```sql
-- SQL 10: RDBMS CONCEPTS

/*
Basic Concepts of Relational Model

Knowing the basic building blocks of the Relational model such as constraints,
primary key, and foreign key,
knowing their properties, and applying them to our data model are the prerequisites
of the relational data model.

Relational Data Model Terms

The relational data model was found out by C. F. Codd in 1970. It is one of the
most common data models today.

The relational data model describes the universe as “a compound of inter-related
relations (or tables).

1. Relation
A relation, also known as a table or file, is a subset of the Cartesian product of
a list of domains represented
by a name. And within a table, each row represents a group of related data values.
A row, or record, is also known
as a tuple. The column in a table is a field and is also referred to as an
attribute. In other words, attributes
is used to define records and every record has a set of attributes.

2. Table

A database contains multiple tables and each table stores the data. Table
Properties:

- A table has a unique name.
- There are no duplicate rows.
- Cells and attributes have to be atomic.
- Entries from columns are from the same domain and they should have same data type
as following:
        - number (numeric, integer, float, smallint,…)
        - character (string)
        - date
        - logical (true or false)
- Each attribute has a distinct name.

3. Column
A relational database stores different information belonging to a domain in an
organized manner.
Being able to use and manage the database effectively requires an understanding of
this organizational method.

The basic storage units are called columns or fields or attributes. These hosts the
basic components of data
```

into which your content can be broken down. When deciding which attributes was required, we need to think about
the information data has. For example, drawing out the common components of the information that we will
store in the database and avoiding the specifics that distinguish one item from another.

4. Domain
Domain refers to the original atomic values that hold the attributes a column can have. An atomic value is no longer
divisible and expresses a structure that has a meaning on its own. For example:

The domain of Gender has a set of possibilities: Male, Female
The domain of the Month has the set of all possible months: {Jan, Feb, Mar…}.
The domain of Age is the set of all floating-point numbers greater than 0 and less than 150.

5. Relationship
Relationships are bridges that allow tables carrying information in an atomic structure to cling to each other.

6. Record
Records contain fields that are related, such as a customer or a product. As noted earlier, a tuple is another term
used for the record.

Records and fields form the basis of all databases. A simple table gives us the clearest picture of how records
and fields work together in a database storage project.

DATATYPES:

check the script number 2

KEY CONSTRAINTS

There are several types of keys, such as Candidate key, Composite key, Primary key (PK), Secondary key, Alternate
key and Foreign key (FK).

1. Candidate key
A candidate key is a simple or composite key that is unique and minimal. It is unique because no two rows in a table
may have the same value at any time. It is minimal because every column is necessary in order to attain uniqueness.

