

# Некоторые термины

- БД – совокупность данных, которые хранятся по определённым правилам и используются для удовлетворения информационных потребностей пользователей.
- Реляционная БД – это БД, где вся информация представляется в виде таблиц

students

id	name	avg_grade	faculty_id
1	Mike	8,2	1
2	John	9,1	3
3	Sara	7,5	1
4	Tim	6,8	2
5	Maria		

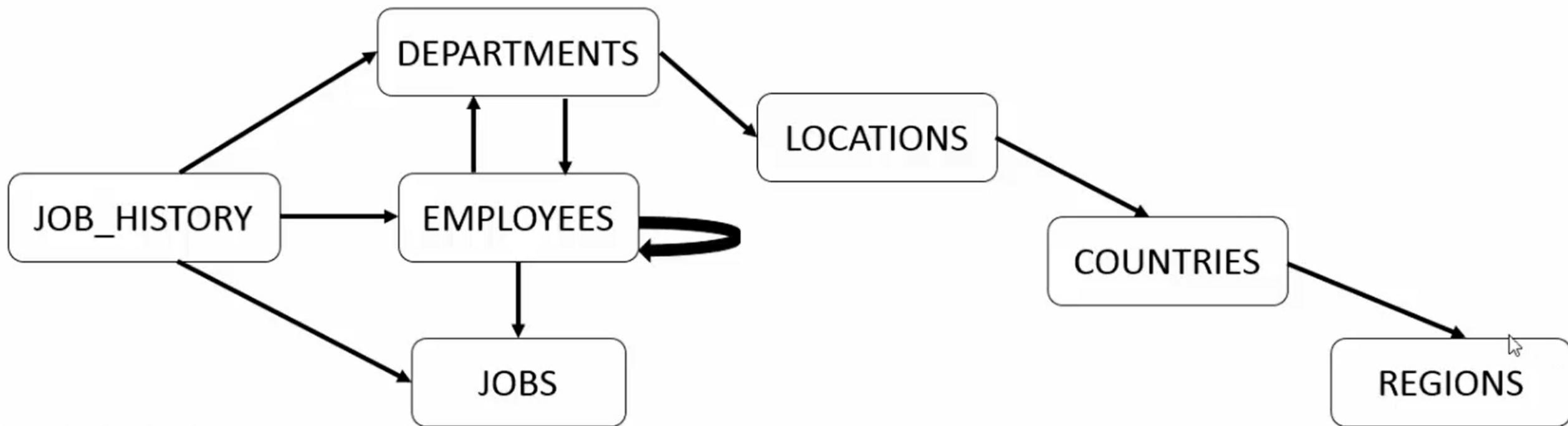
faculty

id	name	dean
1	Economics	James
2	CS	Amy
3	Philology	Luke

# Знакомство с HR схемой

User – это лицо, которое может подключиться к БД (log on process).

DB схема – это все объекты в БД, которые принадлежат одному user-у.



# SQL команды

DML

SELECT

INSERT

UPDATE

DELETE

MERGE

DDL

CREATE

ALTER

DROP

RENAME

TRUNCATE

TCL

COMMIT

ROLLBACK

SAVEPOINT

DCL

GRANT

REVOKE

# Типы данных (data types)

Тип данных – это множество допустимых значений этих данных, а также совокупность операций над ними.

INTEGER

5

0

-200

NUMBER(p, s)

NUMBER(7, 2)

12345.67

123456.7

Где p – precision (общее максимальное количество цифр), s – scale (сколько из общего количества цифр отводится на дробную часть). Максимальное количество цифр на целую часть числа = p – s

# Типы данных (data types)

CHAR(length)

CHAR(8)

privet

Privet, drug

Фиксированная длина

VARCHAR2(size)

VARCHAR2(8)

privet

Privet, drug

Переменная длина

# Типы данных (data types)

DATE

11-Sep-19 23:17:18

Содержит информацию о году, месяце, дне, часе, минуте, секунде.

