Създаване и изтриване на изглед

Изгледи

- Релациите създадени с командата CREATE TABLE съществуват физически на диска.
- Други обекти в базата от данни, които не съществуват физически на диска, но имат поведение на таблици са изгледите (view).
- Често изгледите са наричани и виртуални таблици.
- Това е така защото те не съдържат данни, а само дефиниция от коя таблица да бъдат взети данните.

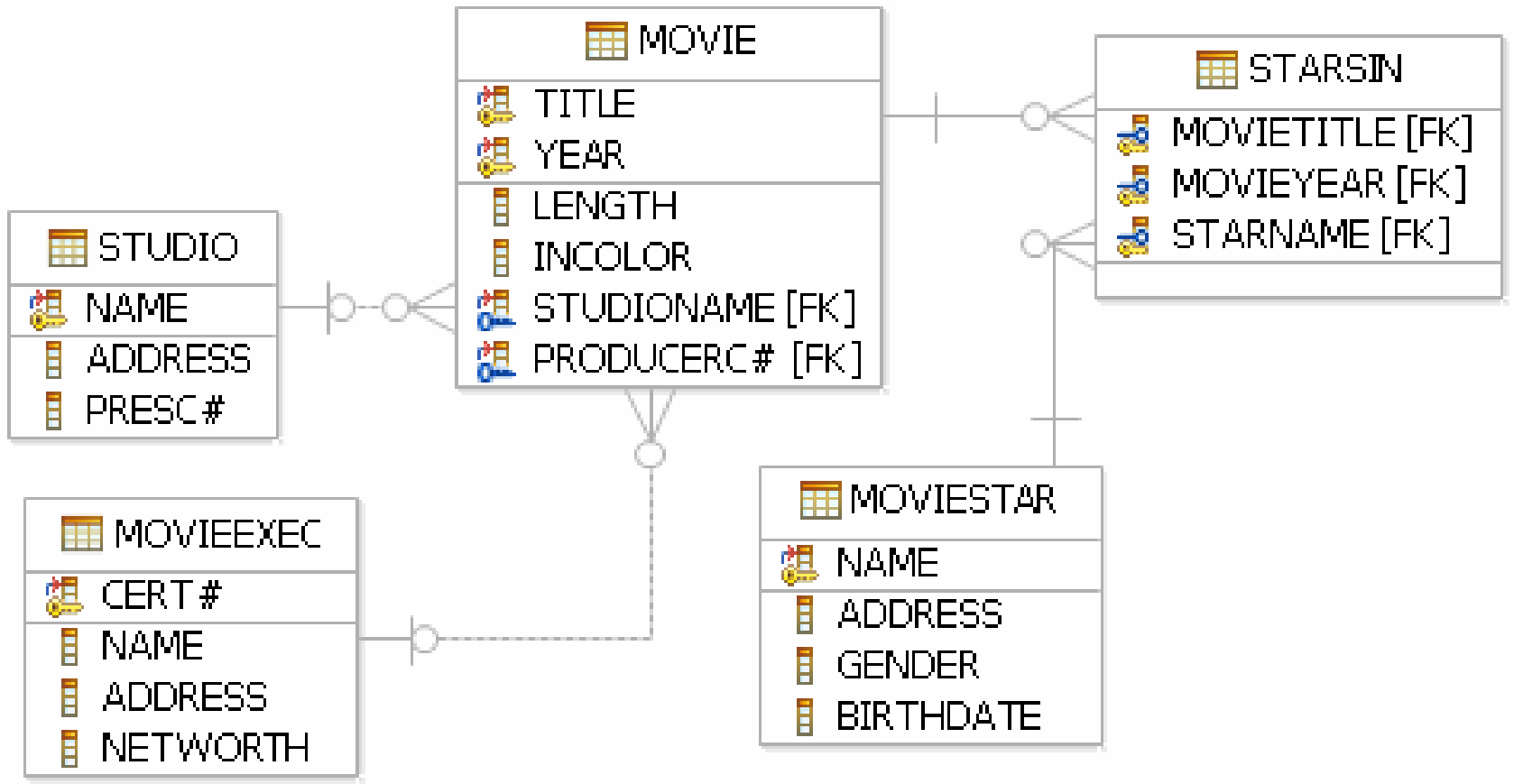
Изгледи

- В SQL изглед се дефинира с:
  - ключовите думи **CREATE VIEW**, последвани от
  - името на изгледа, последвано от
  - ключовата дума *AS*, последвана от
  - SELECT заявка;

```
CREATE VIEW <име_на_изглед>
AS <SELECT заявка>;
```

- Заявката реално е самата дефиниция на изгледа. Тя определя от кои таблици ще бъдат показани данните.

