INTRODUCTION TO DATABASES

OVERVIEW

purpose – to limit data redundancy

data anomalies – insert
– update
– delete

DESIGN LIFECYCLE

Requirements
↓
Conceptual
↓
logical
↓
Physical

CONCEPTUAL MODEL

Entity Relationship (ER) Model used for Conceptual modelling

Independent of db type

Components –

1. Entity – object / table / entity set
2. Attribute – characteristic of entity
3. Relationship – associatn b/w entities

TYPES OF ATTRIBUTES

Simple - can't be subdivided — gender

Composite - can be subdivided — address

Single Valued - One value ONLY — gender

Multi Valued - can have 1+ values — degree

Derived - computed from other att. — age

TYPES OF RELATIONSHIPS

Identifying - PK of an entity is inherited from related entity

→ Weak entities

Non Identifying - Independent PK

→ Strong entities

RELATIONSHIP PARTICIPATION

Connectivity — 1 to 1, 1 to M etc.

Cardinality — min and max occurrences of an entity with related

Degree - No. of entities associated in a rel^{n} :

unary (recursive)

binary

tertiary

+ more (v. rare)

ASSOCIATIVE ENTITY

- aka composite / bridge entity
- used to represent M:N rel^n^ship
- STRONG rel^n^
- converts M:N rel^n^ into 2 (1:M)

RELATIONAL MODEL

Concept of tables (rows x columns)

Rel^n^ — Head + Body
rel^n^ schema — rel^n^ instance

RELATION — Table — rel^n^ heading (column headings) fixed set of attr.
— rel^n^ body (set of rows)
↓
tuples
set of values $v_1, v_2 \dots v_n$

rel^n^ cardinality = no. of tuples
rel^n^ degree = no. of values (in tuple)

tuples are unordered & unique. values in a tuple are atomic → single valued

rel^n^ ≠ table → ordered rows + cols

Functional dependence —

a set of attr. A determines an attr. B iff for each A value there is exactly one value of B in the rel^n^

$A \longrightarrow B$

A determines B OR B depends on A

RELATIONAL MODEL KEYS

Superkey : can uniquely identify a key
many — one attr. / set of attr

$t_1(\text{superkey}) \neq t_2(\text{superkey})$

Candidate Key : minimal superkey
potentially many

Primary Key : chosen candidate
ONE

a PK should be unique
[non intelligent
no change over time
→ eg- name
single attr. } preferable
numeric
security compliant
eg- SSN shouldn't be the PK.

Foreign Key — PK of a rel^n placed in another
↓
or even in the same rel^n

PK and FK create relationships (logical connections) b/w tables in RDBS.

DATA INTEGRITY

Entity Integrity — PK is ① unique and

(2) not null

NULL ≠ zero

N/A unknown undefined doesn't exist

Referential Integrity – FK should match PK or be left null

↓

of the related rel^n

Column/Data Integrity – same data type and range for values in a column.

RELATIONAL ALGEBRA

manipulating table contents using rel^n operators

SELECT (σ) – selecting rows
PROJECT (π) – column selectn } unary

JOIN ($\bowtie$) – combine data from 2 or more tables

– types.

only those rows that match

1. theta $\theta \in \{<, \leq, =, >, \geq\}$
2. equi =
3. natural (advanced equi)
4. outer (left & right)

Natural Join

– multiply tables

delete' rows where PK don't match
delete repeated column

$$T_1 \bowtie T_2 = \pi_{a_1, a_2, a_3}\left(\sigma_{T1.PK = T2.PK}(T_1 \times T_2)\right)$$

NORMALISATION

process that assigns attributes to entities to reduce data redundancy and anomalies

based on PK/FK and functional dependencies

Denormalisatⁿ : Reverse, to increase Speed

prime attributes — part of the key (or whole)
key

non-prime attr. — not part of the key
non-key

prop.(s) a relⁿ should satisfy in ERM:

- entity integrity
- referential integrity
- no M:M relⁿships
- each cell contains atomic values

FUNCTIONAL DEPENDENCY

$$A \longrightarrow B$$

determinant dependent

value of A determines value of B.

