Historia

- Język IBM Sequel (*Structured English Query Language*)
  - część projektu Systemu R w IBM San Jose Research Laboratory (1973)
- Zmiana nazwy na *Structured Query Language* (SQL)
  - nazwa *Sequel* zastrzeżona wcześniej przez brytyjską firmę lotniczą Hawker Siddeley
- Standardy SQL: ANSI i ISO:
  - SQL-86
  - SQL-89
  - SQL-92 (zwany SQL2 – rozszerzenia)
  - SQL:1999 (zwany SQL3 – core i extensions)
  - SQL:2003, SQL:2006 – dodaje XML
  - SQL:2008 – obiektowość dodana
  - SQL:2011, SQL:2016
- Systemy komercyjne oferują większość, jeśli nie wszystkie funkcje SQL-92, a także różne zestawy funkcji z późniejszych standardów i specjalne funkcje zastrzeżone.
  - T-SQL, PL/SQL. SQL/PM, PL/pgSQL

SQL - części

- *DDL* – definiowanie/usuwanie/modyfikowanie schematów relacji,
- *DML* - wyszukiwania informacji w BD i wstawiania/usuwanie/modyfikowania krotek w BD.
- *Integralność* – polecenia DDL do specyfikacji warunków integralności.
- *Definiowanie widoków* – polecenia DDL do definiowania widoków.
- *Kontrola transakcji* –polecenia do specyfikowania początku i końca transakcji.
- *Wbudowany i dynamiczny SQL* – definiuje jak wyrażenia SQL mogą być wbudowane w języki programowania ogólnego przeznaczenia (C/C++, Java).
- *Autoryzacja* – polecenia określające prawa dostępu do relacji i widoków.

Data Definition Language

SQL DDL pozwala na specyfikację informacji o relacjach, w tym :

- Schemat dla każdej relacji.
- Typy wartości powiązane z każdym atrybutem.
- Warunki integralnościowe
- Zbiór indeksów dla każdej relacji.
- Informacje na temat bezpieczeństwa i autoryzacji dla każdej relacji.
- Fizyczna struktura przechowywania każdej relacji na dysku.

Definiowanie tabel

- Relacja definiowana przy pomocy polecenia:

create table *r*
(A_1 D_1,
A_2 D_2,
...,
A_n D_n,
(integrity-constraint$_1$),
...,
(integrity-constraint$_k$))

  - *r* - nazwa relacji
  - każde A_i jest nazwą atrybutu w schemacie relacji *r*
  - D_i jest typem danych wartości atrybutu A_i

- Przykład:

create table *instructor* (
ID **char**(5),
name **varchar**(20),
dept_name **varchar**(20),
salary **numeric**(8,2))

