Entity relationship model

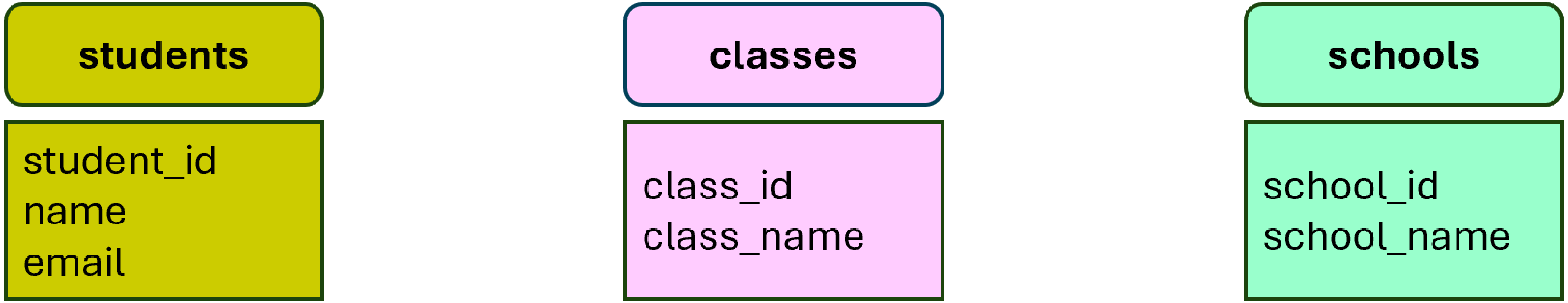
INTRODUCTION TO DATA MODELING IN SNOWFLAKE

Nuno Rocha

Director of Engineering

Introduction to entity-relationship modeling

- **Entity-relationship (ER) model:** Structures normalized data using entities, attributes, and relationships



students

student_id
name
email

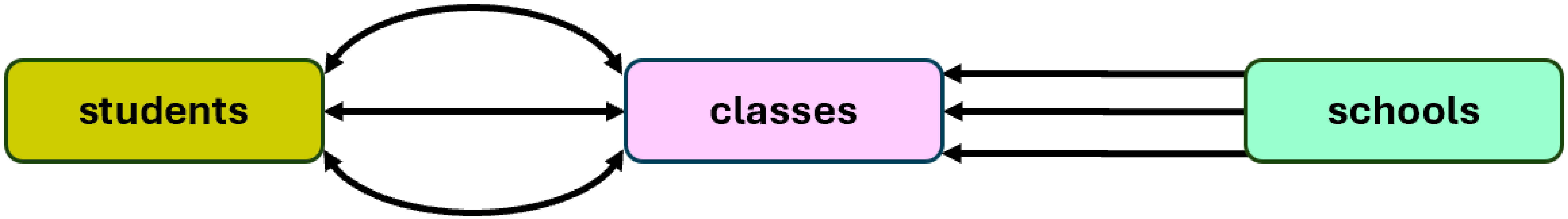
classes

class_id
class_name

schools

school_id
school_name

Example of the entity-relationship model



Students can enroll in multiple classes per semester

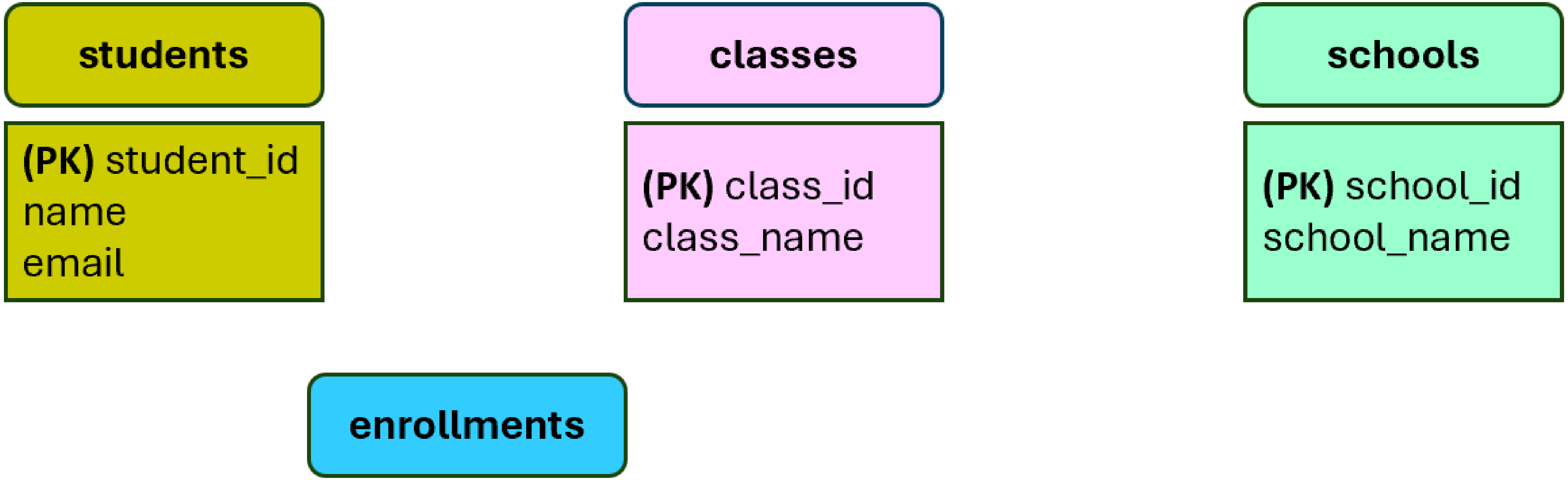
Classes have multiple students enrolled

One school teaches multiple classes

Why choose the ER model

- Clarity
  - Simplifies business connections mirroring real-life interactions
- Organization
  - Breaks down data into related entities, easing information management
- Flexibility
  - Adaptability to grow and change over time

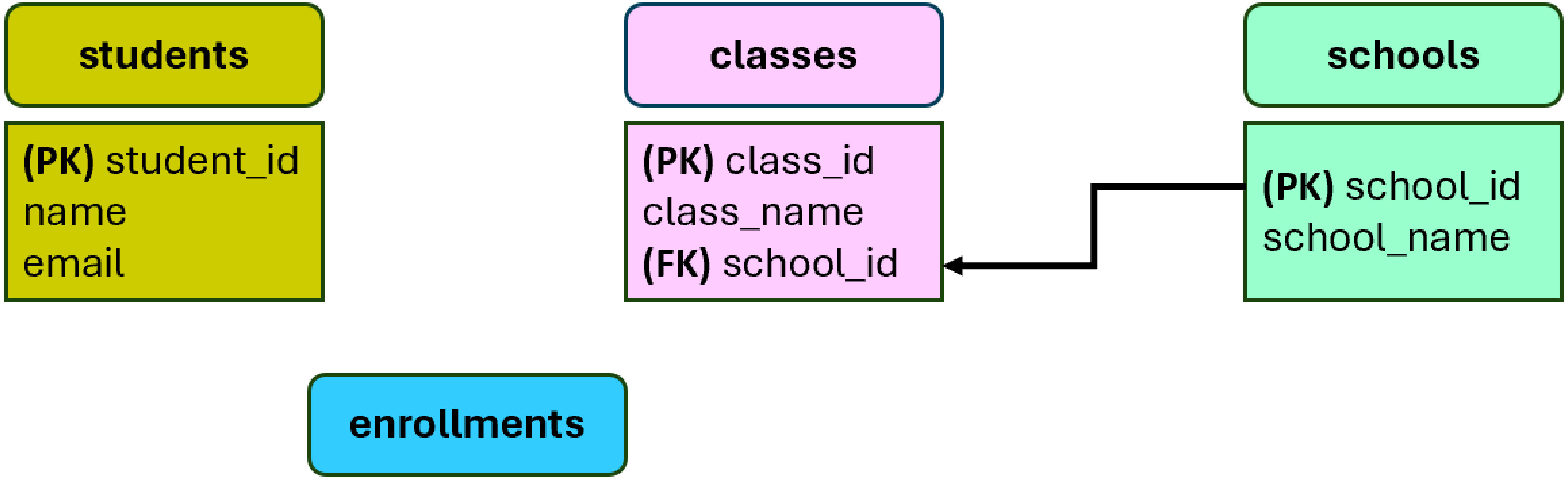
Building the ER model



Building the ER model

- Add relationship between existing entities:

```
ALTER TABLE classes
ADD FOREIGN KEY (school_id) REFERENCES schools(school_id);
```



