Systemy Baz Danych

Zaawansowany SQL

Bartosz Zieliński

WYDZIAŁ FIZYKI
i INFORMATYKI STOSOWANEJ
Uniwersytet Łódzki

SQL a Model Relacyjny

Język zapytań (operacja **SELECT** w SQL jest wzorowany na kombinacji elementów z rachunku relacyjnego i algebry relacyjnej.

Należy jednak pamiętać że model danych zakładany przez SQL różni się nieco od czystego modelu relacyjnego:

- Zduplikowane wiersze: zmienne relacyjne przechowują, a zapytania zwracają multizbiory krotek.
- W "relacjach" zwracanych przez zapytania SQL kilka atrybutów może mieć te same nazwy, mogą też wystąpić atrybuty bez nazwy.
- W SQL kolejność kolumn w definicji zmiennej relacyjnej lub w zapytaniu może być istotna (inaczej niż w modelu relacyjnym).
- Niektóre operacje w SQL noszące nazwy operacji w algebrze relacyjnej mogą się różnić działaniem od swoich odpowiedników w algebrze.

Zapytanie SELECT z Lotu Ptaka

SELECT ⟨*lista wyrażeń definiujących atrybuty wynikowej relacji*⟩
FROM ⟨*opis zmiennych relacyjnych z których korzysta zapytanie*⟩
WHERE ⟨*(Opcj.) warunek selekcji (przed agregacją)*⟩
GROUP BY ⟨*(Opcj.) lista wyrażeń definiujących grupy przy agregacji*⟩
HAVING ⟨*(Opcj.) warunek selekcji (po agregacji)*⟩
ORDER BY ⟨*(Opcj.) opis sortowania wyników*⟩

To nie wszystkie elementy które mogą wystąpić w zapytaniu **SELECT**. Pomijamy klauzule **WITH** definiujące (także rekurencyjnie) lokalne nazywane podzapytania, hierarchiczne klauzule **CONNECT BY**, klauzule obracające kolumny (**PIVOT** i **UNPIVOT**), i wiele innych.

Zapytanie Zwracające Zawartość Wybranej Tabeli

SELECT *
FROM *NazwaZmiennejRelacyjnej*;

Uwaga

- Biblioteki programistyczne takie jak **JDBC** obsługujące operacje bazodanowe pozwalają na odwoływanie się do atrybutów zwracanych przez zapytanie krotek tak przez nazwę jak i przez numer kolejny.
- W przypadku zapytania powyżej kolejność (i nazwy) atrybutów są takie same jak w **CREATE TABLE** *NazwaZmiennejRelacyjnej* (...).

SELECT DISTINCT ... FROM ... WHERE ...

SELECT DISTINCT E_1 **AS** A_1, E_2 **AS** A_2, ..., E_n **AS** A_n
FROM R_1 t_1, R_2 t_2, ..., R_m t_m
WHERE *Warunek*;

- Relacje R_i mogą być zarówno odwołaniami do zmiennych relacyjnych jak i podzapytaniami **SELECT**.
- t_i nazywane są **aliasami tabel** lub **zmiennymi krotkowymi**.
- A_i nazywane są **aliasami kolumn**
- Pozycja atrybutu zwracanej krotki (przy pozycyjnych odwołaniach) jest taka sama jak na liście **SELECT**

Przykład SELECT DISTINCT ... FROM ... WHERE ...

Przykładowe zapytanie

```
1  SELECT DISTINCT e.FirstName||e.LastName AS Imie,
2         d.NazwaWydzialu AS Wydzial
3  FROM Employees e, (
4     SELECT d1.DepartmentName AS NazwaWydzialu,
5            d1.DepartmentId AS IDWydzialu
6     FROM Departments d;
7  ) d WHERE d.IDWydzialu = e.DepartmentId;
```

Rola DISTINCT

```
SELECT p.Y AS Z FROM R p;
```

Oznaczmy relację zwracaną przez powyższe zapytanie przez R' Jeśli

R jest dana przez

X	Y
1	10
2	10
3	20
4	20

to R' jest równa

Z
10
10
20
20

Projekcja w SQL domyślnie nie usuwa duplikatów! Aby wymusić usuwanie duplikatów trzeba skorzystać z **DISTINCT**

- Często nie jest to konieczne (a jest kosztowne), dlatego w dalszych przykładach będziemy pomijać **DISTINCT**.