TIMESTAMP(f)

TIMESTAMP(6)

11-Sep-19 23:17:18.123000



Содержит информацию о году, месяце, дне, часе, минуте, секунде,олях секунды.

# Понятие NULL

NULL – это отсутствие данных

0 или символ пробел – это не NULL. Они занимают место в памяти.

Результат арифметических операций с NULL – это всегда NULL.



# Команда DESCRIBE

```
DESCRIBE SCHEMA.TABLE_NAME;
```

```
DESCRIBE hr.regions;
```

```
DESC hr.regions;
```

```
DESCRIBE regions;
```

```
DESC regions;
```

# **SELECT statement (НЕ меняет данные)**

## **Basic syntax**

`SELECT * FROM table;`

`SELECT * FROM countries;`

`SELECT column(s) FROM table;`

`SELECT country_name FROM countries;`

`SELECT country_id, country_name FROM countries;`

# **SELECT statement (НЕ меняет данные)**

## **DISTINCT**

**SELECT DISTINCT column(s) FROM table;**

**SELECT DISTINCT job\_id FROM job\_history;**

**SELECT DISTINCT job\_id, department\_id  
FROM job\_history;**

# **SELECT statement (НЕ меняет данные)**

## **Expressions in select list**

```
SELECT column(s), expression(s) FROM table;
```

```
SELECT salary*2 FROM employees;
```

```
SELECT first_name, salary, salary*1.5 FROM employees;
```

```
SELECT start_date, end_date, (end_date-start_date)+1  
FROM job_history;
```



# **SELECT statement (НЕ меняет данные)**

## **Alias**

```
SELECT column(s) alias, expression(s) alias FROM table;
```

Alias – это альтернативное имя для столбца или целого выражения

```
SELECT salary*2 AS bonus FROM employees;
```

```
SELECT start_date start, end_date end, (end_date-  
start_date)+1 "Count Of Days" FROM job_history;
```

```
SELECT 'First name is '||first_name||' and last_name is '  
|| last_name our_employees FROM employees;
```

# **SELECT statement (НЕ меняет данные)**

## **Таблица DUAL**

```
SELECT 'privet, |||dorogoy student' AS greeting  
FROM dual;
```

```
SELECT 32*365*24*60*60 AS "moy vozrast v sekundax"  
FROM dual;
```

# SELECT statement (НЕ меняет данные)

Проблемы с одинарными кавычками в тексте

```
SELECT 'It''s my life' AS song FROM dual;
```

Quote (q) operator:

q ' delimiter наш текст с кавычками delimiter '

```
SELECT q'<It's my life>' AS song FROM dual;
```

## **SELECT statement (НЕ меняет данные)**

### **Итог первого знакомства с SELECT**

```
SELECT * | {DISTINCT column(s) alias, expression(s) alias}  
FROM table;
```

# SELECT

## Концепция SELECTION

```
SELECT * | {DISTINCT column(s) alias, expression(s) alias}  
FROM table  
WHERE condition(s);
```

```
SELECT * FROM employees WHERE salary = 4800;
```

```
SELECT first_name, salary FROM employees  
WHERE last_name = 'King';
```

```
SELECT email FROM employees  
WHERE hire_date = '21.09.05';
```



# SELECT

## Операторы сравнения

=

>

<

>=

<=

!=

<>

BETWEEN

IN

IS NULL

LIKE

%

\_

# SELECT

## Логические операторы

Всё выражение принимает значение TRUE тогда и только тогда, когда все условия, объединённые **AND**, по отдельности тоже возвращают TRUE.

Всё выражение принимает значение TRUE тогда и только тогда, когда хотя бы одно из условий, объединённых **OR**, по отдельности тоже возвращает TRUE.

# SELECT

## Логические операторы

Всё выражение принимает значение TRUE тогда и только тогда, когда все условия, объединённые **AND**, по отдельности тоже возвращают TRUE.

Всё выражение принимает значение TRUE тогда и только тогда, когда хотя бы одно из условий, объединённых **OR**, по отдельности тоже возвращает TRUE.

Оператор **NOT** меняет значение условия на противоположное.

