Европейски съюз

ОПАК. Експерти в действие

Европейски социален фонд
Инвестиции в хората

**Проект „Повишаване квалификацията на служителите от администрацията на централно ниво чрез усъвършенстване на знанията и практическите им умения за управление на софтуерни ИТ проекти в съответствие със съвременните методологии“, осъществяван с финансовата подкрепа на Оперативна програма „Административен капацитет” (ОПАК), съфинансирана от Европейския съюз, чрез Европейския социален фонд”, съгласно Договор № К13-22-1/05.03.2014 г.**

# НАРЪЧНИК

## ДЕЙНОСТ 3.
## ПРОВЕЖДАНЕ НА ОБУЧЕНИЕ ПО БАЗИ ДАННИ (ДИЗАЙНЕРИ И ИНЖЕНЕРИ) ЗА 60 СЛУЖИТЕЛИ НА ЦЕНТРАЛНАТА АДМИНИСТРАЦИЯ И ИЗДАВАНЕ НА СЕРТИФИКАТИ ЗА ПРОВЕДЕНОТО ОБУЧЕНИЕ

Изготвен в изпълнение на Договор № Д-37/11.12.2014 г.
между
МИНИСТЕРСТВО НА ТРАНСПОРТА, ИНФОРМАЦИОННИТЕ ТЕХНОЛОГИИ И СЪОБЩЕНИЯТА
и
„КОНСОРЦИУМ ИТ ОБУЧЕНИЯ 2015“ДЗЗД

New Horizons Computer Learning Centers

ciela norma

**„КОНСОРЦИУМ ИТ ОБУЧЕНИЯ 2015“ ДЗЗД**

**София 1040, ж.к. Изток, бул. Драган Цанков 36, СТЦ Интерпред, блок А, ет.6; тел: 024210040; имейл: ittraining2015@newhorizons.bg;**

**Автор:**

Стефан Георгиев

**Одобрил:** Николай Пенев – ръководител на проекта

София, 2015 г.

## Съдържание

## Въведение

Този наръчник е част от учебните материали по бази данни за служители на централната администрация. Той има за цел да подпомогне обучителя и обучавания при подготовката му по бази данни.
Наръчникът няма за цел да бъде книга по бази данни, а в структуриран вид в допълнение на проведения обучителен курс да даде препратки към места в глобалната мрежа, където да получите по-задълбочени познания.

В наръчника се прави въведение в базите данни и последователно надгражда умения за проектиране, дизайн, работа и поддръжка на системи за управление на релационни бази от данни. Документът включва и модули, свързани с обработка и анализ на данни.

# Речник на използваните термини

Речникът е организиран в последователен хронологичен ред според използването на термините.

**СУБД** – система за управление на бази данни.

**Бази данни** – организирана колекция от данни за представяне на определена информация.

**Данни** – набор от стойности, които съхраняваме в базата.

**Информация** – данни, които са извлечени и организирани по определен начин.

**OLTP** (online transactional processing) - транзакционни системи за обработка на данни. Фокусът в тези системи е насочен към бързото въвеждане и промяна на данни.

**OLAP** (online analytical processing) – аналитични системи за обработка на данни. Фокусът в тези системи е насочен към генериране и анализ на данни.

**Primary key –** поле от таблица дефинирано като първичен ключ. Основни характеристики на такова поле са поддържането на уникални стойности и недопускането на празни стойности.

**Foreign Key -** поле от таблица дефинирано като външен ключ. Външният ключ прави референция към първичен ключ на друга таблица, с което се дефинира свързаност на данните от една таблица с друга.

**SQL (**Structured Query Language) – структуриран език за заявки**.** Основен език за манипулиране на данни в СУБД.

**DDL** (Data Definition language) – подразделение на SQL езика за дефиниране на обектите и тяхната структура в СУБД.

**DML** (Data Manipulation language) - подразделение на SQL езика за обработка на данни и извличане на справки от СУБД.

**DCL** (Data Control Language) - подразделение на SQL езика за дефиниране на достъпа до обектите в СУБД.

**Инстанция в база данни** – набор от работещи процеси ползващи единни памет, бази данни и системи за управление на сигурността.

**ORACLE_HOME** – местоположението където е инсталирано ORACLE програмно осигуряване.

**ORACLE_SID** (Oracle System Identifier/ Oracle Site Identifier) – идентификатор на ORACLE система, чрез който тази система да бъде достъпна за потребители.

**SGA** (System Global Area) – област памет заделена от Oracle СУБД необходима за нуждите на системата.

**Сървърен процес** – процес, който обслужва дадена потребителска сесия, която отправя заявки към ORACLE СУБД.

**PGA** (Program Global Area) – област памет необходима за работата на всеки един инициран сървърен процес в ORACLE СУБД.

**Системна таблица** – таблица съдържаща служебни и конфигурационни данни необходими за инициализиране и работа на ORACLE СУБД.

**DATA DICTIONARY** – термин описващ системните таблици в ORACLE СУБД.

**Cold Backup** – архивиране на база данни при спряна СУБД.

**Online Backup** – архивиране на база данни при работеща СУБД.

**CBO** (Cost-Based Optimizer) – механизъм в СУБД, който организира създаването на план за изпълнение на потребителски заявки на база оптимално използване на ресурси.

MS SQL Server - СУБД на фирмата Microsoft.

**TDS**( Tabular Data Stream) - специализиран протокол за достъп до MS SQL server СУБД.

**SNAC** (SQL Native Access Client) – специализиран клиент предоставящ интерфейс за достъп до MS SQL Server СУБД.

**BI** (Business Intelligence) – термин използван в СУБД за описание на инструменти и механизми за извличане на данни от СУБД с цел генериране на справки и анализи необходими за нуждите на работещите със СУБД потребители.

**MSDN** (Microsoft Developer Network) – общо наименование на интернет страници съдържащи техническа документация на Microsoft продукти, включително и системи за управление на данни.

**SSMS** (SQL Server Management Studio) – програмен продукт предоставящ възможности както за конфигуриране на MS SQL Server, така и за генериране на справки в СУБД.

**SSIS** (SQL Server Integration Services) – програмен продукт предоставящ възможности както за обмен на данни между различни СУБД, така и за генериране на справки и анализи, чрез използване на данни от тези СУБД.

# Модул 1: Въведение в базите данни.

**Обща дефиниция.**

По дефиниция базите от данни представляват организирана колекция от данни за представяне на определена информация.

От това определение произлизат други две други определения.

**Данни** – това са набор от стойности, които съхраняваме в базата.

**Информация** – данни, които са извлечени и организирани по определен начин.

Данните се съхраняват в системи за управление на бази данни. Това са компютърни системи, които имат изградени средства и механизми за поддръжка и обработка на данни.

Системите за управление на данни са вече с дълга история и с течение на времето и развитието на технологиите са били допълвани с нови средства и механизми подобряващи работата с данни и извличането на информация. Това е позволило да бъдат разрешени редица проблеми свързани с поддръжката на данни.

**Типове бази данни.**

Като тип системите за управление на бази данни се разделят на еднопотребителски и многопотребителски.

**Еднопотребителските** системи за управление на бази данни, известни още като локални са системи проектирани за работа с един потребител. Типичен представител на тези системи е Microsoft Access. Той е в състава на Microsoft Office и е вече с дълга история. Може да бъде използван за връзка към многопотребителски бази.

Въпреки, че Microsoft Access е предназначен за предимно за еднопотребителска работа в тази система има изградени средства, с които да бъде ползвана и от няколко потребителя. Препоръчително е броят на потребителите да не надвишава 5.

**Многопотребителските** системи за управление на бази данни са проектирани за работа от множество потребители. Въпреки това броят на потребителите не е неограничен. Фактори, които влияят върху броя на потребителите, които системата може да обслужи е оперативната RAM памет предоставена на системата за управление, мрежовата пропускливост и други.

**Модели на бази данни**

Като модел базите данни могат да бъдат йерархични, мрежови, релационни и обектно ориентирани.

В курса ще бъдат разгледани релационните бази данни. Информация за другите модели може да се намери в интернет на адрес: http://en.wikipedia.org/wiki/Database

Релационният модел е предложен през 70-те години на 20-ти век от Едгар Код. При този модел основния обект, който съдържа данните е таблица. Данните се съхраняват в множество таблици като списъци по определени теми като помежду им съществуват връзки изградени с помощта на ключови полета.

**Основни архитектури при бази данни**

Използват се две основни архитектури. OLTP и OLAP.

**OLTP** (OnLine Transactional Processing) – системи с транзакционна архитектура предназначени за транзакционно ориентирани системи. Това са системи улесняващи операциите свързани с манипулирането на данни.

**OLAP** (OnLine Analitical Processing) – системи с аналитична архитектура предназначени за аналитична обработка на информацията. Основната задача е генерирането на справки и анализи.

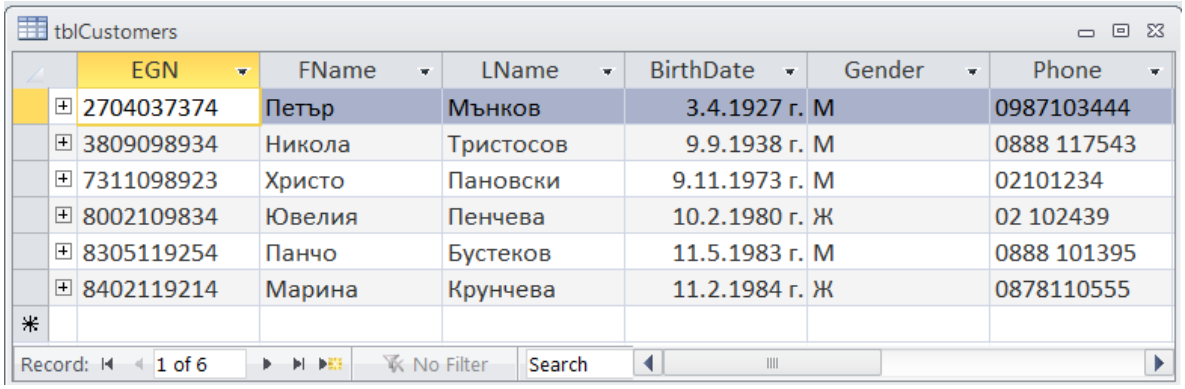
**Въведение в релационните системи за управление на бази данни.**

**Определение -** системи предоставящи методи и средства за управление на съвкупност от свързани данни. Допълнителна информация може да се намери на: http://en.wikipedia.org/wiki/Relational_database

**Типове** – локални и многопотребителски. Тези типове са описани по-горе.

**Архитектури** – транзакционни (OLTP) и аналитични (OLAP).

**Основен обект** – основният обект който съдържа потребителските данни е таблица.


tblCustomers

| EGN | FName | LName | BirthDate | Gender | Phone |
|---|---|---|---|---|---|
| 2704037374 | Петър | Мънков | 3.4.1927 г. | М | 0987103444 |
| 3809098934 | Никола | Тристосов | 9.9.1938 г. | М | 0888 117543 |
| 7311098923 | Христо | Пановски | 9.11.1973 г. | М | 02101234 |
| 8002109834 | Ювелия | Пенчева | 10.2.1980 г. | Ж | 02 102439 |
| 8305119254 | Панчо | Бустеков | 11.5.1983 г. | М | 0888 101395 |
| 8402119214 | Марина | Крунчева | 11.2.1984 г. | Ж | 0878110555 |

Record: 1 of 6 No Filter Search

Таблиците съхраняват данни само за една тема, организирани под формата на полета и записи.

**Полета** (колони) – съдържат атрибути по темата.

**Записи** (редове) – съдържат всички атрибути на една инстанция по темата.

**Ключови полета в базите данни**

- ✓ **Първичен ключ (Primary key)** – едно или няколко полета, с уникална стойност за всеки запис. Първичният ключ има две задължителни характеристики. В такова поле не се допуска да има повтарящи се стойности. такова поле не се допускат празни стойности.
- ✓ **Външен ключ (Foreign Key)** – поле от друга таблица, което се свързва с първичния ключ. Външният ключ е поле дефинирано в една таблица, което реферира към първичния ключ на друга таблица, с която дадената таблица трябва да има връзка.

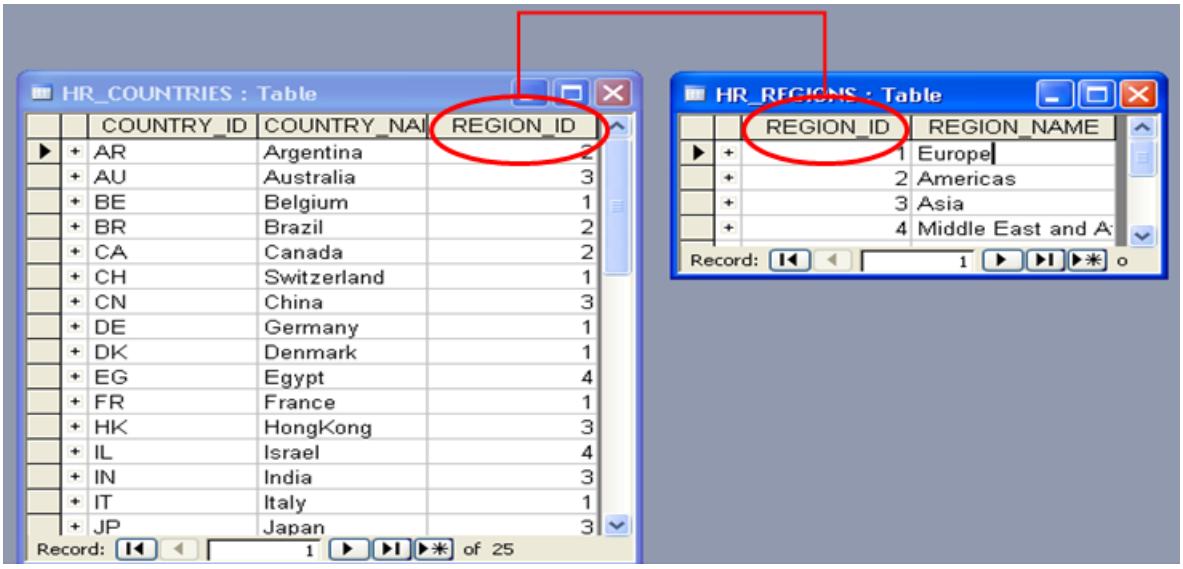
**Връзки между таблици**

**Определение** - начинът, по който данните от една таблица са свързани с данните от друга таблица. Допълнителна информация може да се намери на: http://en.wikipedia.org/wiki/Relational_database

**Типове/Видове връзки между таблици**

- Едно към едно - връзка между първичните ключове на две таблици.
- Едно към много - връзка между първичен ключ и външен ключ.
- Много към едно - връзка между външен ключ и първичен ключ.
- Много към много - връзка, при която участват три таблици. Една от таблиците трябва да съдържа външни ключове към първичните ключове на другите две таблици. Пример: ако имаме таблица за срещите между клиенти и служители, то в тази таблица ще фигурират поле-идентификатор на служител, поле поле-идентификатор на клиент както и други полета описващи срещата като дата и час. Свързаността, между полетата идентификатор на служител и идентификатор на клиент представлява връзката много към много.

Пример за връзка много към едно:




В таблицата HR_REGIONS полето REGION_ID е първичен ключ. В таблицата HR_COUNTRIES полето REGION_ID е външен ключ реферираш към таблицата HR_REGIONS и нейното полето REGION_ID. Стойностите в полето REGION_ID на таблицата HR_COUNTRIES, трябва да имат съответствие със стойностите на полето REGION_ID от таблицата HR_REGIONS.

**Основни обекти в релационните бази данни.**

Таблици – допълнителна ция: http://en.wikipedia.org/wiki/Table_(database)

Изгледи – обект в база данни съдържащ заявка за извличане на данни от база-та. Допълнителна информация: http://en.wikipedia.org/wiki/View_(SQL)

Индекси – механизъм за ускоряване на търсенето в база данни.

Допълнителна информация: http://en.wikipedia.org/wiki/Database_index

Процедури – обект в база данни съдържащ програмен код за изпълнение на действия по поддръжката и манипулирането на данни.

Допълнителна информация: http://en.wikipedia.org/wiki/Stored_procedure

# Модул 2: Основи на проектирането на бази данни

**Жизнен цикъл в разработката на софтуерни системи.**

- ✓ Анализ на изискванията.
- ✓ Проектиране
- ✓ Разработка
- ✓ Внедряване
- ✓ Продуктова фаза

**Анализ на изискванията.**

Определение – методи и техники за извличане и анализ на изискванията към определена софтуерна система.

Анализ на изискванията включва:

- Събиране на данни от заинтересовани лица.
- Анализ на събраната информация.
- Съпоставяне на технически възможности с изискванията на заинтересованите лица.

**Нормализация на база данни.**

Определение – процес, включващ набор от правила за проектиране на база от данни, наречени нормални форми. Процесът на нормализация на база от данни води до:

- ✓ отстраняване на повторения сред данните.
- ✓ икономия на памет и повишено бързодействие.
- ✓ предпазване от аномалии при манипулирането с данните (вмъкване, актуализиране и изтриване) и от загуба на тяхната цялост.
- ✓ осигуряване на оптимална структура на базата от данни.

Допълнителна информация на: http://en.wikipedia.org/wiki/Database_normalization

**Правила за нормализация на база данни.**

- ✓ **Правило 1** – всяко поле трябва да представя еднозначно уникален тип информация.
- ✓ **Правило 2** – всяка таблица трябва да притежава уникален идентификатор-ключ. Всяка колона без ключ трябва да бъде напълно зависима от целия първичен ключ.
- ✓ **Правило 3** – всяко неключово поле трябва да е независимо от другите неключови полета .

**Идентифициране на ключови стойности.**

Стойностите в полетата определени като ключови могат да бъдат от различни типове данни – число, текст или дата. За предпочитане са числовите стойности.

При осъществяването на връзки между таблици основна роля играят и двата вида ключови полета. Първичен ключ и външен ключ.

**Първичен ключ.**

По определение първичният ключ е съставен от едно или повече полета, като в него не се допуска повтаряне на стойности и празни стойности.

За стойности в първичен ключ, могат да се използват както атрибути-характеристики на елемента описват в записа, така и системно зададени стойности. В зависимост от задачата, която се решава могат да се избират едните или другите стойности. Например като характеристика еднозначно определяща гражданин на Република България се използва неговия ЕГН. Ако трябва да имаме таблица с граждани на Република България, то може да се използва ЕГН за първичен ключ. Ако обаче ще поддържаме данни за хора от различни страни, то по-сполучливо е да се използва системно зададен номер.

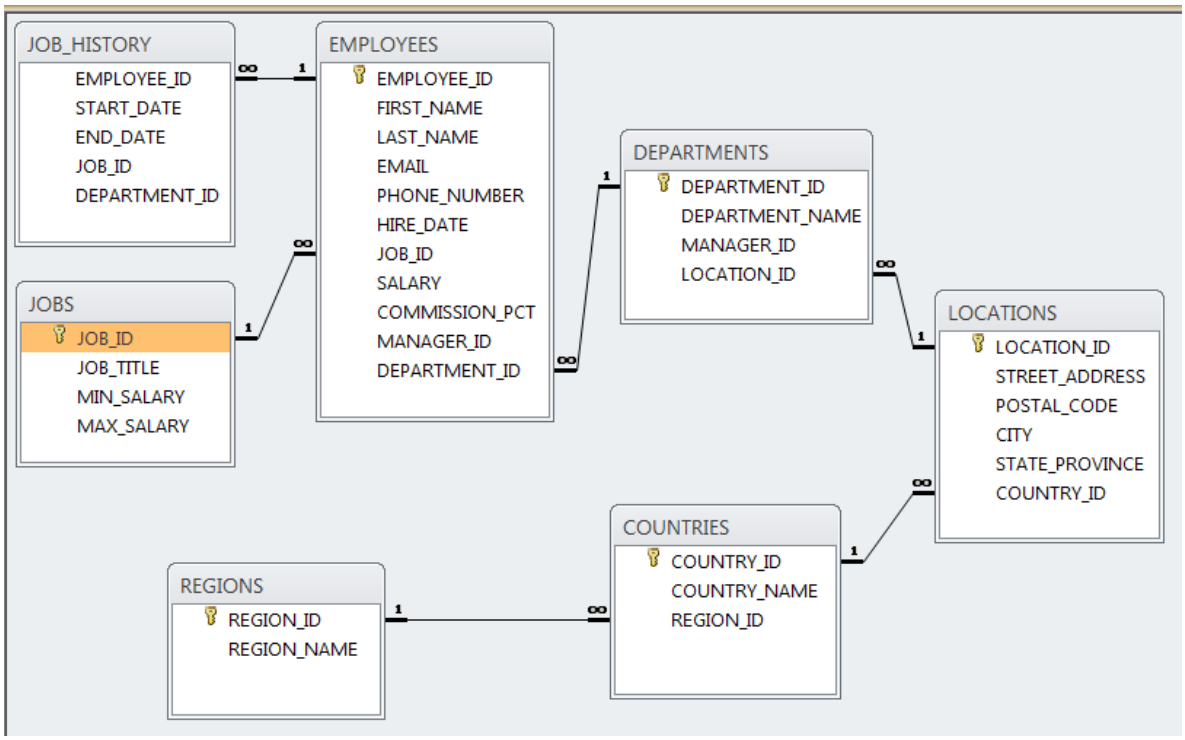
**Външен ключ.**

Т.к. външният ключ прави референция към първичен ключ, то той трябва да е съставен от толкова поета колкото и съответстващия му първичен ключ. Типа на полето/полетата от външния ключ трябва да съвпада с типа на съответстващия първичен ключ.

**Диаграма на връзки между таблици.**

За представяне на връзките между таблици могат да се използват както текстови така и графични описания. Графичната диаграма на връзките е един удобен вариант, при който ясно и точно се виждат таблиците и връзките между ключовите полета.

Пример:




Типовете връзки между таблиците са обяснение в модул 1.

**Диаграми на използваните в модулите за SQL бази.**

Схема Student2.

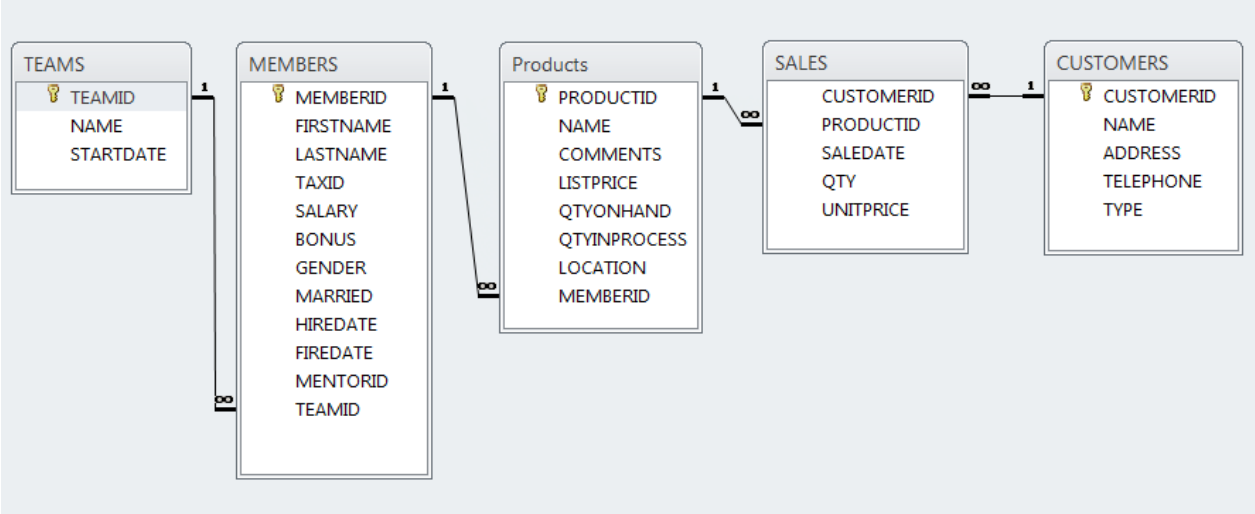
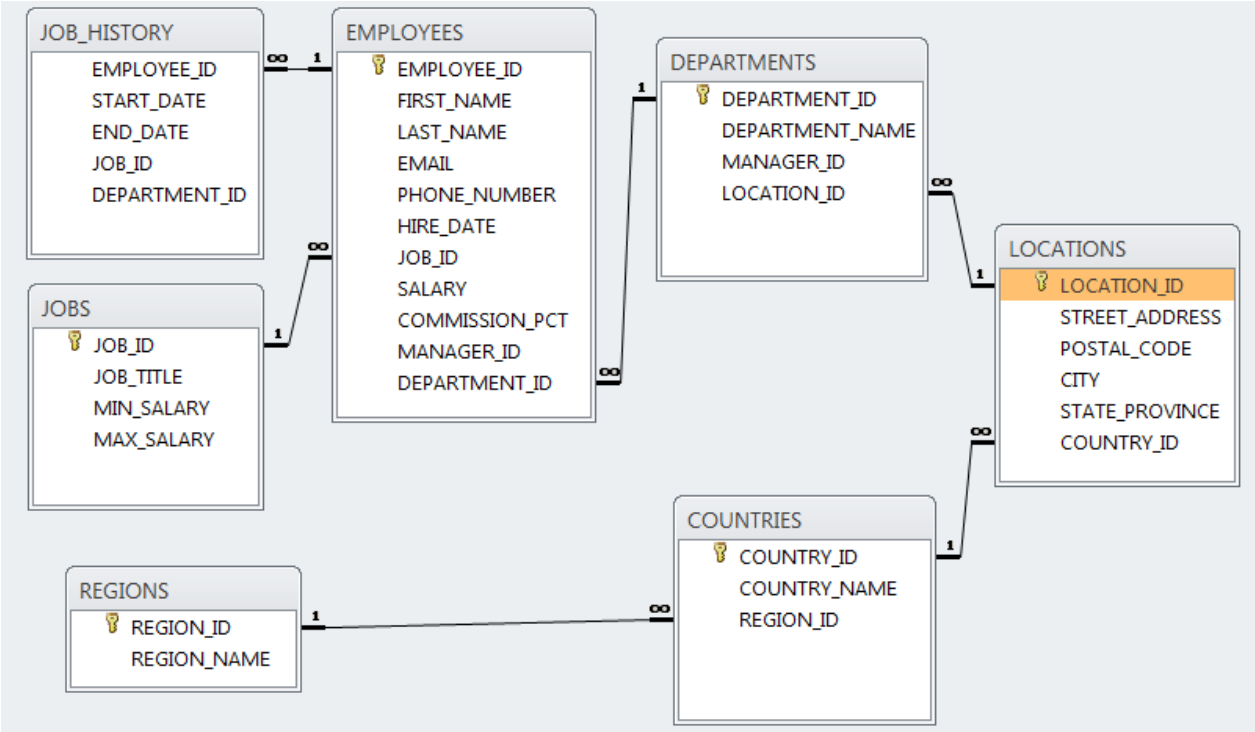



Схема HR.