Formy Uproszczone Zapytań

Pomijanie Nowej Nazwy Atrybutu

- Jeśli na liście **SELECT** pominiemy przemianowanie dla prostego odwołania do atrybutu A wówczas nazwą odpowiadającego atrybutu w wyniku zapytania będzie A.
- Jeśli nie nadamy nazwy atrybutowi zdefiniowanemu złożonym wyrażeniem nie będzie on miał nazwy. Jest to dozwolone w zewnętrznych zapytaniach (aplikacja może odwoływać się do kolumn wyniku pozycyjnie) ale nie w podzapytaniach

```
SELECT p.FirstName, p.Salary,
       p.Salary*0.2 AS Podatek,
       p.FirstName||p.LastName
FROM Employees p;
```

Powyższe zapytanie zwraca relację o atrybutach **FirstName**, **Salary** i **Podatek** oraz atrybut bez nazwy na czwartej pozycji.

Formy Uproszczone Zapytań
Pomijanie Zmiennych Krotkowych

Gdy nie prowadzi to do niejednoznaczności można nie deklarować **zmiennych krotkowych** dla **zmiennych relacyjnych** i odwoływać się do ich atrybutów bezpośrednio. **Zmienne krotkowe** są jednak **obowiązkowe** dla **podzapytań**.

```
SELECT FirstName, LastName, Salary, 0.2*Salary AS Tax
FROM Employees;
```

Gdy korzystamy z **różnych** zmiennych relacyjnych zamiast **zmiennymi krotkowymi** można kwalifikować nazwy atrybutów nazwami tabel.

```
SELECT Employees.Name, Departments.Name
FROM Employees, Departments;
```

Operatory Boolowskie i Atomowe Predykaty

- Warunki **WHERE** można budować łącząc atomowe predykaty przy pomocy operatorów boolowskich **AND**, **OR** i **NOT**.
- Wśród **atomowych predykatów** można wyróżnić operatory porównania =, <>, <, <=, >, >=, gdzie = i <> działają także dla łańcuchów znaków.
- Operator *napis* **LIKE** *wzorzec* porównuje *napis* ze *wzorcem*. We wzorcu można stosować znaki specjalne takie jak "%" (zero lub więcej dowolnych znaków) i "_" (pojedyńczy dowolny znak).
  - Np. zachodzi `'Berlin' LIKE '_erl%'`.
  - `x NOT LIKE y` to to samo co `NOT(x LIKE y)`
- *x* **BETWEEN** *a* **AND** *b* to to samo co `x>=a` **AND** `x<=b`.
- Sprawdzanie **NULL**-owatości: *x* **IS NULL** lub *x* **IS NOT NULL**.
- Sprawdzanie czy *x* należy do zbioru wartości: *x* **IN** $(v_1, v_2, \ldots, v_n)$. Można też skorzystać z podzapytania.

Operator IN z Podzapytaniem

```
SELECT * FROM Empls e
WHERE e.JobId IN (
    SELECT j.JobId FROM Jobs j
    WHERE j.MinSal >= 10000
)
```

Ważne

- Podzapytanie wewnątrz **IN** musi zwrócić relację z **pojedyńczym** atrybutem.
- Nazwa tego atrybutu nie ma znaczenia — może jej nawet nie być. Wynik podzapytania jest traktowany jako **zbiór wartości** a nie jako **relacja**.
- Te same uwagi dotyczą użycia podzapytań także w innych przypadkach gdy chcemy uzyskać **zbiór wartości**, np. wewnątrz operatorów **ALL** i **ANY**.