tamamen unique degerler iceren (id ve tc kim) 2 key var, bunlar candidate key olur. id'yi primary key yaparsak tc
alternate key olur.
2. Composite key
A composite key is composed of two or more attributes, but it must be minimal.

```sql
3. Primary key
The primary key (PK) is a candidate key that is selected by the database designer
to be used as an identifying mechanism
for the whole entity set. It must uniquely identify tuples in a table and not be
null. The primary key is indicated
in the ER model by underlining the attribute.

A primary key is a column or a group of columns that uniquely identifies each row
in a table. You create a primary key
for a table by using the PRIMARY KEY constraint.

Tips:
- If primary key has two or more columns, we must use the PRIMARY KEY constraint as
a table constraint.
- Each table can contain only one primary key.
- SQL Server automatically sets the NOT NULL constraint for all the primary key
columns.
- SQL Server also automatically creates a unique clustered index (or a non-
clustered index if specified as such) when
we create a primary key.

4. Secondary key
A secondary key is an attribute used strictly for retrieval purposes (can be
composite).

5. Alternate key
Alternate keys are all candidate keys not chosen as the primary key.

6. Foreign key
A foreign key (FK) is an attribute in a table that references the primary key in
another table. FK can be null. Both
foreign and primary keys must be of the same data type.

The FOREIGN KEY (FK) constraint defines a column, or combination of columns, whose
values match the PRIMARY KEY (PK) of
another table.

Values in an FK are automatically updated when the PK values in the associated
table are updated/changed.
FK constraints must reference PK or the UNIQUE constraint of another table.
The number of columns for FK must be same as PK or UNIQUE constraint.
If the WITH NOCHECK option is used, the FK constraint will not validate existing
data in a table.
No index is created on the columns that participate in an FK constraint.

For example, the field id in the employee table is a FK to the field id in the
departments table:*/

CREATE TABLE employee
(
    id BIGINT NOT NULL,
    name VARCHAR(20) NULL,
  CONSTRAINT foreignkey_1 FOREIGN KEY (id) REFERENCES dbo.department(id)
);

/*
RELATIONAL INTEGRITY CONSTRAINTS

Constraints are a very important feature in a relational model. In fact, the
relational model supports the well-defined
theory of constraints on attributes or tables.

Constraints are useful because they allow a designer to specify the semantics of
data in the database. Constraints
are the rules that force DBMSs to check that data satisfies the semantics.

1. Domain Integrity

Domain restricts the values of attributes in the relation and is a constraint of
the relational model. However, there
are real-world semantics for data that cannot be specified if used only with domain
constraints.

We need more specific ways to state what data values are or are not allowed and
which format is suitable for an
attribute. For example, the Employee ID (EID) must be unique or the employee
Birthdate is in the
range [Jan 1, 1950, Jan 1, 2000]. Such information is provided in logical
statements called integrity constraints.

2.Referential Integrity
Referential integrity requires that a foreign key must have a matching primary key
or it must be null. This constraint
is specified between two tables (parent and child); it maintains the correspondence
between rows in these tables.
It means the reference from a row in one table to another table must be valid.

3. Entity Integrity
To ensure entity integrity, it is required that every table have a primary key.
Neither the PK nor any part of it can
contain null values.

This is because null values for the primary key mean we cannot identify some rows.
For example, in the EMPLOYEE table,
the Phone cannot be a primary key since some people may not have a telephone.

4. Enterprise Constraints
Enterprise constraints – sometimes referred to as semantic constraints – are
additional rules specified by users or
database administrators and can be based on multiple tables. Here are some
examples.
```

```sql
- A class can have a maximum of 30 students.
- A teacher can teach a maximum of four classes per semester.
- An employee cannot take part in more than five projects.
- The salary of an employee cannot exceed the salary of the employee’s manager.

5. Business Rules
Business rules are obtained from users when gathering requirements. The
requirements-gathering process is very important,
and its results should be verified by the user before the database design is built.
If the business rules are incorrect,
the design will be incorrect, and ultimately the application built will not
function as expected by the users.
Some examples of business rules are:
- A teacher can teach many students.
- A class can have a maximum of 35 students.
- A course can be taught many times, but by only one instructor.
- Not all teachers teach classes.

TABLE or COLUMN CONSTRAINTS

The Optional Column Constraints are NULL, NOT NULL, UNIQUE, PRIMARY KEY and
DEFAULT, used to initialize a value
for a new record. The column constraint NULL indicates that null values are
allowed, which means that a row
can be created without a value for this column. The column constraint NOT NULL
indicates that a value must be
supplied when a new row is created.

To illustrate, we will use the SQL statement CREATE TABLE departments to create the
departments table
with 7 attributes or fields.*/

CREATE TABLE departments
(
    id BIGINT NOT NULL,
    name VARCHAR(20) NULL,
    dept_name VARCHAR(20) NULL,
    seniority VARCHAR(20) NULL,
    graduation VARCHAR(20) NULL,
    salary BIGINT NULL,
    hire_date DATE NULL,
        CONSTRAINT pk_1 PRIMARY KEY (id)
 ) ;

 /*
- The first field is id with a field type of BIGINT. The user cannot leave this
field empty (NOT NULL).
- Similarly, the second field is name with a field type VARCHAR of length 20.
- After all the table columns are defined, a table constraint, identified by the
word CONSTRAINT, is used to create
```

```sql
the primary key:
CONSTRAINT pk_1 PRIMARY KEY(id)

We can use the optional column constraint IDENTITY to provide a unique, incremental
value for that column.
Identity columns are often used with the PRIMARY KEY constraints to serve as the
unique row identifier for
the table. The IDENTITY property can be assigned to a column with a tinyint,
smallint, int, decimal, or numeric
data type. This constraint:

- Generates sequential numbers.
- Does not enforce entity integrity.
- Only one column can have the IDENTITY property.
- Must be defined as an integer, numeric or decimal data type.
- Cannot update a column with the IDENTITY property.
- Cannot contain NULL values.
- Cannot bind defaults and default constraints to the column.

For IDENTITY[(seed, increment)]
- Seed - the initial value of the identity column
- Increment - the value to add to the last increment column

The UNIQUE constraint prevents duplicate values from being entered into a column.

- Both PK and UNIQUE constraints are used to enforce entity integrity.
- Multiple UNIQUE constraints can be defined for a table.
- When a UNIQUE constraint is added to an existing table, the existing data is
always validated.
- A UNIQUE constraint can be placed on columns that accept nulls. Only one row can
be NULL.
- A UNIQUE constraint automatically creates a unique index on the selected column.

This is an examle using the UNIQUE constraint.
 */
 CREATE TABLE employee
(
id BIGINT NOT NULL UNIQUE
name VARCHAR(20) NOT NULL
);

/*
The CHECK constraint restricts values that can be entered into a table.

- It can contain search conditions similar to a WHERE clause.
- It can reference columns in the same table.
- The data validation rule for a CHECK constraint must evaluate to a boolean
expression.
- It can be defined for a column that has a rule bound to it.

This is the general syntax for the CHECK constraint:*/
```

