<epam> Delivering Excellence in Software Engineering

Базы данных и язык SQL

Базы данных

База данных (БД) - некий организованный набор информации.

Система управления базами данных (СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Пример БД

№	Наименование	Ед. изм.	Цена	Кол-во
1	Кирпич	штука	255	10000
2	Краска	литр	580	670
3	Шифер	лист	130	500
…	…	…	…	…
10001	Гвоздь	штука	20	8000
10002	Кабель	метр	100	200

Базовые свойства СУБД

- скорость;
- разграничение доступа;
- гибкость;
- целостность;
- отказоустойчивость.

Базовые функции СУБД

- интерпретация запросов пользователя, сформированных на специальном языке;
- определение данных (создание и поддержка специальных объектов, хранящих поступающие от пользователя данные, ведение внутреннего реестра объектов и их характеристик – так называемого словаря данных);
- исполнение запросов по выбору, изменению или удалению существующих данных или добавлению

Базовые функции СУБД

- безопасность (контроль запросов пользователя на предмет попытки нарушения правил безопасности и целостности, задаваемых при определении данных);
- производительность (поддержка специальных структур для обеспечения максимально быстрого поиска нужных данных);
- архивирование и восстановление данных.

Модель данных в реляционных СУБД

По типу модели данных СУБД делятся на *сетевые, иерархические, реляционные, объектно-ориентированные, объектно-реляционные.*

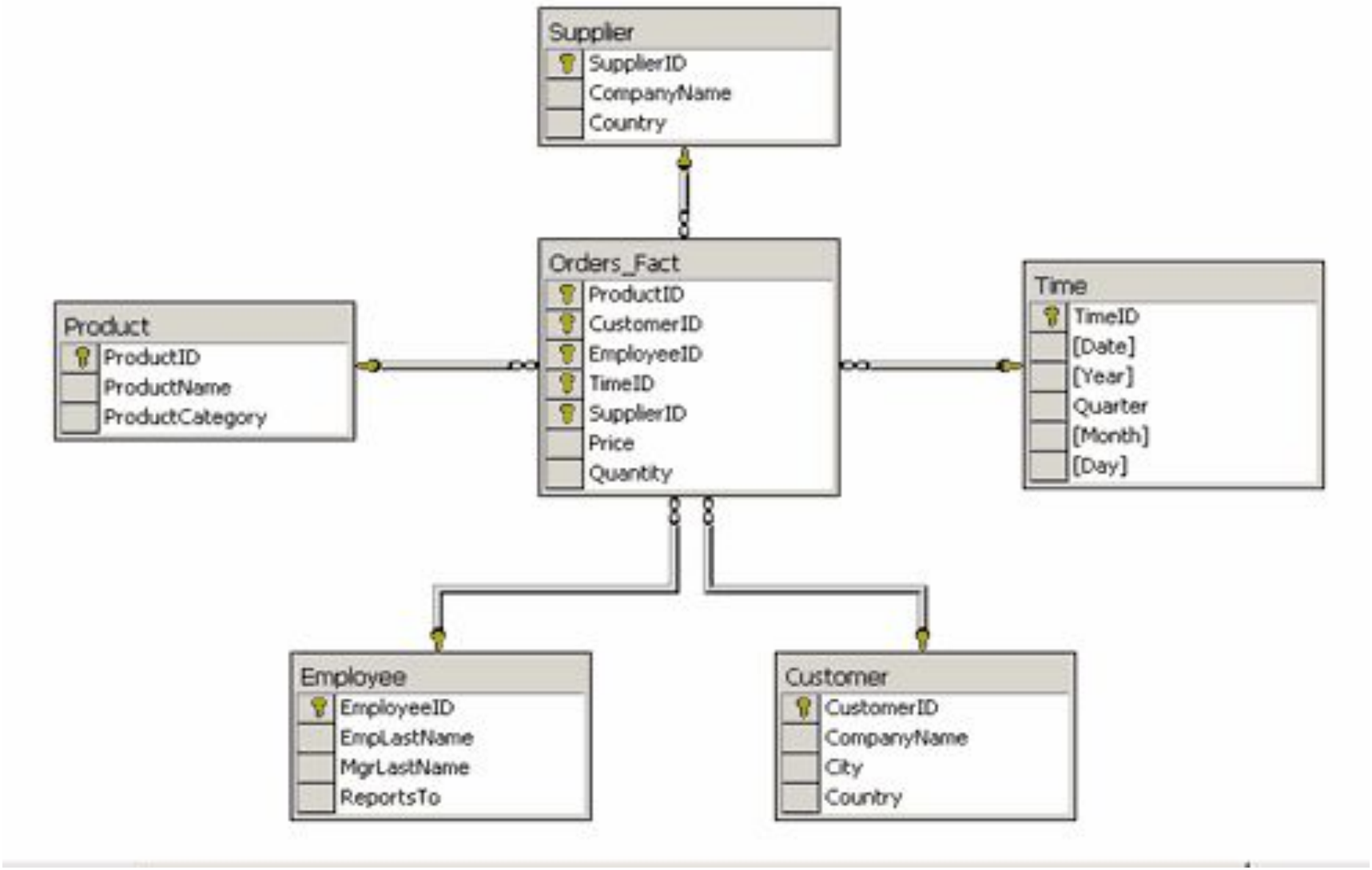
Реляционная СУБД представляет собой совокупность именованных двумерных таблиц данных, логически связанных (находящихся в отношении) между собой.