Operatory Oczekujące Zbioru: ALL i ANY

Z reguły poniższe operatory można zastąpić kombinacjami wywołań innych operatorów:

- $x > \mathbf{ANY}(M)$ to samo co $x > m$ dla któregoś $m \in M$
- $x > \mathbf{ALL}(M)$ to samo co $x > m$ dla każdego $m \in M$

Zamiast $>$ można skorzystać z innych operatorów porównania.

```
SELECT * FROM Empl e WHERE e.sal < ANY(1000,5000);
```

```
SELECT * FROM Empl e WHERE e.sal >= ALL(
   SELECT j.MinSal FROM Jobs j
);
```

Podzapytania Jako Wartości

Wszędzie tam gdzie w zapytaniu oczekiwana jest (pojedyńcza) wartość można użyć podzapytania zwracającego **pojedyńczą** krotkę z **pojedyńczym** atrybutem. Nazwa atrybutu nie jest istotna i może jej nie być.

Podzapytanie jako wartość w WHERE

```
SELECT * FROM Empls e WHERE e.sal = (
   SELECT 2*f.sal FROM Empls f WHERE f.Id=10
);
```

Podzapytanie jako wartość na liście SELECT

```
SELECT e.Name, (
   SELECT f.Name FROM Empls f WHERE f.Id=10
) AS BossName FROM Empls e;
```

Podzapytania Skorelowane

Podzapytania (**nie definiujące** źródłowych relacji w liście **FROM**) mogą zostać skorelowane z nad-zapytaniem przez odwołanie się do **zmiennych krotkowych** nad-zapytania.

- Podzapytania skorelowane można sobie wyobrazić jako wykonywane osobno dla każdej krotki zwracanej przez nad-zapytanie.
- Jeśli podzapytanie jest używane jako wartość wówczas powinno ono zwrócić **pojedyńczy wiersz** dla każdej krotki zapytania otaczającego.

Zapytanie skorelowane na liście SELECT

```
SELECT e.Name, (
    SELECT f.Name FROM Empls f
    WHERE f.Id = e.ManagerId
) AS BossName FROM Empls e;
```

Podzapytania Skorelowane

Inne Przykłady

Podzapytanie skorelowane wewnątrz operatora IN w WHERE

```
SELECT * FROM Empls e
WHERE e.JobId IN (
    SELECT j.JobId FROM Jobs j
    WHERE j.MinSal = e.sal
)
```

Podzapytanie skorelowane wewnątrz WHERE

```
SELECT * FROM Empls e
WHERE e.sal < (
    SELECT j.MinSal FROM Jobs j
    WHERE j.JobId=e.JobId
);
```

Operatory EXISTS i NOT EXISTS

Niech Q będzie podzapytaniem (zwracającym krotki o dowolnej ilości kolumn, niekoniecznie nazwanych. Wtedy $\mathbf{EXISTS}(Q)$ jest spełnione gdy relacja zdefiniowana przez Q jest niepusta.

- Użyteczne głównie gdy Q jest skorelowane z zewnętrznym zapytaniem.

```
SELECT * FROM Empls e WHERE EXISTS(
  SELECT * FROM Jobs j WHERE j.JobId=e.JobId
);
```

WHERE i Złączenia

Zapytanie typu

SELECT * FROM $R\ p_1,\ S\ p_2$
WHERE $p_1.A = p_2.B$

---

zostanie przez DBMS zaimplementowane jako odpowiednie złączenie warunkowe: $R \bowtie_{A=B} S$, nie zaś bezpośrednio jako złożenie selekcji z iloczynem kartezjańskim: $\sigma_{A=B}(R \times S)$

Lepiej jednak skorzystać jawnie z operatorów złączenie w SQL takich jak **NATURAL JOIN**, **INNER JOIN** itp.