База от данни - Movies



Създаване на изглед

- Например изглед, които ни дава информация за всички филми, актьорите които участват в тях и имената на студията, които ги продуцират

```
CREATE VIEW V_MOVIE_INFO
AS
SELECT TITLE, YEAR, STUDIONAME, STARNAME
FROM MOVIE, STARSIN
WHERE TITLE = MOVIETITLE
AND YEAR = MOVIEYEAR
```

- Към изглед се обръщаме по същият начин, както към таблица.
- Например:

```
SELECT * FROM V_MOVIE_INFO;
```

Обръщане към изглед в заявка

- Всеки път когато правим заявка към изгледа, всъщност правим заявка към таблиците, които участват в дефиницията на изгледа.
- В действителност изгледа не съдържа кортежи.
- За това когато правим заявка към изглед, резултатното множество се връща от таблицата (таблиците) участващи в дефиницията на изгледа.
- Така в резултат на една и съща заявка към изгледа могат да бъдат върнати различни резултатни множества, особено ако таблицата е била модифицирана между двете заявки.
- Ако се изтрие някоя от таблиците, които участват в дефиницията на изгледа, то изгледът става неактивен.
- Това означава, че изгледът трябва да се изтрие и създаде наново.

Именуване на колоните в изгледа

Понякога е удобно още със създаването на изгледа да дадем нови имена на атрибутите, които се връщат в резултат на SELECT заявката. Това става със следният синтаксис:

```
CREATE VIEW my_view(A1, A2 …, An)
AS <SELECT заявка>;
```

- За горният пример, синтаксисът ще изглежда така:

```
CREATE VIEW V_MOVIE_INFO(TITLE, YEAR, NAME, ACTOR)
AS
SELECT TITLE, YEAR, STUDIONAME, STARNAME
FROM MOVIE, STARSIN
WHERE TITLE = MOVIETITLE
AND YEAR = MOVIEYEAR
```

Изгледи

- Всяка колона в изгледа трябва да има име, ако в SELECT клаузата има изрази, задължително трябва или с AS да се даде име на колоната, или при CREATE VIEW да се изброят имената на всички колони.
- Същото важи и ако в SELECT клаузата има колони с едни и същи имена. Причината е очевидна - изгледите трябва да могат да се използват в различни заявки, както и таблиците и ако някоя колона на изгледа няма име, няма да можем да я достъпваме.
- Изгледите не могат да бъдат променяни. Ако по някаква причина искаме да променим дефиницията на изглед, тогава трябва да изтрием съществуващият изглед и да го създадем наново със същото име и нова дефиниция.
- Изглед се изтрива с ключовите думи **DROP VIEW**. Например:

```
DROP VIEW V_MOVIE_INFO;
```

Предимства на изгледите

- Едно от предимствата на изгледите е, че могат да бъдат използвани за реализиране на сигурност в базите от данни.
- Изгледите могат да ограничат достъпа на даден потребител, като сведат неговата “видимост” до определени атрибути или кортежи на дадена релация.
- Например изгледът, който създадохме по-горе. Ако имаме потребител, който да има права да прави SELECT върху изгледа и няма права върху таблицата MOVIE, то създаденият от нас изглед ограничава достъпа на потребителя само до име на филм и година на филм. Другата информация за филма е скрита от този потребител.
- Предимство на изгледите е, че могат да бъдат използвани и за ускоряване на заявките. Това е в сила, когато използваме една и съща заявка често пъти. СУБД имат механизъм за кеширане на такива заявки, по този начин изгледите могат да доведат до ускоряване на изпълнението на съответната заявка.