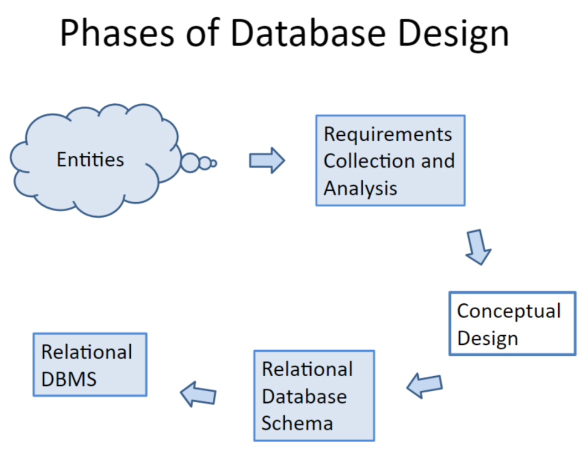
```
[CONSTRAINT constraint_name]
CHECK [NOT FOR REPLICATION] (expression)

CREATE TABLE department
(
    id BIGINT NOT NULL,
    name VARCHAR(20) NULL,
    dept_name VARCHAR(20) NULL,
    seniority VARCHAR(20) NULL,
    graduation VARCHAR(20) NULL,
    salary BIGINT NULL,
    hire_date DATE NULL,
        CONSTRAINT pk_1 PRIMARY KEY (id),
        CHECK (salary BETWEEN 40000 AND 100000)
 ) ;

 /*
The DEFAULT constraint is used to supply a value that is automatically added for a
column if the user does not supply one.

- A column can have only one DEFAULT.
- The DEFAULT constraint cannot be used on columns with a timestamp data type or
identity property.
- DEFAULT constraints are automatically bound to a column when they are created.

ANOMALİ :
 Bütün bilgiler tek bir tabloda olursa; bir bilgiyi silmek, değiştirmek veya update
etmek istediğinde
 sorun yaşarsın. Yanlış bilgiyi silebilirsin ve bir entitiyi tamamen silebilirsin.
Bunlara dikkat etmek
 gerekiyor. Bunun önüne geçmek için geliştirile geliştirile relational database
kavramı oluşmuş ve
 tüm sütunların bir tabloda toplanması ANOMALİ olarak görülmüş ve bundan
vazgeçilmiş.

Anomalilerin giderilmesi işlemine de NORMALİZASYON denilmiştir.
PRIMARY KEY: bir tabloyu tek başına identify etme yeteneğine sahip keydir.
COMPOSITE KEY: Bir tabloda primary key olarak düşündüğün bir alan tek başına o
tabloyu identify
edemiyorsa. Primary key olabilecek ikinci bir alan da tek başına o tabloyu identify
edemiyor fakat
bu iki alan beraberce tabloyu identify edebiliyorsa: bu iki keye composit key
diyoruz.
FUNCTIONAL DEPENDENT olması için o sütunun her bir değerinin dependent olduğu
sütundaki (ki bu primary keydir) yalnız bir unique değer ile tanımlanabilmesi
lazım.

Alanlar (sütunlar) çoğaldıkça mecburen tekrarlar olacaktır. Dolayısıyla tüm
alanları tek bir tabloda
```

```
topladığımızda bu tekrarlardan dolayı tablomuz şişecektir.  Değişiklik yapmak
istediğimizde bu bize
zorluklar çıkaracaktır. Bir tabloda bütün özelliklerin birden olması gereksizdir,
bu performansı da
düşürecektir.
İşte bunlara anomali diyoruz. Anomaliyi düzeltmek için normal formlar ile
normalizasyon yapıyoruz.
Entiti Relationship Diagram (ERD)'da database'in tablolarının normalize edilmiş
halini ve bunlar
arasındaki ilişkileri gösteriyoruz. Eğer ralational database oluşturacaksak bu
normalizasyon mutlaka
yapılmalıdır

The general syntax for the DEFAULT constraint is:*/

[CONSTRAINT constraint_name]
DEFAULT {constant_expression | niladic-function | NULL}
[FOR col_name]

ALTER TABLE departments
ADD CONSTRAINT def_dept_name DEFAULT 'HR' FOR dept_name;
```