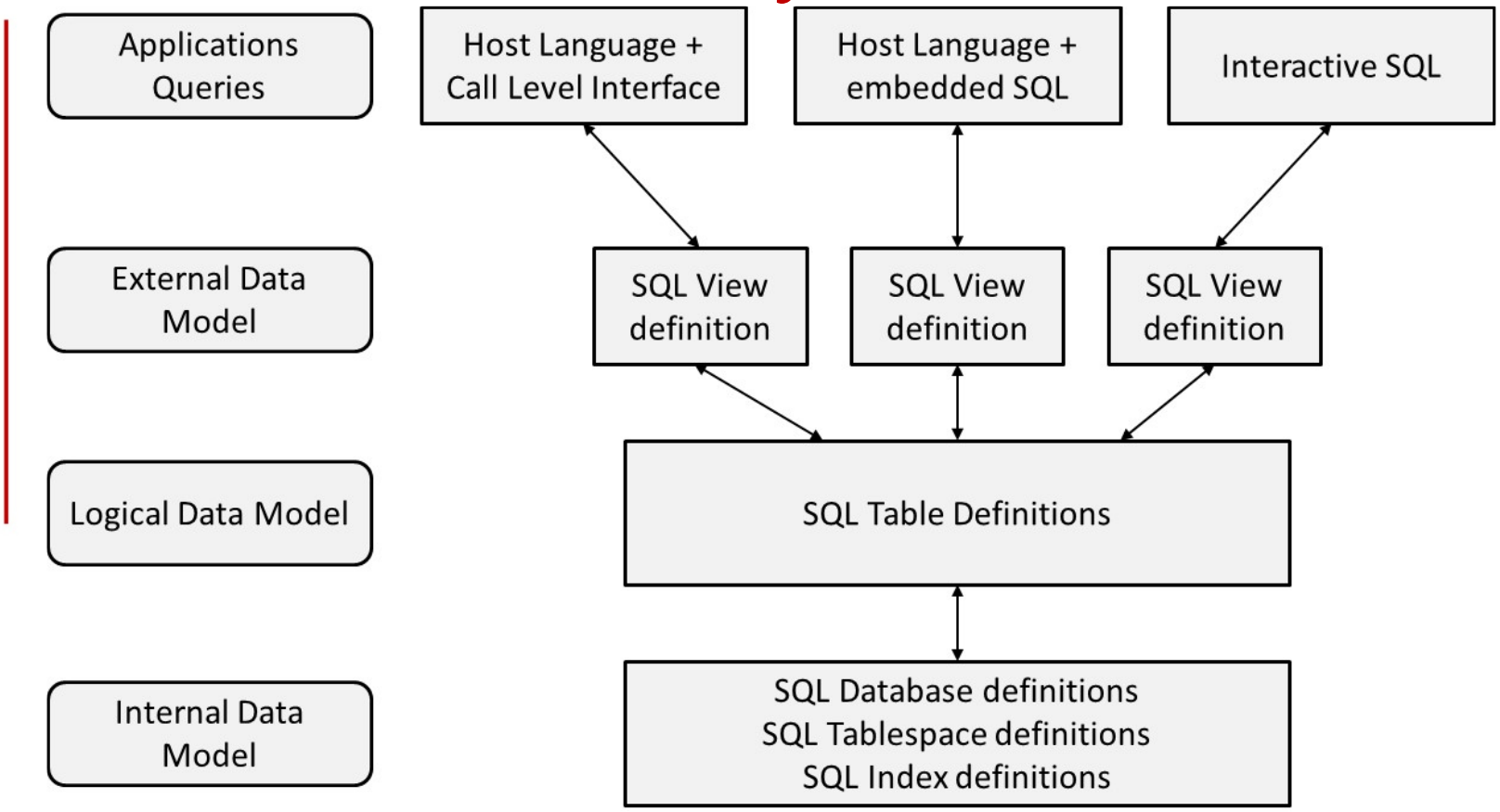
- **insert into** *instructor* **values** (‘10211’, ’Smith’, ’Biology’, 66000);
- **insert into** *instructor* **values** (‘10211’, null, ’Biology’, 66000);

Warunki integralności w *Create Table*

- Rodzaje warunków integralności:
  - **primary key** (A_1, ..., A_n)
  - **foreign key** (A_m, ..., A_n) **references** s
  - **not null**
- SQL zapobiega wszelkim aktualizacjom bazy danych, które naruszają ograniczenia integralności.
- Np:

```
create table instructor (
        ID              char(5),
        name            varchar(20) not null,
        dept_name       varchar(20),
        salary          numeric(8,2),
        primary key (ID),
        foreign key (dept_name) references
          department);
```

Trójwarstwowa architektura baz danych



Kluczowe pojęcia DDL

- Schemat SQL: grupowanie tabel i innych obiektów bazy danych, takich jak widoki, ograniczenia i indeksy, które są logicznie powiązane

```
CREATE SCHEMA PURCHASE AUTHORIZATION BBAESENS
```

- Tabela SQL implementuje relację z modelu relacyjnego

```
CREATE TABLE PRODUCT …
  CREATE TABLE PURCHASE.PRODUCT …
```

Kluczowe pojęcia DDL

```
CREATE DOMAIN PRODTYPE_DOMAIN AS VARCHAR(10)
CHECK (VALUE IN ('white', 'red', 'rose',
'sparkling'))
```