students

(PK) student_id
name
email

classes

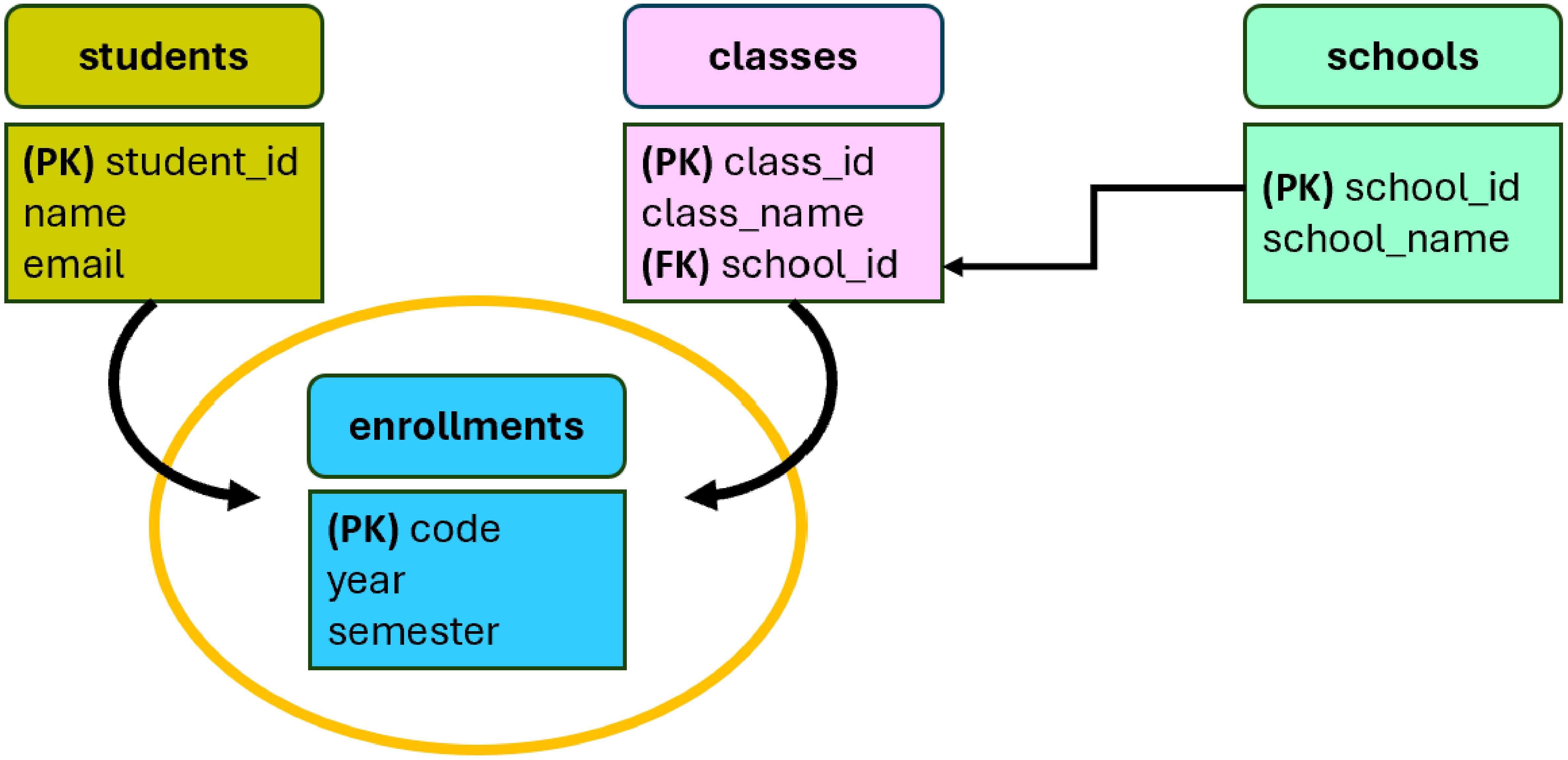
(PK) class_id
class_name
(FK) school_id

schools

(PK) school_id
school_name

enrollments

Building the ER model



students

(PK) student_id
name
email

classes

(PK) class_id
class_name
(FK) school_id

schools

(PK) school_id
school_name

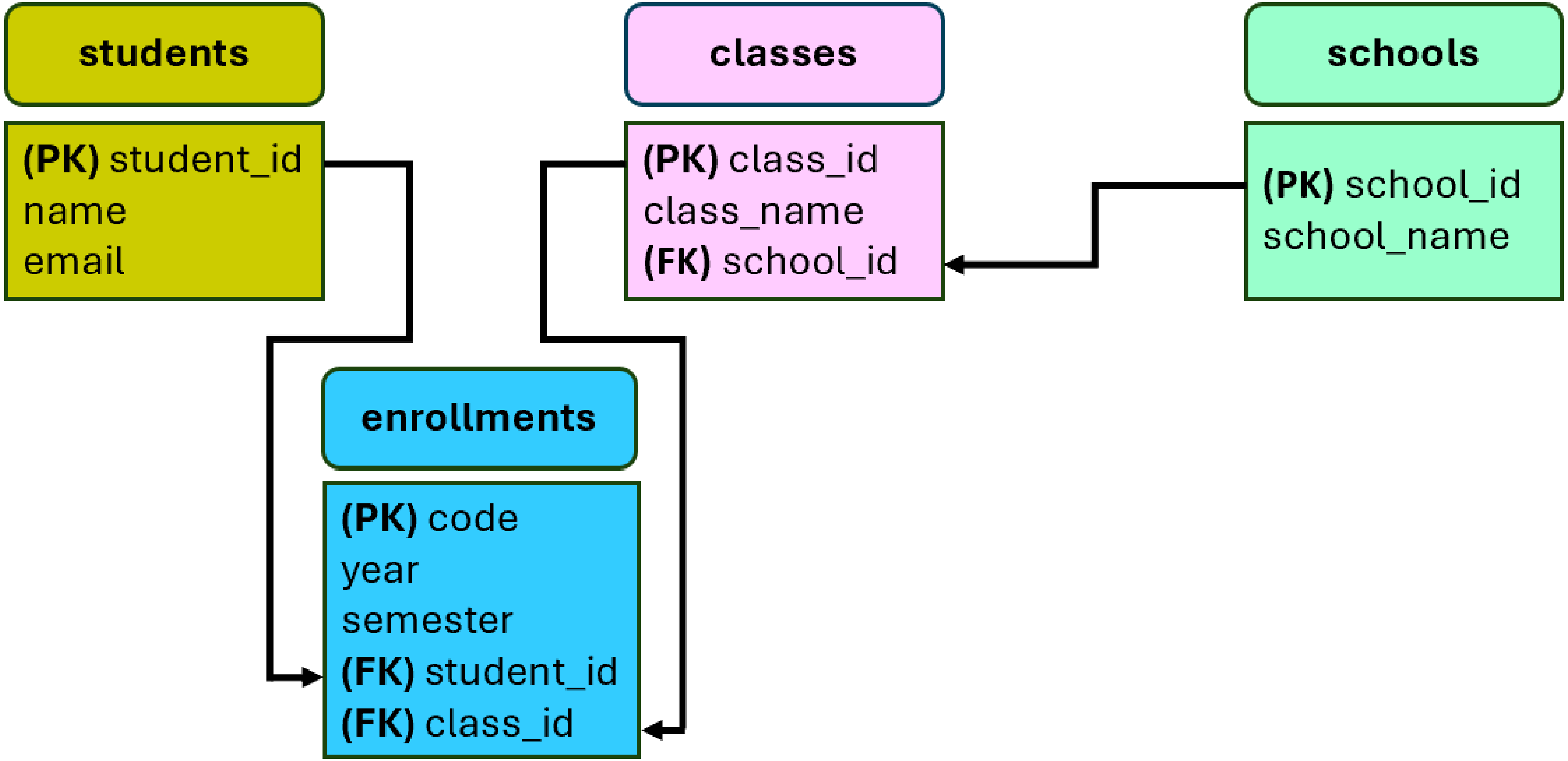
enrollments

(PK) code
year
semester

Building the ER model

```sql
CREATE OR REPLACE TABLE enrollments (
      -- Create a new entity with a unique identifier
      enrollment_id NUMBER(10,0) PRIMARY KEY
      -- Add the entity's attributes
      year NUMBER(4,0),
      semester VARCHAR(255)
      -- Add relationships to other entities
      FOREIGN KEY (student_id) REFERENCES students(student_id),
      FOREIGN KEY (class_id) REFERENCES classes(class_id)
);
```

Building the ER model



students

(PK) student_id
name
email

classes

(PK) class_id
class_name
(FK) school_id

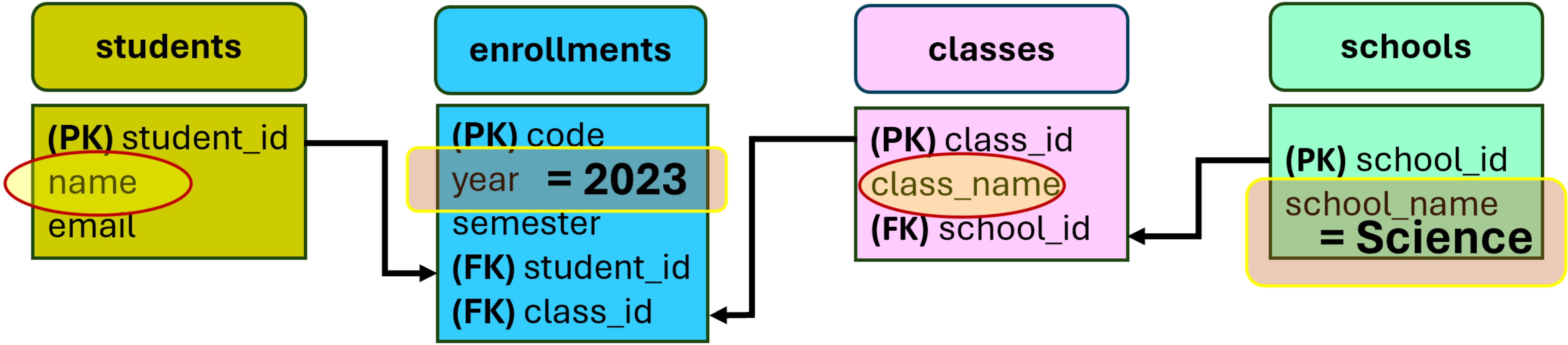
schools

(PK) school_id
school_name

enrollments

(PK) code
year
semester
(FK) student_id
(FK) class_id

Retrieving data from the ER Model



students

(PK) student_id
name
email

enrollments

(PK) code
year = 2023
semester
(FK) student_id
(FK) class_id

classes

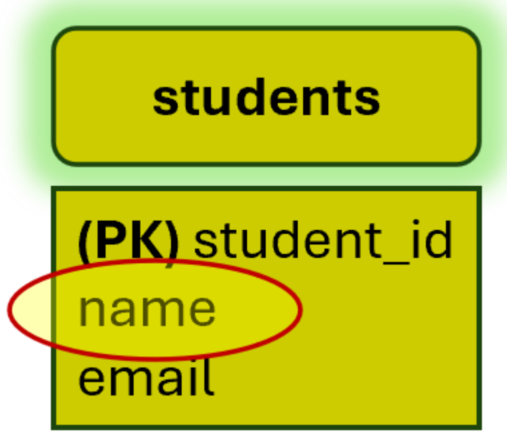
(PK) class_id
class_name
(FK) school_id

schools

(PK) school_id
school_name = Science

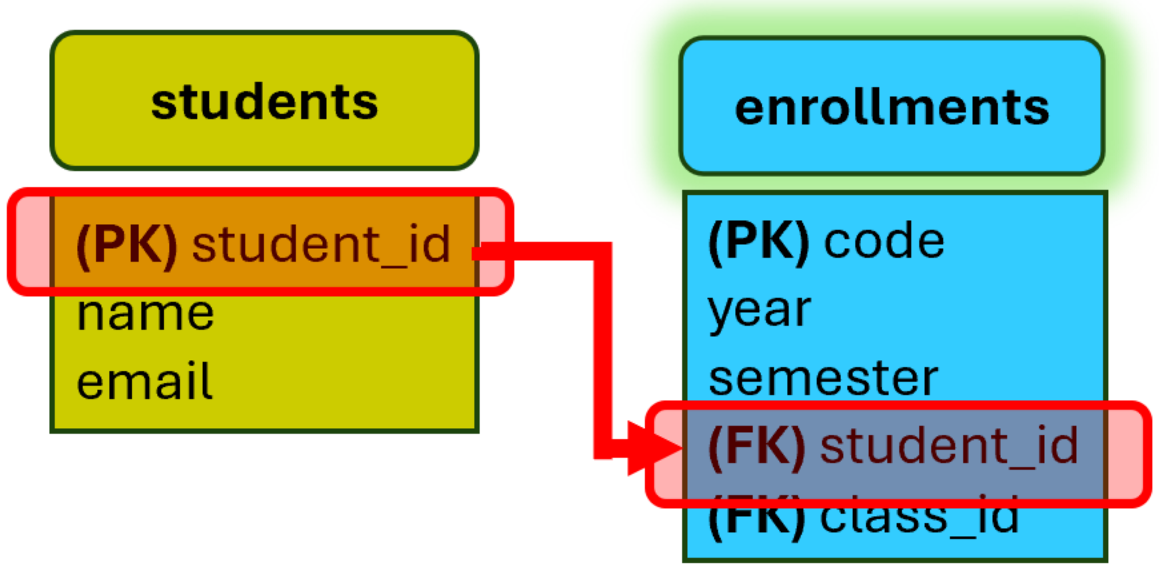
Retrieving data from the ER Model

```sql
SELECT students.name
FROM students;
```



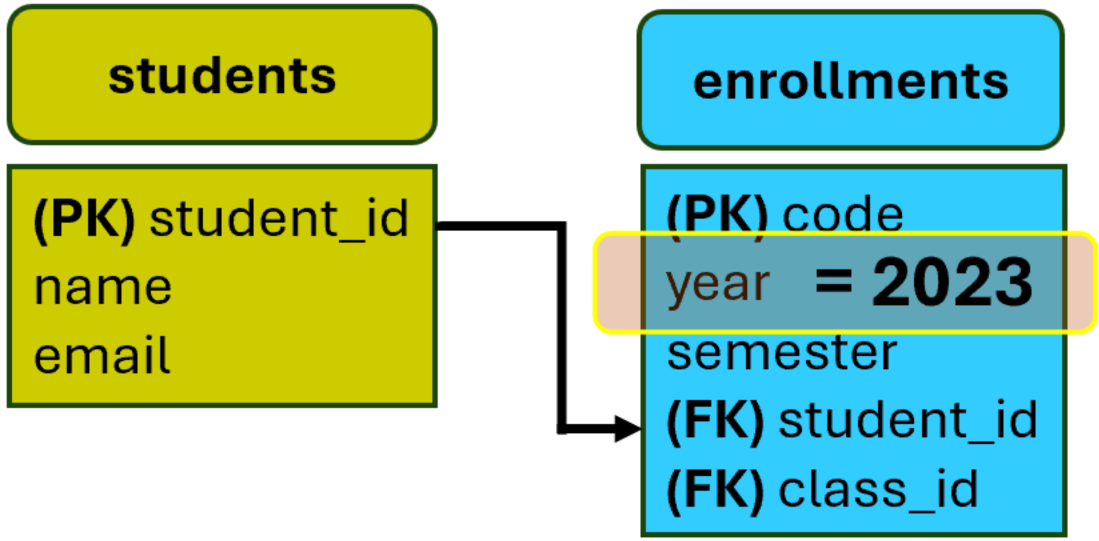
Retrieving data from the ER Model

```
SELECT students.name
FROM students
    JOIN enrollments
    ON students.student_id = enrollments.student_id;
```