# SELECT

## Последовательность выполнения операторов

ПРИОРИТЕТНОСТЬ	ОПЕРАТОР
1	( )
2	/ * - +
3	- +
4	
5	= > < >= <=
6	[NOT] LIKE, IS [NOT] NULL, [NOT] IN
7	[NOT] BETWEEN
8	!= <>
9	NOT
10	AND
11	OR



# SELECT

## ORDER BY

```
SELECT * | {DISTINCT column(s) alias, expression(s) alias}  
FROM table  
WHERE condition(s)  
ORDER BY {col(s) | expr(s) | numeric position}  
{ ASC | DESC } { NULLS FIRST | LAST };
```

```
SELECT first_name, salary FROM employees  
ORDER BY salary;
```

```
SELECT first_name, commission_pct FROM employees  
ORDER BY commission_pct DESC NULLS LAST;
```



# ФУНКЦИИ



Single-row



Multiple-row

## Single-row функции



Character



Numeric



Date



Conversion



General



# Character functions

## Case conversion functions



- LOWER(s)
- UPPER(s)
- INITCAP(s)

Где s – это строка, текст.

# Character functions

## Character manipulations functions

- CONCAT(s, s)
- LENGTH(s)
- LPAD(s, n, p)
- RPAD(s, n, p)
- TRIM({trailing | leading | both} trimstring from s)

Где s – это строка, текст; n – конечная длина текста; p – текст для заполнения; trimstring – текст, который надо срезать;

# Character functions

## Character manipulations functions

- `INSTR(s, search string, start position, Nth occurrence)`
- `SUBSTR(s, start position, number of characters)`
- `REPLACE(s, search item, replacement item)`

Где s – это строка, текст; search string – искомый текст; start position – позиция для начала работы; Nth occurrence – N-ое появление; number of characters – количество символов; search item – искомый элемент; replacement item – заменяющий элемент



# Numemric functions

- `ROUND(n, precision)`
- `TRUNC(n, precision)`
- `MOD(dividend, divisor)`

Где n – это число; precision – точность; dividend - делимое; divisor - делитель.

# Date functions

- SYSDATE
- MONTHS\_BETWEEN(start\_date, end\_date)
- ADD\_MONTHS(date, number\_of\_months)

Где start\_date – дата «с»; end\_date – дата «по»; date – дата; number\_of\_months - количество месяцев.

# Date functions

- NEXT\_DAY(date, day\_of\_the\_week)
- LAST\_DAY(date)



Где date – дата; day of the week - день недели.

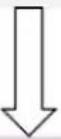
# Date functions

- `ROUND(date, date precision format)`
- `TRUNC(date, date precision format)`

Где `date` – дата; `date precision format` – точность округления;

Век – CC; год – YYYY; четверть – Q; месяц – MM;  
неделя – W; день – DD; час – HH; минута – MI.

# Функции



Single-row



Multiple-row

## Single-row функции



Character



Numeric



Date



Conversion



General

# TO\_CHAR (number to char)

`TO_CHAR(number, format mask, nls_parameters) = T`

Конвертация числа в текст, используя функцию TO\_CHAR означает взять число и сделать из него текст в том виде, в каком указан наш формат, если такой имеется.

# TO\_CHAR (number to char)

Элемент	Описание	Формат	Число	Текст
9	Ширина	99999	18	18
0	Отображение нуля	099999	18	000018
.	Позиция десятичной точки	099999.999	18.35	000018.350
D	Позиция десятичного разделителя	099999D999	18.35	000018.350
,	Позиция запятой	099,999,999	1234567	001,234,567
G	Позиция разделителя групп	099999G999	1234567	001,234,567
\$	Знак \$	\$099999	18	\$000018
L	Локальная валюта	L099999	18	\$000018
MI	Позиция знака -	099999MI	-18	000018-
PR	Скобки для отриц. чисел	099999PR	18	<000018>
S	Префикс + или -	S099999	18	+000018



# TO\_CHAR (date to char)

Дата для примера: '20-SEP-19'