Kluczowe pojęcia DDL

- Ograniczenia kolumn
  - **PRIMARY KEY** ograniczenie definiuje klucz główny tabeli
  - **FOREIGN KEY** ograniczenie definiuje klucz obcy tabeli
  - **UNIQUE** ograniczenie definiuje klucz alternatywny tabeli
  - **NOT NULL** ograniczenie zabrania wartości NULL dla kolumny
  - **DEFAULT** ograniczenie ustawia domyślną wartość dla kolumny
  - **CHECK** ograniczenie definiuje ograniczenie na wartościach kolumn

Polecenie DROP i ALTER

- **DROP** do usuwania obiektów bazy danych
  - może być łączona z CASCADE i RESTRICT
- Np.:

```
DROP SCHEMA PURCHASE CASCADE
DROP SCHEMA PURCHASE RESTRICT
DROP TABLE PRODUCT CASCADE
DROP TABLE PRODUCT RESTRICT
```

Polecenie DROP i ALTER

- **ALTER i**nstrukcja może być użyta do modyfikacji definicji kolumn tabeli
- Np.:

```
ALTER TABLE PRODUCT ADD PRODIMAGE BLOB
```

```
ALTER TABLE SUPPLIER ALTER SUPSTATUS SET
  DEFAULT ‚10’
```

ALTER TABLE *r* **drop** *A*

- gdzie *A* jest nazwą atrybutu relacji *r*
- przez wiele baz danych usuwanie atrybutów nie jest wspierane.

Polecenie DROP i ALTER

SUPPLIER

SUPNR	SUPNAME	SUPADDRESS	SUPCITY	SUPSTATUS
21	Deliwines	240, Avenue of the Americas	New York	20
32	Best Wines	660, Market Street	San Francisco	90
…				

PRODUCT

PRODNR	PRODNAME	PRODTYPE	AVAILABLE_QUANTITY
0119	Chateau Miraval, Cotes de Provence Rose, 2015	rose	126
0154	Chateau Haut Brion, 2008	red	111
…		red	5

SUPPLIES

SUPNR	*PRODNR*	PURCHASE_PRICE	DELIV_PERIOD
21	0289	17.99	1
21	0327	56.00	6
…			

PURCHASE_ORDER

PONR	PODATE	*SUPNR*
1511	2015-03-24	37
1512	2015-04-10	94
…		

PO_LINE

PONR	*PRODNR*	QUANTITY
1511	0212	2
1511	0345	4
…		

Modyfikacje bazy danych

- Usuwanie krotek z danej relacji
- Wstawianie nowych krotek do danej relacji
- Aktualizacja wartości w niektórych krotkach w danej relacji

Usuwanie

- Usuń wszystkich instruktorów

```
delete from instructor
```

- Usuń wszystkich instruktorów z wydziału Finansów

```
delete from instructor
where dept_name= 'Finance';
```

- Usuń wszystkie krotki z relacji *instructor* dla tych instruktorów, którzy powiązani są z wydziałem znajdującym się w budynku Watson.

```
delete from instructor
   where dept name in (select dept name
                       from department
                       where building = 'Watson');
```

Usuwanie c.d.

» Usuń wszystkich instruktorów, których wynagrodzenie jest niższe niż średnie wynagrodzenie instruktorów

delete from *instructor*
where *salary* < (**select avg** (*salary*)
from *instructor*);

- Problem: usuwanie krotek spowodowałoby zmianę średniej pensji
- Rozwiązanie w SQL:

1. Najpierw wylicz **avg** (salary) i znajdź wszystkie krotki do usunięcia

2. Następnie usuń wszystkie znalezione (bez ponownego liczenia **avg** lub ponownego sprawdzenia krotek)

Wstawianie

- Dodaj nową krotkę do *course*

```
insert into course
    values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- lub równoważnie

```
insert into course (course_id, title, dept_name, credits)
  values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- Dodaj nową krotkę do *student* z *tot_creds* set ustawioną na null

```
insert into student
    values ('3003', 'Green', 'Finance', null);
```

Wstawianie c.d.