Реляционная БД

Таблицы состоят из строк и именованных столбцов, **строки** представляют собой экземпляры информационного объекта, **столбцы** – атрибуты объекта. Строки иногда называют **записями**, а столбцы – **полями записи**.

Таким образом, в реляционной модели все данные представлены для пользователя в виде таблиц значений данных, и все операции над базой сводятся к манипулированию таблицами.

Реляционная БД



Связь в реляционной БД

Связи между отдельными таблицами в реляционной модели в явном виде могут не описываться.

Они устанавливаются пользователем при написании запроса на выборку данных и представляют собой условия равенства значений соответствующих полей.

Связь в реляционной БД

Первичный ключ (главный ключ, primary key, PK). Представляет собой столбец или совокупность столбцов, значения которых однозначно идентифицируют строки.

Вторичный ключ (внешний, foreign key, FK) - Столбец или совокупность столбцов, которые в данной таблице не являются первичными ключами, но являются первичными ключами в другой таблице.

Связь в реляционной БД

Сотрудники

Табельный №	Фамилия	Должность	№ отдела
1	Иванов	Начальник	15
2	Петров	Инженер	15
3	Сидоров	Менеджер	10

Отделы

№ отдела	Наименование
15	Производственный отдел
10	Отдел продаж

Ограничения целостности

Целостность базы данных (database integrity) — соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам.

Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, называется **ограничением целостности** (integrity constraint).

Ограничения целостности могут относиться к разным информационным объектам: атрибутам, кортежам, отношениям, связям между ними и т.д.

Ограничения целостности

Для полей (атрибутов) используются следующие виды ограничений:

- Тип и формат поля .
- Задание диапазона значений.
- Недопустимость пустого поля.
- Задание домена.
- Проверка на уникальность значения какого-либо поля. Ограничение позволяет избежать записей-дубликатов.

Ограничения целостности

Ограничения таблицы :

PRIMARY KEY (Имя столбца.,..)
UNIQUE (Имя столбца.,..)
FOREIGN KEY (Имя столбца.,..) **REFERENCES** Имя таблицы [(Имя столбца.,..)] [Ссылочная спецификация]
CHECK Предикат
DEFAULT = <Значение по умолчанию>
NOT NULL

Ссылочная спецификация:

[ON UPDATE {CASCADE | SET NULL | SET DEFAULT | RESTRICTED| NO ACTION}]
[ON DELETE {CASCADE | SET NULL | SET DEFAULT | RESTRICTED| NO ACTION}]

Нормализация

Основная цель **нормализации** – устранение избыточности данных.

- Первая нормальная форма (**1НФ, 1NF**)
- Вторая нормальная форма (**2НФ, 2NF**)
- Третья нормальная форма (**3НФ, 3NF**)
- Нормальная форма Бойса — Кодда (**НФБК, BCNF**)
- Четвёртая нормальная форма (**4НФ, 4NF**)
- Пятая нормальная форма (**5НФ, 5NF**)
- Доменно-ключевая нормальная форма (**ДКНФ, DKNF**).

Нормализация модели данных

Первая нормальная форма:

информация в каждом поле таблицы является неделимой и не может быть разбита на подгруппы.