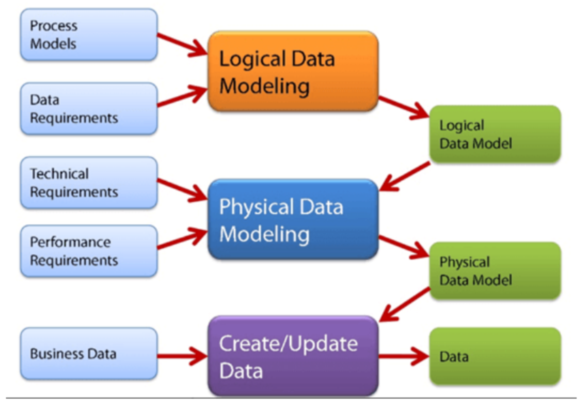
Phases of Database Design

Data modelling is the first step in the process of database design. This step is sometimes considered to be a high-level and abstract design phase, also referred to as conceptual design. The aim of this phase is to describe:

- The data contained in the database (e.g., entities: students, lecturers, courses, subjects)
- The relationships between data items (e.g., students are supervised by lecturers; lecturers teach courses)
- The constraints on data (e.g., student number has exactly eight digits; a subject has four or six units of credit only)


Phases of Database Design
Entities
Requirements Collection and Analysis
Conceptual Design
Relational Database Schema
Relational DBMS

In the second step, the data items, the relationships and the constraints are all expressed using the concepts provided by the high-level data model. Because these concepts do not include the implementation details, the result of the data modelling process is a (semi) formal representation of the database structure. This result is quite easy to understand so it is used as reference to make sure that all the user's requirements are met.



The third step is database design. During this step, we might have two sub-steps: one called database logical design, which defines a database in a data model of a specific DBMS, and another called database physical design, which defines the internal database storage structure, file organization or indexing techniques. These two sub-steps are database implementation and operations/user interfaces building steps.

In the database design phases, data are represented using a certain data model. The data model is a collection of concepts or notations for describing data, data relationships, data semantics and data constraints. Most data models also include a set of basic operations for manipulating data in the database.

DB Development Guidelines

These below are general guidelines that will assist in developing a strong basis for the actual database design (the logical model).

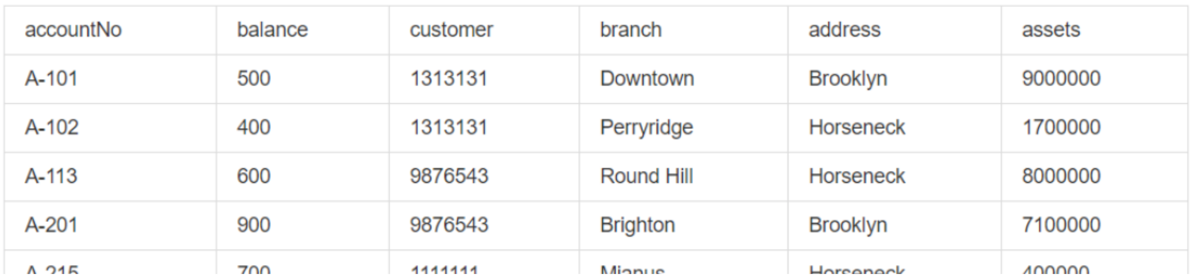
Tips:

1. Document all entities discovered during the information-gathering stage.
2. Document all attributes that belong to each entity. Select candidate and primary keys. Ensure that all non-key attributes for each entity are full-functionally dependent on the primary key.
3. Develop an initial ER diagram and review it with appropriate personnel. (Remember that this is an iterative process.)
4. Create new entities (tables) for multivalued attributes and repeating groups. Incorporate these new entities (tables) in the ER diagram. Review with appropriate personnel.
5. Verify ER modeling by normalizing tables.