# Модул 3: Въведение в SQL.

**Езикът за заявки SQL.**

SQL - Structured Query Language

- Разработен от IBM през1970-те.
- Стандартизиране от ANSI и ISO стандарти.
- Широко използван в софтуерната индустрия.
- Разновидности:
  - ✓ PL/SQL,
  - ✓ SQL  Procedural Language
  - ✓ Transact-SQL

**Подразделения на SQL.**

- **DDL** – Data Definition language

  Към DDL спадат команди като CREATE, DROP, ALTER

- **DML** – Data Manipulation language

  Към DML спадат команди като SELECT, INSERT, UPDATE, DELETE

- **DCL** – Data Control Language

  Към DCL спадат команди като GRANT, REVOKE

Допълнителна информация може да се на мери на: http://bg.wikipedia.org/wiki/SQL

**Интерфейси за работа.**

- Елементи на връзката към база данни.
  - Host name – компютър, на който е инсталирана система за управление на бази данни.
  - Port – комуникационна точка на компютърна операционна система
  - SID – системен идентификатор. Име на инстанция в система за управление на база данни ORACLE
  - User Authentication – автетнификация на потребител чрез използване на име и парола.

- Редактори и среди за работа със системи за управление на бази данни.
  - SQL Plus – информация на: http://en.wikipedia.org/wiki/SQL*Plus
  - SQL Developer - http://en.wikipedia.org/wiki/Oracle_SQL_Developer
  - SSMS - http://en.wikipedia.org/wiki/SQL_Server_Management_Studio

**Създаване на елементарни SELECT заявки.**

SELECT конструкция

- Общо описание – за отправяне на заявки към система за управление на релационни бази данни. SELECT конструкцията се състои от клаузи. Клауза в една заявка наричаме ключова дума, която указва използването на определени елементи и структури в базата данни.
- Обща структура – всяка SELECT конструкция се състои от две задължителни клаузи. Клаузата SELECT и клаузата FROM. SELECT конструкцията завършва със символа **;** (точка и запетая). Например ако трябва да се извлече определена колона от дадена таблица, то трябва да се използва конструкция от следния тип:

  *SELECT <име на колона> FROM <име на таблица>;*

- Могат да се използват различни варианти на SELECT конструкция. Например SELECT конструкция с изброяване на няколко колони в SELECT клаузата:

  *SELECT <име на колона_1>, <име на колона_2>... <име на колона_K>*
  *FROM <име на таблица>;*

- SELECT конструкция с извличане на всички колони от таблица може да се създаде като се опишат всички колони в таблицата или чрез използване на символа *****.

  *SELECT * FROM <име на таблица>;*

**Клаузи на SELECT конструкция.**

| Клауза | Съставена от | Роля |
|---|---|---|
| SELECT | <селективен списък> | Дефинира колоните, които ще формират резултата. |
| FROM | <изходни таблици> | Дефинира таблиците в заявката. |
| WHERE | <условие за филтриране> | Филтрира редовете в заявката. |
| GROUP BY | <списък групиране> | Организира редовете в групи. |
| HAVING | <условие за филтриране> | Филтриране по групи. |
| ORDER BY | <списък сортиране> | Сортиране на резултата |

**Последователност на изпълнение на клаузите в заявка -** редът, в който е написана една заявка не е реда, по който се обработва от Релационната система за управление на бази данни.

```
5: SELECT <селективен списък>

1: FROM   <изходни таблици>

2: WHERE  <условие за филтриране>

3: GROUP BY <списък групиране>

4: HAVING <условие за филтриране>

6: ORDER BY <списък сортиране>
```

**Примери за SELECT конструкция с таблици от демонстрационната база.** В примерите се ползва схема Student2.

```
SELECT * FROM JOBS;

SELECT job_title, min_salary, max_salary FROM jobs;

SELECT first_name,last_name,salary FROM employees;

SELECT jobs.job_title, jobs.min_salary FROM jobs;
```

Фрагмент от резултата на последния пример -

| JOB_TITLE | MIN_SALARY |
|---|---|
| President | 20080 |
| Administration Vice President | 15000 |
| Administration Assistant | 3000 |
| Finance Manager | 8200 |
| Accountant | 4200 |

Производни колони – получават се като се използват математически или текстови операции в SELECT клаузата.

Пример 1: полето salary се умножава по 1.1.

```
SELECT first_name, last_name, salary, salary*1.1, 'USD'
FROM employees;
```

Фрагмент от резултата на този пример

| FIRST_NAME | LAST_NAME | SALARY | SALARY*1.1 |
|---|---|---|---|
| Steven | King | 24000 | 26400 |
| Neena | Kochhar | 17000 | 18700 |
| Lex | De Haan | 17000 | 18700 |
| Alexander | Hunold | 9000 | 9900 |
| Bruce | Ernst | 6000 | 6600 |

Пример 2: Слепване на текстови колони чрез използване на символа `||`.

```
SELECT first_name || ' ' || last_name FROM employees;
```

Фрагмент от резултата на този пример

| FIRST_NAME\|\|''\|\|LAST_NAME |
|---|
| Ellen Abel |
| Sundar Ande |
| Mozhe Atkinson |
| David Austin |
| Hermann Baer |

**Коментари –** коментарите се използват са описание на код в редактора за писане на заявки. Коментарите в SQL се отбелязват с двоен символ - . В някои редактори използването на два последователни символа – води и различно оцветяване на текста, за да се подчертае, че това е коментар. Например:

```
-- това е коментар

SELECT first_name || ' ' || last_name
FROM employees; -- коментар: слепване на полета
```

**Клауза DISTINCT –** премахва дублирането на редове в резултата. Всички редове, които се повтарят се редуцират до един в резултатната таблица. В примерите се ползва схема Student2.

```
SELECT DISTINCT salary FROM members;
```

Резултат от този пример

| SALARY |
|---|
| 50000 |
| 30000 |
| 35000 |
| 75000 |
| 0 |

```
SELECT DISTINCT salary, gender FROM members;
```

Резултат от този пример

| SALARY | GENDER |
|---|---|
| 30000 | F |
| 30000 | M |
| 0 | M |
| 35000 | F |
| 75000 | F |
| 50000 | M |
| 75000 | M |

**Използване на изрази и псевдоними на колони в SELECT клаузата.** В примерите се ползва схема Student2.

```
SELECT firstname, lastname, salary, salary*1.1 AS "New Salary"
FROM members;
```

Резултат от този пример

| FIRSTNAME | LASTNAME | SALARY | New Salary |
|---|---|---|---|
| John | Jones | 75000 | 82500 |
| Mary | Jones | 30000 | 33000 |
| Antonios | Vamvakeros | 30000 | 33000 |
| Louisa | Lindboe | 30000 | 33000 |
| Cara | Montopoli | 35000 | 38500 |
| Michael | Dupre | 50000 | 55000 |
| Migumi | Zona | 75000 | 82500 |
| Wey | Ho | 30000 | 33000 |
| Joseph | Smith | 0 | 0 |

```
SELECT firstname || ' ' || lastname AS Employee
FROM members;
```

Резултат от този пример

| EMPLOYEE |
|---|
| JohnJones |
| MaryJones |
| AntoniosVamvakeros |
| LouisaLindboe |
| CaraMontopoli |
| MichaelDupre |
| MigumiZona |
| WeyHo |
| JosephSmith |

**Използване на псевдоними на таблици** - всяка колона в резултатната таблица трябва да име име. То може да бъде името на полето от таблицата в базата или да се замени само в рамките на заявката с друго име наречено псевдоним**.** В примерите се ползва схема Student2.

```
SELECT mem.firstname, mem.lastname FROM members mem;

SELECT mem.* FROM members mem;
```

Резултат от последния пример

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 500 | John | Jones | 000000000 | 75000 | 5000 | M | Y | 01-FEB-90 | (null) | (null) | 400 |
| 501 | Mary | Jones | 111111111 | 30000 | (null) | F | N | 02-MAR-02 | (null) | 506 | 401 |
| 502 | Antonios | Vamvakeros | 222222222 | 30000 | (null) | M | N | 15-MAR-02 | (null) | 506 | 401 |
| 503 | Louisa | Lindboe | 333333333 | 30000 | 1000 | F | Y | 01-JAN-02 | (null) | 506 | 401 |
| 504 | Cara | Montopoli | 444444444 | 35000 | 2000 | F | Y | 01-JAN-01 | (null) | 506 | 401 |
| 505 | Michael | Dupre | 555555555 | 50000 | 4000 | M | Y | 12-DEC-97 | (null) | 500 | 402 |
| 506 | Migumi | Zona | 666666666 | 75000 | 5000 | F | N | 01-MAR-90 | (null) | 500 | 400 |
| 507 | Wey | Ho | 777777777 | 30000 | (null) | M | N | 01-FEB-90 | (null) | 505 | 402 |
| 508 | Joseph | Smith | 888888888 | 0 | 0 | M | N | 01-FEB-90 | 15-JUN-02 | (null) | (null) |

**Филтриране на данни с WHERE клауза.**

Когато я има WHERE клаузата се изпълнява след FROM клаузата.

*SELECT <списък колони>*

*FROM <име на таблица>*

***WHERE** <критерий за филтриране>;*

WHERE клаузата се използва за задаване на критерии за филтриране. Филтрирането става по определени стойности на указани полета. При задаването на критерии се използват предикати - оператори за сравнение, логически оператори и допълнителни оператори с предварително зададена функционалност.

**WHERE клауза - примери с таблици от демонстрационната база.** В примерите се ползва схема Student2.

**Оператори за сравнение.**

```
SELECT * FROM members
WHERE salary = 30000;

SELECT * FROM members
WHERE salary >= 30000;

SELECT * FROM members
WHERE gender='F';

SELECT * FROM members
WHERE hiredate='01-jan-01';
```

**Логически оператори. Логическо И (AND). Логическо ИЛИ (OR). Логическо отрицание (NOT).**

```
SELECT * FROM members
WHERE salary>40000 AND salary<60000;
```

Резултат от този пример

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 505 | Michael | Dupre | 555555555 | 50000 | 4000 | M | Y | 12-DEC-97 | (null) | 500 | 402 |

```
SELECT * FROM members
WHERE salary=35000 OR salary=75000;
```

Резултат от този пример

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 500 | John | Jones | 000000000 | 75000 | 5000 | M | Y | 01-FEB-90 | (null) | (null) | 400 |
| 501 | Mary | Jones | 111111111 | 30000 | (null) | F | N | 02-MAR-02 | (null) | 506 | 401 |
| 502 | Antonios | Vamvakeros | 222222222 | 30000 | (null) | M | N | 15-MAR-02 | (null) | 506 | 401 |
| 503 | Louisa | Lindboe | 333333333 | 30000 | 1000 | F | Y | 01-JAN-02 | (null) | 506 | 401 |
| 506 | Migumi | Zona | 666666666 | 75000 | 5000 | F | N | 01-MAR-90 | (null) | 500 | 400 |
| 507 | Wey | Ho | 777777777 | 30000 | (null) | M | N | 01-FEB-90 | (null) | 505 | 402 |

```
SELECT * FROM members
WHERE salary= 30000 AND Gender = 'M';

SELECT * FROM members
WHERE salary= 30000 OR Gender = 'M';

SELECT * FROM members
WHERE salary<=40000 OR salary>=60000;
```

```
SELECT * FROM members
WHERE NOT (salary>40000 AND salary<60000);
```

Резултат от този пример

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 500 | John | Jones | 000000000 | 75000 | 5000 | M | Y | 01-FEB-90 | (null) | (null) | 400 |
| 501 | Mary | Jones | 111111111 | 30000 | (null) | F | N | 02-MAR-02 | (null) | 506 | 401 |
| 502 | Antonios | Vamvakeros | 222222222 | 30000 | (null) | M | N | 15-MAR-02 | (null) | 506 | 401 |
| 503 | Louisa | Lindboe | 333333333 | 30000 | 1000 | F | Y | 01-JAN-02 | (null) | 506 | 401 |
| 504 | Cara | Montopoli | 444444444 | 35000 | 2000 | F | Y | 01-JAN-01 | (null) | 506 | 401 |
| 506 | Migumi | Zona | 666666666 | 75000 | 5000 | F | N | 01-MAR-90 | (null) | 500 | 400 |
| 507 | Wey | Ho | 777777777 | 30000 | (null) | M | N | 01-FEB-90 | (null) | 505 | 402 |
| 508 | Joseph | Smith | 888888888 | 0 | 0 | M | N | 01-FEB-90 | 15-JUN-02 | (null) | (null) |

**NULL стойности** – ако клетка от таблица е празна се казва, че клетката съдържа NULL ст-ст. NULL не подлежи на сравняване чрез оператора за сравнение =.

```
SELECT * FROM members
WHERE bonus>=0;
```

Резултат от този пример

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 500 | John | Jones | 000000000 | 75000 | 5000 | M | Y | 01-FEB-90 | (null) | (null) | 400 |
| 503 | Louisa | Lindboe | 333333333 | 30000 | 1000 | F | Y | 01-JAN-02 | (null) | 506 | 401 |
| 504 | Cara | Montopoli | 444444444 | 35000 | 2000 | F | Y | 01-JAN-01 | (null) | 506 | 401 |
| 505 | Michael | Dupre | 555555555 | 50000 | 4000 | M | Y | 12-DEC-97 | (null) | 500 | 402 |
| 506 | Migumi | Zona | 666666666 | 75000 | 5000 | F | N | 01-MAR-90 | (null) | 500 | 400 |
| 508 | Joseph | Smith | 888888888 | 0 | 0 | M | N | 01-FEB-90 | 15-JUN-02 | (null) | (null) |

```
SELECT * FROM members
WHERE bonus IS NULL;
```

Резултат от този пример

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 501 | Mary | Jones | 111111111 | 30000 | (null) | F | N | 02-MAR-02 | (null) | 506 | 401 |
| 502 | Antonios | Vamvakeros | 222222222 | 30000 | (null) | M | N | 15-MAR-02 | (null) | 506 | 401 |
| 507 | Wey | Ho | 777777777 | 30000 | (null) | M | N | 01-FEB-90 | (null) | 505 | 402 |

```
SELECT * FROM members
WHERE bonus IS NOT NULL;
```

Резултат от този пример

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 500 | John | Jones | 000000000 | 75000 | 5000 | M | Y | 01-FEB-90 | (null) | (null) | 400 |
| 503 | Louisa | Lindboe | 333333333 | 30000 | 1000 | F | Y | 01-JAN-02 | (null) | 506 | 401 |
| 504 | Cara | Montopoli | 444444444 | 35000 | 2000 | F | Y | 01-JAN-01 | (null) | 506 | 401 |
| 505 | Michael | Dupre | 555555555 | 50000 | 4000 | M | Y | 12-DEC-97 | (null) | 500 | 402 |
| 506 | Migumi | Zona | 666666666 | 75000 | 5000 | F | N | 01-MAR-90 | (null) | 500 | 400 |
| 508 | Joseph | Smith | 888888888 | 0 | 0 | M | N | 01-FEB-90 | 15-JUN-02 | (null) | (null) |

**Използване на допълнителни оператори.**

Оператор IN(списък стойности) е заместител на няколко OR оператора.

```
SELECT * FROM members
WHERE salary =30000 OR salary =35000 OR salary =50000 OR
salary =75000;
SELECT * FROM members
WHERE salary IN(30000, 35000, 50000, 75000);
```

Двете заявки връщат еднакъв резултат.

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 500 | John | Jones | 000000000 | 75000 | 5000 | M | Y | 01-FEB-90 | (null) | (null) | 400 |
| 501 | Mary | Jones | 111111111 | 30000 | (null) | F | N | 02-MAR-02 | (null) | 506 | 401 |
| 502 | Antonios | Vamvakeros | 222222222 | 30000 | (null) | M | N | 15-MAR-02 | (null) | 506 | 401 |
| 503 | Louisa | Lindboe | 333333333 | 30000 | 1000 | F | Y | 01-JAN-02 | (null) | 506 | 401 |
| 504 | Cara | Montopoli | 444444444 | 35000 | 2000 | F | Y | 01-JAN-01 | (null) | 506 | 401 |
| 505 | Michael | Dupre | 555555555 | 50000 | 4000 | M | Y | 12-DEC-97 | (null) | 500 | 402 |
| 506 | Migumi | Zona | 666666666 | 75000 | 5000 | F | N | 01-MAR-90 | (null) | 500 | 400 |
| 507 | Wey | Ho | 777777777 | 30000 | (null) | M | N | 01-FEB-90 | (null) | 505 | 402 |

```
SELECT * FROM members
WHERE salary NOT IN(30000, 35000, 50000, 75000);
```

Резултат от този пример

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 508 | Joseph | Smith | 888888888 | 0 | 0 | M | N | 01-FEB-90 | 15-JUN-02 | (null) | (null) |

Оператор BETWEEN се използва за филтриране на интервал от стойности с включени краища на интервала.

```
SELECT * FROM members
WHERE salary BETWEEN 35000 AND 60000;

SELECT * FROM members
WHERE salary >= 35000 AND salary <= 60000;
```

Резултатът от тези примери е еднакъв

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 504 | Cara | Montopoli | 444444444 | 35000 | 2000 | F | Y | 01-JAN-01 | (null) | 506 | 401 |
| 505 | Michael | Dupre | 555555555 | 50000 | 4000 | M | Y | 12-DEC-97 | (null) | 500 | 402 |

Оператор LIKE се използва за филтриране на текст по част от него.

Филтриране по първа буква “M”.

```
SELECT * FROM members
WHERE firstname LIKE  'M%';
```

Резултат от този пример

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 501 | Mary | Jones | 111111111 | 30000 | (null) | F | N | 02-MAR-02 | (null) | 506 | 401 |
| 505 | Michael | Dupre | 555555555 | 50000 | 4000 | M | Y | 12-DEC-97 | (null) | 500 | 402 |
| 506 | Migumi | Zona | 666666666 | 75000 | 5000 | F | N | 01-MAR-90 | (null) | 500 | 400 |

Филтриране по последна буква “y”.

```
SELECT * FROM members
WHERE firstname LIKE '%y';
```

Филтриране по съдържаща се буква “a”.

```
SELECT * FROM members
WHERE firstname LIKE '%a%';
```

Филтриране по втора буква “a”. За да се укаже, че преди a има 1 символ се използва символа “ _”

```
SELECT * FROM members
WHERE firstname LIKE '_a%';
```

Резултат от този пример

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 501 | Mary | Jones | lllllllll | 30000 | (null) | F | N | 02-MAR-02 | (null) | 506 | 401 |
| 504 | Cara | Montopoli | 444444444 | 35000 | 2000 | F | Y | 01-JAN-01 | (null) | 506 | 401 |

Допълнителна информация на: http://www.w3schools.com/sql/sql_where.asp

**Сортиране на данни. ORDER BY клауза.**

ORDER BY клаузата се изпълнява последна в рамките на стандартна SELECT конструкция, независимо колко други клаузи съдържа конструкцията.

Пример:

*SELECT <column_name list>*

*FROM <table_name list>*

*WHERE <predicate>*

*ORDER BY <column_name list>;*

В ORDER BY клаузата се описват като списък колоните, по които трябва да се сортира резултатната таблица. Колоните могат да бъдат описани с имената на полетата от таблицата, с псевдоним зададен в SELECT клаузата за съответно поле или по пореден номер от списъка в SELECT клаузата.

За да се укаже начина на сортиране дали да бъде по възходящ или низходящ ред, след всяка колона в списъка на ORDER BY клаузата се използват ключовите думи ASC(възходящ ред ) или DESC (низходящ ред ). Ако не е изрично указана някоя от двете сортировки се подразбира ASC(възходящ ред ).

Примери с таблици от демонстрационната база. В примерите се ползва схема Student2.

```
SELECT firstname, lastname, salary FROM members
ORDER BY firstname, lastname;
```

Сортиране по позиция на колона в SELECT клаузата.

```
SELECT firstname, lastname, salary FROM members
ORDER BY 1, 2;
```

Резултат от горните два примера

| FIRSTNAME | LASTNAME | SALARY |
|---|---|---|
| Antonios | Vamvakeros | 30000 |
| Cara | Montopoli | 35000 |
| John | Jones | 75000 |
| Joseph | Smith | 0 |
| Louisa | Lindboe | 30000 |
| Mary | Jones | 30000 |
| Michael | Dupre | 50000 |
| Migumi | Zona | 75000 |
| Wey | Ho | 30000 |

Сортиране чрез използване на псевдоним на колона.

```
SELECT firstname, lastname, salary*1.1 AS NewSalary
FROM members
ORDER BY NewSalary;
```

Резултат от този пример

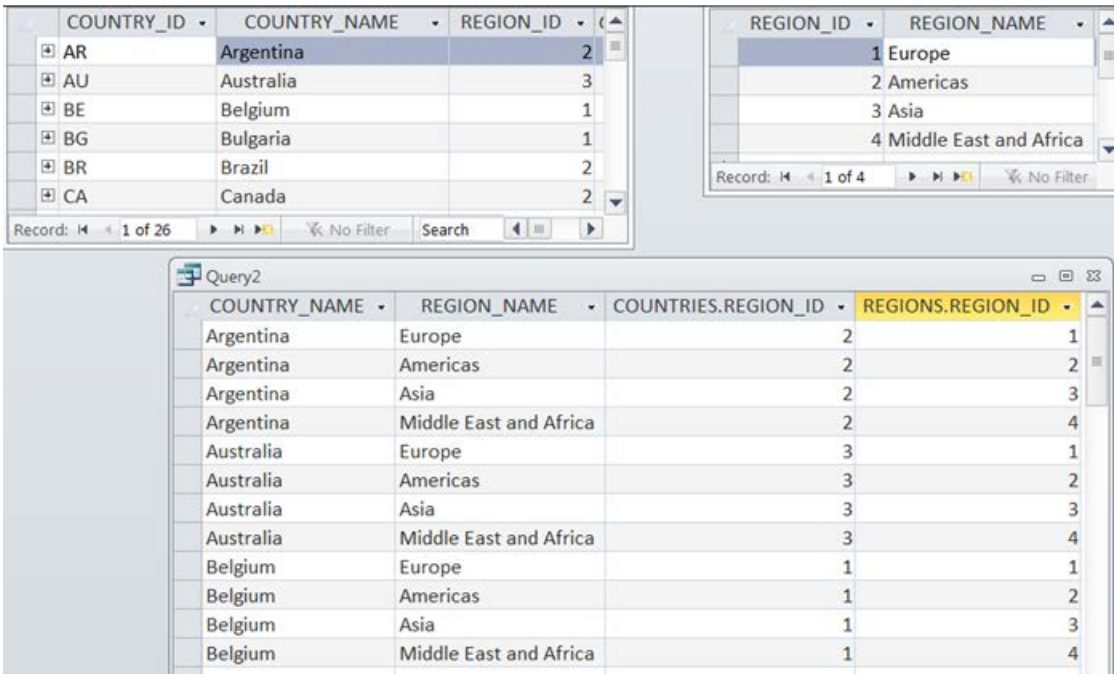
| FIRSTNAME | LASTNAME | NEWSALARY |
|---|---|---|
| Joseph | Smith | 0 |
| Wey | Ho | 33000 |
| Mary | Jones | 33000 |
| Antonios | Vamvakeros | 33000 |
| Louisa | Lindboe | 33000 |
| Cara | Montopoli | 38500 |
| Michael | Dupre | 55000 |
| Migumi | Zona | 82500 |
| John | Jones | 82500 |

# Модул 4: Заявки към множество таблици

**Общи положения.**

- Таблиците, които участват в една SELECT конструкция се изброяват в клаузата FROM разделени със запетая.
- Ако не се използват други клаузи дефиниращи връзки между таблиците, то резултата е обща виртуална таблица съпоставяща всички редове на всяка таблица с всички редове от другите таблици.
- Пример: *SELECT * FROM Regions, Countries;*

Таблицата Query2 демонстрира фрагмент от виртуалната таблица получена от горния пример.



| COUNTRY_ID | COUNTRY_NAME | REGION_ID |
|---|---|---|
| AR | Argentina | 2 |
| AU | Australia | 3 |
| BE | Belgium | 1 |
| BG | Bulgaria | 1 |
| BR | Brazil | 2 |
| CA | Canada | 2 |

Record: 1 of 26 No Filter Search

| REGION_ID | REGION_NAME |
|---|---|
| 1 | Europe |
| 2 | Americas |
| 3 | Asia |
| 4 | Middle East and Africa |

Record: 1 of 4 No Filter

Query2

| COUNTRY_NAME | REGION_NAME | COUNTRIES.REGION_ID | REGIONS.REGION_ID |
|---|---|---|---|
| Argentina | Europe | 2 | 1 |
| Argentina | Americas | 2 | 2 |
| Argentina | Asia | 2 | 3 |
| Argentina | Middle East and Africa | 2 | 4 |
| Australia | Europe | 3 | 1 |
| Australia | Americas | 3 | 2 |
| Australia | Asia | 3 | 3 |
| Australia | Middle East and Africa | 3 | 4 |
| Belgium | Europe | 1 | 1 |
| Belgium | Americas | 1 | 2 |
| Belgium | Asia | 1 | 3 |
| Belgium | Middle East and Africa | 1 | 4 |

**Изпълнение на FROM клауза.**

- FROM клаузата определя изходни таблици, които да се използват в SELECT клаузата.
- FROM клаузата може да съдържа таблици и оператори.
- Резултатът от клаузата FROM е виртуална таблица.
- FROM клаузата може да определя псевдоними на таблиците използвани в другите клаузи.

**Съединение (join)** – дефинира отношения между таблици, с чиято помощ се извлича информация от тях.

**SQL стандарти за свързване на таблици**

- **При ANSI SQL-92**

  *SELECT <списък колони>*

  *FROM <име на таблица1> JOIN <име на таблица2>ON <критерии за сравнение>*

- **При ANSI SQL-89**

  SELECT ...

  FROM *<име на таблица1>, <име на таблица2>*

  WHERE *<критерии за сравнение>*

Допълнителна ин формация може да се намери на: http://bg.wikipedia.org/wiki/SQL

и на : http://www.w3schools.com/sql/sql_join.asp

Препоръка при създаването на заявки между няколко таблици е да се използват псевдоними за таблиците като съкратени имена, с което са възможни по-кратки записи на заявките.

Когато две или повече таблици имат едни и същи имена на полета, то задължително при използване на тези полета те трябва да се описват с пълните си имена:

*име на таблица.име на поле*

**Използване на вътрешно съединение INNER JOIN.**

При вътрешното съединение INNER JOIN се прави съпоставяне на редовете от една таблица с редовете на друга таблица посредством връзка между двете таблици осъществявана с помощта на ключови полета от двете таблици. От едната таблица участва външен ключ (Foreign Key), а от другата първичен ключ(Primary Key).

При вътрешно съединение на практика се изключват редовете без съответствия.

