# Лабораторная работ №3

## Нормализация баз данных

Нормализация - процесс организации структуры базы данных, направленный на устранение избыточности и несогласованности данных.

Для процесса нормализации необходимо использование терминов реляционной алгебры: `первичный ключ` и `внешний ключ`.

`Первичный ключ` ( `Primary Key` ) - это уникальный и неизменяемый идентификатор, однозначно определяющий строку таблицы (записи).

`Внешний ключ` ( `Foreign Key` ) - это атрибут, ссылающийся на `первичный ключ` или на уникальный атрибут другой таблицы.

Процесс нормализация предполагает соблюдение определенного набора правил. Наборы таких правил принято называть `нормальными формами`.

Существует несколько видов нормальных форм:

- `1NF` - первая нормальная форма;
- `2NF` - вторая нормальная форма;
- `3NF` - третья нормальная форма;
- `BCNF` - нормальная форма Бойса-Кодда;
- `4NF` - четвертая нормальная форма;
- `5NF` - пятая нормальная форма;
- `DKNF` - доменно-ключевая нормальная форма;
- `6NF` - шестая нормальная форма (6NF).

На практике применяется нормализация до `3NF`. Более глубокая нормализация усложняет архитектуру базы данных и обычно не дает дополнительных преимуществ.

## Первая нормальная форма

Правило `1NF`: атрибуты таблицы не должны содержать повторяющиеся значений и групп атрибутов, описывающих похожие данные.

Рекомендации:

- добавить дополнительный атрибут для повторяющегося значения;
- удалить из таблицы те атрибуты, которые используются для хранения похожих данных.

Пример. Существует таблица описывающая `товар`:

- `название`;
- `дата поставки`;
- `производитель_1`;
- `производитель_2`;
- `производитель_3`.

Столбцы `производитель_${n}` - содержать похожие данные. Для нормализации таблицы необходимо ввести дополнительную таблицу `производитель`:

- `название`.

Далее необходимо изменить таблицу `товар`:

- `название`;
- `дата поставки`;
- `название производителя`.

## Вторая нормальная форма

Правило: отношение находится в `2NF` только в том случае, если оно находится в `1NF` и атрибут, который не является ключевым (не ключ) зависит от первичного ключа.

Рекомендации:

- определить атрибуты являющиеся первичным ключом;
- определить атрибуты являющиеся внешними ключами;
- определить зависимые атрибуты, которые не являются ключевыми;

Пример. Рассмотрим две таблицы `производитель` и `товар`

`Производитель`

- `название`
- `адрес`

`Товар`:

- `название`
- `дата поставки`
- `название производителя`

Атрибуты являющиеся первичным ключом:

- `Производитель` - `название`;
- `Товар` - *???* - необходимо добавить атрибут являющийся первичным ключом.

Атрибуты являющиеся внешними ключами:

- `Производитель` - отсутствуют;
- `Товар` - `название производителя`.

Атрибуты не являющиеся ключевыми:

- `Производитель` - `адрес`;
- `Товар` - `название`, `дата поставки`;

Итог:
`Производитель`:

- `название` - `первичный ключ`
- `адрес`

`Товар`:

- `идентификатор` - `первичный ключ`
- `название`
- `дата поставки`
- `название производителя` - `внешний ключ`

## Третья нормальная форма

Правило: отношение находится в `3NF` только в том случае, если оно находится во второй нормальной форме ( `2NF` ) и в отношении не существует неключевых атрибутов не зависящих от первичного ключа.

Рекомендации:

- вынести все неключевые атрибуты в отдельную таблицу.

Пример. Рассмотрим таблицу `товар`:

- `идентификатор` - `первичный ключ`
- `название`
- `дата поставки`
- `название производителя` - `внешний ключ`

`Дата поставки` не является ключевым атрибутом и не зависит от ключа.