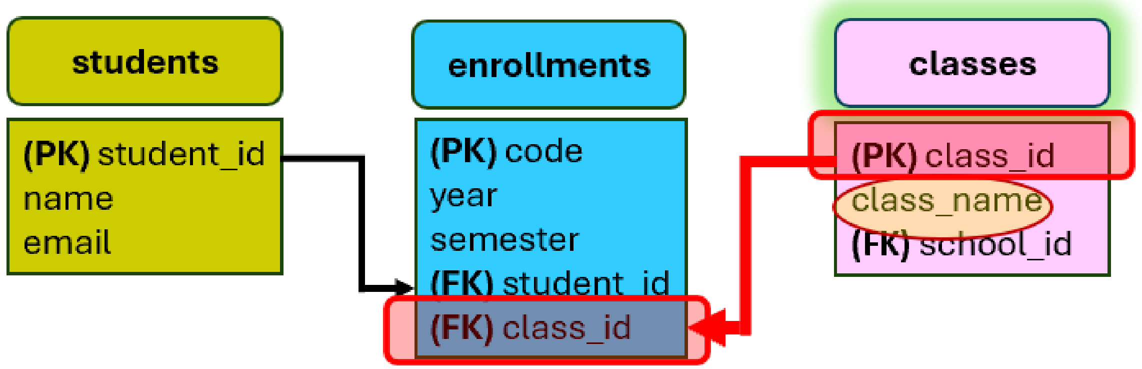
Retrieving data from the ER Model

```
SELECT students.name
FROM students
    JOIN enrollments
    ON students.student_id = enrollments.student_id
WHERE enrollments.year = '2023';
```



Retrieving data from the ER Model

```
SELECT students.name,
    classes.class_name
FROM students
    JOIN enrollments
    ON students.student_id = enrollments.student_id
    JOIN classes
    ON enrollments.class_id = classes.class_id
WHERE enrollments.year = '2023';
```

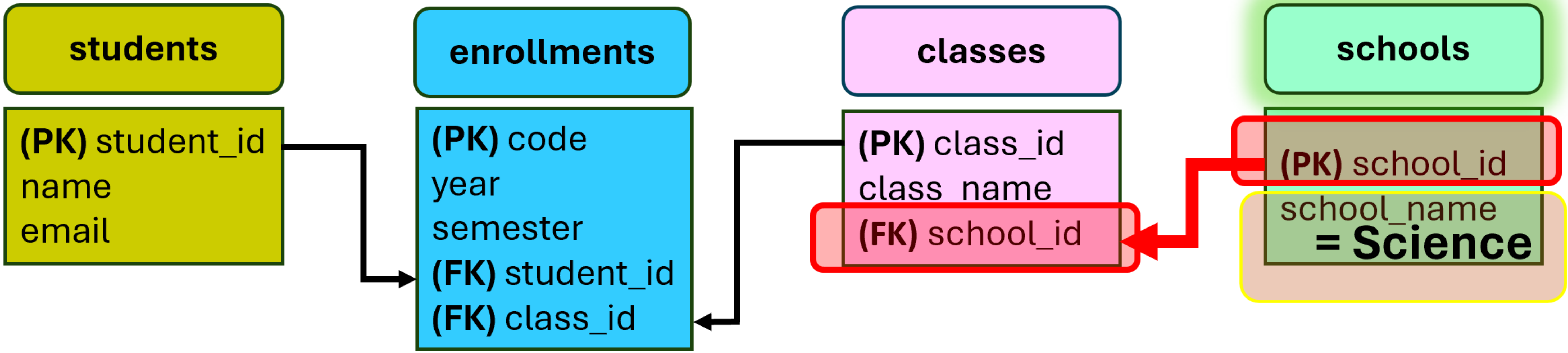


Retrieving data from the ER Model

```sql
SELECT students.name,
    classes.class_name
FROM students
    JOIN enrollments
    ON students.student_id = enrollments.student_id
    JOIN classes
    ON enrollments.class_id = classes.class_id
    JOIN schools
    ON classes.department_id = schools.school_id
WHERE enrollments.year = '2023'
    AND schools.school_name = 'Science';
```

Retrieving data from the ER Model

- `JOIN ON` : SQL clause to combine rows from tables, based `ON` a related column.



students

(PK) student_id
name
email

enrollments

(PK) code
year
semester
(FK) student_id
(FK) class_id

classes

(PK) class_id
class_name
(FK) school_id

schools

(PK) school_id
school_name
= Science

Terminology and functions overview

- **Entity-relationship (ER) model:** Structures normalized data using entities and relationships.
- `SELECT FROM` : SQL command to fetch columns from a table.
- `JOIN ON` : SQL clause to combine rows from tables, based `ON` a related column.
- `WHERE` : SQL clause to filter records based on a set condition.
- `AND` : Logical operator used with `WHERE` clause to combine multiple conditions.

```sql
-- Querying data from merged entities filtered by specific conditions
SELECT column_name
FROM table_name
    JOIN other_table ON table_name.FK = other_table.PK
WHERE column_name  condition  value
    AND column_name  condition  value;
```

Let's practice!

INTRODUCTION TO DATA MODELING IN SNOWFLAKE

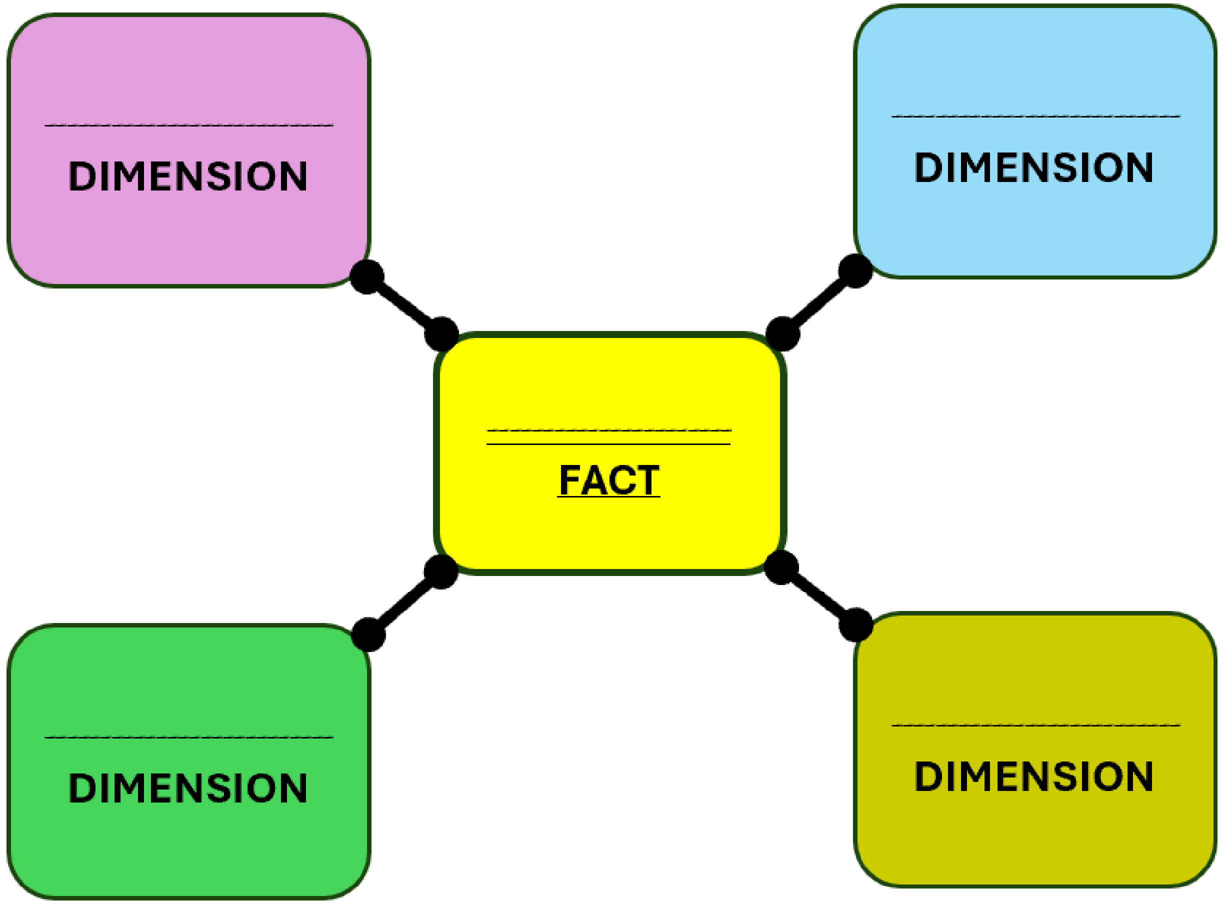
Dimensional modeling

INTRODUCTION TO DATA MODELING IN SNOWFLAKE

Nuno Rocha
Director of Engineering

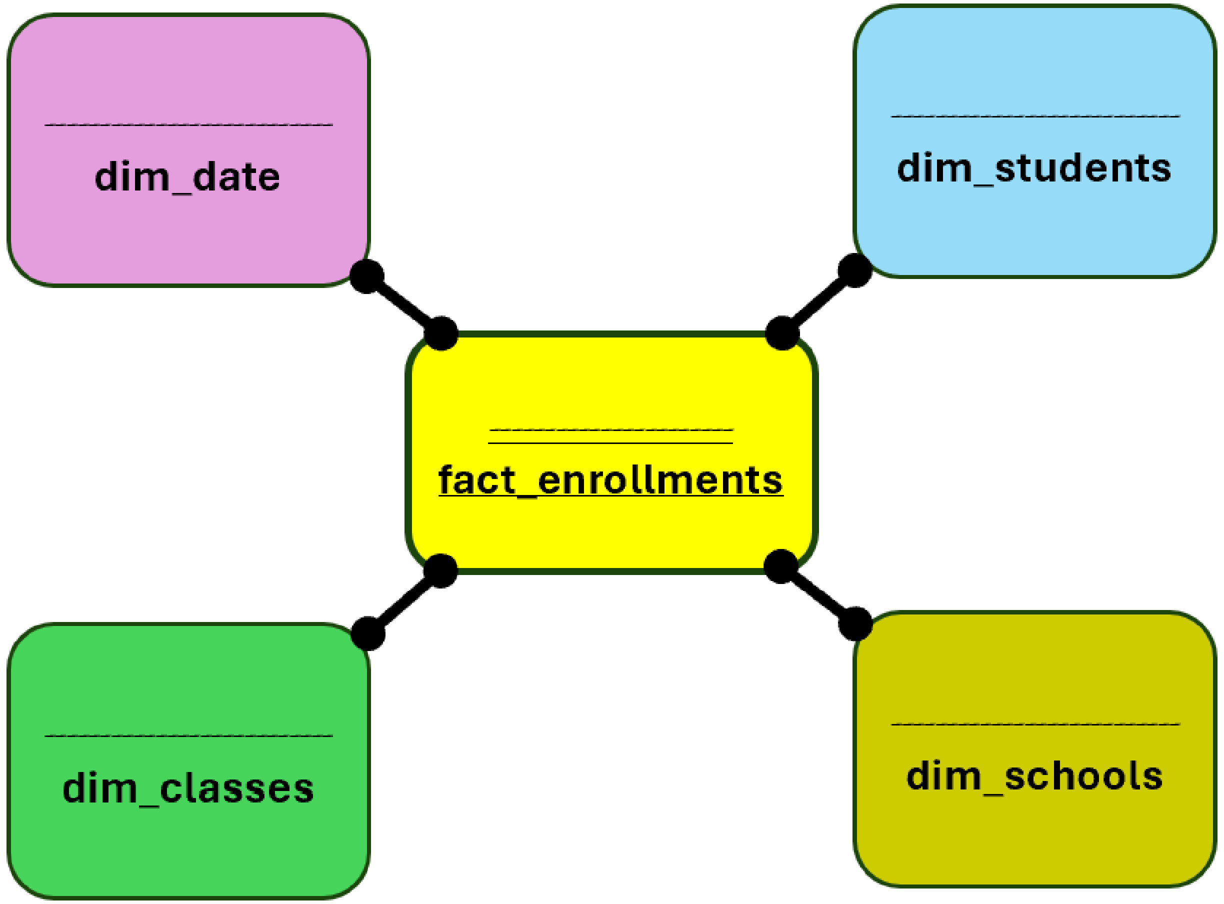
Introduction to the dimensional data model

- **Dimensional modeling:** A data structuring technique that separates measurements (facts) from descriptive details (dimensions) optimized for reporting and analysis.