Agregacja w SQL

- Klauzula **GROUP BY**
- Klauzula **HAVING**
- Funkcje agregujące, m.in: **Max**, **Min**, **Sum**, **Avg**, **Count**

GROUP BY

GROUP BY E_1, E_2, ..., E_n

- Klauzula **GROUP BY** definiuje podział relacji na rozłączne podzbiory (grupy) krotek według wartości wyrażeń $E_1, \ldots, E_n$.
- Każdy podzbiór zdefiniowany przez **GROUP BY** składa się z krotek o identycznej wartości n-tki $(E_1, E_2, \ldots, E_n)$.
- Wyrażenia na liście **SELECT** muszą mieć sens dla całych grup, zatem mogą się odwoływać wyłącznie do wartości $E_1, \ldots, E_n$ oraz wartości obliczanych przez funkcje agregujące.
- Wartość zwracana przez funkcję agregującą zależy od wszystkich krotek w grupie.
- Jeśli nie użyto klauzuli **HAVING** to relacja wynikowa zawiera po jednej krotce dla każdego podzbioru.
- Pominięcie **GROUP BY** przy jednoczesnym użyciu funkcji agregującej oznacza że mamy jeden podzbiór składający się ze wszystkich krotek.

Przykład Grupowania

GROUP BY X, Y*Y

R

X	Y	Z	Grupy
'A'	1	10	('A', 1)
'A'	-1	20	
'A'	2	20	
'A'	2	30	('A', 4)
'A'	-2	25	
'B'	2	5	('B', 4)
'B'	2	10	
'C'	3	35	('C', 9)

Na przykład:

```
SELECT p.X, p.Y*p.Y AS T,
       SUM(p.Z) + p.Y*p.Y AS S
FROM R p
GROUP BY p.X, p.Y*p.Y
```

zwraca relację:

X	T	S
'A'	1	31
'A'	4	79
'B'	4	19
'C'	9	44

HAVING

HAVING *Warunek*

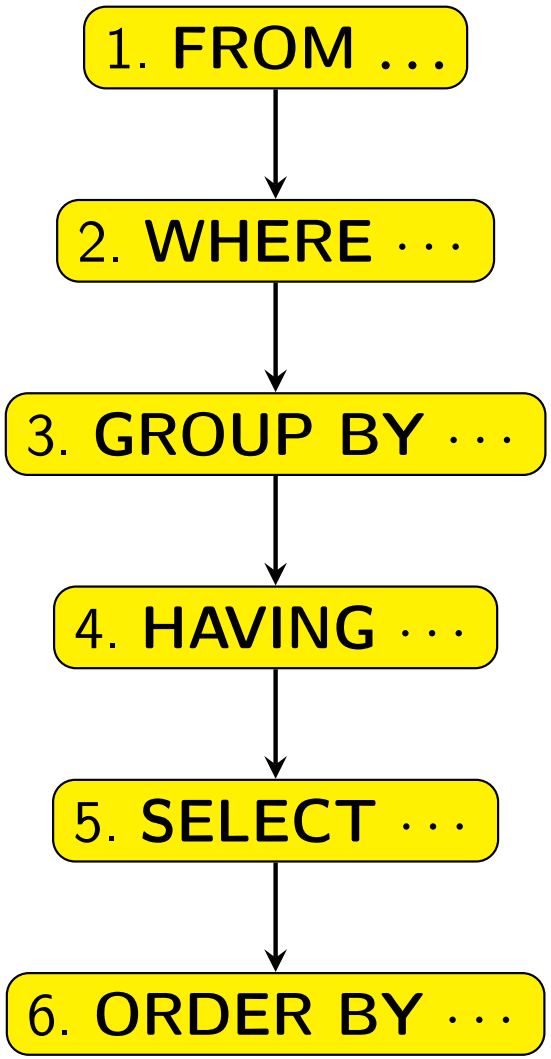
- Klauzula **HAVING** służy do eliminacji grup nie spełniających *Warunku*
- Oznacza to że *Warunek* musi być wyrażeniem logicznym mającym sens dla całej grupy (a nie dla poszczególnych krotek): musi odwoływać się jedynie do wartości $E_1, \ldots, E_n$ z listy **GROUP BY** oraz wartości obliczanych przez funkcje agregujące.
- W wielu DBMS-ach (np. Oracle) *Warunek* nie może się odwoływać do nazw kolumn zdefiniowanych na liście **SELECT**.
- Warunek **WHERE** obliczany jest przed grupowaniem.