Why do we need Normalization?
Generally, a good relational database design must capture all of the necessary attributes and associations. The design should do this with a minimal amount of stored information and no redundant data.

In database design, redundancy is generally undesirable because it causes problems maintaining consistency after updates. However, redundancy can sometimes lead to performance improvements; for example, when redundancy can be used in place of a join to connect data. A join is used when you need to obtain information based on two related tables.

Consider the figure below: customer 1313131 is displayed twice, once for account no. A-101 and again for account A-102. In this case, the customer number is not redundant, although there are deletion anomalies with the table. Having a separate customer table would solve this problem. However, if a branch address were to change, it would have to be updated in multiple places. If the customer number was left in the table as is, then you wouldn't need a branch table and no join would be required, and performance is improved.



accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000

Anomalies

1. Insertion Anomaly:
An insertion anomaly occurs when you are inserting inconsistent information into a table. When we insert a new record, we need to check that the branch data is consistent with existing rows.

2. Update Anomaly:
If a branch changes address, such as the Round Hill branch in the figure above, we need to update all rows referring to that branch. Changing existing information incorrectly is called an update anomaly.

2. Deletion Anomaly:
A deletion anomaly occurs when you delete a record that may contain attributes that shouldn't be deleted. For instance, if we remove information about the last account at a branch, such as account A-101 at the Downtown branch in the figure below, all of the branch information disappears.

The problem with deleting the A-101 row is we don't know where the Downtown branch is located and we lose all information regarding customer 1313131. To avoid these kinds of update or deletion problems, we need to decompose the original table into several smaller tables where each table has minimal overlap with other tables.

Each bank account table must contain information about one entity only, such as the Branch or Customer, as displayed in the figure below.

Following this practice will ensure that when branch information is added or updated it will only affect one record. So, when customer information is added or deleted, the branch information will not be accidentally modified or incorrectly recorded.

How to avoid anomalies?
The best approach to creating tables without anomalies is to ensure that the tables are normalized, and that's accomplished by understanding functional dependencies. FD ensures that all attributes in a table belong to that table. In other words, it will eliminate redundancies and anomalies.

By keeping data separate using individual Project and Employee tables:

1. No anomalies will be created if a budget is changed.
2. No dummy values are needed for projects that have no employees assigned.
3. If an employee's contribution is deleted, no important data is lost.
4. No anomalies are created if an employee's contribution is added.

Functional Dependency
One important theory developed for the entity relational (ER) model involves the notion of functional dependency (FD). The aim of studying this is to improve your understanding of relationships among data and to gain enough formalism to assist with practical database design.

Like constraints, FDs are drawn from the semantics of the application domain. Essentially, functional dependencies describe how individual attributes are related. FDs are a kind of constraint among attributes within a relation and contribute to a good relational schema design. In this chapter, we will look at:

- The basic theory and definition of functional dependency
- The methodology for improving schema designs, also called normalization

A functional dependency (FD) is a relationship between two attributes, typically between the PK and other nonkey attributes within a table. For any relation R, attribute Y is functionally dependent on attribute X (usually the PK), if for every valid instance of X, that value of X uniquely determines the value of Y . This relationship is indicated by the representation below :

X ———>Y
The left side of the above FD diagram is called the determinant, and the right side is the dependent. Here are a few examples.

In the first example, below, SIN determines Name, Address and Birthdate. Given SIN, we can determine any of the other attributes within the table.

SIN ———-> Name, Address, Birthdate

For the second example, SIN and Course determine the date completed (DateCompleted). This must also work for a composite PK.

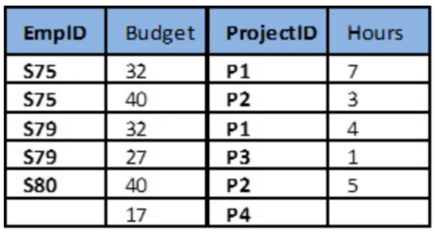
SIN, Course ———> DateCompleted

The third example indicates that ISBN determines Title.

ISBN ———> Title

An Example of Normalization
The figure below shows an example of an employee project table. From this table, we can assume that:

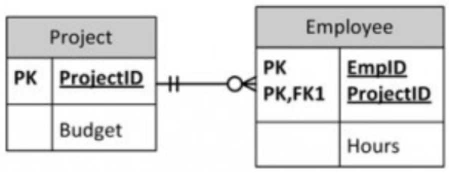
1. EmpID and ProjectID are a composite PK.
2. Project ID determines Budget (i.e., Project P1 has a budget of 32 hours).