Решение: создадим таблицы `заказ`, `заказ содержит` и удалим атрибут `дата поставки` из таблицы `товар`:

`Заказ`

- `идентификатор` - `первичный ключ`
- `дата создания`
- `дата поставки`

`Заказа содержит`

- `идентификатор` - `первичный ключ`;
- `идентификатор заказа` - `внешний ключ` из таблицы `заказ`;
- `идентификатор товара` - `внешний ключ` из таблицы `товар`;

# SQL

## Некоторые функции SQL

### SUM

Рассчитывает сумму значений атрибута:

```
SELECT SUM(${ATTR}) FROM ${TABLE_NAME};
```

### AVG

Рассчитывает среднее значение атрибута:

```
SELECT AVG(${ATTR}) FROM ${TABLE_NAME};
```

### COUNT

Подсчитывает количество записей таблицы:

```
SELECT AVG(${ATTR}) FROM ${TABLE_NAME};
```

### MIN

Находит минимальное значение атрибута:

```
SELECT MIN(${ATTR}) FROM ${TABLE_NAME};
```

### MAX

Находит максимальное значение атрибута:

```
SELECT MAX(${ATTR}) FROM ${TABLE_NAME};
```

## Выборка данных

В лабораторной роботе №2 было рассмотрено базовое применение запросов `SELECT`:

```
SELECT * | ${ATTR1}, ${ATTR2}, ${ATTR3} as ${ALIAS}, ... FROM
${TABLE_NAME} [WHERE ${CONDITION}];
```

Данный тип запросом может содержать дополнительные операторы `SQL`:

- `LIMIT`
- `IN`
- `BETWEEN`
- `ORDER BY`
- `GROUP BY`
- `INNER JOIN`
- `LEFT JOIN`
- `RIGHT JOIN`
- `FULL JOIN`
- `CROSS JOIN`
- `UNION`

## LIMIT

`LIMIT` записывается в конце `SELECT` и позволяет вывести определенное количество строк:

```
SELECT * FROM ${TABLE_NAME} LIMIT ${N}; -- выводит первые N -
записей

SELECT * FROM ${TABLE_NAME} LIMIT ${N} OFFSET ${S}; -- выводит
N записей начиная с S
```

## IS, IN, BETWEEN

`IS`, `IN` и `BETWEEN` используется в секции `WHERE` для фильтрации записей таблицы.

```
SELECT * FROM ${TABLE_NAME} WHERE ${ATTR_2} IS [NOT] NULL; --
проверяет значение атрибута на равенство `NULL`
```

```
SELECT * FROM ${TABLE_NAME} WHERE ${ATTR_2} IN (${VAL_1},
${VAL_2}); -- выводит все записи для которых ATTR_2 == VAL_1
или ATTR_2 == VAL_2

SELECT * FROM ${TABLE_NAME} WHERE ${ATTR_2} = ${VAL_1} OR
${ATT_2} = ${VAL_2}; -- эквивалентное выражение
```

```
SELECT * FROM ${TABLE_NAME} WHERE ${ATTR_1} BETWEEN ${B} AND
${E}; -- эквивалентно

SELECT * FROM ${TABLE_NAME} WHERE ${ATTR_1} > ${B} AND
${ATTR_1} < ${E};



SELECT * FROM ${TABLE_NAME} WHERE ${ATTR_1} NOT BETWEEN ${B}
AND ${E}; -- эквивалентно

SELECT * FROM ${TABLE_NAME} WHERE ${ATTR_1} < ${B} AND
${ATTR_1} > ${E};
```

## AS

Данный оператор используется для именования или переименования столбцов/таблиц в результатах выборки.

```
SELECT ${ALIAS_TABLE}.${ATTR_1} AS SYS_NAME FROM ${TABLE_NAME}
```

```
AS ${ALIAS_TABLE}
```

## ORDER BY

Оператор `ORDER BY` выполняет сортировку результатов выборки.
Можно отсортировать результаты:

- `ASC` - по возрастанию (по умолчанию);
- `DESC` - по убыванию.

```
SELECT * FROM ${TABLE_NAME} ORDER BY ${COLUMN_NAME} [ASC |
DESC];
```

## GROUP BY

Оператор `GROUP BY` используется для объединения результатов выборки.

```
SELECT $FUNC($ATTR) FROM ${TABLE_NAME} GROUP BY
${COLUMN_NAME};
```

## INNER JOIN

Оператор `INNER JOIN` позволяет объединить строки нескольких таблиц по заданному условию
При это, каждая строка первой таблицы (расположена слева), соединяется с каждой строкой из второй таблицы (справа от `INNER JOIN`). Далее производится проверка условия: если условие истинно, то результат объединения попадает в выборку, в противно случае отбрасывается.

```
SELECT * FROM ${TABLE_NAME_1} INNER JOIN ${TABLE_NAME_2} ON
${CONDITION};
```