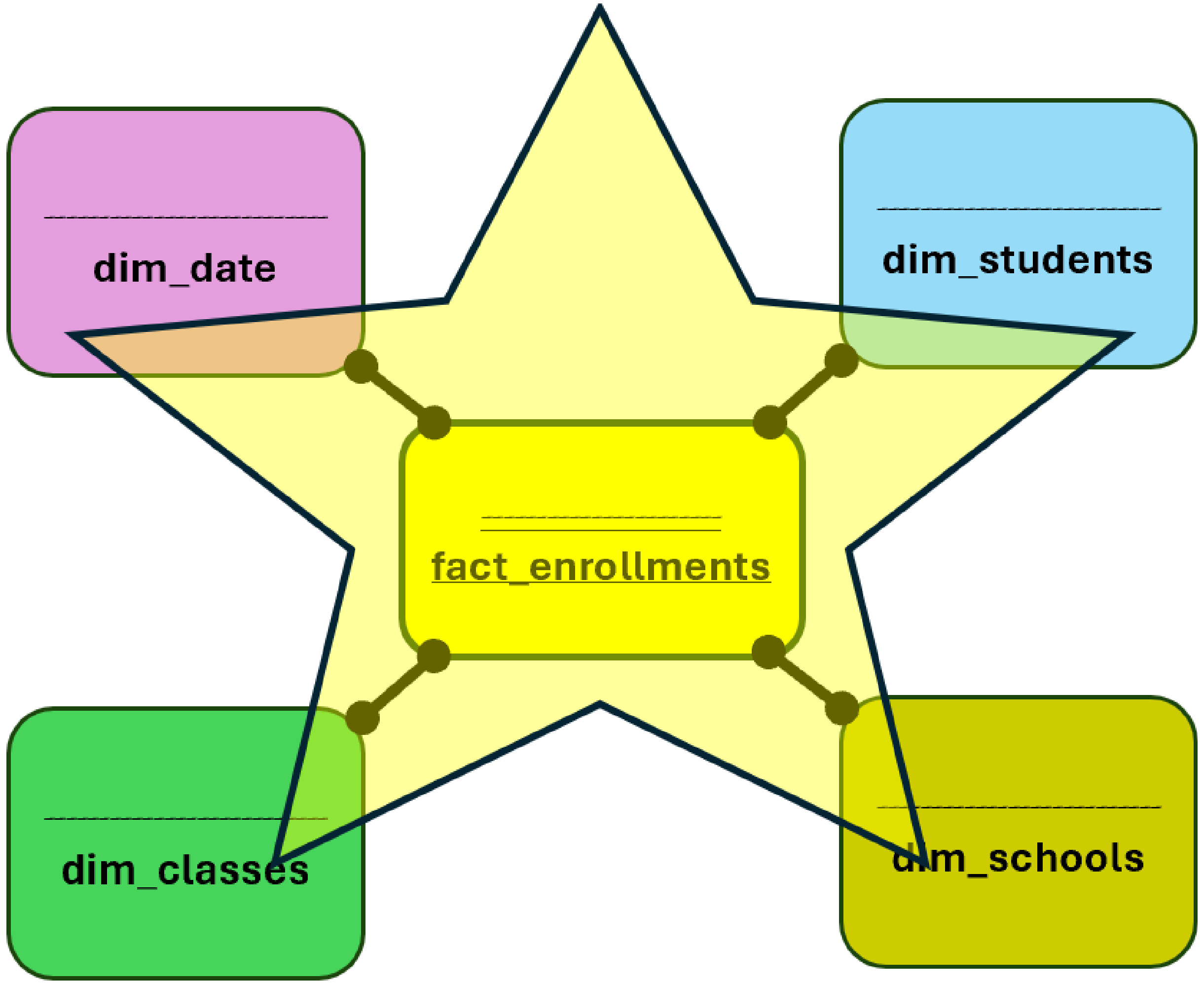


Introduction to the dimensional data model (1)

- **Dimensions:** Entities with categorical data in a dimensional model.
- **Facts:** Entities that capture and quantify activities within the categories in the dimensions.



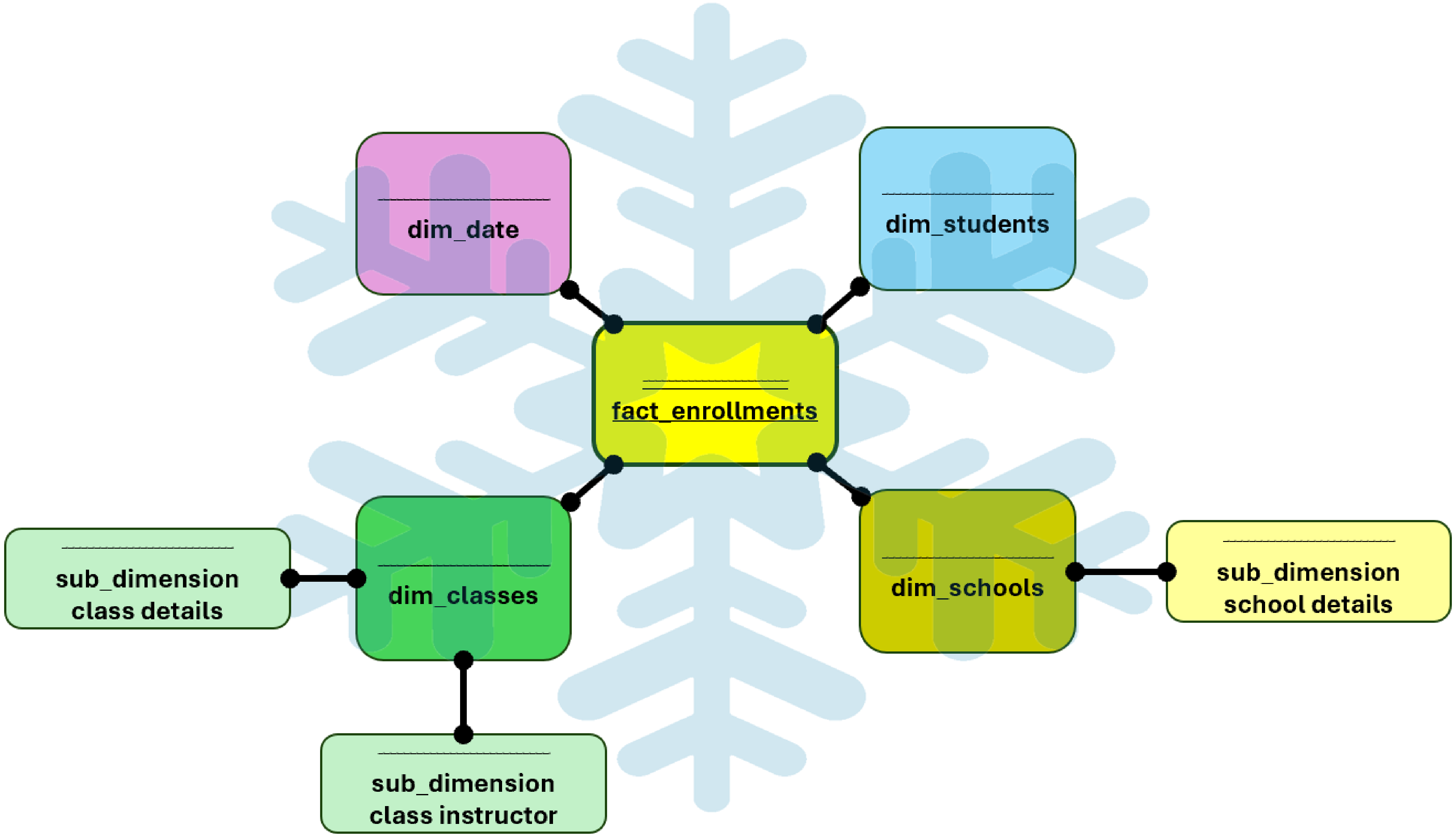
Star dimensional model schema



Star dimensional model schema

- **Snowflake Data Warehouse:** A cloud-based storage and analytics service.
- **Snowflake Schema:** A method of organizing data in a dimensional model which includes sub-dimensions.

Snowflake dimensional model schema


dim_date
dim_students
fact_enrollments
dim_classes
dim_schools
sub_dimension class details
sub_dimension school details
sub_dimension class instructor

Defining dimensions

- Rename entities to dim_EntityName for clarity, following up dimensions in the model:

```sql
ALTER TABLE students RENAME TO dim_students;
```

```sql
ALTER TABLE classes RENAME TO dim_classes;
```

```sql
ALTER TABLE schools RENAME TO dim_schools;
```

Defining date dimension

- Creating the `dim_date` table to store key fixed dates related to student enrollments in school:

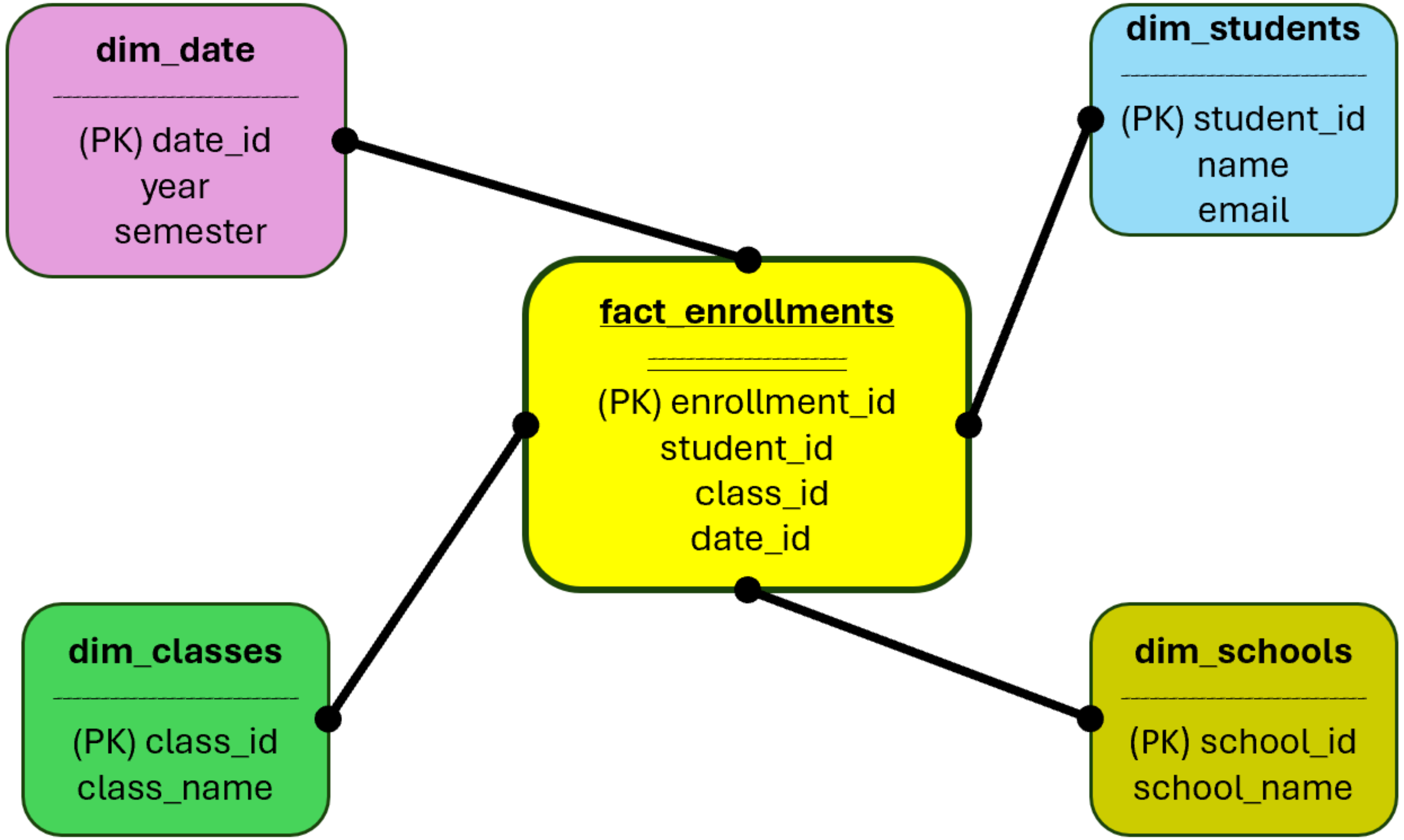
```
CREATE OR REPLACE TABLE dim_date (
    date_id NUMBER(10,0) PRIMARY KEY,
    year NUMBER(4,0),
    semester VARCHAR(255)
);
```