Элемент	Описание	Текст
Y	Последняя цифра года	9 I
YY	Последние 2 цифры года	19
YYY	Последние 3 цифры года	019
YYYY	Год целиком	2019
RR	Год в формате 2-х цифр	19
YEAR	Буквенное написание года (Case-sensitive)	TWENTY NINETEEN
MM	Месяц в формате 2-х цифр	09
MON	3 буквы из названия месяца (Case-sensitive)	SEP
MONTH	Буквенное написание месяца (Case-sensitive)	SEPTEMBER

# TO\_CHAR (date to char)

Дата для примера: '20-SEP-19'

Элемент	Описание	Текст
D	День недели	6
DD	День месяца 2 цифры	20
DDD	День года	263
DAY	3 буквы из назв. дня недели (Case-sensitive)	FRI
DAY	Полное название дня недели (Case-sensitive)	FRIDAY
W	Неделя месяца	3
WW	Неделя года	38
Q	Квартал года	3
CC	Век	21

# TO\_CHAR (date to char)

Дата для примера: '20-SEP-19 16:17:18'

Элемент	Описание	Текст
AM, PM, A.M. и P.M.	Индикатор	PM
HH, <sub>hh</sub> 12 и HH24	Формат времени	04, 04, 16
MI	Минуты	17
SS	Секунды	18
SSSS	Секунды после полуночи	58638
- / . , ? # !	Пунктуация: 'MM.YY'	09.19
"Любой текст"	"Quarter" Q "of" Year'	Quarter 3 of Twenty Nineteen
TH	'DDth "of" Month'	20TH of September
SP	Буквенное написание(spell) 'MmSP Month Yyyysp'	Nine September Two Thousand Nineteen
THSP или SPTH	Комбинация: 'hh24SpTh'	sixteenth

# TO\_DATE (char to date)

`TO_DATE(text, format mask, nls_parameters) = D`

Конвертация текста в дату, используя функцию TO\_DATE означает взять текст и объяснить в своём формате, где и как содержится информация о элементах даты в вашем тексте.

# TO\_NUMBER (number to date)

`TO_NUMBER(text, format mask, nls_parameters) = N`

Конвертация текста в число, используя функцию TO\_NUMBER означает взять текст и объяснить в своём формате, где и как содержится информация о элементах числа в вашем тексте.



# General functions

- NVL(value, ifnull)
- NVL2(value, ifnotnull, ifnull) I
- NULLIF(value1, value2)
- COALESCE(value1, value2, ... , valueN)

# Conditional functions

- **DECODE(expr, comp1, iftrue1, comp2, iftrue2, ..., compN, iftrueN, iffalse)**

- CASE

```
graph TD; CASE[• CASE] --> SimpleCASE[Simple CASE]; CASE --> SearchedCASE[Searched CASE]
```