Total dependency — A determines B and B determines A

for a relatⁿ $(A,B) \longrightarrow (C,D)$:

1. full dependency —

$(A,B) \longrightarrow C$

$(A,B) \longrightarrow (C,D)$

2. partial dependency –

$A \longrightarrow C$

↓

part of PK.

3. transitive dependency

$(A,B) \longrightarrow C$

$C \longrightarrow D$ } $A \longrightarrow D$

↓

functional dependency in non key attribute

NORMALISED FORMS:

1. Unnormalised form (UNF)
   (i) single-named rep^{n}
   ↓
   can't be plural name
   (ii) identify repeating groups
2. First Normal form. (1NF)
   (i) PK identified
   (ii) no repeating groups
3. Second Normal form (2NF)
   (i) 1 NF

(ii) if composite PK:

no partial dependencies

4. Third Normal form (3NF)

(i) 2 NF

(iii) No transitive dependencies

5. Attribute Synthesis

(i) eliminate redundant rel^{n}(s)
→ NO duplicat^{n}.

Steps — formulate UNF — no PK

UNF to INF (remove repeating groups)

INF to 2NF (remove partial dep)

2NF to 3NF (remove trans. dep.)

Attribute Synthesis

LOGICAL MODELLING

TERMINOLOGY :

Conceptual	logical	Physical
Entity	Rel^{n}	Table
Attr.	Attr.	Column
Instance	Tuple	Row
Identifier	PK	P.K
Relat^{n}ship	—	—
—	FK	FK

TRANSFORMING ER DIAGRAMS TO RELN(S)

- key to PK
- rel"ships to PK/FK pairs
- map :
  1. strong } entities
  2. weak }
  3. binary rel"ships
  4. associative entities
  5. unary } rel"ships
  6. tertiary }

Surrogate Keys - ONLY in logical model

(refer slides)

DDL

Data Definition Language (DDL)

- creating db structure
  - CREATE
  - ALTER
  - DROP
  } table

No commit or rollback (auto COMMIT)

Data Manipulation Language (DML)

- populating / depopulating db
  - ↓ INSERT / UPDATE
  - ↓ DELETE
  - SELECT ↓ retrieving data

Data Control Language

- set permission on objects (GRANT)

Data Types

Text

CHAR (size) 'apple' = 'apple---'
VARCHAR2 (size) 'apple' != 'apple--'

Number

NUMBER (precision, scale)

Date / Time

DATE — upto seconds
TIMESTAMP — [upto fract^n of seconds
includes timezone

Constraints –

if not possible to add FK, first CREATE table then ALTER table

types — Table and Column constraint

ex –

```
CREATE TABLE tablename (

    col1 NUMBER(6) NOT NULL,
    col2 VARCHAR(2),                    } col const.
    col3 DATE NOT NULL
    CONSTRAINT const_name
        PRIMARY KEY (col1)              } table const.
    CONSTRAINT const_name
        FOREIGN KEY (col1)
        REFERENCES tablename (col1)

);
```

to change table structure : add/ remove cols + const.

```
ALTER TABLE tablename
```

```
ADD (
  CONSTRAINT const_name
  FOREIGN KEY (col2)
  REFERENCES tablename (col2)
  ON DELETE CASCADE
);
```

REFERENTIAL INTEGRITY

PARENT
PK p_id

CHILD
FK p_id

actions to ensure referential integrity :

1. RESTRICT — No action

   Can't delete p_id in parent unless there's no FK in child corresponding to p_id in Parent

2. CASCADE — Delete in all

   if p_id deleted in parent, then rows containing p_id as FK in child are automatically deleted.

3. NULLIFY — set Null

   if p_id deleted in parent, then cells in child containing p_id as FK are set to NULL

referential integrity constraints are decided in the design phase as per the use case and client req.