Вътрешно съединение може да се реализира и по двата начина според стандартите ANSI SQL-89 или ANSI SQL-92.

**Вътрешно съединение примери.** В примерите се ползва схема Student2.

ANSI SQL-89

```
SELECT firstname,lastname, name
FROM members, teams
WHERE teams.teamid = members.teamid;

SELECT firstname,lastname, name, t.teamid
FROM members m, teams t
WHERE m.teamid=t.teamid;
```

Резултат от горните два примера

| FIRSTNAME | LASTNAME | NAME |
|---|---|---|
| John | Jones | Management |
| Mary | Jones | Production |
| Antonios | Vamvakeros | Production |
| Louisa | Lindboe | Production |
| Cara | Montopoli | Production |
| Michael | Dupre | Engineering |
| Migumi | Zona | Management |
| Wey | Ho | Engineering |

ANSI SQL-92

```
SELECT firstname,lastname, name, t.teamid
FROM teams t INNER JOIN members m ON t.teamid = m.teamid;

SELECT c.name Customer, p.name Product, qty, saledate,
unitprice
FROM Sales s JOIN customers c ON s.customerid=c.customerid
JOIN products p ON s.productid = p.productid;
```

Фрагмент от резултата на този пример

| CUSTOMER | PRODUCT | QTY | SALEDATE | UNITPRICE |
|---|---|---|---|---|
| The Camera Store | Digital camera | 5 | 01-NOV-01 | 350 |
| Hi Tech Supply | Digital camera | 5 | 12-JAN-01 | 600 |
| The Camera Store | Digital camera | 10 | 01-DEC-01 | 300 |
| Computer Planet | Scanner | 5 | 15-JUN-01 | 900 |
| Computer Planet | Scanner | 5 | 07-APR-01 | 900 |
| Computer Planet | Scanner | 1 | 08-APR-01 | 1000 |
| Hi Tech Supply | Scanner | 5 | 12-JAN-01 | 1000 |
| Computer Planet | Hi Res Scanner | 5 | 15-JUN-01 | 1200 |

Допустимо е при конструкции по стандарта ANSI SQL-92 да се пропуска ключовата дума INNER.

Допълнителна ин формация може да се намери на: http://www.w3schools.com/sql/sql_join_inner.asp

**Използване на външни съединения OUTER JOIN.**

**OUTER JOIN** – включва редовете без съответствие от едната или другата таблица в съединението.

**Видове:**

- ✓ LEFT OUTER JOIN
- ✓ RIGHT OUTER JOIN
- ✓ FULL OUTER JOIN

В примерите по-долу се ползва схема Student2.

**LEFT OUTER JOIN –** резултатната таблица ще съдържа всички редове от стоящата в ляво на думата JOIN таблица (в примера това е таблицата teams). Полетата от дясната таблица members ще съдържат данни само при наличие на съответствие.

```
SELECT firstname,lastname, name, t.teamid
FROM teams t LEFT OUTER JOIN members m ON t.teamid = m.teamid;
```

Резултат от този пример

| FIRSTNAME | LASTNAME | NAME | TEAMID |
|---|---|---|---|
| Migumi | Zona | Management | 400 |
| John | Jones | Management | 400 |
| Cara | Montopoli | Production | 401 |
| Louisa | Lindboe | Production | 401 |
| Antonios | Vamvakeros | Production | 401 |
| Mary | Jones | Production | 401 |
| Wey | Ho | Engineering | 402 |
| Michael | Dupre | Engineering | 402 |
| Joseph | Smith | (null) | (null) |

**RIGHT OUTER JOIN–** резултатната таблица ще съдържа всички редове от стоящата в дясно на думата JOIN таблица (в примера това е таблицата members). Полетата от дясната таблица teams ще съдържат данни само при наличие на съответствие.

```
SELECT firstname,lastname, name, t.teamid
FROM teams t RIGHT OUTER JOIN members m ON t.teamid =
m.teamid;
```

Резултат от този пример

| FIRSTNAME | LASTNAME | NAME | TEAMID |
|---|---|---|---|
| John | Jones | Management | 400 |
| Mary | Jones | Production | 401 |
| Antonios | Vamvakeros | Production | 401 |
| Louisa | Lindboe | Production | 401 |
| Cara | Montopoli | Production | 401 |
| Michael | Dupre | Engineering | 402 |
| Migumi | Zona | Management | 400 |
| Wey | Ho | Engineering | 402 |
| (null) | (null) | Advertising | 403 |

**FULL OUTER JOIN–** обединението от резултатите при ляво и дясно съединение.

```
SELECT firstname,lastname, name, t.teamid
FROM teams t FULL OUTER JOIN members m ON t.teamid =
m.teamid;
```

Резултат от този пример

| FIRSTNAME | LASTNAME | NAME | TEAMID |
|---|---|---|---|
| John | Jones | Management | 400 |
| Mary | Jones | Production | 401 |
| Antonios | Vamvakeros | Production | 401 |
| Louisa | Lindboe | Production | 401 |
| Cara | Montopoli | Production | 401 |
| Michael | Dupre | Engineering | 402 |
| Migumi | Zona | Management | 400 |
| Wey | Ho | Engineering | 402 |
| Joseph | Smith | (null) | (null) |
| (null) | (null) | Advertising | 403 |

При външно съединение е допустимо да се пропусне ключовата дума OUTER.

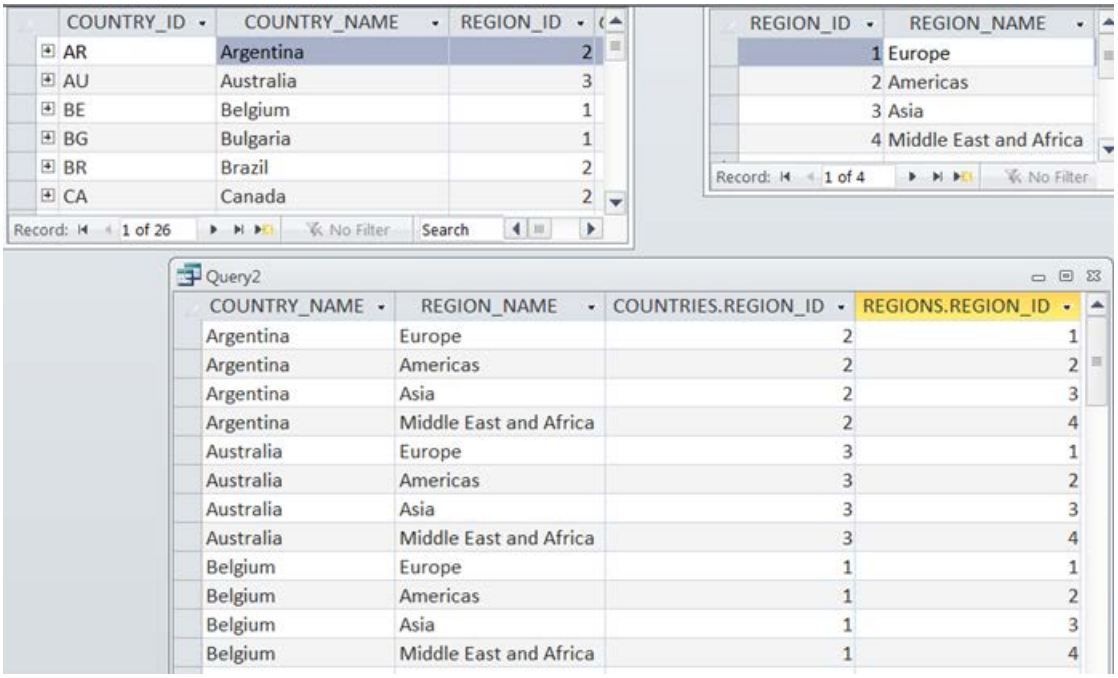
Допълнителна ин формация може да се намери на:
http://www.w3schools.com/sql/sql_join_left.asp
http://www.w3schools.com/sql/sql_join_right.asp
http://www.w3schools.com/sql/sql_join_full.asp

**Използване на съединения от тип CROSS JOIN.**

- Ако не се използват други клаузи дефиниращи връзки между таблиците, то резултата е обща таблица съпоставяща всички редове на всяка таблица с всички редове от другите таблици.
- Същият резултат може да се получи и при използване на CROSS JOIN съединение, с което се подчертава по-горе описания резултат.
- Пример от схема HR.:

  *SELECT * FROM Regions, Countries;*

  *SELECT * FROM Regions CROSS JOIN Countries;*



| COUNTRY_ID | COUNTRY_NAME | REGION_ID |
|---|---|---|
| AR | Argentina | 2 |
| AU | Australia | 3 |
| BE | Belgium | 1 |
| BG | Bulgaria | 1 |
| BR | Brazil | 2 |
| CA | Canada | 2 |

Record: 1 of 26 No Filter Search

| REGION_ID | REGION_NAME |
|---|---|
| 1 | Europe |
| 2 | Americas |
| 3 | Asia |
| 4 | Middle East and Africa |

Record: 1 of 4 No Filter

Query2

| COUNTRY_NAME | REGION_NAME | COUNTRIES.REGION_ID | REGIONS.REGION_ID |
|---|---|---|---|
| Argentina | Europe | 2 | 1 |
| Argentina | Americas | 2 | 2 |
| Argentina | Asia | 2 | 3 |
| Argentina | Middle East and Africa | 2 | 4 |
| Australia | Europe | 3 | 1 |
| Australia | Americas | 3 | 2 |
| Australia | Asia | 3 | 3 |
| Australia | Middle East and Africa | 3 | 4 |
| Belgium | Europe | 1 | 1 |
| Belgium | Americas | 1 | 2 |
| Belgium | Asia | 1 | 3 |
| Belgium | Middle East and Africa | 1 | 4 |

Допълнително описание е дадено в началото на модула в подточка общо описание.

**Използване на съединения от тип SELF JOIN.**

SELF JOIN е термин, който се използва за да бъде представи едно решение при дизайн на база данни. Това решение премахва необходимостта да се ползва втора таблица ако в дадена таблица се създаде поле играещо ролята на външен ключ, който прави референция към първичния ключ на същата таблица. В този случай ще има обръщение на таблицата към самата нея.

Ако в таблицата employees са описани служители на компанията и някои от тях имат за началник на отдел, който е друг служител описан в същата таблица, то е възможно в полето manager_id да се използва стойност от полето employee_id.

EMPLOYEES

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | MANAGER_ID |
|---|---|---|---|
| 100 | Steven | King | |
| 101 | Neena | Kochhar | 100 |
| 102 | Lex | De Haan | 100 |
| 103 | Alexander | Hunold | 102 |
| 104 | Bruce | Ernst | 103 |
| 105 | David | Austin | 103 |
| 106 | Valli | Pataballa | 103 |
| 107 | Diana | Lorentz | 103 |
| 108 | Nancy | Greenberg | 101 |
| 109 | Daniel | Faviet | 108 |
| 110 | John | Chen | 108 |
| 111 | Ismael | Sciarra | 108 |
| 112 | Jose Manuel | Urman | 108 |
| 113 | Luis | Popp | 108 |
| 114 | Den | Raphaely | 100 |

Така служители с employee_id 101 и 102, ще има за началник на отдел служител с номер 100.

Служителите с employee_id 101 и 102 от своя страна ще са началник отдели на служители описани в същата таблица.

В този случай може да се каже, ч е таблицата прави обръщение към самата себе си.

Ако трябва да се направи справка показваща името на служителя и името на негови началник отдел, то използвайки по-горе-описания модел може да се състави следната заявка:

```
SELECT first_name || ' ' || last_name AS Employee,
       first_name || ' ' || last_name AS Manager
FROM employees emp JOIN employees mngr
ON emp.manger_id = mngr.employee_id;
```

# Модул 5: Множествени оператори

**Общо описание на множествените оператори.**

Оператори за работа с множества. Известни също като SET(в смисъл на множество) оператори.

**Предназначение** – за работа с различни множества от данни. Множество данни може да бъде цяла таблица или извадка от дадена таблица.

**Използват се за:**

- Обединяване на данни.
- Намиране на разлики между множества.
- Намиране на съвпадения между множества.

**Видове множествени оператори**

- **UNION**
- **UNION ALL**
- **INTERSECT**
- **MINUS**

**Синтаксис.**

*SELECT конструкция 1*

*Множествен оператор*

*SELECT конструкция 2*

*Множествен оператор*

*SELECT конструкция 3*

*Множествен оператор*

*SELECT конструкция 4*

**Оператор UNION**

**Действие на UNION -** Обединява две множества като редуцира дублиращите се записи до един. Премахва повторенията, които могат да се намерят в обединяваните множества едновременно.

**Пример**: ако обединим 2 таблици с оператора UNION то в резултатната таблица ще получим всички редове от първата таблица + всички редове от втората и от резултата след това обединение ще бъдат премахнати всички дублиращи се редове. От тези дублирани редове в резултата ще остане само по един представител.

Ако имаме следните 2 таблици:

| products_officeA | |
|---|---|
| **product_id** | **product_name** |
| 1 | product1 |
| 2 | product2 |
| 2 | product2 |
| 3 | product3 |
| 4 | product4 |
| 5 | product5 |

| products_officeB | |
|---|---|
| **product_id** | **product_name** |
| 1 | product1 |
| 2 | product2 |
| 3 | product3 |
| 4 | product4 |
| 6 | product6 |
| 7 | product7 |

Резултатът от следната конструкция би бил следния:

```
SELECT product_id, product_name
FROM products_officeA
UNION
SELECT product_id, product_name
FROM products_officeB;
```

| **product_id** | **product_name** |
|---|---|
| 1 | product1 |
| 2 | product2 |
| 3 | product3 |
| 4 | product4 |
| 5 | product5 |
| 6 | product6 |
| 7 | product7 |

**Оператор UNION ALL**

**Действие на UNION ALL -** Обединява две множества като показва всички записи от множествата без значение дали има повторения на записи в обединяваните множества.

**Пример**: ако обединим 2 таблици с оператора UNION ALL то в резултатната таблица ще получим всички редове от първата таблица + всички всички редове от втората таблица.

Ако имаме следните 2 таблици:

| products_officeA | |
|---|---|
| **product_id** | **product_name** |
| 1 | product1 |
| 2 | product2 |
| 2 | product2 |
| 3 | product3 |
| 4 | product4 |
| 5 | product5 |

| products_officeB | |
|---|---|
| **product_id** | **product_name** |
| 1 | product1 |
| 2 | product2 |
| 3 | product3 |
| 4 | product4 |
| 6 | product6 |
| 7 | product7 |

Резултатът от следната конструкция би бил следния:

```
SELECT product_id, product_name
FROM products_officeA
UNION ALL
SELECT product_id, product_name
FROM products_officeB;
```

| product_id | product_name |
|---|---|
| 1 | product1 |
| 2 | product2 |
| 2 | product2 |
| 3 | product3 |
| 4 | product4 |
| 5 | product5 |
| 1 | product1 |
| 2 | product2 |
| 3 | product3 |
| 4 | product4 |
| 6 | product6 |
| 7 | product7 |

**Оператор INTERSECT**

**Действие на INTERSECT** - резултатът от оператора INTERSECT е множеството на записите, които могат да се намерят във всяко от изследваните множества. Т. е. това е множеството на общите записи.

**Пример**: ако за 2 таблици използваме оператора INTERSECT то в резултатната таблица ще получим редовете които ги има както в едната така и в другата. От всеки ред в резултатната таблица ще има само по един. Т.е. повторенията не се показват.

Ако имаме следните 2 таблици:

| products_officeA | |
|---|---|
| **product_id** | **product_name** |
| 1 | product1 |
| 2 | product2 |
| 2 | product2 |
| 3 | product3 |
| 4 | product4 |
| 5 | product5 |

| products_officeB | |
|---|---|
| **product_id** | **product_name** |
| 1 | product1 |
| 2 | product2 |
| 3 | product3 |
| 4 | product4 |
| 6 | product6 |
| 7 | product7 |

Резултатът от следната конструкция би бил следния:

```
SELECT product_id, product_name
FROM products_officeA
INTERSECT
SELECT product_id, product_name
FROM products_officeB;
```

| **product_id** | **product_name** |
|---|---|
| 1 | product1 |
| 2 | product2 |
| 3 | product3 |
| 4 | product4 |

**Оператор MINUS**

**Действие на MINUS -** Резултатът от оператора MINUS е множеството на записите, които могат да се намерят във първото множество, но ги няма в следващото.

В Transact SQL реализацията на Microsoft операторът MINUS е с наименование EXCEPT.

**Пример**: ако за 2 таблици използваме оператора **MINUS** то в резултатната таблица ще получим редовете от първата таблица, които ги няма във втората таблица.

Ако се налага да се получат редовете от втората таблица, които ги няма в първата, то трябва да се състави друга конструкция, при която двете таблици са с разменени места спрямо оператора MINUS.

Ако имаме следните 2 таблици:

| products_officeA | |
|---|---|
| **product_id** | **product_name** |
| 1 | product1 |
| 2 | product2 |
| 2 | product2 |
| 3 | product3 |
| 4 | product4 |
| 5 | product5 |

| products_officeB | |
|---|---|
| **product_id** | **product_name** |
| 1 | product1 |
| 2 | product2 |
| 3 | product3 |
| 4 | product4 |
| 6 | product6 |
| 7 | product7 |

Резултатът от следната конструкция би бил следния:

```
SELECT product_id, product_name
FROM products_officeA
MINUS
SELECT product_id, product_name
FROM products_officeB;
```

| **product_id** | **product_name** |
|---|---|
| 5 | product5 |

**Използване на множествени оператори.**

- Във всяко множество броя на колоните трябва да е еднакъв.
- Подредбата на колоните в множествата трябва да съответства по тип и смисъл.
- Псевдоними за колони на резултатното множество се задават в първото използвано множество.
- ORDER BY клауза може да има само в последното множество.

**Пример:**

*SELECT product_id AS "Номер", product_name AS "Продукт"*

*FROM products_officeA*

*UNION*

*SELECT product_id, product_name*

*FROM products_officeB*

*ORDER BY product_name;*

| Номер | Продукт |
|---|---|
| 1 | product1 |
| 2 | product2 |
| 3 | product3 |
| 4 | product4 |
| 5 | product5 |
| 6 | product6 |
| 7 | product7 |

Примери с таблици от демонстрационната база ползващи множествени оператори. В примерите се използват таблици от схема Student2.

**UNION и UNION с една колона**

```
Select teamid FROM members
UNION
SELECT teamid FROM teams;
```

Резултат от този пример

| TEAMID |
|---|
| 400 |
| 401 |
| 402 |
| 403 |
| (null) |

```
Select teamid FROM members
UNION ALL
SELECT teamid FROM teams;
```

Резултат от този пример

| TEAMID |
|---|
| 400 |
| 401 |
| 401 |
| 401 |
| 401 |
| 402 |
| 400 |
| 402 |
| (null) |
| 400 |
| 401 |
| 402 |
| 403 |

**UNION и UNION с повече от една колона и ORDER BY клауза**

```
Select teamid, mentorid FROM members
UNION ALL
SELECT teamid, leaderid FROM teams;
```

Резултат от този пример

| TEAMID | MENTORID |
|---|---|
| 400 | (null) |
| 401 | 506 |
| 401 | 506 |
| 401 | 506 |
| 401 | 506 |
| 402 | 500 |
| 400 | 500 |
| 402 | 505 |
| (null) | (null) |
| 400 | 500 |
| 401 | 506 |
| 402 | 505 |
| 403 | 500 |

```
Select teamid, mentorid FROM members
UNION
SELECT teamid, leaderid FROM teams;
```

Резултат от този пример

| TEAMID | MENTORID |
|---|---|
| 400 | 500 |
| 400 | (null) |
| 401 | 506 |
| 402 | 500 |
| 402 | 505 |
| 403 | 500 |
| (null) | (null) |

```
Select teamid, mentorid, bonus FROM members
UNION ALL
SELECT teamid, leaderid, 0 FROM teams;
```

Резултат от този пример

| TEAMID | MENTORID | BONUS |
|---|---|---|
| 400 | (null) | 5000 |
| 401 | 506 | (null) |
| 401 | 506 | (null) |
| 401 | 506 | 1000 |
| 401 | 506 | 2000 |
| 402 | 500 | 4000 |
| 400 | 500 | 5000 |
| 402 | 505 | (null) |
| (null) | (null) | 0 |
| 400 | 500 | 0 |
| 401 | 506 | 0 |
| 402 | 505 | 0 |
| 403 | 500 | 0 |

**INTERSECT**

```
Select teamid, mentorid FROM members
INTERSECT
SELECT teamid, leaderid FROM teams;
```

Резултат от този пример

| TEAMID | MENTORID |
|---|---|
| 400 | 500 |
| 401 | 506 |
| 402 | 505 |

**MINUS**

```
Select teamid, mentorid FROM members
MINUS
SELECT teamid, leaderid FROM teams;
```

Резултат от този пример

| TEAMID | MENTORID |
|---|---|
| 400 | (null) |
| 402 | 500 |
| (null) | (null) |

```
SELECT teamid, leaderid FROM teams
MINUS
Select teamid, mentorid FROM members;
```

Резултат от този пример

| TEAMID | LEADERID |
|---|---|
| 403 | 500 |

# Модул 6: Групиране на данни

**Обобщаващи функции.**

Обща информация и правила за използване.

- Генерират обобщена информация за група от стойности. Наричат се още агрегатни функции.
- Групата стойност може да е общото множество от данни формирано от цяла таблица или подмножества в таблицата.
- Резултатът от всяка обобщаваща функция е единична стойност.
- Не е допустимо влагането на агрегатни функции една в друга.

**Видове обобщаващи функции**

- Count - генерира броя на редовете или стойностите в дадена колона различна от NULL

  *SELECT COUNT(jobs_id) FROM Jobs;*

- SUM – генерира аритметичната сума на всички избрани стойности от дадена колона

  *SELECT SUM(salary) FROM Employees;*

- AVG – генерира средното аритметично на всички избрани стойности от дадена колона

  *SELECT AVG(salary) FROM Employees;*

- MAX – генерира най-голямата от всички избрани стойности на дадена колона

  *SELECT MAX(salary) FROM Employees;*

- MIN – генерира най-малката от всички избрани стойности на дадена колона

  *SELECT MIN(salary) FROM Employees;*

Допълнителна информация на: http://www.w3schools.com/sql/sql_functions.asp

**Обобщаващи функции с DISTINCT -** Използваната функция прави своето обобщение само върху уникалните стойности в дадено поле.

*SELECT COUNT(DISTINCT salary) FROM Employees;*

**Групиране на данни с клауза GROUP BY.**

- Позволява да се дефинира подмножество от стойностите в конкретна колона и да се приложат агрегатни функции.
- Полетата, по които се групира се изброяват в **GROUP BY** клаузата разделени със запетая.
- Поле в клаузата SELECT , което не е описано в GROUP BY клаузата, не може да се използва без обобщаваща функция.
- Пример: ако имаме списък на страни и за всяка страна е указан регион, в който се намира то може да се направи обобщение по региони. Колко страни има в даден регион?