- Niech każdy student z wydziału Muzycznego, który zrobił więcej niż 144 godziny zostanie instruktorem na tym wydziale z pensją $18,000.

```
insert into instructor
   select ID, name, dept_name, 18000
    from   student
    where   dept_name = 'Music' and total_cred > 144;
```

- Wyrażenie **select from where** jest w pełni oceniane, zanim którykolwiek z jego wyników zostanie wstawiony do relacji
- W przeciwnym razie zapytania, takie jak
  insert into *table*1 **select** * **from** *table*1
  spowodowałyby problem
- „bulk loader” – w większości BD

Modyfikacje

- Daj podwyżkę wynagrodzenia o 5% wszystkim instruktorom

```
update instructor
   set salary = salary * 1.05
```

- Daj 5% podwyżkę wynagrodzenia instruktorom, którzy zarabiają mniej niż 70000

```
update instructor
   set salary = salary * 1.05
  where salary < 70000;
```

- Daj podwyżkę wynagrodzenia o 5% instruktorom, których wynagrodzenie jest niższe niż średnia

```
update instructor
   set salary = salary * 1.05
   where salary <  (select avg (salary)
                    from instructor);
```

Modyfikacje c.d.

- Zwiększ pensje instruktorów, których pensja przekracza 100 000 $ o 3%, a wszystkich innych o 5%
- Dwa zapytania **update**:

```
update instructor
    set salary = salary * 1.03
    where salary > 100000;
update instructor
    set salary = salary * 1.05
    where salary <= 100000;
```

  - Porządek istotny
  - Lepiej za pomocą wyrażenia **case**

Wyrażenie Case dla warunkowych modyfikacji

- To samo z wyrażeniem *case*

```
update instructor
    set salary = case
                     when salary <= 100000
                   then salary * 1.05
                     else salary * 1.03
                    end
```

```
case
   when pred1 then result1
    when pred2 then result2
   ...
    when predn then resultn
   else result0
end
```

Modyfikacje ze skalarnymi podzapytaniami

- Przelicz i zmodyfikuje wartość tot_creds dla wszystkich studentów

```
update student S
set tot_cred = (select sum(credits)
                from takes, course
                where takes.course_id = course.course_id and
                      S.ID= takes.ID.and
                            takes.grade <> 'F' and
                      takes.grade is not null);
```

- Ustawia *tot_creds* na *null* dla studentów, którzy nie ukończyli żadnego kursu
- Zamiast **select sum**(*credits*):

```
case
   when sum(credits) is not null then sum(credits)
   else 0
end
```

Klauzula With (SQL:1999)

- Klauzula **with** pozwala definiować tymczasowe relacje, których definicja jest dostępna tylko dla zapytania, w którym klauzula **with** pojawia się.
- Znajdź wszystkie wydziały z maksymalnym budżetem

```
with max_budget (value) as
    (select max(budget)
     from department)

select budget
from department, max_budget
where department.budget = max_budget.value;
```

Złożone zapytania w klauzuli With

- Klauzula *with* przydatna do pisania złożonych zapytań
- Wspierana przez większość systemów bazodanowych, z drobnymi różnicami syntaktycznymi
- Znajdź wszystkie wydziały, w których suma pensji jest większa niż średnia suma pensji we wszystkich wydziałach

```
with dept _total (dept_name, value) as
        (select dept_name, sum(salary)
         from instructor
         group by dept_name),
dept_total_avg(value) as
       (select avg(value)
       from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value >= dept_total_avg.value;
```

Warunki integralnościowe

- Chronią przed przypadkowym uszkodzeniem BD, zapewniając, że autoryzowane zmiany w bazie danych nie spowodują utraty spójności danych.
  - Rachunek bieżący musi mieć saldo większe niż $10,000.00
  - Wynagrodzenie pracownika banku musi wynosić co najmniej $4.00 na godzinę
  - Klient musi mieć (inny niż not-null) numer telefonu

Warunki integralnościowe na pojedynczej relacji

- **not null**
- **primary key**
- **unique**
- **default**
- **check** (P), gdzie P jest predykatem