Zapytanie SELECT z Lotu Ptaka Jeszcze Raz

SELECT ⟨*lista wyrażeń definiujących atrybuty wynikowej relacji*⟩
FROM ⟨*opis zmiennych relacyjnych z których korzysta zapytanie*⟩
WHERE ⟨*(Opcj.) warunek selekcji (przed agregacją)*⟩
GROUP BY ⟨*(Opcj.) lista wyrażeń definiujących grupy przy agregacji*⟩
HAVING ⟨*(Opcj.) warunek selekcji (po agregacji)*⟩
ORDER BY ⟨*(Opcj.) opis sortowania wyników*⟩

Kolejność różnych klauzul w zapytaniu **SELECT** jest myląca. Aby dobrze zrozumieć do czego można się odwoływać w różnych częściach zapytania należy wyobrazić sobie kolejność przetwarzania opisaną na następnym slajdzie.
Uwaga: Kolejności tej nie należy rozumieć jako definiującej faktyczną implementację wykonania zapytania

Kolejność Przetwarzania Klauzul Zapytania



1. Obliczana jest relacja wynikowa samego **FROM** (iloczyn kartezjański lub złączenie).
2. Z relacji zdefiniowanej powyżej eliminowane są krotki niespełniające warunku **WHERE**
3. Zbiór krotek które przetrwały **WHERE** jest dzielony na podzbiory według listy wyrażeń **GROUP BY**
4. Eliminowane są podzbiory niespełniające warunku **HAVING**
5. Obliczane są wyrażenia na liście **SELECT** dając w wyniku jedną krotkę dla każdego podzbioru który przetrwał **HAVING**.
6. Otrzymane krotki są sortowane.

Przykłady Zapytań z Agregacją

```
SELECT avg(p.Salary) AS Srednia,
       sum(p.Salary) AS Total
FROM Employees p;
```

```
SELECT p.DepartmentID,
       avg(p.Salary) AS Srednia
FROM Employees p
WHERE p.JobId<>'IT'
GROUP BY p.DepartmentId
HAVING min(p.Salary) > 5000
ORDER BY Srednia;
```

Błędne zapytanie:

```
SELECT p.FirstName, avg(p.Salary) AS Srednia
FROM Employees p GROUP BY p.DepartmentId
```

Standardowe Funkcje Agregujące w SQL

Każdy RDBMS implementuje następujące funkcje agregujące:

- $\mathbf{Min}(E)$ — dla grupy krotek liczy **minimalną** wartość wyrażeń E obliczonych dla każdej krotki grupy.
- $\mathbf{Max}(E)$— dla grupy krotek liczy **maksymaną** wartość wyrażeń E obliczonych dla każdej krotki grupy.
- $\mathbf{Sum}(E)$ — dla grupy krotek liczy sumę wyrażeń E obliczonych dla każdej krotki grupy.
- $\mathbf{Avg}(E)$ — dla grupy krotek liczy średnią wyrażeń E obliczonych dla każdej krotki grupy.
- $\mathbf{Count}()$ — dla grupy krotek liczy
  - $\mathbf{Count}(E)$ — ilość krotek w grupie dla których E nie jest **NULL**
  - $\mathbf{Count}(\mathbf{DISTINCT}\ E)$ — ilość różnych (i nie **NULL**) wartości E obliczonych dla każdej krotki grupy
  - $\mathbf{Count}(*)$ — ilość krotek w grupie

Operatory w SQL Pochodzące z Algebry Relacyjnej

- **CROSS JOIN** (iloczyn kartezjański)
- Operatory złączeń: **NATURAL JOIN** (i wersje zewnętrzne) oraz złączenia warunkowe **INNER JOIN** (i wersje zewnętrzne),
- Operatory teoriomnogościowe: **UNION**, **UNION ALL**, **INTERSECT**, **EXCEPT**.