*SELECT r.region_ID, region_name, COUNT(country_ID)*

*FROM Countries c JOIN Regions r ON c.region_id=r.region_id*

*GROUP BY r.region_ID, region_name;*

| REGION_ID | REGION_NAME | CountOfCOUNTRY_ID |
|---|---|---|
| 1 | Europe | 9 |
| 2 | Americas | 5 |
| 3 | Asia | 6 |
| 4 | Middle East and Africa | 6 |

Допълнителна информация на: http://www.w3schools.com/sql/sql_groupby.asp

Примери с таблици от демонстрационната база. В примерите се използва схемата Student2.

```
SELECT teamid, COUNT(mentorid), COUNT(*)FROM members
GROUP BY teamid;
```

Резултат от този пример

| TEAMID | COUNT(MEMBERID) | COUNT(*) |
|---|---|---|
| 400 | 2 | 2 |
| (null) | 1 | 1 |
| 402 | 2 | 2 |
| 401 | 4 | 4 |

```
SELECT teamid, SUM(salary)FROM members
GROUP BY teamid;
```

Резултат от този пример

| TEAMID | SUM(SALARY) |
|---|---|
| 400 | 150000 |
| (null) | 0 |
| 402 | 80000 |
| 401 | 125000 |

```
SELECT teamid, mentorid, COUNT(*)
FROM members
GROUP BY teamid, mentorid;
```

Резултат от този пример

| TEAMID | MEMBERID | COUNT(*) |
|---|---|---|
| 401 | 503 | 1 |
| 402 | 507 | 1 |
| 401 | 501 | 1 |
| 401 | 502 | 1 |
| 401 | 504 | 1 |
| 402 | 505 | 1 |
| 400 | 506 | 1 |
| (null) | 508 | 1 |
| 400 | 500 | 1 |

**Филтриране на групирани данни чрез клаузата HAVING.**

- Дефинира критерии за елиминиране на определени групи от изходни данни.
- Филтрира подобно на WHERE, но не за редове, а за групи.
- Разлика между клаузата WHERE и клаузата HAVING:
  - ✓ WHERE филтрира преди групирането в GROUP BY и оформя множество от данни, което да бъде групирано.
  - ✓ HAVING филтрира вече групирания резултат.
- Пример:

*SELECT r.region_ID, region_name, COUNT(country_ID)*

*FROM Countries c JOIN Regions r ON c.region_id=r.region_id*

*GROUP BY r.region_ID, region_name;*

*HAVING COUNT(country_ID)> 5;*

Резултат от този пример

| REGION_ID | REGION_NAME | CountOfCOUNTRY_ID |
|---|---|---|
| 1 | Europe | 9 |
| 3 | Asia | 6 |
| 4 | Middle East and Africa | 6 |

Допълнителна информация на: http://www.w3schools.com/sql/sql_having.asp

Примери с таблици от демонстрационната база. В примерите се използва схемата Student2.

```
SELECT customerid, Count(*)
FROM sales
GROUP BY customerid
HAVING COUNT(*)<3;
```

Резултат от този пример

| CUSTOMERID | COUNT(*) |
|---|---|
| 101 | 1 |

```
SELECT customerid, SUM(qty*unitprice) Total
FROM sales
GROUP BY customerid
HAVING SUM(qty*unitprice)>20000;
```

Резултат от този пример

| CUSTOMERID | TOTAL |
|---|---|
| 104 | 30755 |
| 103 | 24900 |

При филтриране по групиращото поле е по-добре да се използва филтриране в WHERE клаузата т.к. след това филтриране редовете подлежащи на групиране ще са по-малко. Следните 2 примера показват и двата варианта за филтриране по групиращата колона, но за предпочитане е този с WHERE.

```
SELECT customerid, SUM(qty*unitprice) Total
FROM sales
GROUP BY customerid
HAVING customerid >= 102;

SELECT customerid, SUM(qty*unitprice) Total
FROM sales
WHERE customerid >= 102
GROUP BY customerid;
```

Резултат от този пример

| CUSTOMERID | TOTAL |
|---|---|
| 102 | 11500 |
| 104 | 30755 |
| 103 | 24900 |

В клаузата HAVING могат да се използват всички оператори за филтриране, които се ползват и в WHERE клаузата.

```
SELECT customerid, SUM(qty*unitprice) Total
FROM sales
GROUP BY customerid
HAVING SUM(qty*unitprice)>10000 and SUM(qty*unitprice)<20000;
```

**Използване на ROLLUP и CUBE.**

- Операторите ROLLUP и CUBE допълват клаузата GROUP BY с допълнителни редова за обща сума по вече групирани полета.
- Те дават обобщения по цялото множество, което е разделено на подмножество посредством клаузата GROUP BY .
- Пример:

*SELECT r.region_ID, region_name, COUNT(country_ID)*

*FROM Countries c JOIN Regions r ON c.region_id=r.region_id*

*GROUP BY ROLLUP(r.region_ID, region_name);*

Резултат от примера:

| REGION_ID | REGION_NAME | COUNT(COUNTRY_ID) |
|---|---|---|
| 1 | Europe | 8 |
| 1 | (null) | 8 |
| 2 | Americas | 5 |
| 2 | (null) | 5 |
| 3 | Asia | 6 |
| 3 | (null) | 6 |
| 4 | Middle East and Africa | 6 |
| 4 | (null) | 6 |
| (null) | (null) | 25 |

Допълнителна информация на: http://oracle-base.com/articles/misc/rollup-cube-grouping-functions-and-grouping-sets.php

Примери за обобщения на данни с таблици от демонстрационната база. В примерите се използва схемата Student2.

```
SELECT customerid, SUM(qty*unitprice) Total FROM sales
GROUP BY customerid
UNION ALL
SELECT NULL, SUM(qty*unitprice)FROM sales;
```

Резултат от примера:

| CUSTOMERID | TOTAL |
|---|---|
| 100 | 14300 |
| 102 | 11500 |
| 101 | 699.98 |
| 104 | 30755 |
| 103 | 24900 |
| (null) | 82154.98 |

```
SELECT customerid, productid, SUM(qty*unitprice) Total FROM sales
GROUP BY ROLLUP(customerid, productid);
```

Резултат от примера:

| CUSTOMERID | PRODUCTID | TOTAL |
|---|---|---|
| 100 | 300 | 4750 |
| 100 | 304 | 9550 |
| 100 | (null) | 14300 |
| 101 | 305 | 699.98 |
| 101 | (null) | 699.98 |
| 102 | 303 | 11500 |
| 102 | (null) | 11500 |
| 103 | 301 | 10000 |
| 103 | 302 | 12000 |
| 103 | 303 | 2900 |
| 103 | (null) | 24900 |
| 104 | 300 | 3000 |
| 104 | 301 | 5000 |
| 104 | 302 | 5000 |
| 104 | 303 | 10000 |
| 104 | 304 | 7755 |
| 104 | (null) | 30755 |
| (null) | (null) | 82154.98 |

```
SELECT customerid, productid, SUM(qty*unitprice) Total FROM sales
GROUP BY ROLLUP(productid, customerid);
```

Резултат от примера:

| CUSTOMERID | PRODUCTID | TOTAL |
|---|---|---|
| 100 | 300 | 4750 |
| 104 | 300 | 3000 |
| (null) | 300 | 7750 |
| 103 | 301 | 10000 |
| 104 | 301 | 5000 |
| (null) | 301 | 15000 |
| 103 | 302 | 12000 |
| 104 | 302 | 5000 |
| (null) | 302 | 17000 |
| 102 | 303 | 11500 |
| 103 | 303 | 2900 |
| 104 | 303 | 10000 |
| (null) | 303 | 24400 |
| 100 | 304 | 9550 |
| 104 | 304 | 7755 |
| (null) | 304 | 17305 |
| 101 | 305 | 699.98 |
| (null) | 305 | 699.98 |
| (null) | (null) | 82154.98 |

```
SELECT customerid, productid, SUM(qty*unitprice) Total FROM sales
GROUP BY CUBE(customerid, productid);
```

Резултат от примера:

| CUSTOMERID | PRODUCTID | TOTAL |
|---|---|---|
| (null) | (null) | 82154.98 |
| (null) | 300 | 7750 |
| (null) | 301 | 15000 |
| (null) | 302 | 17000 |
| (null) | 303 | 24400 |
| (null) | 304 | 17305 |
| (null) | 305 | 699.98 |
| 100 | (null) | 14300 |
| 100 | 300 | 4750 |
| 100 | 304 | 9550 |
| 101 | (null) | 699.98 |
| 101 | 305 | 699.98 |
| 102 | (null) | 11500 |
| 102 | 303 | 11500 |
| 103 | (null) | 24900 |
| 103 | 301 | 10000 |
| 103 | 302 | 12000 |
| 103 | 303 | 2900 |
| 104 | (null) | 30755 |
| 104 | 300 | 3000 |
| 104 | 301 | 5000 |
| 104 | 302 | 5000 |
| 104 | 303 | 10000 |
| 104 | 304 | 7755 |

# Модул 7: Използване на DDL за дефиниране на обекти в базите данни

**Типове данни.**

Типа на данните е основна характеристика за всяко поле на таблица в една система за управление на бази данни.

Съществуват така наречените основни типове данни, които от своя страна могат да се разделят на подтипове. Такива са:

- Текстови данни.
  - char(size), nchar(size), varchar2(size), nvarchar2(size)
- Числови данни.
  - number(p, s)
- Дати.
  - date, timestamp
- Други типове данни.
  - clob, nclob, blob, bfile, xmltype, boolean

Допълнителна информация
на: http://www.w3schools.com/sql/sql_datatypes_general.asp

**Конструкция CREATE**

- Служи за създаване на различни типове обекти
- Създаване на база.

  *CREATE database dbemp*
- Създаване на таблица

  *CREATE table temp*
- Създаване на изглед

  *CREATE view vemp*
- Създаване на индекс

  *CREATE index ixemp*

**Създаване на обект таблица.**

**Конструкция CREATE TABLE**

- Служи за създаване на таблица.

  *CREATE TABLE tEmp (empid number(6,0), name varchar2(75))*

- При използването на CREATE TABLE, задължително трябва да се дефинира име на таблицата и поне едно поле.
- При дефиницията на поле от таблица задължително се указват име на полето и тип на полето.
- Командата CREATE TABLE може да се използва по два начина: чрез изрична дефиниция на структурата на таблицата или чрез копиране на дизайна от друга таблица.
- Пример за изрична дефиниция на структурата на таблицата:

  *CREATE TABLE tEmp (empid number(6,0),*
  *fname varchar2(75),*
  *lname varchar2(75),*
  *salary number(6,2),*
  *hiredate DATE,*
  *departmentid number(6,0),*
  *managerid number(6,0)*
  *);*

- Създаване на таблица от друга таблица чрез използване на SELECT конструкция. Новата таблица приема дефиницията на полетата и данните от таблицата към която е отправена заявката в SELECT конструкцията.
- Пример:

  *CREATE TABLE tEmp AS*

  *SELECT firstname, lastname, salary FROM employees;*

Резултат от примера:

```
table TEMP created.
```

Допълнителна информация на: http://www.w3schools.com/sql/sql_create_table.asp

**Команда DESCRIBE** – дава информация за дизайна на таблица в ORACLE.

*DESCRIBE tEmp;*

Резултат от примера:

```
Name       Null     Type
---------- -------- ------------
FIRST_NAME          VARCHAR2(20)
LAST_NAME  NOT NULL VARCHAR2(25)
SALARY              NUMBER(8,2)
```

**Използване на клаузи Alter Table и Drop Table.**

**Конструкция ALTER TABLE** – използва се за промяна в дизайна на таблица.

- Добавяне на поле в таблица.

*ALTER TABLE <име на таблицата>*
*ADD (<име на поле> <тип>,*
*<име на поле > <тип>);*

- Промяна на характеристика на поле в таблица. Характеристики могат да бъдат тип на поле, допускане на NULL ст-сти, ст-ст по подразбиране и др.

*ALTER TABLE <име на таблицата>*
*MODIFY (<име на поле > <тип>,*
*<име на поле > <тип> <друга х-ка>);*

- Преименуване на поле.

*ALTER TABLE <име на таблицата>*
*RENAME COLUMN <старо име> TO <ново име>;*

- Изтриване на поле.

*ALTER TABLE <име на таблицата>*
*DROP COLUMN <име на колона> ;*
**или**
*ALTER TABLE <име на таблицата>*
*DROP (<име на колона>, <име на колона> );*

**Конструкция DROP TABLE** – използва се за премахване на таблица.

*DROP TABLE <име на таблицата>;*

Пример:

*DROP TABLE tEmp;*

Резултат от примера:

```
table TEMP altered.
```

Конструкцията не генерира изходни данни - генерира се потвърждение за изтриване на обекта.

Допълнителна информация на: http://www.w3schools.com/sql/sql_alter.asp

И на: http://www.w3schools.com/sql/sql_drop.asp

**Характеристики на полета на таблици.**

Освен типа на дадено поле в таблица има и други характеристики, които задават условия, на които трябва да отговарят данните в дадено поле. Тези характеристики са известни като ограничения или ограничаващи фактори – constraint.

Основна тяхна задача е да поддържат интегритета на данните.

- Видове:
  - PRIMARY KEY
  - FOREIGN KEY
  - NOT NULL
  - CHECK
  - UNIQUE
  - DEFAULT

Видовете са описани по-долу.

- Пример:

```
CREATE TABLE tEmp (empid number(6,0) NOT NULL UNIQUE,
                   fname varchar2(75) NOT NULL,
                   lname varchar2(75) NOT NULL,
                   salary number(8,2) DEFAULT(1000),
                   HireDate DATE,
                   departmentid number(6,0),
                   managerid number(6,0)
                   );
DESCRIBE tEmp;
```

Резултат от примера:

```
table TEMP created.
DESCRIBE tEmp
Name         Null     Type
------------ -------- ------------
EMPID                 NUMBER(6)
FNAME        NOT NULL VARCHAR2(75)
LNAME        NOT NULL VARCHAR2(75)
HIREDATE              DATE
ADDRESS               VARCHAR2(40)
SALARY                NUMBER(8,2)
DEPARTMENTID          NUMBER(6)
MANAGERID             NUMBER(6)
```

**Първичен ключ (Primary key)** – едно или няколко полета, с уникална стойност за всеки запис. Първичен ключ може да се дефинира като характеристика на поле.

```
CREATE TABLE tEmp (empid number(6,0) PRIMARY KEY,
                   fname varchar2(75),
                   lname varchar2(75),
                   salary number(6,2),
                   HireDate DATE,
                   departmentid number(6,0),
                   managerid number(6,0)
                   );
DESCRIBE tEmp;
```

Резултат от примера:

```
table TEMP created.
DESCRIBE tEmp
Name         Null     Type
------------ -------- ------------
EMPID        NOT NULL NUMBER(6)
FNAME        NOT NULL VARCHAR2(75)
LNAME        NOT NULL VARCHAR2(75)
HIREDATE              DATE
ADDRESS               VARCHAR2(40)
SALARY                NUMBER(8,2)
DEPARTMENTID          NUMBER(6)
MANAGERID             NUMBER(6)
```

Първичен ключ може да се дефинира след описанието на всички полета на ниво таблица. Резултатът от следващата конструкция е същия като предходния.

```
CREATE TABLE tEmp (empid number(6,0),
                   fname varchar2(75),
                   lname varchar2(75),
                   salary number(6,2),
                   HireDate DATE,
                   departmentid number(6,0),
                   managerid number(6,0),
                   PRIMARY KEY(empid )
                   );
```

Първичен ключ  се дефинира задължително след описанието на всички полета на ниво таблица ако първичния ключ е съставен.

*CREATE TABLE tSales (empid number(6,0),*

*clientid number(6,0),*
*sdate DATE,*

***PRIMARY KEY****(empid, clientid )*

*);*

*DESCRIBE tEmp;*

Резултат от примера:

```
table TSALES created.
DESCRIBE tSales
Name     Null     Type
-------- -------- ---------
EMPID    NOT NULL NUMBER(6)
CLIENTID NOT NULL NUMBER(6)
SDATE             DATE
```

**Външен ключ** - поле от таблица, което се реферира към поле - първичния ключ на друга таблица.

**Дефиниране на Външен ключ.**

Външен ключ може да се дефинира като характеристика на поле.

*CREATE TABLE tEmp*

*(empid number(6,0),*

*fname varchar2(75),*

*lname varchar2(75),*

*salary number(6,2),*

*HireDate DATE,*

*departid number(6,0)* ***REFERENCES*** *tdepartments (departmentid )*

*);*

*DESCRIBE tEmp;*

Резултат от примера:

```
table TEMP created.
DESCRIBE tEmp
Name     Null     Type
-------- -------- ------------
EMPID             NUMBER(6)
FNAME    NOT NULL VARCHAR2(75)
LNAME    NOT NULL VARCHAR2(75)
HIREDATE          DATE
ADDRESS           VARCHAR2(40)
SALARY            NUMBER(8,2)
DEPARTID          NUMBER(6)
```

Външен ключ може да се дефинира след описанието на всички полета на ниво таблица. Резултатът е същият като в горния пример.

*CREATE TABLE tEmp (empid number(6,0),*

*fname varchar2(75),*

*lname varchar2(75),*

*salary number(6,2),*

*HireDate DATE,*

*departmentid number(6,0),*

***FOREIGN KEY (****departmentid)*

***REFERENCES*** *tdepartments (department_id )*

*);*

**Опции при дефиниране на външен ключ в клаузата REFERENCES.**

- **ON DELETE RESTRICT** - забранява изтриване при свързани таблици
- **ON DELETE CASCADE –** при изтриване на запис от референтната таблица изтрива каскадно свързаните с този запис реферираци записи.
- **ON DELETE SET NULL –** при изтриване на запис от референтната таблица записва във външния ключ на реферираците записи ст-ст NULL.
- **ON DELETE SET DEFAULT –** при изтриване на запис от референтната таблица записва във външния ключ на реферираците записи предварително зададена като характеристика ст-ст по подразбиране.
- Пример:

```
CREATE TABLE temployee (
    ….
    tdepartmentid NUMBER(2)
    REFERENCES tdepartment(department_id )
    ON DELETE CASCADE);
```

**Характеристики на полета в таблици.**

Добавяне на ограничения.

```
ALTER TABLE tEmp
        ADD CONSTRAINT emp_dep_fk
        FOREIGN KEY (departmentid)
        REFERENCES tdepartments (department_id )
                );
```

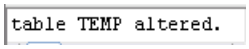
Резултат от примера:

```
table TEMP altered.
```

Спиране на ограничения – за временно спиране вместо премахване.

*ALTER TABLE tEmp  MODIFY CONSTRAINT emp_dep_fk DISABLE;*

Резултат от примера:

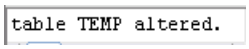



Промяна характеристики на поле.

*ALTER TABLE tEmp*

*MODIFY (salary DEFAULT 1000);*

Резултат от примера:

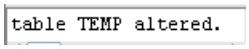



*ALTER TABLE tEmp*

*MODIFY (FNAME  NOT NULL,*

*LNAME NOT NULL);*

Резултат от примера:




Допълнителна информация на: http://www.w3schools.com/sql/sql_constraints.asp

**Създаване и премахване на обект View.**

**Обект VIEW** – обект, чието съдържание се извлича от други таблици посредством SELECT заявка.

При обръщение към View, заявката се изпълнява.

Често се нарича виртуална таблица

**Команда CREATE VIEW** – служи за създаване и дефиниране на обект VIEW. Конструкцията не генерира изходни данни. Генерира се потвърждение за създаване на нов обект.

*CREATE VIEW <име_на_изгледа>*

*AS <SQL заявка>;*

Пример:

*CREATE VIEW Vteamsadnmembers*

*AS SELECT name, firstname, lastname*

*FROM teams t JOIN members m ON t.teamid=m.teamid;*

Резултат от примера:

```
view VTEAMSANDMEMBERS created.
```

**Премахване на обект VIEW. Команда DROP VIEW.**

*DROP VIEW <име_на_изгледа> ;*

Пример:

*DROP VIEW Vteamsadnmembers*

Резултат от примера:

```
view VTEAMSANDMEMBERS dropped.
```

Конструкцията не генерира изходни данни - генерира се потвърждение за изтриване на обекта.

Допълнителна информация може да се намери на:

http://www.w3schools.com/sql/sql_view.asp

# Модул 8: Използване на DML за модифициране на данни

**Въвеждане на данни (Insert клауза).**

INSERT клаузата позволява въвеждане на данни в таблица.

Данни в таблица могат да се въвеждат по два начина. Експлицитно и имплицитно.

**Експлицитно въвеждане на данни** – в конструкцията INSERT INTO изрично се указват имената на полетата, в които ще се въвеждат данни**.**

*INSERT INTO tdepartments (department_id, department_name, manager_id)*

*VALUES(350, 'Support, 110);*

Резултат от примера:

```
1 rows inserted.
```

**Имплицитно въвеждане на данни** – при това въвеждане не се указват имената на полетата, в които ще се въвеждат данни**.** Изисква стойности за всички полета в таблицата като се спазва реда на дефиниция на полетата в таблицата.

*INSERT INTO tdepartments*

*VALUES(350, 'Support, 110);*

Резултат от примера:

```
1 rows inserted.
```

**Въвеждане на данни чрез SELECT** – предоставя възможност да се въвеждат данни с помощта на SQL заявка.

*INSERT INTO tdepartments SELECT department_id, department_name, manager_id*

*FROM departments;*

Резултат от примера:

```
27 rows inserted.
```

Примерите ползват таблици от схема HR, както и таблици създадени в примери от предишния модул в същата схема

Допълнителна информация може да се намери на:

http://www.w3schools.com/sql/sql_insert.asp
http://www.w3schools.com/sql/sql_insert_into_select.asp

**Промяна на данни (Update клауза).**

UPDATE клаузата позволява промяна на данни в таблица.

Синтаксис:

*UPDATE <име на таблица>*

*SET <име на колона1>=<стойност>,*

*<име на колона1>=<стойност>,*

*<име на колона1>=<стойност>*

*WHERE <критерии за филтриране>*

Допълнителна информация може да се намери на:

http://www.w3schools.com/sql/sql_update.asp

**Изтриване на данни (Delete клауза).**

DELETE клаузата позволява да бъдат изтривани записи в таблица.

Синтаксис:

*DELETE FROM <име на таблица>*

*WHERE <критерии за филтриране>*

DELETE клаузата позволява филтриране на записите, които да бъдат изтрити.

Допълнителна информация може да се намери на:

http://www.w3schools.com/sql/sql_delete.asp

Примери за въвеждане, промяна и изтриване на данни с таблици от демонстрационната база. Примерите са с таблици от схема Student2.

```
CREATE TABLE newtbl AS SELECT * FROM members;
```

Резултат:

```
table NEWTBL created.
```

```
SELECT * FROM newtbl;
```

Резултат:

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 500 | John | Jones | 000000000 | 75000 | 5000 | M | Y | 01-FEB-90 | (null) | (null) | 400 |
| 501 | Mary | Jones | 111111111 | 30000 | (null) | F | N | 02-MAR-02 | (null) | 506 | 401 |
| 502 | Antonios | Vamvakeros | 222222222 | 30000 | (null) | M | N | 15-MAR-02 | (null) | 506 | 401 |
| 503 | Louisa | Lindboe | 333333333 | 30000 | 1000 | F | Y | 01-JAN-02 | (null) | 506 | 401 |
| 504 | Cara | Montopoli | 444444444 | 35000 | 2000 | F | Y | 01-JAN-01 | (null) | 506 | 401 |
| 505 | Michael | Dupre | 555555555 | 50000 | 4000 | M | Y | 12-DEC-97 | (null) | 500 | 402 |
| 506 | Migumi | Zona | 666666666 | 75000 | 5000 | F | N | 01-MAR-90 | (null) | 500 | 400 |
| 507 | Wey | Ho | 777777777 | 30000 | (null) | M | N | 01-FEB-90 | (null) | 505 | 402 |
| 508 | Joseph | Smith | 888888888 | 0 | 0 | M | N | 01-FEB-90 | 15-JUN-02 | (null) | (null) |

```
UPDATE newtbl
    SET salary=salary*1.2
WHERE bonus IS NULL;
```

Резултат:

```
3 rows updated.
```

```
SELECT * FROM newtbl;
```

Резултат:

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 500 | John | Jones | 000000000 | 75000 | 5000 | M | Y | 01-FEB-90 | (null) | (null) | 400 |
| 501 | Mary | Jones | 111111111 | 36000 | (null) | F | N | 02-MAR-02 | (null) | 506 | 401 |
| 502 | Antonios | Vamvakeros | 222222222 | 36000 | (null) | M | N | 15-MAR-02 | (null) | 506 | 401 |
| 503 | Louisa | Lindboe | 333333333 | 30000 | 1000 | F | Y | 01-JAN-02 | (null) | 506 | 401 |
| 504 | Cara | Montopoli | 444444444 | 35000 | 2000 | F | Y | 01-JAN-01 | (null) | 506 | 401 |
| 505 | Michael | Dupre | 555555555 | 50000 | 4000 | M | Y | 12-DEC-97 | (null) | 500 | 402 |
| 506 | Migumi | Zona | 666666666 | 75000 | 5000 | F | N | 01-MAR-90 | (null) | 500 | 400 |
| 507 | Wey | Ho | 777777777 | 36000 | (null) | M | N | 01-FEB-90 | (null) | 505 | 402 |
| 508 | Joseph | Smith | 888888888 | 0 | 0 | M | N | 01-FEB-90 | 15-JUN-02 | (null) | (null) |

```
INSERT INTO newtbl (firstname, lastname, teamid)
VALUES('Ivan', 'Ivanov', 403);
```

Резултат:

```
1 rows inserted.
```

```
SELECT * FROM newtbl;
```

Резултат:

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 500 | John | Jones | 000000000 | 75000 | 5000 | M | Y | 01-FEB-90 | (null) | (null) | 400 |
| 501 | Mary | Jones | 111111111 | 36000 | (null) | F | N | 02-MAR-02 | (null) | 506 | 401 |
| 502 | Antonios | Vamvakeros | 222222222 | 36000 | (null) | M | N | 15-MAR-02 | (null) | 506 | 401 |
| 503 | Louisa | Lindboe | 333333333 | 30000 | 1000 | F | Y | 01-JAN-02 | (null) | 506 | 401 |
| 504 | Cara | Montopoli | 444444444 | 35000 | 2000 | F | Y | 01-JAN-01 | (null) | 506 | 401 |
| 505 | Michael | Dupre | 555555555 | 50000 | 4000 | M | Y | 12-DEC-97 | (null) | 500 | 402 |
| 506 | Migumi | Zona | 666666666 | 75000 | 5000 | F | N | 01-MAR-90 | (null) | 500 | 400 |
| 507 | Wey | Ho | 777777777 | 36000 | (null) | M | N | 01-FEB-90 | (null) | 505 | 402 |
| 508 | Joseph | Smith | 888888888 | 0 | 0 | M | N | 01-FEB-90 | 15-JUN-02 | (null) | (null) |
| (null) | Ivan | Ivanov | (null) | (null) | (null) | (null) | (null) | (null) | (null) | (null) | 403 |

```
INSERT INTO newtbl
VALUES(509,'Ivan', 'Ivanov',NULL, 30000, NULL, 'M', 'N', '21-
MAY-15', NULL, 501, 403);
```

Резултат:

```
1 rows inserted.
```

```
SELECT * FROM newtbl;
```

Резултат:

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 500 | John | Jones | 000000000 | 75000 | 5000 | M | Y | 01-FEB-90 | (null) | (null) | 400 |
| 501 | Mary | Jones | 111111111 | 36000 | (null) | F | N | 02-MAR-02 | (null) | 506 | 401 |
| 502 | Antonios | Vamvakeros | 222222222 | 36000 | (null) | M | N | 15-MAR-02 | (null) | 506 | 401 |
| 503 | Louisa | Lindboe | 333333333 | 30000 | 1000 | F | Y | 01-JAN-02 | (null) | 506 | 401 |
| 504 | Cara | Montopoli | 444444444 | 35000 | 2000 | F | Y | 01-JAN-01 | (null) | 506 | 401 |
| 505 | Michael | Dupre | 555555555 | 50000 | 4000 | M | Y | 12-DEC-97 | (null) | 500 | 402 |
| 506 | Migumi | Zona | 666666666 | 75000 | 5000 | F | N | 01-MAR-90 | (null) | 500 | 400 |
| 507 | Wey | Ho | 777777777 | 36000 | (null) | M | N | 01-FEB-90 | (null) | 505 | 402 |
| 508 | Joseph | Smith | 888888888 | 0 | 0 | M | N | 01-FEB-90 | 15-JUN-02 | (null) | (null) |
| (null) | Ivan | Ivanov | (null) | (null) | (null) | (null) | (null) | (null) | (null) | (null) | 403 |
| 509 | Ivan | Ivanov | (null) | 30000 | (null) | M | N | 21-MAY-14 | (null) | 501 | 403 |

```
UPDATE newtbl
SET memberid=510,
    salary=30000,
    gender='M',
    hiredate='21-May-15'
WHERE memberid IS NULL;
```

Резултат:

```
1 rows updated.
```

```
SELECT * FROM newtbl;
```

Резултат:

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 500 | John | Jones | 000000000 | 75000 | 5000 | M | Y | 01-FEB-90 | (null) | (null) | 400 |
| 501 | Mary | Jones | 111111111 | 36000 | (null) | F | N | 02-MAR-02 | (null) | 506 | 401 |
| 502 | Antonios | Vamvakeros | 222222222 | 36000 | (null) | M | N | 15-MAR-02 | (null) | 506 | 401 |
| 503 | Louisa | Lindboe | 333333333 | 30000 | 1000 | F | Y | 01-JAN-02 | (null) | 506 | 401 |
| 504 | Cara | Montopoli | 444444444 | 35000 | 2000 | F | Y | 01-JAN-01 | (null) | 506 | 401 |
| 505 | Michael | Dupre | 555555555 | 50000 | 4000 | M | Y | 12-DEC-97 | (null) | 500 | 402 |
| 506 | Migumi | Zona | 666666666 | 75000 | 5000 | F | N | 01-MAR-90 | (null) | 500 | 400 |
| 507 | Wey | Ho | 777777777 | 36000 | (null) | M | N | 01-FEB-90 | (null) | 505 | 402 |
| 508 | Joseph | Smith | 888888888 | 0 | 0 | M | N | 01-FEB-90 | 15-JUN-02 | (null) | (null) |
| 510 | Ivan | Ivanov | (null) | 30000 | (null) | M | (null) | 21-MAY-14 | (null) | (null) | 403 |
| 509 | Ivan | Ivanov | (null) | 30000 | (null) | M | N | 21-MAY-14 | (null) | 501 | 403 |

```
DELETE newtbl
WHERE memberid>=509;
```