Defining enrollments fact

- Create a fact entity containing references to all the dimensions:

```
CREATE OR REPLACE TABLE fact_enrollments (
    enrollment_id NUMBER(10,0) PRIMARY KEY,
    student_id NUMBER(10,0),
    class_id NUMBER(10,0),
    date_id NUMBER(10,0),
    FOREIGN KEY (student_id) REFERENCES dim_students(student_id),
    FOREIGN KEY (class_id) REFERENCES dim_classes(class_id),
    FOREIGN KEY (date_id) REFERENCES dim_date(date_id)
);
```

Retrieving data from the dimensions



dim_date

(PK) date_id
year
semester

dim_students

(PK) student_id
name
email

fact_enrollments

(PK) enrollment_id
student_id
class_id
date_id

dim_classes

(PK) class_id
class_name

dim_schools

(PK) school_id
school_name

Retrieving data from the dimensions (1)

```
SELECT name,
    class_name
FROM fact_enrollments
    JOIN dim_students -- Joining to get student names
    ON fact_enrollments.student_id = dim_students.student_id
    JOIN dim_classes -- Joining to get class names
    ON fact_enrollments.class_id = dim_classes.class_id
    JOIN dim_schools -- Joining to filter for the 'Science' school
    ON dim_classes.school_id = dim_schools.school_id
    JOIN dim_date -- Joining to restrict data to the year 2023
    ON fact_enrollments.date_id = dim_date.date_id
WHERE dim_schools.school_name = 'Science'
    AND dim_date.year = 2023;
```

Terminology and functions overview

- **Dimensional modeling:** A data structuring technique that separates measurements (facts) from descriptive details (dimensions) optimized for reporting and analysis
- **Dimensions:** Entities with categorical data in a dimensional model
- **Facts:** Entities that capture and quantify activities within the categories in the dimensions
- **`ALTER TABLE`** : SQL command used to modify the structure of an existing entity
- **`RENAME TO`** : SQL command, used with `ALTER TABLE` , to rename an entity
- **`JOIN ON`** : SQL clause to combine rows from tables, based `ON` a related column
- **`WHERE`** : SQL clause to filter records based on a set condition
- **`AND`** : Logical operator used with `WHERE` clause to combine multiple conditions

Functions overview

```
-- Modifying a entity
ALTER TABLE table_name
RENAME TO new_name;
```

```
-- Querying data from merged entities filtered by specific conditions
SELECT
    table_name.column_name,
    other_name.*
FROM table_name
    JOIN other_table
    ON table_name.FK = other_table.PK
WHERE column_name  condition  value
    AND column_name  condition  value;
```

Let's practice!

INTRODUCTION TO DATA MODELING IN SNOWFLAKE

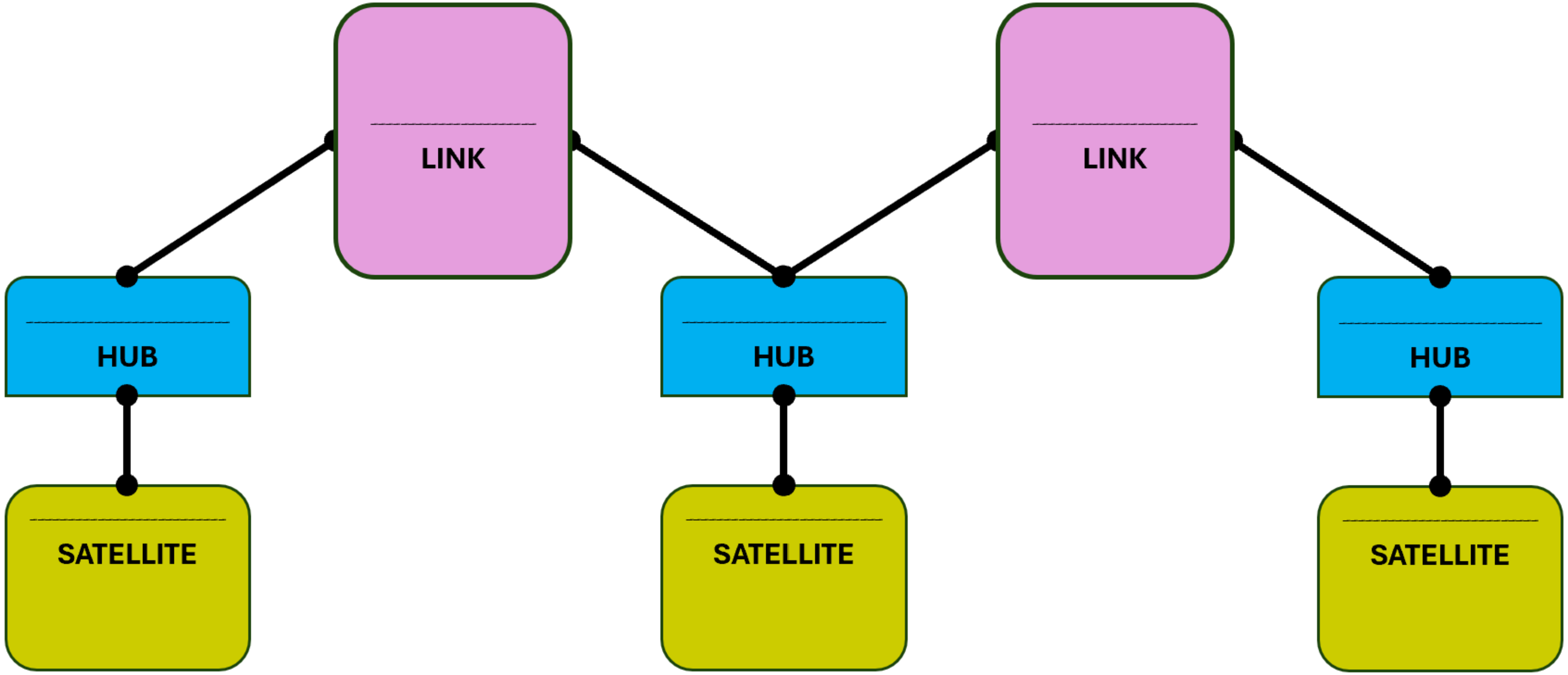
Data Vault

INTRODUCTION TO DATA MODELING IN SNOWFLAKE

Nuno Rocha
Director of Engineering

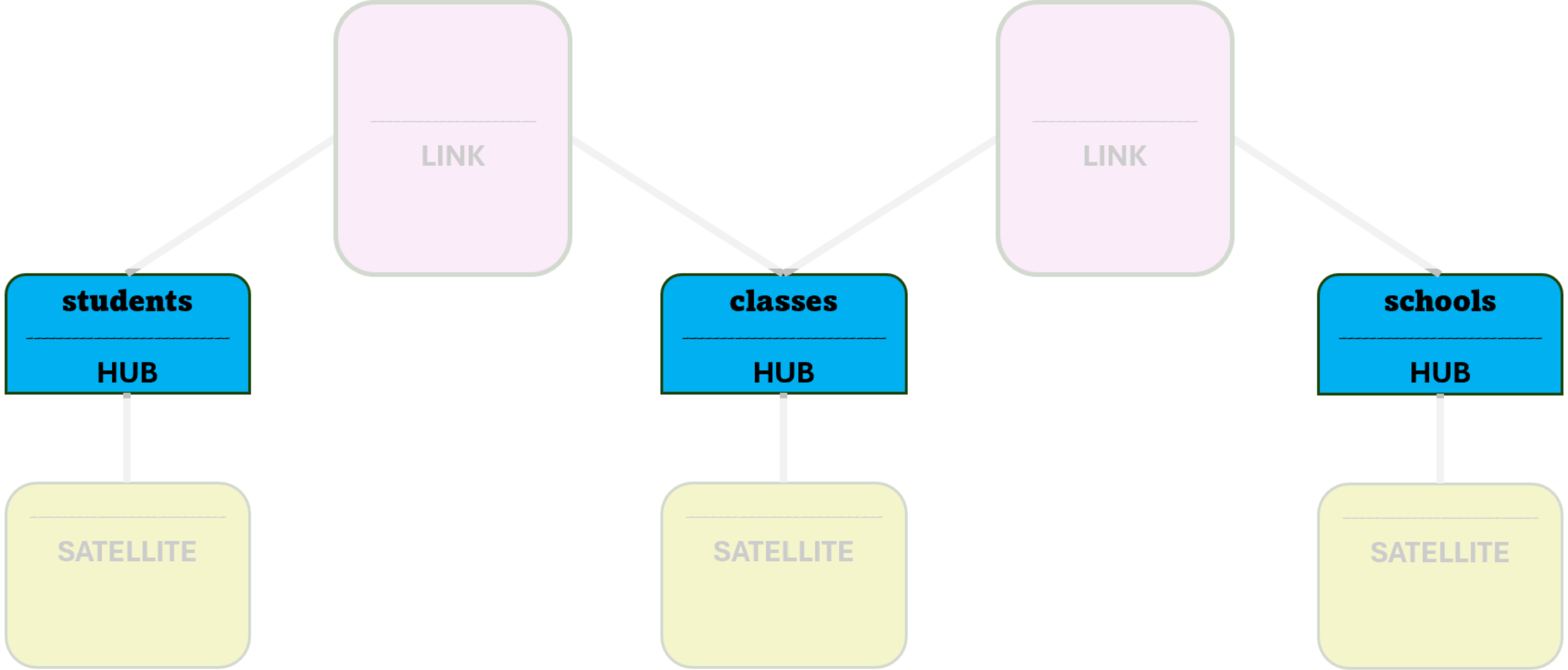
Introduction to the data vault model

- **Data vault model:** A modeling technique focusing on historical data tracking, characterized by using hubs, links, and satellites.