…	Иванов, 15 отдел, начальник	…

Фамилия	Должность	№ отдела
Иванов	Начальник	15

Нормализация модели данных

Вторая нормальная форма:

таблица соответствует 1НФ и в таблице нет неключевых атрибутов, зависящих от части сложного (состоящего из нескольких столбцов) первичного ключа.

№ отдела	Должность	Отдел	Количество сотрудников
15	Начальник	Производственный отдел	1
15	Инженер	Производственный отдел	5
10	Начальник	Отдел продаж	1
10	Менеджер	Отдел продаж	10

Отделы

№ отдела	Наименование
15	Производственный отдел
10	Отдел продаж

Структура

№ отдела	Должность	Количество сотрудников
15	Начальник	1
15	Инженер	5
10	Начальник	1
10	Менеджер	10

Нормализация модели данных

Третья нормальная форма:

таблица соответствует первым двум НФ и все неключевые атрибуты зависят только от первичного ключа и не зависят друг от друга.

Сотрудники

Табельный №	Фамилия	Оклад	Наименование отдела	№ отдела
1	Иванов	500	Производственный отдел	15
2	Петров	400	Производственный отдел	15
3	Иванов	600	Отдел продаж	10

Сотрудники

Табельный №	Фамилия	Оклад	№ отдела
1	Иванов	500	15
2	Петров	400	15
3	Иванов	600	10

Отделы

№ отдела	Наименование
15	Производственный отдел
10	Отдел продаж

Язык SQL

SQL (*Structured Query Language*) – непроцедурный язык взаимодействия приложений и пользователей с реляционными СУБД, состоящий из набора стандартных команд на английском языке.

Отдельные команды изначально никак логически не связаны друг с другом.

Язык SQL

SQL может использоваться как интерактивный (для выполнения запросов) и как встроенный (для построения прикладных программ).

Базовый вариант SQL содержит порядка **40** команд (часто еще называемых **запросами** или **операторами**) для выполнения различных действий внутри СУБД.

Операторы SQL

Выделяют следующие группы операторов SQL:

- операторы определения объектов базы данных (Data Definition Language - **DDL**;
- операторы манипулирования данными (Data Manipulation Language - **DML**);
- команды управления транзакциями (Transaction Control Language – **TCL**);
- операторы защиты и управления данными (Data Control Language – **DCL**).

Операторы SQL

Операторы DDL - определения объектов базы данных :

CREATE DATABASE - создать базу данных
DROP DATABASE - удалить базы данных
CREATE TABLE - создать таблицу
ALTER TABLE - изменить таблицу
DROP TABLE - удалить таблицу
CREATE DOMAIN - создать домен
ALTER DOMAIN - изменить домен
DROP DOMAIN - удалить домен
CREATE VIEW - создать представление
DROP VIEW - удалить представление

Операторы DML - манипулирования данными

SELECT - отобрать строки из таблиц
INSERT - добавить строки в таблицу
UPDATE - изменить строки в таблице
DELETE - удалить строки в таблице

Операторы SQL

Команды управления транзакциями TCL

Используются для управления изменениями данных, производимыми DML-командами. С их помощью несколько DML-команд могут быть объединены в единое логическое целое, называемое транзакцией.

COMMIT - завершить транзакцию и зафиксировать все изменения в БД

ROLLBACK - отменить транзакцию и отменить все изменения в БД

SET TRANSACTION - установить некоторые условия выполнения транзакции

Операторы защиты и управления данными – DCL

GRANT - предоставить привилегии пользователю или приложению на манипулирование объектами
REVOKE - отменить привилегии пользователя или приложения

Язык SQL

звездочка (*) - для обозначения "все";
квадратные скобки **([])** – конструкции, заключенные в эти скобки, являются необязательными (т.е. могут быть опущены);
фигурные скобки **({})** –конструкции, заключенные в эти скобки, должны рассматриваться как целые синтаксические единицы;
многоточие **(...)** – указывает на то, что непосредственно предшествующая ему синтаксическая единица факультативно может повторяться один или более раз;
прямая черта **(|)** – означает наличие выбора из двух или более возможностей.