- część definicji tabeli **create table** albo
- dodawane później: **alter table** *table-name* **add** *constraint*
  - gdzie *constraint* – dowolny warunek integralnościowy na relacji

- inna forma warunków integralnościowych - zależności funkcyjne

Warunek NOT NULL i UNIQUE

- **not null**
  - Deklaracja, że *name* i *budget* muszą byc **not null**

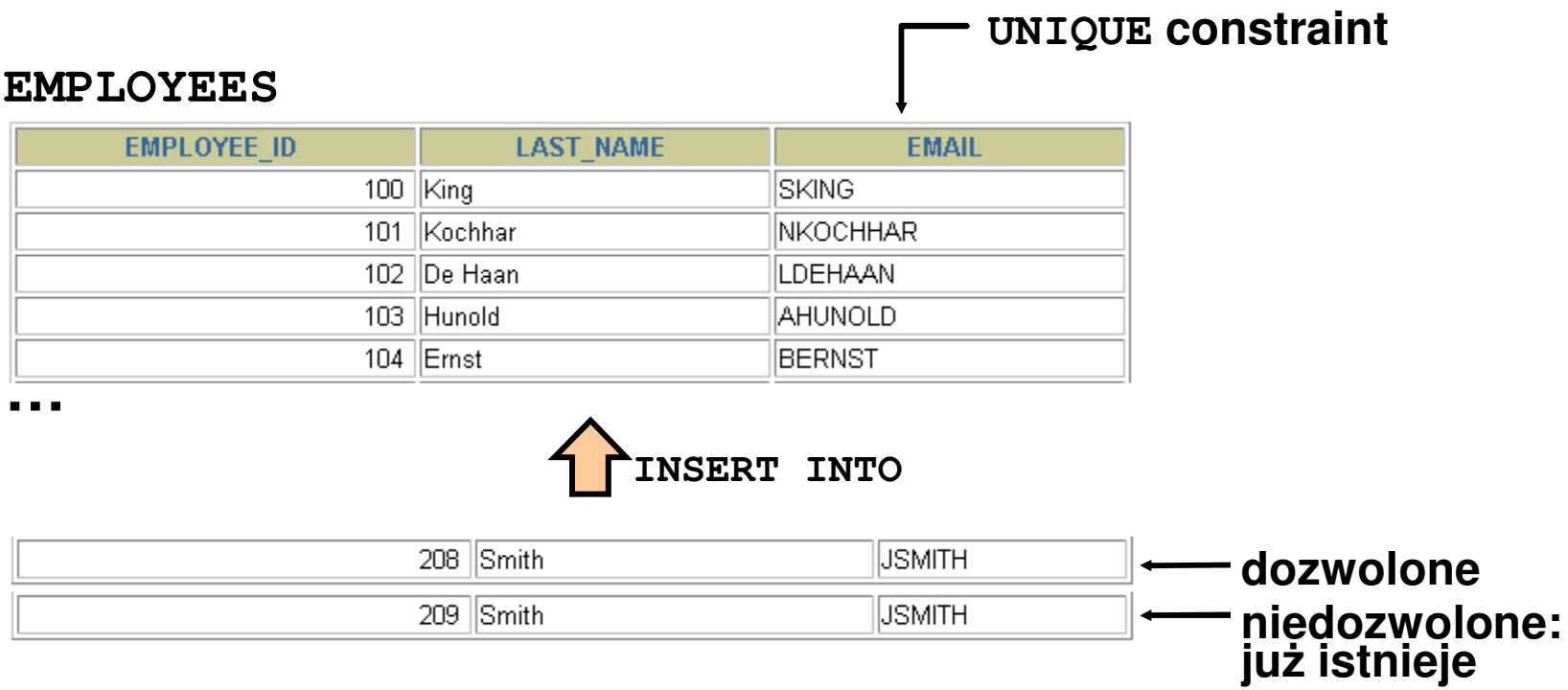
    name **varchar**(20) **not null**
    budget **numeric**(12,2) **not null**
- **unique** (A_1, A_2, …, A_m)
  - Specyfikacja **unique** stwierdza, że atrybuty A_1, A_2, … A_m tworzą klucz kandydujący
  - Klucze kandydujące mogą być null (w przeciwieństwie do kluczy głównych).