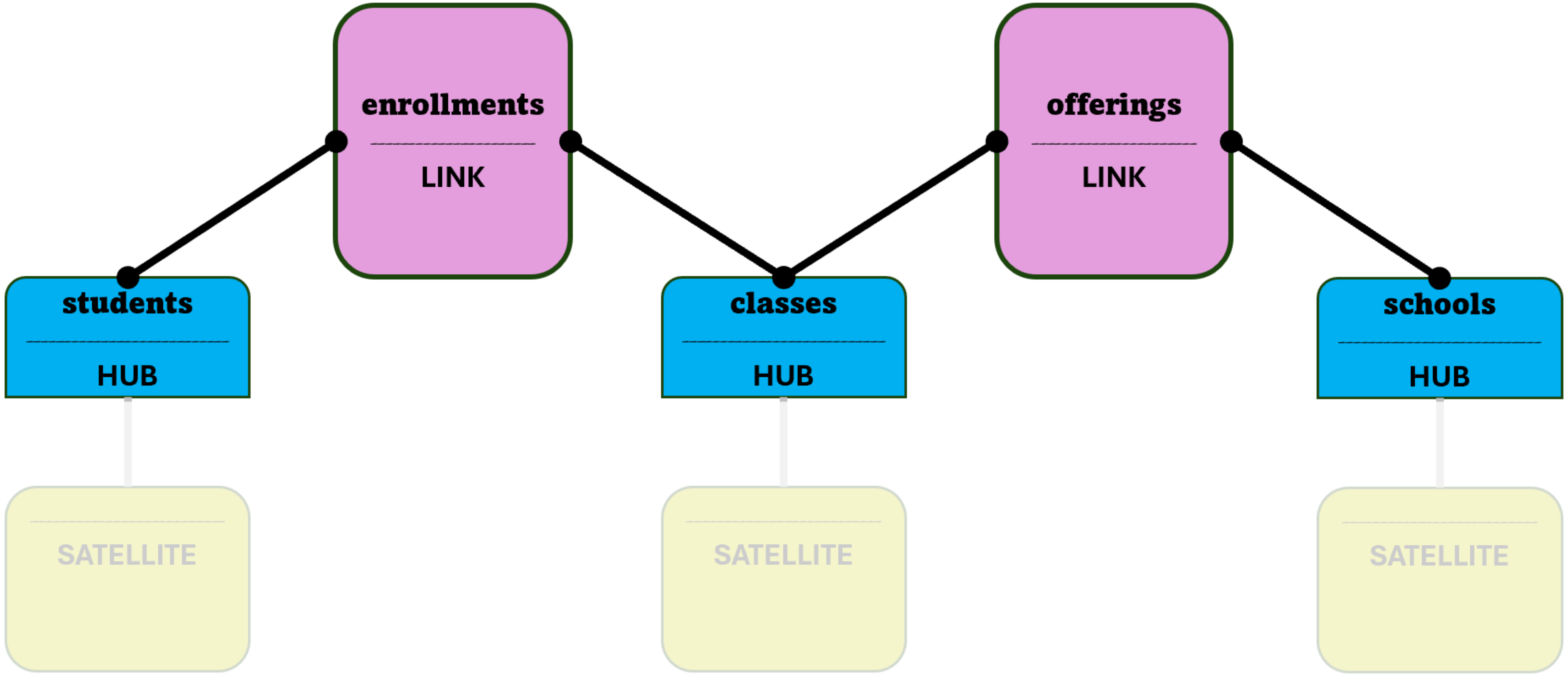
Components of data vault

- **Hubs:** Represent unique business concepts using a singular business key.



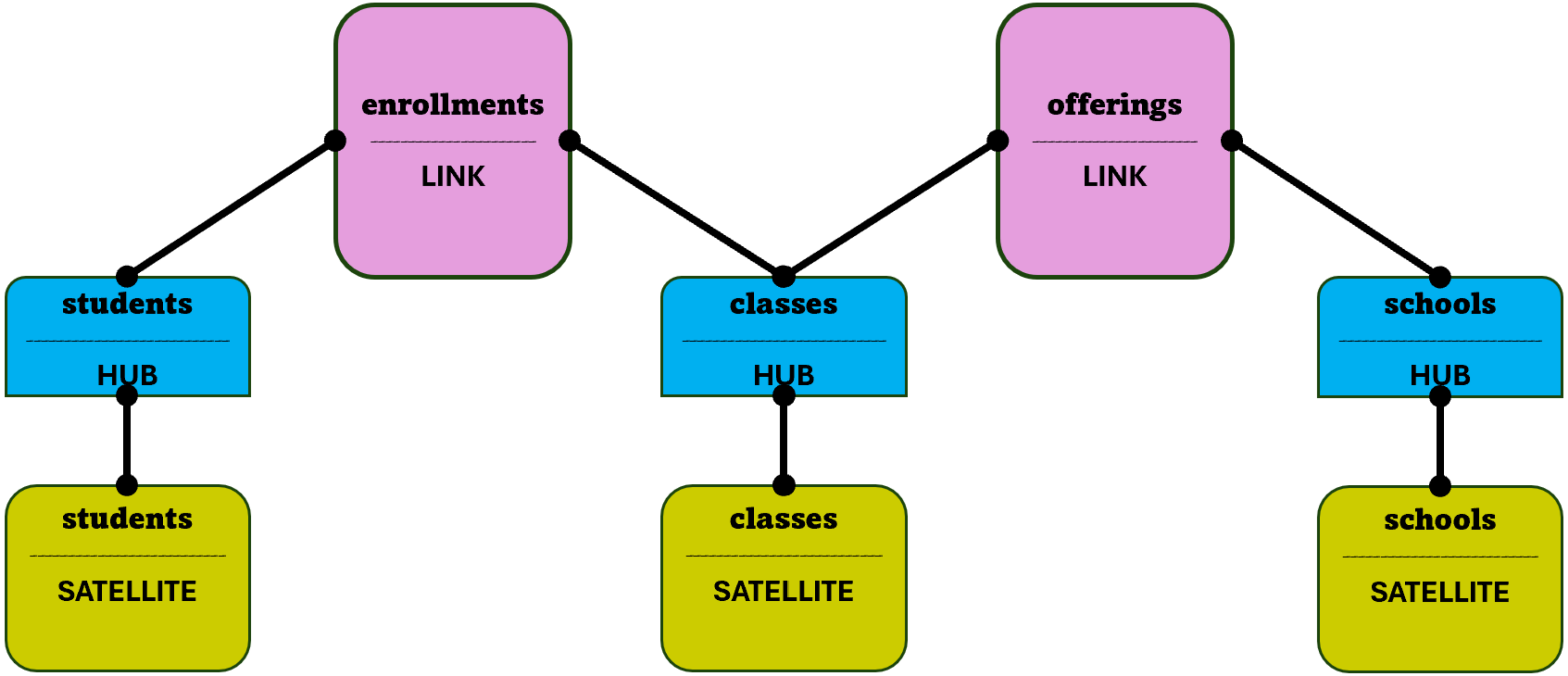
Components of data vault (1)

- **Links:** Capture relationships and interactions between hubs.

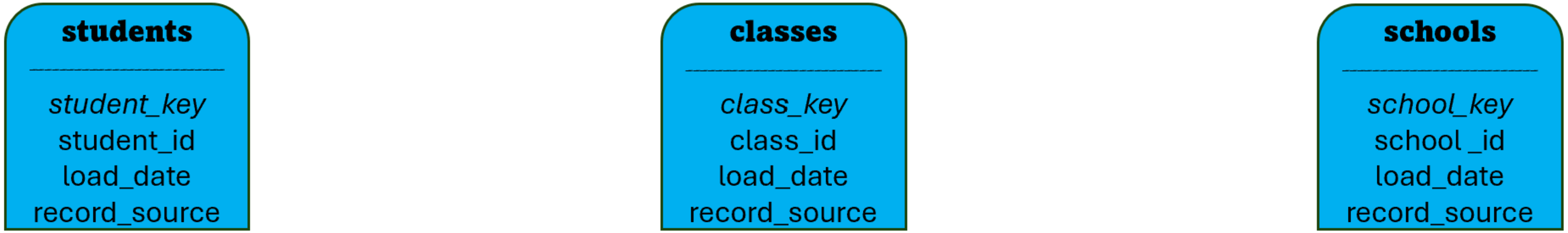


Components of data vault (2)

- **Satellites:** Store descriptive and historical details related to hubs and links.



Building hubs



students

student_key
student_id
load_date
record_source

classes

class_key
class_id
load_date
record_source

schools

school_key
school _id
load_date
record_source

Building hubs (1)

- `AUTOINCREMENT` : Attribute property to automatically generate unique, sequentially increasing numeric values for each new row.

```
CREATE OR REPLACE TABLE hub_students (
    student_key NUMBER(10,0) AUTOINCREMENT PRIMARY KEY
);
```

Building hubs (2)

- Create a new hub with a unique numerical key generated automatically and the hub's concept id:

```sql
CREATE OR REPLACE TABLE hub_students (
    student_key NUMBER(10,0) AUTOINCREMENT PRIMARY KEY,
    student_id NUMBER(10,0)
);
```

[1] Next, we list the business key that identifies each concept, student_id will identify each student

Building hubs (3)

- Add historical tracking attributes:

```sql
CREATE OR REPLACE TABLE hub_students (
    student_key NUMBER(10,0) AUTOINCREMENT PRIMARY KEY,
    student_id NUMBER(10,0),
    load_date TIMESTAMP,
    record_source VARCHAR(255)
);
```

Building hubs (4)

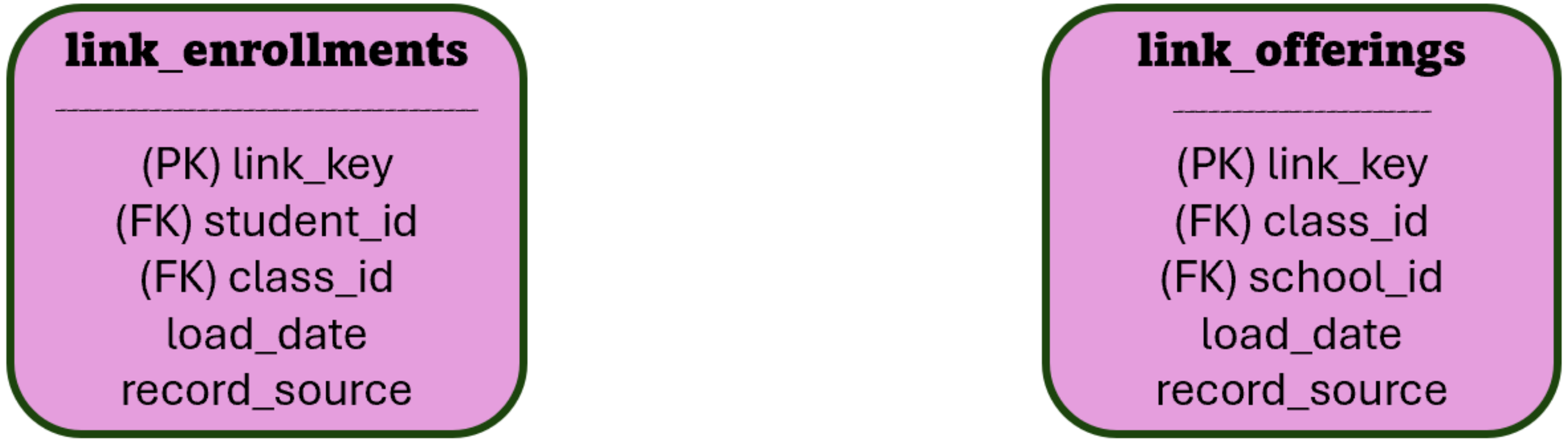
- Create new classes hub:

```
CREATE OR REPLACE TABLE hub_classes (
    class_key NUMBER(10,0)
          AUTOINCREMENT
          PRIMARY KEY,
    class_id NUMBER(10,0),
    load_date TIMESTAMP,
    record_source VARCHAR(255)
);
```

- Create new schools hub:

```
CREATE OR REPLACE TABLE hub_schools (
    school_key NUMBER(10,0)
          AUTOINCREMENT
          PRIMARY KEY,
    school_id NUMBER(10,0),
    load_date TIMESTAMP,
    record_source VARCHAR(255)
);
```

Building links



link_enrollments

(PK) link_key
(FK) student_id
(FK) class_id
load_date
record_source

link_offerings

(PK) link_key
(FK) class_id
(FK) school_id
load_date
record_source

Building links (1)