Simple CASE

Searched CASE

# Conditional function CASE

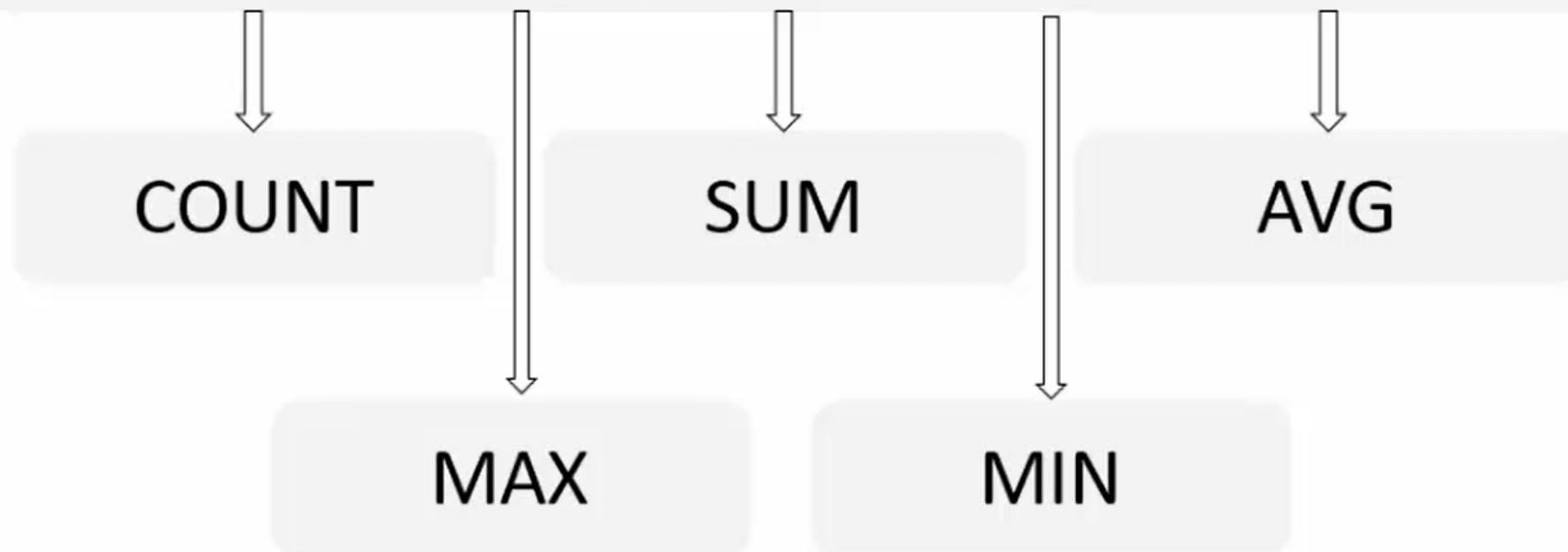
simple

```
CASE expr  
WHEN comp1 THEN iftrue1  
WHEN comp2 THEN iftrue2  
.....  
WHEN compN THEN iftrueN  
ELSE ifffalse  
END
```

searched

```
CASE  
WHEN cond1 THEN iftrue1  
WHEN cond2 THEN iftrue2  
.....  
WHEN condN THEN iftrueN  
ELSE ifffalse  
END
```

# Group functions



# Group functions

COUNT( { \* | {DISTINCT|ALL}expression } )

SUM( {DISTINCT|ALL} expression )

AVG( {DISTINCT|ALL} expression )

MIN( {DISTINCT|ALL} expression )

MAX( {DISTINCT|ALL} expression )

# GROUP BY

```
SELECT * | {DISTINCT column(s) alias, expression(s)
alias, group_f-on(s)(col|expr alias), }
FROM table
WHERE condition(s)
GROUP BY {col(s)|expr(s)}
ORDER BY {col(s)|expr(s)|numeric position}
{ ASC|DESC } { NULLS FIRST|LAST };
```

# GROUP BY WITH HAVING

```
SELECT * | {DISTINCT column(s) alias, expression(s)
alias, group_f-on(s)(col|expr alias), }
FROM table
WHERE condition(s)
GROUP BY {col(s)|expr(s)}
HAVING group_condition(s)
ORDER BY {col(s)|expr(s)|numeric position}
{ ASC|DESC } { NULLS FIRST|LAST };
```

# JOIN



INNER JOIN  
(NATURAL JOIN)



OUTER JOIN



CROSS JOIN

# JOIN



EQUIJOIN



NONEQUIJOIN

ORACLE JOIN



# INNER JOIN

(NATURAL JOIN)



NATURAL  
JOIN

USING

ON

# NATURAL JOIN

```
SELECT column(s)  
FROM table_1  
NATURAL JOIN  
table_2;
```

```
SELECT *  
FROM regions  
NATURAL JOIN  
countries;
```

```
SELECT region_name,  
c.country_name, c.region_id  
FROM regions  
NATURAL JOIN  
countries c;
```

# NATURAL JOIN with USING

```
SELECT column(s)
  FROM table_1
    JOIN
      table_2
    USING (column(s));
```

```
SELECT *
  FROM regions
    JOIN
      countries
    USING (region_id);
```

```
SELECT e.department_id,
        manager_id
  FROM employees e
    JOIN departments d
    USING (department_id);
```