Язык SQL

точка с запятой **(;)** – завершающий элемент предложений SQL;

запятая **(,)** – используется для разделения элементов списков;

пробелы **( )** – могут вводиться для повышения наглядности между любыми синтаксическими конструкциями предложений SQL;

прописные жирные латинские буквы и символы – используются для написания конструкций языка SQL;

строчные буквы – используются для написания конструкций, которые должны заменяться конкретными значениями, выбранными пользователем;

SELECT

Для выборки данных используется команда **SELECT**.

SELECT [DISTINCT] <список столбцов>
FROM <имя таблицы> [**JOIN** <имя таблицы> **ON** <условия связывания>]
[**WHERE** <условия выборки>]
[**GROUP BY** <список столбцов для группировки> [**HAVING** <условия выборки групп>]]
[**ORDER BY** <список столбцов для сортировки>]

Секция DISTINCT

Если в результирующем наборе данных встречаются одинаковые строки (значения всех полей совпадают), можно от них избавиться, указав ключевое слово **DISTINCT** перед списком столбцов.

SELECT DISTINCT Position **FROM** Employees

Секция FROM

Перечень таблиц, из которых производится выборка данных, указывается в секции **FROM**. Выборка возможна как из одной таблицы, так и из нескольких логически взаимосвязанных.

Логическая взаимосвязь осуществляется с помощью подсекции **JOIN**.

На каждую логическую связь пишется отдельная подсекция.

Внутри подсекции указывается условие связи двух таблиц (обычно по условию равенства первичных и вторичных ключей).

Пример выборки

Employees

TabNum	Name	Position	DeptNum	Salary
1	Иванов	Начальник	15	1000
2	Петров	Инж.	15	500
3	Сидоров	Менеджер	10	700

Departments

DeptNum	City	Name
15	1	Производственный отдел
10	2	Отдел продаж

Cities

City	Name
1	Минск
2	Москва

Пример выборки

SELECT Employees.TabNum, Employees.Name, Departments.Name

FROM Employees

JOIN Departments **ON** Employees.DeptNum = Departments.DeptNum

1	Иванов	Производственный отдел
2	Петров	Производственный отдел
3	Сидоров	Отдел продаж

Пример выборки

```
SELECT Employees.TabNum, Employees.Name,
Departments.Name,
Cities.Name
FROM Employees
JOIN Departments ON Employees.DeptNum =
Departments.
DeptNum
JOIN Cities ON Departments.City = Cities.City
```

1	Иванов	Производственный отдел	Минск
2	Петров	Производственный отдел	Минск
3	Сидоров	Отдел продаж	Москва

Секция JOIN

SELECT Table1.Field1, Table2.Field2

FROM Table1

JOIN Table2

ON Table2.ID1 =Table1.ID1

AND Table2.ID2 =Table1.ID2

AND ….

Секция JOIN

Тип	Результат
JOIN	В результирующем наборе присутствуют только записи, значения связанных полей в которых совпадают.
LEFT JOIN	В результирующем наборе присутствуют все записи из Table1 и соответствующие им записи из Table2. Если соответствия нет, поля из Table2 будут пустыми
RIGHT JOIN	В результирующем наборе присутствуют все записи из Table2 и соответствующие им записи из Table1. Если соответствия нет, поля из Table1 будут пустыми
FULL JOIN	В результирующем наборе присутствуют все записи из Table1 и соответствующие им записи из Table2. Если соответствия нет – поля из Table2 будут пустыми. Записи из Table2, которым не нашлось пары в Table1, тоже будут присутствовать в результирующем наборе. В этом случае поля из Table1 будут пустыми.
CROSS JOIN	Результирующий набор содержит все варианты комбинации строк из Table1 и Table2. Условие соединения при этом не указывается.

Секция JOIN

Table1

Key1	Field1
1	A
2	B
3	C

Table2

Key2	Field2
1	AAA
2	BBB
2	CCC
4	DDD

```
SELECT Table1.Field1, Table2.Field2
FROM Table1
JOIN Table2 ON Table1.Key1 = Table2.Key2
```

A	AAA
B	BBB
B	CCC

Секция JOIN

SELECT Table1.Field1, Table2.Field2
FROM Table1
LEFT JOIN Table2 **ON** Table1.Key1 = Table2.Key2

A	AAA
B	BBB
B	CCC
C	

SELECT Table1.Field1, Table2.Field2
FROM Table1
RIGHT JOIN Table2 **ON** Table1.Key1 = Table2.Key2