- Create a link entity with a unique numerical key generated automatically:

```
CREATE OR REPLACE TABLE link_enrollments (
    link_key NUMBER(10,0) AUTOINCREMENT PRIMARY KEY
);
```

Building links (2)

- Add relationships to other entities:

```
CREATE OR REPLACE TABLE link_enrollments (
    link_key NUMBER(10,0) AUTOINCREMENT PRIMARY KEY,
    student_key NUMBER(10,0),
    class_key NUMBER(10,0),
    FOREIGN KEY (student_key) REFERENCES hub_students(student_key),
    FOREIGN KEY (class_key) REFERENCES hub_classes(class_key)
);
```

Building links (3)

- Add historical tracking attributes:

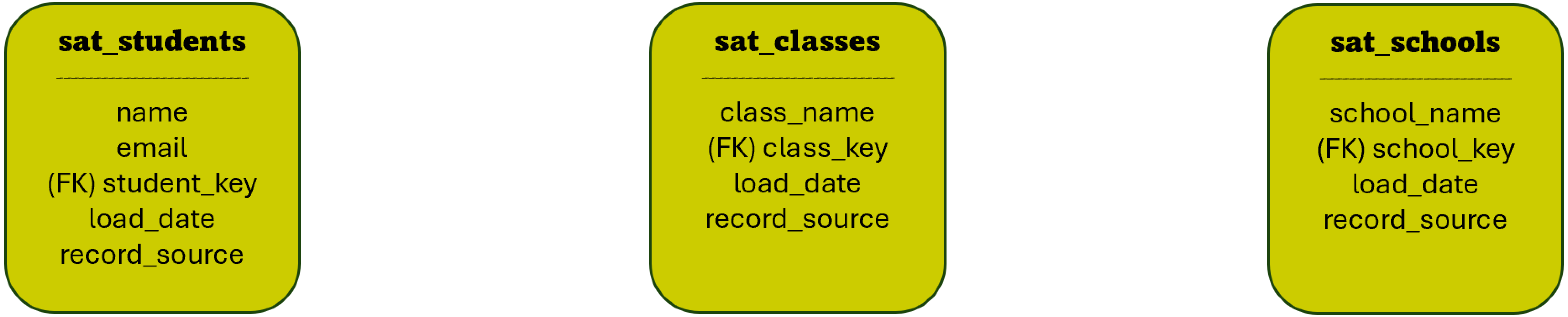
```
CREATE OR REPLACE TABLE link_enrollments (
    link_key NUMBER(10,0) AUTOINCREMENT PRIMARY KEY,
    student_key NUMBER(10,0),
    class_key NUMBER(10,0),
    load_date TIMESTAMP,
    record_source VARCHAR(255),
    FOREIGN KEY (student_key) REFERENCES hub_students(student_key),
    FOREIGN KEY (class_key) REFERENCES hub_classes(class_key)
);
```

Building links (4)

- Create new offerings link entity:

```sql
CREATE OR REPLACE TABLE link_offerings (
    link_key NUMBER(10,0) AUTOINCREMENT PRIMARY KEY,
    class_key NUMBER(10,0),
    school_key NUMBER(10,0),
    load_date TIMESTAMP,
    record_source VARCHAR(255),
    FOREIGN KEY (class_key) REFERENCES hub_classes(class_key),
    FOREIGN KEY (school_key) REFERENCES hub_schools(school_key)
);
```

Building satellites



sat_students

name
email
(FK) student_key
load_date
record_source

sat_classes

class_name
(FK) class_key
load_date
record_source

sat_schools

school_name
(FK) school_key
load_date
record_source

Building satellites (1)

- Create a new satellite entity listing all concept attributes:

```sql
CREATE OR REPLACE TABLE sat_student (
    name VARCHAR(255),
    email VARCHAR(255)
);
```

Building satellites (2)

- Add historical tracking attributes:

```
CREATE OR REPLACE TABLE sat_student (
    name VARCHAR(255),
    email VARCHAR(255),
    load_date TIMESTAMP,
    record_source VARCHAR(255)
);
```

Building satellites (3)

- Add a link between the satellite and its respective hub:

```sql
CREATE OR REPLACE TABLE sat_student (
    student_key NUMBER(10,0),
    name VARCHAR(255),
    email VARCHAR(255),
    load_date TIMESTAMP,
    record_source VARCHAR(255),
    FOREIGN KEY (student_key) REFERENCES hub_students(student_key)
);
```

Building satellites (4)

- Create a new class satellite:

```
CREATE OR REPLACE TABLE sat_class (
    class_key NUMBER(10,0),
    class_name VARCHAR(255),
    load_date TIMESTAMP,
    record_source VARCHAR(255),
    FOREIGN KEY (class_key)
    REFERENCES hub_classes(class_key)
);
```

- Create a new class school:

```
CREATE OR REPLACE TABLE sat_school (
    school_key NUMBER(10,0),
    school_name VARCHAR(255),
    load_date TIMESTAMP,
    record_source VARCHAR(255),
    FOREIGN KEY (school_key)
    REFERENCES hub_schools(school_key)
);
```

Terminology and functions overview

- **Data vault model:** A modeling technique focusing on historical data tracking, characterized by using hubs, links, and satellites.
- **Hubs:** Represent unique business concepts using a singular business key.
- **Links:** Capture relationships and interactions between hubs.
- **Satellites:** Store descriptive and historical details related to hubs and links.
- `CREATE OR REPLACE TABLE` : SQL command to create or replace a table structure.
- `PRIMARY KEY` : SQL clause to define a column as the unique identifier.
- `AUTOINCREMENT` : Attribute property to automatically generate unique, sequentially increasing numeric values for each new row.
- `FOREIGN KEY (...) REFERENCES (...)` : SQL clause to create a link between two tables.

Functions overview

```sql
CREATE OR REPLACE TABLE table_name (
      -- Create an auto generated unique value as primary key
      unique_key column_datatype AUTOINCREMENT PRIMARY KEY,
      other_business_key column_datatype,
      foreign_column column_datatype,
      other_foreign_column column_datatype,
      -- Adding relationship with other entities
      FOREIGN KEY(foreign_column) REFERENCES foreign_table(PK_from_foreign_table),
      FOREIGN KEY(other_foreign) REFERENCES foreign_table(PK_from_other_foreign)
);
```

Let's practice!

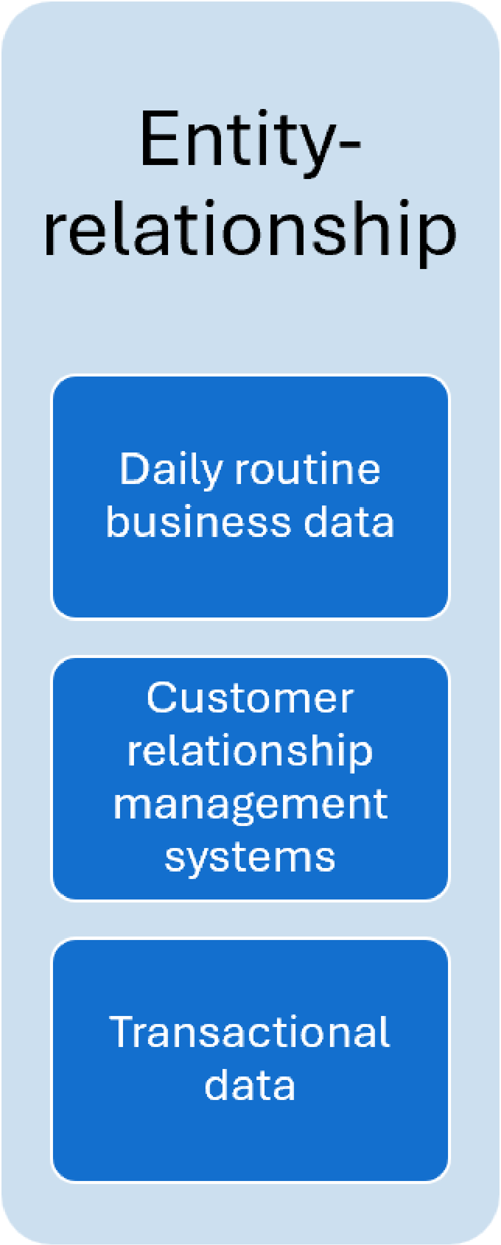
INTRODUCTION TO DATA MODELING IN SNOWFLAKE

Choosing the Right Approach

INTRODUCTION TO DATA MODELING IN SNOWFLAKE

Nuno Rocha
Director of Engineering

Use cases for each modeling technique



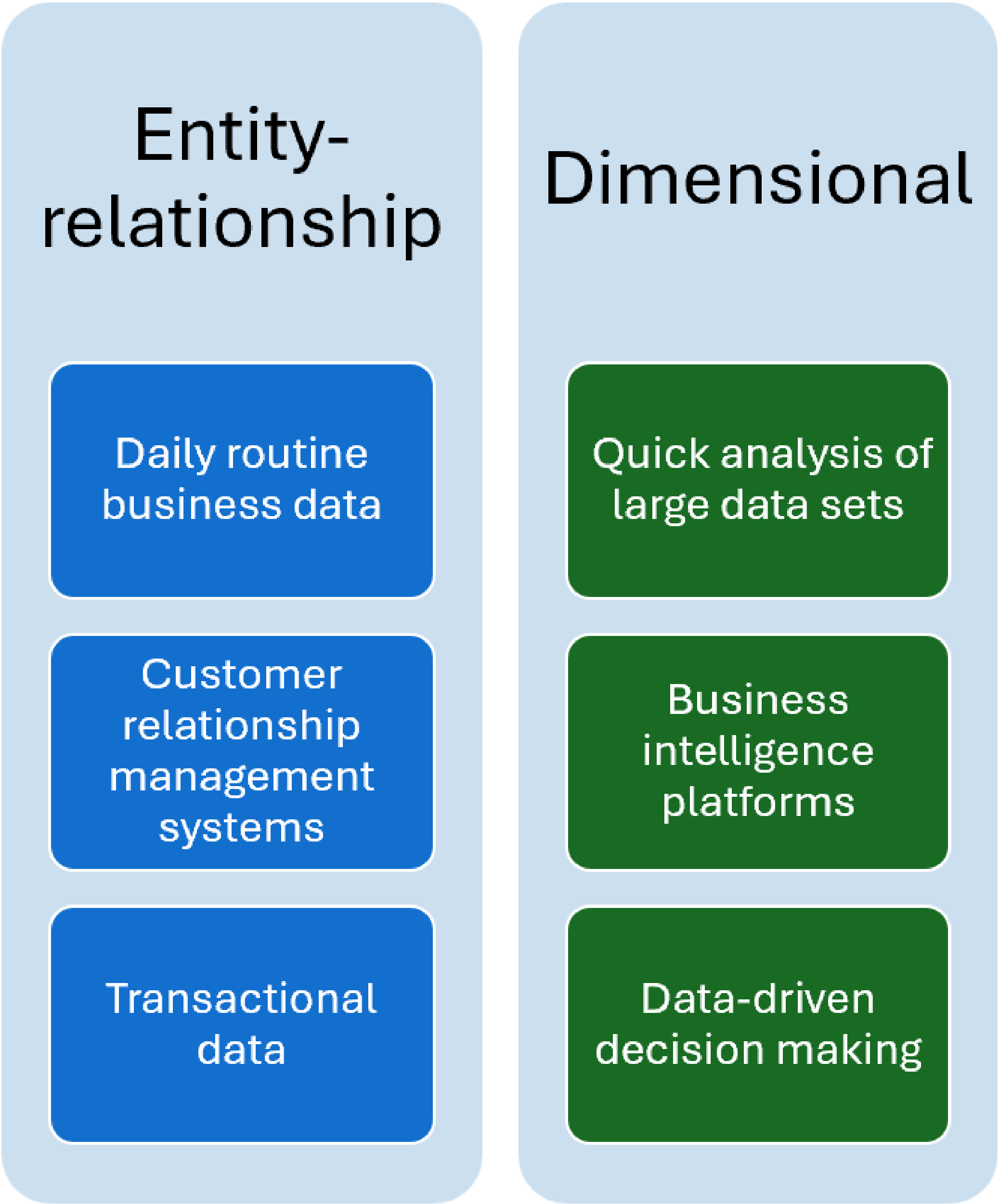
Entity-relationship

Daily routine business data

Customer relationship management systems

Transactional data

Use cases for each modeling technique (1)