# NATURAL JOIN with ON

SELECT column(s)

FROM table\_1

JOIN

table\_2

ON (column1 = column2);

```
SELECT *  
FROM regions r  
JOIN  
countries c  
ON(r.region_id=c.region_id);
```

```
SELECT region_id  
FROM regions r  
JOIN  
countries c  
ON(r.region_id=c.region_id);
```

# NONEQUIJOIN with ON

```
SELECT column(s)
      FROM table_1
            JOIN
              table_2
      ON (column1 {оператор неравенства} column2);
```

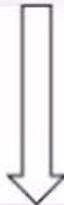
# OUTER JOIN



LEFT OUTER  
JOIN



RIGHT OUTER  
JOIN



FULL OUTER  
JOIN

# OUTER JOIN

```
SELECT column(s) FROM  
table_1 LEFT OUTER JOIN table_2  
ON (column1 = column2);
```

```
SELECT column(s) FROM  
table_1 RIGHT OUTER JOIN table_2  
ON (column1 = column2);
```

```
SELECT column(s) FROM  
table_1 FULL OUTER JOIN table_2  
ON (column1 = column2);
```

# CROSS JOIN

```
SELECT column(s)  
FROM table_1  
CROSS JOIN  
table_2;
```

```
SELECT *  
FROM regions  
CROSS JOIN  
countries;
```

```
SELECT department_id,  
FROM employees e  
CROSS JOIN  
departments d;
```

# **TYPES OF SUBQUERIES**



**SINGLE ROW**



**MULTIPLE ROW**



**SCALAR**

# SINGLE ROW SUBQUERIES

```
SELECT first_name, last_name, salary  
      FROM employees  
     WHERE salary <  
          (SELECT MAX(salary)/5 FROM employees);
```

```
SELECT salary FROM employees  
     WHERE salary >=  
          (SELECT first_name, salary FROM employees  
             WHERE employee_id = 180);
```

```
SELECT salary FROM employees  
     WHERE salary =  
          (SELECT salary FROM employees  
             WHERE employee_id > 180);
```



# MULTIPLE ROW SUBQUERIES

```
SELECT salary FROM employees  
      WHERE salary IN  
(SELECT salary FROM employees  
      WHERE employee_id > 180);
```

```
SELECT first_name, last_name, salary FROM employees  
WHERE salary > ANY (SELECT salary FROM employees  
      WHERE department_id = 100);
```

```
SELECT first_name, last_name, salary FROM employees  
WHERE salary > ANY (SELECT salary, first_name FROM  
      employees WHERE department_id = 100);
```

# CORRELATED SUBQUERIES

```
SELECT e1.first_name, e1.last_name, e1.salary  
      FROM employees e1  
     WHERE e1.salary >  
          (SELECT avg(e2.salary) FROM employees e2  
           WHERE e2.department_id=e1.department_id);
```

# SET OPERATORS



UNION



UNION ALL



INTERSECT



MINUS



# UNION ALL

UNION ALL объединяет 2 аутпут множества в одно простым присоединением.

```
SELECT * FROM jobs  
WHERE job_id LIKE '%MAN%'  
      UNION ALL  
      SELECT * FROM jobs  
      WHERE job_id LIKE '%MAN%'
```

# UNION

UNION объединяет 2 аутпут множества в одно, удаляя при этом дубликаты и сортируя его.

```
SELECT * FROM jobs  
WHERE job_id LIKE '%MAN%'  
UNION  
SELECT * FROM jobs  
WHERE job_id LIKE '%MAN%'
```

# INTERSECT

INTERSECT 2-ух аутпут множеств возвращает только общие строки, удаляя при этом дубликаты и сортируя результат.

```
SELECT * FROM jobs  
WHERE job_id LIKE '%MAN%'  
INTERSECT  
SELECT * FROM jobs  
WHERE job_id LIKE '%MAN%'
```