Updatable и read-only изгледи

- Според дефиницията на изгледа, изгледите могат да бъдат updatable (т.е. такива през, които може да бъде променена оригиналната таблица) и read-only (т.е. такива през, които може само да се извежда информация от оригиналната таблица, но таблицата не може да бъде променяна).
- Има определени изисквания, на които трябва да отговаря дефиницията на един изглед за да бъде той updatable. Ако ги обобщим, те включват следните ограничения:
  - Заявката на updatable изглед не може да включва съединения, подзаявки, групиране и агрегатни функции
  - В общи линии заявката на updatable изглед трябва да бъде само върху една таблица, като трябва да включва всички колони от таблицата. Позволено е да не се включват само тези колони които имат стойности по подразбиране или позволяват NULL.
- Всички останли изгледи, които не отговарят на горните условия са read-only.

Пример за updatable и read-only изгледи

- Пример за read-only изглед е изгледът V_MOVIE_INFO Той включва съединение на две таблици, като ползва само част от колоните на тези таблици.
- По тези причини, през този изглед не могат да бъдат обновени нито една от двете таблици от дефиницията.
- Пример за updatable изглед е следният:

```
CREATE VIEW V_MOVIE_1980
AS
SELECT *
FROM MOVIE
WHERE YEAR > 1980
```

Пример за updatable и read-only изгледи

- През този изглед могат да бъдат правени следните заявки:

```
INSERT INTO V_MOVIE_1980
  VALUES ('Lego', 2016, 100, 'Y', 'MGM', 199);

DELETE FROM V_MOVIE_1980
 WHERE TITLE = 'Lego';

UPDATE V_MOVIE_1980
  SET LENGTH = NULL
  WHERE TITLE = 'Lego';
```

Пример за updatable и read-only изгледи

- През горният изглед могат да бъдат направени и следните промени:

```
UPDATE V_MOVIE_1980
   SET LENGTH = NULL
   WHERE YEAR = 1977
```

- Въпреки че, дефиницията на изгледа ни ограничава да виждаме само записите след 1980 година, с горната заявка обновихме запис от таблицата, който не се вижда от изгледа.

Пример за updatable и read-only изгледи

- Аналогична е ситуацията и при следния INSERT:
- Тоест отново вмъкваме запис през изгледа, а самият изглед не може да види този запис.

```
CREATE VIEW V_MOVIE_1980
AS
SELECT *
FROM MOVIE
WHERE YEAR > 1980


INSERT INTO V_MOVIE_1980
VALUES('NEW MOVIE', 1967, 120, 'Y', 'MGM', 123)
```

Изгледи и опция CHECK

- При такива случаи, ако искаме да избегнам подобни нежелани ефекти се използва опцията WITH CHECK OPTION, която се записва при дефинирането на изгледа.
- Смисълът на тази опция е да не позволява през изгледа да бъдат променяни редове от оригиналната таблица, които след промяната няма да отговарят на условието в WHERE клаузата на изгледа и съответно няма да могат да бъдат видени от него. Например:

```
CREATE VIEW V_MOVIE_1980
AS
SELECT *
FROM MOVIE
WHERE YEAR > 1980
WITH CHECK OPTION;
```

- Сега ако повторим горните заявки, СУБД ще изведе съобщение за грешка, че INSERT заявката нарушава условието на CHECK опцията от дефиницията на изгледа.

Дефиниране на изглед от изглед

- Изгледите са виртуални таблици, това прави възможно дефинирането на изглед в чиято дефиниция участва изглед. Например изгледът:

```
CREATE VIEW V_MOVIE_MGM
AS
SELECT *
FROM V_MOVIE_1980
WHERE STUDIONAME = 'MGM';
```

- Със такива изгледи се работи по същият начин, както и с представените до момента изгледи.
- За тях важат същите правила за updatable и read-only изгледи, както и за WITH CHECK OPTION.