EmpID	Budget	ProjectID	Hours
S75	32	P1	7
S75	40	P2	3
S79	32	P1	4
S79	27	P3	1
S80	40	P2	5
	17	P4	

Next, let's look at some possible anomalies that might occur with this table during the following steps.

1. Action: Add row {S85, 35, P1, 9}
2. Problem: There are two tuples with conflicting budgets
3. Action: Delete tuple {S79, 27, P3, 1}
4. Problem: Step #3 deletes the budget for project P3
5. Action: Update tuple {S75, 32, P1, 7} to {S75, 35, P1, 7}
6. Problem: Step #5 creates two tuples with different values for project P1's budget
7. Solution: Create a separate table, each, for Projects and Employees, as shown below.



Normalization

Normalization should be part of the database design process. However, it is difficult to separate the normalization process from the ER modelling process so the two techniques should be used concurrently.

Use an entity relation diagram (ERD) to provide the big picture, or macro view, of an organization's data requirements and operations. This is created through an iterative process that involves identifying relevant entities, their attributes and their relationships.

Normalization procedure focuses on characteristics of specific entities and represents the micro view of entities within the ERD.

What Is Normalization?
Normalization is the branch of relational theory that provides design insights. It is the process of determining how much redundancy exists in a table. The goals of normalization are to:

- Be able to characterize the level of redundancy in a relational schema
- Provide mechanisms for transforming schemas in order to remove redundancy

Normalization theory draws heavily on the theory of functional dependencies. Normalization theory defines six normal forms (NF). Each normal form involves a set of dependency properties that a schema must satisfy and each normal form gives guarantees about the presence and/or absence of update anomalies. This means that higher normal forms have less redundancy, and as a result, fewer update problems.

Normal Forms
All the tables in any database can be in one of the normal forms we will discuss next. Ideally we only want minimal redundancy for PK to FK. Everything else should be derived from other tables. There are six normal forms, but we will only look at the first four, which are:

- First normal form (1NF)
- Second normal form (2NF)
- Third normal form (3NF)
- Boyce-Codd normal form (BCNF)

BCNF is rarely used.

First Normal Form (1NF)
In the first normal form, only single values are permitted at the intersection of each row and column; hence, there are no repeating groups.

To normalize a relation that contains a repeating group, remove the repeating group and form two new relations.

The PK of the new relation is a combination of the PK of the original relation plus an attribute from the newly created relation for unique identification.

Process for 1NF

We will use the Student_Grade_Report table below, from a School database, as our example to explain the process for 1NF.

Student_Grade_Report (StudentNo, StudentName, Major, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade)

- In the Student Grade Report table, the repeating group is the course information. A student can take many courses.
- Remove the repeating group. In this case, it's the course information for each student.
- Identify the PK for your new table.
- The PK must uniquely identify the attribute value (StudentNo and CourseNo).
- After removing all the attributes related to the course and student, you are left with the student course table (**StudentCourse**).
- The Student table (**Student**) is now in first normal form with the repeating group removed.
- The two new tables are shown below.

Student (StudentNo, StudentName, Major)

StudentCourse(StudentNo, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade)

How to update 1NF anomalies

StudentCourse (StudentNo, CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation, Grade)

- To add a new course, we need a student.
- When course information needs to be updated, we may have inconsistencies.

- To delete a student, we might also delete critical information about a course.

Second Normal Form (2NF)
For the second normal form, the relation must first be in 1NF. The relation is automatically in 2NF if, and only if, the PK comprises a single attribute.

If the relation has a composite PK, then each non-key attribute must be fully dependent on the entire PK and not on a subset of the PK (i.e., there must be no partial dependency or augmentation).

Process for 2NF

To move to 2NF, a table must first be in 1NF.

- The Student table is already in 2NF because it has a single-column PK.
- When examining the Student Course table, we see that not all the attributes are fully dependent on the PK; specifically, all course information. The only attribute that is fully dependent is grade.
- Identify the new table that contains the course information.
- Identify the PK for the new table.
- The three new tables are shown below.

Student (<u>StudentNo</u>, StudentName, Major)

CourseGrade (<u>StudentNo, CourseNo</u>, Grade)

CourseInstructor (CourseNo, CourseName, InstructorNo, InstructorName, InstructorLocation)

How to update 2NF anomalies

- When adding a new instructor, we need a course.
- Updating course information could lead to inconsistencies for instructor information.
- Deleting a course may also delete instructor information.