# MINUS

MINUS 2-ух аутпут множеств возвращает только те строки, которые есть в первом множестве, но нет во втором, удаляя при этом дубликаты и сортируя результат.

```
SELECT * FROM jobs  
WHERE job_id LIKE '%MAN%'  
MINUS  
SELECT * FROM jobs  
WHERE job_id LIKE '%MAN%'
```

# DML COMMANDS



SELECT



INSERT



UPDATE



DELETE



MERGE

# INSERT

```
INSERT INTO table_name  
    column(s)  
VALUES (value(s));
```

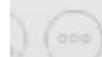
```
INSERT INTO table_name  
    column(s)  
SUBQUERY;
```



# UPDATE

```
UPDATE table_name  
      SET  
    column(s) = value(s)  
 WHERE condition(s);
```

```
UPDATE table_name  
      SET  
    column(s) = subquery(s)  
 WHERE column = subquery;
```



# **DELETE**

```
DELETE  
FROM table_name  
WHERE condition(s);
```

```
DELETE  
FROM table_name  
WHERE column = subquery;
```



# MERGE

```
MERGE INTO table_name1 t1
USING {table_name2|subquery} t2
ON (t1.column = t2.column)
    WHEN MATCHED THEN
        UPDATE SET column=value
        DELETE WHERE condition
    WHEN NOT MATCHED THEN
        INSERT(value1, value2)
VALUES (column1, column2);
```



# ACID

ATOMICITY - АТОМАРНОСТЬ

CONSISTENCY - СОГЛАСОВАННОСТЬ

ISOLATION - ИЗОЛИРОВАННОСТЬ

DURABLE - ДОЛГОВЕЧНОСТЬ

# TCL COMMANDS



COMMIT



ROLLBACK



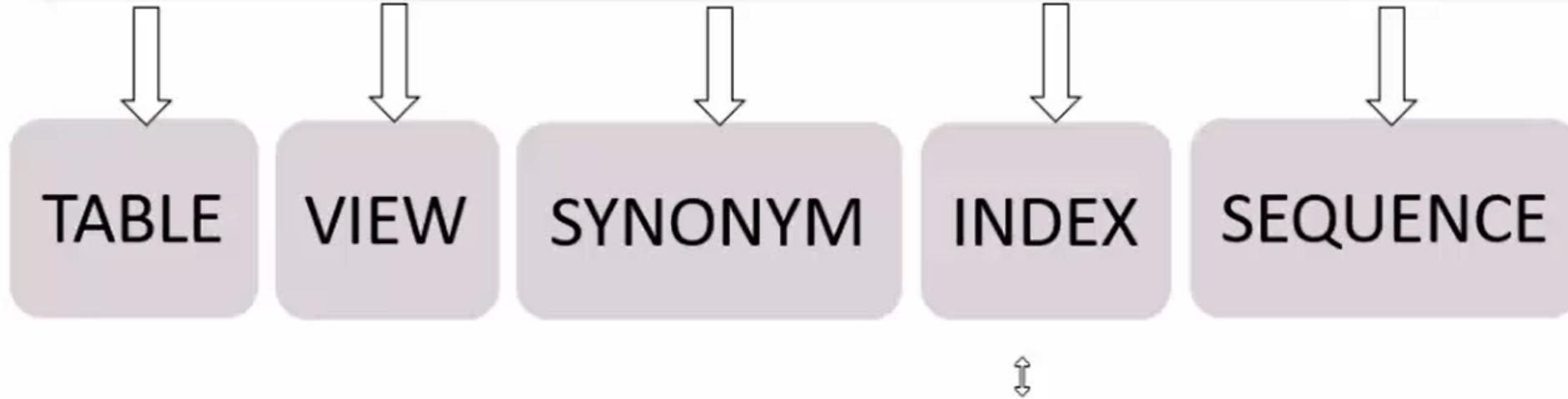
SAVEPOINT

ROLLBACK TO SAVEPOINT savepoint\_name;



SAVEPOINT savepoint\_name;