Резултат:

```
2 rows deleted.
```

```
SELECT * FROM newtbl;
```

Резултат:

| MEMBERID | FIRSTNAME | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 500 | John | Jones | 000000000 | 75000 | 5000 | M | Y | 01-FEB-90 | (null) | (null) | 400 |
| 501 | Mary | Jones | 111111111 | 36000 | (null) | F | N | 02-MAR-02 | (null) | 506 | 401 |
| 502 | Antonios | Vamvakeros | 222222222 | 36000 | (null) | M | N | 15-MAR-02 | (null) | 506 | 401 |
| 503 | Louisa | Lindboe | 333333333 | 30000 | 1000 | F | Y | 01-JAN-02 | (null) | 506 | 401 |
| 504 | Cara | Montopoli | 444444444 | 35000 | 2000 | F | Y | 01-JAN-01 | (null) | 506 | 401 |
| 505 | Michael | Dupre | 555555555 | 50000 | 4000 | M | Y | 12-DEC-97 | (null) | 500 | 402 |
| 506 | Migumi | Zona | 666666666 | 75000 | 5000 | F | N | 01-MAR-90 | (null) | 500 | 400 |
| 507 | Wey | Ho | 777777777 | 36000 | (null) | M | N | 01-FEB-90 | (null) | 505 | 402 |
| 508 | Joseph | Smith | 888888888 | 0 | 0 | M | N | 01-FEB-90 | 15-JUN-02 | (null) | (null) |

**Изтриване на данни от таблица с команда TRUNCATE.**

Командата **TRUNCATE** изтрива всички данни от указаната таблица. Няма клауза за филтриране на данни.

```
TRUNCATE TABLE newtbl;
```

Резултат:

```
table NEWTBL truncated.
```

```
SELECT * FROM newtbl;
```

Резултат:

| MEMBERID | FIRSTN... | LASTNAME | TAXID | SALARY | BONUS | GENDER | MARRIED | HIREDATE | FIREDATE | MENTORID | TEAMID |
|---|---|---|---|---|---|---|---|---|---|---|---|

# Модул 9: Въведение в Oracle

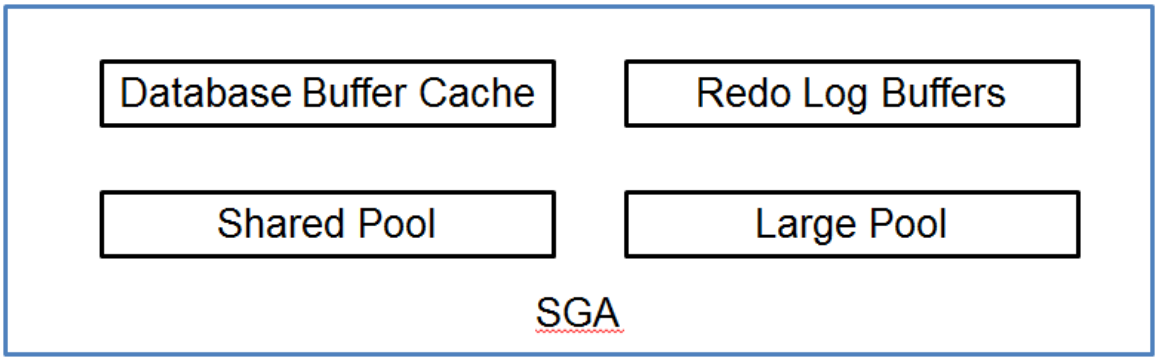
**Общи положения при Oracle инсталация.**

**Основни термини.**

- **ORACLE_HOME** - при инсталация на Oracle DB софтуер, всички компоненти се инсталират в една папка наречена ***Oracle Home*** directory.
- **ORACLE_SID** - всяка Oracle DB трябва да има уникален системен идентификатор.
- При Windows ORACLE_HOME и ORACLE_SID могат да се изчетат от Windows Registry.
- Всяка ORACLE система се състои от два основни компонента ORACLE Database и ORACLE Instance.
- **ORACLE Database** - всяка Oracle база данни се състои от набор файлове.
    - Файлове с данни - Datafiles
    - Лог файлове - Redo log files
    - Контролен файл - Control File
    - Параметричен файл - Parameter file
- **ORACLE Instance** - набор от работещи процеси в паметта съставя ORACLE Instance.
    - ORACLE Instance трябва да е стартиран, за да бъдат достъпни файловете съставящи ORACLE Database.
    - Един ORACLE Instance може да управлява само една ORACLE Database.

**Структури в паметта ползвани от ORACLE Instance.**

***System Global Area(SGA)*** **–** област в RAM паметта ползвана от ORACLE система за управление на бази данни. SGA се подразделя на следните области:




- Всяка инстанция има своя SGA.
- SGA се заема при стартиране на инстанцията.
- Database Buffer Cache – за поместване на данни от диска в памет, която е по-бърза като достъп.
- Redo Log Buffers – съдържа записи на транзакции.
- Shared Pool – компоненти за управление на SQL конструкции.
- Large Pool – за управление на големи блокове с данни.

***Program Global Area(PGA)*** – за всеки сървърен процес в ORACLE се заделя и област от RAM памет необходима за работата му.

***User Global Area (UGA)*** – памет асоциирана с потребителска сесия.

**Идентифициране на процесите в един Oracle Instance.**

- System Monitor (SMON) – мониторинг на инстанцията. Изпълнява възстановяване на инстанцията(instance recovery) при стартиране ако е необходимо. Почиства временните сегменти ако не са необходими повече и обединява свободното пространство.
- Process Monitor (PMON) – мониторинг на потребителските сесии. Почиства след ненормално приключила потребителска сесия.
- Log Writer (LGWR) – Прехвърля записи от Log Buffer в online redo log файловете.
- Database Writer (DBWn) – Записва променените в Buffer cashe записи във файловете с данни.

- Checkpoint Process (CKPT) – изпълнява database checkpoint. При checkpoint всички променени буфери се записват по дисковете. Файловете с данни съдържат времето на последния checkpoint в хедъра си.
- Recoverer (RECO) – възстановява неуспешни транзакции.
- Manageability Monitor (MMON) - управлява статистиките в Oracle server
- Memory Manager (MMAN) – управлява размера на SGA
- Archiver Processes (ARCn) – управлява архивирането на online redo log файловете.

**Сървърни процеси** - за всяка клиентска сесия в ORACLE се заделя сървърен процес.

- Функции:
  - ✓ Преглежда и изпълнява клиентските SQL заявки
  - ✓ Извлича необходимите данни според заявката.
- *Program Global Area*(PGA) – за всеки сървърен процес в ORACLE се заделя и област от RAM памет необходима за работата му.
- Типове сървърни процеси:
  - ✓ Dedicated Server процеси – всяка клиентска сесия има свой собствен сървърен процес.
  - ✓ Shared Server процеси – потребителите споделят процеси.

Допълнителна информация може да се намери на:

https://docs.oracle.com/cd/E11882_01/server.112/e40540/memory.htm#CNCPT007

https://docs.oracle.com/cd/E11882_01/server.112/e40540/process.htm#CNCPT008

**Идентифициране на файловете на Oracle база данни.**

***Файлове за данни.***

- Съдържат потребителски и мета данни.
- Тип на файловете с данни - бинарен.
- Роля на Сървърните процеси - Сървърните процеси отговарят за прехвърлянето на данни от файловете с данни в Buffer Cache.
- Роля на *Database Writer* (DBW*n*) процес – отговаря за записа на променени данни от Buffer Cache във файловете с данни.

***Структури във файловете с данни.***

- Blocks – блокове с данни. Последователност от сектори на диска. 2K, 4K, 8K, 16K, 32K. За Oracle 11g - 8K по подразбиране.
- Extents – последователност от блокове с данни.
- Segments – съставен е от поне един extent. Всеки обект в базата се асоциира с конкретен сегмент.

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/E11882_01/server.112/e40540/logical.htm#CNCPT004

***Контролни файлове***

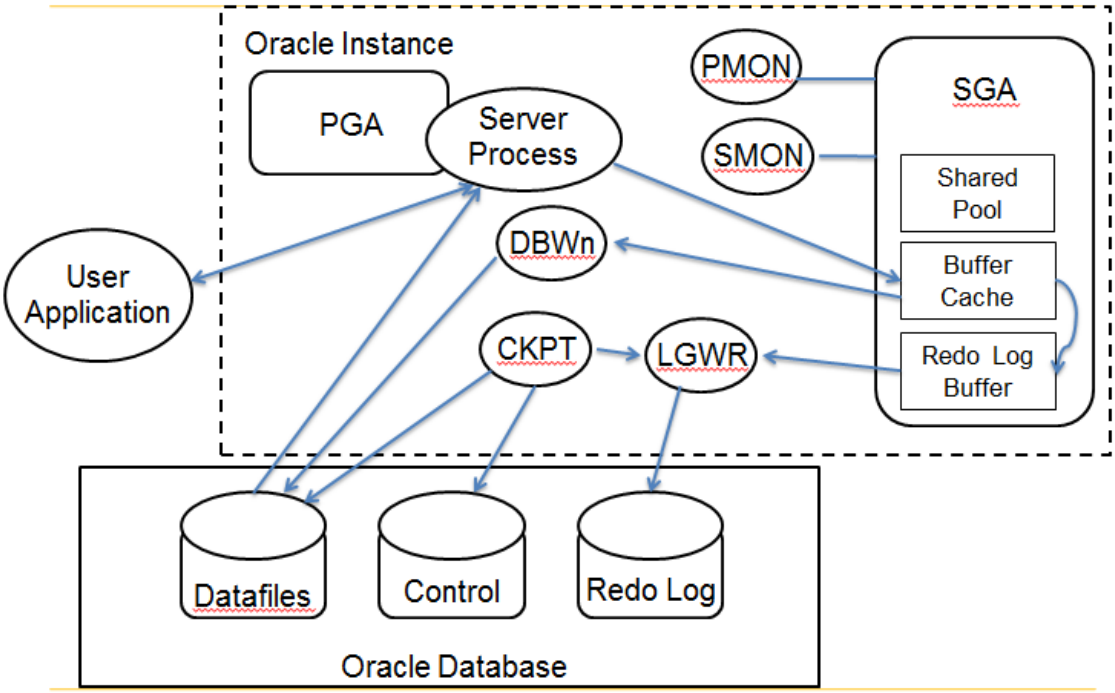
- Съдържат служебна информация за базата данни.
- Основен файл за базата данни.
- Копия на контролен файл – поради важността на контролния файл се правят копия.

***Redo Log файлове***

- Съдържат информация за всички транзакции в базата данни.
- Основен механизъм за възстановяване на данни.
- Redo buffer – разположен е в SGA и с помощта на Log Writer (LGWR) от него се прехвърлят записи в online redo log файловете.

**Описание на архитектурата и основните операции в Oracle база данни.**

По-горе описаните процеси в **Oracle Instance,** както и гореописаните области от паметта се обвързват в следната схема:




При инициализиране на потребителска сесия от клиентско приложение(User Application) се заделя сървърен процес. Той проверява Buffer Cache дали изискваните данни се намират там. Ако ги няма то сървърния процес изисква от качването им от файловете за данните в паметта. Ако се налага обработка на данните то тя се извършва в паметта. Резултата се връща към клиентската сесия.

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/E11882_01/server.112/e40540/process.htm#CNCPT008

**Използване на SYS и SYSTEM logins в Oracle база данни.**

- При инсталация на Oracle database се създават привилегировани потребители SYS и SYSTEM.
- SYS – може да изпълнява всички администраторски функции.
- SYSTEM – потребител създаден, за ежедневни администраторски задачи.

# Модул 10: Стартиране и спиране на Oracle база данни

**SYSDBA и SYSOPER системни привилегии .**

Използват се за изпълнение на административни задачи като стартиране и спиране на инстанция.

**SYSOPER** е привилегия за системни оператори.

**SYSOPER системни привилегии:**

- STARTUP – стартиране на инстанция
- SHUTDOWN – спиране на инстанция
- CREATE SPFILE – създаване на файл с конфигурационни параметри
- CONNECT – отваряне на сесия към инстанция
- ALTER DATABASE – команда за промяна на база със следните параметри
  - OPEN
  - MOUNT
  - ARCHIVELOG
  - BACKUP
  - RECOVER – complete recovery

SYSOPER се използва за ежедневните администраторски задачи, като например създаването на архиви и управление на ORACLE база данни.

**SYSDBA** позволява изпълнението на задачи от DBA (Data Base Administrators).

**SYSDBA системни привилегии:**

- SYSDBA включва всички привилегии на SYSOPER, както и:
  - ✓ CREATE DATABASE
  - ✓ DROP DATABASE
  - ✓ ALTER DATABASE
  - ✓ RECOVER – всякаква форма на partial recovery

Пример за използване на SYSDBA може да бъде първоначалното създаване на ORACLE база данни.

**Стартиране на Oracle инстанция.**

- Фази при нормално стартиране:

  *Down* -> **NOMOUNT** -> **MOUNT** -> **OPEN**

- 1. **NOMOUNT** фаза(**STARTUPNOMOUNT**).
  - Изчита се параметричния файл
  - Инициализира се Oracle instance.
- 2. **MOUNT** stage (**STARTUPMOUNT**)– изчита се информацията от контролния файл.
- 3. **OPEN** stage (**STARTUP OR STARTUPOPEN**) – отварят се файловете с данни и Log файловете.

- Restricted mode – DBA може да стартира базата, така че да не се допускат потребителски сесии. Ползва се командата STARTUP RESTRICT.
- След като базата е стартирана в определено състояние, не се допуска преминаване в предходното.

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/B28359_01/server.111/b31189/ch12045.htm#SQPUG127

**Спиране на Oracle инстанция.**

- **SHUTDOWN** или **SHUTDOWN NORMAL** - не се позволява нови сесии. Изчаква текущите сесии да приключат. Всички файлове се затварят след като се запишат промените.
- **SHUTDOWN TRANSACTIONAL** - не се допускат нови сесии. След приключване на текущите транзакции се спират всички сесии. Всички файлове се затварят след като се запишат промените.
- **SHUTDOWN IMMEDIATE** – непотвърдените транзакции се анулират. Всички сесии се прекратяват незабавно. Всички файлове се затварят. Ако има непотвърдени промени то те се анулират.
- **SHUTDOWN ABORT** - използва се когато някой от процесите в ORACLE INSTANCE спре аварийно.

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/B28359_01/server.111/b31189/ch12042.htm

**Промяна на стартови режими на Oracle инстанция.**

След стартиране на Oracle instance с командата **STARTUP NOMOUNT** съществуват опции за отваряне на асоциираните файлове с данни.

- ALTER DATABASE MOUNT;
- ALTER DATABASE OPEN;

С цел изпълнение на администраторски задачи е възможно превключване на базата в Restricted режим. Това е режим, при който само потребители със зададена RESTRICTED SESSION привилегия могат да достъпват базата. Пример: превключване в този режим за изпълнение на дейности по поддръжка на базата, като разширяване на файловете с данни.

- ALTER SYSTEM ENABLE RESTRICTED SESSION;
- ALTER SYSTEM DISABLE RESTRICTED SESSION;

Превключване на базата в режим само за четене и обратно може да се направи със следните команди:

- ALTER DATABASE OPEN READ ONLY;
- ALTER DATABASE OPEN READ WRITE;

Когато е необходимо да се създаде архив без спиране на базата е възможно да се ползва режим SUSPEND. При превключване в този режим текущите входно изходни операции се завършват, а новите се зареждат на опашка докато не се изключи режима с командата RESUME.

- ALTER SYSTEM SUSPEND
- ALTER SYSTEM RESUME

Базата може да бъде превключена в режим QUIESCE, при който единствено са разрешение операции за потребители SYS и SYSTEM.

- ALTER SYSTEM QUIESCE RESTRICTED;
- ALTER SYSTEM UNQUIESCE;

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/B28359_01/backup.111/b28273/rcmsynta006.htm#RCMRF105

http://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_2013.htm#SQLRF00902

**Запис на активността на базата.**

- Alert Log - текстов файл, в който се записват информативни съобщения и съобщения за грешка.
- Местоположението на Alert Log може да са получи от инициализиращия параметър **BACKGROUND_DUMP_DEST**.

**Data Dictionary** - във всяка система за управление на база данни се съхраняват служебни данни необходими за работата на системата. Тези данни се съхраняват в системни таблици. Информацията от служебните таблици може да се извлича с помощта на заявки, обекти от тип View и системни функции. Съвкупността от тези служебни таблици, обекти от тип View и, системни функции в ORACLE Database се нарича Data Dictionary. Информацията от тези обекти се извлича с помощта на SELECT конструкции.

**Data Dictionary информация относно Startup/Shutdown може да се извлече от следните обекти.**

- V$DATABASE - информация за базата.
- V$INSTANCE – информация за текущия ORACLE instance.

# Модул 11: Параметрични файлове

**Предназначение и описание.**

Предназначение – за конфигуриране и настройка. Oracle 11g съдържа 250 параметъра.

Oracle reference guide има секция Basic Initialization Parameters с описани 27 параметъра които се нуждаят от настройки за добра производителност.

Повечето параметри имат стойности по подразбиране.

Параметрите биват 3 основни типа:

- Наследени.
- Зависими от Операционната система.
- Променливи.

Допълнителна информация може да се намери на:

https://docs.oracle.com/cd/B28359_01/server.111/b28320/initparams002.htm#REFRN00101

**Динамични и статични параметри**

Статичните параметри не подлежат на промяна докато е стартиран ORACLE instance.

Динамичните параметри могат да се променят при работеща база.

**V$SYSTEM_PARAMETER** view съдържа информация кои параметри са статични и кои динамични.

- ISSES_MODIFIABLE – поле, чиято стойност определя дали даден параметър може да се променя с ALTER SESSION команда.
- ISSYS_MODIFIABLE – поле, чиято стойност определя дали даден параметър може да се променя с ALTER SYSTEM команда. Възможните стойности са:
    - FALSE – не подлежи на промяна
    - IMMEDIATE – промяната ще се отрази на всички сесии
    - DEFERRED – промяната ще се отрази на нови сесии

**Параметрични файлове.**

- **PFILE –** текстов файл с описани параметри и техните стойности.
- **SPFILE –** бинарен файл съдържащ параметри и стойности.
- Стандартно се намират в: OracleHome\database или OracleHome\dbs.

**Предимства на SPFILE пред PFILE.**

- Чрез SPFILE е възможна динамична промяна на параметри.
- Направените промени на динамични параметри в SPFILE остават и след рестарт.
- Ползването на SPFILE позволява стартиране на базата от отдалечен клиент.

**Добавяне, премахване и промяна на параметрични файлове.**

**Създаване на SPFILE от PFILE и обратно.**

- Създаване на PFILE от SPFILE:

  *CREATE PFILE FROM SPFILE;*

  *CREATE PFILE='filename' FROM SPFILE ='filename';*

- Създаване на SPFILE от PFILE:

  *CREATE SPFILE FROM PFILE;*

  *CREATE SPFILE='filename' FROM PFILE ='filename' ;*

- Не е възможно преместването на SPFILE докато инстанцията е стартирана.

**Администриране.**

**Обхват на промените при параметри.**

Обхватът определя къде са валидни стойностите на динамичните параметри.

Съществуват 3 стойности на обхват:

- **MEMORY –** в паметта.
- **SPFILE –** в параметричния файл.
- **BOTH –** в паметта и в параметричния файл едновременно.

**Настройване на инициализиращи параметри.**

**Сесийни параметри.**

- Някои параметри могат да бъдат валидни в рамките на определена сесия.
- Не влияят върху базата или инстанцията.
- National Language Setting (**NLS**) параметрите са пример за сесийни параметри.

  *ALTER SESSION SET NLS_DATE_FORMAT='DD.MM.YYYY';*

Преглед на параметричните стойности както на системни така и за сесийни параметри може да се направи с помощта на командата SHOW PARAMETER или с някой от описаните по-долу Data Dictionary обекти.

**Data Dictionary информация за параметри.**

- **V$SYSTEM_PARAMETER** – информация за актуалните параметри на инстанцията.
- **V$PARAMETER** - информация за актуалните параметри на текущата сесия.
- **V$SPPARAMETER** – показва съдържанието на SP файла.

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/B19306_01/server.102/b14237/initparams003.htm#i1124392

# Модул 12: Файлове с данни

**Предназначение и описание.**

Предназначение – файловете за данни поддържат съдържанието на ORACLE база-та данни.

Всяка ORACLE 11g база данни има поне 2 файла с данни.

Всеки файл за данни задължително се асоциира с една определена ORACLE база данни.

Един или няколко Database Writer processes (DBW*n*) са отговорни за записа на промените във файловете с данни.

Сървърните процеси се занимават с изчитането на данни от файловете с данни.

**Таблични пространства (Tablespaces) -** логически структури за групиране на обекти в базата данни.

Всяка ORACLE база данни е логически разделена на едно или повече таблични пространства .

При ORACLE 11g - SYSTEM и SYSAUX.

**Характеристики на Таблични пространства.**

Всяко таблично пространство принадлежи само към една база.

Едно таблично пространство се състои от един или повече файлове за данни.

Един файл за данни може да принадлежи само на едно таблично пространство.

Общ обем на таблично пространство – обема на всички файлове за данни на съот-ветното таблично пространство както и временни файлове.

Таблично пространство се ползва и като единица подлежаща на архивиране.

**Типове таблични пространства.**

- Permanent– използват се за постоянни обекти на базата данни.
- Temporary – временно съхранение на сесийни данни.
- Undo– предишни версии на данни необходими за запазване на консистент-ност или възстановяване.

**SYSTEM и SYSAUX таблични пространства.**

Таблиците на Data Dictionary се намират в SYSTEM.

- SYSTEM табличното пространство се създава при създаване на базата.
- Ако не съществува, то базата не може да се стартира.
- Стартирана инстанция ще спре да работи, ако по някаква причина SYSTEM табличното пространство стане недостъпно.

Oracle 11g база данни изисква наличието на SYSAUX.

- SYSAUX табличното пространство съдържа компоненти на ORACLE Enterprise Manager, Recovery Manager, ORACLE Scheduler и др.
- С помощта на SYSAUX табличното пространство Data Dictionary таблиците са отделени от таблиците на другите компоненти на Oracle Database.
- SYSAUX табличното пространство е необходимо при стартиране на инстанцията, но Oracle инстанцията няма да спре да работи ако SYSAUX табличното пространство стане недостъпно.

Не се допуска преименоване на SYSTEM и SYSAUX.

**Създаване на структури във файлове с данни (tablespaces).**

**Създаване на таблични пространства.**

- **CREATE TABLESPACE команда**

  *CREATE [UNDO | TEMPORARY] TABLESPACE ts_name*

  *[DATAFILE | TEMPFILE] file_specification;*

  Освен създаван на стандартни таблични пространства за съхранение на данни е възможно създаването и на:

  - **Temporary** – за временно съхранение на данни.
  - **UNDO** – използва се за консистентност и възстановяване на данни.

  Файловете на табличното пространство могат да са DATAFILE и TEMPFILE.

- **Форма на файловата спецификация.** Служи за описание местоположението на файловете на табличното пространство като и първоначален размер, максимален размер и параметър за разширяване на файла. Размерите на файла и размера на разширяване могат да се задават в KB, MB, GB или TB,

  *'/directory/filename' SIZE xx[K|M|G|T]*

  *[AUTOEXTEND [ON|OFF] [NEXT xx[K|M|G|T]*

  *[MAXSIZE xx|UNLIMITED]]*

Компоненти на файловата спецификация:

Първоначален размер – задава се с ключовата дума SIZE.

Параметър за разширяване на файла – ако се налага файлът да се уголеми можа да се използва автоматично разширяване AUTOEXTEND с указание с колко да е това разширение чрез ключовата дума NEXT.

Максимален размер на файла - задава се с ключовата дума MAXSIZE.

**Пример:** създаване на таблично пространство course_data, с файл course_data01.dbf, чийто първоначален размер е 100 MB, максимален размер 500 MB и автоматично нарастване 100MB.

*CREATE TABLESPACE course_data*

*DATAFILE 'c:\oraclexe\app\oracle\oradata\XE\course_data01.dbf' SIZE 100M*

*AUTOEXEND ON NEXT 100M MAXSIZE 500M;*

Резултат:

```
tablespace COURSE_DATA created.
```

Ако една операция не може като цяло да се изпълни в паметта то тя се разделя на части, които се изпълняват поотделно. Информацията за това се записва във временни сегменти. Временните(TEMPORARY) таблични пространства съхраняват такава информация в рамките на определена сесия. Пример за такава операция е сортирането на данни. По принцип могат да бъдат създавани множество временни таблични пространства, но е достатъчно и едно.

*CREATE TEMPORARY TABLESPACE temp_data*

*TEMPFILE 'c:\oraclexe\app\oracle\oradata\XE\temp_data01.dbf' SIZE 100M*

*AUTOEXEND ON;*

Резултат:

```
temporary TABLESPACE created.
```

**Dictionary и Locally Managed таблични пространства.**

- При създаването на обект в таблично пространство за него се заделя поне един екстент.
- Управлението може да се осъществява по 2 начина:
  - Dictionary-Managed Tablespace (DMT) - – описанието на заеманите екстенти се пази в таблици от DATA DICTIONARY
  - Locally Managed Tablespace (LMT) - описанието е в хедъра на файловете с данни.
- Предимства на LMT:
  - Управлението е по-ефективно.
  - Не е необходимо обединяване на свободни екстенти.
  - Липсва фрагментация при освобождаване на пространство.

**Пример:** създаване на таблично пространство course_data, с файл course_data01.dbf, чийто първоначален размер е 100 MB, максимален размер 500 MB и автоматично нарастване 100MB. Управлението на екстентите ще е локално във файла за данни. Определянето на размера на екстента ще става автоматично.

*CREATE TABLESPACE course_data*

*DATAFILE 'c:\oraclexe\app\oracle\oradata\XE\course_data01.dbf' SIZE 100M*

*AUTOEXEND ON NEXT 100M MAXSIZE 500M*

*EXTENT MANGEMENT LOCAL AUTOALLOCATE;*

Резултат:

```
tablespace STUDENT_DATA created.
```

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/B28359_01/server.111/b28310/tspaces002.htm#ADMIN11359

**Администриране.**

**Таблично пространство по подразбиране.**

- SYSTEM – по подразбиране за всички потребители.
- Може да се променя за конкретен потребител.
- Може и да се укаже да бъде друго табличното пространство по подразбиране.

**Квота на Таблично пространство.**

- На потребителите се дефинира квота на използваното пространство.

  *CREATE USER username IDENTIFIED BY password*

  *QUOTA UNLIMITED ON ts_name;*

  *ALTER USER username QUOTA xx [K | M] ON ts_name;*
- Квота 0 за даден потребител означава, че той не може да създава обекти.
- **UNLIMITED TABLESPACE –** неограничена квота **.**

**Премахване и промяна на таблично пространство.**

- Необходима е DROP TABLESPACE привилегия. Пример:

  *DROP TABLESPACE ts_name;*
- Необходима е ALTER TABLESPACE привилегия.

