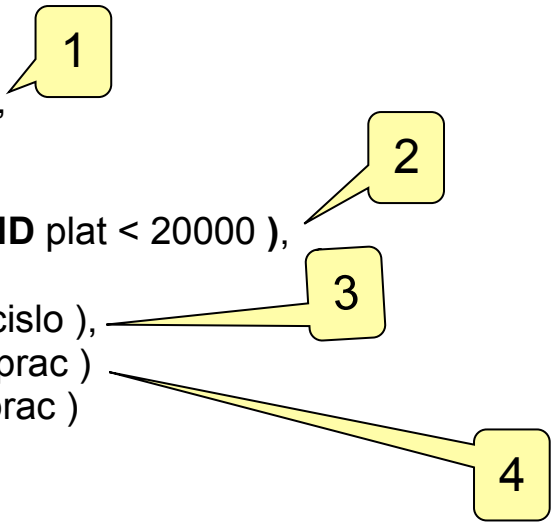
If the primary key consists of multiple attributes, we can not expres it by an attribute based integrity constraint as none of the attributes is primary key. The cpnstraint on primary key has to be expressed by a table-based integrity constraint in such a case.

In the example above, the primary key is formed by a pair (*TITLE*, *DateProd*).

PRIMARY KEY je one of possible table-based integrity constraints.

CREATE TABLE V (integrity constraints)

```
CREATE TABLE osoby (
  os_cislo   NUMBER(5)    NOT NULL,
  rod_cis    VARCHAR2(30) NOT NULL UNIQUE,
  jmeno      VARCHAR2(30) NOT NULL,
  prijmeni   VARCHAR2(30) NOT NULL,
  plat       NUMBER(5)    CHECK ( plat > 5000 AND plat < 20000 ),
  cislo_prac NUMBER(5)    NOT NULL,
  CONSTRAINT pk_osoby PRIMARY KEY ( os_cislo ),
  CONSTRAINT fk_prac  FOREIGN KEY ( cislo_prac )
                      REFERENCES pracoviste ( cislo_prac )
                      ON UPDATE CASCADE
                      ON DELETE CASCADE
)
```



1. Multiple integrity constrains may be specified for a column at the same time.
2. Integrity constrain may have a form of a generic condition
3. Any integrity constrain of a column may be expressed in terpms of an integrity constrain on a table level.
4. Referential integrity

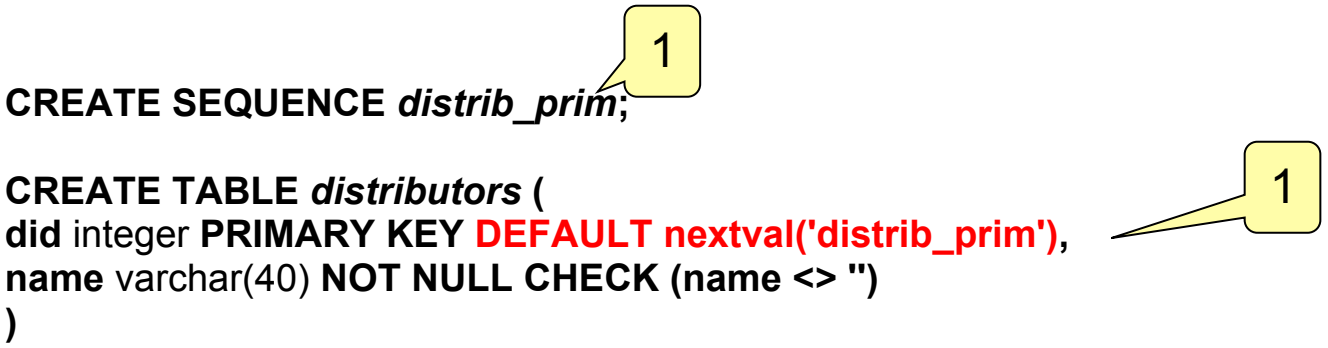
CREATE TABLE VI (integrity constraints)

```
CREATE TABLE COURSES (
  code       VARCHAR2(10) PRIMARY KEY,
  name         VARCHAR2(30) NOT NULL,
  credits       NUMBER(2) DEFAULT 2,
);
```

Default value of an attribute:

If a row will be inserted that does not expressed a value of the „credits“ attribute, its value will not be NULL but 2, as it is its default value (see the CREATE TABLE statement above).