A	AAA
B	BBB
B	CCC
	DDD

Секция JOIN

SELECT Table1.Field1, Table2.Field2
FROM Table1
FULL JOIN Table2 **ON** Table1.Key1 = Table2.Key2

A	AAA
B	BBB
B	CCC
	DDD
C	

SELECT Table1.Field1, Table2.Field2
FROM Table1
CROSS JOIN Table2

A	AAA
A	BBB
A	CCC
A	DDD
B	AAA
B	BBB
B	CCC
B	DDD
C	AAA
C	BBB
C	CCC
C	DDD

Секция WHERE

WHERE [**NOT**] <условие1> [**AND** | **OR** <условие2>]

Условие представляет собой конструкцию вида:

<столбец таблицы, константа или выражение>

<оператор сравнения> *<столбец таблицы, константа или выражение>*

или

IS [NOT] NULL

или

[NOT] LIKE <шаблон>

или

[NOT] IN (<список значений>)

или

[NOT] BETWEEN <нижняя граница> **AND** <верхняя граница>

Операторы сравнения

<	Меньше
<=	Меньше либо равно
<>	Не равно
>	Больше
>=	Больше либо равно
=	Равно

Примеры запросов с операторами сравнения:
*SELECT * FROM Table WHERE Field > 100*
*SELECT * FROM Table WHERE Field1 <= (Field2 + 25)*

Выражение **IS [NOT] NULL** проверяет данные на [не]пустые значения:
*SELECT * FROM Table WHERE Field IS NOT NULL*
*SELECT * FROM Table WHERE Field IS NULL*

Операторы сравнения

[NOT] LIKE - используется при проверке текстовых данных на [не]соответствие заданному шаблону. Символ **‘%’** (процент) в шаблоне заменяет собой любую последовательность символов, а символ **‘_’** (подчеркивание) – один любой символ.

*SELECT * FROM Employees WHERE Name LIKE ‘Иван%’*

SELECT * FROM Employees WHERE Name LIKE ‘__д%’

Операторы сравнения

[NOT] IN проверяет значения на [не]вхождение в определенный список.

*SELECT * FROM Employees WHERE Position IN ('Начальник', 'Менеджер')*

[NOT] BETWEEN проверяет значения на [не] попадание в некоторый диапазон:

*SELECT * FROM Employees WHERE Salary BETWEEN 200 AND 500*

Операторы сравнения

```
SELECT *
FROM Employees
WHERE Position IN ('Начальник', 'Менеджер')
AND Salary BETWEEN 200 AND 500

SELECT *
FROM Employees
WHERE (Position = 'Начальник' OR Position = 'Менеджер')
AND Salary BETWEEN 200 AND 500

SELECT *
FROM Employees
WHERE NOT (Position = 'Начальник' OR Position =
'Менеджер')
```

Секция ORDER BY

ORDER BY - предназначена для сортировки строк результирующего набора данных.

ORDER BY Field1 [ASC | DESC] [, Field2 [ASC | DESC]] [, …]

ASC (по умолчанию) предписывает производить сортировку по возрастанию, а **DESC** – по убыванию.

```
SELECT *
FROM Employees
WHERE Position = 'Начальник'
ORDER BY Salary DESC
```

```
SELECT *
FROM Employees
ORDER BY DeptNum, Salary DESC
```

```
SELECT *
FROM Employees
ORDER BY DeptNum ASC, Salary DESC
```

Групповые функции

Функция	Описание
SUM(Field)	Вычисляет сумму по указанной колонке
MIN(Field)	Вычисляет минимальное значение по указанной колонке
MAX(Field)	Вычисляет максимальное значение по указанной колонке
AVG(Field)	Вычисляет среднее значение по указанной колонке
COUNT(*)	Вычисляет количество строк в результирующей выборке
COUNT(Field)	Вычисляет количество не пустых значений в колонке

SELECT MAX(SALARY)
FROM Employees

SELECT COUNT(*)
FROM Employees

Секция GROUP BY

GROUP BY - разбивает итоговую выборку на подгруппы.

GROUP BY Field1 [, Field2] [, …]

SELECT DeptNum, **MAX**(SALARY)
FROM Employees
GROUP BY DeptNum

В этом случае функция **MAX** будет считаться отдельно для всех записей с одинаковым значением поля **DeptNum**.