CROSS JOIN

```
SELECT *
FROM Employees p CROSS JOIN Departments q;
```

to to samo co

```
SELECT *
FROM Employees p, Departments q;
```

Złączenia Naturalne

R **NATURAL JOIN** *S*

- Dokonuje złączenia naturalnego (jak w algebrze relacyjnej) na wspólnych (noszących te same nazwy) atrybutach z obu tabel.
- Relacja wynikowa będzie miała atrybuty obu relacji źródłowych (z usuniętymi duplikatami).
  - Nie jest legalne (w **SELECT**, **WHERE** itp.) odwoływanie się do wspólnych atrybutów obu relacji kwalifikowanych zmiennymi krotkowymi.
- Stosowanie tej operacji nie jest zalecane ponieważ atrybuty złączenia nie są podane jawnie i może dojść do złączenia na większej ilości atrybutów niż życzył sobie programista.
- Wersje zewnętrzne: **NATURAL LEFT OUTER JOIN**, **NATURAL RIGHT OUTER JOIN** i **NATURAL FULL OUTER JOIN**.
- Przykład:

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

Złączenia Warunkowe

R p **INNER JOIN** *S q* **ON** *Warunek*

- Dokonuje złączenia warunkowego relacji *R* i *S* opisywanego *Warunkiem*
- Wersje zewnętrzne: **LEFT OUTER JOIN**, **RIGHT OUTER JOIN** i **FULL OUTER JOIN**.

Przykład

```
SELECT e.FirstName, e.LastName, e.Salary
       m.FirstName AS ManagerFN,
       m.LastName AS MangerLN
FROM Employees e LEFT OUTER JOIN Employees m
     ON m.EmployeeId=e.ManagerId
WHERE e.Salary > 5000;
```

Przykład Złączenia Warunkowego z Podzapytaniem

```
SELECT e.FirstName, e.LastName,
       e.Salary, m.AvgSalary, m.DepartmentId
FROM Employees e INNER JOIN (
   SELECT d.DepartmentId,
          Avg(d.Salary) AS AvgSalary
   FROM Employees d
   GROUP BY d.DepartmentId
) m ON e.DepartmentId = m.DepartmentId
  AND e.Salary > m.AvgSalary
```

UNION, INTERSECT i EXCEPT

Operatory mnogościowe łączą ze sobą kompletne zapytania **SELECT**:

SELECT ... FROM ... *Operator* **SELECT ... FROM ...**

gdzie *Operator* to jeden z:

- **UNION** — zwraca krotki z obu zapytań bez duplikatów. Usuwanie duplikatów jest kosztowne.
- **UNION ALL** — zwraca krotki z obu zapytań (możliwe duplikaty)
- **INTERSECT** — zwraca tylko krotki zwrócone przez oba zapytania
- **EXCEPT** — zwraca tylko te krotki zwrócone przez pierwsze zapytanie których nie zwróciło drugie.

Listy **SELECT** w obu zapytaniach muszą mieć tą samą długość a wyrażenia na odpowiadających pozycjach muszą zwracać wartości zgodnych typów. Nazwy kolumn wyniku zapytania są brane z pierwszego z zapytań składowych.

Przykład UNION ALL

```
SELECT e.FirstName, e.LastName, d.DepartmentName
FROM Employees e INNER JOIN Departments d
     ON e.DepartmentId = d.DepartmentId
UNION ALL
SELECT e.FirstName, e.LastName, NULL
FROM Employees e
WHERE e.DepartmentId NOT IN (
   SELECT dd.DepartmentId
   FROM Departments dd
)
ORDER BY FirstName, LastName;
```

Sortowanie Krotek

Do wybrania kolejności zwracania przez DBMS krotek wynikowych do aplikacji która wysłała zapytania służy klauzula **ORDER BY**. Może ona pojawić się **tylko** w najbardziej zewnętrznym zapytaniu