CREATE TABLE VII (generated ids)



1. Definition of a sequence that will be named *distrib_prim* in given case.
2. When inserting a new row (without specifying value for *did*), its default value will be evalueted by *nextval* function with the nase of the sequence as an attribute.

This particular case is not an SQL standard. It is a syntax of DBMS PostgreSQL. Generated values standardized in SQL2006.

REFERENTIAL INTEGRITY

```
CREATE TABLE employee (
  emp_num   NUMBER(5)    PRIMARY KEY,
  ssn    VARCHAR2(30) NOT NULL UNIQUE,
  dept_id NUMBER(5)    NOT NULL,
  CONSTRAINT fk_dept  FOREIGN KEY ( dept_id )
                REFERENCES pracoviste ( dept_id )
                ON UPDATE CASCADE
                ON DELETE CASCADE
)
```

1. Integrity constraint *fk_dept* says, that attribute *dept_id* is a foreign key pointing to such a row of the *department* table, whose primary key's value equals to the value of the *dept_id* attribute.
2. It means that rows representing all persons working in the given department have the same value of the *dept_id* column that refers to their common department.
3. What happens if somebody changes the value of primary key of the departments table? All the rows belonging to this department would suddebly point to a non-existing department. This is what should not happen. *ON UPDATE ...* section specify, how to solve this problem. In the particular case thsi section says *ON UPDATE* ***CASCADE***. This means that the respective *depertment* primary key will be modified and in parallel, the value of foreign key in all rows of table *employee* that point to this department will be updated with the new value of the primary key.

REFERENTIAL INTEGRITY II

```
CREATE TABLE employee (
  emp_num   NUMBER(5)    PRIMARY KEY,
  ssn    VARCHAR2(30) NOT NULL UNIQUE,
  dept_id NUMBER(5)    NOT NULL,
  CONSTRAINT fk_dept  FOREIGN KEY ( dept_id )
                REFERENCES pracoviste ( dept_id )
                ON UPDATE CASCADE
                ON DELETE CASCADE
)
```

4. What happens if somebody deletes a row of the *department* table that represents a common deprtoment of one or more employees? Rows representing these employees would suddenly point to a non-existing depertment. This is what should not happen. The *ON DELETE …* section specifies how this problem shall be handled. In our particular case this section specifies *ON DELETE* ***CASCADE***. This means that the respectice department will be deleted. However, all its employees will be deleted in parallel.

REFERENTIAL INTEGRITY III

```
CREATE TABLE employee (
  emp_num   NUMBER(5)    PRIMARY KEY,
  ssn    VARCHAR2(30) NOT NULL UNIQUE,
  dept_id NUMBER(5)    NOT NULL,
  CONSTRAINT fk_dept  FOREIGN KEY ( dept_id )
                REFERENCES pracoviste ( dept_id )
                ON UPDATE CASCADE
                ON DELETE CASCADE
)
```

What are options other than ***CASCADE***?

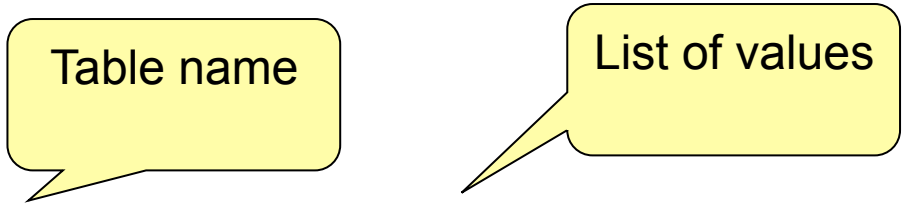
1. RESTRICT – the modification that would violate the referential integrity will not be carried out. DBMS rejects execution of the modification of the primary key (ON UPDATE) or deletion of the row (ON DELETE) – an exception will be thrown.

2. SET NULL – the modification of the *department* table will be executed, but value of the foreign key of all rows of *employee* table that pointed to the deleted department will be set to NULL. It means they will not point to the respective department any more.