# DB OBJECTS



# ПРАВИЛА НАЗВАНИЯ ОБЪЕКТОВ

Длина названия должна быть от 1 до 30 символов

Нельзя использовать зарезервированные слова

Названия должны начинаться с букв

Помимо букв в названии можно использовать цифры и символы «\_», «#», «\$»

Прописные буквы будут автоматически конвертированы в заглавные

# NAMESPACE

TABLE

SEQUENCE

PRIVATE  
SYNONYM

VIEW

INDEX

CONSTRAINT

PUBLIC  
SYNONYM

# ДРУГИЕ ТИПЫ ДАННЫХ

TIMESTAMP WITH TIMEZONE

TIMESTAMP WITH LOCAL TIMEZONE

INTERVAL YEAR TO MONTH

INTERVAL DAY TO SECOND

CLOB

BLOB

LONG

ROWID

# TABLE CREATION

```
CREATE TABLE schema.table ORGANIZATION HEAP  
  (column_name datatype DEFAULT expr,  
   column_name datatype DEFAULT expr,  
    ...);
```

```
CREATE TABLE schema.table  
  AS  
  subquery;
```

# ALTERING TABLE

```
ALTER TABLE table_name  
ADD (column_name data_type DEFAULT expr);
```

```
ALTER TABLE table_name  
MODIFY (column_name data_type DEFAULT expr);
```

```
ALTER TABLE table_name  
DROP COLUMN column_name;
```

```
ALTER TABLE table_name  
SET UNUSED COLUMN column_name;  
ALTER TABLE table_name DROP UNUSED COLUMNS;
```

```
ALTER TABLE table_name  
RENAME COLUMN column_name1 TO column_name2;
```

```
ALTER TABLE table_name READ ONLY;
```

# TRUNCATE

```
TRUNCATE TABLE  
schema.table_name;
```



# DROP

```
DROP TABLE  
schema.table_name;
```



# CONSTRAINT TYPES

UNIQUE принуждает столбец(цы) содержать только уникальные значения. Исключение – null.

NOT NULL не разрешает столбцам содержать значение “null”

PRIMARY KEY принуждает столбец(цы) содержать только уникальные значения и не разрешает содержать значение “null”

FOREIGN KEY принуждает использовать только значения из определённого столбца таблицы-родителя или значение “null”. 

CHECK принуждает использовать только значения, которые удовлетворяют его условию(ям)

# ТИПЫ ИНДЕКСОВ



B-TREE



BITMAP

```
CREATE { UNIQUE | BITMAP } INDEX  
schema.index_name ON  
schema.table_name (column1, column2, ...);
```

```
DROP INDEX schema.index_name;
```

# ТИПЫ VIEW



SIMPLE

- One table
- No functions
- No aggregation



COMPLEX

- Join tables
- Functions
- Aggregation



# VIEW

```
CREATE OR REPLACE {FORCE | NOFORCE} VIEW  
schema.view_name (alias1, alias2, ...)  
AS subquery  
WITH CHECK OPTION {CONSTRAINT constraint_name}  
WITH READ ONLY {CONSTRAINT constraint_name};
```

```
ALTER VIEW schema.view_name COMPILE;
```

```
DROP VIEW schema.view_name;
```

# SYNONYM

```
CREATE PUBLIC SYNONYM synonym_name  
FOR object_name;
```

```
ALTER PUBLIC SYNONYM synonym_name  
COMPILE;
```

```
DROP PUBLIC SYNONYM synonym_name;
```

# SEQUENCE

```
CREATE SEQUENCE schema.sequence_name
INCREMENT BY number
START WITH number
{MAXVALUE number | NOMAXVALUE}
{MINVALUE number | NOMINVALUE}
{CYCLE | NOCYCLE}
{CACHE number | NOCACHE};
```

# SEQUENCE

```
ALTER SEQUENCE schema.sequence_name  
INCREMENT BY number  
{MAXVALUE number | NOMAXVALUE}  
{MINVALUE number | NOMINVALUE}  
{CYCLE | NOCYCLE}  
{CACHE number | NOCACHE};
```

```
DROP SEQUENCE schema.sequence_name;
```