- **Null Values**

  null signifies an unknown value or a value that does not exist. The predicate "is null" can be used to check for null values.

  e.g.) Find all loan number that appear in the loan relation with [not] null values for amount.

```
select loan_number
from loan
where amount is [not] null
```

  All aggregate operations except count(*) ignore tuples with null values on the aggregated attributes.

- **Nested Sub-queries**

  A sub-query that is nested within another query (a select-from-where expression)

- **Nested Sub-queries: Set Membership**

  Check if each data of an attribute belongs to a set

  e.g.) Find all customers who gave both an account and a loan at the bank.

```
(select customer_name from depositor)
intersect
(select customer_name from borrower)
```

```
select distinct customer_name
from borrower
where customer_name in (
     select customer_name from depositor)
```

  e.g.) Find all customers who gave a loan at the bank but do not have an account at the bank.

```
(select customer_name from borrower)
exept
(select customer_name from depositor)
```

```
select distinct customer_name
from borrower
where customer_name not in (
     select customer_name from depositor)
```

- **Nested Sub-queries: Set Comparison**

  Check if each data of an attribute is same as data in a set

  some checks if at least one value of the set satisfies the condition

  all checks if all values of the set satisfy the condition

  e.g.) Find all branches that have greater assets than some branch located in Brooklyn.

```
select distinct T.branch_name
from branch as T, branch as S
where T.assets > S.assets and (
     S.branch_city = 'Brooklyn')
```

```
select branch_name
from branch
where assets > some (
     select assets
     from branch
     where branch_city = 'Brooklyn')
```

  e.g.) Find the names of all branches that have grater assets than all branches located in Brooklyn.

```
select branch_name
from branch
where assets > all (
     select asstes
     from branch
     where branch_city = 'Brooklyn')
```

- **View Definition**

  Defining a view means to define a data set shown to user. A view is a kind of a virtual relation

```
create view <view name> as <query expr>
```

  e.g.) A view consisting of branches and their customers.

```
create view all_customer as (
     select branch_name, customer_name
     from depositor, account
     where depositor.account_number
            = account.account_number)
     union (
     select branch_name, customer_name
     from borrower, loan
     where borrower.loan_number
            = loan.loan_number)
```

- **Modification of the Database: Deletion**

```
delete from <rel name> where <predicate>
```

e.g.) Delete all tuples of accounts released at the Perryridge branch in account relation.

```
delete from account
where branch_name = 'Perryridge'
```

e.g.) Delete all accounts at every branch located in the city 'Brooklyn'.

```
delete from account
where branch_name in (
     select branch_name
     from branch
     where branch_city = 'Brooklyn')
```

- **Modification of the Database: Insertion**

```
insert into <rel name> values <value>
```

e.g.) Add a new tuple to account relation.

```
insert into account
values ('A-123', 'Perryridge', 1200)
```

```
insert into account
     (branch_name, balance, account_number)
values ('A-123', 'Perryridge', 1200)
```

- **Modification of the Database: Updates**

```
update <rel name> set <arithmetic expr>
where <predicate>
```

e.g.) Increase all account balance by 5%.

update account
set balance = balance * 1.05

e.g.) Increase all accounts with balances over $100 by 6%, all other accounts receive 5%.

```
update account
set balance = balance * 1.06
where balance > 100;
update account
set balance = balance * 1.05
where balance <= 100
```

- **Entity-Relationship Model**

A database can be modeled as a collection of entities and relationship among entities.

- **Entity** is an object that is distinguishable from other objects. e.g.) student, company, customer, etc.

Entities have their own properties, that is, attributes

- **Entity Set** is a set of entities that share the same properties. e.g.) set of all students, etc.

- **Relationship** is an association among several entities.

- **Relationship set** is a relation consisting of n entities

$$\{(e_1, e_2, \dots, e_n) | e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$$

e.g.) if depositor is a relationship set between customer and account,

$$\text{depositor} = \{(\text{Hayes, A-102}), (\text{Johnson, A-101}), \dots\}$$

$$(\text{Hayes, A-102}) \in \text{depositor}$$

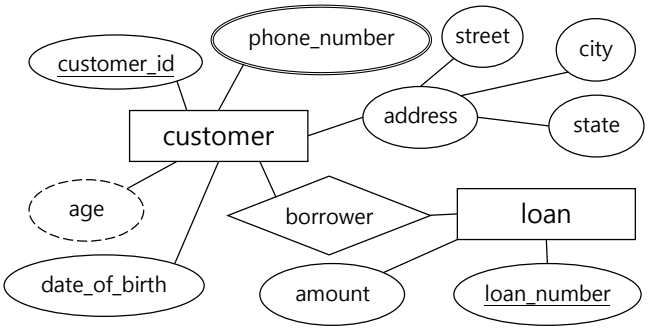
- **Degree of a Relationship Set**

The number of entity sets that participate in a relationship set. If relationship sets that involve two entity sets are the binary relationship set (or degree two).