↗ populating the database

INSERT INTO tablename (col1, col2)

VALUES (value1, value2)

COMMIT; ⟶ to make changes to db permanent

CREATE SEQUENCE seq_name
INCREMENT BY 1;

↳ auto increment of numeric PK

INSERT INTO tablename (col1, col2)
VALUES (seq_name.nextval, val2)

↓

initiating a seq OR increment a seq

INSERT INTO tablename2 (col1, col2)
VALUES (seq_name.currval, val2)

↓

getting the current value

↳ dump the table

DROP TABLE tablename

Drop vs Delete ?

delete data (rows) in the table
OR
drop entire table

SQL - I

SELECT col1, col2, col3
FROM tablename
WHERE condition

Search predicates :

1. comparison

=, !=, <>, <=, <, >=, >
col1 > 5000

2. range

col1 BETWEEN 1000 AND 3000

→ inclusive

3. set membership

IN

col1 IN ('value1', 'value2')

4. pattern match

LIKE (_ or %)

WHERE col1 LIKE 'M%'

WHERE col2 LIKE 'FM913_'

5. NULL

WHERE col1 IS NULL

Rows to be retrieve — TRUE
other FALSE & UNKNOWN
→ NULL

Combining predicates

1. AND (2)
2. OR (3)
3. NOT (1)

} instead use ( )

Arithmetic Operations

SELECT col1, col2/10 FROM tablename

NVL function

to replace null with a value

```
SELECT    col1,
    NVL   (col2, '0'),
    NVL   (col3, 'NA')
FROM tablename;
```

Renaming Column

```
SELECT col1, col2 AS new_col2
FROM tablename;
```

Sorting query results

```
SELECT col1, col2
FROM tablename
ORDER BY col2 DESC
```

↓ by default ASC.

also – "NULLS FIRST" and "NULLS LAST"

Removing Duplicates

```
SELECT DISTINCT col1
FROM tablename
WHERE condition
ORDER BY col1
```

SQL JOINS

Types –

1. ON

```
FROM table1 JOIN table2
ON table1.attr = table2.attr
```

2. USING

```
FROM table1 JOIN table2
USING (attr)
```

3. NATURAL JOIN

```
FROM table1
NATURAL JOIN table2
```

→ can only use join in FROM

Joining multiple tables –

```
SELECT    t1.col1 , t1.col2, t2.col1
FROM ((table1 t1 JOIN table2 t2
        ON t1.col1 = t2.col2)
      JOIN table3 t3
        ON t1.col1 = t3.col1))
ORDER BY  t1.col1, t2.col1
```

Using Date :

1. to_char – for output

```
SELECT to_char (sysdate,'dd-Mon-yyyy')
FROM dual
```

2. to_data – for inserting (or comparing)

```
INSERT INTO tablename
VALUES(to_date ('01-Mar-1997','dd-Mon-yyyy'))
```

TRANSACTION MANAGEMENT

UPDATE

– change the value of existing data

```
UPDATE tablename
SET    col1 = value1,
       col2 = value2
WHERE  col3 = value3
```

DELETE

– remove data from db

```
DELETE   FROM tablename
WHERE    col1 = value1 ;
```

TRANSACTION

an action that reads / writes data to database — either SELECT
UPDATE
INSERT
or combination these.

transaction properties

1. ATOMICITY

   Trans^n must be entirely completed ABORTED otherwise

2. CONSISTENCY

   must take db from one consistent state to another.

3. ISOLATION

   no interference among concurrent transactions

4. DURABILITY

   once trans^n is completed changes are made permanent (to db)
   NO undo

CONCURRENCY

co-ordinating simultaneous execut^n of tran^n(s) in a multi-user db.

prob(s) — lost updates

uncommitted data
inconsistent retrieval

Concurrency Management

— Locking mechanism :

to overcome prob(s) caused by interleaved transactn(s)

trans — gain locks prior to opern
release on compn
→ controlled by Lock Manager.

Lock — A) Binary — locked
Unlocked

too restrictive !!