**Преименуване на таблично пространство.**

- Възможно е от ORACLE 10g.
- Табличното пространство и неговите файлове за данни трябва да са online.
- COMPATIBLE параметъра да е поне 10.0.0.
- Не е възможно да се преименоват SYSTEM и SYSAUX.

**Преименуване на файлове за данни.**

- Необходима е ALTER DATABASE привилегия.
- Процедура за преименуване на файл за данни:
  - ✓ Спира се инстанцията.
  - ✓ Преименува се файла с команди на OS.
  - ✓ Стартира се инстанцията в MOUNT режим.
  - ✓ Стартира се командата **ALTER DATABASE RENAME FILE**, с която се обновяват контролните файлове.
  - ✓ Преминава се в режим OPEN database.

**Data Dictionary информация за файлове с данни.**

- **DBA_TABLESPACES** – информация за табличните пространства.
- **DBA_DATA_FILES** – информация за файловете с данни.
- **DBA_FREE_SPACE** – информация за свободно пространство в базата.
- **DBA_SEGMENTS** – информация за сегментите в базата.

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/B28359_01/server.111/b28310/dfiles.htm#i1106090

# Модул 13: Контролни файлове

**Описание на контролни файлове.**

- Контролният файл е малък бинарен файл, който съдържа жизненоважна за системата информация.
- Всяка ORACLE база данни се нуждае от поне 1 контролен файл за да може да се стартира.
- При всяка промяна на физическата структура е необходимо да се прави архив на контролния файл.
- **CONTROL_FILES** параметър съдържа местоположението на контролните файлове.

**Пример:** при архивиране на контролния файл е необходимо да се знае неговото местоположение. Това може да се определи с помощта на командата SHOW PARAMETER Control_Files стартирана в програмата SQL plus.

**Информация в контролния файл.**

- Име на базата.
- Имена на табличните пространства на базата.
- Имена и местоположение на файловете за данни и redo logs.
- Log история
- История на архивирането на Log файловете.
- Checkpoint информация
- Текущ log sequence number

Т.к. контролният файл е от изключително важно значение е възможно да има няколко негови копия. Те са записани като местоположение в параметъра control_files.

**Пример** за извличане на списъка с копия на контролния файл:

*SELECT value FROM v$parameterw2 WHERE name='control_files';*

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/B28359_01/server.111/b28310/control.htm#i1006088

**Администриране на контролни файлове.**

**Backup на контролен файл.**

- COLD backup – копиране на контролен файл при спряна инстанция.
- On Line backup – архивиране при работеща инстанция чрез създаване на бинарно копие.

**Пример**:

*ALTER DATABASE BACKUP CONTROLFILE TO '/tmp/controlback.ctl‘*

- On Line backup – архивиране при работеща инстанция чрез създаване на trace файл.

**Пример**

*ALTER DATABASE BACKUP CONTROLFILE TO TRACE AS '/temp/controltrace.txt'*

**Възстановяване на контролен файл – процедура.**

- SHUTDOWN ABORT.
- Копира се архивното бинарно копие в същото местоположение където е бил стария контролен файл със същото име.
- STARTUP MOUNT

RECOVER DATABASE USING BACKUP CONTROLFILE UNTIL CANCEL;

- При поискване от системата се указва определен redo log файл.
- При появата на съобщение MEDIA RECOVERY COMPLETE се стартира следната команда:

  ALTER DATABASE OPEN RESETLOGS
- Препоръчителен е незабавен backup.

**Преместване на контролен файл - процедура.**

- Модифицира се параметричния файл според новото местоположение на контролния файл.
- Инстанцията се спира.
- Премества се контролния файл чрез OS команда.
- Инстанцията се стартира.

**Data Dictionary информация за контролни файлове.**

- **V$CONTROLFILE** – показва списък на всички контролни файлове.
- **V$CONTROLFILE_RECORD_SECTION** – информация за контролния файл.

# Модул 14: Log файлове

**Предназначение на Online Redo Logs .**

- Redo Log поддържа записи за всички транзакции в Oracle база данни.
- Redo е основен механизъм за възстановяване.
- Redo Log Buffer – буфер, който е източник на записи за Redo Log файловете.
- Log Writer (LGWR) процес – процес, който е отговорен за прехвърляне на записи от Redo Log Buffer в Redo Log файловете.

**Параметри на базата за Redo.**

- Log_BUFFER – определя размера на log буфера в паметта.
- FAST_START_MTTR_TARGET – средно време за възстановяване в секунди. Максималното време, за което данните могат да престоят в паметта преди DBWn процес да ги запише на диск.

**Redo Log групи.**

- Redo записите се записват във файлове, така че да оцелеят при спиране на инстанцията.
- Redo Log файловете са организирани в групи.
- Необходими са минимум 2 групи като във всяка група да има поне 1 файл.
- Log Writer (LGWR) последователно използва всяка една група.

**Брой на Redo Log групите.**

- Зависи от това дали ще се прави архив на Redo Log групите.
- Ако не се прави архив 2-3 групи са достатъчни.
- Ако се прави архив броят трябва да е съобразен така, че архивът да приключи преди Log Writer (LGWR) процеса да използва архивираната група.

Допълнителна информация може да се намери на:

**http://docs.oracle.com/cd/B28359_01/server.111/b28310/onlineredo.htm#i1007497**

**Управление на Log файловете.**

**Създаване на Redo Log.**

- Създаване на група:

  *ALTER DATABASE ADD LOGFILE ('path\filename') SIZE xxx;*

- Добавяне на член към съществуваща група:

  *ALTER DATABASE ADD LOGFILE MEMBER 'path\filreame'*

  *TO GROUP x;*

**Премахване на Redo Log.**

- Преди премахване на дадена група тя трябва да е неактивна.
- Може да се провери статуса в **V$LOG** Data Dictionary view.
- Ако след премахване на дадена група ще остане само една то операцията не е позволена.
- Премахване на група:

  *ALTER DATABASE DROP LOGFILE GROUP x*

- Премахване на член от група:

  *ALTER DATABASE DROP LOGFILE MEMBER 'path\filename'*

**Преименуване на Redo Log файлове.**

- Преименуване и преместване са еднозначни значения в Oracle база данни.
- Стъпки:
  - ✓ Инстанцията се спира.
  - ✓ С OS команда се извършва преименуването.
  - ✓ MOUNT на инстанцията.
  - ✓ Чрез ALTER DATABASE се променя redo log файл името в Oracle.
  - ✓ Open database.

**Превключване на Redo Log групите.**

- Превключването е автоматично.
- Може да се наложи изрично превключване с цел операции по поддръжка.
- **Пример**: използва се командата: *ALTER SYSTEM SWITCH LOGFILE;*

**Конфигуриране на Archive Log.**

**Архивиране на Redo Log групите.**

- За да се архивират Redo Log групите базата трябва да е в ARCHIVELOG режим.
- Предимства на **ARCHIVELOG** режим.
  - Hot backup е възможен
  - Възможно е възстановяване до времето на срива.
  - Възможно е възстановяване към определен момент от време.
- При конфигурирането на базата да архивира redo logs се указва местоположение на архивните копия;

**Конфигуриране на Archive Log режим - процедура.**

- Модифицира се параметър в SPFile

**Пример**: ALTER SYSTEM SET log_archive_dest_1=’L:\ArchLog\’ SCOPE=spfile;

- Спира се инстанцията.
- MOUNT
- Разрешава се автоматично архивиране.

**Пример**: ALTER DATABASE ARCHIVELOG

- OPEN database
- Shutdown database
- BACKUP

**Data Dictionary информация за redo log.**

- **V$LOGFILE** – показва log групите и техните членове.
- **V$LOG** – показва информация от контролния файл касаеща redo log.
- **V$LOG_HISTORY** – дава историческа информация относно redo log.

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/B28359_01/server.111/b28310/archredo.htm#i1008343

# Модул 15: Индекси

**Типове индекси в Oracle .**

**Въведение в индексите.**

- Използват се за ускоряване изпълнението на заявки.
- Друго предназначение на индексите е поддръжката на уникални стойности в дадено поле.
- Индексите могат да бъдат и съставни като се използва повече от едно поле.

**Видове индекси в ORACLE.**

**B-tree индекс.**

- Най-често индексните структури ползват B-tree индекс.
- Дървовидна структура – балансирано дърво.
- Рядко надвишава 3-4 нива
- Root node – корен на дървото.
- Leaf node – листата на дървото. Могат да съдържат ключова стойност, по която е изграден индекса с препратка към физическата таблица или целите редове на таблицата.

**Bitmap индекс**

- Bitmap индекса използва битове за да отбележи наличието на стойност по дадена колона в определен ред.
- Bitmap индекса е по-сполучлив да се използва при Data Warehouse системи.
- Трябва да се избягва използването му при OLTP.
- Възможно е комбинирането на на няколко Bitmap индекса.

**Насоки и препоръки за използване на индекси.**

Индекси трябва да се създават за полета, по които се прави често търсене. Полетата в WHERE клаузите на заявките са кандидати за създаване на индекси.

Възможно е индекс да се създава по повече от едно поле.

За полета дефинирани като първичен ключ и UNIQUE constraint автоматично се създава индекс.

Индексите трябва да се обслужват при добавяне и промяна на данни, така че е необходимо броят им да е съобразен с реалното им използване.

**NULL стойности и индекси**

- B-tree индекс не съхранява NULL стойности в индексен ключ от една колона.
- B-tree индекс няма да съдържа ред на съставен индексен ключ, при който всички колони са NULL.
- Ако Query optimizer определи, че дадена стойност я няма в индекса се изпълнява цялостно сканиране на таблицата.

**Създаване и управление на индекси.**

Команда **CREATE INDEX** – служи за създаване на индекс.

*CREATE [UNIQUE] INDEX index_name ON table_name (col1 [, col2..]*

*[LOGGING|NOLOGGING] [ONLINE] TABLESPACE tablespace_name;*

- Изисквания за използване на командата (поне едно от следните):
  - ✓ Права на собственост върху базата.
  - ✓ INDEX object привилегия върху таблицата, която се индексира.
  - ✓ INDEX ANY TABLE системна привилегия.

Команда **DROP INDEX -** служи за премахване на индекс.

*DROP INDEX index_name;*

- Изисквания за използване на командата (поне едно от следните):
  - ✓ Права на собственост върху индекса.
  - ✓ DROP ANY TABLE системна привилегия.

**REBUILD и местене на индекси.**

- Фрагментация на индекс – при промяна на данни или въвеждане на нови в поле, по което има изграден индекс се стига до момент, в който индексът е фрагментиран-неподреден и неефективен.
- REBUILD помага да се премахне фрагментацията.

  *ALTER INDEX index_name REBUILD [ONLINE];*
- Преместване на индекс.

  *ALTER INDEX index_name REBUILD TABLESPACE ts_name [ONLINE];*

**Компресиране на индекс.**

- Води до намаляване обема заеман от индекса.
- COMPRESS клауза в CREATE INDEX конструкция.

  *CREATE [UNIQUE] INDEX index_name ON table_name (column_list)*

  *TABLESPACE tablespace_name COMPRESS ;*
- Необходимо е да се направи REBUILD на индекса когато се разрешава или забранява компресиране.

**Data Dictionary информация за индекси.**

- **DBA_INDEXES** – обща информация за индексите в базата.
- **DBA_IND_COLUMNS** – информация кои са индексните колони.

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/B28359_01/server.111/b28310/indexes.htm#i1007132

# Модул 16: Oracle Optimizer

**Общо описание на Oracle Optimizer.**

Oracle Optimizer определя възможно най-ефективния начин за изпълнението на SQL конструкции.

Oracle използва Cost-Based Optimizer (CBO).

Определящи фактори за избор на най-ефективен план:

- ✓ Размер на таблицата.
- ✓ Среден брой редове в блок.
- ✓ Наличие на индекси

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/E25054_01/server.1111/e16638/optimops.htm

**Статистики и необходимост от тях.**

Cost-Based Optimizer (CBO) се нуждае от факти относно данните в базата и производителността на системата.

За нуждите на *Cost-Based Optimizer* е изграден механизъм за автоматично събиране на статистики.

ORACLE 11g подържа график за периодично събиране на статистики.

**Събиране, премахване и използване на статистики.**

**DBMS_STATS** *пакет* съдържа процедури за събиране на статистики.

*GATHER_TABLE_STATS*

*GATHER_INDEX_STATS*

2 основни начина за натрупване на статистики:

- ➢ Изследване на всички редове в таблицата.
- ➢ Изследване на % редове – извадка.

**DBMS_STATS** *пакет* се ползва също и за премахване на статистики относно различни обекти в базата.

*DELETE_COLUMN_STATS*

*DELETE_TABLE_STATS*

*DELETE_INDEX_STATS*

*DELETE_SCHEMA_STATS*

**Data Dictionary информация за статистики.**

- **DBA_TAB_STATISTICS** – обща информация относно статистиките.
- **DBA_IND_STATISTICS** – информация за статистиките свързани с индекси.
- **DBA_TAB_COL_STATISTICS** – информация относно статистики по колони.
- **DBA_TAB_HISTOGRAMS** – информация за крайни стойности в статистиките за колоните в таблиците.

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/E25054_01/server.1111/e16638/stats.htm

# Модул 17: Back UP на Oracle база данни

**Описание на offline cold backup.**

Backup и recovery операциите са от първостепенно значение са всяка система за управление на бази данни.

Backup и recovery стратегия – стратегия по отношение на архивирането на данни и поддръжка на архивите с данни. Основна цел е да бъдат възстановени данните при евентуален срив без загуба на данни и за възможно най-кратък срок.

Backup и recovery стратегията трябва да отговаря на организационните изисквания по отношение на допустимото време за възстановяване. Т.е. в зависимост от тези изисквания да бъде определен период, през който да се правят архивите, тип на архива, местоположение на архива, период на съхранение на архива.

В зависимост от местоположението различаваме Физически и логически Backup.

**Физически Backup** – устройството за Backup се управлява локално.

**Логически Backup** – устройството за Backup е логическо и неговото управление е изнесено външно за системата.

В зависимост от типа на Backup различаваме следните архивирания.

**Offline Backup**

Най-лесен като принцип.

Нарича се още *cold backup,* т.к. инстанцията е спряна.

Offline Backup изисква да се копират следните физически компоненти:

- ✓ Файловете за данни.
- ✓ Контролният/те файл/ове.
- ✓ Online redo log файловете.

**Процедура за изпълнение на offline cold backup.**

- Спира се инстанцията.
- Копират се физическите файлове в Backup местоположение.
- Стартиране на инстанцията.

**Пример** за подготовка на offline cold backup.

За да се направи такова архивиране е необходимо да се знаят местоположенията на файловете, които ще бъдат архивирани(копирани). Такава информация може да се събере със следния скрипт:

```
SPOOL c:\backuplist.txt
SELECT file_name FROM dba_data_files
SELECT member FROM v$logfile
SELECT name FROM v$controlfile
SELECT value FROM v$system_parameter WHERE name = 'spfile'
SPOOL OFF
```

Командата SPOOL се използва за пренасочване на резултатите от скриптовете. В примера пренасочването е към текстови файл.

**Допълнителни препоръки при Backup.**

- Backup на програмното осигуряване от ORACLE_HOME directory.
- Параметрични файлове.
- Password файлове.
- Скриптови файлове.

**Други възможности за Backup.**

- Online backups(hot backups) – архивиране при работеща система.
- Online backups изискват базата да е конфигурирана в ARCHIVELOG режим.

**RMAN(**Recovery Manager) – ORACLE клиент, предназначен да изпълнява операции свързани с Backup и Restore.

- Създава cold backup.
- Създава hot backup.
- Възможност за възстановяване на блокове.
- Възможност за компресия.
- Автоматично архивиране на контролни файлове и SPFILE.

**Пример** за offline backup с използване на RMAN:

*C:\rman*

*RMAN> SHUTDOWN IMMEDIATE;*

*RMAN> STARTUP MOUNT;*

*RMAN> BACKUP DATABASE;*

Описание на командите в примера.

1. Стартира се RMAN от команден ред на операционната система. При това се отваря команден режим на RMAN.
2. От командният ред на RMAN се спира ORACLE с помощта на командата SHUTDOWN IMMEDIATE;
3. Стартира се базата в MOUNT режим с командата STARTUP MOUNT;
4. Стартира се командата BACKUP DATABASE, при което базата се архивира в мястото за архив по подразбиране. То се указва още при инсталация на ORACLE, но може да преконфигурира и по-късно. При тази команда RMAN архивира всички компоненти на ORACLE DATABASE.

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/B28359_01/server.111/b28318/backrec.htm#CNCPT031

# Модул 18: Възстановяване на Oracle база данни

**Общи положения.**

Restore – процес, при който се копират файлове от backup media.

Recovery – процес, при който се възстановяват промените настъпили между последния backup и срива на системата.

Режим **ARCHIVELOG –** режим, при който се прави архив на Redo Log файловете. Дава възможност за възстановяване на базата до момента на срива.

**Възстановяване с помощта на Offline Backups при NOARCHIVE база.**

Ако поне 1 файл за данни липсва или е повреден, то се налага възстановяване на цялата база.

**Процедура** за възстановяване на Oracle база данни от offline cold backup:

- ✓ Ако инстанцията все още е стартирана, то тя трябва да се спре.
- ✓ Копират се файловете за данни, контролни файлове и online redo logs от направен преди това backup.
- ✓ Стартира се инстанцията

**Възстановяване с допълнителни файлове**

Възстановяването може да е съпроводено и с възстановяване на допълнително файлове ако те са били архивирани. Такива файлове са:

- Параметрични файлове
- password файлове
- RDBMS софтуер – файлове от ORACLE програмното осигуряване.

**Пример**: ако малко или повече повредата е засегнала ORACLE програмното осигуряване то може да се наложи предварително да бъде направена нова инсталация на ORACLE програмното осигуряване и след това да бъдат копирани файловете съставящи ORACLE DATABASE, т.е да се възстанови базата данни. За това възстановяване на базата данни се прилага процедурата описана по-горе.

Допълнителна информация може да се намери на:

http://docs.oracle.com/cd/B28359_01/server.111/b28318/backrec.htm#CNCPT031

# Модул 19: Въведение в MS SQL Server 2012

**Въведение в Microsoft SQL Server.**

Microsoft SQL Server е система за управление на релационни бази данни.

Разполага с широк набор от средства и инструменти за поддържане на данни.

Това е интегрирана платформа която предоставя различни услуги като:

- Database Engine – основната услуга за поддържане на базите данни.
- Integration Services – основна услуга за поддържане на бази данни.
- Analysis Services – средства за анализ на данни.
- Reporting Services – генериране на отчети.
- Master Data Services – набор от средства и правила за уеднаквяване на записи от различни източници.
- Data Quality Services – задаване и следена на правила за качество на данните.
- Replication – синхронизиране на данни.
- Full-Text Search – пълнотекстово търсене.
- PowerPivot, Power View – средства за анализи и справки.

Допълнителна информация може да се намери на:

https://technet.microsoft.com/en-us/library/cc645993(v=sql.110).aspx

**Варианти на MS SQL Server.**

Microsoft SQL Server се предлага в следните пакети(варианти) разделени в 3 категории.

- Premium – категория на пакети с включени всички услуги, инструменти и механизми налични в Microsoft SQL Server без ограничения в използваните ресурси.
  - Enterprise – този пакет е само софтуерен.
  - Parallel Data Warehouse – това е единственият пакет, който се предлага в комбинация с хардуер, т.к. предлага паралелна обработка, при която има зависимост между хардуер и софтуер.
- Core – основни варианти, при които има ограничения по отношение на ресурси и услуги.
  - Standard – вариант, предназначен за компании, които се нуждаят от основната услуга на Microsoft SQL Server за поддържане на данни.

- Business Intelligence – освен поддържане на данни включва и услуги за генериране на анализ и отчети.

- Other – допълнителни варианти
  - Express – основана безплатна версия с ограничения в използването на ресурси и услуги.
  - Developer – предназначен за компании, чиято дейност е разработката на програмни продукти.

Допълнителна информация може да се намери на:

https://technet.microsoft.com/en-us/library/ms144275(v=sql.110).aspx

**Клиент – сървър архитектура.**

Microsoft SQL Server се базира на клиент сървър архитектура, при която Microsoft SQL Server предоставя услуги ползвани от клиентски приложения. За реализацията на такава архитектура се ползва специфичен протокол и начини за автентификация

- **TDS( Tabular Data Stream) протокол**
  - SQL Native Access Client (SNAC) – отговорен за транслирането към TDS.
  - Транспортни среди, които ползва TDS могат да бъдат:
    - TCP/IP – стандартен мрежови протокол
    - NamedPipes - мрежови протокол на Microsoft
    - Shared Memory – транспортна среда използвана при положение, че клиента и сървъра са на една и съща физическа машина.
- **Начини за автентификация**
  - Windows logins
  - Windows групи
  - SQL Server logins

  Начините за автентификация са разгледани в модул 22.

**Средства за работа с MS SQL Server.**

**SQL Server Management Studio**

- Обща среда за:
  - Генериране на заявки.
  - Конфигуриране на MS SQL Server.

  - Управление и наблюдение на MS SQL Server.
  - Администриране на бази данни.
- Лесна за използване графична среда.
- Система включваща проектна организация на скриптове за автоматизиране на задачи.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/ms174173.aspx

**BI** (Business Intelligence) – термин използван в СУБД за описание на проекти, обекти, инструменти и механизми за извличане на данни от СУБД с цел генериране на справки и анализи необходими за нуждите на работещите със СУБД потребители.

**BI проекти** – проекти свързани с реализацията на решения подпомагащи дейността на работещите със СУБД потребители за генериране на справки и анализи.

**SQL Server Data Tools.**

- Интегрирана среда за работа с BI обекти. Обектите са предварително създадени програмни компоненти всеки, от които е със специфична функционалност.
- Включва създаването на BI проекти за:
  - Analysis Services – проекти ползващи услугата Analysis Services на MS SQL Server за създаване на компоненти за анализ на данни и достъп до тези анализи.
  - Integration Services– проекти свързани с прехвърляне на данни от даден източник към определен приемник.
  - Reporting Services– проекти свързани с генерирането на справки.
- Графична среда за реализиране на BI проекти.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/data/tools.aspx

**Конфигуриране на услуги на MS SQL Server.**

**SQL Server Configuration Manager.**

Предоставя средства и инструменти за конфигуриране на:

- SQL Server Services – услугите предоставяни от MS SQL server.
- Network Ports and Listeners – мрежовата конфигурация на MS SQL server.
- Server Aliases – конфигуриране на сървърни псевдоними на клиент.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/ms174212.aspx

**Допълнителни SQL Server инструменти.**

- SQL Server Profiler – продукт за следене на събития в MS SQL server.
- Database Engine Tuning Advisor – използва се за анализ на заявки.
- Master Data Services Configuration Manager – използва се за конфигуриране на услугата Master Data Services.
- Reporting Services Configuration Manager – използва се за конфигуриране на услугата Reporting Services.
- Data Quality Services Client– използва се за конфигуриране на услугата Data Quality Services.

**Помощна информация.**

- MSDN:

  https://msdn.microsoft.com/bg-bg
- Books Online:

  https://technet.microsoft.com/en-us/library/ms130214(v=sql.110).aspx

# Модул 20: Инсталиране на инстанция на MS SQL Server 2012

**SQL Server архитектура.**

SQL Server се състои от различни компоненти, които работят заедно. Три са основните категории компоненти.

- Query Execution Layer – управлява оптимизацията и изпълнението на заявки.
- Storage Engine Layer – управлява поддръжката на данни в базите с данни.
- SQL OS Layer – с помощта на програмни интерфейси управлява паметта и графиците за изпълнение на задачи.

**Подготовка за инсталация.**

**Хардуерни изисквания на MS SQL Server.**

- CPU – всички съвременни процесори отговарят на изискванията за работа на MS SQL Server.
- RAM – оперативната памет е от значение за производителността на всяка компютърна система. Тя трябва да е съобразена с броя на едновременно работещите потребители и с обема на обработваните данни.
- Дисково пространство - трябва да е съобразено с обема на съхраняваните данни и да има достатъчен капацитет за нови данни.
- Виртуализация - MS SQL Server може да работи във виртуална среда като от значение е дисковете използвани за съхранение за данни физически заделени изключително и само за работата на SQL Server.

**Използване на памет от MS SQL Server.**

- Buffer Pool
  - Съдържа data cache
  - Предоставя памет за SQL Server компонентите
  - Разделен е на по 8K страници (pages)
- SQL OS автоматично усвоява необходимата му памет.
  - Разполага с механизъм за предизвестие при намаляване на паметта.
  - Min и Max опции за конфигуриране.

**Планиране на хардуерните изисквания.**

- CPU – съвременните процесори покриват изискванията на MS SQL Server.
- RAM – да се ползва препоръчителен обем според документацията от Microsoft. По възможност да се ползва колкото се може повече памет.
- Мрежови ресурси
  - Клиенти – един или повече мрежови интерфейси трябва да бъдат предоставяне за клиентски сесии.
  - Административен канал – добра практика е да има отделен канал за административни задачи.
- Дискова подсистема
  - RAID, SAN
  - Предварителен тест – Microsoft предоставят два безплатни продукта за тестване на дисковата система симулиращи четене и запис във файловете за данни и log файловете това са: SQLIOSIM и SQLIO.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/library/bb500442(v=sql.110).aspx

https://www.microsoft.com/en-us/download/details.aspx?id=20163

https://support.microsoft.com/en-us/kb/231619

**Софтуерни изисквания на MS SQL Server.**

**Изисквания към операционната система.**

- За предпочитане е 64 битова ОС.
- Поддържани операционни системи
  - ✓ Сървърни
  - ✓ Клиентски
- Да се избягва инсталация върху домейн контролер.

**Допълнителни софтуерни изисквания към MS SQL Server 2012 .**

- .NET Framework 4.0
- PowerShell 2.0
- Windows Installer 4.5
- Internet Explorer 6 SP1 or later
- Network
  - Shared memory
  - Named pipes
  - TCP/IP

**Типове файлове и препоръки за тяхното местоположение.**

- Файлове за данни - при възможност се разполагат на отделни дискове.
- Transaction Log файлове - при възможност се разполагат на отделен диск.
- Tempdb – при възможност се разполага на отделен диск.

**Collation -** набор от правила дефиниращи:

- Кодова таблица
- Отчитане на малки и големи букви
- Сортиране (подредба) на символи.

Допълнителна информация може да се намери на:
https://msdn.microsoft.com/en-us/library/ms143506(v=sql.110).aspx

**Инсталиране на MS SQL Server.**

- Трябва да са спазени предварителните изисквания – ОС, хардуер, софтуер.
- SQL Setup пакет.
  - ✓ Проверка на необходимите условия за инсталация (софтуер и хардуер)
  - ✓ Избор на компоненти.
  - ✓ Конфигуриране на инстанцията.
  - ✓ Инсталиране

**Проверка след инсталиране на MS SQL Server.**

- Проверка на инсталираните услуги - SQL Server Configuration Manager.
- Грешките по време на инсталацията се записват във файл.