- **Types of Attribute**

Simple-Composite / Single-Multi / Derived

- **E-R Diagrams**



Rectangles represent entity sets.
Diamonds represent relationship sets.
Ellipses represent attribute.
Double ellipses represent multivalued attribute.
Dashed ellipses denote derived attribute.
Underlined attribute indicates the primary key.

- **Cardinality Limits**

left	right	example
1	1	person : id card
0..1	1	driving license id : person
0..*	0..*	person : book
1..*	1	person : birth place

- **Good Decomposition** way can restore the original relation using natural join operation.

- **Anomalies** are inconvenient or error-prone situation arising when we process the tables.

Student Num	Course Num	Student Name	Addr	Course
S21	9201	Jones	Edinburgh	Accounts
S21	9267	Jones	Edinburgh	Accounts
S24	9267	Smith	Glasgow	Physics
S30	9201	Richards	Manchester	Computing
S30	9322	Richards	Manchester	Maths

- **Deletion Anomaly** exists when certain attributes are lost because of the deletion of other attribute.

  e.g.) If student S30 is the last student to leave the course, all information about the course is lost.

- **Insertion Anomaly** occurs when certain attributes cannot be inserted into the database without the presence of other attribute.

  e.g.) Can't add a new course unless there have at least one student enrolled on the course.

- **Update Anomaly** exists when one or more instances of duplicated data is updated, but not all.

  e.g.) Consider Jones moving address. It needs to update all instances of Jones's address.

- **Functional Dependency (FD)**

  A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y

  $$X \rightarrow Y$$

  X is determinant, Y is dependent on X.

  When satisfying $X \rightarrow Y$, if two tuples have the same value for X, they must have the same value for Y.

  If X is a key of a relation, then X functionally determines all attributes Y in the relation.

- **Full FD**

  A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y.

- **Partial FD**

  E.g. if $\{X, Y\} \rightarrow \{C\}$ but also $\{X\} \rightarrow \{C\}$ then $\{C\}$ is partially functionally dependent of $\{X, Y\}$

- **Transitive FD**

  $\{A\} \rightarrow \{B\}$, $\{B\} \rightarrow \{C\}$, $\{A\} \rightarrow \{C\}$ but $\{B\} \nrightarrow \{A\}$

  If Y is a candidate key, there is no anomaly occurred by the Transitive FD.

- **Multi-Valued Dependency (MVD)**

  $$X \rightarrow\rightarrow Y$$

  A set of different values for Y on each X value.

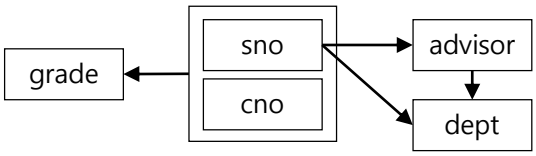
-

- **First Normal Form (1NF)**

The purpose of 1NNF is to reduce non-atomic values.

All domains of a relation are atomic value.

Does not allow: composite attributes / multi-valued attributes



➢ FD

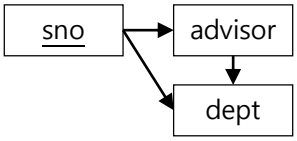
$$\{\text{sno},\ \text{cno}\} \rightarrow \text{grade}$$
$$\text{sno} \rightarrow \text{advisor}$$
$$\text{sno} \rightarrow \text{dept}$$
$$\text{advisor} \rightarrow \text{dept}$$

- **Second Normal Form (2NF)**

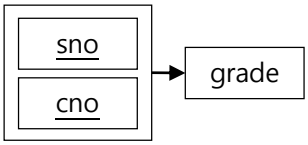
The purpose of 2NF is to remove partial FD.

$$1\text{NF} \overset{\text{Projection}}{\underset{\text{Join}}{\longleftrightarrow}} 2\text{NF}$$

A relation is in 2NF if the relation is in 1NF and every non-prime attribute A in the relation is fully dependent on the primary key.



[advise]



[course] / {sno} ref. advise

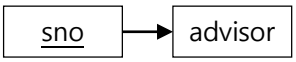
Foreign key makes relationship among relations. A primary key in a referenced relation can be a foreign key.

- **Third Normal Form (3NF)**

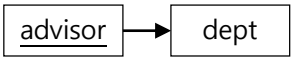
The purpose of 3NF is to reduce transitive FD.

$$2\text{NF} \overset{\text{Projection}}{\underset{\text{Join}}{\longleftrightarrow}} 3\text{NF}$$

A relation is 3NF if the relation is in 2NF and no non-prime attribute A in the relation is transitively dependent on the primary key.