B) Shared + Exclusive

↓ common lock
read access

↓ ONLY if no other locks.
write access

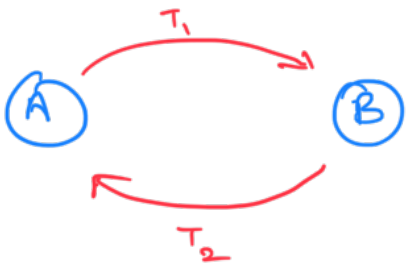
LOCK GRANULARITY

refers to size of units locked:

1. db level — table
2. table level — row
3. page level — disk page
4. record level — diff. rows
5. attr. level — same row, diff column

DEADLOCK

T_1 has lock on A and req.s lock on B
T_2 has lock on B and req.s lock on A



Prevention –
acquire all locks and release when done
↳ if can not, try later.

Detection & Recovery
Lock manager monitors wait-for graph & forces 1 trans. to abort
↓
victim sel^n

victim sel^n algo – avoid trans. that has been running for long and has performed many updates

instead choose that haven't made any changes OR involved in more than one deadlock.

Db restart vs recovery.
↓ ↓
soft crash hard crash

TRANSACTION LOG

for each SQL statement:
↳ trans. comp.

1. record beginning of trans.

2. opⁿ type
3. table affected
4. before and after value
5. prev. & next trans.
6. commit

Checkpointing – every 15 min. or 20 transⁿ

- new trans – temp. halted
- current tr. – suspended
- completed tr. – changes made permanent

→ exeⁿ resumed

policies – write through
immediately updated
right before commit pt.
redo + undo list
deferred write
db updated only after commit pt.
→ no rollback req.d

refer slides for backup & recovery → general info.

- bup – on site + off site
- recovery – from bup (redo)

SQL II

Aggregate functions – Count, Max, Min, Sum, Avg

Clause { SELECT
FROM
WHERE
GROUP BY → }

GROUP BY
HAVING → predicate / search cond^{n}
ORDER BY ;

→ SQL statement

group by — for agg. funct^{n} (optional)

having — to put cond^{n}(s) on group by.

having vs where ?

→ where applied to ALL rows but having on groups defined by group by.

attributes used in SELECT, HAVING and ORDER BY must be included in GROUP BY

SUBQUERIES

Query within a query.

```
SELECT  *
FROM    tablename
WHERE   col1 > ( SELECT  avg (col1)
                 FROM    tablname );
```

↳ subquery

types of subqueries —

1. single value
2. multiple rows (one column)
3. multiple rows, multiple cols.

Comparison operators for subquery —

1. =, <, > single value

2. IN equality
   ALL, ANY in equality

```
SELECT *
FROM tablename
WHERE col1 > [ANY] ( SELECT avg (col1)
                     FROM tablename
                     GROUP BY col2)
ORDER BY col1, col2, col3 ;
```

Subquery can be NESTED
CORELATED
(refer wk 10 slides) INLINE (derived table)

SQL III

CASE

used in select, to evaluate an attr and output a value based on eval^{n}.

```
SELECT
    col1,
    case col2
        when 'val1' then 'value 1'
        when 'val2' then 'value 2'
    end as column 2
    case col3
        when val1 < 2 then 'value 1'
        when val2 > 2 then 'value 2'
        else 'value 3'
    end as column 3
FROM tablename
```

VIEWS

virtual table derived from 1 or more tables.

```
CREATE VIEW view_name AS
  SELECT col1, max(col2)
  FROM tablename
  GROUP BY col1
```

OUTER JOIN

1. Full Outer Join
   info on both table (incomplete)

```
SELECT * FROM
table1 t1 full outer join
table2 t2 on t1.attr = t2.attr;
```

2. Left Outer Join
   all rows on left table and only matching rows from right.

```
SELECT * FROM
table1 t1 left outer join
table2 t2 on t1.attr = t2.attr;
```

3. Right Outer Join
   all rows on right table an only matching rows from left.

```
SELECT * FROM
table1 t1 right outer join
table2 t2 on t1.attr = t2.attr
```

SET OPERATORS

1. union all (including duplicates)
2. union
3. intersect
4. minus

Mongo DB

```
db.tablename.insertOne({})
db.tablename.insertMany([{},{}])
```

```
db.tablename.find({}).pretty()
db.tablename.updateOne({})
db.tablename.deleteOne({})
db.tablename.deleteMany({})
```