2/17
ER = Entity Relationship
ERD includes: attribute type entity type relationship type

Table 6.4 Mapping an ER model to a relational model

ER model	Relational model
Entity type	Relation
Weak entity type	Foreign key
1:1 or 1:N relationship type	Foreign key
M:N relationship type	New relation with two foreign keys
n-ary relationship type	New relation with *n* foreign keys
Simple attribute type	Attribute type
Composite attribute type	Component attribute types
Multi-valued attribute type	Relation and foreign key
Key attribute type	Primary or alternative key

Design Principles

- A table models one Entity Type and an Entity Type is modeled by one table
- Each field in a table represents an attribute of an entity
- Each field in a table is assigned a strict primitive data type
- Each table has a Primary Key (PK) which is made up of one or more fields
- Each child table has a Foreign Key (FK) that points to its parent(s)

- Each *m:n* relationship is modeled with a junction table

Normalization

Normalization is the process of splitting relations into well structured relations that allow users to insert, delete, and update tuples without introducing database. Without normalization many problems can occur when trying to load an integrated conceptual model into the DBMS. These problems arise from relations that are generated directly from user views are called anomalies. There are three types of anomalies: update, deletion and insertion anomalies.

DATA ANOMALY

Insert Anomaly - is the inability to add data to the database due to absence of other data.

E.g. if you want to add a row in Current_Student and the values of(cno, cname, credits) are not exist in Classes, you are unable to insert such a row into Current_Student due to absence of (cno, cname, credits).

Delete Anomaly - is the unintended loss of data due to deletion of other data.

E.g. if you want to delete a class, then there will be loss of data in Current_Student since some students are taking this class.

Update Anomaly - is a data inconsistency that results from data redundancy and a partial update. It should be clear that redundancy of any kind can always lead to anomalies—because redundancy means, loosely, that some piece of information is represented twice, and so there's always the possibility that the two representations don't agree (i.e., if one is updated and the other isn't).

Common SQL Transforms

*These are from BQ's standard SQL and may slightly differ from MySQL/ Oracle/Postgres SQL syntax

- CREATE TABLE T2 AS SELECT ...
- SELECT a, b, c FROM T1 UNION ALL SELECT d, e, f FROM T2
- SELECT a, b, c FROM T1 UNION DISTINCT SELECT d, e, f FROM T2
- SELECT CAST('2020-02-17' AS DATE) AS new_date
- SELECT SAFE_CAST(xyz AS DATE) …

- SELECT CAST(SUBSTR('1000-00)', 0, 4) AS INT64) AS number

- SELECT ROW_NUMBER() OVER() AS row_num --Does not require the ORDER BY clause in OVER(). Returns the sequential row ordinal (1-based) of each row for each ordered partition. If the ORDER BY clause is unspecified then the result is non-deterministic.

- SELECT GENERATE_UUID() AS uuid --Returns a random universally unique identifier (UUID) as a STRING

```
+--------------------------------------+
| uuid                                 |
+--------------------------------------+
| 4192bff0-e1e0-43ce-a4db-912808c32493 |
+--------------------------------------+
```

Normal Forms

1NF: A database schema is in 1NF *iff* all attributes have scalar values.
2NF: 1NF + all non-key attributes must be *functionally determined* by the *entire* primary key.
3NF: 2NF + all non-key attributes must be *functionally determined* by *only* the primary key.

Jupiter Notebook code example:

https://github.com/cs327e-spring2020/snippets