

INTRODUCTION TO DATABASES

WEEK 1 — OVERVIEW

purpose — to limit data redundancy

data anomalies —
— insert
— update
— delete

DESIGN LIFECYCLE

Requirements



Conceptual

↓
logical

↓
Physical

WEEK 2 — 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. Relationships — associate 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 attr. 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 reln:

unary (recursive)

binary

tertiary

+ more (rare)

ASSOCIATIVE ENTITY

- aka composite / bridge entity
- used to represent M:N relationship
- STRONG relⁿ
- Converts M:N relⁿ into 2 (1:M)

WEEK 3 - RELATIONAL MODEL

Concept of tables (rows x columns)

relⁿ — Head + Body
relⁿ relⁿ
schema instance

RELATION — Table — relⁿ heading
(column heading)
fixed set of attr.)

— relⁿ body
(set of rows)



tuples

set of values $v_1, v_2 \dots v_n$

relⁿ cardinality = no. of tuples

relⁿ degree = no. of values (in tuple)

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

relⁿ \neq table \rightarrow 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 relⁿ

$A \rightarrow B$

A determines B OR B depends on A

RELATIONAL MODEL KEYS

Superkey : can uniquely identify a key
many one attr. or set of attrs

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

Candidate Key : minimal superkey
potentially many

Primary Key : chosen candidate
ONE

a PK should be unique

- [non intelligent
- [no change over time
 - ex - name
- [single attr. } preferable
- [numeric }
- [security compliant
- [ex - SSN shouldn't be the PR.

Foreign Key - PK of a relⁿ placed

↓
or even in
the same relⁿ

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 \neq zero

N/A unknown undefined doesn't exist

Referential Integrity - FK should match
PK or be left
null
of the related relⁿ

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

RELATIONAL ALGEBRA

manipulating table contents using relⁿ operators

SELECT (σ) - selecting rows
PROJECT (π) - column selection

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

- types:

only those rows that match

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

Natural Join

- multiply tables

delete rows where PK don't match
delete repeated column

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

WEEK 4 - NORMALISATION

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

based on PK/FK and functional dependencies

Denormalisation : 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 relationships
- each cell contains atomic values

FUNCTIONAL DEPENDENCY

A → B
determinant dependent

value of A determines value of B.

Total dependency - A determines B and
B determines A

for a relation (A,B) → (C,D) :

1. full dependency -

$(A, B) \rightarrow C$

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

2. partial dependency -

$A \rightarrow C$

↓
part of PK.

3. transitive dependency

$(A, B) \rightarrow C$ }
 $C \rightarrow D$ $A \rightarrow D$

↓
functional dependency in non key
attribute

NORMALISED FORMS :

1. Unnormalised form (UNF)

(i) single - named rep"

↓
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 relⁿ(s)
→ no duplicatⁿ.

Steps - formulate UNF - no PK

UNF to 1NF (remove
repeating groups)

1NF to 2NF (remove
partial dep.)

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

Attribute Synthesis

WEEK 5 - LOGICAL MODELLING

TERMINOLOGY :

Conceptual	Logical	Physical
Entity	Rel ⁿ	Table
Atr.	Atr.	Column
Instance	Tuple	Row
Identifier	PK	P.K
Relationship	-	-
-	FR	FR

TRANSFORMING ER DIAGRAMS TO RELN(S)

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

Surrogate Keys - ONLY in logical model

(refer slides)

WEEK 6 - 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

DELETE

SELECT

receiving 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
TIMESTAMP

upto seconds
upto fractⁿ 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),
col3 DATE NOT NULL
CONSTRAINT const-name
PRIMARY KEY (col1)
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
FR p-id

actions to ensure referential integrity :

1. RESTRICT — No action

can't delete p-id in parent unless there's no FR 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.curval, val2)

↓
getting the current value

→ dump the table

DROP TABLE tablename

Drop vs Delete ?

delete data (rows) in the table
OR

drop entire table

WEEK 7 - SQL - I

SELECT col1, col2, col3
FROM tablename
WHERE condition

Search predicates :

1. comparison

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

2. range
col1 BETWEEN 1000 AND 8000
inclusive

3. set membership

IN
col1 IN ('value1', 'value2')

4. pattern match

LIKE (- or %)
WHERE col1 LIKE 'm %'
WHERE col2 LIKE 'fin%is_'

5. NULL
WHERE col1 IS NULL

Rows to be retrieve — TRUE
other FALSE & UNKNOWN

NULL

Combining predicates

1. AND (2)
2. OR (3) } instead use ()
3. NOT (1)

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 Duplicate

```
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_date — for inserting
(or comparing)

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

WEEK 8 — 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

Transⁿ 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 transⁿ is completed changes are made permanent (to db)
No undo

CONCURRENCY

co-ordinating simultaneous execⁿ of transⁿ(s) in a multi-use db.

prob(s) — lost updates

uncommitted data
inconsistent retrieval

Concurrency Management

- Locking mechanism:

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

trans - gain locks prior to exec
release on commit
→ controlled by Lock Manager.

Lock - A) Binary [locked Unlocked]
 too restrictive !!

B) Shared + Exclusive



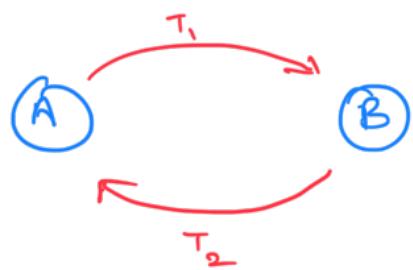
LOCK GRANULARITY

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

T₁ has lock on A and req.s lock on B
T₂ 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 selⁿ

victim selⁿ 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. op^m type() T
3. table(s) 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

→ execⁿ resumed

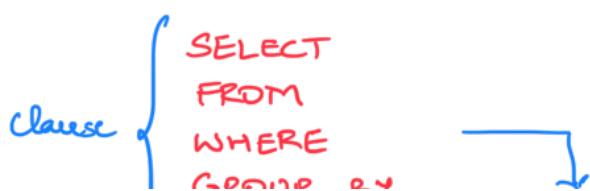
policies - write through
immediately updated
right before commit pt.
redo + undo list
deferred write
db updated only after
commit pt.
→ no rollback req'd

refer slides for backup & recovery
→ general info.

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

WEEK 9 — SQL II

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



HAVING → predicate/search cond
ORDER BY ;

→ SQL statement

group by — for agg. funct (optional)

having — to put cond(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 tablename);
```

↳ 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
(refer wk 10 slides) CORRELATED
INLINE (derived table)

WEEK 10 - SQL III

CASE

used in select, to evaluate an attr and output a value based on evalⁿ.

```
SELECT
    col1,
    case col2
        when 'val1' then 'value1'
        when 'val2' then 'value2'
    end as column2
    case col3
        when val1 < 2 then 'value1'
        when val2 > 2 then 'value2'
        else 'value3'
    end as column3
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 and 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({3}).pretty()
db.tablename . updateOne({3})
db.tablename . deleteOne({3})
db.tablename . deleteMany({3})