Изгледи и опция CASCADED CHECK

- Някои СУБД подържат и опцията WITH CASCADED CHECK OPTION, чиито смисъл е да не позволява през изгледите - текущият и този върху които е дефиниран съответния изглед, да бъдат променяни редове от оригиналната таблица, които след промяната няма да отговарят на условията в WHERE клаузата на изгледите.
- Понеже проверката се извършва каскадно за всички изгледи от тук идва и думата CASCADED в опцията.
- Например:

```
CREATE VIEW V_MOVIE_MGM
AS
SELECT *
FROM V_MOVIE_1980
WHERE STUDIONAME = 'MGM'
WITH CASCADED CHECK OPTION;
```

Изгледи и опция CASCADED CHECK

- Този изглед ще позволява само заявки, при които годината на филма е < 1980 и името на студиото е MGM

```
INSERT INTO V_MOVIE_MGM
VALUES('NEW MOVIE', 1967, 120, 'Y', 'MGM', 123);
```

- Отново ще доведат до извеждане на съобщение за грешка от СУБД, че INSERT заявката нарушава условието на CHECK опцията от дефиницията на изгледа.

Тригери

Тригери

- Тригерите са обекти в базите от данни, които се различават от другите ограничения по това, че се изпълняват при настъпването на дадено събитие, в следствие на което ако поставеното условие в тригера е удовлетворено се изпълнява определено действие.
- По тази причина, тригерите често са наричани и правила от тип “събитие-условие-действие”.
- Събитията, които могат да задействат тригер са INSERT, UPDATE и DELETE.
- Тригерите се обекти в базата от данни, които се дефинират за конкретна таблица или изглед и веднъж създадени се задействат автоматично.

Тригери

- Действията INSERT, UPDATE и DELETE за съответната таблица, водят до задействане на тригера и изпълняване на съответното действие дефинирано в тригера.
- Тригерите се задействат автоматично при споменатите команди.
- Те не могат да бъдат задействани по друг начин, освен посредством модификация на таблицата или изгледа за които са дефинирани.

Синтаксис

- Обобщеният синтаксис за създаване на тригер е следният:

```
CREATE TRIGGER <име_на_тригер>
{AFTER |INSTEAD OF |BEFORE}  {INSERT |
UPDATE | DELETE} ON my_table
REFERENCING NEW ROW | TABLE AS N OLD ROW
| TABLE AS O
FOR EACH ROW | STATEMENT
WHEN(<условие>)
                    <действие>;
```

Обяснение

- При създаването на един тригер, може да бъде указано, кога да се изпълни действието на тригера: преди настъпване на събитието (BEFORE), след настъпване на събитието (AFTER) или вместо самото действие (INSTEAD OF).
- Също така в дефиницията на тригера се указва и кое е събитието, което задейства тригера (INSERT, UPDATE или DELETE).
- Възможно е тригерът да бъде задействан и при трите действия, за които да се изпълнява едно и също действие. В този случай при дефиниране на тригера се указват и трите команди.

Обяснение

- Ако събитието което задейства тригерът е UPDATE, освен името на таблицата (или изгледа), може да бъде указана и конкретна колона от таблицата, чиято промяна да води до задействане на тригера.
- В дефиницията на тригера в клаузата REFERENCING се пази и информация за стойностите на кортежите преди и след настъпване на събитието (при действие UPDATE), за новите стойности които ще бъдат вмъкнати (при действие INSERT) и за стойностите, които са били изтрити (при действие DELETE).
- Не всички СУБД поддържат REFERENCING клауза. При някои от тях, достъпът до стойностите на кортежите преди и след настъпване на събитието се осъществява посредством специални таблици.
- Например за MySQL и Oracle таблицата NEW съдържа новите стойности, а таблицата OLD съдържа старите стойности. За MSSQL това са специалните таблици INSERTED и DELETED, съответно за новите и старите стойности.

Обяснение

- Дали действието от тригера да бъде изпълнено за всеки кортеж, засегнат от събитието или еднократно за съответното събитие се указва в клаузите (FOR EACH ROW или FOR EACH STATEMENT).
- Ако тригерът е FOR EACH ROW тогава се използва (REFERENCING NEW ROW AS N OLD ROW AS O). Ако тригерът е FOR EACH STATEMENT тогава се използва (REFERENCING NEW TABLE AS N OLD TABLE AS O).
- Отново не всички СУБД поддържат клаузата FOR EACH STATEMENT. При MSSQL дори тази клауза не се използва.
- И на последно място в дефиницията на тригера се задава условие, което да бъде проверено преди да се изпълни действието на тригера. То се указва в скоби след WHEN клаузата.
- В тези случаи тялото на тригера ще се изпълни само ако е изпълнено условието в WHEN клаузата. WHEN клаузата не е задължителна, може да имаме действие на тригера и без да има WHEN клауза.
- Не всички СУБД поддържат WHEN клауза.