Warunek UNIQUE

- Dopuszcza jedną wartość *null*
- Można tworzyć wiele warunków UNIQUE na tabelę
- Definiowane na jednej lub więcej kolumn

```
USE Northwind
ALTER TABLE dbo.Suppliers
ADD
CONSTRAINT U_CompanyName
  UNIQUE NONCLUSTERED (CompanyName)
```

Warunek UNIQUE



Warunek DEFAULT

- **create table** *student*

  (ID **varchar** (5)**,**

  name **varchar** (20) **not null,**

  dept_name **varchar** (20)

  tot_cred **numeric** (3,0) **default** 0**,**

  primary key (ID))

- **insert into** *student(ID,name,dept_name)*

  values (12789, 'Newman','Comp Sci.')

Warunek CHECK

- Klauzula **check** (P) określa predykat P, który musi być spełniony przez każdą krotkę relacji

Przykład: sprawdzenie, że semestr może być jesienny, zimowy, wiosenny, letni:

```
create table section (
    course_id varchar (8),
    sec_id varchar (8),
    semester varchar (6),
    year numeric (4,0),
    building varchar (15),
    room_number varchar (7),
    time slot id varchar (4),
    primary key (course_id, sec_id, semester, year),
    check (semester in ('Fall', 'Winter', 'Spring', 'Summer'))
);
```

Warunek CHECK

- Używany z INSERT i UPDATE
- Tabela/kolumna może zawierać wiele *check*
- Aby zmodyfikować istniejący *check*, trzeba go usunąć i utworzyć od nowa
- Może odnosić się do innych kolumn w tej samej tabeli
  - Np. sprawdzanie czy jeżeli klient jest z USA, to ma 2-literowy kod w state
- Nie może:
  - Zawierać podzapytań

```
USE Northwind
ALTER TABLE dbo.Employees
ADD
CONSTRAINT CK_birthdate
CHECK (BirthDate > '01-01-1900' AND BirthDate <
getdate())
```

Zależność referencyjna

- Zapewnia, że wartość pojawiająca się w jednej relacji dla danego zestawu atrybutów pojawia się również dla pewnego zestawu atrybutów w innej relacji.
  - Np. Jeżeli “Biology” jest nazwą wydziału pojawiającą się w jednej z krotek relacji *instructor*, to istnieje krotka w relacji *department* dla “Biology”.
- Niech A będzie zbiorem atrybutów. Niech R i S będą dwiema relacjami zawierającymi atrybuty A i A jest kluczem głównym S. A jest **kluczem obcym** w R jeśli dla dowolnej wartości A występującej w R wartości te pojawiają się również w S.

Zależność referencyjna c.d.

- *Klucze obce* mogą być specyfikowane jako część wyrażenia SQL **create table**

  foreign key (*dept_name*) **references** *department*

- Domyślnie, klucz obcy odwołuje się do atrybutów klucza głównego tabeli, do której istnieje odwołanie.
- SQL pozwala na jawne określenie listy atrybutów relacji, do której istnieje odwołanie.

  foreign key (*dept_name*) **references** *department* (*dept_name*)

Zależność referencyjna

- Klucz obcy ma tę samą dziedzinę, co klucz główny, do którego się odnosi i albo występuje jako jego wartość, albo NULL
- Co dzieje się z kluczami obcymi po zaktualizowaniu lub usunięciu klucza głównego?
- Możliwości:
  - **ON UPDATE/DELETE CASCADE**: aktualizacja/usunięcie powinno być kaskadowane do wszystkich odnoszących się krotek
  - **ON UPDATE/DELETE RESTRICT:** aktualizacja/usuwanie jest zatrzymywane, jeśli istnieją krotki odnoszące się
  - **ON UPDATE/DELETE SET NULL:** klucze obce w odsyłających krotkach są ustawiane na NULL
  - **ON UPDATE/DELETE SET DEFAULT:** klucze obce w odsyłających krotkach są ustawione na ich domyślną wartość