Результат работы оператор `INNER JOIN` похож на результат операции пересечения множеств.

$$L = \{\}; R = \{\};$$

$$IJ \to L \cap R;$$

## LEFT JOIN

Оператор `LEFT JOIN` также как и `INNER JOIN` позволяет объединить строки нескольких таблиц в одну но по другому правилу: для данного оператора важен порядок следования таблиц, таблица слева получает приоритет:

- вначале происходит объединение таблиц по принципу `INNER JOIN`;
- затем для оставшихся строк из левой таблицы подставляются значения `NULL` в поля правой таблицы.

```
SELECT * FROM ${TABLE_NAME_1} LEFT JOIN ${TABLE_NAME_2} ON
${CONDITION};
```

$$L = \{\}; R = \{\}$$

$$LJ \to L;$$

## RIGHT JOIN

Оператор `RIGHT JOIN` похож на оператор `LEFT JOIN`, но приоритет получает таблица справа также как и `INNER JOIN`:

- вначале происходит объединение таблиц по принципу `INNER JOIN`;
- затем для оставшихся строк из правой таблицы подставляются значения `NULL` в поля левой таблицы.

```
SELECT * FROM ${TABLE_NAME_1} RIGHT JOIN ${TABLE_NAME_2} ON
${CONDITION};
```

$$L = \{\}; R = \{\};$$

$$RJ \to R;$$

## FULL JOIN

Оператор `FULL JOIN` объединяет строки таблиц, но для него не важен порядок следования таблиц (нет приоритета). Данный оператор является

симметричным:

- сначала выполняется `INNER JOIN`;
- затем выполняется `LEFT JOIN`;
- затем выполняется `RIGHT JOIN`.

```
SELECT * FROM ${TABLE_NAME_1} FULL JOIN ${TABLE_NAME_2} ON
${CONDITION};
```

$$L = \{\}; R = \{\};$$
$$CJ \to L \cup R;$$

## CROSS JOIN

Данный оператор образует декартово произведение строк таблиц

```
SELECT * FROM ${TABLE_NAME_1} CROSS JOIN ${TABLE_NAME_2};
```

$$L = \{\}; R = \{\};$$
$$CJ \to L \times R;$$

## UNION

`UNION` объединяет таблицы друг с другом (аналог объединения множеств):

```
SELECT * FROM ${TABLE_NAME_2}
UNION
SELECT * FROM ${TABLE_NAME_2}; -- объединение с удаление
дубликатов

SELECT * FROM ${TABLE_NAME_1}
UNION ALL
SELECT * FROM ${TABLE_NAME_2}; -- объединение без удаления
дубликатов
```

# Задание

## Задание 1

Нормализовать базу данных `university` до `3NF`.

## Задание 2

Небольшая компания состоит из 4 отделов:

- отдел продаж;
- отдел рекламы;
- бухгалтерия;
- ИТ отдел.

В отделе продаж работает 4 сотрудника:

- руководитель отдела;
- старший специалист;
- 2 специалиста.

В отделе рекламы 3 сотрудника: руководитель отдела и 2 специалиста.

В бухгалтерии: главный бухгалтер и 2 бухгалтера.

В ИТ отделе: руководитель отдела и 2 системных администратора.

Также в компании работает генеральный директор и финансовый директор.

1. Необходимо спроектировать базу данных, которая учитывает:
    - сотрудников;
    - отделы;
    - заработную плату сотрудников.
2. Базу необходимо заполнить данными. Каждый сотрудник должен иметь данные о начислении заработной платы за год (12 месяцев).
3. **[*]** Помимо заработной платы, некоторые сотрудники компании получают премиальные выплаты в конце года (предусмотреть код выплат).
4. Вывести среднюю заработную плату по отделам.

5. Вывести среднюю заработную плату руководителей отделов.
6. Вести минимальную и максимальную заработную плату в компании.
7. **[*]** Вывести сотрудников получающих премиальные выплаты.
8. **[*]** Вывести в порядке возрастания заработной платы все должности компании.