3. SET DEFAULT – similar as SET NULL – applicable if the foreign key has defined a default value.

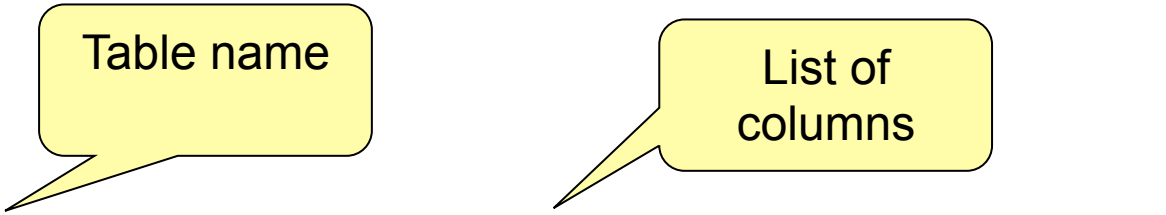
Modifikátory *CASCADE, RSTRICT, SET NULL, SET DEFAULT* se v sekcích *ON UPDATE* a *ON DELETE* nastavují nezávisle.

INSERT INTO



INSERT INTO *EMPLOYEE* **VALUES (** *611, 'Dinh Melissa', 2963* **)**

List of columns not expressed – values will be assigend to columns in the order in which the columns have been defined in the CREATE TABLE statement.



INSERT INTO *EMPLOYEE* **(** *EMPNUM, EMPNAME* **)**
VALUES (*611, 'Dinh Melissa'* **)**

The section VALUES specifies one or more (comma separated) tuples of values. The values will be assigend to columns in the order given by the list of columns.

Columns not introduced in the column list will get the NULL value. In this case EMPPHONE will get NULL.

SELECT I

SELECT *<list of columns or *>*
FROM *<relation definition>*
WHERE *<selection condition>*
GROUP BY *<list of columns>*
HAVING *<group filtering condition>*
ORDER BY *<list of column_defs>*

column_def ::= <column name> [<asc|desc>]

Logical operators:

- = equals
- <= less than or equal
- < les than
- >= greater than or equal
- > greater than
- <> not equal
- != not equal

SELECT II

PACKAGE table

PACKID	PACKNAME	PACKVER	PACKTYPE	PACKCOST
AC01	Boise Accounting	3.00	Accounting	725.83
DB32	Manta	1.50	Database	380.00
DB33	Manta	2.10	Database	430.18
SS11	Limitless View	5.30	Spreadsheet	217.95
WP08	Words & More	2.00	Word Processing	185.00
WP09	Freeware Processing	4.27	Word Processing	30.00

```
SELECT PACKID, PACKNAME, PACKCOST
      FROM PACKAGE
      WHERE   PACKCOST >= 200 AND
                  PACKCOST <= 400
```

```
SELECT PACKID, PACKNAME, PACKCOST
      FROM PACKAGE
      WHERE PACKCOST
                  BETWEEN 200 AND 400
```

Result:

PACKID	PACKNAME	PACKCOST
DB32	Manta	380.00
SS11	Limitless View	217.95

SELECT III

PACKAGE table

PACKID	PACKNAME	PACKVER	PACKTYPE	PACKCOST
AC01	Boise Accounting	3.00	Accounting	725.83
DB32	Manta	1.50	Database	380.00
DB33	Manta	2.10	Database	430.18
SS11	Limitless View	5.30	Spreadsheet	217.95
WP08	Words & More	2.00	Word Processing	185.00
WP09	Freeware Processing	4.27	Word Processing	30.00

boolean predicate LIKE
character % is a wildcard, i.e. matches with any charatcter (sub)string

SELECT *PACKID, PACKNAME*
FROM *PACKAGE*
WHERE *PACKNAME* ***LIKE*** *'%&%'*

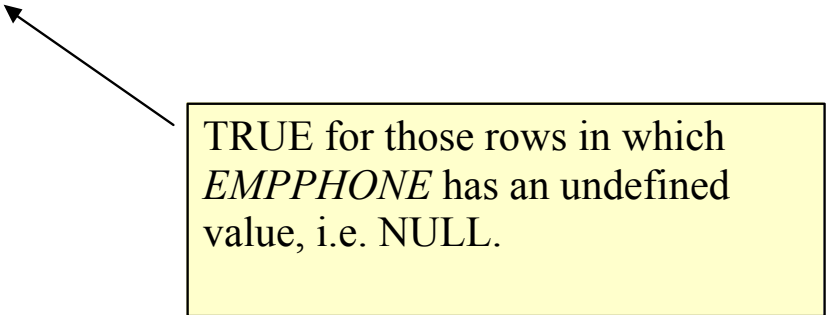
Result:

PACKID	PACKNAME
WP08	Words & More

SELECT IV

Precate **"IS NULL"** is equal to **„true“** iff the respective coulmn has assigned no value

SELECT *EMPNUM, EMPNAME*
FROM *EMPLOYEE*
WHERE *EMPPHONE* **IS NULL**



SELECT V (arithemetic operators)

PACKAGE table

PACKID	PACKNAME	PACKVER	PACKTYPE	PACKCOST
AC01	Boise Accounting	3.00	Accounting	725.83
DB32	Manta	1.50	Database	380.00
DB33	Manta	2.10	Database	430.18
SS11	Limitless View	5.30	Spreadsheet	217.95
WP08	Words & More	2.00	Word Processing	185.00
WP09	Freeware Processing	4.27	Word Processing	30.00

SELECT PACKID, PACKNAME**, (.90 *** PACKCOST **)**
FROM **PACKAGE**

Name of this column generated by DBMS client

Result:

PACKID	PACKNAME	EXP1
AC01	Boise Accounting	635.25
DB32	Manta	342.00
DB33	Manta	387.16
SS11	Limitless View	196.16
WP08	Words & More	166.50
WP09	Freeware Processing	27.00

i.e. 0.9 * 725.83
0.9 * 380.00
0.9 * 430.18

SELECT VI

PACKAGE table

PACKID	PACKNAME	PACKVER	PACKTYPE	PACKCOST
AC01	Boise Accounting	3.00	Accounting	725.83
DB32	Manta	1.50	Database	380.00
DB33	Manta	2.10	Database	430.18
SS11	Limitless View	5.30	Spreadsheet	217.95
WP08	Words & More	2.00	Word Processing	185.00
WP09	Freeware Processing	4.27	Word Processing	30.00

```
SELECT PACKID, PACKNAME, PACKTYPE
FROM PACKAGE
WHERE PACKTYPE IN ('Database',
                   'Spreadsheet',
                   'Word Processing')
```

```
SELECT PACKID, PACKNAME,
PACKTYPE
FROM PACKAGE
WHERE PACKTYPE = 'Database' OR
      PACKTYPE = Spreadsheet' OR
      PACKTYPE = 'Word Processing'
```

Result:

PACKID	PACKNAME	PACKTYPE
DB32	Manta	Database
DB33	Manta	Database
SS11	Limitless View	Spreadsheet
WP08	Words & More	Word Processing
WP09	Freeware Processing	Word Processing

SELECT VII (sorting)

PACKAGE table

PACKID	PACKNAME	PACKVER	PACKTYPE	PACKCOST
AC01	Boise Accounting	3.00	Accounting	725.83
DB32	Manta	1.50	Database	380.00
DB33	Manta	2.10	Database	430.18
SS11	Limitless View	5.30	Spreadsheet	217.95
WP08	Words & More	2.00	Word Processing	185.00
WP09	Freeware Processing	4.27	Word Processing	30.00

Order of rows in the result of the query is undefined unless specified by OREDR clause:

SELECT *PACKID, PACKNAME, PACKTYPE, PACKCOST*
FORM *PACKAGE*
ORDER BY *PACKTYPE, PACKCOST* **DESC**

Rows will be oredered primarily by *PACKTYPE.* The order of rows with an equal value of *PACKTYPE* will be defined by *PACKCOST*.
DESC ... descending
ASC ... ascending (default)

Result:

PACKID	PACKNAME	PACKTYPE	PACKCOST
AC01	Boise	Accounting	725.83
DB33	Manta	Database	430.18
DB32	Manta	Database	380.00
SS11	Limitless View	Spreadsheet	217.95
WP08	Words & More	Word Processing	185.00
WP09	Freeware Processing	Word Processing	30.00