Entity-relationship

- Daily routine business data
- Customer relationship management systems
- Transactional data

Dimensional

- Quick analysis of large data sets
- Business intelligence platforms
- Data-driven decision making

Use cases for each modeling technique (2)

Entity-relationship	Dimensional	Data vault
Daily routine business data	Quick analysis of large data sets	Long-term historical data tracking
Customer relationship management systems	Business intelligence platforms	Regulatory reporting or auditing
Transactional data	Data-driven decision making	Complete data history

Technical considerations

	+	−
Entity-Relationship model	Simple	Difficult when data grows enormously

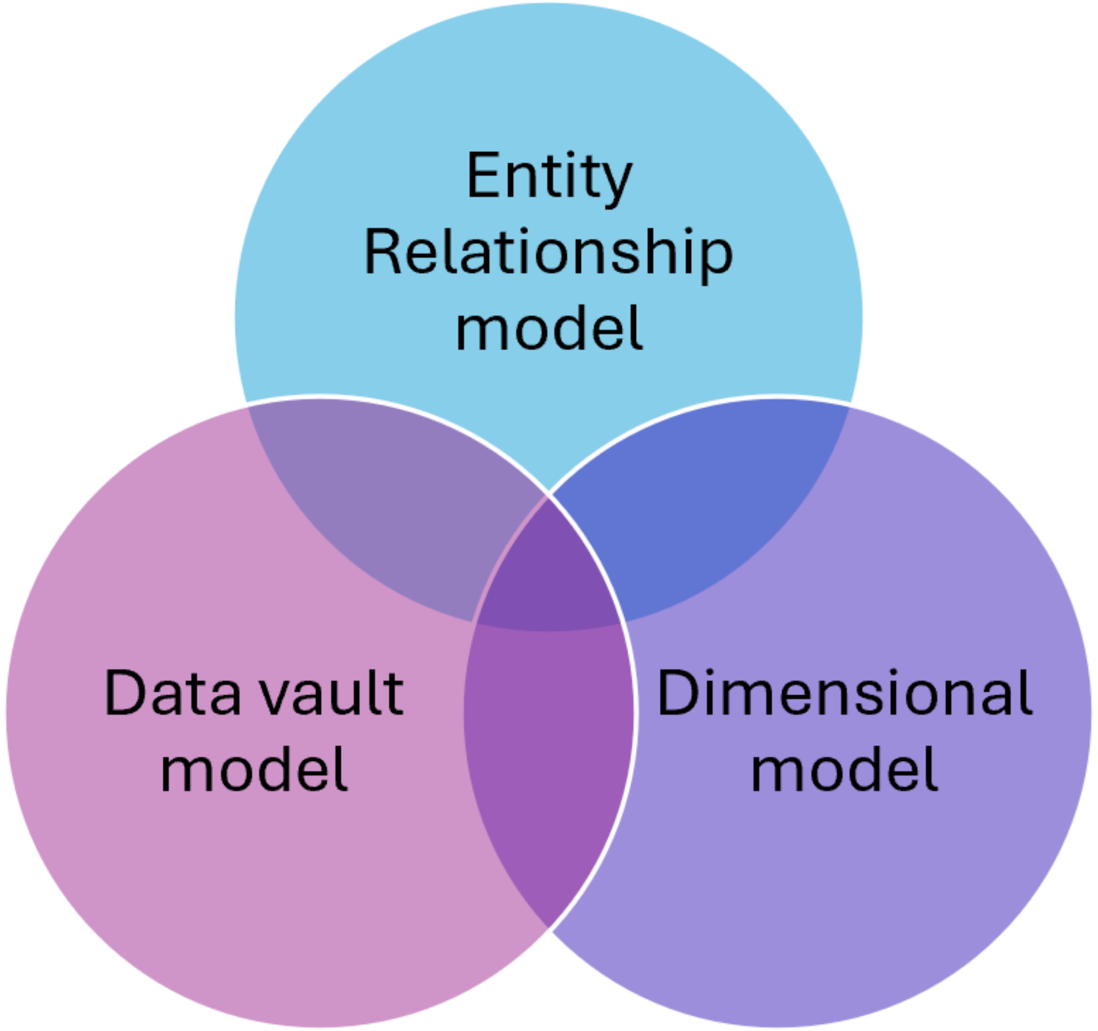
Technical considerations (1)

	+	−
Entity-Relationship model	Simple	Difficult when data grows enormously
Dimensional model	Fast query performance	Requires more maintenance

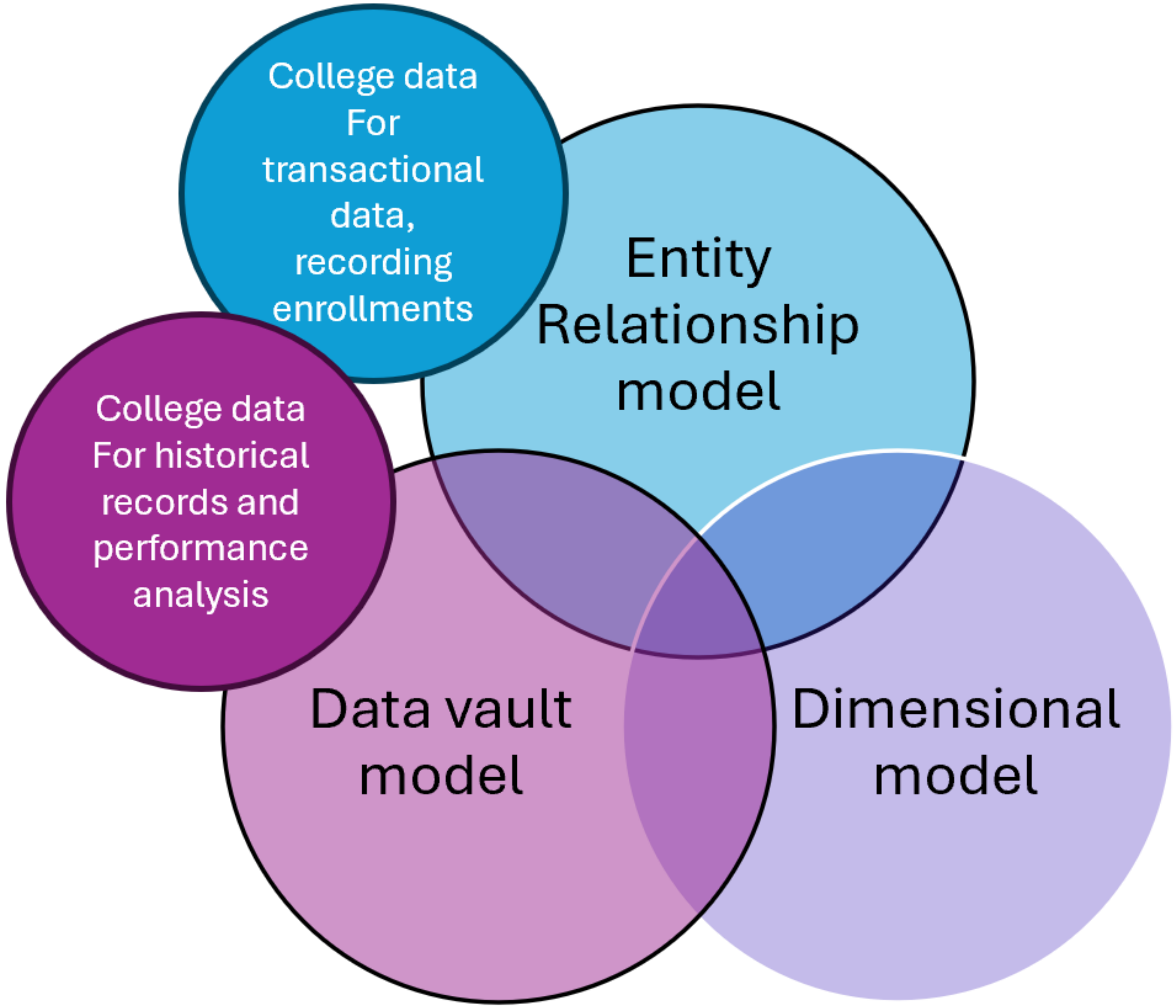
Technical considerations (2)

	+	−
Entity-Relationship model	Simple	Difficult when data grows enormously
Dimensional model	Fast query performance	Requires more maintenance
Data Vault model	Highly scalable	Require managing additional elements

Data models in action



Data models in action (1)


College data For transactional data, recording enrollments
College data For historical records and performance analysis
Entity Relationship model
Data vault model
Dimensional model

Retrieving data from the models

- Retrieve detailed data from students by referencing the hub entity and its satellite:

```
SELECT
    hs.student_key,
    ss.student_name
FROM hub_students AS hs
    JOIN sat_student AS ss ON hs.student_key = ss.student_key;
```

Retrieving data from the models (1)

- `LEFT JOIN ON` : SQL clause that combines all rows from the left entity with the matching rows from the right table, based `ON` a key

```sql
SELECT
    hs.student_key,
    ss.student_name
FROM hub_students AS hs
    JOIN sat_student AS ss ON hs.student_key = ss.student_key
    LEFT JOIN link_enrollment AS le ON hs.student_key = le.student_key
```

Retrieving data from the models (2)

- `COUNT` : SQL aggregate function that returns the number of items in a group.
- `GROUP BY` : SQL clause to aggregate data that have the same values.

```sql
SELECT
    hs.student_key,
    ss.student_name,
    COUNT(le.class_key) AS NumberOfEnrollments
FROM hub_students AS hs
    JOIN sat_student AS ss ON hs.student_key = ss.student_key
    LEFT JOIN link_enrollment AS le ON hs.student_key = le.student_key
GROUP BY hs.student_key,
    ss.student_name
```

Retrieving data from the models (3)

- `MAX` : SQL aggregate function that finds the highest value in a set of values for an attribute.

```
SELECT
    hs.student_key,
    ss.student_name,
    COUNT(le.class_key) AS NumberOfEnrollments
    MAX(sc.load_date) AS MostRecentEnrollmentDate
FROM hub_students hs
    JOIN sat_student ss ON hs.student_key = ss.student_key
    LEFT JOIN link_enrollment le ON hs.student_key = le.student_key
    LEFT JOIN sat_class sc ON le.class_key = sc.class_key
GROUP BY hs.student_key,
    ss.student_name;
```

Functions overview

- `SELECT FROM` : SQL command to fetch columns from an entity
- `JOIN ON` : SQL clause combining rows from entities based ON a related attribute
- `LEFT JOIN ON` : SQL clause that combines all rows from the left entity with the matching rows from the right table, based `ON` a key. If there's no match, the result will still show the left entity rows with empty values for the right attributes
- `COUNT` : SQL aggregate function that returns the number of items in a group
- `MAX` : SQL aggregate function that finds the highest value in a set of values for an attribute
- `GROUP BY` : SQL clause to aggregate data that have the same values

Functions overview

```sql
SELECT column_name,
    COUNT(another_column) AS alias_name,
    MAX(other_column) AS alias_name
FROM table_name table_alias
    -- Merge entities based on their keys
    JOIN other_table AS other_alias
    ON table_alias.FK = other_alias.PK
    LEFT JOIN another_table AS another_alias
    ON table_alias.FK = other_alias.PK
-- Aggregate data by specific columns
GROUP BY column_name;
```

Let's practice!

INTRODUCTION TO DATA MODELING IN SNOWFLAKE