ORDER BY E_1 M_1, E_2 M_2, ..., E_n M_n

O kolejności krotek decyduje najpierw E_1, dla krotek o równych wartościach E_1 decyduje E_2, itd. Kolejność dla E_i jest rosnąca bądź malejąca zależnie od M_i którym może być albo **ASC** (rosnąca) albo **DESC** (malejąca).

```
SELECT e.FirstName||' '||e.LastName AS Name
       e.Salary
FROM Employees e
ORDER BY Salary ASC, Name DESC;
```

Zapytania a Operacje Modyfikujące Dane
INSERT

Zapytania **SELECT** mogą być też częścią składową operacji modyfikujących dane.

Wstawianie do zmiennej relacyjnej wierszy zwróconych przez zapytanie (dialekt SQL RDBMS Oracle)

```
INSERT INTO Employees(FirstName, LastName, Salary)
SELECT e.Imie, e.Nazwisko,e.Pensja
FROM Pracownicy e
WHERE e.Salary > 5000;
```

Zapytania a Operacje Modyfikujące Dane
DELETE i UPDATE

DELETE z podzapytaniem: skasować pracowników których wyniki są poniżej średniej dla departamentu w którym pracują:

```
DELETE FROM Employees e
WHERE Performance < (
   SELECT Avg(d.Performance) FROM Employees d
   WHERE d.DepartmentId = e.DepartmentId
);
```

UPDATE z podzapytaniem: podwyższyć wszystkim pracownikom pensję o 0.1 minimalnej pensji w departamencie w którym pracują:

```
UPDATE Employees e
SET Salary = Salary + 0.1 * (
   SELECT Min(d.Salary) FROM Employees d
   WHERE d.DepartmentId = e.DepartmentId
);
```

Widoki (Perspektywy)

W bazie danych oprócz zmiennych relacyjnych zawierających relacje mogą znajdować się także specjalne zmienne zwane **widokami** (albo inaczej **perspektywami** przechowujące zapytania **SELECT**.

Deklaracja perspektywy w SQL:

CREATE VIEW *NazwaWidoku* **AS**
SELECT ...

- Widoku można użyć wszędzie tam gdzie można użyć zmiennej relacyjnej **o ile widok nie jest przez to polecenie modyfikowany**
- **Modyfikowalne widoki** wykraczają poza zakres tych zajęć
- Wykonywanie polecenia odwołującego się do widoku zaczyna się od podstawienia zawartości widoku pod jego nazwę jako podzapytania.

Przykład Deklaracji Widoku

```
CREATE VIEW AvgSalaries AS
SELECT Avg(d.Salary) AS AvgSalary, d.DepartmentId
FROM Employees d GROUP BY d.DepartmentId;
```

Kiedy wykonywane jest zapytanie

```
SELECT * FROM Employees e INNER JOIN AvgSalaries m
ON e.DepartmentId = m.DepartmentId
```

najpierw podstawiane jest zapytanie przechowywane w **AvgSalaries**:

```
SELECT * FROM Employees e INNER JOIN (
   SELECT Avg(d.Salary) AS AvgSalary, d.DepartmentId
   FROM Employees d GROUP BY d.DepartmentId
) m ON e.DepartmentId = m.DepartmentId
```

Dalej zapytanie jest wykonywane normalnie.

Zastosowania Perspektyw

Wśród zastosowań perspektyw można wymienić:

- Modularyzacja złożonych zapytań:
  - Jeśli jakieś podzapytanie pojawia się w wielu poleceniach można je zapisać w widoku i wykorzystać w tych zapytaniach.
- Ograniczanie dostępu do danych:
  - Większość RDBMS pozwala ograniczyć dostęp użytkowników tylko do niektórych (całych) zmiennych relacyjnych.
  - Ograniczenie dostępu do części relacji przechowywanej w zmiennej relacyjnej jest trudniejsze — można tu skorzystać z widoków.
  - Odbieramy użytkownikowi prawa do czytania zmiennej relacyjnej i tworzymy widok (który użytkownik będzie mógł czytać) zdefiniowany zapytaniem zwracającym tą część relacji do której czytania użytkownik ma prawo.
  - Z operacjami modyfikującymi dane jest trudniej, w razie potrzeby można zaimplementować operacje modyfikujące dane przy pomocy **triggerów** (wyzwalaczy)