[studentadvise] / {advisor} ref. advisordept

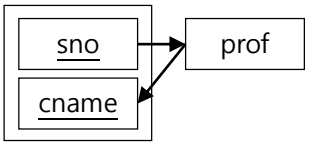


[advisordept]

- **Boyce/Codd Normal Form (BCNF)**

The purpose of BCNF is to make all determinants to be candidate keys.

A relation schema R is in BCNF if all X satisfying $X \rightarrow A$ in R is a super key (or candidate key) of R



[takecourse]

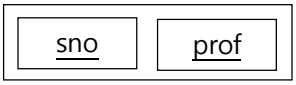
➢ Candidate key

$$\{\text{sno},\ \text{cname}\}$$
$$\{\text{sno},\ \text{prof}\}$$

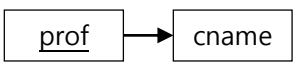
➢ FD

$$\{\text{sno},\ \text{cname}\} \rightarrow \text{prof}$$
$$\text{prof} \rightarrow \text{cname}$$

above BCNF is below.



[enrollprof] / {prof} ref. cprof



[cprof]

- **Forth Normal Form (4NF)**

  The purpose of 4NF is to remove Multi-Valued Dependency (MVD).

- **Fifth Normal Form (5NF)**

  Good decomposition way can restore the original relation using natural join operation.

- **Optimization Considers**

  Reduce the number of disk access / Reduce the size of intermediate result / Reduce the response time.

$$\mathrm{R}\bowtie\left(\sigma_{S_c}(S)\right)=\sigma_{S_c}(R\bowtie S)$$

  The former is more efficient.

- **Transaction Example**

```
1. read(A)
2. A := A - 50
3. write(A)
4. read(B)
5. B := B + 50
6. write(B)
```

- **Transaction: Atomicity**

  If the transaction fails after step 3 and before step 6, money will be "lost" leading to an inconsistent database state.

  The system should ensure that a partially executed transaction should correctly update the database or does not update.

- **Transaction: Consistency**

  The sum of A and B should be unchanged by the execution of the transaction.

  Execution of a transaction preserves the consistency of the database.

- **Transaction: Isolation**

  If between steps 3 and 6, another transaction accesses the partially updated database, the result of the other may be incorrect.

```
3. write(A)
   read(A), read(B), print(A+B)
4. read(B)
```

  Isolation means each of multiple transactions should finish individually with correct results.

- **Transaction: Durability**

  Once the user has been notified that the transaction has completed (i.e., the transfer of the $50 was successfully done), the result must persist, even if there are software or hardware failures after the transaction.

  Assuming that hard disc rarely loses the data in comparison with memory, recording the result in the disc before completing the transaction can be a solution.

- **Transaction State: Active**

  The initial state and executing state. The transaction stays in this state while it is executing.

- **Transaction State: Partially Committed**

  Just after the final operation of the transaction was executed.

- **Transaction State: Failed**

  When normal execution cannot proceed before committing the transaction.

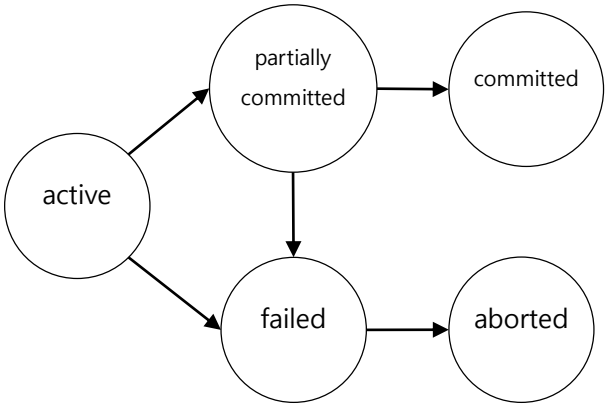
- **Transaction State: Aborted**

  When the transaction was rolled back prior to the initial state due to transaction fail. Two options after the transaction was aborted: restart/kill the transaction.

- **Transaction State: Committed**

  After successfully completing the transaction.

- **Transaction State Diagram**



- **Domain constraints**

  The most elementary type of integrity constraint. Test values inserted in the database.

- **Referential Integrity**

$$\Pi_{\alpha}(r_2) \subseteq \Pi_{\mathrm{K1}}(r1)$$

  α is a foreign key referencing K_1 in relation r_1.

  domain of α is subset of domain of K_1.

  Ensure that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.

- **Cascading Actions in SQL**

```
create table account(
  balanch-name     char(15),
  account-number   char(10) not null,
  balance          integer,
  primary key(account-number),
  foreign key(branch-name)
    references branch
  on delete cascade
  on update cascade
)
```

-