Third Normal Form (3NF)
To be in third normal form, the relation must be in second normal form. Also all transitive dependencies must be removed; a non-key attribute may not be functionally dependent on another non-key attribute.

Process for 3NF

- Eliminate all dependent attributes in transitive relationship(s) from each of the tables that have a transitive relationship.
- Create new table(s) with removed dependency.
- Check new table(s) as well as table(s) modified to make sure that each table has a determinant and that no table contains inappropriate dependencies.
- See the four new tables below.

Student (StudentNo, StudentName, Major)

CourseGrade (StudentNo, CourseNo, Grade)

Course (CourseNo, CourseName, InstructorNo)

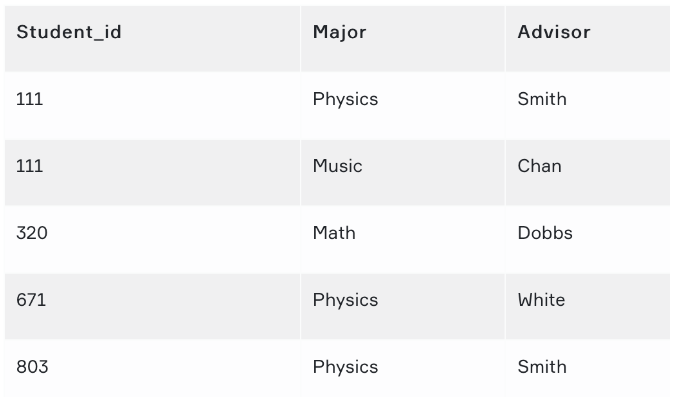
Instructor (InstructorNo, InstructorName, InstructorLocation)

At this stage, there should be no anomalies in third normal form.

Boyce-Codd Normal Form (BCNF)
When a table has more than one candidate key, anomalies may result even though the relation is in 3NF. Boyce-Codd normal form is a special case of 3NF. A relation is in BCNF if, and only if, every determinant is a candidate key.

BCNF Example 1

Consider the following table (**St_Maj_Adv**).



Student_id	Major	Advisor
111	Physics	Smith
111	Music	Chan
320	Math	Dobbs
671	Physics	White
803	Physics	Smith

The semantic rules (business rules applied to the database) for this table are:

1. Each Student may major in several subjects.
2. For each Major, a given Student has only one Advisor.
3. Each Major has several Advisors.
4. Each Advisor advises only one Major.
5. Each Advisor advises several Students in one Major.

The functional dependencies for this table are listed below. The first one is a candidate key; the second is not.

1. Student_id, Major ——> Advisor
2. Advisor ——> Major

Anomalies for this table include:

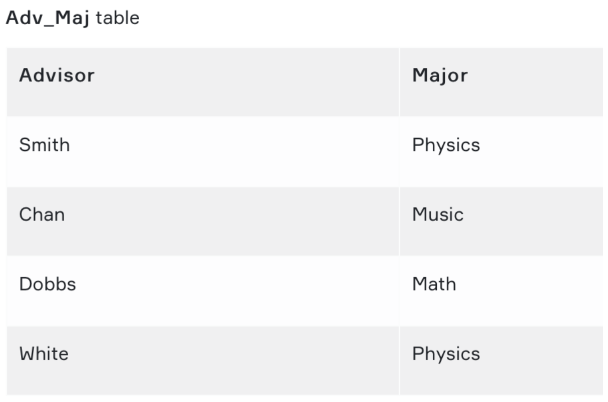
1. Delete – student deletes advisor info
2. Insert – a new advisor needs a student
3. Update – inconsistencies

Note: No single attribute is a candidate key.

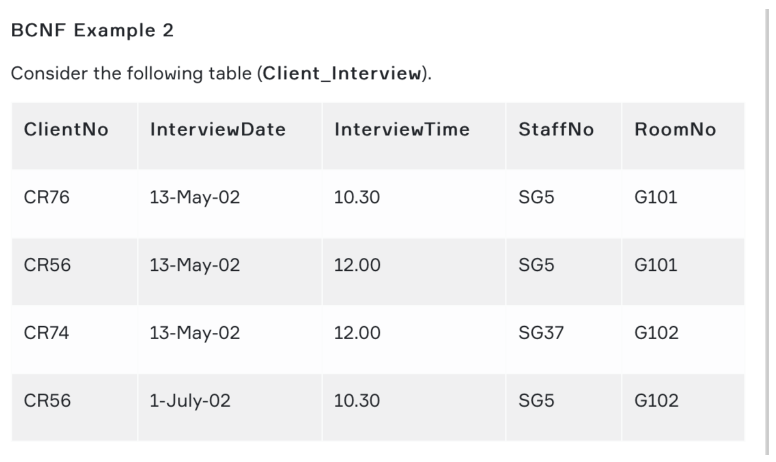
PK can be Student_id, Major or Student_id, Advisor.