Transakcje

Operacje DML (także zapytania) na bazie danych odbywają się w ramach **transakcji**

- W ramach jednej transakcji można umieścić wiele operacji DML.
- Menedżer transakcji danego RDBMS-u zapewnia że wszystkie operacje danej transakcji odbywają się jako jedna całość w izolacji od innych transakcji, tzn. że spełnione są warunki **ACID** (*Atomicity, Consistency, Isolation, Durability*).
- Każdą transakcję można zakończyć na dwa sposoby:
  - **Zatwierdzając** ją — od tego momentu, nawet jeśli nastąpi awaria systemu, zapewniona jest trwałość zmian.
  - **Wycofując** wszystkie zmiany w bazie danych wprowadzone przez tą transakcję.

Rozpoczynanie i Kończenie Transakcji

Rozpoczynanie Transakcji

Transakcje rozpoczyna się (zależnie od dialektu SQL) jawnym poleceniem, np. **BEGIN** lub rozpoczyna się automatycznie pierwszym poleceniem DML wysłanym po zakończeniu poprzedniej transakcji.

Kończenie Transakcji

- **COMMIT** — zatwierdza ostatecznie bieżącą transakcję.
- **ROLLBACK** — wycofuje wszystkie efekty bieżącej transakcji.

Autocommit

W wielu przypadkach RDBMS pozwala na ustawienie **automatycznego zatwierdzania**. Oznacza to że każde polecenie DML jest wykonywane w osobnej transakcji która jest automatycznie zatwierdzana (lub wycofywana w przypadku błędu) po jego zakończeniu.

Własności ACID

- **A**tomicity (atomowość) — wykonają się albo wszystkie operacje transakcji albo żadna. Efekty działania przerwanych transakcji będą wycofane.
- **C**onsistency (spójność) — po zakończeniu transakcji spełnione będą wszystkie więzy spójności.
- **I**solation (izolacja) — Każda transakcja wykonuje się tak jakby była jedyną wykonywaną w bazie danych a dokonywane przez nią modyfikacje danych nie są widoczne dla innych transakcji do czasu jej zatwierdzenia.
- **D**urability (trwałość) — po zakończeniu transakcji nawet awaria systemu nie spowoduje utraty modyfikacji danych.

Współbieżność i Spójność

W szczególności RDBMS musi zapewnić spójny widok danych dla każdej transakcji przy współbieżności dostępu

Negatywne Zjawiska Którym Trzeba Zapobiec

- **Dirty reads** — transakcja czyta dane które zostały zmodyfikowane przez inną, jeszcze nie zatwierdzoną transakcję.
- **Nonrepeatable reads** — przy ponownym odczycie tych samych danych transakcja widzi że w międzyczasie inna **zatwierdzona** transakcja zmodyfikowała lub skasowała część wierszy.
- **Phantom reads** — przy ponownym wykonaniu tego samego zapytania transakcja widzi nowe wiersze wstawione przez inną, zatwierdzoną transakcję.

Poziomy Izolacji ANSI/ISO

Poziom Izolacji	**Dirty read**	**Nonrepeatable read**	**Phantom read**
Read uncommited	$\surd$	$\surd$	$\surd$
Read commited	—	$\surd$	$\surd$
Repeatable read	—	—	$\surd$
Serializable	—	—	—

- RDBMS-y udostępniają kilka poziomów izolacji (czasem różniących się od poziomów ANSI/ISO).
- Poziom izolacji można wybrać przy rozpoczęciu transakcji.
- Zwykle domyślnym poziomem jest **READ COMMITED**