  %programfiles%\Microsoft SQL Server\110\Setup Bootstrap\Log

**UPGRADE на MS SQL Server.**

- На място върху съществуваща инсталация.
  - ✓ Лесен автоматизиран вариант.
  - ✓ Не се изисква допълнителен хардуер.
- Side-by-side.
  - ✓ Не се работи директно върху продуктовата среда.
  - ✓ Могат да се проведат предварителни тестове.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/library/bb500395(v=sql.110).aspx

# Модул 21: Бази данни в MS SQL Server

**Описание на файлове на бази данни.**

- Всяка база се съхранява в определени за нея файлове за данни.
- Файлове за данни
  - Primary Data file: .mdf
  - Secondary data file: .ndf
  - Данните се съхраняват с страници, като всяка страница е от по 8KB.
  - 8 последователно свързани страници(pages) образуват extent.
- Transaction Log file: .ldf

**Определяне на местоположение и брой на файловете.**

- При възможност файловете за данни и Transaction Log файловете трябва да са на различни дискове.
- Броят на файловете за данни се определя според изискванията за производителност и поддръжка.
  - Могат да се разполага на няколко диска.
  - По-малки файлове, по-лесно се обслужват
  - backup и restore са фактори, с които трябва да се съобразим.
- Transaction Log file – 1 файл за база е достатъчен.

**Подсигуряване на достатъчен файлов капацитет.**

- Предварителна оценка за нарастването на файловете.
  - Тестове с данни
  - Препоръки от производителя на приложението.
- Осигуряване на достатъчен резерв.
  - Файловете се организират с достатъчно свободно пространство.
  - Редовно наблюдение.
  - Да се планира регулярна промяна на размера.
  - autogrowth опцията да е включена.

**Системни бази.**

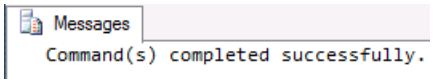
- Master – съдържа мета данни в системни таблици необходими за конфигурирането и работата на системата за бази данни Database Engine.
- MSDB – съдържа конфигурационни данни необходими за работата на SQL Server Agent.
- Model – шаблон за създаването на нови бази.
- Tempdb – използва се за временно съхранение на данни. При рестартиране на инстанция се изчиства и връща към първоначален размер.
- Resource – съдържа набор от системни изгледи(view), функции и процедури. Тази база е скрита.

**Създаване на потребителски бази.**

- С помощта на графичен интерфейс от SSMS (SQL Server Management Studio) .
- Чрез използване на T-SQL

```
CREATE DATABASE Brand
ON
( NAME = Brand_dat,
   FILENAME = 'D:\Data\Brand.mdf',  SIZE = 100MB,
   MAXSIZE = 500MB,  FILEGROWTH = 20% )
LOG ON
( NAME = Brand_log,
   FILENAME = 'L:\Logs\Brand.ldf',  SIZE = 20MB,
   MAXSIZE = UNLIMITED,  FILEGROWTH = 10MB );
```

Резултатът от тази команда е потвърждение на операцията.



**Промяна на база данни.**

- Файловете за данни могат да бъдат добавяне, премахвани, разширявани или свивани.
- Операциите могат да се извършват online чрез:
  - SSMS
  - ALTER DATABASE команда
  - DBCC команда (CHECKDB и SHRINKFILE опции)
- Премахването на файл изисква предварително данните му да бъдат прехвърлени по другите файлове с данни.

**Файлови групи.**

- Именувани групи от файлове за данни с цел указване на определено местоположение на обекти в базата.
- С помощта на файлови групи обектите в една база могат да се разполагат на различни дискове.
- Групите могат да се и фактор в стратегията за архивиране и възстановяване.
- Типове:
  - Primary – задължителна група при създаване на нова база.
  - Други именувани групи.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-US/library/ms186312.aspx

https://msdn.microsoft.com/en-us/ms189563.aspx

**Работа с файлове на бази данни.**

**Преместване на файлове с потребителски данни**

Основни методи:

- Detach/Attach команди.
- ALTER DATABASE.
- Backup/Restore команди.
- Copy Database Wizard - съветник с последователни стъпки.

**Преместване на файлове на системните бази.**

- Единствено resource базата не може да се премества.
- Master базата изисква промяна на Startup параметър на SQL Server service. Файлът задължително се премества с команда на ОС.
- За останалите бази се използва ALTER DATABASE командата.

Допълнителна информация може да се намери на:

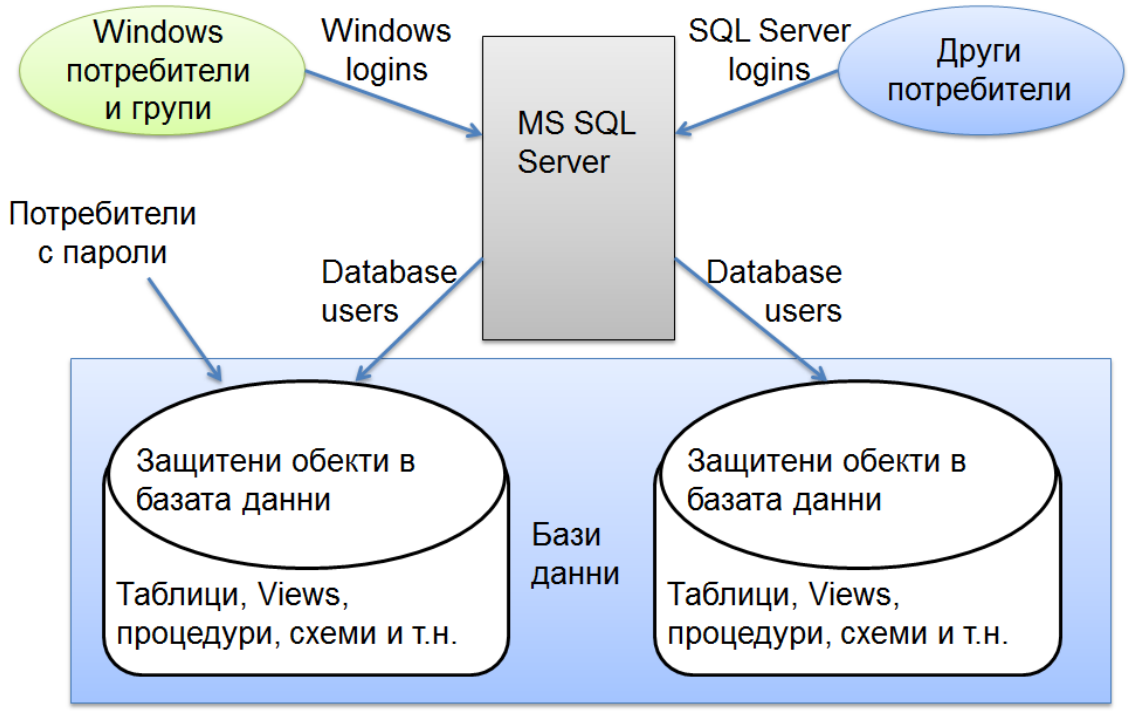
https://msdn.microsoft.com/library/ms345483.aspx

https://msdn.microsoft.com/en-us/library/ms345408.aspx

# Модул 22: Системи за поддръжка сигурността на достъп до MS SQL Server

**Проверка на достъпа до MS SQL Server.**

**Общ преглед на сигурността в MS SQL Server.**




**Видове автентификация SQL Server.**

- Автентификация – процес на установяване на идентичността.
- SQL Server може да бъде конфигуриран в 2 режима:
  - Windows Authentication
  - SQL Server and Windows Authentication

**Управление на Windows Logins.**

- Създаването на Windows Logins може да стане чрез:
  - Object Explorer в SSMS.
  - T-SQL CREATE LOGIN конструкция.
- Премахване – чрез командата DROP.

**Създаване на Windows Logins.**

```
CREATE LOGIN [DEMO\PStudov] FROM WINDOWS
WITH DEFAULT_DATABASE=[tempdb],
DEFAULT_LANGUAGE=[us_english];
GO
CREATE LOGIN [DEMO\Salesteam] FROM WINDOWS;
GO
```

**Управление на SQL Server Logins.**

- Създаването на SQL Server Logins може да стане чрез:
  - Object Explorer в SSMS.
  - Скриптове на T-SQL - CREATE LOGIN конструкция. (T-SQL или Transact SQL е модификацията на Microsoft за езика SQL)
- Премахване – чрез командата DROP.
- ALTER LOGIN команда при:
  - Смяна на парола.
  - Disable/Enable.

**Създаване на SQL Server Logins.**

```
CREATE LOGIN SalesMngr
WITH PASSWORD = 'P@ssw0rd', CHECK_POLICY = ON;
GO
CREATE LOGIN SalesApplication
WITH PASSWORD = 'P@ssw0rd', CHECK_POLICY = OFF;
GO
```

**Предоставяне на права за използване на база данни.**

- За да може един login да има достъп до база данни то в базата данни трябва да се създаде обект database user, който да се асоциира със съответен login. Асоциирането става чрез Security ID (SID) при създаването на database user.
- Създаването на Database User може да стане чрез:
  - Object Explorer в SSMS.
  - T-SQL CREATE USER конструкция.

**Създаване на Database User.**

```
CREATE USER SalesM
FOR LOGIN SalesM;
GO
CREATE USER PStudov
FOR LOGIN [AdventureWorks\ PStudov];
GO
CREATE USER SalesApplication
FOR LOGIN SalesApplication ;
GO
```

**Вградени потребители.**

Всяка база данни в MS SQL Server съдържа по подразбиране два вградени потребителя

- Dbo – който има специални позволения да извършва всички операции в базата. SA login както и членовете на роля sysadmin са асоциирани към dbo и са собственици на всички бази.
- Guest - позволява на login без user да достъпва базата. По подразбиране е заключен.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/library/aa337562.aspx
https://msdn.microsoft.com/en-us/library/aa337545.aspx

**Създаване на потребители директно към конкретна база.**

- Позволено е когато базата е с включена опция ‘contained‘.
- Три типа
  - Windows Потребител без login.
  - Потребител от Windows група без login.
  - Contained database user с парола.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/ff929071.aspx

**Проблем с неасоциирани идентификатори Security ID(**SID**) между login и database user.**

За да може един обект login да има достъп до база данни то в базата данни трябва да се създаде обект database user, който да се асоциира със съответен login. Асоциирането става с помоща на идентификатор Security ID (SID) при създаването на database user. При тази асоциация и двата обекта имат еднакъв идентификатор Security ID (SID).

- От казаното по- горе за обектите Logins и Database Users може да се каже следното:
  - И двата обекта имат имена.
  - И двата обекта имат идентификатор SID.
  - Ако няма съответствие на SID между Login и Database User, то те не се асоциират един с друг.
- Проблемът се получава при Restore или Attach на база.
- Проблемът има 2 начина за разрешаване:
  - Указване на SID при създаване на login.
  - Коригиране на login SID в базата след restore или attach.

```
ALTER USER TestUser WITH LOGIN = TestUser;
GO
```

**Използване на роли в до MS SQL Server.**

Сървърни роли – използват се за задаване на правомощия на група потребители в сървъра (logins)

- Фиксирани роли.
- Потребителски дефинирани роли.

Роли в базите данни.

- Фиксирани роли.
- Потребителски дефинирани роли.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/library/ms188659.aspx

https://msdn.microsoft.com/en-us/ms189121.aspx

**Предоставяне на права за използване на ресурси.**

Команди:

- GRANT – задава позволение.
- DENY – изрично забраняване на достъп
  - Използва се при наследени права.
- REVOKE – премахва GRANT или DENY.
- WITH GRANT – потребител, на който му се даде дадено правомощие с тази допълнителна опция има правото да дава съответното дадено му правомощие на други потребители.

Допълнителна информация може да се намери на:

https://technet.microsoft.com/en-us/library/ms187965(v=sql.110).aspx

https://technet.microsoft.com/en-us/library/ms188338(v=sql.110).aspx

https://technet.microsoft.com/en-us/library/ms187728(v=sql.110).aspx

**Одитиране на MS SQL Server.**

**Възможности за одит:**

- Тригери – обекти в базите данни асоциирани с действията INSERT, UPDATE или DELETE върху дадена таблица.
- SQL Trace – набор от системни процедури отнасящи се до определени събития в работата на MS SQL Server.
- SQL Server Audit – обекти в MS SQL Server позволяващи дефинирането на събития, при настъпването на които се прави запис в определени за целта местоположения.

**Използване на SQL Server Audit.**

- Базиран е на механизма Extended Events. Този механизъм представлява съвкупност от процедури, които следят различните събития възникващи в MS SQL Server.
- Основни елементи:
  - Обект Audit – дефинира общите компоненти на системата за одит като местоположение на резултатите от одита. Общ обект ползван от събития описани в одитните спецификации.
  - Server спецификации – описание на следени събития настъпили на ниво сървър. Асоциира се с Обект Audit, за да може резултата да бъде записан в определено местоположение.
  - Database Audit спецификации – описание на следени събития настъпили на ниво база. Асоциира се с Обект Audit, за да може резултата да бъде записан в определено местоположение.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/cc280386.aspx

https://msdn.microsoft.com/en-us/library/cc280525.aspx

https://msdn.microsoft.com/en-us/library/cc280424.aspx

# Модул 23: Архивиране на бази данни в MS SQL Server

**Стратегии за архивиране и възстановяване.**

Определяне на стратегия за BACKUP.

- Комбиниране на различни типове BACKUP:
  - Data based Backups (Full, Differential, Copy, Filegroup, File)
  - Log based Backups (Log , Tail Log)
- Изисквания по отношение съхранението на данни:
  - Колко време има за възстановяване?
  - Колко е приемливия обем данни, който може да се загуби?

Други фактори при определяне на стратегия за BACKUP.

- Тип и честота на правения BACKUP.
- Медия, която ще се ползва.
- Период на съхранение на конкретен BACKUP и медия.
- Тестова политика.

**Понятия при създаването на BACKUP.**

- Backup set – всяко едно стартиране на команда за backup генерира един backup set
- Media set – това е контейнер за съхранение на един или повече backup set. Примерно файл. В 1 файл може да има един или повече backup set.
- Backup devices – могат да бъдат физически или логически устройства.

Допълнителна информация може да се намери на:

https://technet.microsoft.com/en-us/library/ms191239%28v=sql.105%29.aspx

**SQL Server Transaction Logging.**

Transaction Logs – съдържат история на действията изпълнявани в СУБД.

Дейности изпълнявани от СУБД при модифициране на данни:

- Дадено приложение заявява промяна на данни.
- СУБД извършва промяна на данните в buffer cache.
- Промяната се записва в transaction log файл.
- По-късно checkpoint процес записва промените и във файловете за данни.

**Transaction Log файл.**

- Съдържа информация, която да позволи операциите по възстановяване при срив:
  - Roll back
  - Recovery
- Transaction Log файловете се попълват в хронологичен ред и се преизползват.
- Базират се на recovery model опцията на базата данни.

**Модели за архивиране.**

Recovery модели на база данни в SQL Server

- Simple – при този режим на базата log файла съдържа история на транзакциите за по-кратък период от време. Не се изисква архивиране на log файл.
- Full – log файла съдържа пълна история на транзакциите. Изисква се архивиране на log файл.
- Bulk Logged – режим предвиден за операции при голям обем от данни. Включва се преди тези операции и се изключва след като те приключат.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/ms189275.aspx

**Типове BACKUP в SQL Server.**

- Full – backup на цялата база.
- Differential – backup на промените от последния backup до настоящия момент.
- Partial – съдържа основните файлови групи(Primary Filegroups).
- Transaction Log – backup на Transaction Log файла на базата.
- Tail-log Backup - backup на Transaction Log файла след настъпване на срив.
- File/File Group – backup само на файл или само на файлова група.
- Copy Only – самостоятелен backup, който се използва за пренос на данни.

**Архивиране на файлове с данни и LOG файлове.**

**Full BACKUP.**

BACKUP на цялата база.

BACKUP на активната част на log файла на базата.

Може да се направи чрез:

- графичния интерфейс на SSMS.
- T-SQL команда.

```
BACKUP DATABASE AdventureWorks
TO DISK =
 'L:\SQLBackups\AW.bak'
WITH INIT;
```

**BACKUP компресия.**

- Възможна е от SQL Server 2008.
- Значително намалява обема при BACKUP.
- Намалява I/O за сметка на CPU
- По- малко време за BACKUP
- По- малко време за RESTORE
- Не може в един media set да им компресиран и некомпресиран backup set.

**Differential BACKUP.**

- BACKUP на промените в базата от последния FULL BACKUP .
- BACKUP на активната част на log файла на базата.
- Всеки Differential BACKUP е независим от останалите.
- Задължителен FULL BACKUP преди Differential.

```
BACKUP DATABASE AdventureWorks
TO DISK = 'L:\SQLBackups\AW_Diff.bak'
WITH DIFFERENTIAL, INIT;
```

**Transaction Log BACKUP.**

- BACKUP на Transaction Log само.
- BACKUP на тази част на log файла, която е след последно направения Transaction Log Backup.
- Премахва неактивната час на log файла.
- Моделът на базата трябва да е full или bulk-logged.

```
BACKUP DATABASE AdventureWorks
TO DISK = 'L:\SQLBackups\AW_Diff.bak'
WITH DIFFERENTIAL, INIT;
```

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/library/ms186865(v=sql.110).aspx

**Управление на архивирането.**

- BACKUP не спира базата
- При BACKUP може да се наблюдава намаляване на бързодействие.
- Базата трябва да online.
- Transaction Log BACKUP може да се направи след като базата се е повредила, но при положение, че Transaction Log файла е цял.

**Tail-log BACKUP.**

- Служи, за да прихващане на последната информация от Transaction Log преди RESTORE. Това е backup на Transaction Log файла след настъпване на срив.
- Опции:
  - NORECOVERY – използва се ако ще следва операция по възстановяване директно от Transaction Log файла.
  - CONTINUE_AFTER_ERROR – архивиране на Transaction Log файл, след което трябва да започне възстановяване от backup верига файлове, в която Tail-log BACKUP ще е последен.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/ms179314.aspx

https://technet.microsoft.com/en-us/library/dd297499%28v=sql.105%29.aspx

# Модул 24: Възстановяване на бази данни в MS SQL Server

**Описание на процесите по възстановяване.**

**Типове на възстановяване (RESTORE).**

- Пълно възстановяване на база данни при модел Simple recovery.
- Пълно възстановяване на база данни при модел Full recovery.
- Възстановяване на системните база данни.
- Възстановяване на единични файлове.

**Предварителна подготовка.**

- Създаване на tail-log backup ако е необходим.
- Идентифициране на необходимия за възстановяване backup.
  - Last Full, File или Filegroup backup.
  - Log backup при full и bulk-logged recovery моделите.

**Фази на Restore процеса.**

- Има 3 фази:
  - Data Copy – при която се копират данните.
  - Redo – при която се записват потвърдените транзакции.
  - Undo - при която се отказват незавършените транзакции.
- Redo и Undo често се определят като един общ процес се наричат Recovery.

Допълнителна информация може да се намери на:
https://msdn.microsoft.com/en-us/library/ms191253(v=sql.110).aspx

**WITH RECOVERY опция при възстановяване.**

- WITH RECOVERY (по подразбиране)
- WITH NORECOVERY.
- Restore процесът трябва да се изпълнява с една от следните опции:
  - WITH NORECOVERY за всички архиви без последния.
  - WITH RECOVERY за последния архив.

**Възстановяване на база данни.**

Команда RESTORE DATABASE

Ред на възстановяване:

- FULL
- Последен differential ако е приложимо)
- Transaction logs (ако е приложимо)

```
RESTORE DATABASE AdventureWorks
FROM DISK = 'D:\SQLBackups\AW.bak'
WITH RECOVERY;
```

**Възстановяване на Transaction log.**

- Команда RESTORE Log Backups.
- Може да има няколко log архива за възстановяване.
- Възстановяват се последователно в определен ред.

```
RESTORE LOG AdventureWorks
FROM DISK = 'D:\SQLBackups\AWLogs.bak'
WITH NORECOVERY;
```

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/library/ms188312(v=sql.110).aspx

**Възстановяване на данни в определен момент от време.**

- Може да се дефинира като:
  - ✓ Дата и час
  - ✓ Име на транзакция
- Базата трябва да е в модел FULL recovery.
- Опции за възстановяване STOPAT и STOPATMARK.

**Възстановяване на системни бази.**

- Master
  - ✓ Изисква архивиране.
  - ✓ Recovery Model: Simple
  - ✓ Възстановяване при Single User Mode
- Model
  - ✓ Изисква архивиране.
  - ✓ Recovery Model: Simple или Full
- Msdb
  - ✓ Изисква архивиране.
  - ✓ Recovery Model: Simple
  - ✓ Tempdb – не изисква архивиране.

**Възстановяване на данни от индивидуални файлове – процедура.**

1. Създаване на tail-log backup
2. Възстановяване на повредения файл от архив.
3. Възстановяване на differential file backup (ако е приложимо).
4. Възстановяване от transaction logs (ако е приложимо).
5. Recover на базата.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/library/ms179451.aspx

https://msdn.microsoft.com/en-us/ms190190.aspx

# Модул 25: Поддръжка на MS SQL Server

**Проверка на интегритета на база данни в MS SQL Server.**

**DBCC CHECKDB.**

- DBCC CHECKDB е команда, която се използва за проверка на логически и физически интегритет на базата данни в MS SQL Server.
  - Разпределение на страниците с данни в базата.
  - Консистентност на таблици и индекси.
  - Консистентност на каталога на базата.
- Стартира се в online режим.
- Препоръчително е да се стартира често.
- Опции на командата DBCC CHECKDB.
  - PHYSICAL_ONLY – за проверка само на физическия интегритет.
  - NOINDEX – не се проверяват индексни структури.
  - EXTENDED_LOGICAL_CHECKS – към основните проверки се добавят и проверки на индексите на специални обекти като индексни изгледи(View) и XML индекси.
  - TABLOCK – при изпълнение на командата се прави заключване на цели таблици.
  - ALL_ERRORMSGS – при наличие на грешки се визуализират всички съобщения, а не както е по поразбиране само на първите 200.
  - NO_INFOMSGS – визуализират се само съобщенията за грешка ако има такива.
  - ESTIMATEONLY – прави се оценка на използваното пространство в TEMPDB базата, което е необходимо за изпълнение на командата.

- Командата DBCC CHECKDB предоставя и опции за възстановяване при повреди на базата.
  - Базата трябва да е в SINGLE_USER режим. Режим при който не се допускат потребителски сесии.
  - REPAIR_REBUILD – за възстановяване без загуба на данни ако това е възможно
  - REPAIR_ALLOW_DATA_LOSS – ако не е възможно друго се прави опит за възстановяване с реална загуба на данни.
  - При повреди в базата за предпочитане е възстановяване от архив.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/library/ms176064(v=sql.110).aspx

**Използване на Индекси.**

**Table Scan** – при липса на индексни структури, търсенето в дадена таблица се извършва като се претърсва цялата таблица от начало до край.

**Използване на индекс** – индексите се използват за да може да се ускори търсенето. Това става чрез изграждането на индексни структури.

**B-Tree индекс** – MS SQL Server използван този вид индекс.

**Типове индекси:**

- Clustered index
- Nonclustered index

Допълнителна информация може да се намери на:

https://technet.microsoft.com/en-us/library/ms175049%28v=sql.110%29.aspx

**Фрагментация на индекс.**

- Причини за фрагментиране – SQL Server реорганизира индексните страници при промяна на данните в тях.
- Типове фрагментация:
  - Internal – няма добра запълваемост на страниците с данни.
  - External – няма последователност между страниците с данни в индексната структура.

**Параметри дефиниращи свободно пространство в страниците на индекса.**

- FILLFACTOR (leaf level) – параметър, който определя каква да е запълваемостта на страниците с данни на последното ниво(leaf level) в индексната структура.
- PAD_INDEX (intermediate & root levels) – параметър, който определя каква да е запълваемостта на страниците с в първото ниво(root) и в междинните нива(intermediate) на индексната структура.
- **Пример**:

```
ALTER TABLE HR.Employees
 ADD CONSTRAINT PK_Emp_Employee_ID
 PRIMARY KEY CLUSTERED ( Employee_ID ASC)
WITH (PAD_INDEX  = OFF, FILLFACTOR = 70);
```

**Поддръжка на индекс.**

- Команда Rebuild
  - Създава наново индекса.
  - Операцията се нуждае от свободно място в базата.
  - Голям обем в transaction log
- Команда Reorganize
  - Сортира страниците.
  - По-малък обем в transaction log
  - Пример за използване на Reorganize за индексите в една таблица:

```
ALTER INDEX ALL ON dbo.Employees REORGANIZE ;
```

**Операции свързани с индекси.**

- Create, Rebuild, Drop могат да се извършват online.
- Disable – за временно спиране на индекс.
- Използване на статистики.
  - AUTO_UPDATE_STATISTICS
  - UPDATE STATISTICS
  - sp_updatestats
- Пример за операции свързани с индекси. Използване на Rebuild

```
ALTER INDEX ALL ON Production.Product
REBUILD WITH (FILLFACTOR = 80, SORT_IN_TEMPDB = ON,
        STATISTICS_NORECOMPUTE = ON);
```

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/library/ms189858.aspx

**Автоматизиране на дейности с Maintenance plan.**

- Служи за организиране на ежедневни задачи по поддръжка на базата данни.
- Използва механизма на SQL Server Agent.
- Използва SSIS за изпълнението на задачи.

**Наблюдение на Database Maintenance планове.**

- За наблюдение кога и как(успешно/неуспешно)са се изпълнявали Database Maintenance планове може да се използва Job Activity Monitor – програмен интерфейс достъпен чрез SQL Server Management Studio.
- Резултатите се съхраняват в msdb, текстов файл или се изпращат на дефиниран оператор.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/ms191002.aspx

# Модул 26: Автоматизиране управлението на MS SQL Server

**Автоматизиране управлението – общи положения.**

- Автоматизиране на регулярни задачи.
- Изпълнение на задачите по график.
- Наблюдение на СУБД.
- Откриване и въздействие на възникнали проблеми.

**Средства за автоматизиране на управлението в MS SQL Server.**

- Използване на механизъм за управление на обекти от тип Job, при който се конфигурират задачи за изпълнение по зададен график или в следствие на настъпило събитие..
- Alert – сървърен обект, който следи за възникване на евентуално събитие.
- Operator – сървърен обект описващ към кого да се изпращат съобщения от Job и Alert.

**Въведение в SQL Server Agent.**

- Компонент на SQL Server, който отговаря за автоматизацията.
- Служи за управление на обекти като:
  - Job
  - Alert
  - Operator
- Работи като услуга(Windows service) в операционната система.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/ms189237.aspx

**Управление на задачи.**

- Job – описание на последователност от задачи.
- Различни типове – всеки обект Job може да се конфигурира да изпълнява задачи от различни типове като:
  - T-SQL(Transact SQL) команди
  - Command line скриптове
  - PowerShell скриптове
- Конфигурират се с помощта на SSMS(SQL Server Mamagement Studio).