BEFORE Синтаксис за DB2

```
CREATE TRIGGER TRIG_MOVIE_LENGTH
BEFORE UPDATE OF LENGTH ON MOVIE
REFERENCING NEW AS N OLD AS O
FOR EACH ROW
WHEN(N.LENGTH < 0) SET N.LENGTH = O.LENGTH
```

- С горният синтаксис се създава тригер с име TRIG_MOVIE_LENGTH. Той се задейства при UPDATE на колоната LENGTHна таблицата Movie.
- Ако новата стойност е отрицателно число, действието на тригера я променя на старата дължина на филм.
- Тригерът се задейства за всеки ред засегнат от UPDATE.

Тестване на тригера

- За да тестваме как работи тригера, трябва да направим UPDATE за Movie на колоната LENGTH. Например:

```
SELECT * FROM MOVIE
WHERE STUDIONAME = 'MGM'

UPDATE MOVIE
SET LENGTH = -1
WHERE STUDIONAME = 'MGM'
```

- Горният UPDATE би трябвало да промени дължините на всички филми на студио – MGM. Но заради дефинирания тригер дължините ще останат не променени.

AFTER синтаксис за DB2

```
CREATE TRIGGER TRIG_MGM_AVG
AFTER INSERT ON MOVIE
REFERENCING NEW TABLE AS N
FOR EACH STATEMENT
WHEN((SELECT AVG(LENGTH)
      FROM MOVIE
      WHERE STUDIONAME='MGM') > 500)
DELETE FROM MOVIE
WHERE TITLE IN (SELECT TITLE FROM N)
AND YEAR IN (SELECT YEAR FROM N)@
```

- С горният синтаксис се създава тригер с име TRIG_MGM_AVG. Той се задейства при INSERT в таблицата MOVIE.
- Ако средната стойност на дължината за 'MGM' е по-голяма от 500, действието на тригерът изтрива редът, който последно сме вмъкнали и който е причината за това дължината да е по-висока от 500. Действието се извършва еднократно за конкретен INSERT.

Тестване на тригера

- За да тестваме как работи тригера, трябва да направим INSERT в таблицата Movie. Например:

```
SELECT * FROM MOVIE;

INSERT INTO MOVIE
VALUES('TITLE1',1999, 5000, 'Y', 'MGM', 199);

SELECT * FROM MOVIE;
```

Пример за INSTEAD OF тригер

- Заместващите тригери са тригери, чиито действия се изпълняват вместо самото събитие.
- Този вид тригери се дефинират за изгледи, тъй като вмъкването на ред, обновяването на ред и изтриването на ред за изглед не е смислена операция.
- Нека да създадем изглед `V_STUDIO`, който да ни дава следната информация за студията:

```
CREATE VIEW V_STUDIO
AS
SELECT NAME, ADDRESS, PRESC#
 FROM STUDIO;
```

Пример за INSTEAD OF тригер

- Искаме да създадем тригер, който да се задейства при INSERT в изгледа V_STUDIO;
- Тъй като вмъкването в изгледа реално означава вмъкване в таблицата STUDIO, действието на тригерът ще реализира точно това.
- Синтаксис за DB2:

```
CREATE TRIGGER TRIG_V_STUDIO
INSTEAD OF INSERT ON V_STUDIO
REFERENCING NEW AS N
FOR EACH ROW
    INSERT INTO STUDIO VALUES(N.NAME,
N.ADDRESS, N.PRESC#);
```

Тестване на тригера

- За да тестваме как работи тригера, трябва да направим INSERT в изгледът V_AGENCIES. Например:

```
SELECT *
 FROM STUDIO;


INSERT INTO V_STUDIO
 VALUES ('New Studio', 'Bulgaria', 199);

SELECT *
 FROM STUDIO;
```