To reduce the **St_Maj_Adv** relation to BCNF, you create two new tables:

1. **St_Adv** (Student_id, Advisor)
2. **Adv_Maj** (Advisor, Major)


Adv_Maj table

Advisor	Major
Smith	Physics
Chan	Music
Dobbs	Math
White	Physics


BCNF Example 2

Consider the following table (**Client_Interview**).

ClientNo	InterviewDate	InterviewTime	StaffNo	RoomNo
CR76	13-May-02	10.30	SG5	G101
CR56	13-May-02	12.00	SG5	G101
CR74	13-May-02	12.00	SG37	G102
CR56	1-July-02	10.30	SG5	G102

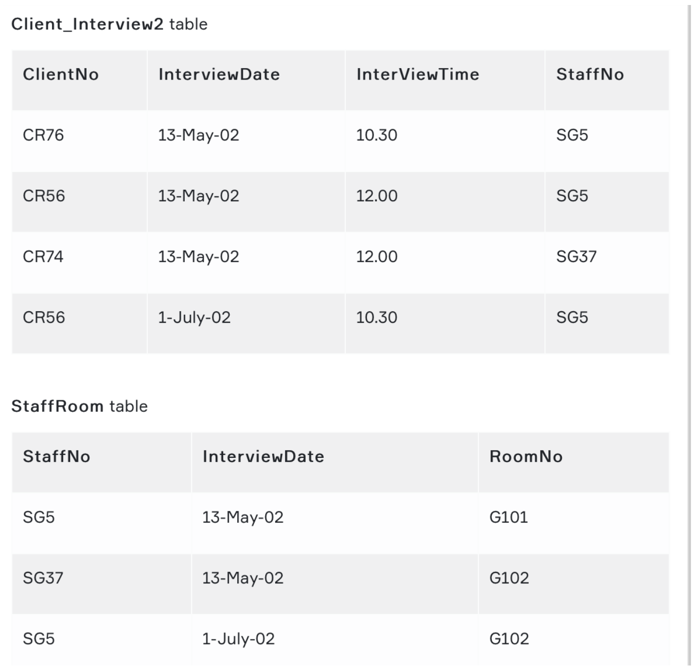
FD1 – ClientNo, InterviewDate –> InterviewTime, StaffNo, RoomNo (PK)

FD2 – staffNo, interviewDate, interviewTime –> clientNO (candidate key: CK)

FD3 – roomNo, interviewDate, interviewTime –> staffNo, clientNo (CK)

FD4 – staffNo, interviewDate –> roomNo

A relation is in BCNF if, and only if, every determinant is a candidate key. We need to create a table that incorporates the first three FDs (Client_Interview2 table) and another table (StaffRoom table) for the fourth FD.


Client_Interview2 table

ClientNo	InterviewDate	InterViewTime	StaffNo
CR76	13-May-02	10.30	SG5
CR56	13-May-02	12.00	SG5
CR74	13-May-02	12.00	SG37
CR56	1-July-02	10.30	SG5

StaffRoom table

StaffNo	InterviewDate	RoomNo
SG5	13-May-02	G101
SG37	13-May-02	G102
SG5	1-July-02	G102

Normalization and Database Design

During the normalization process of database design, make sure that proposed entities meet required normal form before table structures are created. Many real-world databases have been improperly designed or burdened with anomalies if improperly modified during the course of time. You may be asked to redesign and modify existing databases. This can be a large undertaking if the tables are not properly normalized.

Key Terms and Abbreviations

Boyce-Codd normal form (BCNF): a special case of 3rd NF

first normal form (1NF): only single values are permitted at the intersection of each row and column so there are no repeating groups

normalization: the process of determining how much redundancy exists in a table

second normal form (2NF): the relation must be in 1NF and the PK comprises a single attribute

semantic rules: business rules applied to the database

third normal form (3NF): the relation must be in 2NF and all transitive dependencies must be removed; a non-key attribute may not be functionally dependent on another non-key attribute