**Операции с job.**

- При създаване на Job се конфигурират една или повече стъпки на изпълнение.
- За всяка стъпка може да се укаже действие при успех или неуспех.
- График за изпълнение на Job – кога да бъде изпълнен дадения обект Job.
- Scripting на Job - SSMS(SQL Server Mamagement Studio) поддържа функционалност за генериране на скрипт за създаване на дефиниран вече обект Job.

**История на изпълнение на job.**

- Историята за изпълнение на даден Job се съхранява в системната база msdb.
- Историята на изпълнение може да се види с помощта на SSMS.
- Job Activity monitor – позволява да се преглеждат действащите обекти Job.
- Историята има период на съхранение, който може да се конфигурира.
- Историята може да се види и чрез заявки към системни таблици.

**Диагностика на неизпълнен job.**

- SQL Server Agent не е стартиран.
- Преглед историята на изпълнение.
- Даденият Job е заключен.
- Свързаните обекти са недостъпни.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/library/ms187880(v=sql.110).aspx

**Конфигуриране на сигурността на SQL Server Agent.**

- Всеки SQL Server Agent job трябва да е стартиран в определен контекст от гледна точка на сигурност. Задачите се изпълняват от определен потребителски акаунт, който да има възможност за достъп до необходимите за изпълнение ресурси
- Достъп до обекти в SQL Server – трябва да се провери дали използвания потребителски акаунт има достъп до нужните обекти.
- Достъп до ресурси извън SQL Server– трябва да се провери дали използвания потребителски акаунт има достъп до нужните ресурси.
- Мрежови достъп - – трябва да се провери дали използвания потребителски акаунт има достъп до нужните мрежови ресурси.
- Proxy Account – обект в MS SQL сървър, с чиято помощ се променя контекста на изпълнение от гледна точка на потребителски акаунт.

**Превключване на  контекста при изпълнение на стъпки в job.**

- Стъпки от тип T-SQL
  - SQL Agent действа от името на собственика на съответния job.
  - Ако собственика е член на sysadmin роля се използва SQL Agent service акаунт.
  - Членовете на sysadmin ролята могат да указват друг акаунт.
- Други типове стъпки.
  - Членовете на sysadmin ролята могат да ползват SQL Agent service акаунт.
  - Може да бъде използван Proxy Account.

**Диагностика на неизпълнен job при проблеми със сигурността.**

- Проверка дали даденият Job не  е стартиран.
- Преглед историята на изпълнение за използван акаунт.
- Проверка на позволенията на ползвания акаунт.

Допълнителна информация може да се намери на:

https://technet.microsoft.com/en-us/library/ms190926%28v=sql.110%29.aspx

**Използване на Credentials и Proxy Accounts.**

**Credential -** SQL Server обект съхраняващ информация за автентификация.

- Ползват се за достъп до външни ресурси.
- Паролата се съхранява в криптирана форма.
- Създават се през SSMS или T-SQL.
- Съхраняват се в master базата.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/library/ms190703(v=sql.110).aspx

**Proxy Account -** акаунт, с чиято помощ SQL Server Agent достъпва външни ресурси.

- Създават се през SSMS или dbo.sp_add_proxy процедура от msdb.
- Дефинират се за специфична SQL Server подсистема.
- Стъпките в даден job се асоциират с някоя от подсистемите.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-US/library/ms175834(v=sql.110).aspx

# Модул 27: Мониторинг на MS SQL Server

**Database Mail -** имплементация на стандартен SMTP протокол.

- Позволява компоненти в базата данни да изпращат e-mail.
- Има наличен Database Mail Configuration Wizard.
- Могат да се конфигурират различни профили.

**Database Mail профили.**

- Дефинират се един или повече e-mail акаунти.
- Профил по подразбиране – използва се в случаите когато няма дефиниран акаунт.
- Типове профили:
  - Private – достъпен е само за определени потребители. Те се описват при конфигуриране на профила.
  - Public – за общо ползване от потребителите.

**Конфигуриране сигурността на Database Mail.**

- Работи в контекста на SQL Server Engine service акаунта.
- Ползва съхранени процедури, които по подразбиране са заключени. За да може да бъде използван Database Mail е необходимо да бъдат отключени, което става при стартиране конфигурацията на Database Mail.
- Роля DatabaseMailUserRole в msdb – само членовете на тази роля могат да ползват Database Mail за изпращане на съобщения.
- Забрана за изпращане на специфични файлови разширения.
- Private профилите са ограничени за ползване от конкретни потребители и роли.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/ms189635.aspx

## Конфигуриране на Operators и Alerts.

### SQL Server Agent Operator.

- SQL Server Agent Operator – обект в MS SQL Server, който дефинира субект или група, които могат да получават съобщения от job или alert. За всеки оператор може да се дефинира график на ползване.
- Създаване на operator:
  - T-SQL
  - SSMS
- Fail-safe operator – дефиниран оператор, който се ползва извън графика на останалите дефинирани оператори..
- Използване на operator – може да бъде ползван за изпращане на съобщения от обекти Job и Alert..

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/library/ms179336(v=sql.110).aspx

### SQL Server Alert.

- Обект в MS SQL Server с конфигурация за следене на настъпил събитие и на предварително дефиниран отговор на това събитие.
- Могат да бъдат активирани от:
  - SQL Server събития
  - WMI(Windows management instrumentarium) събития
- Могат да:
  - Изпратят съобщение на дефиниран operator.
  - Стартират обект Job.

**Създаване на SQL Server Alert**

- Чрез SSMS или системната процедура sp_add_alert.
- Дефинира се начин на прихващане на събитие.
- Описват се действията, които да бъдат предприети.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/ms180982

https://technet.microsoft.com/en-us/library/ms187827%28v=sql.110%29.aspx

**Мониторинг на производителността на MS SQL Server.**

- Средства за наблюдение:
  - DMO
  - Activity Monitor
  - Performance Monitor
  - SQL Server Profiler
  - Extended Events Profiler

**DMO.**

- Dynamic Management Objects – views и функции предоставящи информация за състоянието на SQL Server.
- Организирани са по категории:
  - sys.dm_exec_% - информация свързана с изпълнение на заявки.
  - sys.dm_os_% - информация свързана с операционната система.
  - sys.dm_tran_% - информация свързана с изпълнение на транзакции.
  - sys.dm_io_% - информация свързана с входно-изходни операции
  - sys.dm_db_% - информация свързана с базите данни.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/library/ms188754.aspx

**Activity Monitor.**

- Компонент на SSMS.
- Дава информация за SQL Server процеси, изчаквания, I/O операции и заявки.
- Използва DMO.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/hh212951.aspx

**Windows Performance Monitor.**

- Ползва се, за събиране и преглед на системни метрики. Елемент на операционната система Windows
- Мониторинг в реално време.
- Области на мониторинг:
  - ✓ CPU
  - ✓ Памет
  - ✓ Дискови системи
  - ✓ SQL Server

**Data Collector.**

- Компонент на SQL Server.
- Служи за събиране на данни за състояние чрез DMO с намалено натоварване на SQL Server.
- Съхранява данни в определен период.
- Съдържа предварително дефинирани отчети.
- Позволява централизирано съхранение на данни за няколко инстанции на SQL Server.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/bb677356.aspx

**SQL Server Profiler.**

- Инструмент за изследване дейностите в SQL Server.
- Използва SQL Trace  програмен интерфейс с помощта на системни процедури.
- Може да се ползва при различни сценарии:
  - Debug – изследване на грешки.
  - Monitoring – наблюдение на работоспособността на SQL Server
  - Позволява съхранение на резултатите във файл или таблица.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/ms181091

# Модул 28: Импортиране и експортиране на данни в MS SQL Server

**Трансфер на данни – общи положения.**

При трансфера на данни имаме три основни етапи, които се представят, чрез абревиатурата ETL – Extract(извличане), Transform(Трансформиране), Load(Зареждане).

**Средства за импорт и експорт на данни в MS SQL Server.**

- Bulk Copy Program (bcp) – програма за импорт и експорт от команден ред.
- BULK INSERT – Transact SQL клауза.
- OPENROWSET(BULK) – функция в Transact SQL
- Import / Export Wizard – програмен интерфейс от тип съветник с предварително дефинирани стъпки за импорт и експорт.
- XML Bulk Load - програмен интерфейс за зареждане на XML данни.

**Трансфериране на големи обеми данни. Подобряване производителността при трансфер на данни.**

- Изключване на ограничения (constraints), индекси и тригери.
- Минимизиране на заключвания.
- Минимизиране на операциите с transaction log.
- Минимизиране конвертирането на данни.

**SQL Server Integration Services (SSIS) -** подпомага реализирането на ETL решения.

- SSIS пакети – описание и конфигурация на ETL. Могат да се стартират със следните програми и обекти:
  - Dtexec – програма, чрез която могат да се стартират SSIS пакети.
  - Dtexecui - програма, чрез която могат да се стартират SSIS пакети.
  - SQL Server Agent jobs - обект, чрез който могат да се стартират SSIS пакети.
- SSIS пакети могат да се създават с:
  - SQL Server Business Intelligence Development Studio – име на средата за разработка на програмни решения в Microsoft SQL Server
  - Import / Export Wizard – екранен съветник за импортиране/експортиране на данни.

**Bcp.**

- Command line инструмент за импорт и експорт на данни.
- Ползва форматиращ файл.
- Създаване на форматиращ файл:

  *bcp Adv.Sales.Currency format nul -T -c –x -f Cur.xml*

- Експорт на данни във файл.

  *bcp Adv.Sales.Currency out Cur.dat -T –c*

- Импорт на данни с форматиращ файл.

  *bcp tempdb.Sales.Currency2 in Cur.dat -T -f Cur.xml*

**BULK INSERT клауза.**

```
BULK INSERT AdventureWorks.Sales.OrderDetail
  FROM 'f:\orders\neworders.txt'
  WITH
    (
      FIELDTERMINATOR ='|',
      ROWTERMINATOR ='|\n'
    );
GO
```

**OPENROWSET функция.**

```
SELECT *
   FROM OPENROWSET(BULK 'c:\mssql\export.csv',
        FORMATFILE = 'c:\mssql\format.fmt',
        FIRSTROW = 2) AS a;
GO
```

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/ms141026.aspx

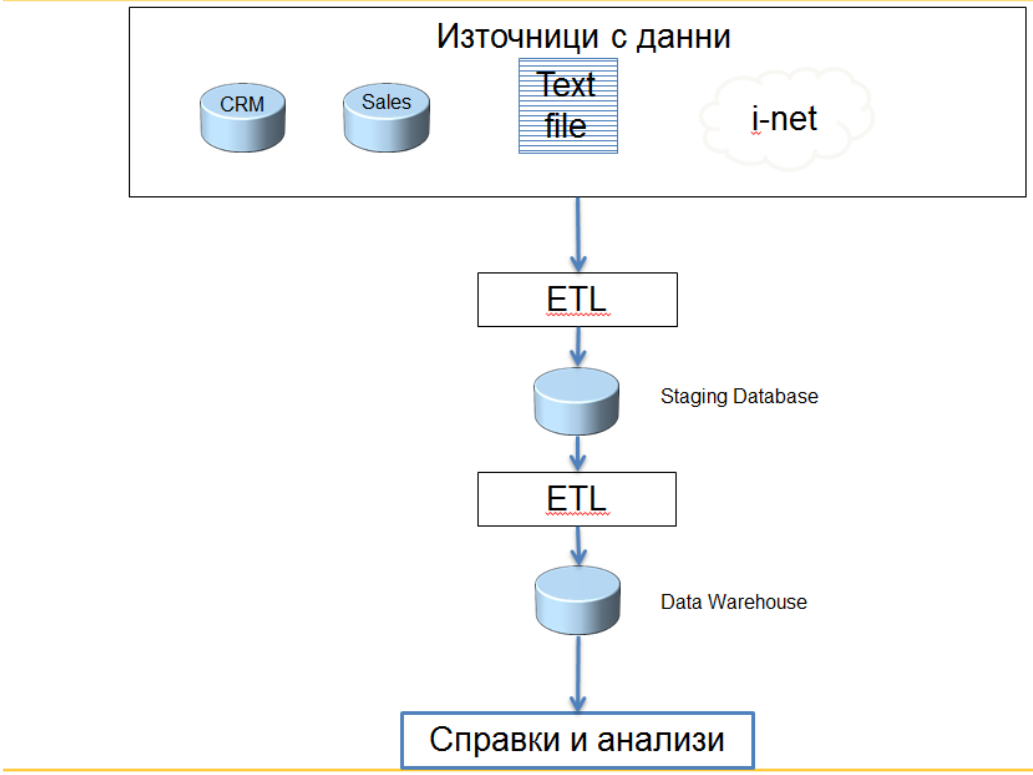
https://technet.microsoft.com/en-us/library/ms175937%28v=sql.110%29.aspx

# Модул 29: Data Warehouse решения

**Общи положения и предпоставки.**

- Data Warehouse - централизирана база, в която се обединяват данни от различни системи за отчет и анализ.
- Съдържа голям обем исторически данни.
- Оптимизирано решение за отправяне на заявки.
- Данните се въвеждат периодично.
- Денормализация на данните.
- Работи се с голям обем редове.
- Данните са статични.
- База за BI проекти.

**Компоненти на Data Warehouse.**



**Видове архитектури и решения.**

- Centralized Data Warehouse – архитектура при която, данните се събират на едно централизирано място от където потребителите от различни звена черпят информация.
- Departmental Data Mart – при тази архитектура за всяко звено има изградена база с данните необходими за конкретното звено.
- Hub and Spoke – данните се събират в единна база след, което се разпределят по базите на отделните звена.

**ETL процеси.**

- Staging – място където се определя:
  - ✓ Кои са данни?
  - ✓ Какъв е формата на данните?
- Необходими ли са трансформации на данните?
- Инкрементален ETL:
  - ✓ Идентификация на данните за трансфер.
  - ✓ Добавяне или обновяване на данните?

**Data Quality и Master Data Management.**

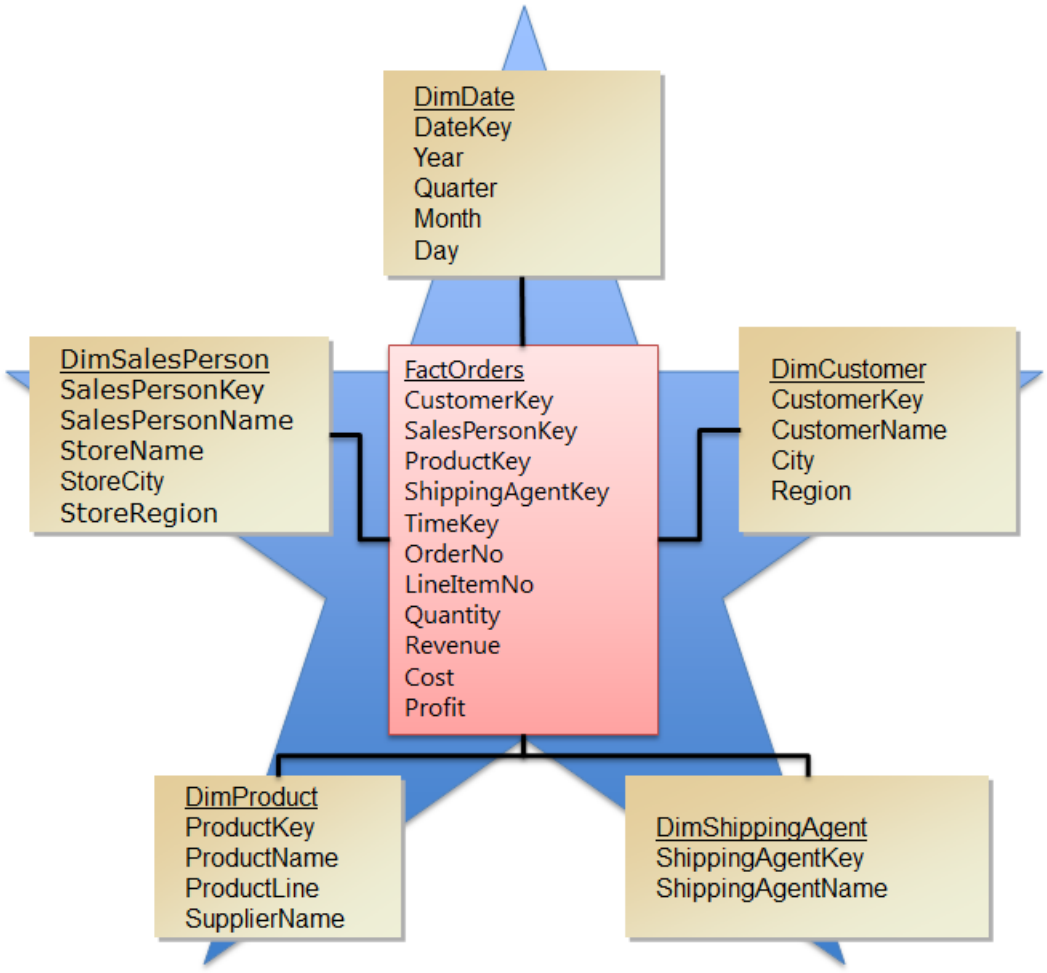
- Data Quality :
  - ✓ Валидиране на данни.
  - ✓ Валидиране на консистентност.
  - ✓ Идентифициране на липсващи ст-сти.
  - ✓ Премахване на дублирания.
- Master Data Management:
  - ✓ Удостоверяване на еднакви данни в различни системи.
  - ✓ Налагане на правила, на които трябва да отговарят данните?

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/ff877917.aspx
https://msdn.microsoft.com/en-us/ee633763.aspx

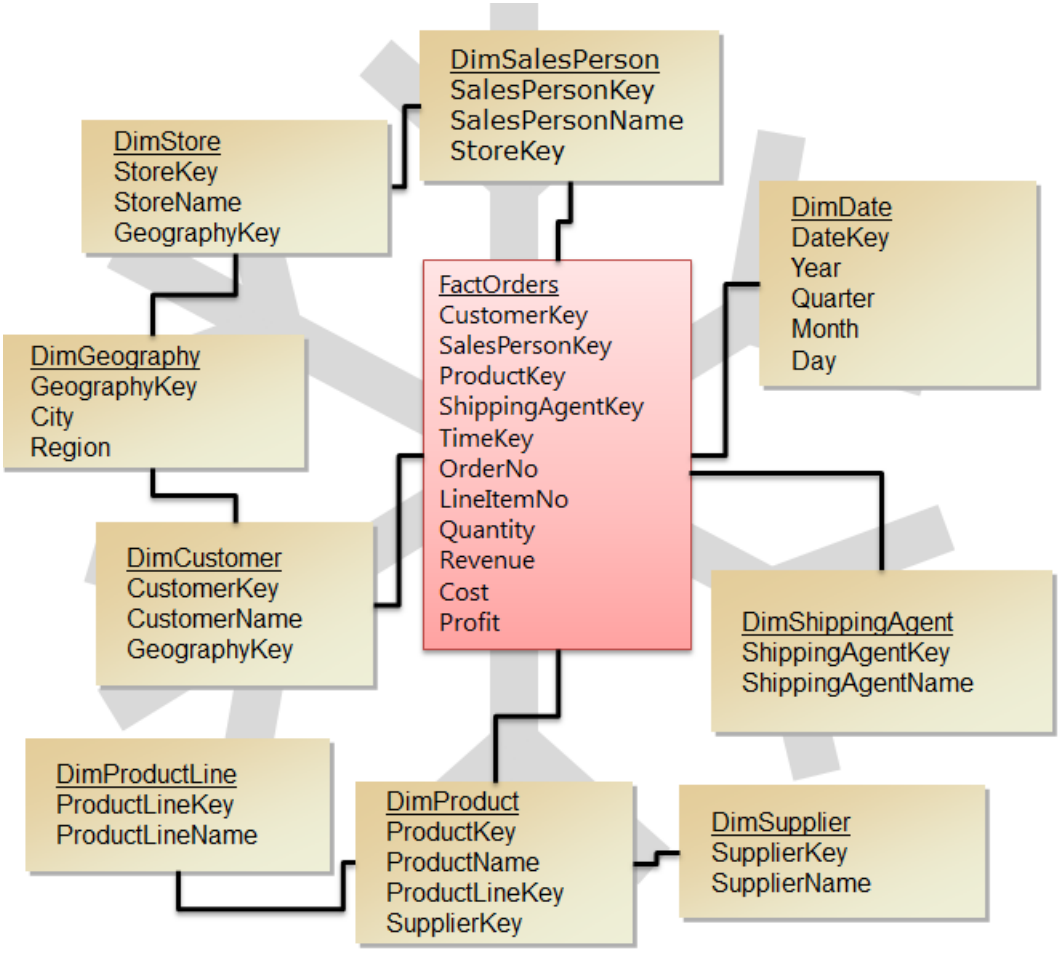
**Dimensional Modeling – моделиране в различни измерения.**

- Фокусът е върху обобщаване на данни по различни измерения.
- Примери за измерения:
  - Продажби за тримесечие. Измерението е тримесечие.
  - Печалба по продуктова категория. Измерението е категория.
  - Общ обем доставки за регион. Измерението е регион.
- Видове таблици:
  - Dimension таблици – описват по какво ще се правят справки.
  - Fact таблици – данни-факти за справките.
- Видове схеми:
  - Star – схема звезда. При тази схема една Fact таблица е свързана с няколко Dimention таблици чрез ключови полета.




  - Snowflake – схема снежинка. При тази схема При тази схема една Fact таблица е свързана с няколко Dimention таблици чрез ключови

полета. В допълнение Dimention таблиците са свързани с допълнителни таблици даващи детайли, които могат да бъдат споделяни от поне две Dimention таблици.




**Общи съображения относно Dimension таблиците.**

- Данните са денормализирани.
- Ключови полета – с цел запазване на история за промяна на данните се предприема:
  - Създаване на нови системни ключове.
  - Запазване на оригиналните ключове.

**Общи съображения относно Fact таблиците.**

- Ниво на подробност:
  - До най-малки подробности спрямо всички измерения.
  - Ако са нужни различни нива се създават няколко Fact таблици.
- Ключови полета.
  - Първичният ключ обикновено е съставен от външните ключове на включените измерения.

**Измерение за време –** в много случаи е удобно да се създаде и използва таблица съдържаща като първичен ключ дати и допълнителни полета описващи компонентите на тези дати като ден, месец, година, тримесечие, ден от седмицата, седмица на годината и други.

**Инструменти и способи използвани в Data Warehouse решения.**

- ETL инструменти
  - SQL Server Integration Services (SSIS) – описание е дадено в модул 28.
  - The Import and Export Data Wizard– описание е дадено в модул 28
  - Използване на Transact-SQL клаузата BULKINSERT и функцията OPENROWSET – описание е дадено в модул 28.
  - bcp– описание е дадено в модул 28.
  - Replication – репликация на бази данни. Механизъм, който уеднаквява структурата и данните на две бази.
- Business intelligence(BI) инструменти – инструменти улесняващи генерирането на справки и анализи чрез използване на данни от дадена СУБД.
  - Microsoft Excel – програма за електронни таблици.
  - SQL Server Analysis Services – програма за анализ на данни
  - SQL Reporting Services – програма за генериране на отчети.
  - Power View – Уеб базирано приложение за генериране на отчети.

**Предварително изследване на данни**

- Необходимо е за:
  - ✓ Интерпретиране на кодове и стойности.
  - ✓ Определяне на типа и размера на данните.
  - ✓ Откриване на проблеми с качеството на данните.
- Взимане на извадки.
- Изследване на данните може да се направи с.
  - ✓ Microsoft Excel.
  - ✓ Data Profiling – компонент в SSIS предназначен за анализ на данни..

**Въведение в SSIS(SQL Server Integration Services)** - платформа за ETL операции, която предоставя различни компоненти за използване в продукта Data Tools.

- Control flow – основен компонент за пакетиране на подкомпоненти.
- Data flow – компонент реализиращ изтеглянето на данни от даден източник, трансформирането на данни и зареждането им в приемник на база.
- Гореописаните компоненти могат да се реализират в Data Tools като проект (SSIS project)
- SSIS package – обединява един или повече SSIS проекта в обща обект, който може да бъде стартиран от обект Job в MS SQL Server.

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/ms141026.aspx

# Модул 30: Анализ и обработка на данни

**Общи положения.**

- Необходимост от анализ и обработка на данни – основно предназначение на системите за управление на бази данни е поддържането на данни с цел извличане на информация за анализи и справки.
- Инструменти за анализ и обработка на данни.
  - MS Excel
  - Analysis services
  - Reporting services
  - Power Pivot
  - Power View

Допълнителна информация може да се намери на:

https://msdn.microsoft.com/en-us/bb522607.aspx

https://msdn.microsoft.com/en-us/ms159106.aspx

https://www.microsoft.com/en-us/server-cloud/solutions/business-intelligence/

**Microsoft Excel като средство за анализ на данни.**

- Таблично представяне на данни – електронни таблици.
- Широк набор от инструменти и функции за анализ и обработка на данни.
- Връзка към системи за управление на бази данни.

**Свързване на Microsoft Excel със системи за управление на бази данни.**

- Импортиране на данни:
  - Текстови файлове
  - MS SQL Server
  - Oracle
  - MS Access
- Създаване на заявки към база данни.

Допълнителна информация може да се намери на:

https://support.office.com/en-us/article/Connect-a-SQL-Server-database-to-your-workbook-22c39d8d-5b60-4d7e-9d4b-ce6680d43bad

**Инструменти и функции в Excel за анализ и обработка на данни.**

- Инструменти:
  - Условно форматиране

    *Home -> Styles -> Conditional Formatting…*
  - Data validation

    *Data -> Data Tools-> Data Validation…*
  - Text to columns

    *Data -> Data Tools-> Text to columns*
  - Remove duplicates
  - *Data -> Data Tools-> Remove duplicates*

- Функции.
  - Статистически – MAX, MIN, COUNT, AVERAGE
  - Логически – IF, OR, AND
  - Функции за търсене – VLOOKUP, HLOOKUP
  - Функции за дата и час – MONTH, YEAR, WORKDAY, NETWORKDAY

Допълнителна информация може да се намери на:

http://blog.newhorizons.bg/%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D1%8F/%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D1%8F/microsoft/office/

# Списък с полезни препратки

Microsoft:

https://msdn.microsoft.com/en-us/library/bb545450.aspx

https://www.microsoft.com/en-us/server-cloud/products/sql-server/

Oracle:

http://www.oracle.com/technetwork/database/enterprise-edition/documentation/index.html

https://asktom.oracle.com

https://www.oracle.com/corporate/press/index.html

IBM:

http://www.ibm.com/developerworks/

New Horizons:

http://blog.newhorizons.bg/