Секция HAVING

```
SELECT DeptNum, MAX(SALARY)
FROM Employees
GROUP BY DeptNum
HAVING MAX(SALARY) > 1000
```

Секции **HAVING** и **WHERE** взаимно дополняют друг друга. Сначала с помощью ограничений **WHERE** формируется итоговая выборка, затем выполняется разбивка на группы по значениям полей, заданных в **GROUP BY**. Далее по каждой группе вычисляется групповая функция и в заключение накладывается условие **HAVING**.

INSERT

INSERT INTO *<имя таблицы> [(<список имен колонок>)]*
VALUES*(<список констант>)*

INSERT INTO Employees(TabNum, Name, Position, DeptNum, Salary)
VALUES (5, ‘Сергеев’, ‘Старший менеджер’, 15, 850)

Employees

TabNum	Name	Position	DeptNum	Salary
1	Иванов	Начальник	15	1000
2	Петров	Инженер	15	500
3	Сидоров	Менеджер	10	700
45	Сергеев	Старший менеджер	15	850

INSERT

INSERT INTO Employees(TabNum, Name, DeptNum, Salary)
VALUES (45, ‘Сергеев’, 15, 850)

Employees

TabNum	Name	Position	DeptNum	Salary
1	Иванов	Начальник	15	1000
2	Петров	Инженер	15	500
3	Сидоров	Менеджер	10	700
45	Сергеев		15	850

INSERT INTO Employees
VALUES (45, ‘Сергеев’, ‘Старший менеджер’, 15, 850)

INSERT INTO Employees
VALUES (45, ‘Сергеев’, **NULL**, 15, 850)

INSERT

INSERT INTO <имя таблицы> [(<список имен колонок>)]
<команда **SELECT**>

INSERT INTO Table1(Field1, Field2)
SELECT Field3, (Field4 + 5) **FROM** Table2

DELETE

***DELETE FROM** <имя таблицы> [**WHERE** <условия поиска>]*

Если опустить секцию условий поиска **WHERE**, из таблицы будут удалены все записи. Иначе – только записи, удовлетворяющие критериям поиска. Форматы секций **WHERE** команд **SELECT** и **DELETE** аналогичны.

DELETE FROM Employees

DELETE FROM Employees **WHERE** TabNum = 45

UPDATE

UPDATE < имя таблицы>
SET <имя колонки> = <новое значение> , <имя колонки> = <новое значение>, …
WHERE <условия поиска>]

UPDATE Employees
SET Salary = Salary + 100

UPDATE Employees
SET Position = ‘Старший менеджер’, Salary = 1000
WHERE TabNum = 45 **AND** Position **IS NULL**

CREATE TABLE

CREATE TABLE <имя таблицы>
(
<имя колонки> <тип колонки>[(<размер колонки>)]
[<ограничение целостности уровня колонки>]
[, <имя колонки> <тип колонки>[(<размер колонки>)]
[<ограничение целостности уровня колонки>]]
[, …]
[<ограничение целостности уровня таблицы>]
[,<ограничение целостности уровня таблицы>]
[, …]
)

CREATE TABLE

```
CREATE TABLE Departments
(
DeptNum int NOT NULL PRIMARY KEY,
Name varchar(80) NOT NULL
)

CREATE TABLE Employees
(
TabNum int NOT NULL PRIMARY KEY,
Name varchar(100) NOT NULL,
Position varchar(200),
DeptNum int,
Salary decimal(10, 2) DEFAULT 0,
CONSTRAINT FK_DEPARTMENT FOREIGN KEY (DeptNum)
REFERENCES Departments(DeptNum)
)
```

ALTER TABLE

Команда **ALTER TABLE** позволяет добавлять новые колонки и ограничения целостности, удалять их, менять типы колонок, переименовывать колонки.

ALTER TABLE Departments **ADD COLUMN** City int

ALTER TABLE Departments **DROP COLUMN** City

ALTER TABLE Departments **ADD**

CONSTRAINT FK_City

FOREIGN KEY (City)

REFERENCES Cities(City)

ALTER TABLE Departments **DROP CONSTRAINT**

FK_City

DROP TABLE

Удаление ранее созданной таблицы производится командой DROP TABLE:

DROP TABLE <Название таблицы>

Вопросы?