Функции

Функции - скаларни

```
CREATE FUNCTION deptname(p_empid VARCHAR(6))
RETURNS VARCHAR(30)
SPECIFIC deptname
BEGIN ATOMIC
    DECLARE v_department_name VARCHAR(30);
    DECLARE v_err VARCHAR(70);
    SET v_department_name = (
        SELECT d.deptname FROM department d, employee e
            WHERE e.workdept=d.deptno AND e.empno= p_empid);
    SET v_err = 'Error: employee ' || p_empid || ' was not found';
    IF v_department_name IS NULL THEN
        SIGNAL SQLSTATE '80000' SET MESSAGE_TEXT=v_err;
    END IF;
  RETURN v_department_name;
END
```

Функции - извикване

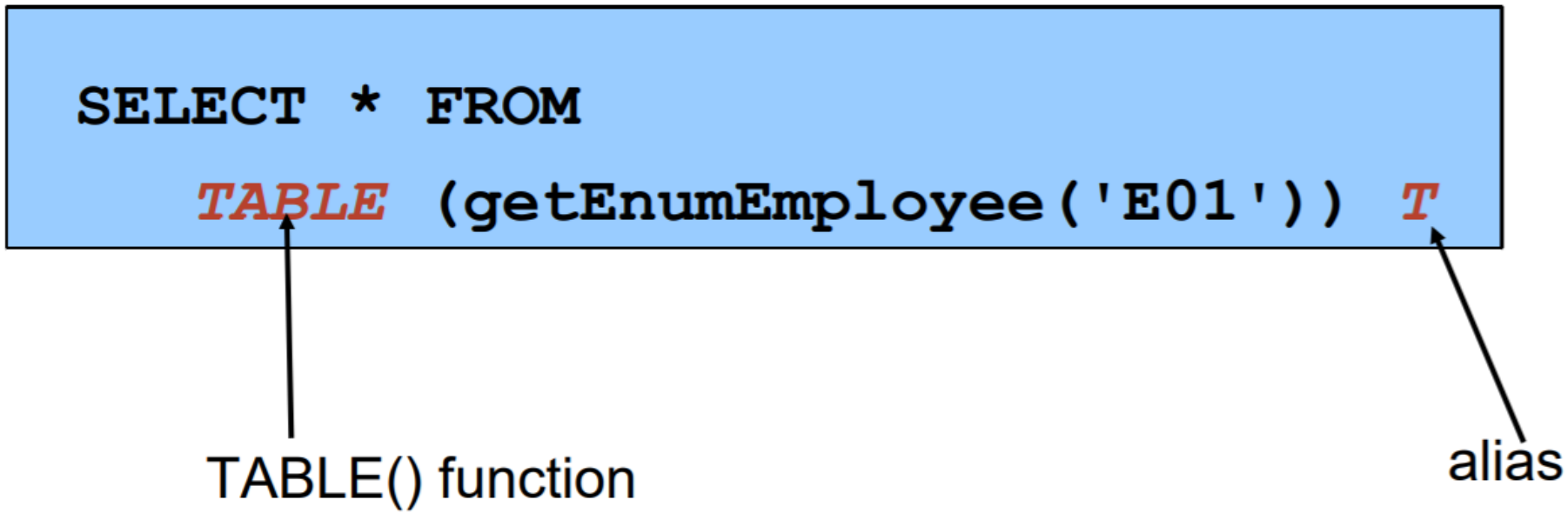
```
SELECT DEPTNAME('000010') FROM SYSIBM.SYSDUMMY1

VALUES DEPTNAME('000010')
```

Функции - таблични

```
CREATE FUNCTION getEnumEmployee(p_dept VARCHAR(3))
RETURNS TABLE
     (empno CHAR(6),
      lastname VARCHAR(15),
      firstnme VARCHAR(12))
SPECIFIC getEnumEmployee
RETURN
  SELECT e.empno, e.lastname, e.firstnme
    FROM employee e
    WHERE e.workdept=p_dept
```

Функции - извикване

```
SELECT * FROM
    TABLE (getEnumEmployee('E01')) T
```



База от данни SAMPLE

DB2ADMIN, DB2INST1